Specifica Tecnica

SnakeByte (Gruppo 1):
Valeria Baleanu, Leonardo Pellizzon, Filippo Venzo, Giuseppe De Fina,
Francesco Pasqual, Christian Libralato, Luca Granziero
(2109911, 2111006, 2113705, 2113187, 2103119, 2101047, 2075512)

Informazioni documento			
Versione	**Data**	**Stato**	**Destinatari**
1.0.0	05/04/2026	Approvato	**Interni:** SnakeByte **Esterni:** prof. Vardanega Tullio, prof. Cardin Riccardo, Vimar

Contatti: snakebyteteam@gmail.com

Registro delle modifiche					
Versione	**Data**	**Autore**	**Verificatore**	**Approvatore**	**Descrizione**
1.0.0	05/04/2026	-	-	C. Libralato	Approvazione
0.3.0	04/04/2026	V. Baleanu	G. De Fina	-	Aggiunta moduli backend
0.2.0	03/04/2026	C. Libralato	G. De Fina	-	Aggiunta moduli frontend
0.1.0	24/03/2026	L. Pellizon	C. Libralato	-	Prima Stesura

Indice

Elenco delle figure

1 Introduzione

1.1 Contenuto del documento

La specifica tecnica è un documento che descrive in dettaglio l'architettura, le tecnologie e i pattern utilizzati per lo sviluppo del prodotto. Questo documento è destinato a fornire una guida chiara per quella che è la struttura e le componenti di esso.
Deve essere visto come un riferimento per la progettazione e la codifica del prodotto, in modo da garantire coerenza con il *Proof of Concept*.

1.1.1 Struttura del documento

Il documento è suddiviso in 4 macro-sezioni:

- **Introduzione**: espone il contenuto e lo scopo del documento, oltre che i Riferimenti Normativi e Informativi;
- **Tecnologie**: descrive le tecnologie utilizzate per la codifica del prodotto, con una breve descrizione di ognuna;
- ***Pattern***: descrive i *pattern* utilizzati per la progettazione del prodotto, con una breve descrizione di ognuno;
- **Architettura logica**: descrive l'architettura logica del prodotto;
- **Architettura di *deployment***: descrive in che modo vengono distribuiti i componenti del prodotto;
- **Dettagli progettuali**: descrivono le micro-architetture dei componenti del prodotto, con diagrammi e descrizioni dettagliate.

1.2 Glossario

All'interno del documento possono essere presenti termini il cui significato può risultare ambiguo o sconosciuto, essi sono riportati in italico e alla loro prima occorrenza è associata una G a pedice ad indicare la presenza del suddetto termine all'interno del $glossario_G$ che è disponibile a seguente link: Glossario.

1.2.1 Riferimenti Normativi

- **Norme di Progetto 2.0.0**:
https://snakebyteteam.github.io/3-PB/Documenti_interni/Norme_di_progetto/Norme_di_Progetto_v2.0.0.pdf
(consultato il 5/04/2026).
- **Vimar View4Life Capitolato di Ingegneria del Software Università di Padova 2025 - 2026:**
https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C9.pdf
(consultato il 5/04/2026).

1.2.2 Riferimenti Informativi

- **Glossario**:
https://snakebyteteam.github.io/3-PB/Documenti_interni/Glossario/glossario_v2.0.0.pdf
(consultato il 5/04/2026).

2 Tecnologie

Il progetto si basa su un insieme di tecnologie moderne e robuste, selezionate per le loro capacità di supportare efficacemente un'architettura esagonale e garantire scalabilità, affidabilità e manutenibilità del Sistema. La scelta tecnologica è stata guidata dalla necessità di creare una piattaforma per la gestione di residenze protette che possa operare in modo efficiente, mantenendo elevati standard di prestazioni, sicurezza e resilienza. Le tecnologie adottate sono state organizzate in categorie in base al loro ruolo all'interno dell'architettura: linguaggi di programmazione per lo sviluppo del codice, *framework* per la strutturazione del *frontend* e del *backend*, librerie per la gestione dell'autenticazione, soluzioni per la persistenza dei dati, strumenti per la virtualizzazione e il *deployment*, e infine tecnologie per la verifica della qualità del *software*. La scelta di *TypeScript*$_G$ come linguaggio trasversale a tutti i livelli applicativi, affiancato da *Angular*$_G$ per il *frontend* e *NestJS*$_G$ per il *backend*, ha permesso di mantenere una base di codice omogenea e fortemente tipizzata, riducendo il rischio di errori e facilitando la collaborazione tra i membri del gruppo. L'adozione di *Docker*$_G$ è stata dettata dal requisito esplicito del capitolato di seguire il principio di *Infrastructure as Code*: la containerizzazione garantisce che l'intero Sistema sia replicabile con un singolo comando, indipendentemente dall'ambiente di esecuzione, assicurando portabilità e uniformità tra i diversi ambienti di sviluppo, test e produzione. Di seguito sono elencate e descritte le tecnologie utilizzate, evidenziando le loro caratteristiche principali.

2.1 Linguaggi di programmazione

Tecnologia	Versione	Descrizione
Typescript	1.2	Linguaggio di programmazione *open source* sviluppato da *Microsoft*, estende la sintassi di *JavaScript* in modo che qualunque programma scritto in *JavaScript* sia anche in grado di funzionare con *TypeScript* senza nessuna modifica. È stato progettato per lo sviluppo di grandi applicazioni ed è destinato a essere compilato in *JavaScript* per poter essere interpretato da qualunque *web browser* o *app*.

Tabella 2: Linguaggi di programmazione

2.2 Framework

Tecnologia	Versione	Descrizione
NestJS	11.0.1	*Framework* per lo sviluppo di applicazioni *server-side*, costruito su *Node.js* e *TypeScript*. Fornisce un'architettura modulare e scalabile, ispirata a concetti di programmazione orientata agli oggetti.
Angular	15.0.0	*Framework* per lo sviluppo di applicazioni *web*, sviluppato da *Google*. Utilizza *TypeScript* e offre un'architettura basata su componenti, facilitando la creazione di interfacce utente dinamiche e reattive.

Tabella 3: Framework

2.3 Librerie

Tecnologia	Versione	Descrizione
Jwt	9.0.0	Libreria per la gestione dei *JSON Web Token* (JWT) in *Node.js*, utilizzata per l'autenticazione e l'autorizzazione degli utenti. Permette di creare, verificare e decodificare *token* JWT in modo semplice e sicuro.

Tabella 4: Librerie

2.4 Tecnologie per la persistenza dei dati

Tecnologia	Versione	Descrizione
PostgreSQL$_G$	18.0.0	Sistema di gestione di *database* relazionali *open source*, noto per la sua robustezza, scalabilità e conformità agli standard SQL.
TimescaleDB$_G$	2.26.3	Estensione di PostgreSQL per la gestione di dati *timeseries*.

Tabella 5: Tecnologie per la persistenza dei dati

2.5 Tecnologie per *deploy* e containerizzazione

Tecnologia	Versione	Descrizione
Docker	29.4.0	Piattaforma di containerizzazione che consente agli sviluppatori di creare, distribuire e eseguire applicazioni in ambienti isolati chiamati container.

Tabella 6: Tecnologie per *deploy* e containerizzazione

2.6 Tecnologie per l'analisi dinamica del codice

Tecnologia	Versione	Descrizione
Jest	29.5.14	*Framework* di testing *JavaScript* sviluppato da *Facebook*, progettato per essere semplice da configurare e utilizzare. Supporta il testing di unità, integrazione e *snapshot*.
Vitest	4.1.2	*Framework* di testing unitario basato su Vite, caratterizzato da un'esecuzione estremamente rapida, supporto nativo a ESM e piena compatibilità con l'API di Jest.

Tabella 7: Tecnologie per l'analisi dinamica del codice

2.7 Tecnologie per l'analisi statica del codice

Tecnologia	Versione	Descrizione
SonarQube$_G$	9.9.0	*Tool* di analisi statica del codice che consente di identificare problemi di qualità e sicurezza nel codice sorgente.

Tabella 8: Tecnologie per l'analisi statica del codice

3 Design pattern

3.1 Dependency injection

3.1.1 Descrizione

Il *pattern* ***Dependency Injection*** è una tecnica di progettazione *software* che consente di gestire le dipendenze tra i componenti in modo più flessibile e modulare. Invece di creare direttamente le dipendenze all'interno dei componenti, queste vengono fornite dall'esterno (tramite "injection").

Sono principalmente 2 i metodi per implementare la *Dependency Injection*:

- ***Constructor Injection***: le dipendenze vengono fornite attraverso il costruttore del componente;
- ***Setter Injection***: le dipendenze vengono fornite attraverso metodi *setter*.

3.1.2 Ragione dell'utilizzo

L'utilizzo del *pattern Dependency Injection* consente di migliorare la testabilità, la manutenibilità e la flessibilità del codice. In particolare, permette di:

- ridurre l'accoppiamento tra i componenti;
- centralizzare la gestione delle dipendenze, facilitando la configurazione e la manutenzione del Sistema;
- facilitare la sostituzione delle dipendenze con implementazioni alternative (utile durante la fase di test).

3.2 Adapter

3.2.1 Descrizione

L'***Adapter*** è un *pattern* di progettazione che consente di adattare l'interfaccia di una classe esistente a quella richiesta da un *client*, senza modificare il codice della classe esistente. In questo modo, è possibile integrare componenti con interfacce incompatibili senza dover riscrivere il codice esistente. Esistono 2 tipi di *adapter*:

- *class adapter*: utilizza l'ereditarietà per adattare l'interfaccia di una classe esistente a quella richiesta da un *client*;
- *object adapter*: utilizza la composizione per adattare l'interfaccia di una classe esistente a quella richiesta da un *client*.

3.2.2 Ragione dell'utilizzo

L'utilizzo del *pattern* ***Adapter*** consente di integrare componenti con interfacce incompatibili senza dover modificare il codice esistente, migliorando la flessibilità e la manutenibilità del Sistema. In particolare è stato utilizzato il *class adapter* che ereditano le porte primarie e secondarie, implementando così le funzionalità richieste dalle interfacce.

3.3 Strategy

3.3.1 Descrizione

Il *pattern* ***Strategy*** è un *pattern* comportamentale che consente di definire una famiglia di algoritmi, incapsularli in classi separate e renderli intercambiabili. Il *pattern* permette di variare l'algoritmo utilizzato indipendentemente dai *client* che ne fanno uso, delegando la scelta della strategia concreta al contesto che la invoca. I componenti principali del *pattern* sono:

- ***Strategy***: interfaccia comune a tutte le strategie concrete, che definisce il contratto che ciascuna di esse deve rispettare;
- ***ConcreteStrategy***: implementazione concreta dell'interfaccia *Strategy*, che realizza un algoritmo specifico;
- ***Context***: classe che mantiene un riferimento a una *Strategy* e la invoca per eseguire l'operazione richiesta, senza conoscere i dettagli dell'implementazione.

3.3.2 Ragione dell'utilizzo

Il *pattern* ***Strategy*** è stato adottato nella sezione *analytics* del Sistema per gestire le diverse tipologie di analisi sui dati degli impianti. Ciascuna metrica — quali il consumo energetico, la rilevazione delle presenze, le anomalie di impianto e la frequenza degli allarmi — è implementata come una strategia concreta che implementa l'interfaccia `AnalyticsStrategy`. Il servizio `AnalyticsService` funge da contesto: itera su tutte le strategie registrate, le esegue per l'impianto richiesto e aggrega i risultati in una lista di *plot*. Questo approccio consente di:

- aggiungere nuove tipologie di analytics senza modificare il codice esistente, nel rispetto del principio *Open/Closed*;
- isolare ciascun algoritmo di analisi in una classe dedicata, migliorando la leggibilità e la manutenibilità del codice;
- testare ogni strategia in modo indipendente, semplificando la fase di verifica e validazione.

3.4 Repository

3.4.1 Descrizione

Il *pattern* ***Repository*** è un *pattern* architetturale che introduce un livello di astrazione tra la logica applicativa e il meccanismo di accesso ai dati. Anziché interrogare direttamente il *database* all'interno dei servizi, le operazioni di persistenza vengono delegate a componenti dedicati — i *repository* — che espongono un'interfaccia uniforme indipendente dai dettagli implementativi dello strato di persistenza. I componenti principali del *pattern* sono:

- ***Repository Interface***: interfaccia che definisce il contratto per le operazioni di accesso ai dati, disaccoppiando la logica applicativa dall'implementazione concreta;
- ***Repository Implementation***: classe concreta che implementa l'interfaccia, gestendo direttamente la comunicazione con il database.

3.4.2 Ragione dell'utilizzo

Il *pattern* ***Repository*** è stato adottato per gestire tutte le operazioni di accesso al *database*. Ogni *repository* è composto da un'interfaccia — che funge da porta di uscita nell'architettura esagonale — e da un'implementazione concreta che esegue le *query* SQL tramite un *pool* di connessioni. Questo approccio consente di:

- isolare completamente la logica di *business* dai dettagli di accesso al *database*, nel rispetto del principio di separazione delle responsabilità;
- rendere sostituibile lo strato di persistenza senza impatto sui livelli superiori dell'architettura;
- semplificare il testing della logica applicativa, potendo sostituire l'implementazione concreta con un *mock* durante la fase di verifica.

3.5 Observer

3.5.1 Descrizione

Il *pattern* ***Observer*** è un *pattern* comportamentale che definisce una relazione uno-a-molti tra oggetti: quando un oggetto — detto *subject* o *observable* — cambia stato, tutti gli oggetti dipendenti da esso — detti *observer* o *subscriber* — vengono notificati e aggiornati automaticamente. I componenti principali del *pattern* sono:

- ***Observable***: oggetto che mantiene lo stato e notifica i propri *subscriber* al verificarsi di un evento o di un cambiamento;
- ***Observer/Subscriber***: oggetto che si iscrive all'*observable* per ricevere le notifiche e reagire di conseguenza;
- ***Subscription***: rappresenta il legame tra *observable* e *subscriber*.

3.5.2 Ragione dell'utilizzo

Il *pattern* ***Observer*** è stato adottato nel *frontend Angular* tramite la libreria *RxJS*, che ne fornisce un'implementazione matura e integrata nell'ecosistema del *framework*. In particolare, il *pattern* è stato impiegato per gestire la ricezione delle notifiche *push* provenienti dal *backend* — quali allarmi e aggiornamenti di stato degli impianti — e per aggiornare la UI in modo reattivo al loro arrivo, senza necessità di *polling*. Questo approccio consente di reagire in tempo reale agli eventi di impianto, garantendo all'operatore sanitario una visione aggiornata della situazione senza intervento manuale.

3.6 Architettura

3.7 Architettura logica

Il Sistema è progettato seguendo l'architettura esagonale, che prevede una chiara separazione tra il nucleo logico del sistema e le interfacce esterne.

Il nucleo logico, rappresentato dal dominio e dalle regole di *business*, è isolato dai dettagli di implementazione e dalle dipendenze esterne, come *database*, *framework* e librerie.

Le porte definiscono le interfacce attraverso cui il nucleo interagisce con il mondo esterno, ne esistono di due tipi:

- le porte primarie, che rappresentano i punti di ingresso al Sistema ovvero consentono a componenti esterni di interagire con il nucleo logico;
- le porte secondarie, che rappresentano i punti di uscita che permettono al nucleo logico di interagire con componenti esterni, come *database* o servizi di terze parti.

Gli adattatori, invece, sono le implementazioni concrete delle porte, in base al tipo di porta che implementano, si distinguono in:

- adattatori primari, che implementano le porte primarie e consentono a componenti esterni di interagire con il nucleo logico;
- adattatori secondari, che implementano le porte secondarie e consentono al nucleo logico l'integrazione con specifiche tecnologie o servizi.

I servizi fanno parte della *business logic* e rappresentano le funzionalità principali del Sistema, implementando le regole di *business* e coordinando le interazioni tra i vari componenti del nucleo logico.

3.8 Architettura di deployment

L'architettura di *deployment* del Sistema è composta da tre *container Docker*$_G$: *backend* monolitico, *frontend* come *Single Page Application* sviluppata in Angular, i cui file statici sono serviti, in produzione, da nginx, e *database* PostgreSQL con TimescaleDB.
L'orchestrazione dei tre *container* avviene tramite l'utilizzo di Docker Compose.

I container di ***backend*** e ***frontend*** comunicano tra loro tramite API *RESTful*$_G$.

Il *backend* è responsabile della gestione della logica di business, dell'accesso ai dati, dell'autenticazione e della comunicazione con il *server* Vimar tramite protocollo *API KNX IoT 3rd-party*$_G$. Il *frontend* si occupa della presentazione e dell'interazione con l'utente.

Attualmente tutti i container si trovano in un singolo nodo.

4 Moduli implementati

4.1 Backend

Il *backend* è composto dai seguenti moduli:

- **Alarms**: gestisce la creazione, la modifica, l'abilitazione e l'eliminazione delle regole di allarme, nonché il monitoraggio e la risoluzione degli allarmi attivi;
- **Analytics**: si occupa del calcolo e dell'aggregazione dei dati statistici degli impianti, producendo i grafici relativi a consumi, presenze, temperature e allarmi;
- **Auth Vimar**: gestisce l'integrazione con il sistema di autenticazione OAuth2 di Vimar, consentendo il collegamento dell'account MyVimar al Sistema;
- **Auth**: si occupa dell'autenticazione degli utenti interni al Sistema, inclusa la gestione dei token JWT e il flusso di cambio password al primo accesso;
- **Cache**: fornisce un livello di memorizzazione temporanea dei dati più frequentemente acceduti, al fine di ridurre il carico sul *database* e migliorare le prestazioni del Sistema;
- **Device**: gestisce il recupero e l'aggiornamento dello stato dei dispositivi IoT presenti negli appartamenti, interfacciandosi con le API KNX IoT di Vimar;
- **Notification**: si occupa della generazione e della distribuzione delle notifiche agli utenti in seguito allo scatto di un allarme;
- **Plant**: gestisce le informazioni relative agli impianti *View Wireless*, incluse le stanze e i dispositivi associati a ciascun appartamento;
- **Subscription**: gestisce le sottoscrizioni alle notifiche *push* provenienti dagli impianti tramite il meccanismo di *Subscription* dello standard KNX IoT;
- **Users**: gestisce la creazione, la modifica e l'eliminazione degli utenti del Sistema, inclusa la generazione delle password temporanee per i nuovi Operatori Sanitari;
- **Wards**: gestisce i reparti (aggregati di appartamenti) della residenza protetta, comprese le assegnazioni degli appartamenti e degli Operatori Sanitari a ciascun reparto.

4.1.1 Alarms

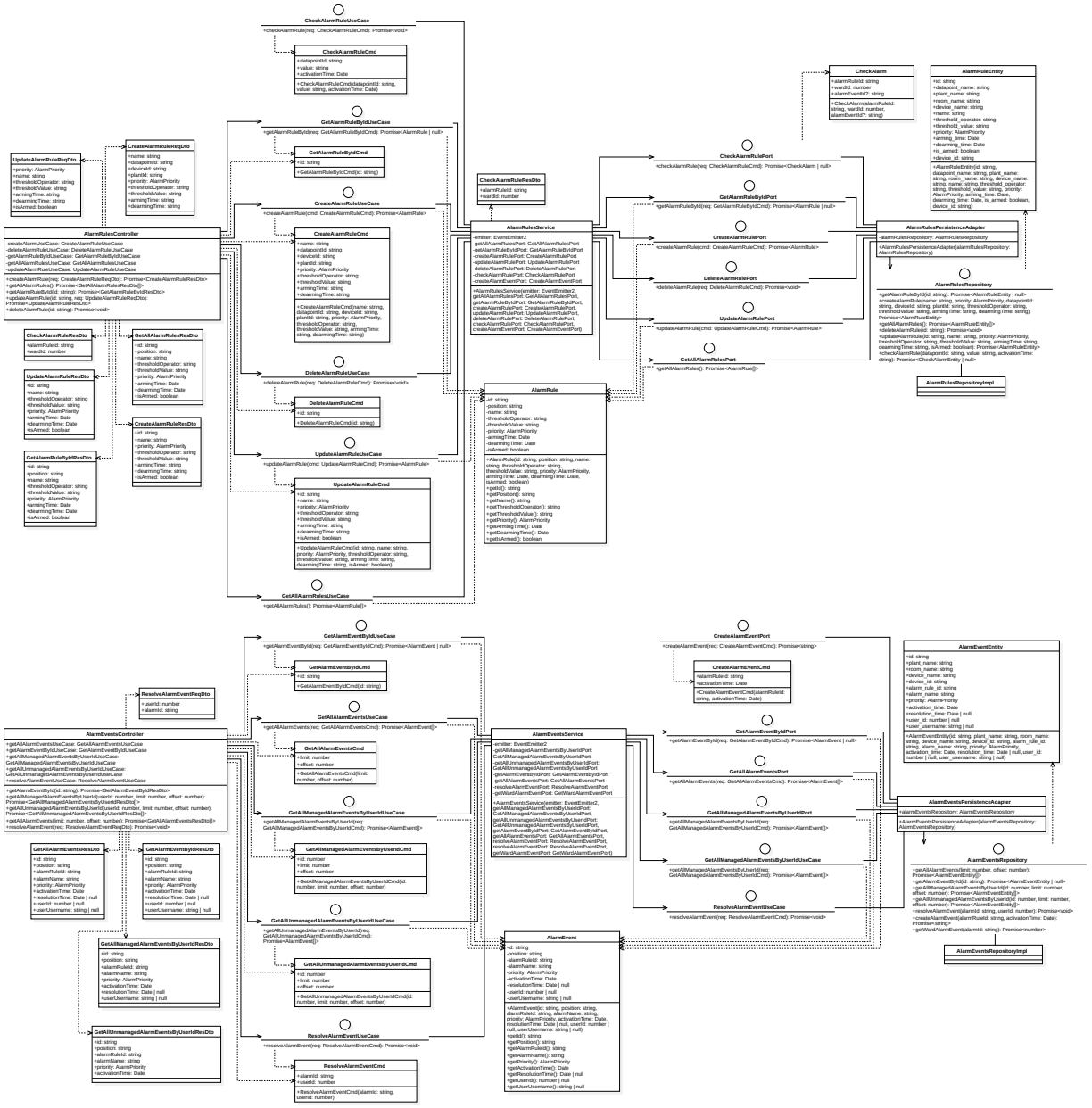

Figura 1: Diagramma delle classi del modulo Alarms

4.1.1.1 CreateAlarmRuleReqDto

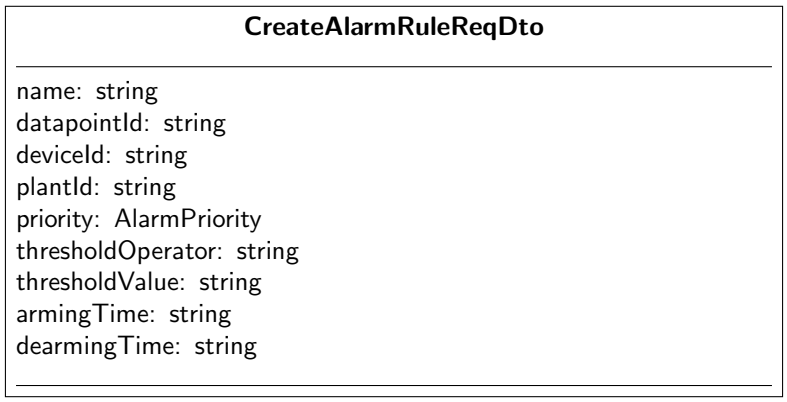

Figura 2: Diagramma della classe CreateAlarmRuleReqDto

Descrizione: DTO di richiesta per la creazione di una nuova regola di allarme

4.1.1.2 ResolveAlarmEventReqDto

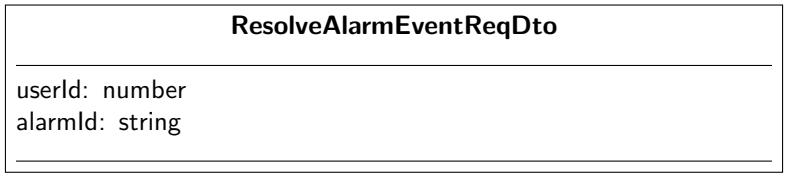

Figura 3: Diagramma della classe ResolveAlarmEventReqDto

Descrizione: DTO di richiesta per la risoluzione di un evento di allarme

4.1.1.3 UpdateAlarmRuleReqDto

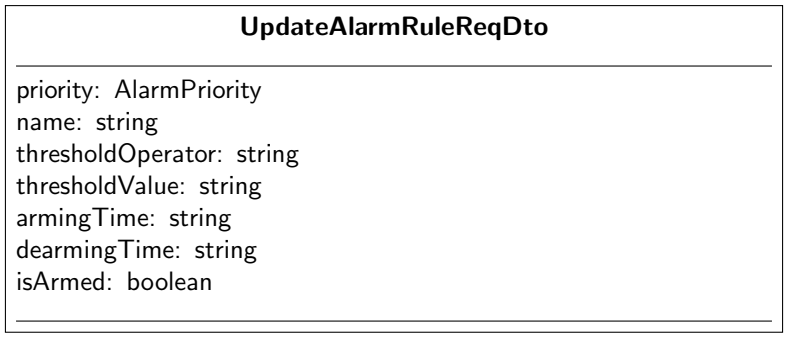

Figura 4: Diagramma della classe UpdateAlarmRuleReqDto

Descrizione: DTO di richiesta per l'aggiornamento di una regola di allarme esistente

4.1.1.4 CheckAlarmRuleResDto

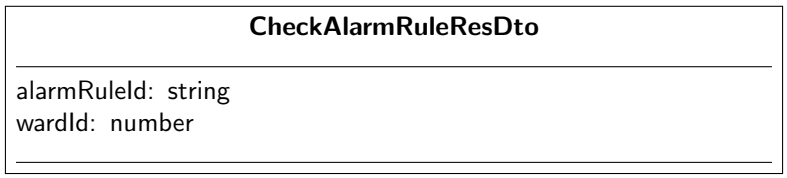

Figura 5: Diagramma della classe CheckAlarmRuleResDto

Descrizione: DTO di risposta per la verifica di una regola di allarme

4.1.1.5 CreateAlarmRuleResDto

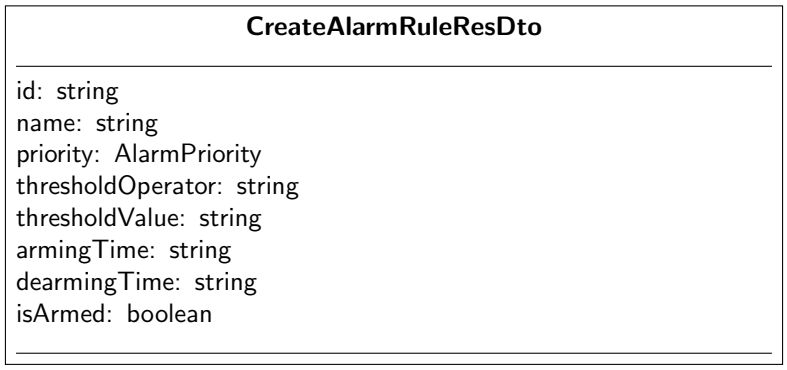

Figura 6: Diagramma della classe CreateAlarmRuleResDto

Descrizione: DTO di risposta restituito dopo la creazione di una regola di allarme

4.1.1.6 GetAlarmEventByIdResDto

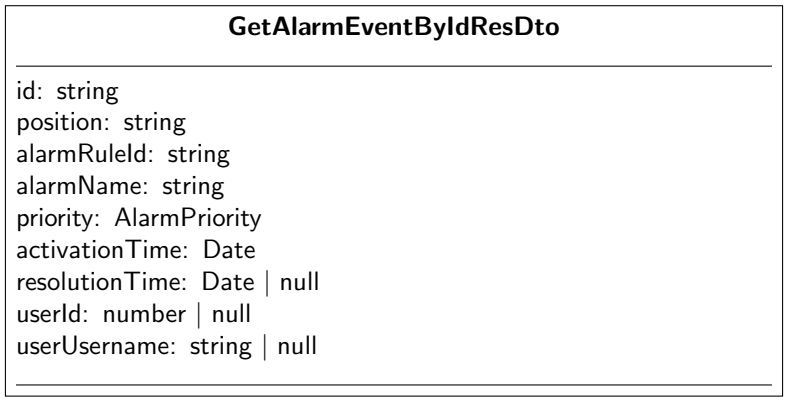

Figura 7: Diagramma della classe GetAlarmEventByIdResDto

Descrizione: DTO di risposta contenente i dettagli di un singolo evento di allarme

4.1.1.7 GetAlarmRuleByIdResDto

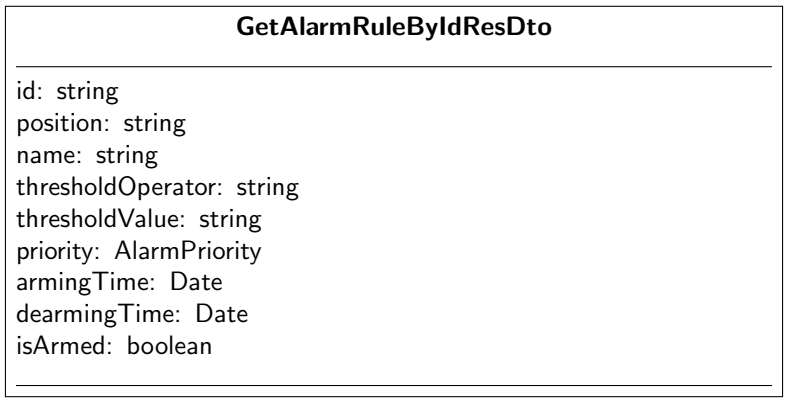

Figura 8: Diagramma della classe GetAlarmRuleByIdResDto

Descrizione: DTO di risposta contenente i dettagli di una singola regola di allarme

4.1.1.8 GetAllAlarmEventsResDto

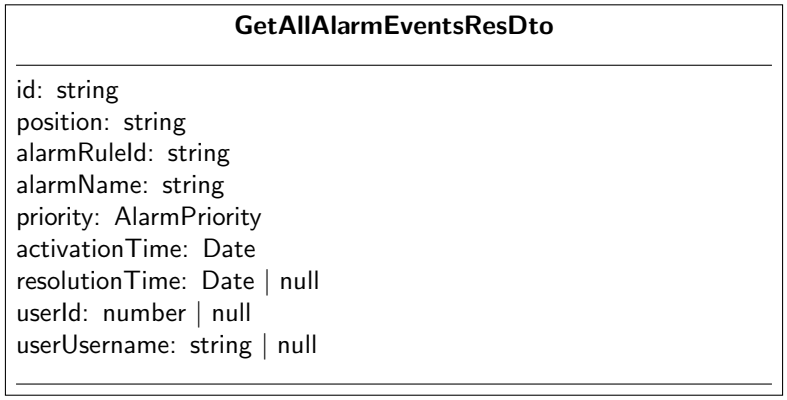

Figura 9: Diagramma della classe GetAllAlarmEventsResDto

Descrizione: DTO di risposta per un elemento della lista di tutti gli eventi di allarme

4.1.1.9 GetAllAlarmRulesResDto

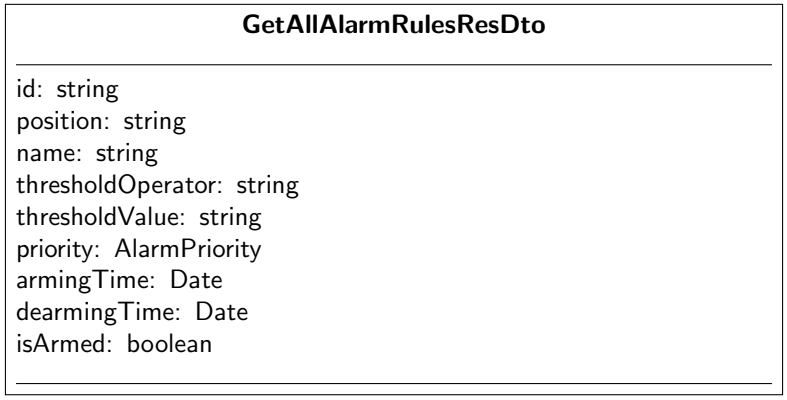

Figura 10: Diagramma della classe GetAllAlarmRulesResDto

Descrizione: DTO di risposta per un elemento della lista di tutte le regole di allarme

4.1.1.10 GetAllManagedAlarmEventsByUserIdResDto

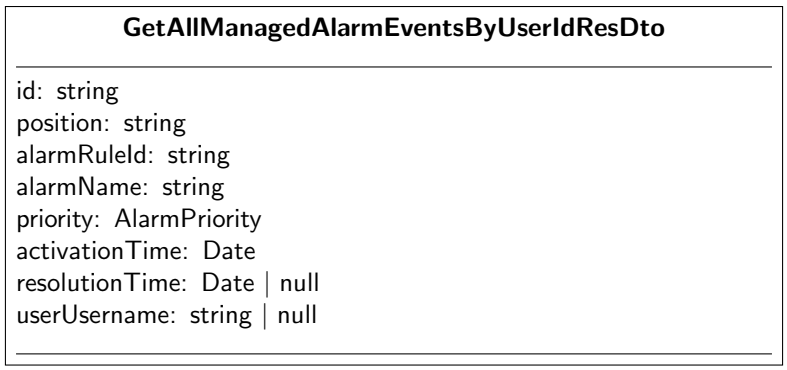

Figura 11: Diagramma della classe GetAllManagedAlarmEventsByUserIdResDto

Descrizione: DTO di risposta per un elemento della lista degli eventi gestiti da un utente specifico

4.1.1.11 GetAllUnmanagedAlarmEventsByUserIdResDto

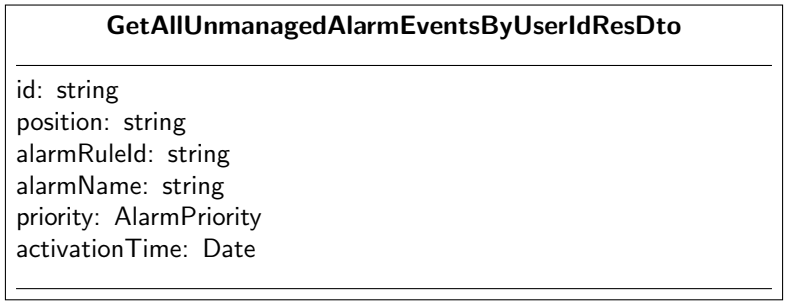

Figura 12: Diagramma della classe GetAllUnmanagedAlarmEventsByUserIdResDto

Descrizione: DTO di risposta per un elemento della lista degli eventi non ancora gestiti assegnati a un utente

4.1.1.12 UpdateAlarmRuleResDto

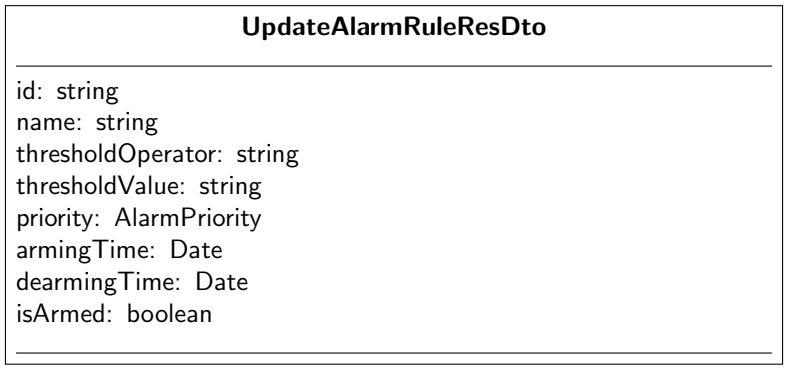

Figura 13: Diagramma della classe UpdateAlarmRuleResDto

Descrizione: DTO di risposta restituito dopo l'aggiornamento di una regola di allarme

4.1.1.13 AlarmEvent

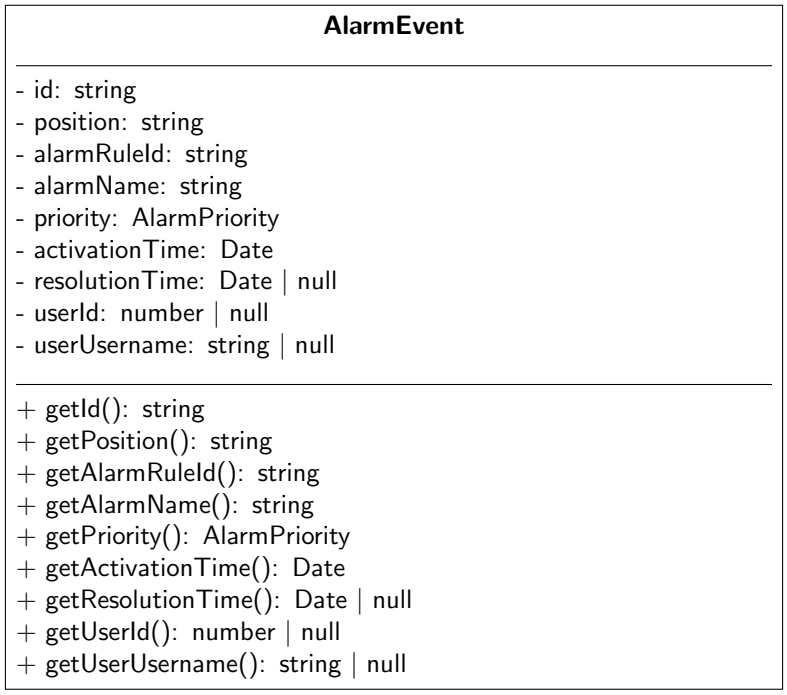

Figura 14: Diagramma della classe AlarmEvent

Descrizione: Modello di dominio che rappresenta un evento di allarme generato da una regola

Descrizione dei metodi della classe:

- `getId(): string`: Restituisce l'identificativo univoco dell'evento
- `getPosition(): string`: Restituisce la posizione del dispositivo
- `getAlarmRuleId(): string`: Restituisce l'identificativo della regola associata
- `getAlarmName(): string`: Restituisce il nome dell'allarme
- `getPriority(): AlarmPriority`: Restituisce la priorità dell'evento
- `getActivationTime(): Date`: Restituisce il timestamp di attivazione
- `getResolutionTime(): Date | null`: Restituisce il timestamp di risoluzione
- `getUserId(): number | null`: Restituisce l'identificativo dell'utente risolutore
- `getUserUsername(): string | null`: Restituisce lo username dell'utente risolutore

4.1.1.14 AlarmRule

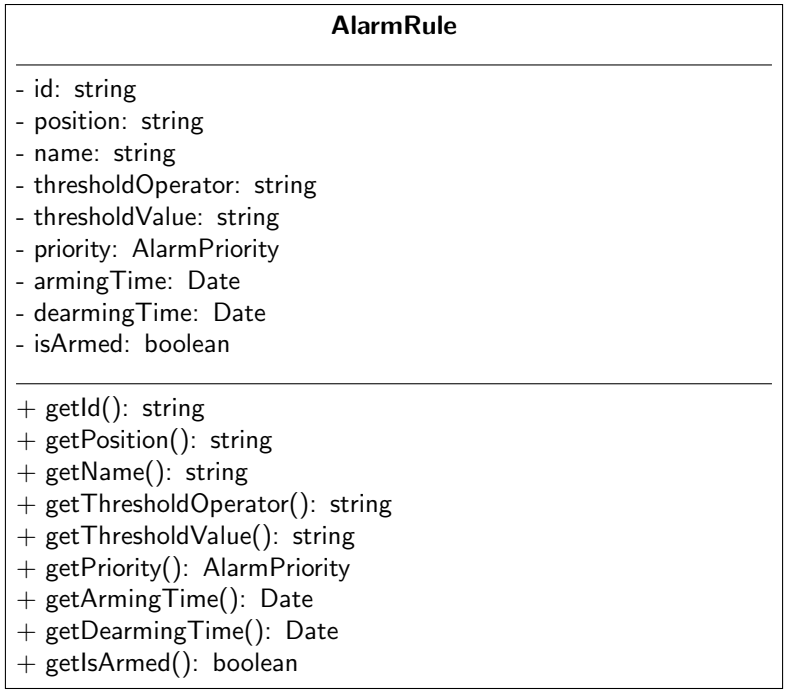

Figura 15: Diagramma della classe AlarmRule

Descrizione: Modello di dominio che rappresenta una regola di allarme configurabile

Descrizione dei metodi della classe:

- `getId(): string`: Restituisce l'identificativo univoco della regola
- `getPosition(): string`: Restituisce la posizione del dispositivo monitorato
- `getName(): string`: Restituisce il nome della regola
- `getThresholdOperator(): string`: Restituisce l'operatore di confronto della soglia
- `getThresholdValue(): string`: Restituisce il valore soglia
- `getPriority(): AlarmPriority`: Restituisce la priorità della regola
- `getArmingTime(): Date`: Restituisce l'orario di attivazione
- `getDearmingTime(): Date`: Restituisce l'orario di disattivazione
- `getIsArmed(): boolean`: Restituisce lo stato di armamento della regola

4.1.1.15 CheckAlarm

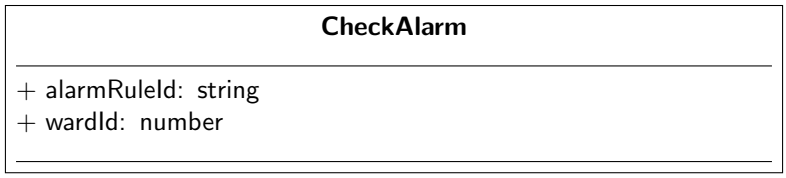

Figura 16: Diagramma della classe CheckAlarm

Descrizione: Oggetto di dominio risultante dalla verifica di una regola di allarme

4.1.1.16 AlarmEventsController

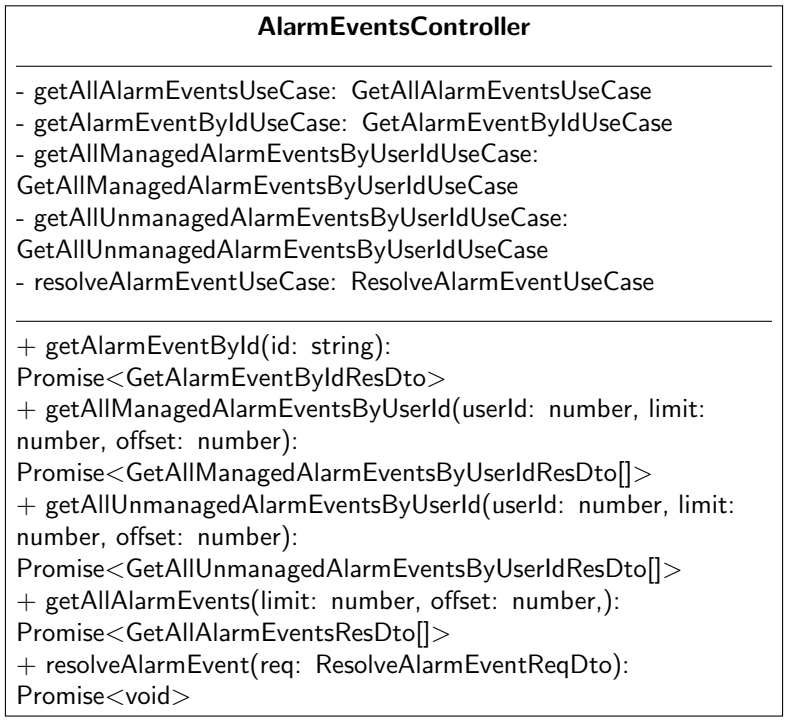

Figura 17: Diagramma della classe AlarmEventsController

Descrizione: Controller REST che gestisce le richieste HTTP relative agli eventi di allarme

Descrizione dei metodi della classe:

- `getAlarmEventById(id: string): Promise<GetAlarmEventByIdResDto>`: Espone l'endpoint per il recupero di un evento di allarme tramite il suo identificativo
- `getAllManagedAlarmEventsByUserId(userId: number, limit: number, offset: number): Promise<GetAllManagedAlarmEventsByUserIdResDto[]>`: Espone l'endpoint per il recupero paginato degli eventi gestiti da un utente
- `getAllUnmanagedAlarmEventsByUserId(userId: number, limit: number, offset: number): Promise<GetAllUnmanagedAlarmEventsByUserIdResDto[]>`: Espone l'endpoint per il recupero paginato degli eventi non gestiti assegnati a un utente
- `getAllAlarmEvents(limit: number, offset: number,): Promise<GetAllAlarmEventsResDto[]>`: Espone l'endpoint per il recupero paginato di tutti gli eventi di allarme
- `resolveAlarmEvent(req: ResolveAlarmEventReqDto): Promise<void>`: Espone l'endpoint per la risoluzione di un evento di allarme da parte di un utente

4.1.1.17 AlarmRulesController

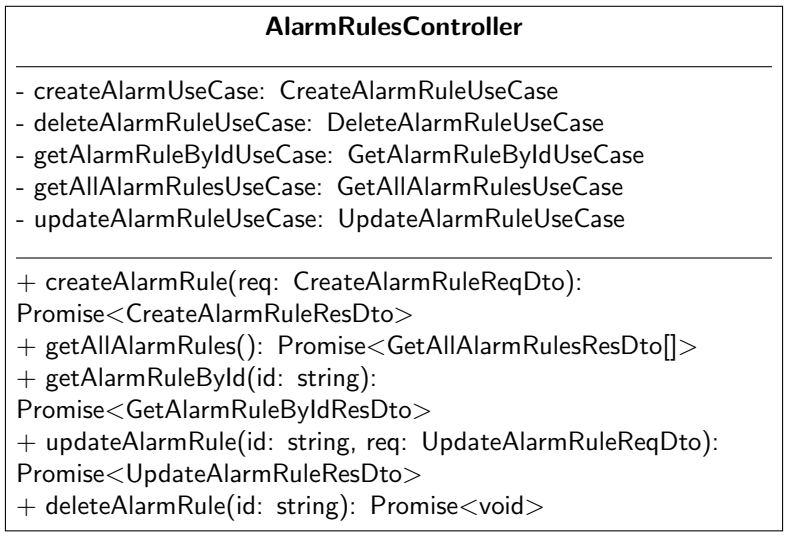

Figura 18: Diagramma della classe AlarmRulesController

Descrizione: Controller REST che gestisce le richieste HTTP relative alle regole di allarme

Descrizione dei metodi della classe:

- `createAlarmRule(req: CreateAlarmRuleReqDto): Promise<CreateAlarmRuleResDto>`: Espone l'endpoint per la creazione di una nuova regola di allarme
- `getAllAlarmRules(): Promise<GetAllAlarmRulesResDto[]>`: Espone l'endpoint per il recupero di tutte le regole di allarme
- `getAlarmRuleById(id: string): Promise<GetAlarmRuleByIdResDto>`: Espone l'endpoint per il recupero di una regola tramite il suo identificativo
- `updateAlarmRule(id: string, req: UpdateAlarmRuleReqDto): Promise<UpdateAlarmRuleResDto>`: Espone l'endpoint per l'aggiornamento di una regola di allarme esistente
- `deleteAlarmRule(id: string): Promise<void>`: Espone l'endpoint per l'eliminazione di una regola di allarme

4.1.1.18 CheckAlarmRuleCmd

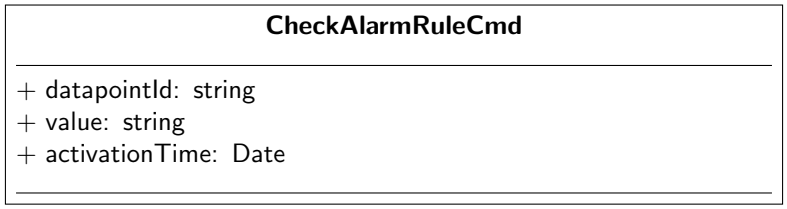

Figura 19: Diagramma della classe CheckAlarmRuleCmd

Descrizione: Comando per la verifica di una regola di allarme rispetto al valore corrente di un datapoint

4.1.1.19 CreateAlarmEventCmd

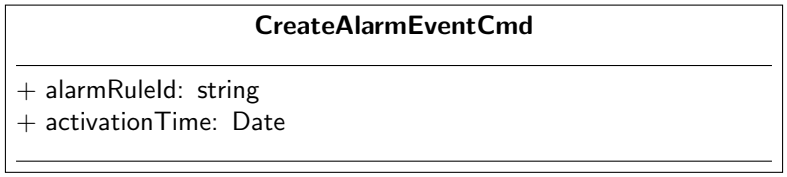

Figura 20: Diagramma della classe CreateAlarmEventCmd

Descrizione: Comando per la creazione di un nuovo evento di allarme

4.1.1.20 CreateAlarmRuleCmd

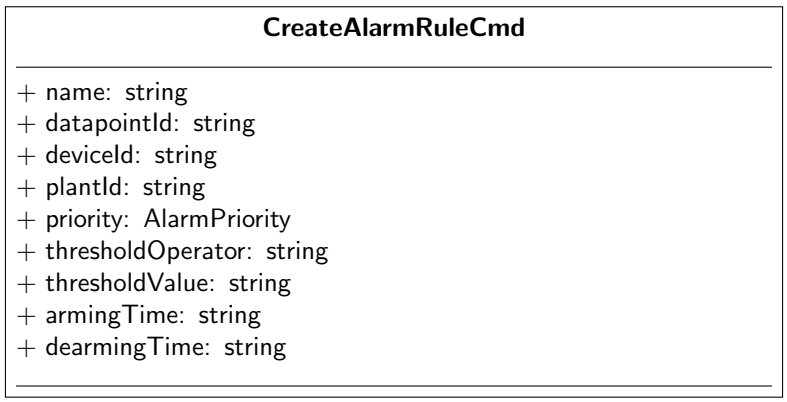

Figura 21: Diagramma della classe CreateAlarmRuleCmd

Descrizione: Comando per la creazione di una nuova regola di allarme

4.1.1.21 DeleteAlarmRuleCmd

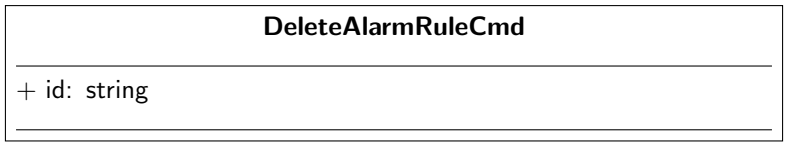

Figura 22: Diagramma della classe DeleteAlarmRuleCmd

Descrizione: Comando per l'eliminazione di una regola di allarme

4.1.1.22 GetAlarmEventByIdCmd

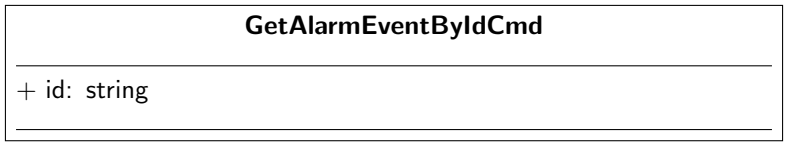

Figura 23: Diagramma della classe GetAlarmEventByIdCmd

Descrizione: Comando per il recupero di un evento di allarme tramite identificativo

4.1.1.23 GetAlarmRuleByIdCmd

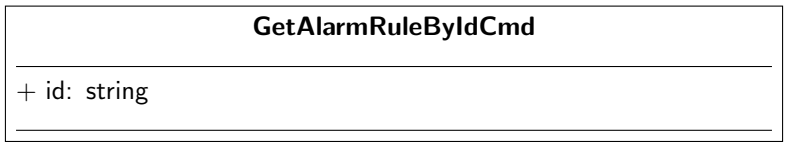

Figura 24: Diagramma della classe GetAlarmRuleByIdCmd

Descrizione: Comando per il recupero di una regola di allarme tramite identificativo

4.1.1.24 GetAllAlarmEventsCmd

Figura 25: Diagramma della classe GetAllAlarmEventsCmd

Descrizione: Comando per il recupero paginato di tutti gli eventi di allarme

4.1.1.25 GetAllManagedAlarmEventsByUserIdCmd

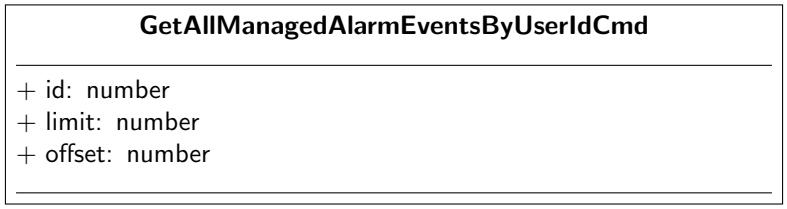

Figura 26: Diagramma della classe GetAllManagedAlarmEventsByUserIdCmd

Descrizione: Comando per il recupero paginato degli eventi di allarme gestiti da un utente

4.1.1.26 GetAllUnmanagedAlarmEventsByUserIdCmd

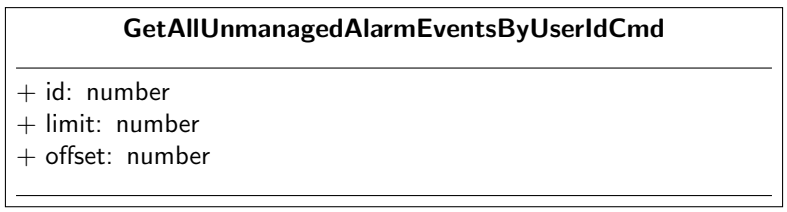

Figura 27: Diagramma della classe GetAllUnmanagedAlarmEventsByUserIdCmd

Descrizione: Comando per il recupero paginato degli eventi di allarme non ancora gestiti assegnati a un utente

4.1.1.27 ResolveAlarmEventCmd

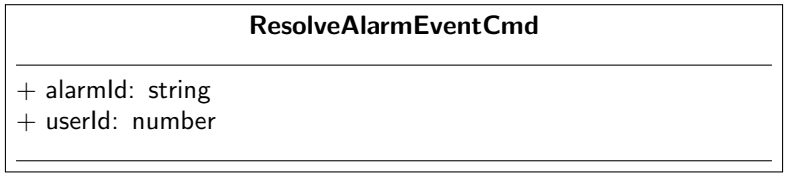

Figura 28: Diagramma della classe ResolveAlarmEventCmd

Descrizione: Comando per la risoluzione di un evento di allarme attivo

4.1.1.28 TriggerActiveAlarmCmd

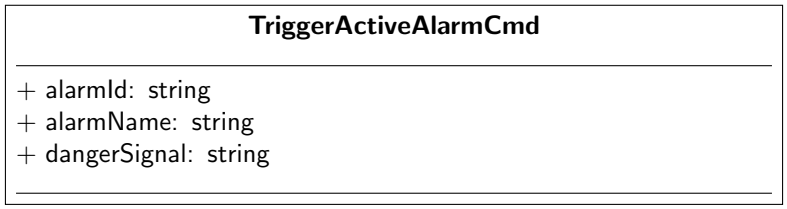

Figura 29: Diagramma della classe TriggerActiveAlarmCmd

Descrizione: Comando per la notifica e propagazione di un allarme attivo

4.1.1.29 GetWardAlarmEventCmd

Figura 30: Diagramma della classe GetWardAlarmEventCmd

Descrizione: Comando per il recupero del ward associato a un evento di allarme

4.1.1.30 UpdateAlarmRuleCmd

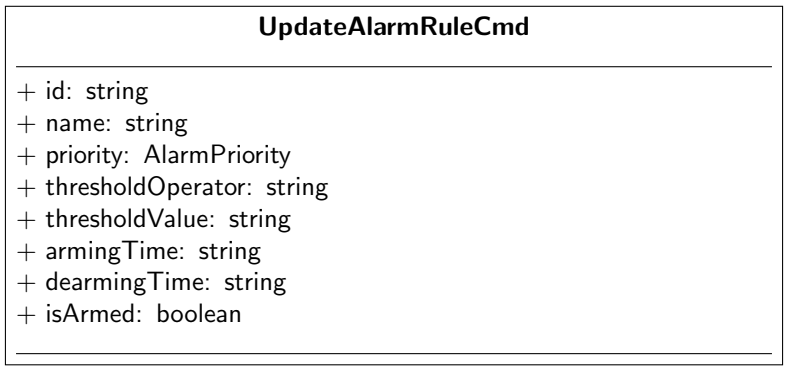

Figura 31: Diagramma della classe UpdateAlarmRuleCmd

Descrizione: Comando per l'aggiornamento di una regola di allarme esistente

4.1.1.31 CheckAlarmRuleUseCase

Figura 32: Diagramma dell'interfaccia CheckAlarmRuleUseCase

Descrizione: Interfaccia del use case per la verifica di una regola di allarme

Descrizione dei metodi dell'interfaccia:

- `checkAlarmRule(req: CheckAlarmRuleCmd): Promise<void>`: Verifica se il valore corrente di un datapoint viola la soglia di una regola di allarme

4.1.1.32 CreateAlarmRuleUseCase

Figura 33: Diagramma dell'interfaccia CreateAlarmRuleUseCase

Descrizione: Interfaccia del use case per la creazione di una nuova regola di allarme

Descrizione dei metodi dell'interfaccia:

- `createAlarmRule(cmd: CreateAlarmRuleCmd): Promise<AlarmRule>`: Crea e persiste una nuova regola di allarme, restituendo il modello creato

4.1.1.33 DeleteAlarmRuleUseCase

DeleteAlarmRuleUseCase

+ deleteAlarmRule(req: DeleteAlarmRuleCmd): Promise<void>

Figura 34: Diagramma dell'interfaccia DeleteAlarmRuleUseCase

Descrizione: Interfaccia del use case per l'eliminazione di una regola di allarme

Descrizione dei metodi dell'interfaccia:

- `deleteAlarmRule(req: DeleteAlarmRuleCmd): Promise<void>`: Elimina la regola di allarme identificata dal comando

4.1.1.34 GetAlarmEventByIdUseCase

GetAlarmEventByIdUseCase

+ getAlarmEventById(req: GetAlarmEventByIdCmd): Promise<AlarmEvent | null>

Figura 35: Diagramma dell'interfaccia GetAlarmEventByIdUseCase

Descrizione: Interfaccia del use case per il recupero di un singolo evento di allarme

Descrizione dei metodi dell'interfaccia:

- `getAlarmEventById(req: GetAlarmEventByIdCmd): Promise<AlarmEvent | null>`: Recupera un evento di allarme tramite identificativo, restituisce null se non trovato

4.1.1.35 GetAlarmRuleByIdUseCase

GetAlarmRuleByIdUseCase

+ getAlarmRuleById(req: GetAlarmRuleByIdCmd): Promise<AlarmRule | null>

Figura 36: Diagramma dell'interfaccia GetAlarmRuleByIdUseCase

Descrizione: Interfaccia del use case per il recupero di una singola regola di allarme

Descrizione dei metodi dell'interfaccia:

- `getAlarmRuleById(req: GetAlarmRuleByIdCmd): Promise<AlarmRule | null>`: Recupera una regola di allarme tramite identificativo, restituisce null se non trovata

4.1.1.36 GetAllAlarmEventsUseCase

GetAllAlarmEventsUseCase

+ getAllAlarmEvents(req: GetAllAlarmEventsCmd): Promise<AlarmEvent[]>

Figura 37: Diagramma dell'interfaccia GetAllAlarmEventsUseCase

Descrizione: Interfaccia del use case per il recupero paginato di tutti gli eventi di allarme

Descrizione dei metodi dell'interfaccia:

- `getAllAlarmEvents(req: GetAllAlarmEventsCmd): Promise<AlarmEvent[]>`: Recupera in modo paginato tutti gli eventi di allarme presenti nel sistema

4.1.1.37 GetAllAlarmRulesUseCase

GetAllAlarmRulesUseCase

+ getAllAlarmRules(): Promise<AlarmRule[]>

Figura 38: Diagramma dell'interfaccia GetAllAlarmRulesUseCase

Descrizione: Interfaccia del use case per il recupero di tutte le regole di allarme

Descrizione dei metodi dell'interfaccia:

- `getAllAlarmRules(): Promise<AlarmRule[]>`: Recupera tutte le regole di allarme configurate nel sistema

4.1.1.38 GetAllManagedAlarmEventsByUserIdUseCase

GetAllManagedAlarmEventsByUserIdUseCase

+ getAllManagedAlarmEventsByUserId(req: GetAllManagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>

Figura 39: Diagramma dell'interfaccia GetAllManagedAlarmEventsByUserIdUseCase

Descrizione: Interfaccia del use case per il recupero degli eventi gestiti da un utente

Descrizione dei metodi dell'interfaccia:

- `getAllManagedAlarmEventsByUserId(req: GetAllManagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>`: Recupera in modo paginato gli eventi di allarme già gestiti da un utente specifico

4.1.1.39 GetAllUnmanagedAlarmEventsByUserIdUseCase

GetAllUnmanagedAlarmEventsByUserIdUseCase

+ getAllUnmanagedAlarmEventsByUserId(req: GetAllUnmanagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>

Figura 40: Diagramma dell'interfaccia GetAllUnmanagedAlarmEventsByUserIdUseCase

Descrizione: Interfaccia del use case per il recupero degli eventi non gestiti di un utente

Descrizione dei metodi dell'interfaccia:

- `getAllUnmanagedAlarmEventsByUserId(req: GetAllUnmanagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>`: Recupera in modo paginato gli eventi di allarme non ancora gestiti assegnati a un utente

4.1.1.40 ResolveAlarmEventUseCase

ResolveAlarmEventUseCase

+ resolveAlarmEvent(req: ResolveAlarmEventCmd): Promise<void>

Figura 41: Diagramma dell'interfaccia ResolveAlarmEventUseCase

Descrizione: Interfaccia del use case per la risoluzione di un evento di allarme

Descrizione dei metodi dell'interfaccia:

- `resolveAlarmEvent(req: ResolveAlarmEventCmd): Promise<void>`: Segna un evento di allarme come risolto, associandolo all'utente che lo ha gestito

4.1.1.41 TriggerActiveAlarmUseCase

TriggerActiveAlarmUseCase

+ triggerActiveAlarm(cmd: TriggerActiveAlarmCmd): Promise<void>

Figura 42: Diagramma dell'interfaccia TriggerActiveAlarmUseCase

Descrizione: Interfaccia del use case per la notifica e propagazione di un allarme attivo

Descrizione dei metodi dell'interfaccia:

- `triggerActiveAlarm(cmd: TriggerActiveAlarmCmd): Promise<void>`: Propaga la notifica di un allarme attivo ai sistemi e agli utenti interessati

4.1.1.42 UpdateAlarmRuleUseCase

UpdateAlarmRuleUseCase

+ updateAlarmRule(cmd: UpdateAlarmRuleCmd): Promise<AlarmRule>

Figura 43: Diagramma dell'interfaccia UpdateAlarmRuleUseCase

Descrizione: Interfaccia del use case per l'aggiornamento di una regola di allarme

Descrizione dei metodi dell'interfaccia:

- `updateAlarmRule(cmd: UpdateAlarmRuleCmd): Promise<AlarmRule>`: Aggiorna i parametri di una regola di allarme esistente e restituisce il modello aggiornato

4.1.1.43 AlarmEventsService

AlarmEventsService

- emitter: EventEmitter2
- getAllManagedAlarmEventsByUserIdPort: GetAllManagedAlarmEventsByUserIdPort
- getAllUnmanagedAlarmEventsByUserIdPort: GetAllUnmanagedAlarmEventsByUserIdPort
- getAlarmEventByIdPort: GetAlarmEventByIdPort
- getAllAlarmEventsPort: GetAllAlarmEventsPort
- resolveAlarmEventPort: ResolveAlarmEventPort
- getWardAlarmEventPort: GetWardAlarmEventPort

+ getAlarmEventById(req: GetAlarmEventByIdCmd): Promise<AlarmEvent | null>
+ getAllAlarmEvents(req: GetAllAlarmEventsCmd): Promise<AlarmEvent[]>
+ getAllManagedAlarmEventsByUserId(req: GetAllManagedAlarmEventsByUserIdCmd): romise<AlarmEvent[]>
+ getAllUnmanagedAlarmEventsByUserId(req: GetAllUnmanagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>
+ resolveAlarmEvent(req: ResolveAlarmEventCmd): Promise<void>

Figura 44: Diagramma della classe AlarmEventsService

Descrizione: Servizio applicativo che implementa i use case relativi agli eventi di allarme

Descrizione dei metodi della classe:

- `getAlarmEventById(req: GetAlarmEventByIdCmd): Promise<AlarmEvent | null>`: Recupera un evento di allarme per identificativo delegando alla porta di persistenza

- `getAllAlarmEvents(req: GetAllAlarmEventsCmd): Promise<AlarmEvent[]>`: Recupera in modo paginato tutti gli eventi di allarme
- `getAllManagedAlarmEventsByUserId(req: GetAllManagedAlarmEventsByUserIdCmd): romise<AlarmEvent[]>`: Recupera gli eventi gestiti da un utente delegando alla porta di persistenza
- `getAllUnmanagedAlarmEventsByUserId(req: GetAllUnmanagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>`: Recupera gli eventi non gestiti di un utente delegando alla porta di persistenza
- `resolveAlarmEvent(req: ResolveAlarmEventCmd): Promise<void>`: Risolve un evento di allarme e notifica gli interessati tramite l'emettitore di eventi

4.1.1.44 AlarmRulesService

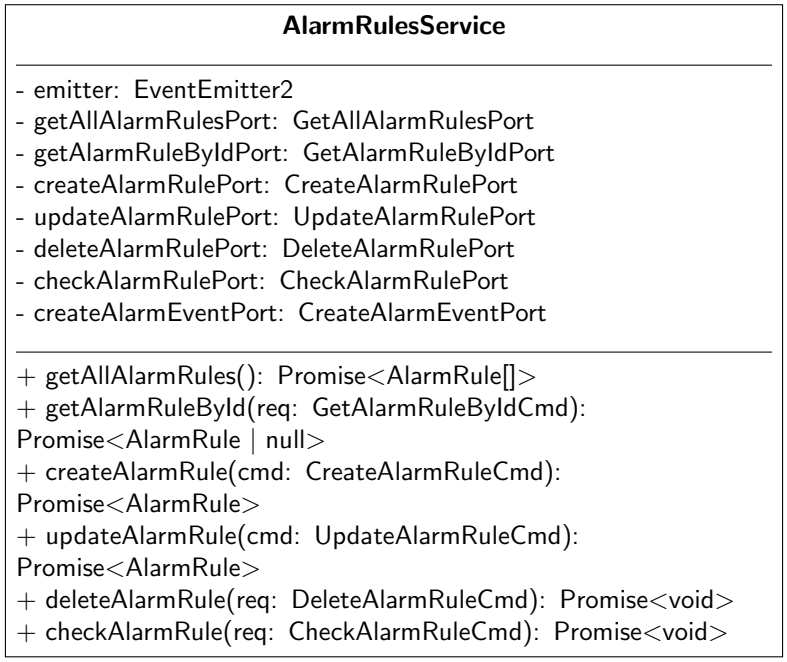

Figura 45: Diagramma della classe AlarmRulesService

Descrizione: Servizio applicativo che implementa i use case relativi alle regole di allarme

Descrizione dei metodi della classe:

- `getAllAlarmRules(): Promise<AlarmRule[]>`: Recupera tutte le regole di allarme configurate nel sistema
- `getAlarmRuleById(req: GetAlarmRuleByIdCmd): Promise<AlarmRule | null>`: Recupera una regola di allarme per identificativo
- `createAlarmRule(cmd: CreateAlarmRuleCmd): Promise<AlarmRule>`: Crea una nuova regola di allarme e pubblica l'evento di creazione
- `updateAlarmRule(cmd: UpdateAlarmRuleCmd): Promise<AlarmRule>`: Aggiorna una regola di allarme e pubblica l'evento di aggiornamento
- `deleteAlarmRule(req: DeleteAlarmRuleCmd): Promise<void>`: Elimina una regola di allarme e pubblica l'evento di eliminazione

- `checkAlarmRule(req: CheckAlarmRuleCmd): Promise<void>`: Verifica una regola di allarme e genera un evento se la soglia è violata

4.1.1.45 CheckAlarmRulePort

CheckAlarmRulePort

+ checkAlarmRule(req: CheckAlarmRuleCmd): Promise<CheckAlarm | null>

Figura 46: Diagramma dell'interfaccia CheckAlarmRulePort

Descrizione: Porta di uscita per la verifica di una regola di allarme verso il livello di persistenza

Descrizione dei metodi dell'interfaccia:

- `checkAlarmRule(req: CheckAlarmRuleCmd): Promise<CheckAlarm | null>`: Verifica la regola rispetto al valore del datapoint, restituisce il risultato o null se non violata

4.1.1.46 CreateAlarmEventPort

CreateAlarmEventPort

+ createAlarmEvent(req: CreateAlarmEventCmd): Promise<string>

Figura 47: Diagramma dell'interfaccia CreateAlarmEventPort

Descrizione: Porta di uscita per la creazione e persistenza di un evento di allarme

Descrizione dei metodi dell'interfaccia:

- `createAlarmEvent(req: CreateAlarmEventCmd): Promise<string>`: Persiste un nuovo evento di allarme e restituisce il suo identificativo univoco

4.1.1.47 CreateAlarmRulePort

CreateAlarmRulePort

+ createAlarmRule(cmd: CreateAlarmRuleCmd): Promise<AlarmRule>

Figura 48: Diagramma dell'interfaccia CreateAlarmRulePort

Descrizione: Porta di uscita per la creazione e persistenza di una regola di allarme

Descrizione dei metodi dell'interfaccia:

- `createAlarmRule(cmd: CreateAlarmRuleCmd): Promise<AlarmRule>`: Persiste una nuova regola di allarme e restituisce il modello creato

4.1.1.48 DeleteAlarmRulePort

DeleteAlarmRulePort

+ deleteAlarmRule(req: DeleteAlarmRuleCmd): Promise<void>

Figura 49: Diagramma dell'interfaccia DeleteAlarmRulePort

Descrizione: Porta di uscita per l'eliminazione di una regola di allarme

Descrizione dei metodi dell'interfaccia:

- `deleteAlarmRule(req: DeleteAlarmRuleCmd): Promise<void>`: Elimina la regola di allarme specificata dal livello di persistenza

4.1.1.49 GetAlarmEventByIdPort

GetAlarmEventByIdPort

+ getAlarmEventById(req: GetAlarmEventByIdCmd): Promise<AlarmEvent | null>

Figura 50: Diagramma dell'interfaccia GetAlarmEventByIdPort

Descrizione: Porta di uscita per il recupero di un singolo evento di allarme

Descrizione dei metodi dell'interfaccia:

- `getAlarmEventById(req: GetAlarmEventByIdCmd): Promise<AlarmEvent | null>`: Recupera un evento di allarme per identificativo dal livello di persistenza

4.1.1.50 GetAlarmRuleByIdPort

GetAlarmRuleByIdPort

+ getAlarmRuleById(req: GetAlarmRuleByIdCmd): Promise<AlarmRule | null>

Figura 51: Diagramma dell'interfaccia GetAlarmRuleByIdPort

Descrizione: Porta di uscita per il recupero di una singola regola di allarme

Descrizione dei metodi dell'interfaccia:

- `getAlarmRuleById(req: GetAlarmRuleByIdCmd): Promise<AlarmRule | null>`: Recupera una regola di allarme per identificativo dal livello di persistenza

4.1.1.51 GetAllAlarmEventsPort

GetAllAlarmEventsPort

+ getAllAlarmEvents(req: GetAllAlarmEventsCmd): Promise<AlarmEvent[]>

Figura 52: Diagramma dell'interfaccia GetAllAlarmEventsPort

Descrizione: Porta di uscita per il recupero paginato di tutti gli eventi di allarme

Descrizione dei metodi dell'interfaccia:

- `getAllAlarmEvents(req: GetAllAlarmEventsCmd): Promise<AlarmEvent[]>`: Recupera in modo paginato tutti gli eventi di allarme dal livello di persistenza

4.1.1.52 GetAllAlarmRulesPort

GetAllAlarmRulesPort

+ getAllAlarmRules(): Promise<AlarmRule[]>

Figura 53: Diagramma dell'interfaccia GetAllAlarmRulesPort

Descrizione: Porta di uscita per il recupero di tutte le regole di allarme

Descrizione dei metodi dell'interfaccia:

- `getAllAlarmRules(): Promise<AlarmRule[]>`: Recupera tutte le regole di allarme dal livello di persistenza

4.1.1.53 GetAllManagedAlarmEventsByUserIdPort

GetAllManagedAlarmEventsByUserIdPort

+ getAllManagedAlarmEventsByUserId(req: GetAllManagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>

Figura 54: Diagramma dell'interfaccia GetAllManagedAlarmEventsByUserIdPort

Descrizione: Porta di uscita per il recupero degli eventi gestiti da un utente

Descrizione dei metodi dell'interfaccia:

- `getAllManagedAlarmEventsByUserId(req: GetAllManagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>`: Recupera in modo paginato gli eventi gestiti da un utente dal livello di persistenza

4.1.1.54 GetAllUnmanagedAlarmEventsByUserIdPort

GetAllUnmanagedAlarmEventsByUserIdPort

+ getAllUnmanagedAlarmEventsByUserId(req: GetAllUnmanagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>

Figura 55: Diagramma dell'interfaccia GetAllUnmanagedAlarmEventsByUserIdPort

Descrizione: Porta di uscita per il recupero degli eventi non gestiti di un utente

Descrizione dei metodi dell'interfaccia:

- `getAllUnmanagedAlarmEventsByUserId(req: GetAllUnmanagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>`: Recupera in modo paginato gli eventi non gestiti di un utente dal livello di persistenza

4.1.1.55 GetWardAlarmEventPort

GetWardAlarmEventPort

+ getWardAlarmEvent(cmd: GetWardAlarmEventCmd): Promise<number>

Figura 56: Diagramma dell'interfaccia GetWardAlarmEventPort

Descrizione: Porta di uscita per il recupero del ward associato a un evento di allarme

Descrizione dei metodi dell'interfaccia:

- `getWardAlarmEvent(cmd: GetWardAlarmEventCmd): Promise<number>`: Recupera l'identificativo del ward associato a un evento di allarme

4.1.1.56 ResolveAlarmEventPort

ResolveAlarmEventPort

+ resolveAlarmEvent(req: ResolveAlarmEventCmd): Promise<void>

Figura 57: Diagramma dell'interfaccia ResolveAlarmEventPort

Descrizione: Porta di uscita per la risoluzione di un evento di allarme

Descrizione dei metodi dell'interfaccia:

- `resolveAlarmEvent(req: ResolveAlarmEventCmd): Promise<void>`: Aggiorna lo stato dell'evento di allarme come risolto nel livello di persistenza

4.1.1.57 UpdateAlarmRulePort

UpdateAlarmRulePort

+ updateAlarmRule(cmd: UpdateAlarmRuleCmd): Promise<AlarmRule>

Figura 58: Diagramma dell'interfaccia UpdateAlarmRulePort

Descrizione: Porta di uscita per l'aggiornamento di una regola di allarme

Descrizione dei metodi dell'interfaccia:

- `updateAlarmRule(cmd: UpdateAlarmRuleCmd): Promise<AlarmRule>`: Aggiorna i dati di una regola di allarme nel livello di persistenza

4.1.1.58 AlarmEventEntity

AlarmEventEntity

+ id: string
+ plant_name: string
+ room_name: string
+ device_name: string
+ device_id: string
+ alarm_rule_id: string
+ alarm_name: string
+ priority: AlarmPriority
+ activation_time: Date
+ resolution_time: Date | null
+ user_id: number | null
+ user_username: string | null

Figura 59: Diagramma della classe AlarmEventEntity

Descrizione: Entità di persistenza che rappresenta un evento di allarme nel database

4.1.1.59 AlarmRuleEntity

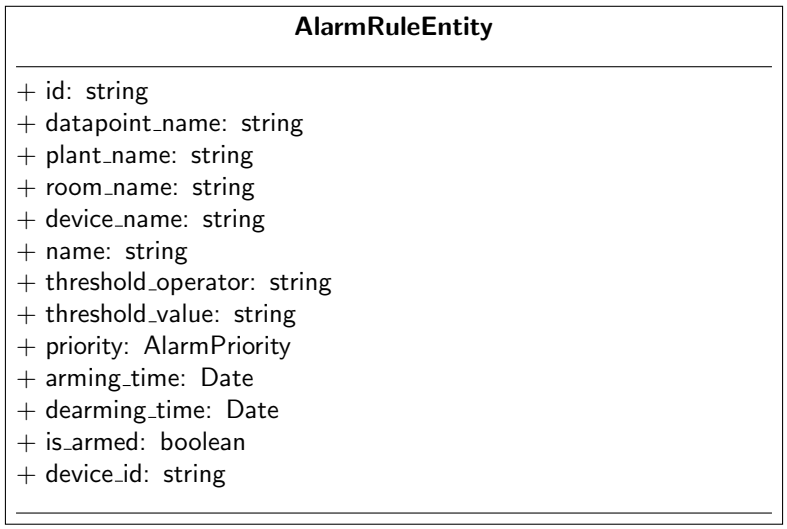

Figura 60: Diagramma della classe AlarmRuleEntity

Descrizione: Entità di persistenza che rappresenta una regola di allarme nel database

4.1.1.60 CheckAlarmEntity

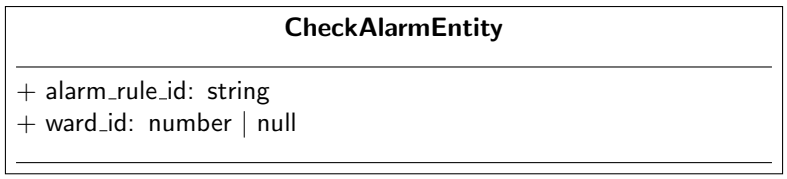

Figura 61: Diagramma della classe CheckAlarmEntity

Descrizione: Entità di persistenza risultante dalla verifica di una regola di allarme

4.1.1.61 AlarmEventsPersistenceAdapter

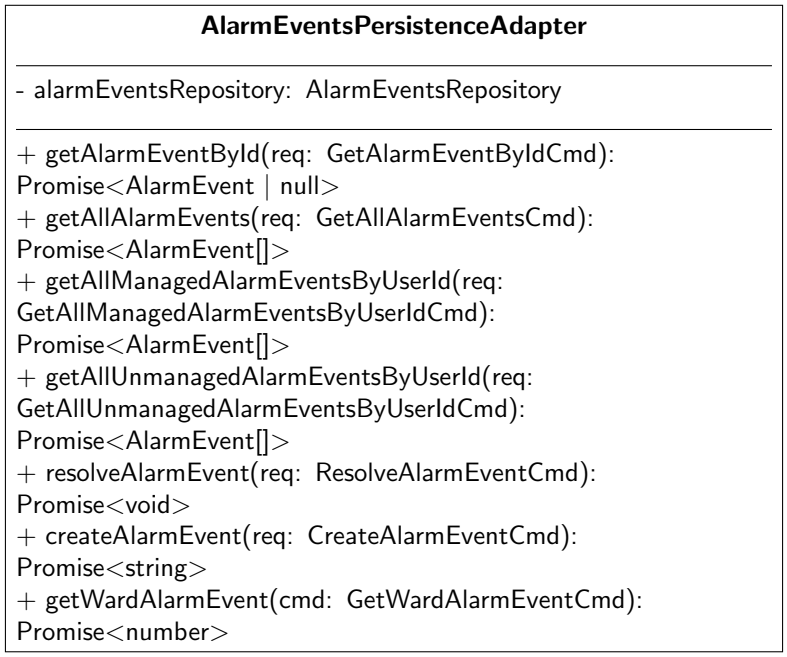

Figura 62: Diagramma della classe AlarmEventsPersistenceAdapter

Descrizione: Adattatore di persistenza che implementa le porte relative agli eventi di allarme

Descrizione dei metodi della classe:

- `getAlarmEventById(req: GetAlarmEventByIdCmd): Promise<AlarmEvent | null>`: Recupera e mappa un evento di allarme per identificativo dal repository
- `getAllAlarmEvents(req: GetAllAlarmEventsCmd): Promise<AlarmEvent[]>`: Recupera e mappa in modo paginato tutti gli eventi di allarme dal repository
- `getAllManagedAlarmEventsByUserId(req: GetAllManagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>`: Recupera e mappa gli eventi gestiti da un utente dal repository
- `getAllUnmanagedAlarmEventsByUserId(req: GetAllUnmanagedAlarmEventsByUserIdCmd): Promise<AlarmEvent[]>`: Recupera e mappa gli eventi non gestiti di un utente dal repository
- `resolveAlarmEvent(req: ResolveAlarmEventCmd): Promise<void>`: Delega al repository la risoluzione di un evento di allarme
- `createAlarmEvent(req: CreateAlarmEventCmd): Promise<string>`: Delega al repository la creazione di un nuovo evento di allarme
- `getWardAlarmEvent(cmd: GetWardAlarmEventCmd): Promise<number>`: Recupera l'identificativo del ward associato a un evento tramite il repository

4.1.1.62 AlarmRulesPersistenceAdapter

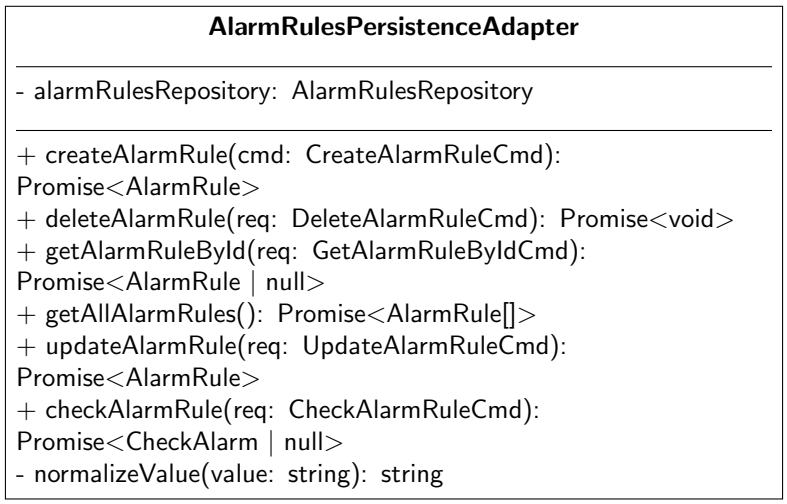

Figura 63: Diagramma della classe AlarmRulesPersistenceAdapter

Descrizione: Adattatore di persistenza che implementa le porte relative alle regole di allarme

Descrizione dei metodi della classe:

- `createAlarmRule(cmd: CreateAlarmRuleCmd): Promise<AlarmRule>`: Delega al repository la creazione di una nuova regola e mappa il risultato nel modello di dominio
- `deleteAlarmRule(req: DeleteAlarmRuleCmd): Promise<void>`: Delega al repository l'eliminazione della regola specificata
- `getAlarmRuleById(req: GetAlarmRuleByIdCmd): Promise<AlarmRule | null>`: Recupera e mappa una regola per identificativo dal repository
- `getAllAlarmRules(): Promise<AlarmRule[]>`: Recupera e mappa tutte le regole di allarme dal repository
- `updateAlarmRule(req: UpdateAlarmRuleCmd): Promise<AlarmRule>`: Delega al repository l'aggiornamento della regola e mappa il risultato
- `checkAlarmRule(req: CheckAlarmRuleCmd): Promise<CheckAlarm | null>`: Delega al repository la verifica della regola e mappa il risultato
- `normalizeValue(value: string): string`: Normalizza il valore del datapoint prima del confronto con la soglia

4.1.1.63 AlarmEventsRepository

AlarmEventsRepository

+ createAlarmEvent(alarmRuleId: string, activationTime: Date): Promise<string>
+ getAlarmEventById(id: string): Promise<AlarmEventEntity | null>
+ getAllAlarmEvents(limit: number, offset: number): Promise<AlarmEventEntity[]>
+ getAllManagedAlarmEventsByUserId(id: number, limit: number, offset: number): Promise<AlarmEventEntity[]>
+ getAllUnmanagedAlarmEventsByUserId(id: number, limit: number, offset: number): Promise<AlarmEventEntity[]>
+ resolveAlarmEvent(alarmId: string, userId: number): Promise<void>
+ getWardAlarmEvent(alarmId: string): Promise<number>

Figura 64: Diagramma dell'interfaccia AlarmEventsRepository

Descrizione: Interfaccia del repository per le operazioni di persistenza degli eventi di allarme

Descrizione dei metodi dell'interfaccia:

- `createAlarmEvent(alarmRuleId: string, activationTime: Date): Promise<string>`: Inserisce un nuovo evento di allarme nel database e restituisce il suo identificativo
- `getAlarmEventById(id: string): Promise<AlarmEventEntity | null>`: Recupera un evento di allarme per identificativo, restituisce null se non trovato
- `getAllAlarmEvents(limit: number, offset: number): Promise<AlarmEventEntity[]>`: Recupera in modo paginato tutti gli eventi di allarme presenti nel database
- `getAllManagedAlarmEventsByUserId(id: number, limit: number, offset: number): Promise<AlarmEventEntity[]>`: Recupera in modo paginato gli eventi di allarme risolti da un utente specifico
- `getAllUnmanagedAlarmEventsByUserId(id: number, limit: number, offset: number): Promise<AlarmEventEntity[]>`: Recupera in modo paginato gli eventi di allarme non ancora risolti assegnati a un utente
- `resolveAlarmEvent(alarmId: string, userId: number): Promise<void>`: Aggiorna nel database l'evento indicato impostando l'utente risolutore e il timestamp di risoluzione
- `getWardAlarmEvent(alarmId: string): Promise<number>`: Recupera l'identificativo del ward associato all'evento di allarme specificato

4.1.1.64 AlarmRulesRepository

AlarmRulesRepository

+ checkAlarmRule(datapointId: string, value: string, activationTime: string): Promise<CheckAlarmEntity | null>
+ createAlarmRule(name: string, priority: AlarmPriority, datapointId: string, deviceId: string, plantId: string, thresholdOperator: string, thresholdValue: string, armingTime: string, dearmingTime: string): Promise<AlarmRuleEntity>
+ deleteAlarmRule(id: string): Promise<void>
+ getAlarmRuleById(id: string): Promise<AlarmRuleEntity | null>
+ getAllAlarmRules(): Promise<AlarmRuleEntity[]>
+ updateAlarmRule(id: string, name: string, priority: AlarmPriority, thresholdOperator: string, thresholdValue: string, armingTime: string, dearmingTime: string, isArmed: boolean): Promise<AlarmRuleEntity>

Figura 65: Diagramma dell'interfaccia AlarmRulesRepository

Descrizione: Interfaccia del repository per le operazioni di persistenza delle regole di allarme

Descrizione dei metodi dell'interfaccia:

- `checkAlarmRule(datapointId: string, value: string, activationTime: string): Promise<CheckAlarmEntity | null>`: Verifica se il valore del datapoint viola la soglia della regola all'orario specificato
- `createAlarmRule(name: string, priority: AlarmPriority, datapointId: string, deviceId: string, plantId: string, thresholdOperator: string, thresholdValue: string, armingTime: string, dearmingTime: string): Promise<AlarmRuleEntity>`: Inserisce una nuova regola di allarme nel database con i parametri forniti
- `deleteAlarmRule(id: string): Promise<void>`: Elimina dal database la regola di allarme con l'identificativo specificato
- `getAlarmRuleById(id: string): Promise<AlarmRuleEntity | null>`: Recupera una regola di allarme per identificativo, restituisce null se non trovata
- `getAllAlarmRules(): Promise<AlarmRuleEntity[]>`: Recupera tutte le regole di allarme presenti nel database
- `updateAlarmRule(id: string, name: string, priority: AlarmPriority, thresholdOperator: string, thresholdValue: string, armingTime: string, dearmingTime: string, isArmed: boolean): Promise<AlarmRuleEntity>`: Aggiorna i parametri della regola di allarme specificata nel database

4.1.1.65 AlarmEventsRepositoryImpl

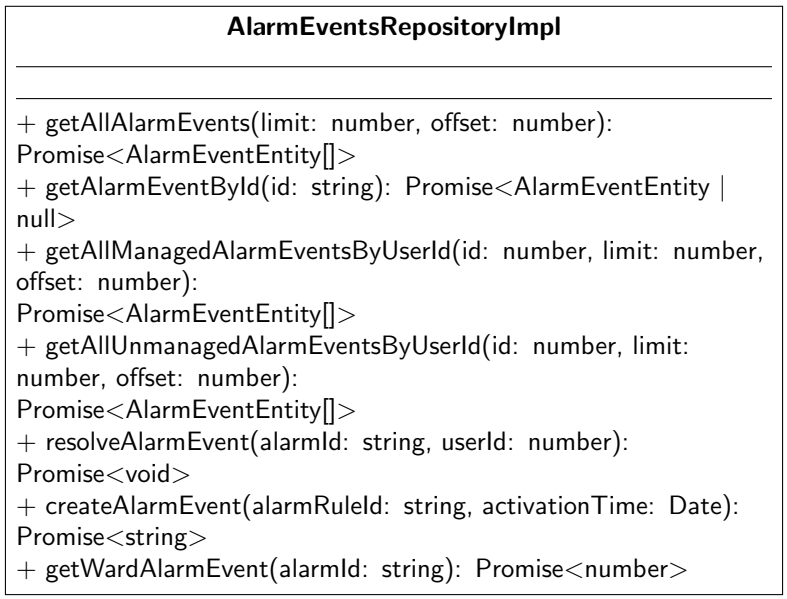

Figura 66: Diagramma della classe AlarmEventsRepositoryImpl

Descrizione: Implementazione concreta del repository per gli eventi di allarme

Descrizione dei metodi della classe:

- `getAllAlarmEvents(limit: number, offset: number): Promise<AlarmEventEntity[]>`: Implementa il recupero paginato di tutti gli eventi di allarme tramite query al database
- `getAlarmEventById(id: string): Promise<AlarmEventEntity | null>`: Implementa il recupero di un evento per identificativo tramite query al database
- `getAllManagedAlarmEventsByUserId(id: number, limit: number, offset: number): Promise<AlarmEventEntity[]>`: Implementa il recupero paginato degli eventi gestiti da un utente tramite query al database
- `getAllUnmanagedAlarmEventsByUserId(id: number, limit: number, offset: number): Promise<AlarmEventEntity[]>`: Implementa il recupero paginato degli eventi non gestiti di un utente tramite query al database
- `resolveAlarmEvent(alarmId: string, userId: number): Promise<void>`: Implementa la risoluzione di un evento aggiornando i relativi campi nel database
- `createAlarmEvent(alarmRuleId: string, activationTime: Date): Promise<string>`: Implementa l'inserimento di un nuovo evento di allarme nel database
- `getWardAlarmEvent(alarmId: string): Promise<number>`: Implementa il recupero del ward associato a un evento tramite query al database

4.1.1.66 AlarmRulesRepositoryImpl

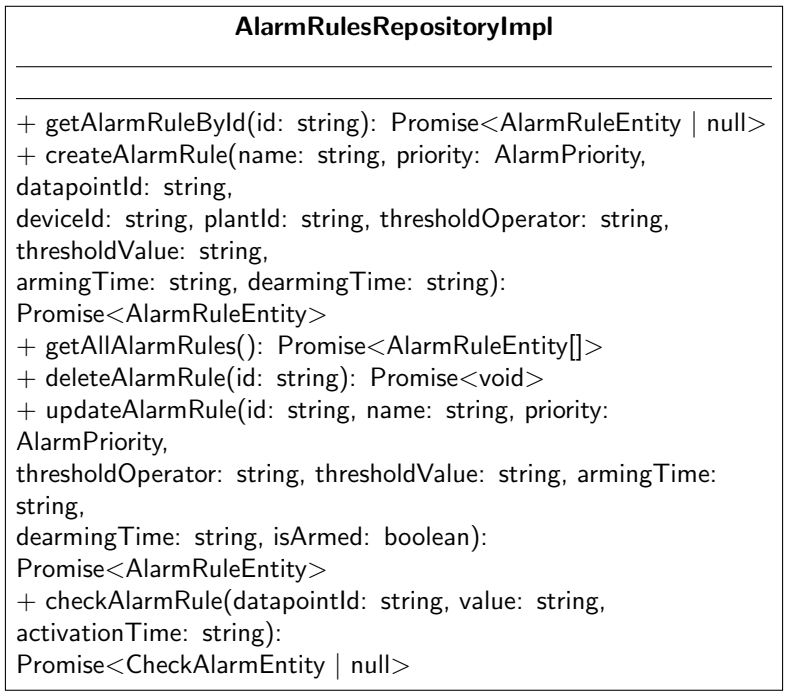

Figura 67: Diagramma della classe AlarmRulesRepositoryImpl

Descrizione: Implementazione concreta del repository per le regole di allarme

Descrizione dei metodi della classe:

- `getAlarmRuleById(id: string): Promise<AlarmRuleEntity | null>`: Implementa il recupero di una regola per identificativo tramite query al database
- `createAlarmRule(name: string, priority: AlarmPriority, datapointId: string, deviceId: string, plantId: string, thresholdOperator: string, thresholdValue: string, armingTime: string, dearmingTime: string): Promise<AlarmRuleEntity>`: Implementa l'inserimento di una nuova regola di allarme nel database
- `getAllAlarmRules(): Promise<AlarmRuleEntity[]>`: Implementa il recupero di tutte le regole di allarme tramite query al database
- `deleteAlarmRule(id: string): Promise<void>`: Implementa l'eliminazione di una regola tramite query al database
- `updateAlarmRule(id: string, name: string, priority: AlarmPriority, thresholdOperator: string, thresholdValue: string, armingTime: string, dearmingTime: string, isArmed: boolean): Promise<AlarmRuleEntity>`: Implementa l'aggiornamento dei parametri di una regola nel database
- `checkAlarmRule(datapointId: string, value: string, activationTime: string): Promise<CheckAlarmEntity | null>`: Implementa la verifica di una regola rispetto al valore del datapoint tramite query al database

4.1.2 Analytics

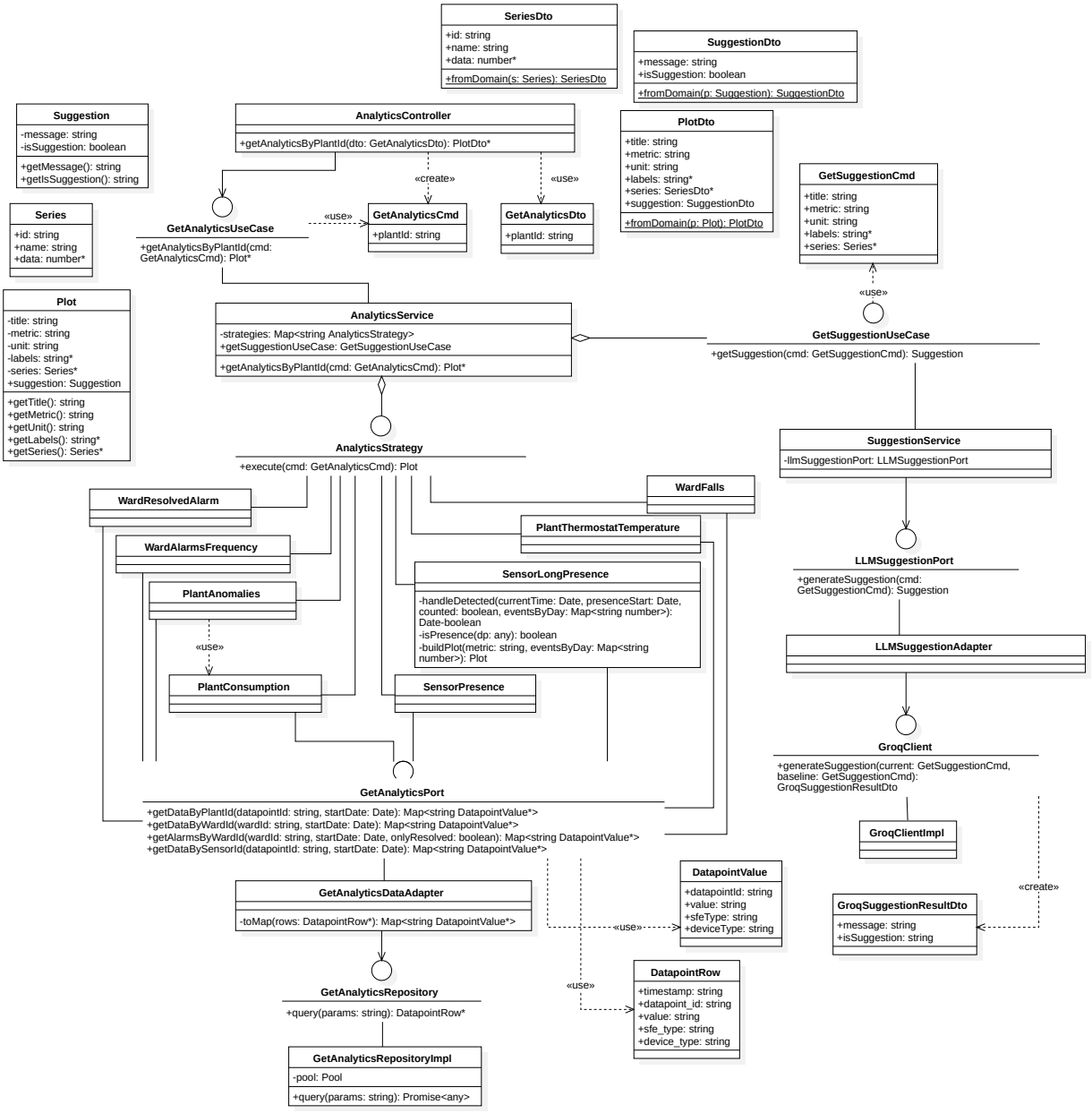

Figura 68: Diagramma delle classi del modulo Analytics

4.1.2.1 GetAnalyticsDto

Figura 69: Diagramma della classe GetAnalyticsDto

Descrizione: DTO di input per l'endpoint REST. Trasporta l'identificativo dell'impianto ricevuto nella richiesta HTTP verso il controller, che lo converte in un `GetAnalyticsCmd`.

4.1.2.2 PlotDto

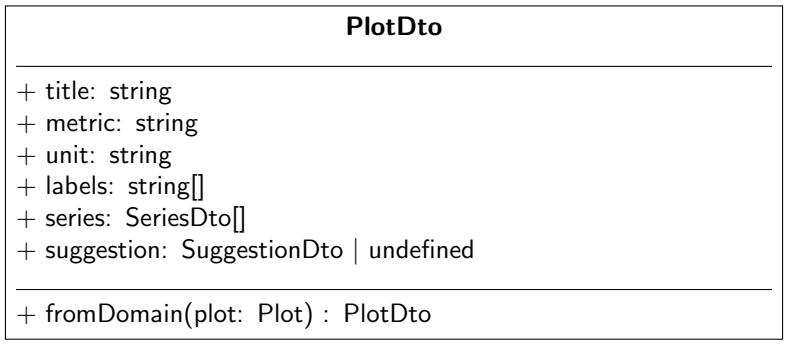

Figura 70: Diagramma della classe PlotDto

Descrizione: DTO di output che serializza un oggetto `Plot` per il trasporto via API REST. Espone il metodo factory `fromDomain` per la conversione dall'entità di dominio, includendo opzionalmente il `SuggestionDto`.

Descrizione dei metodi della classe:

- `fromDomain(plot: Plot) : PlotDto`: Costruisce il DTO a partire dall'oggetto di dominio Plot

4.1.2.3 SeriesDto

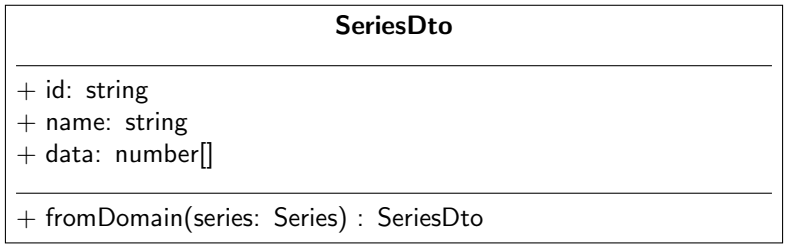

Figura 71: Diagramma della classe SeriesDto

Descrizione: DTO di output che serializza un oggetto `Series`. Espone il metodo factory `fromDomain` e riproduce i campi `id`, `name` e `data` in forma piatta adatta alla serializzazione JSON.

Descrizione dei metodi della classe:

- `fromDomain(series: Series) : SeriesDto`: Costruisce il DTO a partire dall'oggetto di dominio Series

4.1.2.4 SuggestionDto

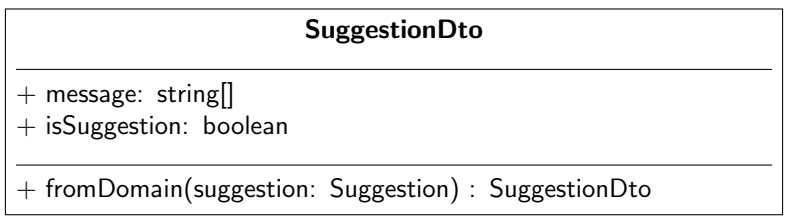

Figura 72: Diagramma della classe SuggestionDto

Descrizione: DTO di output che serializza un oggetto `Suggestion`. Espone il metodo factory `fromDomain` e trasporta i messaggi e il flag `isSuggestion` verso il client.

Descrizione dei metodi della classe:

- `fromDomain(suggestion: Suggestion) : SuggestionDto`: Costruisce il DTO a partire dall'oggetto di dominio Suggestion

4.1.2.5 AnalyticsController

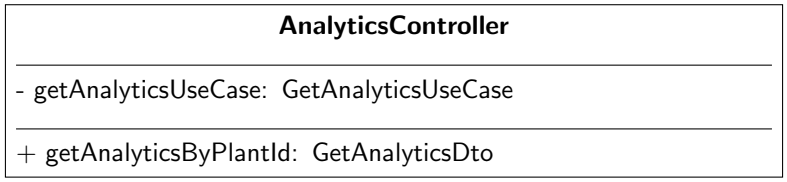

Figura 73: Diagramma della classe AnalyticsController

Descrizione: Controller REST che espone l'endpoint per il recupero delle analytics di un impianto. Riceve la richiesta HTTP, la delega al `GetAnalyticsUseCase` e restituisce al client la lista di plot e relativi suggerimenti serializzati come DTO.

Descrizione dei metodi della classe:

- `getAnalyticsByPlantId: GetAnalyticsDto`: Riceve l'identificativo dell'impianto dalla richiesta HTTP, costruisce un `GetAnalyticsCmd` e delega al `GetAnalyticsUseCase`, restituendo la lista di *PlotDto* serializzati

4.1.2.6 GetAnalyticsCmd

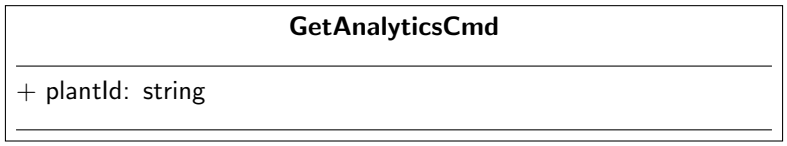

Figura 74: Diagramma della classe GetAnalyticsCmd

Descrizione: Comando applicativo che trasporta l'identificativo dell'impianto necessario per avviare il recupero delle analytics.

4.1.2.7 GetSuggestionCmd

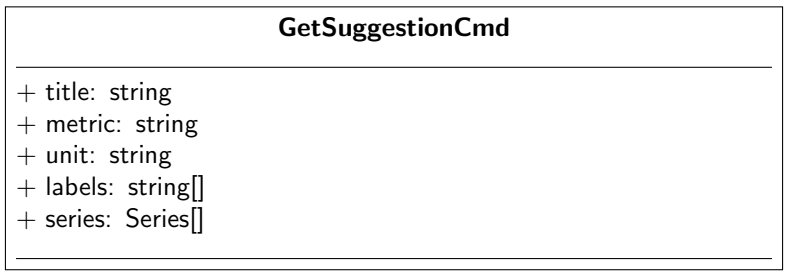

Figura 75: Diagramma della classe GetSuggestionCmd

Descrizione: Comando applicativo che racchiude tutti i dati necessari alla generazione di un suggerimento.

4.1.2.8 GetAnalyticsUseCase

GetAnalyticsUseCase

+ getAnalyticsByPlantId(cmd: GetAnalyticsCmd) : Promise<Plot[]>

Figura 76: Diagramma dell'interfaccia GetAnalyticsUseCase

Descrizione: Porta di ingresso (driving port) che definisce il contratto per il recupero delle analytics di un impianto. Espone il metodo `getAnalyticsByPlantId`, implementato da `AnalyticsService`.

Descrizione dei metodi dell'interfaccia:

- `getAnalyticsByPlantId(cmd: GetAnalyticsCmd) : Promise<Plot[]>`: Restituisce tutte le analytics disponibili per l'impianto identificato dal comando

4.1.2.9 GetSuggestionUseCase

GetSuggestionUseCase

+ getSuggestion(cmd: GetSuggestionCmd) : Promise<Suggestion>

Figura 77: Diagramma dell'interfaccia GetSuggestionUseCase

Descrizione: Porta di ingresso che definisce il contratto per la generazione di suggerimenti energetici tramite LLM. Espone il metodo `getSuggestion`, implementato da `SuggestionService`.

Descrizione dei metodi dell'interfaccia:

- `getSuggestion(cmd: GetSuggestionCmd) : Promise<Suggestion>`: Richiede la generazione di un suggerimento per il risparmio energetico confrontando l'analytics corrente con la baseline

4.1.2.10 AnalyticsService

AnalyticsService

- strategies: Map<string, AnalyticsStrategy>
- getSuggestionUseCase: GetSuggestionUseCase

+ getAnalyticsByPlantId(cmd: GetAnalyticsCmd) : Promise<Plot[]>

Figura 78: Diagramma della classe AnalyticsService

Descrizione: Servizio applicativo che implementa `GetAnalyticsUseCase`. Itera su tutte le strategie di analytics registrate nella mappa `strategies`, le esegue per l'impianto richiesto, genera i suggerimenti e aggrega i plot risultanti in una lista da restituire al controller.

Descrizione dei metodi della classe:

- `getAnalyticsByPlantId(cmd: GetAnalyticsCmd) : Promise<Plot[]>`: Esegue tutte le strategie di analytics registrate per l'impianto e aggrega i risultati in una lista di plot

4.1.2.11 SuggestionService

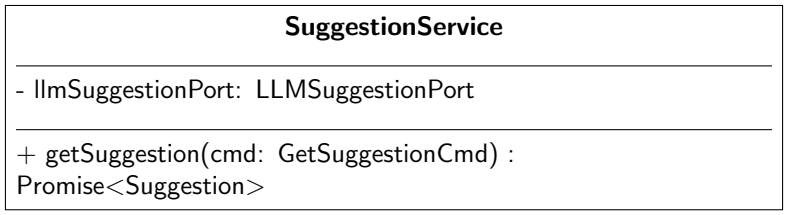

Figura 79: Diagramma della classe SuggestionService

Descrizione: Servizio applicativo che implementa `GetSuggestionUseCase`. Riceve il comando con i dati dell'analytics corrente e delega interamente al modello LLM — tramite `LLMSuggestionPort` — il confronto con la baseline e la generazione del suggerimento.

Descrizione dei metodi della classe:

- `getSuggestion(cmd: GetSuggestionCmd) : Promise<Suggestion>`: Delega al modello LLM il confronto tra analytics corrente e baseline e restituisce i suggerimenti generati, se presenti

4.1.2.12 GetAnalyticsPort

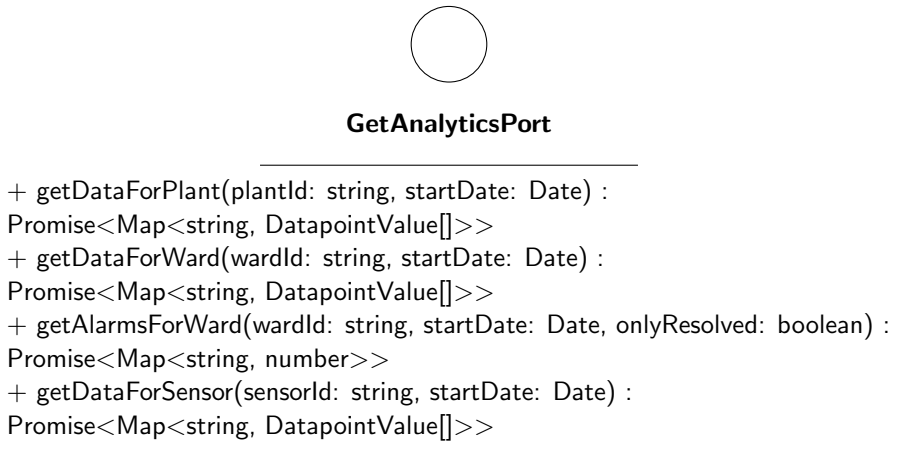

Figura 80: Diagramma dell'interfaccia GetAnalyticsPort

Descrizione: Porta di uscita (driven port) che definisce il contratto per l'interrogazione dei datapoint sul database. Espone metodi specializzati per il recupero dei dati per impianto, reparto, allarmi di reparto e singolo sensore. Viene implementata da `GetAnalyticsData`.

Descrizione dei metodi dell'interfaccia:

- `getDataForPlant(plantId: string, startDate: Date) : Promise<Map<string, DatapointValue[]>>`: Recupera dal database i datapoint dell'impianto identificato da `plantId` a partire da `startDate`, restituendoli come mappa indicizzata per timestamp
- `getDataForWard(wardId: string, startDate: Date) : Promise<Map<string, DatapointValue[]>>`: Recupera dal database i datapoint di tutti gli impianti del reparto identificato da `wardId` a partire da `startDate`, restituendoli come mappa indicizzata per timestamp
- `getAlarmsForWard(wardId: string, startDate: Date, onlyResolved: boolean) : Promise<Map<string, number>>`: Recupera dal database gli allarmi del reparto identificato da `wardId` a partire da `startDate`; se `onlyResolved` è true, filtra i soli allarmi risolti
- `getDataForSensor(sensorId: string, startDate: Date) : Promise<Map<string, DatapointValue[]>>`: Recupera dal database i datapoint del sensore identificato da `sensorId` a partire da `startDate`, restituendoli come mappa indicizzata per timestamp

4.1.2.13 LLMSuggestionPort

LLMSuggestionPort

+ generateSuggestion(cmd: GetSuggestionCmd) : Promise<Suggestion>

Figura 81: Diagramma dell'interfaccia LLMSuggestionPort

Descrizione: Porta di uscita che definisce il contratto per la comunicazione con il modello LLM. Espone `generateSuggestion`, implementata da `LLMSuggestionAdapter`.

Descrizione dei metodi dell'interfaccia:

- `generateSuggestion(cmd: GetSuggestionCmd) : Promise<Suggestion>`: Invia i dati dell'analytics corrente e della baseline al modello LLM e restituisce eventuali suggerimenti per il risparmio energetico

4.1.2.14 Plot

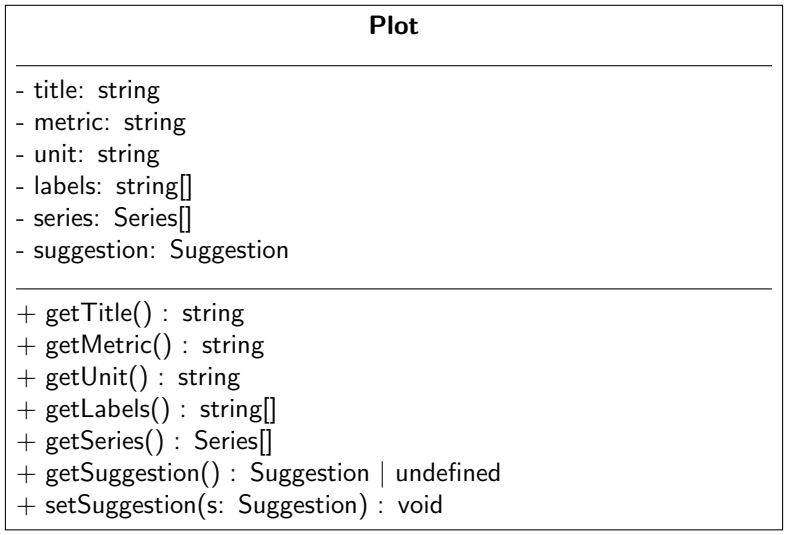

Figura 82: Diagramma della classe Plot

Descrizione: Entità di dominio che rappresenta un grafico analytics completo. Aggrega titolo, metrica, unità, etichette temporali, serie di dati e, opzionalmente, un suggerimento LLM associato. Espone getter per tutti i campi e un setter per il suggerimento.

Descrizione dei metodi della classe:

- `getTitle() : string`: Restituisce il titolo del plot
- `getMetric() : string`: Restituisce il nome della metrica rappresentata
- `getUnit() : string`: Restituisce l'unità di misura della metrica
- `getLabels() : string[]`: Restituisce le etichette dell'asse temporale
- `getSeries() : Series[]`: Restituisce le serie di dati del plot
- `getSuggestion() : Suggestion | undefined`: Restituisce il suggerimento energetico associato al plot, se presente
- `setSuggestion(s: Suggestion) : void`: Associa un suggerimento energetico al plot

4.1.2.15 Series

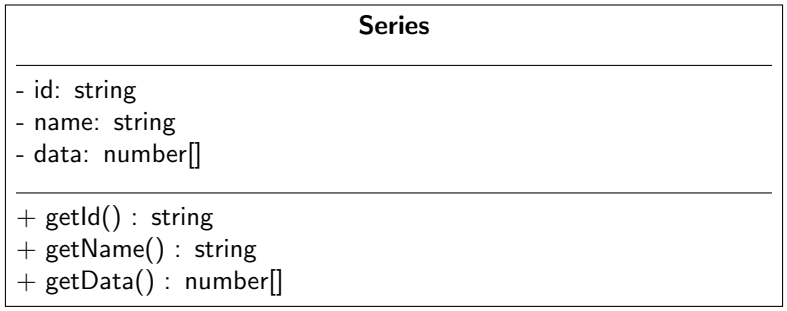

Figura 83: Diagramma della classe Series

Descrizione: Value object di dominio che rappresenta una singola serie di dati all'interno di un `Plot`. Contiene un identificativo, un nome descrittivo e l'array di valori numerici allineati alle etichette temporali del plot padre.

Descrizione dei metodi della classe:

- `getId() : string`: Restituisce l'identificativo della serie
- `getName() : string`: Restituisce il nome della serie
- `getData() : number[]`: Restituisce i valori numerici della serie

4.1.2.16 Suggestion

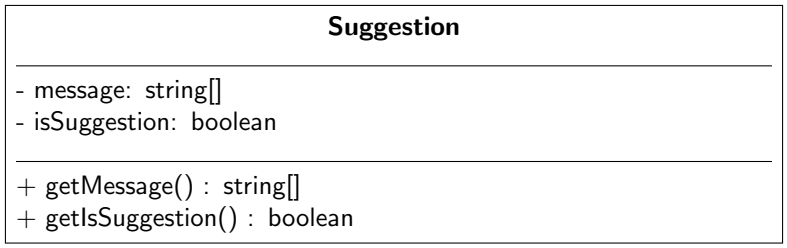

Figura 84: Diagramma della classe Suggestion

Descrizione: Value object di dominio che incapsula il risultato della generazione LLM: una lista di messaggi testuali e un flag booleano che distingue tra una risposta generica e un suggerimento effettivo basato su un'anomalia rilevata.

Descrizione dei metodi della classe:

- `getMessage() : string[]`: Restituisce i messaggi del suggerimento
- `getIsSuggestion() : boolean`: Restituisce true se è presente un suggerimento effettivo generato dal modello

4.1.2.17 GetAnalyticsData

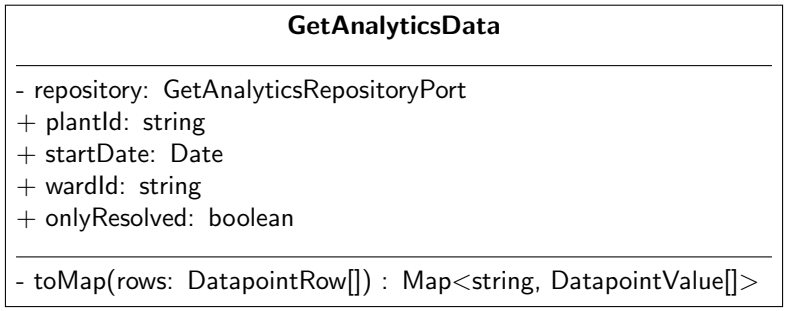

Figura 85: Diagramma della classe GetAnalyticsData

Descrizione: Adapter che implementa la porta `GetAnalyticsPort`. Si occupa di richiamare `GetAnalyticsRepositoryImpl`, convertendo le righe grezze restituite in mappe di `DatapointValue` indicizzate per timestamp tramite il metodo `toMap`.

Descrizione dei metodi della classe:

- `toMap(rows: DatapointRow[]) : Map<string, DatapointValue[]>`: Converte una lista di righe grezze di datapoint in una mappa indicizzata per timestamp ISO

4.1.2.18 LLMSuggestionAdapter

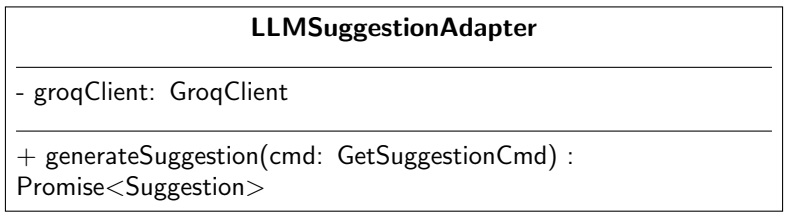

Figura 86: Diagramma della classe LLMSuggestionAdapter

Descrizione: Adapter che implementa la porta `LLMSuggestionPort`. Fa da ponte tra il dominio applicativo e le API Groq, inoltrando i dati dell'analytics corrente al modello LLaMA 3.3 e restituendo il suggerimento generato come oggetto `Suggestion`.

Descrizione dei metodi della classe:

- `generateSuggestion(cmd: GetSuggestionCmd) : Promise<Suggestion>`: Inoltra la richiesta di generazione suggerimento al client Groq e restituisce il risultato

4.1.2.19 GetAnalyticsRepositoryPort

Figura 87: Diagramma dell'interfaccia GetAnalyticsRepositoryPort

Descrizione: Porta di uscita di basso livello che astrae l'accesso diretto al database. Definisce il metodo `query` per l'esecuzione di SQL grezzo, implementato da `GetAnalyticsRepositoryImpl`.

Descrizione dei metodi dell'interfaccia:

- `query(sql: string) : Promise<DatapointRow[] | any[]>`: Esegue una query SQL sul database e restituisce le righe risultanti

4.1.2.20 GetAnalyticsRepositoryImpl

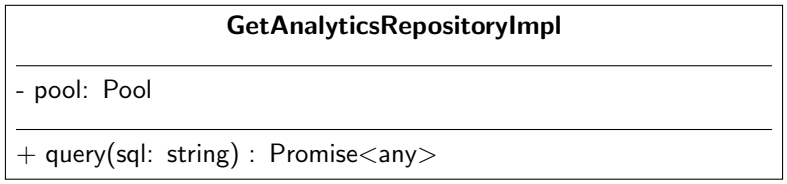

Figura 88: Diagramma della classe GetAnalyticsRepositoryImpl

Descrizione: Implementazione concreta di `GetAnalyticsRepositoryPort`. Gestisce un pool di connessioni e delega l'esecuzione delle query SQL al pool, restituendo i risultati grezzi allo strato applicativo.

Descrizione dei metodi della classe:

- `query(sql: string) : Promise<any>`: Esegue una query SQL sul database tramite il pool di connessioni e restituisce i risultati grezzi

4.1.2.21 AnalyticsStrategy

AnalyticsStrategy

+ execute(cmd: GetAnalyticsCmd) : Promise<Plot>

Figura 89: Diagramma dell'interfaccia AnalyticsStrategy

Descrizione: Interfaccia che definisce il contratto del pattern Strategy per le analytics. Ogni strategia concreta implementa `execute`, che riceve un `GetAnalyticsCmd` e restituisce un `Plot`.

Descrizione dei metodi dell'interfaccia:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Esegue la strategia di analytics e restituisce il plot risultante

4.1.2.22 PlantAnomalies

PlantAnomalies

+ execute(cmd: GetAnalyticsCmd) : Promise<Plot>

Figura 90: Diagramma della classe PlantAnomalies

Descrizione: Strategia che analizza i consumi energetici dell'impianto e conta i valori anomali, ovvero quelli che superano una soglia elevata rispetto ai valori restituiti da `PlantConsumption`. Restituisce un plot con il numero di anomalie aggregate nel periodo.

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Conta le anomalie di consumo energetico dell'impianto: un valore viene considerato anomalo se supera una soglia elevata rispetto ai valori di PlantConsumption

4.1.2.23 PlantConsumption

PlantConsumption

- analyticsPort: GetAnalyticsPort

+ execute(cmd: GetAnalyticsCmd) : Promise<Plot>

Figura 91: Diagramma della classe PlantConsumption

Descrizione: Strategia che recupera i dati di consumo energetico dell'impianto tramite `GetAnalyticsPort` e restituisce il plot dei consumi nel periodo richiesto. Funge anche da riferimento per la rilevazione delle anomalie in `PlantAnomalies`.

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Calcola e restituisce il plot del consumo energetico dell'impianto nel periodo richiesto

4.1.2.24 PlantThermostatTemperature

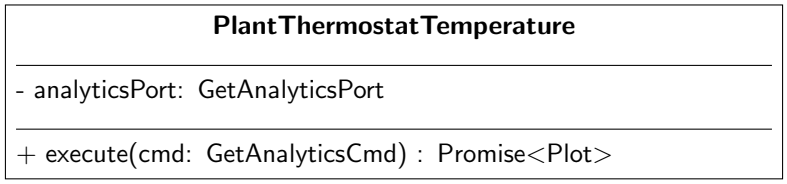

Figura 92: Diagramma della classe PlantThermostatTemperature

Descrizione: Strategia che recupera le temperature rilevate dai termostati dell'impianto tramite `GetAnalyticsPort` e le aggrega in un plot temporale, utile per analizzare l'andamento termico degli appartamenti.

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Calcola e restituisce il plot delle temperature rilevate dai termostati dell'impianto nel periodo richiesto

4.1.2.25 SensorLongPresence

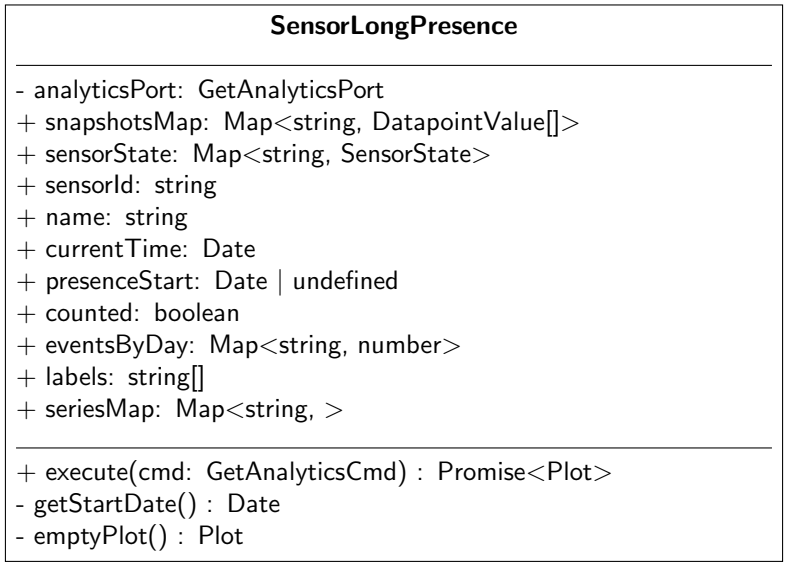

Figura 93: Diagramma della classe SensorLongPresence

Descrizione: Strategia che rileva le presenze prolungate (oltre 30 minuti).

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Rileva le presenze prolungate (oltre 30 minuti) contando i cambi di stato dei sensori e restituisce il plot aggregato per giorno
- `getStartDate() : Date`: Calcola la data di inizio del periodo di analisi a partire dalla configurazione del comando
- `emptyPlot() : Plot`: Restituisce un plot vuoto da usare come fallback in assenza di dati

4.1.2.26 SensorPresence

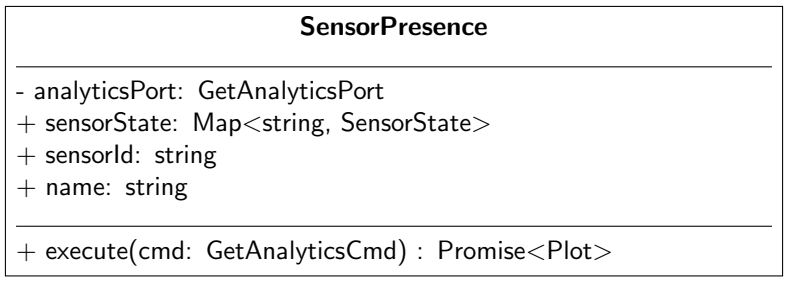

Figura 94: Diagramma della classe SensorPresence

Descrizione: Strategia che rileva le presenze leggendo direttamente i valori forniti dai datapoint dei sensori.

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Rileva le presenze leggendo direttamente i valori forniti dai datapoint dei sensori e restituisce il plot aggregato per giorno

4.1.2.27 WardAlarmsFrequency

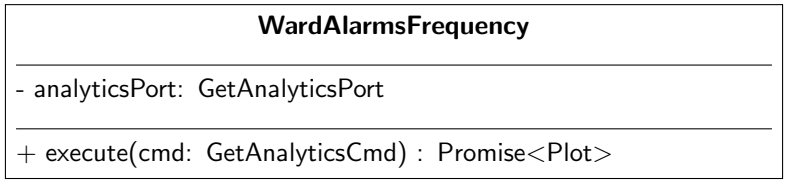

Figura 95: Diagramma della classe WardAlarmsFrequency

Descrizione: Strategia che calcola la frequenza giornaliera degli allarmi inviati nel reparto, indipendentemente dal loro stato di risoluzione. Recupera i dati tramite `GetAnalyticsPort` e li aggrega in un plot di frequenza per giorno.

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Calcola la frequenza degli allarmi inviati nel reparto (indipendentemente dalla risoluzione) aggregata per giorno

4.1.2.28 WardFalls

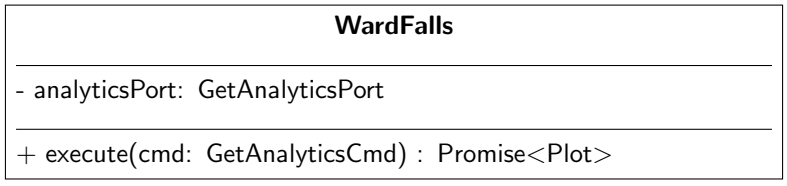

Figura 96: Diagramma della classe WardFalls

Descrizione: Strategia che conta gli eventi di caduta rilevati dai sensori negli appartamenti del reparto. Recupera gli eventi tramite `GetAnalyticsPort` e restituisce un plot con il numero di cadute aggregate per giorno.

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Conta e restituisce il plot degli eventi di caduta rilevati dai sensori negli appartamenti del reparto, aggregati per giorno

4.1.2.29 WardResolvedAlarm

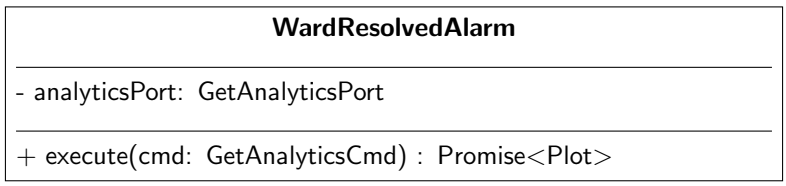

Figura 97: Diagramma della classe WardResolvedAlarm

Descrizione: Strategia che traccia, per ciascun giorno del periodo analizzato, sia il totale degli allarmi del reparto sia il sottoinsieme di quelli risolti, consentendo di valutare il tasso di risoluzione nel tempo.

Descrizione dei metodi della classe:

- `execute(cmd: GetAnalyticsCmd) : Promise<Plot>`: Traccia per il reparto il numero di allarmi totali e il numero di allarmi risolti, aggregati per giorno

4.1.2.30 GroqClient

GroqClient

+ generateSuggestion(current: GetSuggestionCmd, baseline: GetSuggestionCmd) : Promise<GroqSuggestionResultDto>

Figura 98: Diagramma dell'interfaccia GroqClient

Descrizione: Interfaccia che definisce il contratto del client Groq. Dichiara i campi `current` e `baseline` come dati di input per la generazione del suggerimento, implementata da `GroqClientImpl`.

Descrizione dei metodi dell'interfaccia:

- `generateSuggestion(current: GetSuggestionCmd, baseline: GetSuggestionCmd) : Promise<GroqSuggestionResultDto>`: Effettua la chiamata al LLM per generare i suggerimenti a partire dai dati correnti e di baseline

4.1.2.31 GroqClientImpl

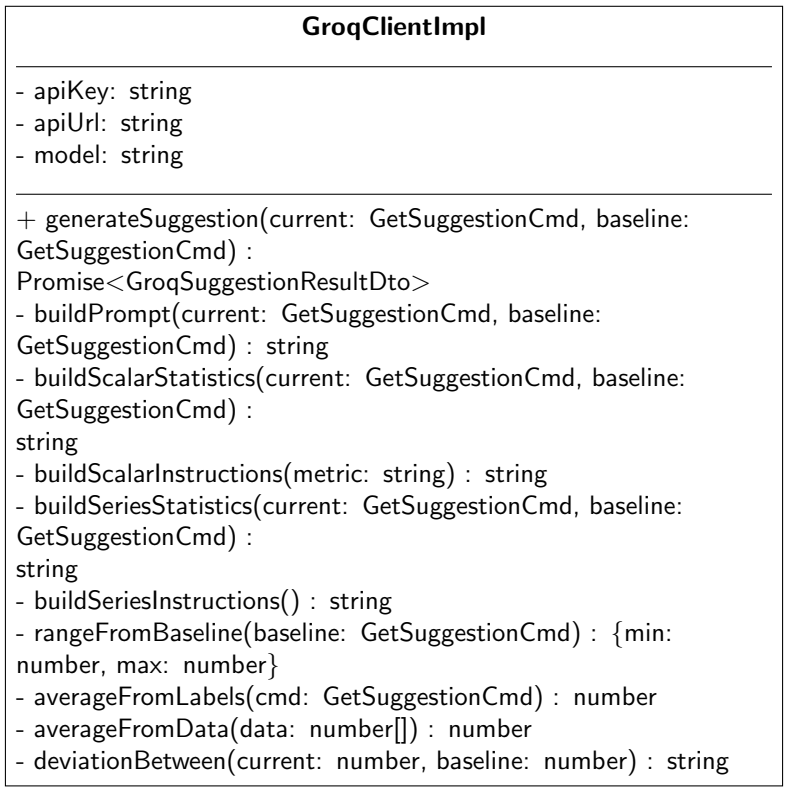

Figura 99: Diagramma della classe GroqClientImpl

Descrizione: Implementazione concreta di `GroqClient`. Costruisce prompt contestualizzati (stagione, metrica, statistiche) e li invia al modello LLaMA 3.3 tramite le API Groq. Gestisce sia metriche scalari che multi-sensore, validando la risposta JSON prima di restituirla; in caso di errore o risposta non conforme restituisce un suggerimento vuoto.

Descrizione dei metodi della classe:

- `generateSuggestion(current: GetSuggestionCmd, baseline: GetSuggestionCmd) : Promise<GroqSuggestionResultDto>`: Invia il prompt costruito a partire dai dati correnti e di baseline all'API Groq e restituisce il suggerimento parsato; in caso di metrica non supportata, serie vuote, errore HTTP o risposta malformata restituisce un suggerimento vuoto
- `buildPrompt(current: GetSuggestionCmd, baseline: GetSuggestionCmd) : string`: Costruisce il prompt completo da inviare al modello, includendo contesto stagionale, descrizione della metrica e la sezione statistica appropriata (scalare o per serie)
- `buildScalarStatistics(current: GetSuggestionCmd, baseline: GetSuggestionCmd) : string`: Produce la sezione statistica del prompt per metriche scalari: per la temperatura include il range accettabile dalla baseline, per le altre metriche calcola media corrente, media baseline e deviazione percentuale
- `buildScalarInstructions(metric: string) : string`: Costruisce le istruzioni comportamentali del prompt per metriche scalari, differenziando le soglie e le azioni suggerite in base alla metrica specifica
- `buildSeriesStatistics(current: GetSuggestionCmd, baseline: GetSuggestionCmd) : string`: Produce la sezione statistica del prompt per metriche multi-sensore, calcolando per ciascuna serie la media corrente, la media baseline e la deviazione percentuale

- `buildSeriesInstructions() : string`: Costruisce le istruzioni comportamentali del prompt per metriche multi-sensore, richiedendo al modello di analizzare ogni sensore individualmente e segnalare deviazioni oltre il 30%
- `rangeFromBaseline(baseline: GetSuggestionCmd) : {min: number, max: number}`: Estrae il valore minimo e massimo dalla prima serie della baseline, usati come range accettabile per la metrica temperatura
- `averageFromLabels(cmd: GetSuggestionCmd) : number`: Calcola la media aritmetica dei valori numerici validi contenuti nella prima serie del comando
- `averageFromData(data: number[]) : number`: Calcola la media aritmetica di un array di valori numerici
- `deviationBetween(current: number, baseline: number) : string`: Calcola e formatta la deviazione percentuale del valore corrente rispetto alla baseline; restituisce `"0.0"` se la baseline è zero

4.1.3 MyVimar Integration

4.1.3.1 PlantAuthDto

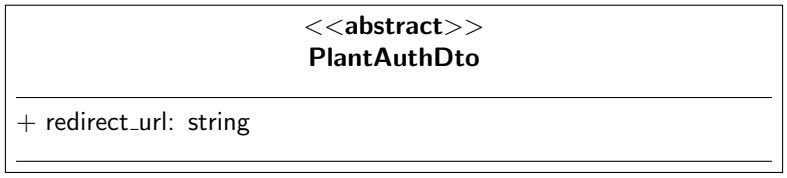

Figura 100: Diagramma della classe astratta PlantAuthDto

Descrizione: DTO di input che trasporta le credenziali di autenticazione per un impianto MyVimar

4.1.3.2 PrepareOAuthTicketDto

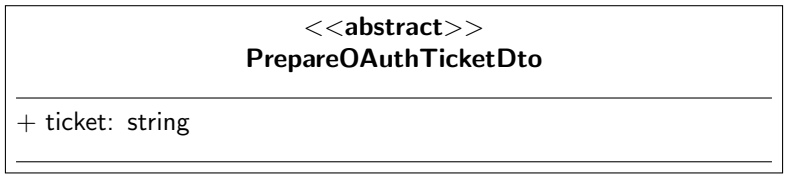

Figura 101: Diagramma della classe astratta PrepareOAuthTicketDto

Descrizione: DTO di input per la preparazione del ticket OAuth, contenente i dati necessari all'avvio del flusso di autorizzazione

4.1.3.3 MyVimarAccountStatusDto

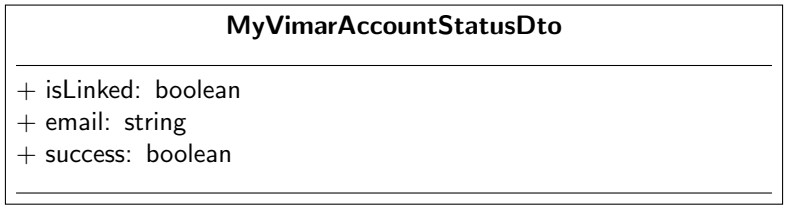

Figura 102: Diagramma della classe MyVimarAccountStatusDto

Descrizione: DTO di output che trasporta lo stato del collegamento dell'account MyVimar dell'utente

4.1.3.4 TokensDto

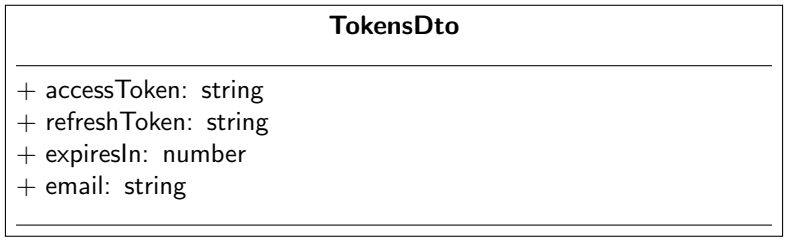

Figura 103: Diagramma della classe TokensDto

Descrizione: DTO di output che trasporta la coppia di token restituita dall'API MyVimar dopo il completamento del flusso OAuth

4.1.3.5 TokenPair

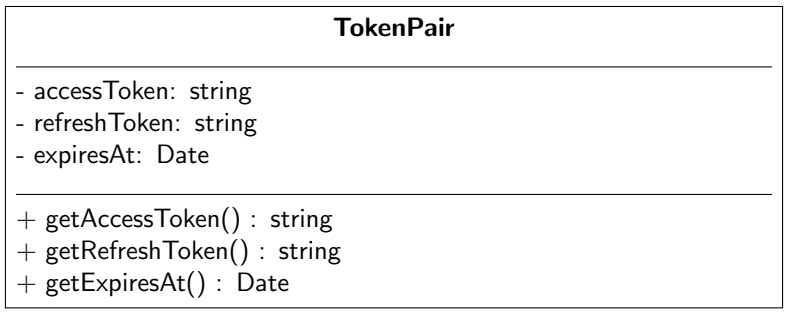

Figura 104: Diagramma della classe TokenPair

Descrizione: Oggetto di dominio che rappresenta una coppia di token OAuth (accesso e refresh) con la relativa scadenza

Descrizione dei metodi della classe:

- `getAccessToken() : string`: Restituisce il token di accesso
- `getRefreshToken() : string`: Restituisce il token di refresh
- `getExpiresAt() : Date`: Restituisce il timestamp di scadenza del token

4.1.3.6 ApiAuthTicketController

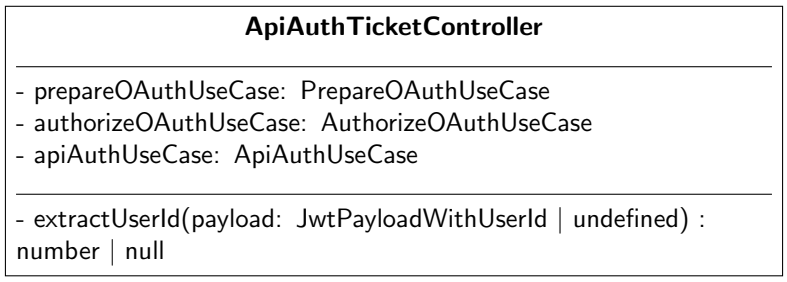

Figura 105: Diagramma della classe ApiAuthTicketController

Descrizione: Controller REST che gestisce il flusso OAuth lato ticket: generazione dell'URL di login, preparazione e autorizzazione del ticket

Descrizione dei metodi della classe:

- `extractUserId(payload: JwtPayloadWithUserId | undefined) : number | null`: Estrae l'ID utente dal payload JWT, restituendo null se assente

4.1.3.7 ApiAuthVimarController

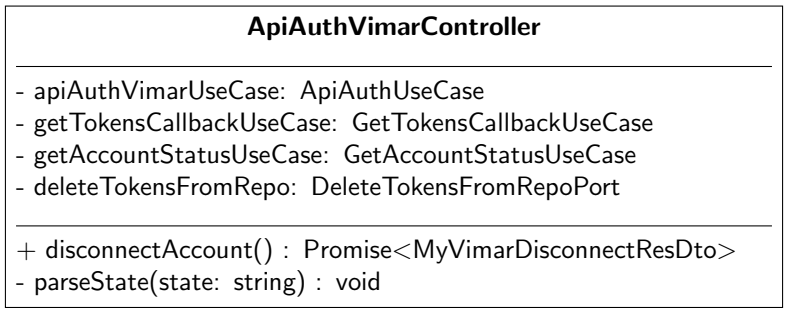

Figura 106: Diagramma della classe ApiAuthVimarController

Descrizione: Controller REST che gestisce il flusso OAuth lato MyVimar: redirect, callback, stato account e disconnessione

Descrizione dei metodi della classe:

- `disconnectAccount() : Promise<MyVimarDisconnectResDto>`: Disconnette l'account MyVimar dell'utente corrente eliminando i token salvati
- `parseState(state: string) : void`: Deserializza il parametro `state` ricevuto nel callback OAuth

4.1.3.8 ApiAuthUseCase

Figura 107: Diagramma dell'interfaccia ApiAuthUseCase

Descrizione: Porta di ingresso per la generazione dell'URL di login del flusso OAuth MyVimar

Descrizione dei metodi dell'interfaccia:

- `getLoginUrl(state?: string) : string`: Genera e restituisce l'URL di autorizzazione MyVimar, includendo opzionalmente il parametro `state`

4.1.3.9 AuthorizeOAuthUseCase

AuthorizeOAuthUseCase

+ authorizeOAuth(ticket: string) : Promise<number | null>

Figura 108: Diagramma dell'interfaccia AuthorizeOAuthUseCase

Descrizione: Porta di ingresso per la validazione e il consumo del ticket OAuth

Descrizione dei metodi dell'interfaccia:

- `authorizeOAuth(ticket: string) : Promise<number | null>`: Consuma il ticket OAuth e restituisce l'ID utente associato, oppure null se il ticket non è valido

4.1.3.10 GetAccountStatusUseCase

GetAccountStatusUseCase

+ getAccountStatus(userId: number) : Promise<{isLinked: boolean}>

Figura 109: Diagramma dell'interfaccia GetAccountStatusUseCase

Descrizione: Porta di ingresso per il recupero dello stato di collegamento dell'account MyVimar

Descrizione dei metodi dell'interfaccia:

- `getAccountStatus(userId: number) : Promise<{isLinked: boolean}>`: Restituisce lo stato di collegamento dell'account MyVimar per l'utente indicato

4.1.3.11 GetTokensCallbackUseCase

GetTokensCallbackUseCase

+ getTokens(code: string, userId: number) : any

Figura 110: Diagramma dell'interfaccia GetTokensCallbackUseCase

Descrizione: Porta di ingresso per il recupero e il salvataggio dei token al termine del flusso OAuth

Descrizione dei metodi dell'interfaccia:

- `getTokens(code: string, userId: number) : any`: Esegue il recupero dei token di accesso e di rinnovo utilizzando il codice di autorizzazione ottenuto dal fornitore di identità e l'identificativo dell'utente.

4.1.3.12 PrepareOAuthUseCase

PrepareOAuthUseCase

+ prepareOAuth(userId: number) : Promise<string>

Figura 111: Diagramma dell'interfaccia PrepareOAuthUseCase

Descrizione: Porta di ingresso per la generazione del ticket OAuth da usare nel flusso di autorizzazione

Descrizione dei metodi dell'interfaccia:

- `prepareOAuth(userId: number) : Promise<string>`: Genera un ticket OAuth monouso per l'utente e restituisce il ticket come stringa

4.1.3.13 ApiAuthTokensService

ApiAuthTokensService

- eventEmitter: EventEmitter2
- getTokensWithCodePort: GetTokensWithCodePort
- writeTokensRepoPort: WriteTokensRepoPort

Figura 112: Diagramma della classe ApiAuthTokensService

Descrizione: Servizio applicativo che gestisce il recupero e il salvataggio dei token al completamento del flusso OAuth, pubblicando gli eventi corrispondenti

4.1.3.14 ApiAuthVimarService

ApiAuthVimarService

- authorizeHost: string
- clientId: string
- redirectUri: string

+ getLoginUrl(state?: string) : string

Figura 113: Diagramma della classe ApiAuthVimarService

Descrizione: Servizio applicativo che implementa `ApiAuthUseCase`. Costruisce l'URL di login OAuth MyVimar a partire dalle variabili di configurazione

Descrizione dei metodi della classe:

- `getLoginUrl(state?: string) : string`: Costruisce e restituisce l'URL di autorizzazione MyVimar con i parametri OAuth

4.1.3.15 OAuthTicketService

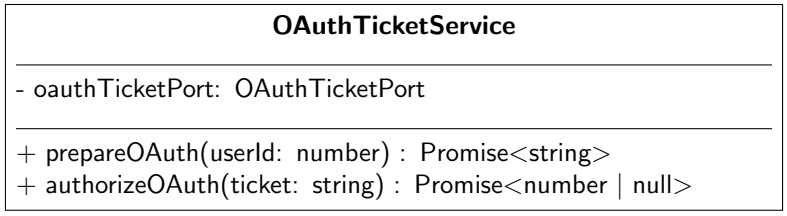

Figura 114: Diagramma della classe OAuthTicketService

Descrizione: Servizio applicativo che gestisce la generazione e il consumo dei ticket OAuth monouso per il flusso di autorizzazione

Descrizione dei metodi della classe:

- `prepareOAuth(userId: number) : Promise<string>`: Genera un ticket OAuth monouso, lo persiste con scadenza e restituisce il ticket
- `authorizeOAuth(ticket: string) : Promise<number | null>`: Consuma il ticket OAuth e restituisce l'ID utente associato, oppure null se scaduto o non trovato

4.1.3.16 TokenService

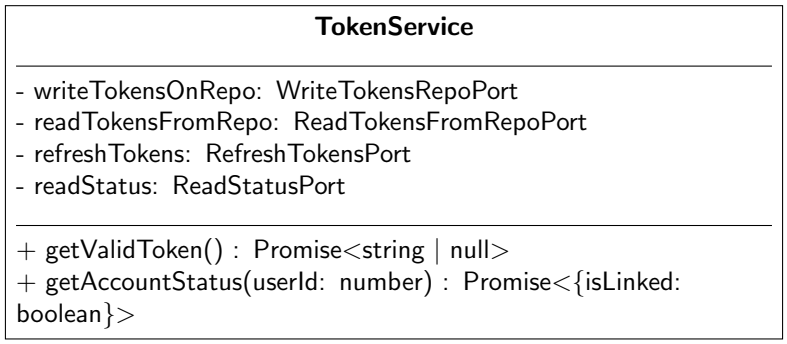

Figura 115: Diagramma della classe TokenService

Descrizione: Servizio applicativo che gestisce il ciclo di vita dei token OAuth: lettura, validazione, rinnovo e verifica dello stato di collegamento

Descrizione dei metodi della classe:

- `getValidToken() : Promise<string | null>`: Restituisce un token di accesso valido, rinnovandolo se scaduto; restituisce null se non disponibile
- `getAccountStatus(userId: number) : Promise<{isLinked: boolean}>`: Restituisce lo stato di collegamento dell'account MyVimar per l'utente indicato

4.1.3.17 DeleteTokensFromRepoPort

Figura 116: Diagramma dell'interfaccia DeleteTokensFromRepoPort

Descrizione: Porta di uscita per l'eliminazione dei token OAuth dal repository

Descrizione dei metodi dell'interfaccia:

- `deleteTokens() : Promise<boolean>`: Elimina i token salvati nel repository; restituisce true se l'operazione è avvenuta con successo

4.1.3.18 GetTokensWithCodePort

Figura 117: Diagramma dell'interfaccia GetTokensWithCodePort

Descrizione: Porta di uscita per il recupero dei token tramite codice di autorizzazione OAuth

Descrizione dei metodi dell'interfaccia:

- `getTokensWithCode(code: string) : Promise<{tokenPair: TokenPair}>`: Scambia il codice di autorizzazione OAuth con la coppia di token

4.1.3.19 GetValidTokenPort

Figura 118: Diagramma dell'interfaccia GetValidTokenPort

Descrizione: Porta di uscita per il recupero di un token di accesso valido

Descrizione dei metodi dell'interfaccia:

- `getValidToken() : Promise<string | null>`: Restituisce un token di accesso valido oppure null se non disponibile o scaduto

4.1.3.20 OAuthTicketPort

OAuthTicketPort

+ saveTicket(ticket: string, userId: number, expiresAt: Date) : Promise<boolean>
+ consumeTicket(ticket: string) : Promise<number | null>

Figura 119: Diagramma dell'interfaccia OAuthTicketPort

Descrizione: Porta di uscita per la gestione del ciclo di vita dei ticket OAuth monouso

Descrizione dei metodi dell'interfaccia:

- `saveTicket(ticket: string, userId: number, expiresAt: Date) : Promise<boolean>`: Persiste un ticket OAuth con l'ID utente associato e la scadenza
- `consumeTicket(ticket: string) : Promise<number | null>`: Consuma il ticket restituendo l'ID utente associato, oppure null se non valido

4.1.3.21 ReadStatusPort

ReadStatusPort

+ readStatus(userId: number) : Promise<{isLinked: boolean}>

Figura 120: Diagramma dell'interfaccia ReadStatusPort

Descrizione: Porta di uscita per la lettura dello stato di collegamento dell'account MyVimar

Descrizione dei metodi dell'interfaccia:

- `readStatus(userId: number) : Promise<{isLinked: boolean}>`: Restituisce lo stato di collegamento dell'account MyVimar per l'utente indicato

4.1.3.22 ReadTokensFromRepoPort

ReadTokensFromRepoPort

+ readTokens() : Promise<TokenPair>

Figura 121: Diagramma dell'interfaccia ReadTokensFromRepoPort

Descrizione: Porta di uscita per la lettura dei token OAuth dal repository

Descrizione dei metodi dell'interfaccia:

- `readTokens() : Promise<TokenPair>`: Legge e restituisce la coppia di token salvata nel repository

4.1.3.23 RefreshTokensPort

RefreshTokensPort

+ refreshTokens(refreshToken: string) : Promise<TokenPair | null>

Figura 122: Diagramma dell'interfaccia RefreshTokensPort

Descrizione: Porta di uscita per il rinnovo della coppia di token tramite refresh token

Descrizione dei metodi dell'interfaccia:

- `refreshTokens(refreshToken: string) : Promise<TokenPair | null>`: Rinnova i token tramite il refresh token; restituisce la nuova coppia o null in caso di errore

4.1.3.24 WriteTokensRepoPort

WriteTokensRepoPort

+ writeTokens(tokens: TokenPair, userId?: number, email?: string) : Promise<boolean>

Figura 123: Diagramma dell'interfaccia WriteTokensRepoPort

Descrizione: Porta di uscita per il salvataggio dei token OAuth nel repository

Descrizione dei metodi dell'interfaccia:

- `writeTokens(tokens: TokenPair, userId?: number, email?: string) : Promise<boolean>`: Salva la coppia di token nel repository, opzionalmente associandola all'utente e all'email

4.1.3.25 DeleteOAuthTicketCachePort

DeleteOAuthTicketCachePort

+ deleteTicket(ticket: string) : Promise<boolean>

Figura 124: Diagramma dell'interfaccia DeleteOAuthTicketCachePort

Descrizione: Porta di repository per l'eliminazione dei ticket OAuth dalla cache

Descrizione dei metodi dell'interfaccia:

- `deleteTicket(ticket: string) : Promise<boolean>`: Elimina il ticket OAuth indicato dalla cache

4.1.3.26 DeleteTokensCachePort

DeleteTokensCachePort

+ deleteTokens() : Promise<boolean>

Figura 125: Diagramma dell'interfaccia DeleteTokensCachePort

Descrizione: Porta di repository per l'eliminazione dei token OAuth dalla cache

Descrizione dei metodi dell'interfaccia:

- `deleteTokens() : Promise<boolean>`: Elimina i token dalla cache

4.1.3.27 GetTokensFromApiPort

GetTokensFromApiPort

+ getTokensWithCode(code: string) : Promise<TokensDto | null>

Figura 126: Diagramma dell'interfaccia GetTokensFromApiPort

Descrizione: Porta di repository per il recupero dei token dall'API MyVimar tramite codice OAuth

Descrizione dei metodi dell'interfaccia:

- `getTokensWithCode(code: string) : Promise<TokensDto | null>`: Scambia il codice di autorizzazione con i token tramite le API MyVimar

4.1.3.28 ReadOAuthTicketCacheEntity

ReadOAuthTicketCacheEntity

+ readValidTicket(ticket: string) : Promise<ReadOAuthTicketCacheEntity | null>

Figura 127: Diagramma dell'interfaccia ReadOAuthTicketCacheEntity

Descrizione: Porta di repository per la lettura dei ticket OAuth validi dalla cache

Descrizione dei metodi dell'interfaccia:

- `readValidTicket(ticket: string) : Promise<ReadOAuthTicketCacheEntity | null>`: Legge un ticket valido dalla cache; restituisce null se scaduto o non trovato

4.1.3.29 ReadStatusRepoPort

ReadStatusRepoPort

+ readStatus(userId: number) : Promise<string | null>

Figura 128: Diagramma dell'interfaccia ReadStatusRepoPort

Descrizione: Porta di repository per la lettura dello stato di collegamento dell'account MyVimar

Descrizione dei metodi dell'interfaccia:

- `readStatus(userId: number) : Promise<string | null>`: Legge lo stato di collegamento dell'account per l'utente indicato

4.1.3.30 ReadTokensCachePort

ReadTokensCachePort

+ readTokens() : Promise<TokenEntity | null>

Figura 129: Diagramma dell'interfaccia ReadTokensCachePort

Descrizione: Porta di repository per la lettura dei token OAuth dalla cache

Descrizione dei metodi dell'interfaccia:

- `readTokens() : Promise<TokenEntity | null>`: Legge i token dalla cache; restituisce null se non presenti

4.1.3.31 RefreshTokensFromApiPort

RefreshTokensFromApiPort

+ refresh(refreshToken: string) : Promise<TokensDto | null>

Figura 130: Diagramma dell'interfaccia RefreshTokensFromApiPort

Descrizione: Porta di repository per il rinnovo dei token tramite le API MyVimar

Descrizione dei metodi dell'interfaccia:

- `refresh(refreshToken: string) : Promise<TokensDto | null>`: Rinnova i token tramite le API MyVimar usando il refresh token

4.1.3.32 WriteOAuthTicketCachePort

WriteOAuthTicketCachePort

+ writeTicket(ticket: string, userId: number, expiresAt: Date) : Promise<boolean>

Figura 131: Diagramma dell'interfaccia WriteOAuthTicketCachePort

Descrizione: Porta di repository per la scrittura dei ticket OAuth nella cache

Descrizione dei metodi dell'interfaccia:

- `writeTicket(ticket: string, userId: number, expiresAt: Date) : Promise<boolean>`: Scrive un ticket OAuth nella cache con l'ID utente e la scadenza associati

4.1.3.33 WriteTokensCachePort

WriteTokensCachePort

+ writeTokens(accessToken: string, refreshToken: string,
expiresAt: Date, userId: number, email: string) : Promise<boolean>

Figura 132: Diagramma dell'interfaccia WriteTokensCachePort

Descrizione: Porta di repository per la scrittura dei token OAuth nella cache

Descrizione dei metodi dell'interfaccia:

- `writeTokens(accessToken: string, refreshToken: string, expiresAt: Date, userId: number, email: string) : Promise<boolean>`: Scrive la coppia di token nella cache con tutti i metadati associati

4.1.3.34 OAuthTicketEntity

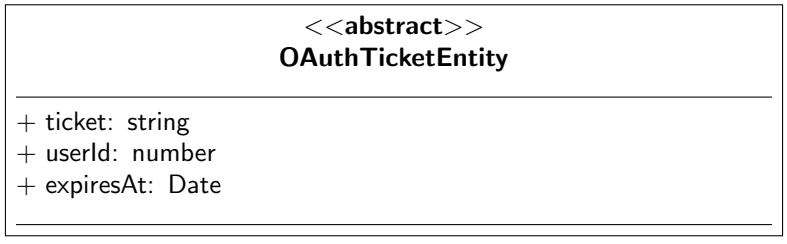

Figura 133: Diagramma della classe astratta OAuthTicketEntity

Descrizione: Interfaccia dell'entità di persistenza che rappresenta un ticket OAuth salvato in cache, con i campi di ID utente e scadenza

4.1.3.35 TokenEntity

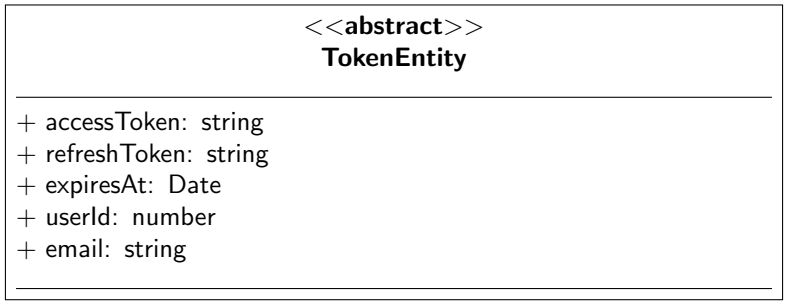

Figura 134: Diagramma della classe astratta TokenEntity

Descrizione: Interfaccia dell'entità di persistenza che rappresenta una coppia di token OAuth salvata in cache

4.1.3.36 DeleteTokensFromRepoAdapter

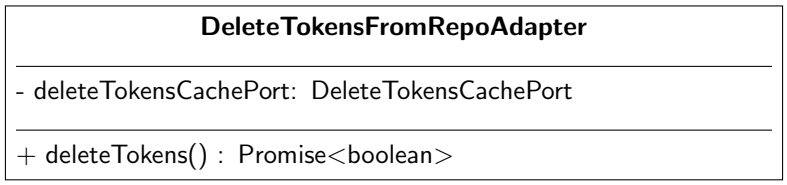

Figura 135: Diagramma della classe DeleteTokensFromRepoAdapter

Descrizione: Adapter che implementa `DeleteTokensFromRepoPort`, delegando l'eliminazione dei token alla porta di cache

Descrizione dei metodi della classe:

- `deleteTokens() : Promise<boolean>`: Elimina i token dal repository delegando alla cache

4.1.3.37 GetTokenWithCodeAdapter

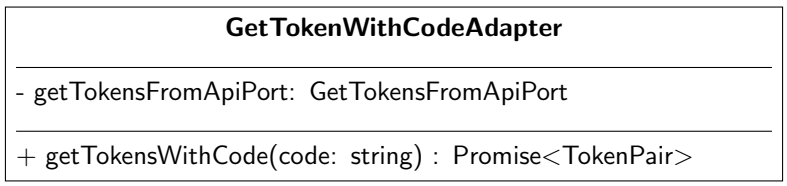

Figura 136: Diagramma della classe GetTokenWithCodeAdapter

Descrizione: Adapter che implementa `GetTokensWithCodePort`, delegando il recupero dei token all'API MyVimar

Descrizione dei metodi della classe:

- `getTokensWithCode(code: string) : Promise<TokenPair>`: Recupera i token dall'API MyVimar tramite il codice OAuth e li restituisce come `TokenPair`

4.1.3.38 OAuthTicketAdapter

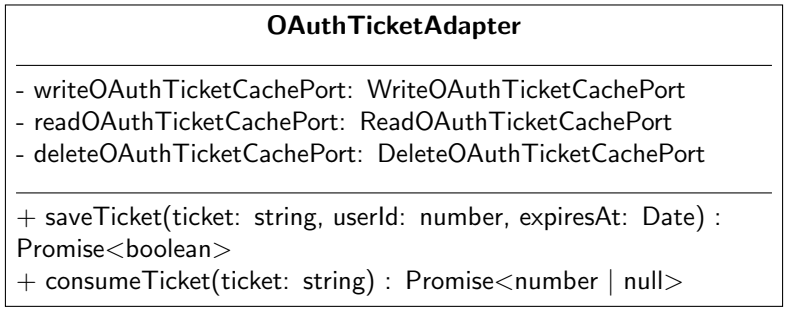

Figura 137: Diagramma della classe OAuthTicketAdapter

Descrizione: Adapter che implementa `OAuthTicketPort`, orchestrando lettura, scrittura e cancellazione dei ticket OAuth sulla cache

Descrizione dei metodi della classe:

- `saveTicket(ticket: string, userId: number, expiresAt: Date) : Promise<boolean>`: Persiste un ticket OAuth nella cache con l'ID utente e la scadenza
- `consumeTicket(ticket: string) : Promise<number | null>`: Legge e rimuove il ticket dalla cache, restituendo l'ID utente associato o null

4.1.3.39 ReadStatusAdapter

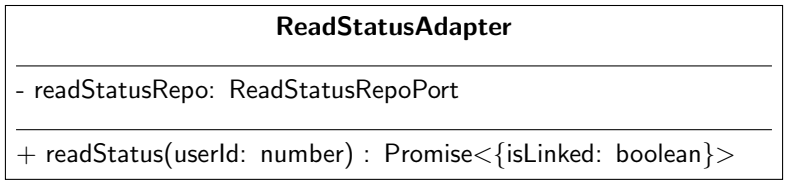

Figura 138: Diagramma della classe ReadStatusAdapter

Descrizione: Adapter che implementa `ReadStatusPort`, delegando la lettura dello stato al repository

Descrizione dei metodi della classe:

- `readStatus(userId: number) : Promise<{isLinked: boolean}>`: Legge lo stato di collegamento dell'account MyVimar per l'utente indicato

4.1.3.40 ReadTokensFromRepoAdapter

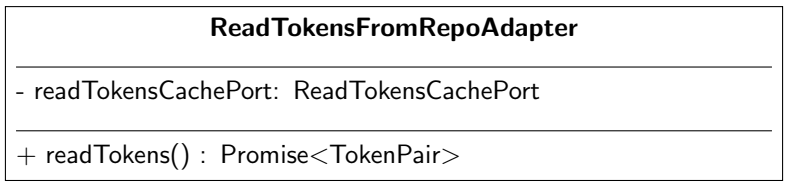

Figura 139: Diagramma della classe ReadTokensFromRepoAdapter

Descrizione: Adapter che implementa `ReadTokensFromRepoPort`, delegando la lettura dei token alla cache

Descrizione dei metodi della classe:

- `readTokens() : Promise<TokenPair>`: Legge i token dalla cache e li restituisce come oggetto di dominio `TokenPair`

4.1.3.41 RefreshTokensAdapter

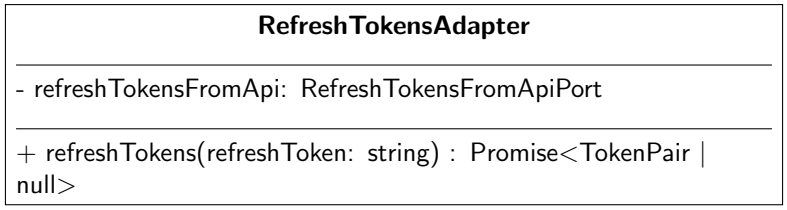

Figura 140: Diagramma della classe RefreshTokensAdapter

Descrizione: Adapter che implementa `RefreshTokensPort`, delegando il rinnovo dei token alle API MyVimar

Descrizione dei metodi della classe:

- `refreshTokens(refreshToken: string) : Promise<TokenPair | null>`: Rinnova i token tramite le API MyVimar e li restituisce come `TokenPair`, oppure null in caso di errore

4.1.3.42 WriteTokensRepoAdapter

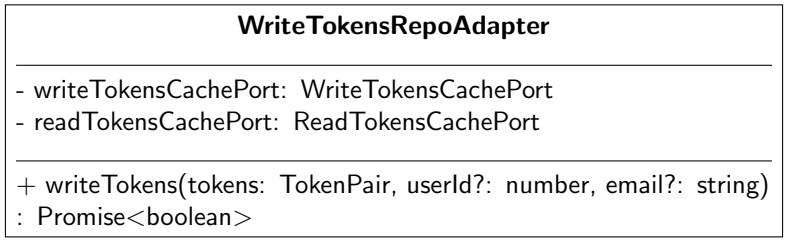

Figura 141: Diagramma della classe WriteTokensRepoAdapter

Descrizione: Adapter che implementa `WriteTokensRepoPort`, delegando la scrittura dei token alla cache

Descrizione dei metodi della classe:

- `writeTokens(tokens: TokenPair, userId?: number, email?: string) : Promise<boolean>`: Salva la coppia di token nella cache, opzionalmente con ID utente ed email

4.1.3.43 GetTokensFromApiImpl

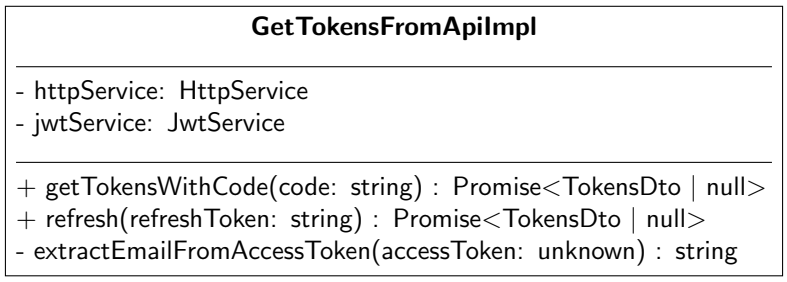

Figura 142: Diagramma della classe GetTokensFromApiImpl

Descrizione: Implementazione concreta di `GetTokensFromApiPort` e `RefreshTokensFromApiPort`; esegue le chiamate HTTP alle API MyVimar per ottenere e rinnovare i token

Descrizione dei metodi della classe:

- `getTokensWithCode(code: string) : Promise<TokensDto | null>`: Scambia il codice OAuth con i token chiamando le API MyVimar; restituisce null in caso di errore
- `refresh(refreshToken: string) : Promise<TokensDto | null>`: Rinnova i token tramite le API MyVimar usando il refresh token; restituisce null in caso di errore
- `extractEmailFromAccessToken(accessToken: unknown) : string`: Decodifica il token JWT di accesso e ne estrae il campo email

4.1.3.44 OAuthTicketCacheImpl

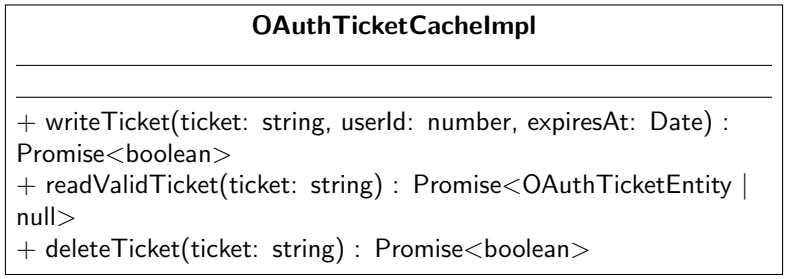

Figura 143: Diagramma della classe OAuthTicketCacheImpl

Descrizione: Implementazione concreta delle porte di cache per i ticket OAuth; gestisce scrittura, lettura e cancellazione tramite il sistema di cache

Descrizione dei metodi della classe:

- `writeTicket(ticket: string, userId: number, expiresAt: Date) : Promise<boolean>`: Scrive un ticket OAuth nella cache con ID utente e scadenza
- `readValidTicket(ticket: string) : Promise<OAuthTicketEntity | null>`: Legge un ticket dalla cache verificandone la validità; restituisce null se scaduto o assente
- `deleteTicket(ticket: string) : Promise<boolean>`: Elimina il ticket indicato dalla cache

4.1.3.45 TokenCacheImpl

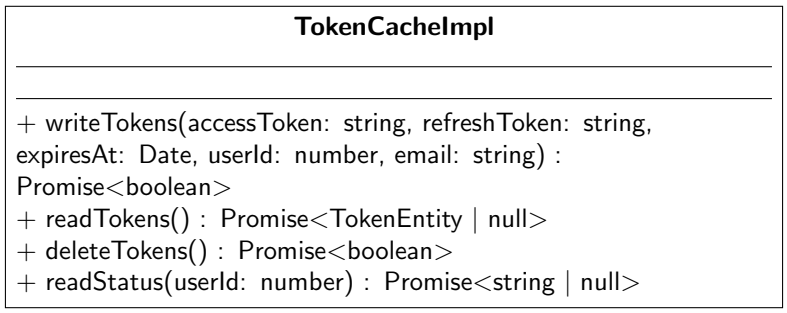

Figura 144: Diagramma della classe TokenCacheImpl

Descrizione: Implementazione concreta delle porte di cache per i token OAuth; gestisce lettura, scrittura, cancellazione e lettura dello stato di collegamento

Descrizione dei metodi della classe:

- `writeTokens(accessToken: string, refreshToken: string, expiresAt: Date, userId: number, email: string) : Promise<boolean>`: Scrive la coppia di token nella cache con tutti i metadati associati
- `readTokens() : Promise<TokenEntity | null>`: Legge i token dalla cache; restituisce null se non presenti
- `deleteTokens() : Promise<boolean>`: Elimina i token dalla cache
- `readStatus(userId: number) : Promise<string | null>`: Legge lo stato di collegamento dell'account per l'utente indicato

4.1.4 Auth

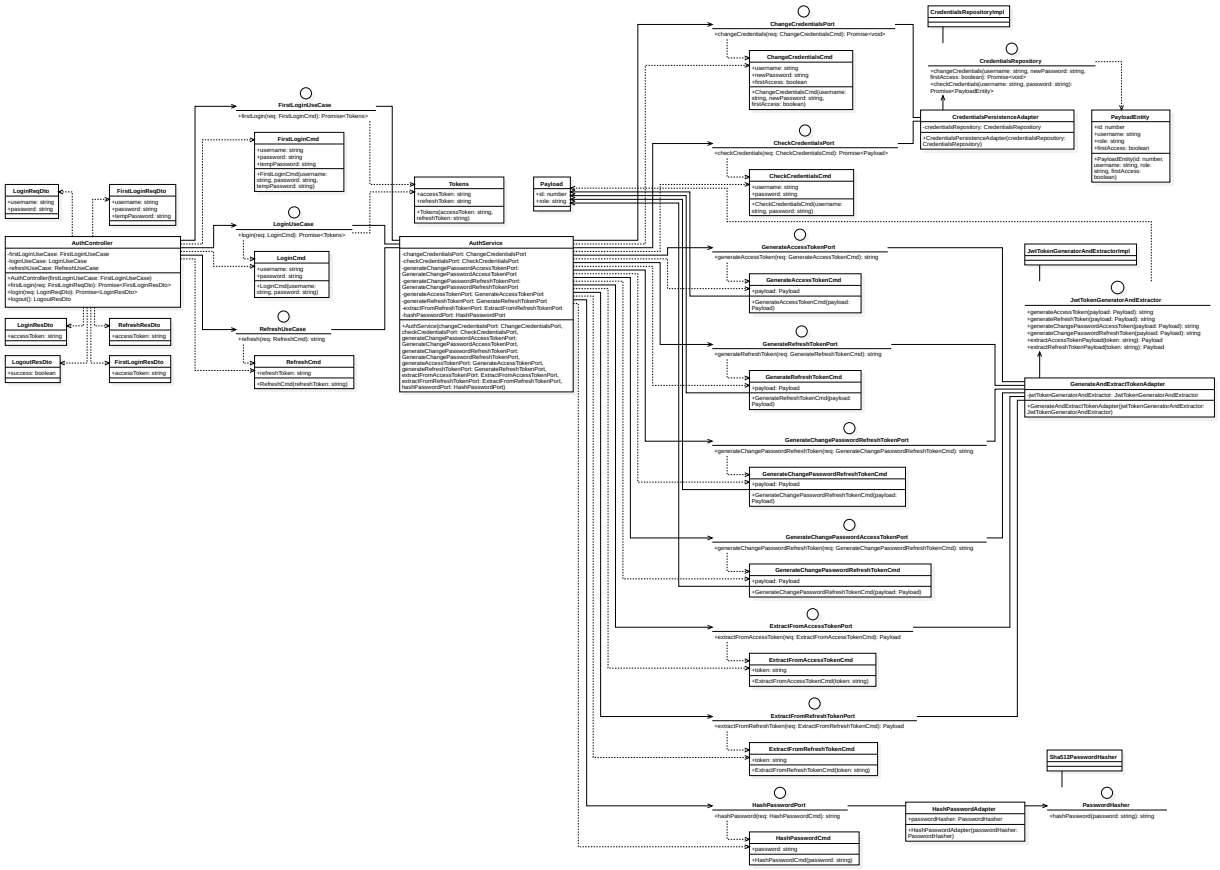

Figura 145: Diagramma delle classi del modulo Auth

4.1.4.1 FirstLoginReqDto

FirstLoginReqDto
username: string password: string tempPassword: string

Figura 146: Diagramma della classe FirstLoginReqDto

Descrizione: DTO di richiesta per il primo accesso, contenente credenziali temporanee e nuova password

4.1.4.2 LoginReqDto

LoginReqDto
username: string password: string

Figura 147: Diagramma della classe LoginReqDto

Descrizione: DTO di richiesta per l'autenticazione standard

4.1.4.3 RefreshReqDto

Figura 148: Diagramma della classe RefreshReqDto

Descrizione: DTO di richiesta per il rinnovo dell'access token

4.1.4.4 FirstLoginResDto

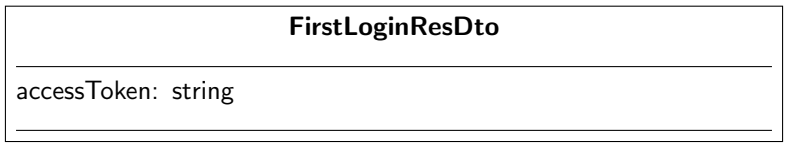

Figura 149: Diagramma della classe FirstLoginResDto

Descrizione: DTO di risposta restituito al completamento del primo accesso

4.1.4.5 LoginResDto

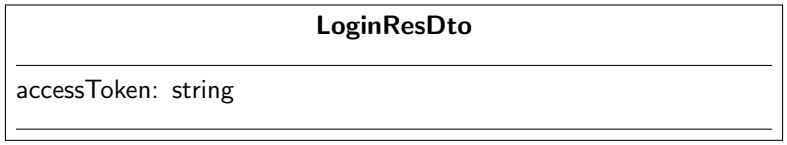

Figura 150: Diagramma della classe LoginResDto

Descrizione: DTO di risposta restituito a seguito di un'autenticazione avvenuta con successo

4.1.4.6 LogoutResDto

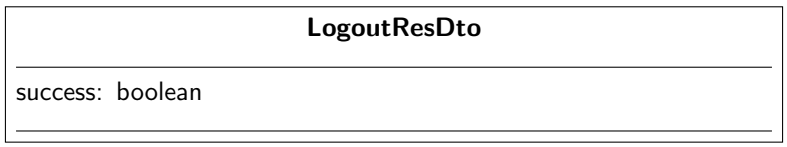

Figura 151: Diagramma della classe LogoutResDto

Descrizione: DTO di risposta restituito a seguito di una richiesta di logout

4.1.4.7 RefreshResDto

Figura 152: Diagramma della classe RefreshResDto

Descrizione: DTO di risposta restituito a seguito del rinnovo dell'access token

4.1.4.8 Payload

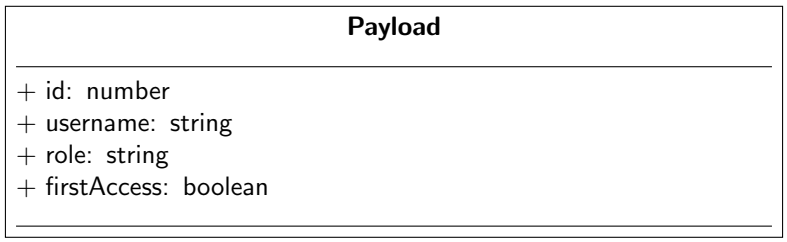

Figura 153: Diagramma della classe Payload

Descrizione: Modello di dominio che rappresenta il payload contenuto nei token JWT

4.1.4.9 Tokens

Figura 154: Diagramma della classe Tokens

Descrizione: Modello di dominio che raggruppa la coppia di token JWT generati all'autenticazione

4.1.4.10 AuthController

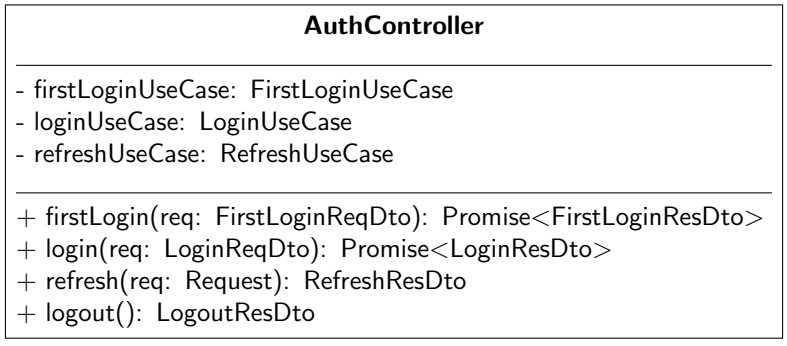

Figura 155: Diagramma della classe AuthController

Descrizione: Controller REST che gestisce le richieste HTTP relative all'autenticazione

Descrizione dei metodi della classe:

- `firstLogin(req: FirstLoginReqDto): Promise<FirstLoginResDto>`: Espone l'endpoint per il primo accesso con cambio della password temporanea
- `login(req: LoginReqDto): Promise<LoginResDto>`: Espone l'endpoint per l'autenticazione standard con username e password
- `refresh(req: Request): RefreshResDto`: Espone l'endpoint per il rinnovo dell'access token tramite refresh token
- `logout(): LogoutResDto`: Espone l'endpoint per l'invalidazione della sessione corrente

4.1.4.11 ChangeCredentialsCmd

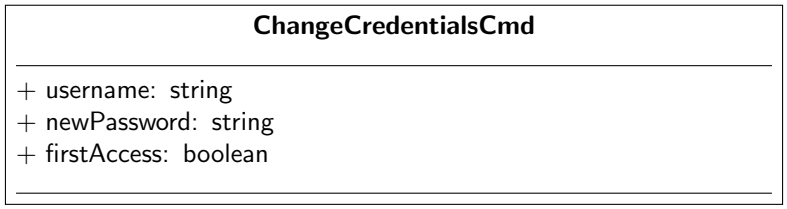

Figura 156: Diagramma della classe ChangeCredentialsCmd

Descrizione: Comando per l'aggiornamento delle credenziali di un utente

4.1.4.12 CheckCredentialsCmd

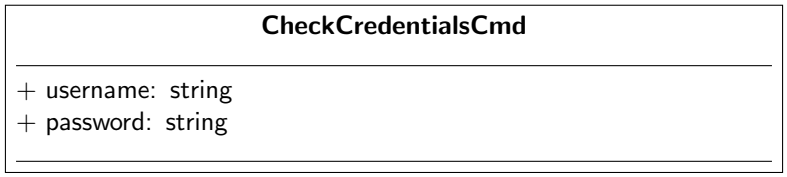

Figura 157: Diagramma della classe CheckCredentialsCmd

Descrizione: Comando per la verifica delle credenziali di accesso

4.1.4.13 CheckFirstAccessCmd

Figura 158: Diagramma della classe CheckFirstAccessCmd

Descrizione: Comando per la verifica se un utente deve ancora completare il primo accesso

4.1.4.14 ExtractFromAccessTokenCmd

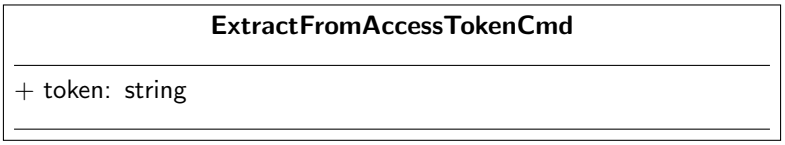

Figura 159: Diagramma della classe ExtractFromAccessTokenCmd

Descrizione: Comando per l'estrazione del payload da un access token JWT

4.1.4.15 ExtractFromRefreshTokenCmd

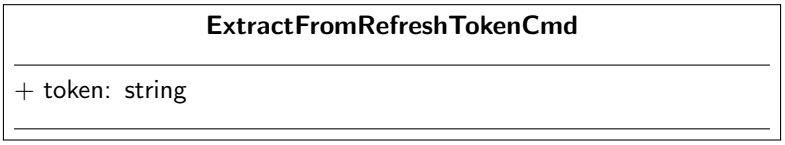

Figura 160: Diagramma della classe ExtractFromRefreshTokenCmd

Descrizione: Comando per l'estrazione del payload da un refresh token JWT

4.1.4.16 FirstLoginCmd

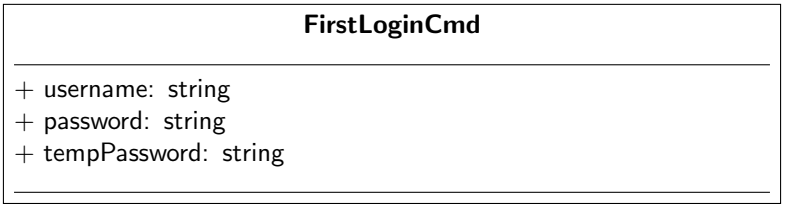

Figura 161: Diagramma della classe FirstLoginCmd

Descrizione: Comando per l'esecuzione del flusso di primo accesso

4.1.4.17 GenerateAccessTokenCmd

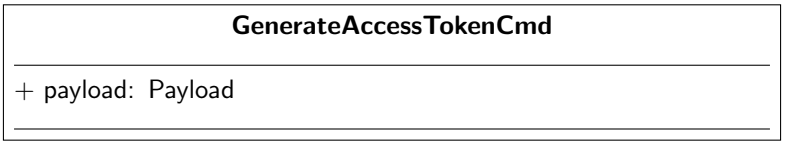

Figura 162: Diagramma della classe GenerateAccessTokenCmd

Descrizione: Comando per la generazione di un access token JWT standard

4.1.4.18 GenerateChangePasswordAccessTokenCmd

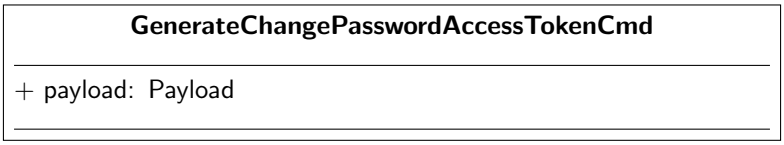

Figura 163: Diagramma della classe GenerateChangePasswordAccessTokenCmd

Descrizione: Comando per la generazione di un access token JWT dedicato al cambio password

4.1.4.19 GenerateChangePasswordRefreshTokenCmd

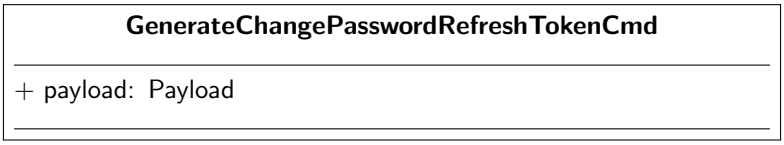

Figura 164: Diagramma della classe GenerateChangePasswordRefreshTokenCmd

Descrizione: Comando per la generazione di un refresh token JWT dedicato al cambio password

4.1.4.20 GenerateRefreshTokenCmd

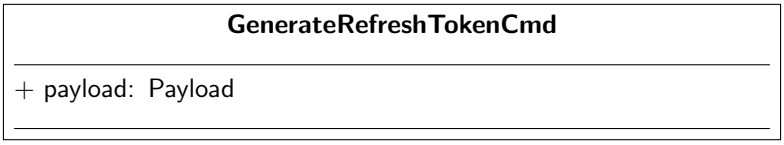

Figura 165: Diagramma della classe GenerateRefreshTokenCmd

Descrizione: Comando per la generazione di un refresh token JWT standard

4.1.4.21 HashPasswordCmd

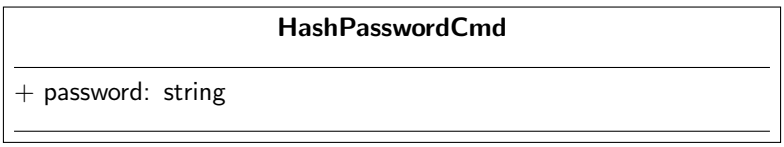

Figura 166: Diagramma della classe HashPasswordCmd

Descrizione: Comando per l'hashing sicuro di una password

4.1.4.22 LoginCmd

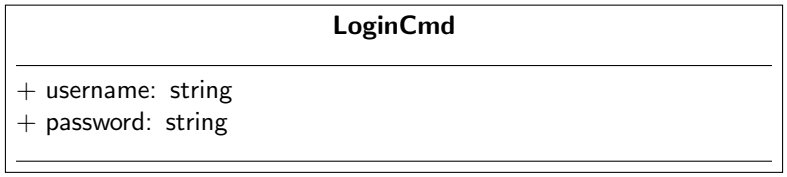

Figura 167: Diagramma della classe LoginCmd

Descrizione: Comando per l'esecuzione del flusso di autenticazione standard

4.1.4.23 RefreshCmd

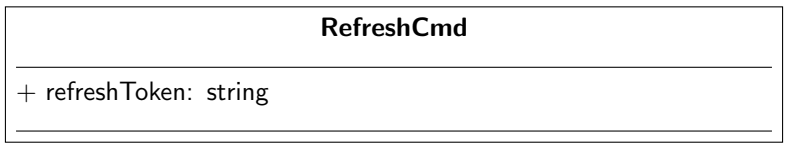

Figura 168: Diagramma della classe RefreshCmd

Descrizione: Comando per il rinnovo dell'access token tramite refresh token

4.1.4.24 FirstLoginUseCase

Figura 169: Diagramma dell'interfaccia FirstLoginUseCase

Descrizione: Interfaccia del use case per la gestione del primo accesso

Descrizione dei metodi dell'interfaccia:

- `firstLogin(req: FirstLoginCmd): Promise<Tokens>`: Gestisce il primo accesso verificando le credenziali temporanee e aggiornando la password

4.1.4.25 LoginUseCase

Figura 170: Diagramma dell'interfaccia LoginUseCase

Descrizione: Interfaccia del use case per la gestione dell'autenticazione standard

Descrizione dei metodi dell'interfaccia:

- `login(req: LoginCmd): Promise<Tokens>`: Verifica le credenziali e genera la coppia di token JWT per la sessione

4.1.4.26 RefreshUseCase

RefreshUseCase

+ refresh(req: RefreshCmd): string

Figura 171: Diagramma dell'interfaccia RefreshUseCase

Descrizione: Interfaccia del use case per il rinnovo dell'access token

Descrizione dei metodi dell'interfaccia:

- `refresh(req: RefreshCmd): string`: Valida il refresh token e genera un nuovo access token

4.1.4.27 AuthService

AuthService

- changeCredentialsPort: ChangeCredentialsPort
- checkCredentialsPort: CheckCredentialsPort
- generateChangePasswordAccessTokenPort: GenerateChangePasswordAccessTokenPort
- generateChangePasswordRefreshTokenPort: GenerateChangePasswordRefreshTokenPort
- generateAccessTokenPort: GenerateAccessTokenPort
- generateRefreshTokenPort: GenerateRefreshTokenPort
- extractFromAccessTokenPort: ExtractFromAccessTokenPort
- extractFromRefreshTokenPort: ExtractFromRefreshTokenPort
- hashPasswordPort: HashPasswordPort

+ firstLogin(req: FirstLoginCmd): Promise<Tokens>
+ login(req: LoginCmd): Promise<Tokens>

Figura 172: Diagramma della classe AuthService

Descrizione: Servizio applicativo che implementa i use case di autenticazione

Descrizione dei metodi della classe:

- `firstLogin(req: FirstLoginCmd): Promise<Tokens>`: Implementa il flusso di primo accesso coordinando verifica credenziali, aggiornamento password e generazione token
- `login(req: LoginCmd): Promise<Tokens>`: Implementa il flusso di autenticazione standard coordinando verifica credenziali e generazione token

4.1.4.28 ChangeCredentialsPort

ChangeCredentialsPort

+ changeCredentials(req: ChangeCredentialsCmd): Promise<void>

Figura 173: Diagramma dell'interfaccia ChangeCredentialsPort

Descrizione: Porta di uscita per l'aggiornamento delle credenziali utente

Descrizione dei metodi dell'interfaccia:

- `changeCredentials(req: ChangeCredentialsCmd): Promise<void>`: Aggiorna username, password e flag di primo accesso dell'utente nel sistema di persistenza

4.1.4.29 CheckCredentialsPort

CheckCredentialsPort

+ checkCredentials(req: CheckCredentialsCmd): Promise<Payload>

Figura 174: Diagramma dell'interfaccia CheckCredentialsPort

Descrizione: Porta di uscita per la verifica delle credenziali utente

Descrizione dei metodi dell'interfaccia:

- `checkCredentials(req: CheckCredentialsCmd): Promise<Payload>`: Verifica le credenziali e restituisce il payload dell'utente autenticato

4.1.4.30 ExtractFromAccessTokenPort

ExtractFromAccessTokenPort

+ extractFromAccessToken(req: ExtractFromAccessTokenCmd): Payload

Figura 175: Diagramma dell'interfaccia ExtractFromAccessTokenPort

Descrizione: Porta di uscita per l'estrazione del payload da un access token JWT

Descrizione dei metodi dell'interfaccia:

- `extractFromAccessToken(req: ExtractFromAccessTokenCmd): Payload`: Decodifica e valida un access token restituendone il payload

4.1.4.31 ExtractFromRefreshTokenPort

ExtractFromRefreshTokenPort

+ extractFromRefreshToken(req: ExtractFromRefreshTokenCmd): Payload

Figura 176: Diagramma dell'interfaccia ExtractFromRefreshTokenPort

Descrizione: Porta di uscita per l'estrazione del payload da un refresh token JWT

Descrizione dei metodi dell'interfaccia:

- `extractFromRefreshToken(req: ExtractFromRefreshTokenCmd): Payload`: Decodifica e valida un refresh token restituendone il payload

4.1.4.32 GenerateAccessTokenPort

GenerateAccessTokenPort

+ generateAccessToken(req: GenerateAccessTokenCmd): string

Figura 177: Diagramma dell'interfaccia GenerateAccessTokenPort

Descrizione: Porta di uscita per la generazione di un access token JWT standard

Descrizione dei metodi dell'interfaccia:

- `generateAccessToken(req: GenerateAccessTokenCmd): string`: Genera un access token JWT firmato a partire dal payload fornito

4.1.4.33 GenerateChangePasswordAccessTokenPort

GenerateChangePasswordAccessTokenPort

+ generateChangePasswordAccessToken(req: GenerateChangePasswordAccessTokenCmd): string

Figura 178: Diagramma dell'interfaccia GenerateChangePasswordAccessTokenPort

Descrizione: Porta di uscita per la generazione di un access token JWT dedicato al cambio password

Descrizione dei metodi dell'interfaccia:

- `generateChangePasswordAccessToken(req: GenerateChangePasswordAccessTokenCmd): string`: Genera un access token JWT con scopo limitato al cambio password

4.1.4.34 GenerateChangePasswordRefreshTokenPort

GenerateChangePasswordRefreshTokenPort

+ generateChangePasswordRefreshToken(req: GenerateChangePasswordRefreshTokenCmd): string

Figura 179: Diagramma dell'interfaccia GenerateChangePasswordRefreshTokenPort

Descrizione: Porta di uscita per la generazione di un refresh token JWT dedicato al cambio password

Descrizione dei metodi dell'interfaccia:

- `generateChangePasswordRefreshToken(req: GenerateChangePasswordRefreshTokenCmd): string`: Genera un refresh token JWT con scopo limitato al cambio password

4.1.4.35 GenerateRefreshTokenPort

GenerateRefreshTokenPort

+ generateRefreshToken(req: GenerateRefreshTokenCmd): string

Figura 180: Diagramma dell'interfaccia GenerateRefreshTokenPort

Descrizione: Porta di uscita per la generazione di un refresh token JWT standard

Descrizione dei metodi dell'interfaccia:

- `generateRefreshToken(req: GenerateRefreshTokenCmd): string`: Genera un refresh token JWT firmato a partire dal payload fornito

4.1.4.36 HashPasswordPort

HashPasswordPort

+ hashPassword(req: HashPasswordCmd): string

Figura 181: Diagramma dell'interfaccia HashPasswordPort

Descrizione: Porta di uscita per l'hashing sicuro delle password

Descrizione dei metodi dell'interfaccia:

- `hashPassword(req: HashPasswordCmd): string`: Applica una funzione di hashing alla password e restituisce il digest risultante

4.1.4.37 PayloadEntity

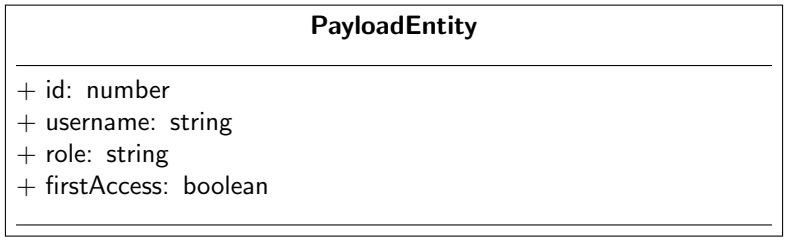

Figura 182: Diagramma della classe PayloadEntity

Descrizione: Entità di persistenza che rappresenta i dati utente estratti dal database per la costruzione del payload JWT

4.1.4.38 HashPasswordAdapter

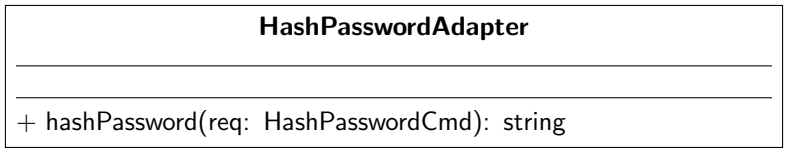

Figura 183: Diagramma della classe HashPasswordAdapter

Descrizione: Adattatore che implementa la porta di hashing delle password delegando a una libreria crittografica

Descrizione dei metodi della classe:

- `hashPassword(req: HashPasswordCmd): string`: Applica l'algoritmo di hashing alla password e restituisce il digest

4.1.4.39 CredentialsPersistenceAdapter

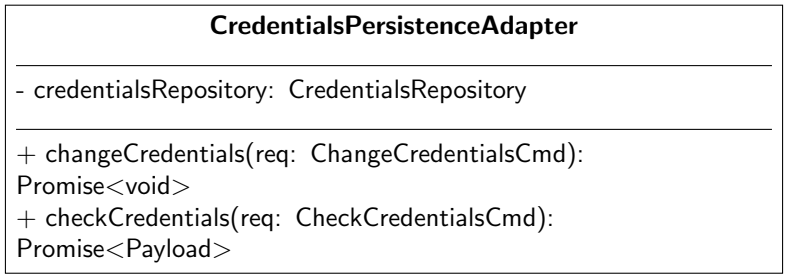

Figura 184: Diagramma della classe CredentialsPersistenceAdapter

Descrizione: Adattatore di persistenza che implementa le porte relative alle credenziali utente

Descrizione dei metodi della classe:

- `changeCredentials(req: ChangeCredentialsCmd): Promise<void>`: Delega al repository l'aggiornamento delle credenziali dell'utente
- `checkCredentials(req: CheckCredentialsCmd): Promise<Payload>`: Delega al repository la verifica delle credenziali e mappa il risultato nel modello di dominio

4.1.4.40 GenerateAndExtractTokenAdapter

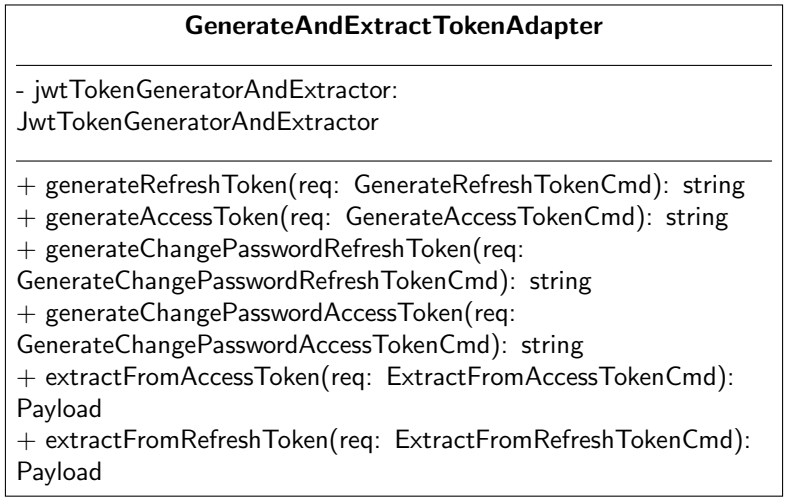

Figura 185: Diagramma della classe GenerateAndExtractTokenAdapter

Descrizione: Adattatore che implementa le porte di generazione ed estrazione dei token JWT

Descrizione dei metodi della classe:

- `generateRefreshToken(req: GenerateRefreshTokenCmd): string`: Delega al componente JWT la generazione del refresh token standard
- `generateAccessToken(req: GenerateAccessTokenCmd): string`: Delega al componente JWT la generazione dell'access token standard
- `generateChangePasswordRefreshToken(req: GenerateChangePasswordRefreshTokenCmd): string`: Delega al componente JWT la generazione del refresh token di cambio password
- `generateChangePasswordAccessToken(req: GenerateChangePasswordAccessTokenCmd): string`: Delega al componente JWT la generazione dell'access token di cambio password
- `extractFromAccessToken(req: ExtractFromAccessTokenCmd): Payload`: Delega al componente JWT l'estrazione del payload dall'access token
- `extractFromRefreshToken(req: ExtractFromRefreshTokenCmd): Payload`: Delega al componente JWT l'estrazione del payload dal refresh token

4.1.4.41 CredentialsRepository

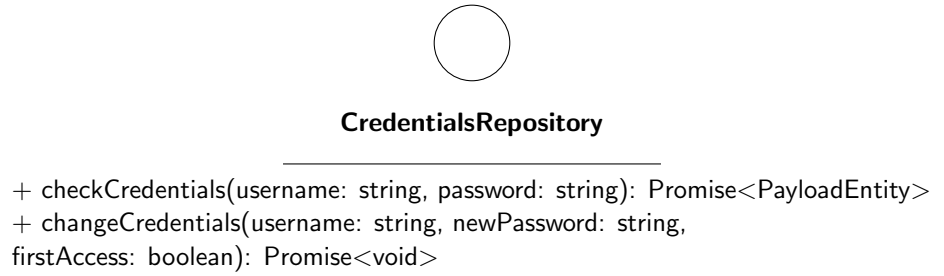

Figura 186: Diagramma dell'interfaccia CredentialsRepository

Descrizione: Interfaccia del repository per le operazioni di persistenza relative alle credenziali utente

Descrizione dei metodi dell'interfaccia:

- `checkCredentials(username: string, password: string): Promise<PayloadEntity>`: Verifica nel database la corrispondenza tra username e password e restituisce i dati dell'utente
- `changeCredentials(username: string, newPassword: string, firstAccess: boolean): Promise<void>`: Aggiorna nel database la password e il flag di primo accesso dell'utente specificato

4.1.4.42 JwtTokenGeneratorAndExtractor

JwtTokenGeneratorAndExtractor

+ generateAccessToken(payload: Payload): string
+ generateRefreshToken(payload: Payload): string
+ generateChangePasswordAccessToken(payload: Payload): string
+ generateChangePasswordRefreshToken(payload: Payload): string
+ extractAccessTokenPayload(token: string): Payload
+ extractRefreshTokenPayload(token: string): Payload

Figura 187: Diagramma dell'interfaccia JwtTokenGeneratorAndExtractor

Descrizione: Interfaccia per la generazione e l'estrazione dei payload dei token JWT

Descrizione dei metodi dell'interfaccia:

- `generateAccessToken(payload: Payload): string`: Genera un access token JWT firmato con il payload fornito
- `generateRefreshToken(payload: Payload): string`: Genera un refresh token JWT firmato con il payload fornito
- `generateChangePasswordAccessToken(payload: Payload): string`: Genera un access token JWT con scopo limitato al cambio password
- `generateChangePasswordRefreshToken(payload: Payload): string`: Genera un refresh token JWT con scopo limitato al cambio password
- `extractAccessTokenPayload(token: string): Payload`: Decodifica e valida un access token restituendone il payload
- `extractRefreshTokenPayload(token: string): Payload`: Decodifica e valida un refresh token restituendone il payload

4.1.4.43 PasswordHasher

PasswordHasher

+ hashPassword(password: string): string

Figura 188: Diagramma dell'interfaccia PasswordHasher

Descrizione: Interfaccia per l'hashing sicuro delle password

Descrizione dei metodi dell'interfaccia:

- `hashPassword(password: string): string`: Applica una funzione di hashing one-way alla password e restituisce il digest risultante

4.1.5 Cache

4.1.5.1 CollectionMetaDto

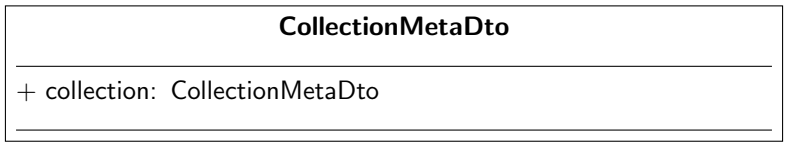

Figura 189: Diagramma della classe CollectionMetaDto

Descrizione: Wrapper per i metadati di paginazione restituiti dall'API esterna

4.1.5.2 DatapointAttributesDto

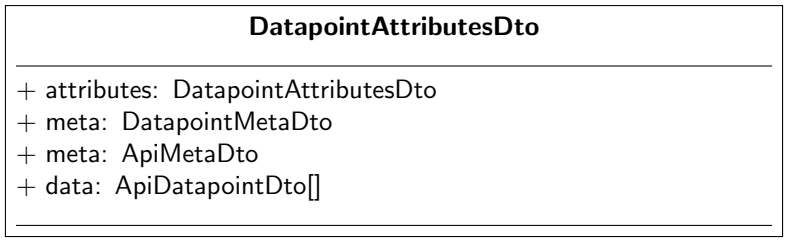

Figura 190: Diagramma della classe DatapointAttributesDto

Descrizione: DTO che mappa gli attributi di un datapoint ricevuto dall'API esterna

4.1.5.3 DeviceAttributesDto

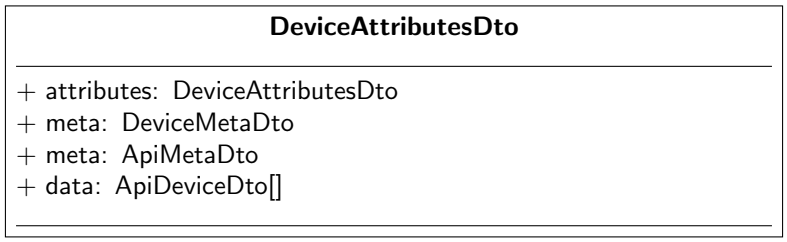

Figura 191: Diagramma della classe DeviceAttributesDto

Descrizione: DTO che mappa gli attributi di un dispositivo ricevuto dall'API esterna

4.1.5.4 ApiPlantAttributesDto

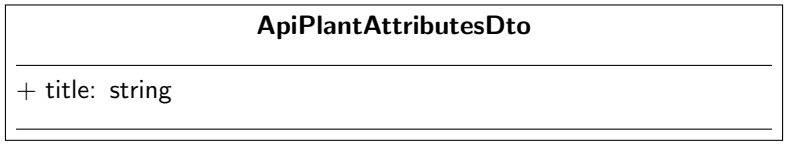

Figura 192: Diagramma della classe ApiPlantAttributesDto

Descrizione: DTO che mappa gli attributi principali di un impianto ricevuto dall'API esterna

4.1.5.5 ApiPlantMetaDto

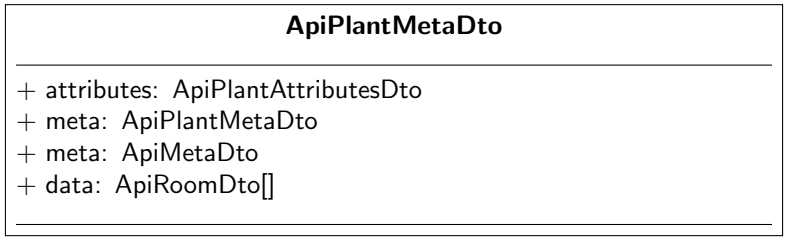

Figura 193: Diagramma della classe ApiPlantMetaDto

Descrizione: DTO che mappa i metadati di un impianto ricevuto dall'API esterna

4.1.5.6 PlantSeekResponseDto

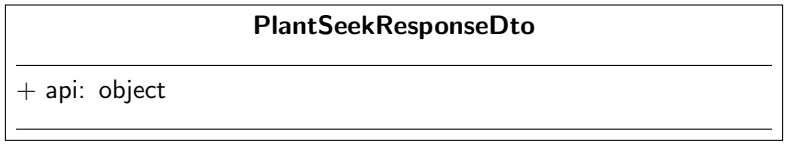

Figura 194: Diagramma della classe PlantSeekResponseDto

Descrizione: DTO di risposta per la ricerca di un impianto tramite ID

4.1.5.7 NotificationLinkDto

Figura 195: Diagramma della classe NotificationLinkDto

Descrizione: DTO che rappresenta un link incluso nel payload di una notifica esterna

4.1.5.8 HttpCacheController

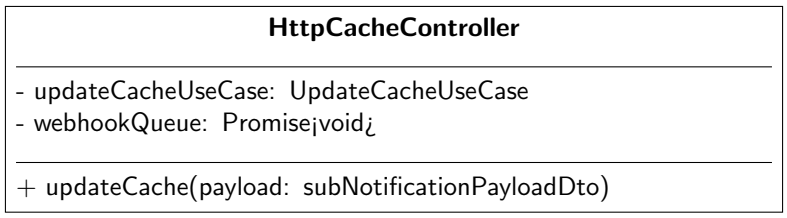

Figura 196: Diagramma della classe HttpCacheController

Descrizione: Controller HTTP che gestisce i webhook in ingresso e delega l'aggiornamento della cache

Descrizione dei metodi della classe:

- `updateCache(payload: subNotificationPayloadDto)`: Riceve il webhook e avvia l'aggiornamento della cache per gli impianti indicati

4.1.5.9 EventCacheController

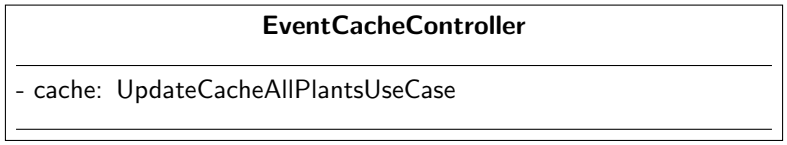

Figura 197: Diagramma della classe EventCacheController

Descrizione: Controller eventi che ascolta gli eventi interni e avvia l'aggiornamento della cache per tutti gli impianti

4.1.5.10 FetchNewCacheCmd

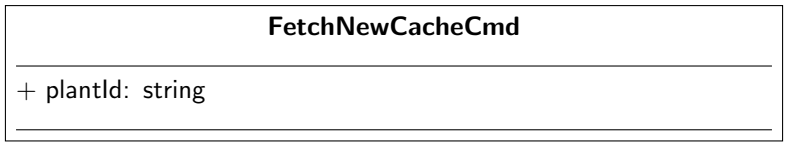

Figura 198: Diagramma della classe FetchNewCacheCmd

Descrizione: Comando che trasporta token e plantId per recuperare la struttura aggiornata di un impianto

4.1.5.11 GetValidCacheCmd

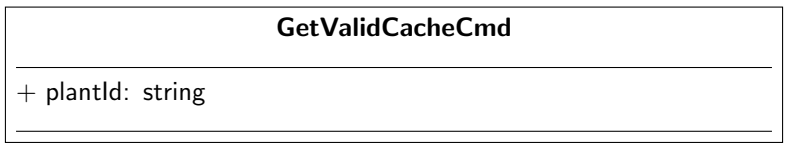

Figura 199: Diagramma della classe GetValidCacheCmd

Descrizione: Comando che trasporta gli ID degli impianti per cui aggiornare la cache

4.1.5.12 UpdateCacheUseCase

UpdateCacheUseCase

+ updateCache(cmd: GetValidCacheCmd) : Promise<boolean>

Figura 200: Diagramma dell'interfaccia UpdateCacheUseCase

Descrizione: Porta in ingresso per aggiornare la cache di uno o più impianti specifici

Descrizione dei metodi dell'interfaccia:

- `updateCache(cmd: GetValidCacheCmd) : Promise<boolean>`: Aggiorna la cache per gli impianti specificati nel comando

4.1.5.13 UpdateCacheAllPlantsUseCase

UpdateCacheAllPlantsUseCase

+ updateAllCache() : Promise<boolean>

Figura 201: Diagramma dell'interfaccia UpdateCacheAllPlantsUseCase

Descrizione: Porta in ingresso per aggiornare la cache di tutti gli impianti

Descrizione dei metodi dell'interfaccia:

- `updateAllCache() : Promise<boolean>`: Avvia l'aggiornamento della cache per tutti gli impianti disponibili

4.1.5.14 SyncCacheService

SyncCacheService

- fetchCachePort: FetchNewCachePort
- writeStructurePort: WriteCachePort
- getAllPlantIdsPort: GetAllPlantIdsPort
- emitter: EventEmitter2

+ updateCache(cmd: GetValidCacheCmd) : Promise<boolean>
+ updateAllCache() : Promise<boolean>

Figura 202: Diagramma della classe SyncCacheService

Descrizione: Servizio applicativo che orchestra il recupero e la persistenza della struttura degli impianti in cache

Descrizione dei metodi della classe:

- `updateCache(cmd: GetValidCacheCmd) : Promise<boolean>`: Recupera e persiste la struttura degli impianti indicati nel comando
- `updateAllCache() : Promise<boolean>`: Recupera tutti gli ID impianto e aggiorna la cache per ciascuno

4.1.5.15 FetchNewCachePort

FetchNewCachePort

+ fetch(cmd: FetchNewCacheCmd) : Promise<Plant>

Figura 203: Diagramma dell'interfaccia FetchNewCachePort

Descrizione: Porta di uscita per recuperare la struttura aggiornata di un impianto dall'esterno

Descrizione dei metodi dell'interfaccia:

- `fetch(cmd: FetchNewCacheCmd) : Promise<Plant>`: Recupera la struttura aggiornata di un impianto dato il comando

4.1.5.16 GetAllPlantIdsPort

GetAllPlantIdsPort

+ getAllPlantIds() : Promise<string[]>

Figura 204: Diagramma dell'interfaccia GetAllPlantIdsPort

Descrizione: Porta di uscita per ottenere la lista completa degli ID impianto

Descrizione dei metodi dell'interfaccia:

- `getAllPlantIds() : Promise<string[]>`: Restituisce la lista di tutti gli ID impianto disponibili

4.1.5.17 WriteCachePort

WriteCachePort

+ writeStructure(plant: Plant) : Promise<boolean>

Figura 205: Diagramma dell'interfaccia WriteCachePort

Descrizione: Porta di uscita per persistere la struttura di un impianto in cache

Descrizione dei metodi dell'interfaccia:

- `writeStructure(plant: Plant) : Promise<boolean>`: Persiste la struttura di un impianto nella cache

4.1.5.18 RoomEntity

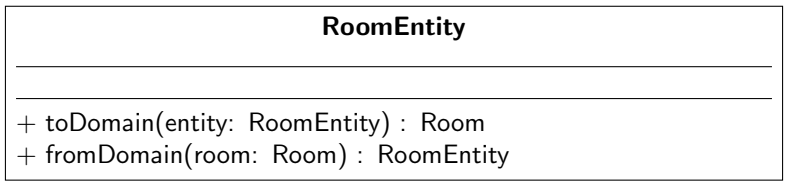

Figura 206: Diagramma della classe RoomEntity

Descrizione: Entità di persistenza di una stanza, con metodi di conversione da/verso il modello di dominio

Descrizione dei metodi della classe:

- `toDomain(entity: RoomEntity) : Room`: Converte l'entità persistita nel corrispondente oggetto di dominio
- `fromDomain(room: Room) : RoomEntity`: Converte l'oggetto di dominio nella corrispondente entità persistita

4.1.5.19 FetchNewCacheAdapter

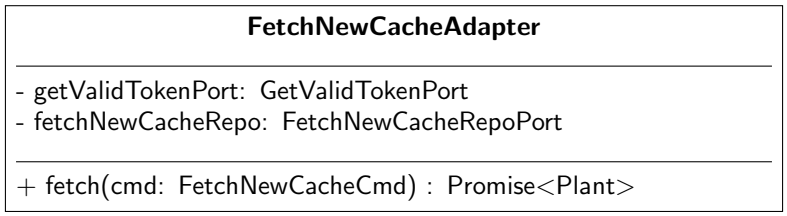

Figura 207: Diagramma della classe FetchNewCacheAdapter

Descrizione: Adapter che implementa FetchNewCachePort coordinando autenticazione e recupero dati

Descrizione dei metodi della classe:

- `fetch(cmd: FetchNewCacheCmd) : Promise<Plant>`: Ottiene un token valido e delega il fetch della struttura al repository

4.1.5.20 GetAllPlantIdsAdapter

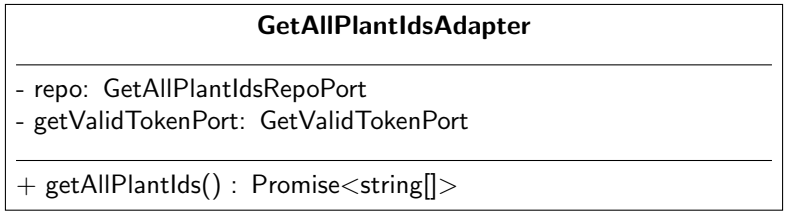

Figura 208: Diagramma della classe GetAllPlantIdsAdapter

Descrizione: Adapter che implementa GetAllPlantIdsPort coordinando autenticazione e recupero degli ID

Descrizione dei metodi della classe:

- `getAllPlantIds() : Promise<string[]>`: Ottiene un token valido e delega la lettura degli ID al repository

4.1.5.21 WriteCacheAdapter

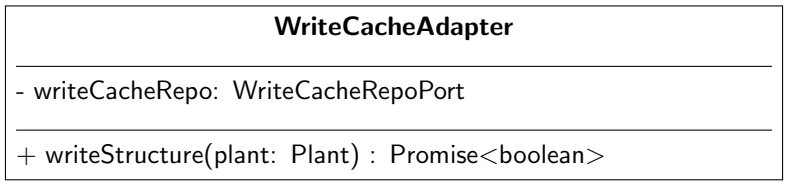

Figura 209: Diagramma della classe WriteCacheAdapter

Descrizione: Adapter che implementa WriteCachePort delegando la scrittura al repository di persistenza

Descrizione dei metodi della classe:

- `writeStructure(plant: Plant) : Promise<boolean>`: Delega la persistenza della struttura dell'impianto al repository

4.1.5.22 FetchNewCacheRepoPort

Figura 210: Diagramma dell'interfaccia FetchNewCacheRepoPort

Descrizione: Contratto del repository HTTP per recuperare la struttura di un impianto

Descrizione dei metodi dell'interfaccia:

- `fetch(validToken: string, plantId: string) : Promise<PlantDto | null>`: Recupera la struttura di un impianto dall'API esterna dato token e plantId

4.1.5.23 GetAllPlantIdsRepoPort

Figura 211: Diagramma dell'interfaccia GetAllPlantIdsRepoPort

Descrizione: Contratto del repository HTTP per ottenere tutti gli ID impianto

Descrizione dei metodi dell'interfaccia:

- `getAllPlantIds(validToken: string) : Promise<string[]>`: Recupera la lista di tutti gli ID impianto dall'API esterna dato un token valido

4.1.5.24 WriteCacheRepoPort

WriteCacheRepoPort

+ write(plant: PlantEntity) : Promise<boolean>

Figura 212: Diagramma dell'interfaccia WriteCacheRepoPort

Descrizione: Contratto del repository di persistenza per salvare la struttura di un impianto

Descrizione dei metodi dell'interfaccia:

- `write(plant: PlantEntity) : Promise<boolean>`: Persiste una PlantEntity nella cache

4.1.5.25 FetchStructureCacheImpl

FetchStructureCacheImpl

- httpService: HttpService

+ fetch(validToken: string, plantId: string) : Promise<PlantDto | null>
+ getAllPlantIds(validToken: string) : Promise<string[]>
- fetchRoom(validToken: string, plantId: string, room: ApiRoomDto) : Promise<RoomDto>
- fetchDevice(validToken: string, plantId: string, device: any) : Promise<DeviceDto>
- fetchDatapoint(datapoint: ApiDatapointDto) : DatapointDto

Figura 213: Diagramma della classe FetchStructureCacheImpl

Descrizione: Implementazione HTTP che recupera la struttura completa di un impianto componendo chiamate per stanze, dispositivi e datapoint

Descrizione dei metodi della classe:

- `fetch(validToken: string, plantId: string) : Promise<PlantDto | null>`: Recupera la struttura completa di un impianto orchestrando le chiamate a stanze, dispositivi e datapoint
- `getAllPlantIds(validToken: string) : Promise<string[]>`: Recupera la lista di tutti gli ID impianto dall'API esterna
- `fetchRoom(validToken: string, plantId: string, room: ApiRoomDto) : Promise<RoomDto>`: Recupera i dettagli di una singola stanza e i suoi dispositivi

- `fetchDevice(validToken: string, plantId: string, device: any) : Promise<DeviceDto>`: Recupera i dettagli di un singolo dispositivo e i suoi datapoint
- `fetchDatapoint(datapoint: ApiDatapointDto) : DatapointDto`: Mappa un datapoint grezzo dell'API nel DTO interno

4.1.5.26 StructureCacheImpl

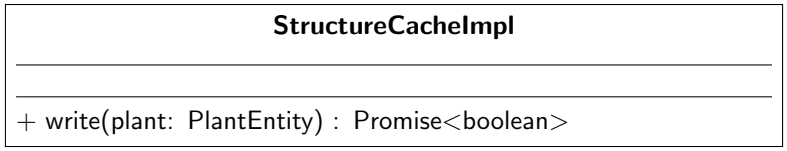

Figura 214: Diagramma della classe StructureCacheImpl

Descrizione: Implementazione del repository che persiste la struttura di un impianto nella cache

Descrizione dei metodi della classe:

- `write(plant: PlantEntity) : Promise<boolean>`: Salva la struttura dell'impianto nel sistema di persistenza della cache

4.1.6 Devices

4.1.6.1 WriteDatapointDto

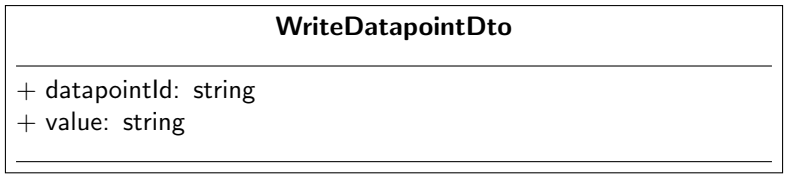

Figura 215: Diagramma della classe WriteDatapointDto

Descrizione: DTO per il payload di scrittura di un valore datapoint verso l'API esterna

4.1.6.2 DatapointDto

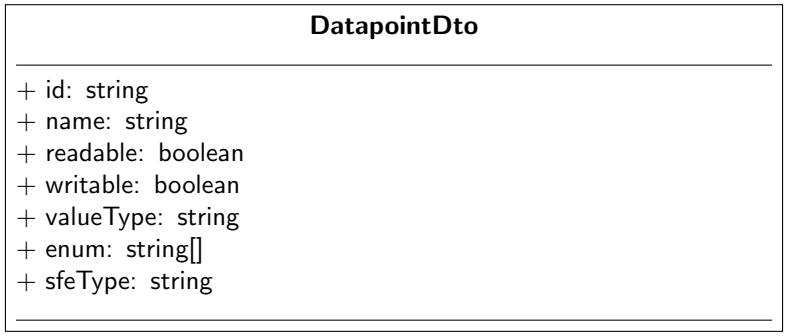

Figura 216: Diagramma della classe DatapointDto

Descrizione: DTO interno che rappresenta un datapoint normalizzato

4.1.6.3 DatapointValueDto

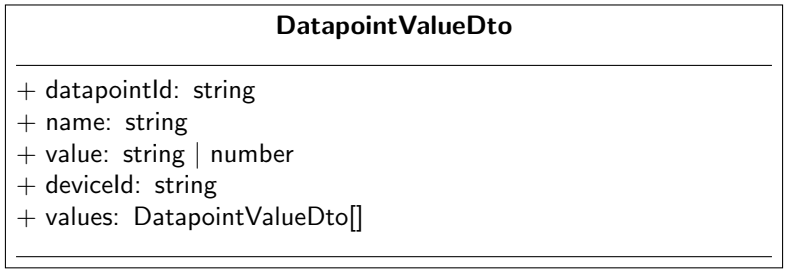

Figura 217: Diagramma della classe DatapointValueDto

Descrizione: DTO interno che rappresenta il valore corrente di un datapoint di un dispositivo

4.1.6.4 DeviceDto

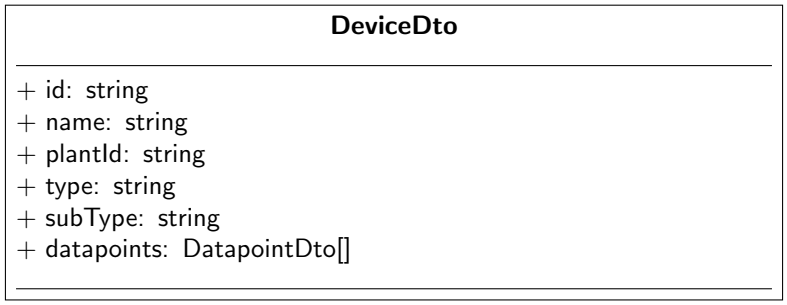

Figura 218: Diagramma della classe DeviceDto

Descrizione: DTO interno che rappresenta un dispositivo normalizzato con i suoi datapoint

4.1.6.5 DatapointValueAttributesDto

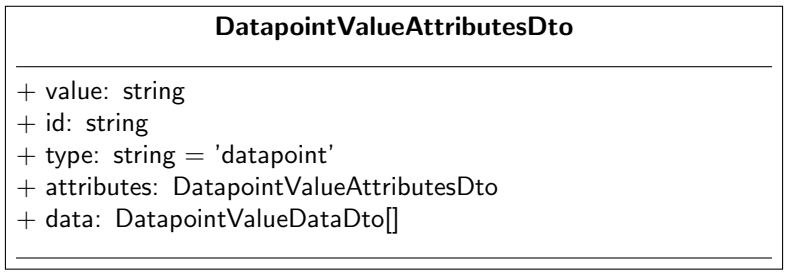

Figura 219: Diagramma della classe DatapointValueAttributesDto

Descrizione: DTO che struttura la richiesta di scrittura di un valore datapoint verso l'API esterna

4.1.6.6 Datapoint

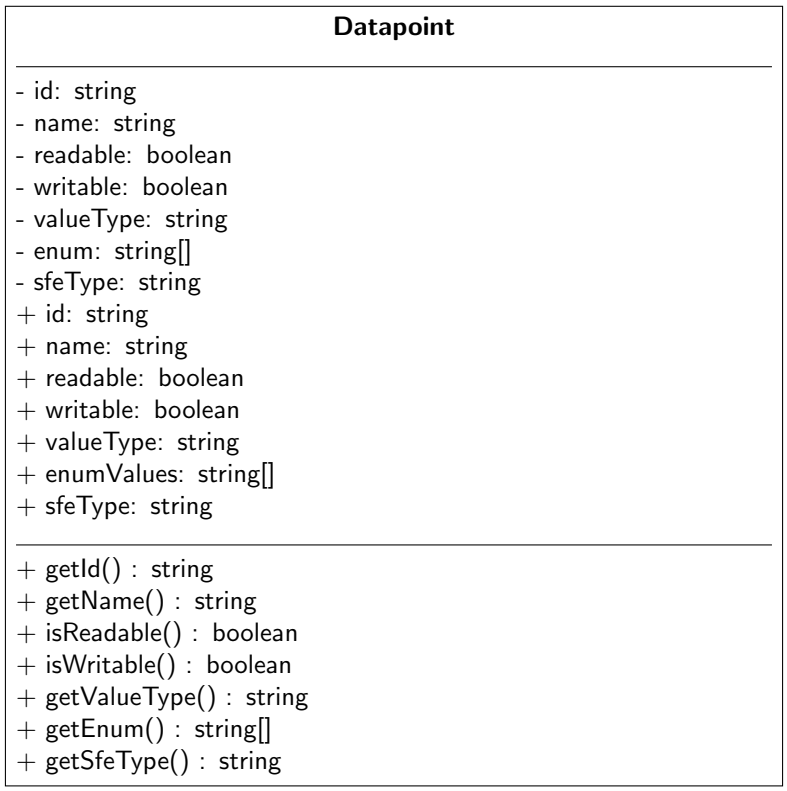

Figura 220: Diagramma della classe Datapoint

Descrizione: Oggetto di dominio che rappresenta un datapoint di un dispositivo

Descrizione dei metodi della classe:

- `getId() : string`: Restituisce l'ID del datapoint
- `getName() : string`: Restituisce il nome del datapoint
- `isReadable() : boolean`: Indica se il datapoint è leggibile
- `isWritable() : boolean`: Indica se il datapoint è scrivibile
- `getValueType() : string`: Restituisce il tipo di valore del datapoint
- `getEnum() : string[]`: Restituisce i valori enumerati ammessi
- `getSfeType() : string`: Restituisce il tipo SFE del datapoint

4.1.6.7 DeviceValue

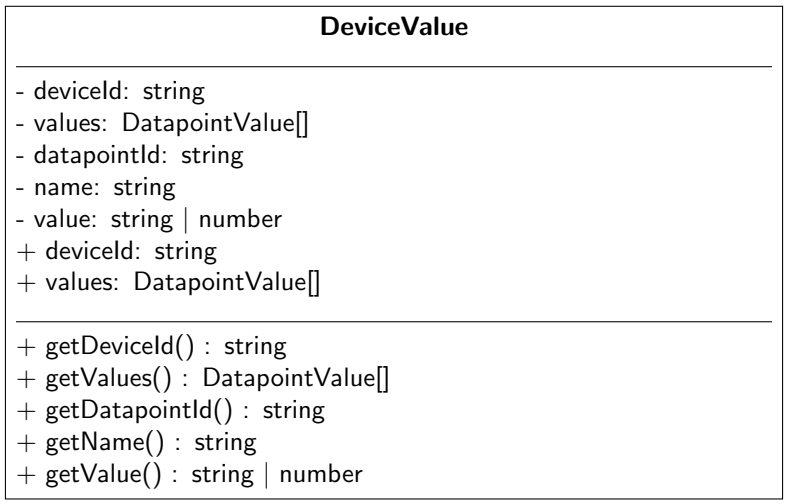

Figura 221: Diagramma della classe DeviceValue

Descrizione: Oggetto di dominio che rappresenta i valori correnti dei datapoint di un dispositivo

Descrizione dei metodi della classe:

- `getDeviceId() : string`: Restituisce l'ID del dispositivo
- `getValues() : DatapointValue[]`: Restituisce i valori di tutti i datapoint del dispositivo
- `getDatapointId() : string`: Restituisce l'ID del datapoint
- `getName() : string`: Restituisce il nome del datapoint
- `getValue() : string | number`: Restituisce il valore corrente del datapoint

4.1.6.8 Device

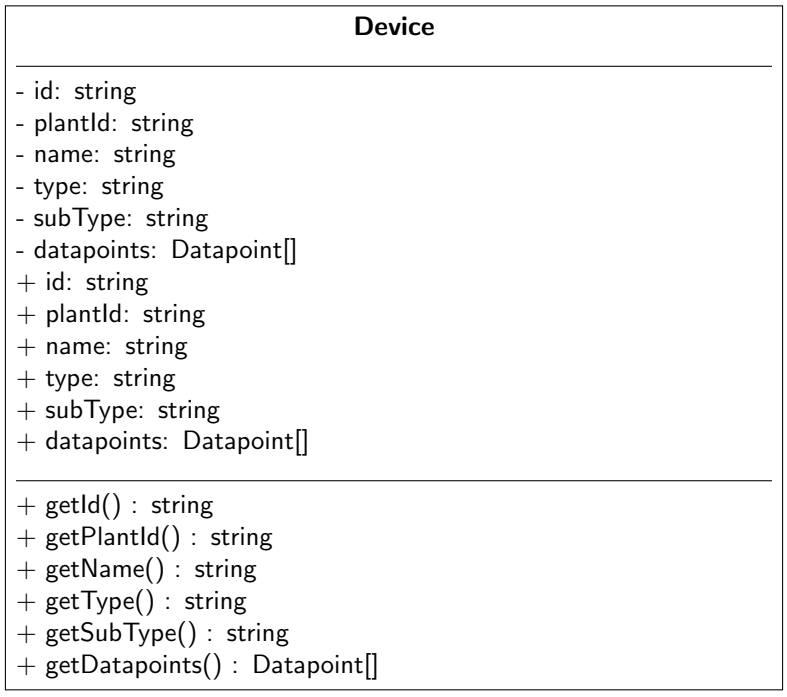

Figura 222: Diagramma della classe Device

Descrizione: Aggregato di dominio che rappresenta un dispositivo con i suoi datapoint

Descrizione dei metodi della classe:

- `getId() : string`: Restituisce l'ID del dispositivo
- `getPlantId() : string`: Restituisce l'ID dell'impianto
- `getName() : string`: Restituisce il nome del dispositivo
- `getType() : string`: Restituisce il tipo del dispositivo
- `getSubType() : string`: Restituisce il sotto-tipo del dispositivo
- `getDatapoints() : Datapoint[]`: Restituisce i datapoint associati al dispositivo

4.1.6.9 DeviceController

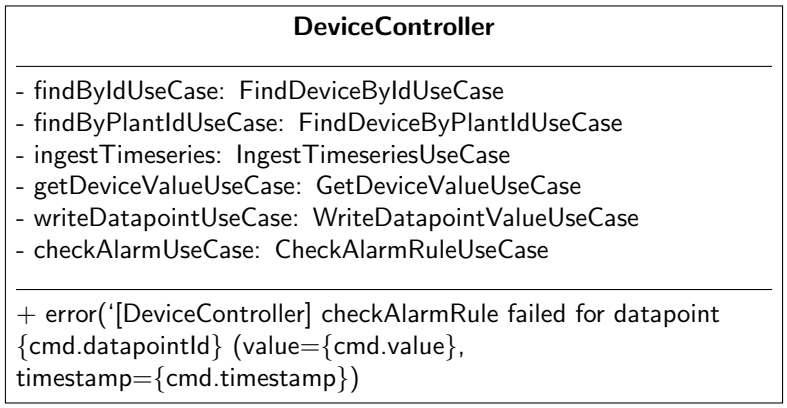

Figura 223: Diagramma della classe DeviceController

Descrizione: Controller eventi che riceve i comandi sui dispositivi e delega l'esecuzione ai relativi casi d'uso

Descrizione dei metodi della classe:

- `error('[DeviceController] checkAlarmRule failed for datapoint {cmd.datapointId} (value={cmd.value}, timestamp={cmd.timestamp})`: Gestisce e logga il fallimento della verifica delle regole di allarme

4.1.6.10 FindDeviceByDatapointIdCmd

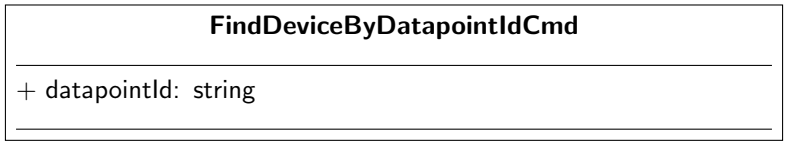

Figura 224: Diagramma della classe FindDeviceByDatapointIdCmd

Descrizione: Comando che trasporta il datapointId per cercare il dispositivo associato

4.1.6.11 FindDeviceByIdCmd

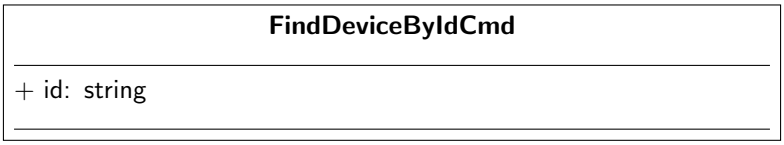

Figura 225: Diagramma della classe FindDeviceByIdCmd

Descrizione: Comando che trasporta l'ID per cercare un dispositivo specifico

4.1.6.12 FindDeviceByPlantIdCmd

Figura 226: Diagramma della classe FindDeviceByPlantIdCmd

Descrizione: Comando che trasporta il plantId per cercare tutti i dispositivi di un impianto

4.1.6.13 GetDeviceValueCmd

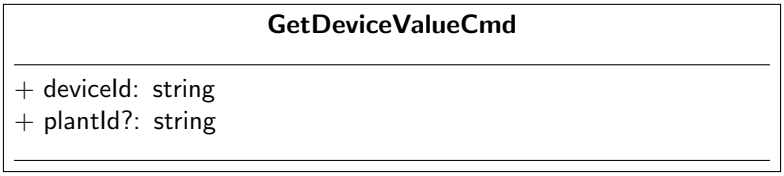

Figura 227: Diagramma della classe GetDeviceValueCmd

Descrizione: Comando che trasporta i dati per leggere il valore corrente di un dispositivo

4.1.6.14 IngestTimeseriesCmd

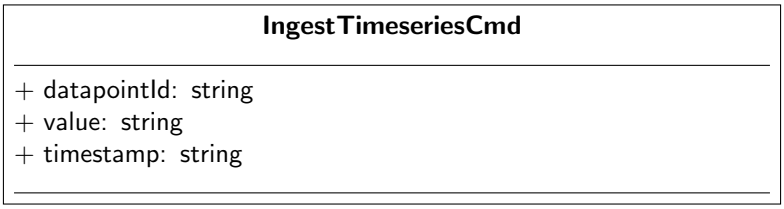

Figura 228: Diagramma della classe IngestTimeseriesCmd

Descrizione: Comando che trasporta datapointId, valore e timestamp per l'ingestione in serie storica

4.1.6.15 WriteDatapointValueCmd

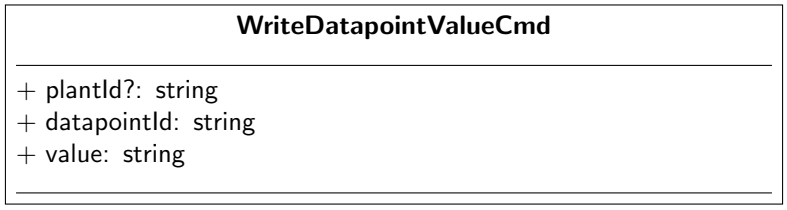

Figura 229: Diagramma della classe WriteDatapointValueCmd

Descrizione: Comando che trasporta i dati per scrivere un valore su un datapoint

4.1.6.16 FindDeviceByDatapointIdUsecase

FindDeviceByDatapointIdUsecase

+ findByDatapointId(cmd: FindDeviceByDatapointIdCmd) : Promise<Device>

Figura 230: Diagramma dell'interfaccia FindDeviceByDatapointIdUsecase

Descrizione: Porta in ingresso per cercare un dispositivo tramite ID del suo datapoint

Descrizione dei metodi dell'interfaccia:

- `findByDatapointId(cmd: FindDeviceByDatapointIdCmd) : Promise<Device>`: Cerca il dispositivo associato al datapointId specificato

4.1.6.17 FindDeviceByIdUseCase

FindDeviceByIdUseCase

+ findById(cmd: FindDeviceByIdCmd) : Promise<Device>

Figura 231: Diagramma dell'interfaccia FindDeviceByIdUseCase

Descrizione: Porta in ingresso per cercare un dispositivo tramite ID

Descrizione dei metodi dell'interfaccia:

- `findById(cmd: FindDeviceByIdCmd) : Promise<Device>`: Cerca un dispositivo tramite il suo ID univoco

4.1.6.18 FindDeviceByPlantIdUseCase

FindDeviceByPlantIdUseCase

+ findByPlantId(cmd: FindDeviceByPlantIdCmd) : Promise<Device[]>

Figura 232: Diagramma dell'interfaccia FindDeviceByPlantIdUseCase

Descrizione: Porta in ingresso per cercare tutti i dispositivi di un impianto

Descrizione dei metodi dell'interfaccia:

- `findByPlantId(cmd: FindDeviceByPlantIdCmd) : Promise<Device[]>`: Cerca tutti i dispositivi associati a un impianto

4.1.6.19 GetDeviceValueUseCase

GetDeviceValueUseCase

+ getDeviceValue(cmd: GetDeviceValueCmd) : Promise<DeviceValue>

Figura 233: Diagramma dell'interfaccia GetDeviceValueUseCase

Descrizione: Porta in ingresso per leggere i valori correnti di un dispositivo

Descrizione dei metodi dell'interfaccia:

- `getDeviceValue(cmd: GetDeviceValueCmd) : Promise<DeviceValue>`: Recupera il valore corrente di tutti i datapoint di un dispositivo

4.1.6.20 IngestTimeseriesUseCase

IngestTimeseriesUseCase

+ ingestTimeseries(cmd: IngestTimeseriesCmd) : Promise<void>

Figura 234: Diagramma dell'interfaccia IngestTimeseriesUseCase

Descrizione: Porta in ingresso per ingestare una misura in serie storica

Descrizione dei metodi dell'interfaccia:

- `ingestTimeseries(cmd: IngestTimeseriesCmd) : Promise<void>`: Ingerisce una misura nella serie storica del datapoint specificato

4.1.6.21 WriteDatapointValueUseCase

WriteDatapointValueUseCase

+ writeDatapointValue(cmd: WriteDatapointValueCmd) : Promise<void>

Figura 235: Diagramma dell'interfaccia WriteDatapointValueUseCase

Descrizione: Porta in ingresso per scrivere un valore su un datapoint

Descrizione dei metodi dell'interfaccia:

- `writeDatapointValue(cmd: WriteDatapointValueCmd) : Promise<void>`: Scrive un valore su un datapoint tramite l'API esterna

4.1.6.22 DeviceService

DeviceService

- findByIdPort: FindDeviceByIdPort
- findByPlantIdPort: FindDeviceByPlantIdPort
- ingestTimeseriesPort: IngestTimeseriesPort
- getDeviceValuePort: GetDeviceValuePort
- writeDatapointPort: WriteDatapointValuePort
- findByDatapointIdPort: FindDeviceByDatapointIdPort

+ findById(cmd: FindDeviceByIdCmd) : Promise<Device>
+ findByPlantId(cmd: FindDeviceByPlantIdCmd) : Promise<Device[]>
+ ingestTimeseries(cmd: IngestTimeseriesCmd) : Promise<void>
+ getDeviceValue(cmd: GetDeviceValueCmd) : Promise<DeviceValue>
+ writeDatapointValue(cmd: WriteDatapointValueCmd) : Promise<void>
+ findByDatapointId(cmd: FindDeviceByDatapointIdCmd) : Promise<Device>

Figura 236: Diagramma della classe DeviceService

Descrizione: Servizio applicativo che implementa i casi d'uso del modulo Device orchestrando le porte di uscita

Descrizione dei metodi della classe:

- `findById(cmd: FindDeviceByIdCmd) : Promise<Device>`: Delega la ricerca del dispositivo per ID alla porta corrispondente
- `findByPlantId(cmd: FindDeviceByPlantIdCmd) : Promise<Device[]>`: Delega la ricerca dei dispositivi per impianto alla porta corrispondente
- `ingestTimeseries(cmd: IngestTimeseriesCmd) : Promise<void>`: Delega l'ingestione della misura alla porta corrispondente

- `getDeviceValue(cmd: GetDeviceValueCmd) : Promise<DeviceValue>`: Delega la lettura del valore corrente del dispositivo alla porta corrispondente
- `writeDatapointValue(cmd: WriteDatapointValueCmd) : Promise<void>`: Delega la scrittura del valore sul datapoint alla porta corrispondente
- `findByDatapointId(cmd: FindDeviceByDatapointIdCmd) : Promise<Device>`: Delega la ricerca del dispositivo per datapointId alla porta corrispondente

4.1.6.23 FindDeviceByDatapointIdPort

FindDeviceByDatapointIdPort

+ findByDatapointId(cmd: FindDeviceByDatapointIdCmd) : Promise<Device>

Figura 237: Diagramma dell'interfaccia FindDeviceByDatapointIdPort

Descrizione: Porta di uscita per cercare un dispositivo tramite ID del suo datapoint

Descrizione dei metodi dell'interfaccia:

- `findByDatapointId(cmd: FindDeviceByDatapointIdCmd) : Promise<Device>`: Cerca il dispositivo associato al datapointId specificato

4.1.6.24 FindDeviceByIdPort

FindDeviceByIdPort

+ findById(cmd: FindDeviceByIdCmd) : Promise<Device>

Figura 238: Diagramma dell'interfaccia FindDeviceByIdPort

Descrizione: Porta di uscita per cercare un dispositivo per ID

Descrizione dei metodi dell'interfaccia:

- `findById(cmd: FindDeviceByIdCmd) : Promise<Device>`: Cerca un dispositivo tramite il suo ID univoco

4.1.6.25 FindDeviceByPlantIdPort

FindDeviceByPlantIdPort

+ findByPlantId(cmd: FindDeviceByPlantIdCmd) : Promise<Device[]>

Figura 239: Diagramma dell'interfaccia FindDeviceByPlantIdPort

Descrizione: Porta di uscita per cercare tutti i dispositivi di un impianto

Descrizione dei metodi dell'interfaccia:

- `findByPlantId(cmd: FindDeviceByPlantIdCmd) : Promise<Device[]>`: Cerca tutti i dispositivi associati a un impianto

4.1.6.26 GetDeviceValuePort

GetDeviceValuePort

+ getDeviceValue(cmd: GetDeviceValueCmd) : Promise<DeviceValue>

Figura 240: Diagramma dell'interfaccia GetDeviceValuePort

Descrizione: Porta di uscita per leggere i valori correnti di un dispositivo

Descrizione dei metodi dell'interfaccia:

- `getDeviceValue(cmd: GetDeviceValueCmd) : Promise<DeviceValue>`: Recupera i valori correnti di tutti i datapoint di un dispositivo

4.1.6.27 IngestTimeseriesPort

IngestTimeseriesPort

+ ingestTimeseries(cmd: IngestTimeseriesCmd) : Promise<void>

Figura 241: Diagramma dell'interfaccia IngestTimeseriesPort

Descrizione: Porta di uscita per ingestare una misura in serie storica

Descrizione dei metodi dell'interfaccia:

- `ingestTimeseries(cmd: IngestTimeseriesCmd) : Promise<void>`: Ingerisce una misura nella serie storica del datapoint specificato

4.1.6.28 WriteDatapointValuePort

WriteDatapointValuePort

+ writeDatapointValue(cmd: WriteDatapointValueCmd) : Promise<void>

Figura 242: Diagramma dell'interfaccia WriteDatapointValuePort

Descrizione: Porta di uscita per scrivere un valore su un datapoint

Descrizione dei metodi dell'interfaccia:

- `writeDatapointValue(cmd: WriteDatapointValueCmd) : Promise<void>`: Scrive un valore su un datapoint tramite l'API esterna

4.1.6.29 DatapointEntity

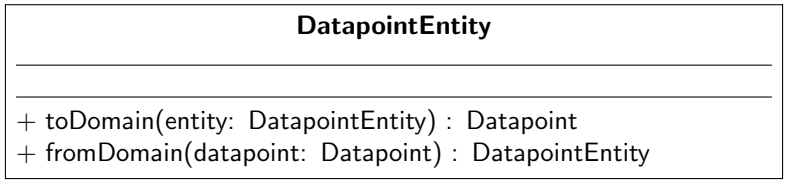

Figura 243: Diagramma della classe DatapointEntity

Descrizione: Entità di persistenza di un datapoint, con metodi di conversione da/verso il modello di dominio

Descrizione dei metodi della classe:

- `toDomain(entity: DatapointEntity) : Datapoint`: Converte l'entità persistita nel corrispondente oggetto di dominio
- `fromDomain(datapoint: Datapoint) : DatapointEntity`: Converte l'oggetto di dominio nella corrispondente entità persistita

4.1.6.30 DeviceEntity

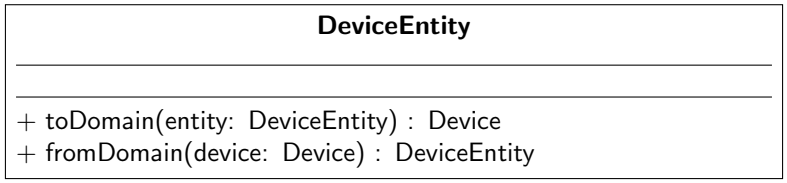

Figura 244: Diagramma della classe DeviceEntity

Descrizione: Entità di persistenza di un dispositivo, con metodi di conversione da/verso il modello di dominio

Descrizione dei metodi della classe:

- `toDomain(entity: DeviceEntity) : Device`: Converte l'entità persistita nel corrispondente oggetto di dominio
- `fromDomain(device: Device) : DeviceEntity`: Converte l'oggetto di dominio nella corrispondente entità persistita

4.1.6.31 FindDeviceByDatapointIdAdapter

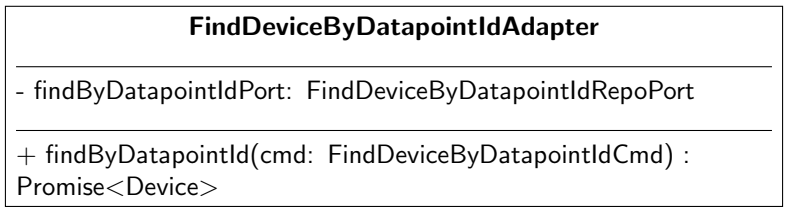

Figura 245: Diagramma della classe FindDeviceByDatapointIdAdapter

Descrizione: Adapter che implementa FindDeviceByDatapointIdPort delegando la ricerca al repository

Descrizione dei metodi della classe:

- `findByDatapointId(cmd: FindDeviceByDatapointIdCmd) : Promise<Device>`: Recupera l'entità dal repository e la converte nel modello di dominio

4.1.6.32 FindDeviceByIdAdapter

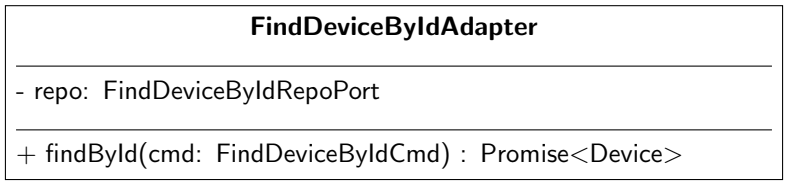

Figura 246: Diagramma della classe FindDeviceByIdAdapter

Descrizione: Adapter che implementa FindDeviceByIdPort delegando la ricerca al repository

Descrizione dei metodi della classe:

- `findById(cmd: FindDeviceByIdCmd) : Promise<Device>`: Recupera l'entità dal repository e la converte nel modello di dominio

4.1.6.33 FindDeviceByPlantIdAdapter

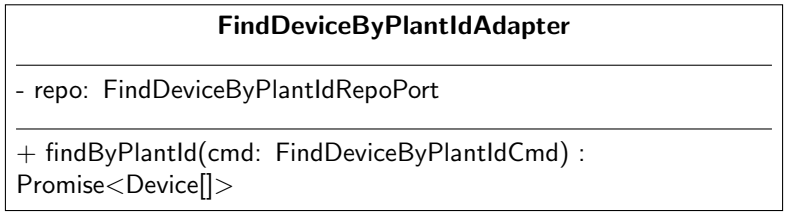

Figura 247: Diagramma della classe FindDeviceByPlantIdAdapter

Descrizione: Adapter che implementa FindDeviceByPlantIdPort delegando la ricerca al repository

Descrizione dei metodi della classe:

- `findByPlantId(cmd: FindDeviceByPlantIdCmd) : Promise<Device[]>`: Recupera le entità dal repository e le converte nei modelli di dominio

4.1.6.34 GetDeviceValueAdapter

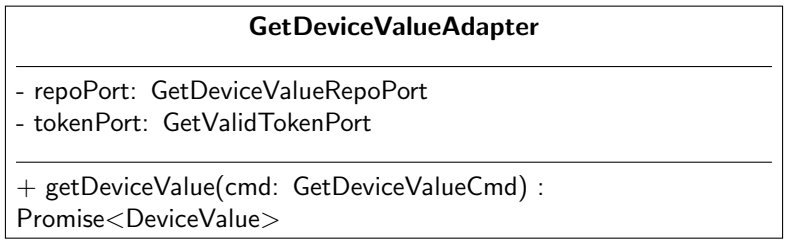

Figura 248: Diagramma della classe GetDeviceValueAdapter

Descrizione: Adapter che implementa GetDeviceValuePort coordinando autenticazione e lettura dei valori

Descrizione dei metodi della classe:

- `getDeviceValue(cmd: GetDeviceValueCmd) : Promise<DeviceValue>`: Ottiene un token valido e delega la lettura dei valori al repository

4.1.6.35 IngestTimeseriesAdapter

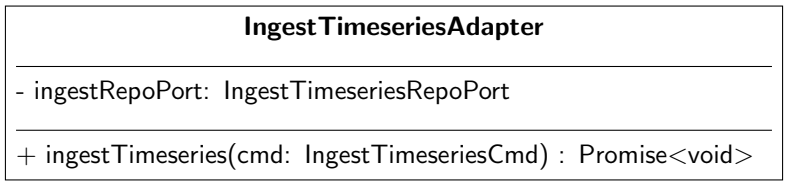

Figura 249: Diagramma della classe IngestTimeseriesAdapter

Descrizione: Adapter che implementa IngestTimeseriesPort delegando l'ingestione al repository

Descrizione dei metodi della classe:

- `ingestTimeseries(cmd: IngestTimeseriesCmd) : Promise<void>`: Delega l'ingestione della misura al repository

4.1.6.36 WriteDatapointValueAdapter

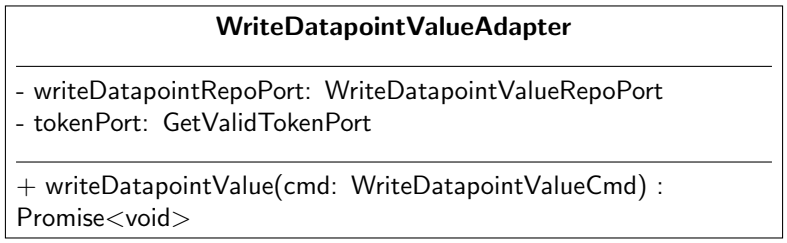

Figura 250: Diagramma della classe WriteDatapointValueAdapter

Descrizione: Adapter che implementa WriteDatapointValuePort coordinando autenticazione e scrittura del valore

Descrizione dei metodi della classe:

- `writeDatapointValue(cmd: WriteDatapointValueCmd) : Promise<void>`: Ottiene un token valido e delega la scrittura del valore al repository

4.1.6.37 FindDeviceByDatapointIdRepoPort

FindDeviceByDatapointIdRepoPort

+ findByDatapointId(datapointId: string) : Promise<DeviceEntity | null>

Figura 251: Diagramma dell'interfaccia FindDeviceByDatapointIdRepoPort

Descrizione: Contratto del repository di persistenza per cercare un dispositivo tramite datapointId

Descrizione dei metodi dell'interfaccia:

- `findByDatapointId(datapointId: string) : Promise<DeviceEntity | null>`: Cerca nel database il dispositivo associato al datapointId specificato

4.1.6.38 FindDeviceByIdRepoPort

FindDeviceByIdRepoPort

+ findById(id: string) : Promise<DeviceEntity | null>

Figura 252: Diagramma dell'interfaccia FindDeviceByIdRepoPort

Descrizione: Contratto del repository di persistenza per cercare un dispositivo per ID

Descrizione dei metodi dell'interfaccia:

- `findById(id: string) : Promise<DeviceEntity | null>`: Cerca nel database un dispositivo tramite il suo ID univoco

4.1.6.39 FindDeviceByPlantIdRepoPort

FindDeviceByPlantIdRepoPort

+ findByPlantId(plantId: string) : Promise<DeviceEntity[] | null>

Figura 253: Diagramma dell'interfaccia FindDeviceByPlantIdRepoPort

Descrizione: Contratto del repository di persistenza per cercare i dispositivi di un impianto

Descrizione dei metodi dell'interfaccia:

- `findByPlantId(plantId: string) : Promise<DeviceEntity[] | null>`: Cerca nel database tutti i dispositivi associati a un impianto

4.1.6.40 GetDeviceValueRepoPort

GetDeviceValueRepoPort

+ getDeviceValue(validToken: string, plantId: string, deviceId: string) : Promise<DatapointExtractedDto[]>

Figura 254: Diagramma dell'interfaccia GetDeviceValueRepoPort

Descrizione: Contratto del repository HTTP per leggere i valori correnti di un dispositivo

Descrizione dei metodi dell'interfaccia:

- `getDeviceValue(validToken: string, plantId: string, deviceId: string) : Promise<DatapointExtractedDto[]>`: Legge i valori correnti dei datapoint di un dispositivo tramite API esterna

4.1.6.41 IngestTimeseriesRepoPort

IngestTimeseriesRepoPort

+ ingestTimeseries(datapointId: string, value: string, timestamp: string) : Promise<boolean>

Figura 255: Diagramma dell'interfaccia IngestTimeseriesRepoPort

Descrizione: Contratto del repository di persistenza per l'ingestione di misure in serie storica

Descrizione dei metodi dell'interfaccia:

- `ingestTimeseries(datapointId: string, value: string, timestamp: string) : Promise<boolean>`: Ingerisce una misura nella serie storica del datapoint specificato

4.1.6.42 WriteDatapointValueRepoPort

WriteDatapointValueRepoPort

+ writeDeviceValue(validToken: string, plantId: string, datapointId: string, value: string) : Promise<boolean>

Figura 256: Diagramma dell'interfaccia WriteDatapointValueRepoPort

Descrizione: Contratto del repository HTTP per scrivere un valore su un datapoint

Descrizione dei metodi dell'interfaccia:

- `writeDeviceValue(validToken: string, plantId: string, datapointId: string, value: string) : Promise<boolean>`: Scrive un valore su un datapoint tramite API esterna

4.1.6.43 DeviceApiImpl

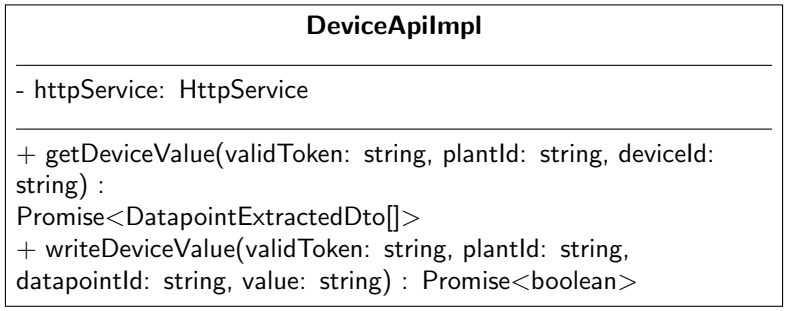

Figura 257: Diagramma della classe DeviceApiImpl

Descrizione: Implementazione HTTP dei repository che accedono all'API esterna per lettura e scrittura dei valori dei dispositivi

Descrizione dei metodi della classe:

- `getDeviceValue(validToken: string, plantId: string, deviceId: string) : Promise<DatapointExtractedDto[]>`: Chiama l'API esterna per leggere i valori correnti dei datapoint di un dispositivo
- `writeDeviceValue(validToken: string, plantId: string, datapointId: string, value: string) : Promise<boolean>`: Chiama l'API esterna per scrivere un valore su un datapoint

4.1.6.44 DeviceRepositoryImpl

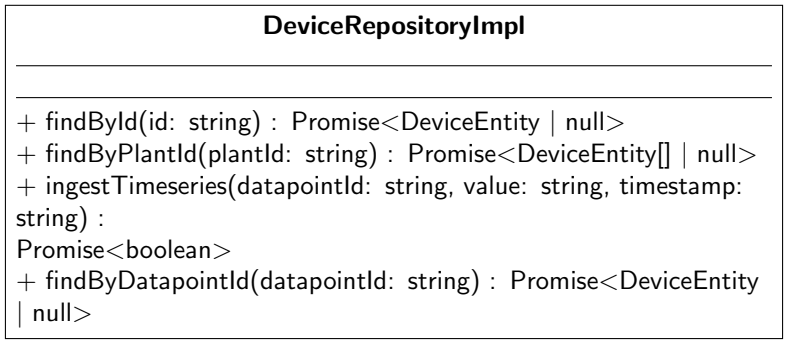

Figura 258: Diagramma della classe DeviceRepositoryImpl

Descrizione: Implementazione del repository di persistenza per le operazioni di lettura e ingestione sui dispositivi

Descrizione dei metodi della classe:

- `findById(id: string) : Promise<DeviceEntity | null>`: Cerca nel database un dispositivo tramite il suo ID univoco
- `findByPlantId(plantId: string) : Promise<DeviceEntity[] | null>`: Cerca nel database tutti i dispositivi associati a un impianto

- `ingestTimeseries(datapointId: string, value: string, timestamp: string) : Promise<boolean>`: Persiste una misura nella serie storica del datapoint specificato
- `findByDatapointId(datapointId: string) : Promise<DeviceEntity | null>`: Cerca nel database il dispositivo associato al datapointId specificato

4.1.7 Notifications

4.1.7.1 Notification

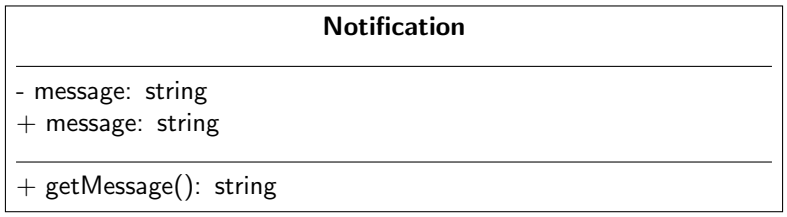

Figura 259: Diagramma della classe Notification

Descrizione: Oggetto di dominio che rappresenta una notifica

Descrizione dei metodi della classe:

- `getMessage(): string`: Restituisce il messaggio della notifica

4.1.7.2 EventNotificationController

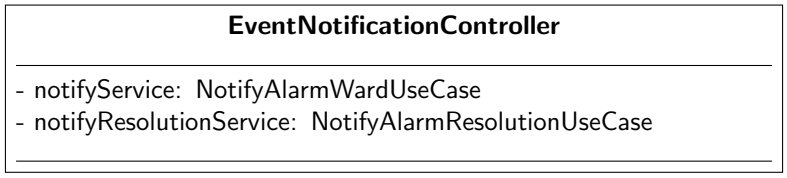

Figura 260: Diagramma della classe EventNotificationController

Descrizione: Controller eventi che ascolta gli eventi interni e delega l'invio delle notifiche ai relativi casi d'uso

4.1.7.3 NotifyAlarmResolutionCmd

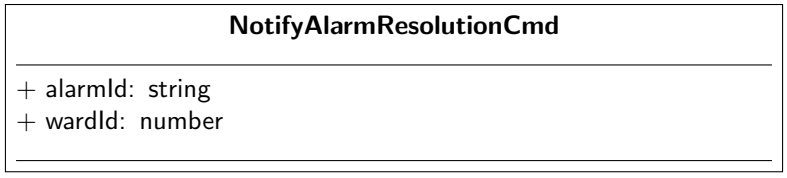

Figura 261: Diagramma della classe NotifyAlarmResolutionCmd

Descrizione: Comando che trasporta alarmId e wardId per notificare la risoluzione di un allarme

4.1.7.4 NotifyAlarmWardCmd

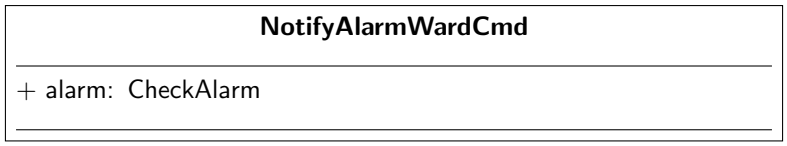

Figura 262: Diagramma della classe NotifyAlarmWardCmd

Descrizione: Comando che trasporta wardId e i dati dell'allarme per notificare un reparto

4.1.7.5 WriteNotificationCmd

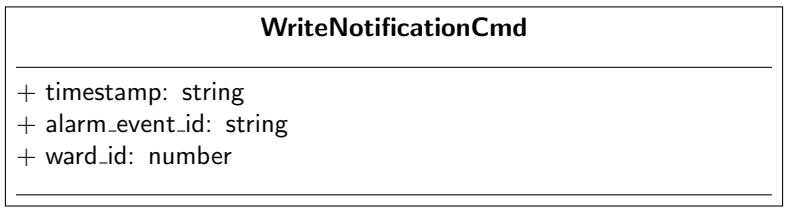

Figura 263: Diagramma della classe WriteNotificationCmd

Descrizione: Comando che trasporta i dati per persistere una notifica

4.1.7.6 NotifyAlarmWardUseCase

NotifyAlarmWardUseCase

`+ notifyAlarmWard(cmd: NotifyAlarmWardCmd): Promise<void>`

Figura 264: Diagramma dell'interfaccia NotifyAlarmWardUseCase

Descrizione: Porta in ingresso per notificare un allarme attivo a un reparto

Descrizione dei metodi dell'interfaccia:

- `notifyAlarmWard(cmd: NotifyAlarmWardCmd): Promise<void>`: Invia la notifica di un allarme attivo al reparto indicato

4.1.7.7 NotifyAlarmResolutionUseCase

NotifyAlarmResolutionUseCase

`+ notifyAlarmResolution(cmd: NotifyAlarmResolutionCmd): Promise<void>`

Figura 265: Diagramma dell'interfaccia NotifyAlarmResolutionUseCase

Descrizione: Porta in ingresso per notificare la risoluzione di un allarme

Descrizione dei metodi dell'interfaccia:

- `notifyAlarmResolution(cmd: NotifyAlarmResolutionCmd): Promise<void>`: Invia la notifica di risoluzione di un allarme al reparto indicato

4.1.7.8 NotificationsService

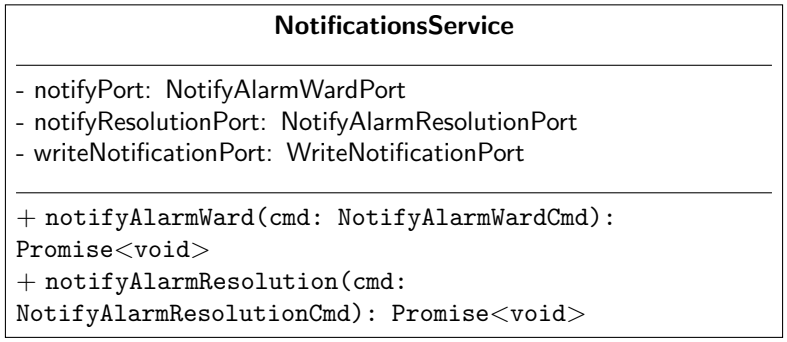

Figura 266: Diagramma della classe NotificationsService

Descrizione: Servizio applicativo che orchestra l'invio e la persistenza delle notifiche di allarme

Descrizione dei metodi della classe:

- `notifyAlarmWard(cmd: NotifyAlarmWardCmd): Promise<void>`: Invia la notifica dell'allarme e persiste la notifica
- `notifyAlarmResolution(cmd: NotifyAlarmResolutionCmd): Promise<void>`: Invia la notifica di risoluzione dell'allarme

4.1.7.9 NotifyAlarmWardPort

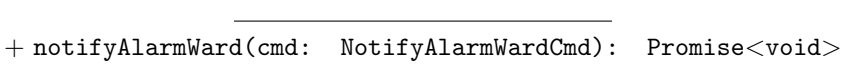

Figura 267: Diagramma dell'interfaccia NotifyAlarmWardPort

Descrizione: Porta di uscita per inviare la notifica di un allarme a un reparto

Descrizione dei metodi dell'interfaccia:

- `notifyAlarmWard(cmd: NotifyAlarmWardCmd): Promise<void>`: Invia la notifica di un allarme attivo al reparto indicato

4.1.7.10 NotifyAlarmResolutionPort

NotifyAlarmResolutionPort

+ `notifyAlarmResolution(cmd: NotifyAlarmResolutionCmd): Promise<void>`

Figura 268: Diagramma dell'interfaccia NotifyAlarmResolutionPort

Descrizione: Porta di uscita per inviare la notifica di risoluzione di un allarme

Descrizione dei metodi dell'interfaccia:

- `notifyAlarmResolution(cmd: NotifyAlarmResolutionCmd): Promise<void>`: Invia la notifica di risoluzione di un allarme al reparto indicato

4.1.7.11 WriteNotificationPort

WriteNotificationPort

+ `writeNotification(cmd: WriteNotificationCmd): Promise<boolean>`

Figura 269: Diagramma dell'interfaccia WriteNotificationPort

Descrizione: Porta di uscita per persistere una notifica

Descrizione dei metodi dell'interfaccia:

- `writeNotification(cmd: WriteNotificationCmd): Promise<boolean>`: Persiste una notifica nel sistema di storage

4.1.7.12 NotifyAlarmWardAdapter

NotifyAlarmWardAdapter

- notifyRepo: NotifyAlarmWardRepoPort

+ `notifyAlarmWard(cmd: NotifyAlarmWardCmd): Promise<void>`

Figura 270: Diagramma della classe NotifyAlarmWardAdapter

Descrizione: Adapter che implementa NotifyAlarmWardPort delegando l'invio al repository WebSocket

Descrizione dei metodi della classe:

- `notifyAlarmWard(cmd: NotifyAlarmWardCmd): Promise<void>`: Delega l'invio della notifica dell'allarme al repository

4.1.7.13 NotifyAlarmResolutionAdapter

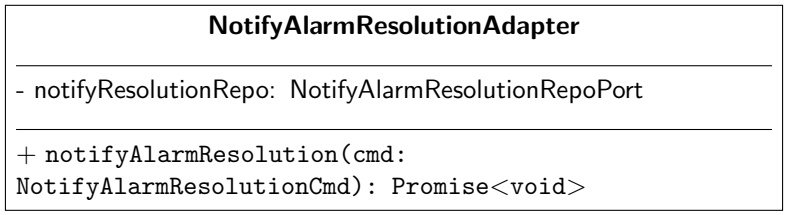

Figura 271: Diagramma della classe NotifyAlarmResolutionAdapter

Descrizione: Adapter che implementa NotifyAlarmResolutionPort delegando l'invio al repository WebSocket

Descrizione dei metodi della classe:

- `notifyAlarmResolution(cmd: NotifyAlarmResolutionCmd): Promise<void>`: Delega l'invio della notifica di risoluzione al repository

4.1.7.14 WriteNotificationAdapter

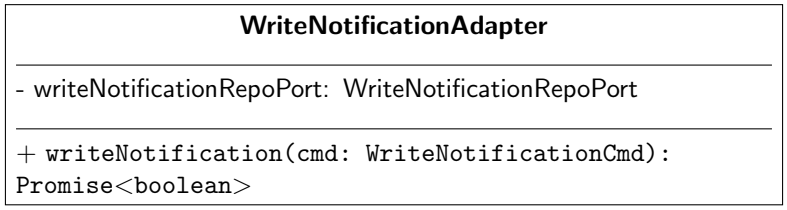

Figura 272: Diagramma della classe WriteNotificationAdapter

Descrizione: Adapter che implementa WriteNotificationPort delegando la scrittura al repository di persistenza

Descrizione dei metodi della classe:

- `writeNotification(cmd: WriteNotificationCmd): Promise<boolean>`: Delega la persistenza della notifica al repository

4.1.7.15 NotifyAlarmWardRepoPort

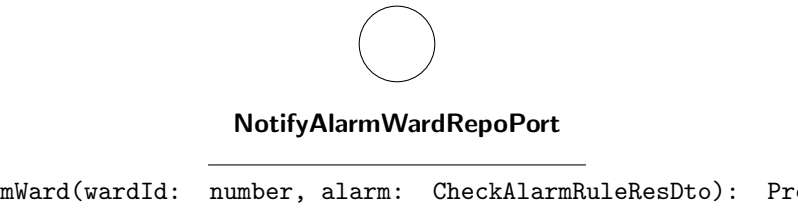

Figura 273: Diagramma dell'interfaccia NotifyAlarmWardRepoPort

Descrizione: Contratto del repository WebSocket per notificare un allarme attivo a un reparto

Descrizione dei metodi dell'interfaccia:

- `notifyAlarmWard(wardId: number, alarm: CheckAlarmRuleResDto): Promise<void>`: Invia la notifica di un allarme al reparto tramite WebSocket

4.1.7.16 NotifyAlarmResolutionRepoPort

NotifyAlarmResolutionRepoPort

+ `notifyAlarmResolution(alarmId: string, wardId: number): Promise<void>`

Figura 274: Diagramma dell'interfaccia NotifyAlarmResolutionRepoPort

Descrizione: Contratto del repository WebSocket per notificare la risoluzione di un allarme

Descrizione dei metodi dell'interfaccia:

- `notifyAlarmResolution(alarmId: string, wardId: number): Promise<void>`: Invia la notifica di risoluzione di un allarme al reparto tramite WebSocket

4.1.7.17 WriteNotificationRepoPort

WriteNotificationRepoPort

+ `writeNotification(ward_id: number, alarm_id: string, timestamp: string): Promise<boolean>`

Figura 275: Diagramma dell'interfaccia WriteNotificationRepoPort

Descrizione: Contratto del repository di persistenza per salvare una notifica

Descrizione dei metodi dell'interfaccia:

- `writeNotification(ward_id: number, alarm_id: string, timestamp: string): Promise<boolean>`: Persiste una notifica nel database con wardId, alarmId e timestamp

4.1.7.18 NotificationsGateway

NotificationsGateway

+ `notifyAlarmWard(wardId: number, alarm: CheckAlarmRuleResDto): Promise<void>`
+ `notifyAlarmResolution(alarmId: string, wardId: number): Promise<void>`

Figura 276: Diagramma della classe NotificationsGateway

Descrizione: Implementazione WebSocket che invia le notifiche di allarme e risoluzione ai reparti connessi

Descrizione dei metodi della classe:

- `notifyAlarmWard(wardId: number, alarm: CheckAlarmRuleResDto): Promise<void>`: Emette l'evento di allarme attivo al reparto tramite WebSocket
- `notifyAlarmResolution(alarmId: string, wardId: number): Promise<void>`: Emette l'evento di risoluzione dell'allarme al reparto tramite WebSocket

4.1.7.19 NotificationRepositoryImpl

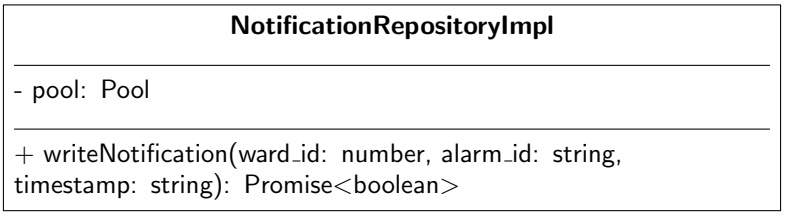

Figura 277: Diagramma della classe NotificationRepositoryImpl

Descrizione: Implementazione del repository di persistenza per il salvataggio delle notifiche nel database

Descrizione dei metodi della classe:

- `writeNotification(ward_id: number, alarm_id: string, timestamp: string): Promise<boolean>`: Persiste una notifica nel database con wardId, alarmId e timestamp

4.1.8 Plants

4.1.8.1 PlantDto

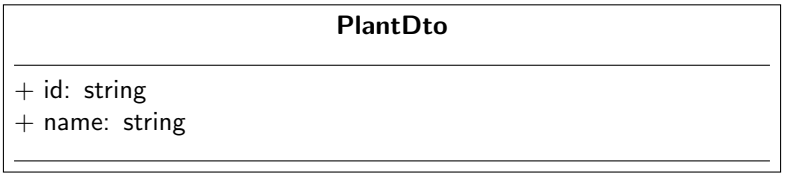

Figura 278: Diagramma della classe PlantDto

Descrizione: DTO interno che rappresenta un impianto normalizzato

4.1.8.2 RoomDto

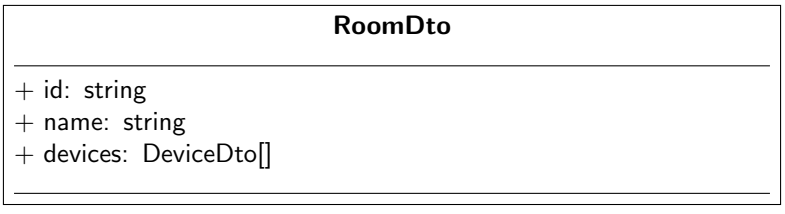

Figura 279: Diagramma della classe RoomDto

Descrizione: DTO interno che rappresenta una stanza normalizzata con i suoi dispositivi

4.1.8.3 Plant

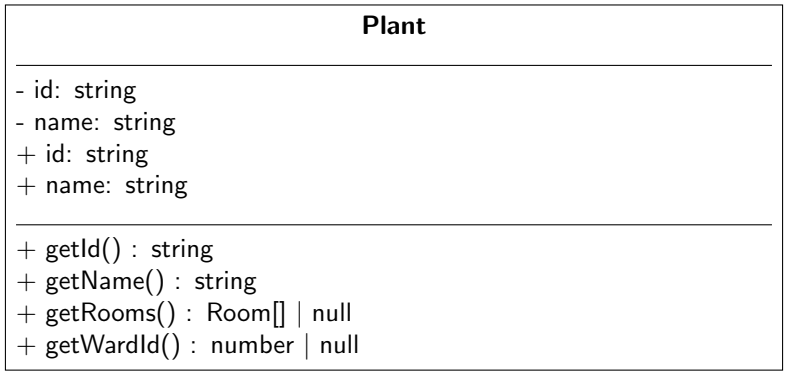

Figura 280: Diagramma della classe Plant

Descrizione: Aggregato di dominio che rappresenta un impianto con le sue stanze e il reparto associato

Descrizione dei metodi della classe:

- `getId() : string`: Restituisce l'ID dell'impianto
- `getName() : string`: Restituisce il nome dell'impianto
- `getRooms() : Room[] | null`: Restituisce le stanze dell'impianto, se presenti
- `getWardId() : number | null`: Restituisce l'ID del reparto associato all'impianto, se presente

4.1.8.4 Room

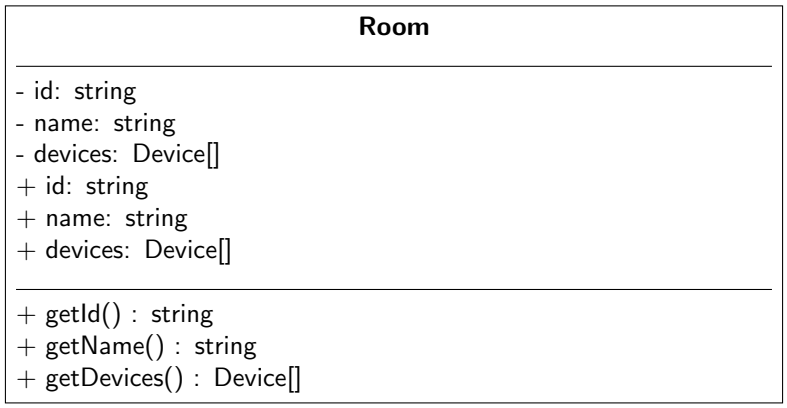

Figura 281: Diagramma della classe Room

Descrizione: Oggetto di dominio che rappresenta una stanza con i suoi dispositivi

Descrizione dei metodi della classe:

- `getId() : string`: Restituisce l'ID della stanza
- `getName() : string`: Restituisce il nome della stanza
- `getDevices() : Device[]`: Restituisce i dispositivi presenti nella stanza

4.1.8.5 PlantController

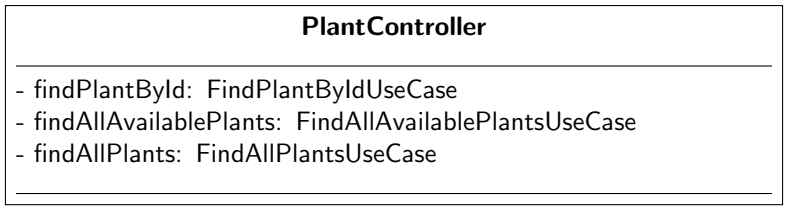

Figura 282: Diagramma della classe PlantController

Descrizione: Controller HTTP che espone gli endpoint per la ricerca degli impianti e delega ai casi d'uso

4.1.8.6 FindPlantByIdCmd

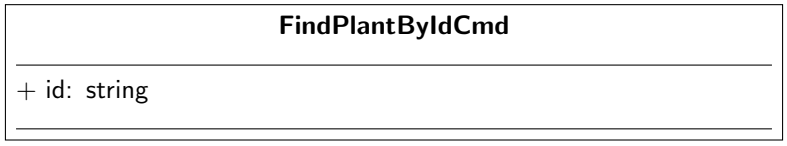

Figura 283: Diagramma della classe FindPlantByIdCmd

Descrizione: Comando che trasporta il plantId per cercare un impianto specifico

4.1.8.7 FindPlantByIdUseCase

FindPlantByIdUseCase

+ findById(cmd: FindPlantByIdCmd) : Promise¡Plant>

Figura 284: Diagramma dell'interfaccia FindPlantByIdUseCase

Descrizione: Porta in ingresso per cercare un impianto per ID

Descrizione dei metodi dell'interfaccia:

- `findById(cmd: FindPlantByIdCmd) : Promise<Plant>`: Cerca un impianto tramite il suo ID univoco

4.1.8.8 FindAllAvailablePlantsUseCase

FindAllAvailablePlantsUseCase

+ findAllAvailablePlants() : Promise¡Plant[]>

Figura 285: Diagramma dell'interfaccia FindAllAvailablePlantsUseCase

Descrizione: Porta in ingresso per ottenere tutti gli impianti disponibili

Descrizione dei metodi dell'interfaccia:

- `findAllAvailablePlants() : Promise<Plant[]>`: Restituisce tutti gli impianti disponibili per l'utente corrente

4.1.8.9 FindAllPlantsUseCase

FindAllPlantsUseCase

+ findAllPlants() : Promise¡Plant[]>

Figura 286: Diagramma dell'interfaccia FindAllPlantsUseCase

Descrizione: Porta in ingresso per ottenere tutti gli impianti del sistema

Descrizione dei metodi dell'interfaccia:

- `findAllPlants() : Promise<Plant[]>`: Restituisce tutti gli impianti presenti nel sistema

4.1.8.10 PlantService

PlantService

- findByIdPort: FindPlantByIdPort
- findAllAvailablePlantsPort: FindAllAvailablePlantsPort
- findAllPlantsPort: FindAllPlantsPort

+ findById(cmd: FindPlantByIdCmd) : Promise¡Plant>
+ findAllAvailablePlants() : Promise¡Plant[]>
+ findAllPlants() : Promise¡Plant[]>

Figura 287: Diagramma della classe PlantService

Descrizione: Servizio applicativo che implementa i casi d'uso del modulo Plant orchestrando le porte di uscita

Descrizione dei metodi della classe:

- `findById(cmd: FindPlantByIdCmd) : Promise<Plant>`: Delega la ricerca dell'impianto per ID alla porta corrispondente
- `findAllAvailablePlants() : Promise<Plant[]>`: Delega il recupero degli impianti disponibili alla porta corrispondente
- `findAllPlants() : Promise<Plant[]>`: Delega il recupero di tutti gli impianti alla porta corrispondente

4.1.8.11 FindPlantByIdPort

FindPlantByIdPort

+ findById(cmd: FindPlantByIdCmd) : Promise¡Plant | null>

Figura 288: Diagramma dell'interfaccia FindPlantByIdPort

Descrizione: Porta di uscita per cercare un impianto per ID

Descrizione dei metodi dell'interfaccia:

- `findById(cmd: FindPlantByIdCmd) : Promise<Plant | null>`: Cerca un impianto tramite il suo ID univoco

4.1.8.12 FindAllAvailablePlantsPort

FindAllAvailablePlantsPort

+ findAllAvailablePlants() : Promise<Plant[] | null>

Figura 289: Diagramma dell'interfaccia FindAllAvailablePlantsPort

Descrizione: Porta di uscita per ottenere tutti gli impianti disponibili

Descrizione dei metodi dell'interfaccia:

- `findAllAvailablePlants() : Promise<Plant[] | null>`: Restituisce tutti gli impianti disponibili per l'utente corrente

4.1.8.13 FindAllPlantsPort

FindAllPlantsPort

+ findAllPlants() : Promise<Plant[] | null>

Figura 290: Diagramma dell'interfaccia FindAllPlantsPort

Descrizione: Porta di uscita per ottenere tutti gli impianti del sistema

Descrizione dei metodi dell'interfaccia:

- `findAllPlants() : Promise<Plant[] | null>`: Restituisce tutti gli impianti presenti nel sistema

4.1.8.14 PlantEntity

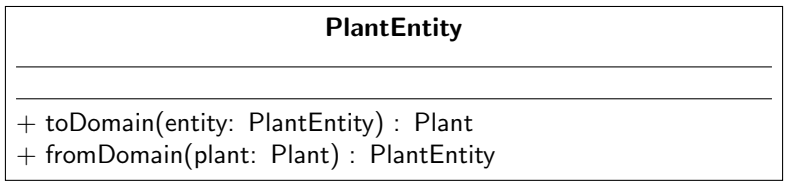

Figura 291: Diagramma della classe PlantEntity

Descrizione: Entità di persistenza di un impianto, con metodi di conversione da/verso il modello di dominio

Descrizione dei metodi della classe:

- `toDomain(entity: PlantEntity) : Plant`: Converte l'entità persistita nel corrispondente oggetto di dominio
- `fromDomain(plant: Plant) : PlantEntity`: Converte l'oggetto di dominio nella corrispondente entità persistita

4.1.8.15 FindPlantByIdAdapter

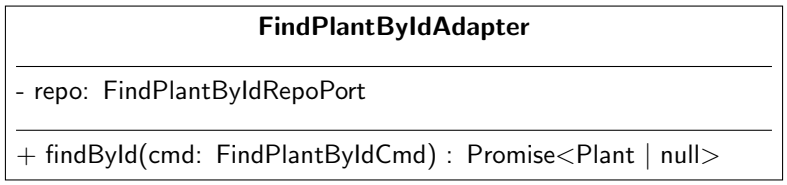

Figura 292: Diagramma della classe FindPlantByIdAdapter

Descrizione: Adapter che implementa FindPlantByIdPort delegando la ricerca al repository

Descrizione dei metodi della classe:

- `findById(cmd: FindPlantByIdCmd) : Promise<Plant | null>`: Recupera l'entità dal repository e la converte nel modello di dominio

4.1.8.16 FindAllAvailablePlantsAdapter

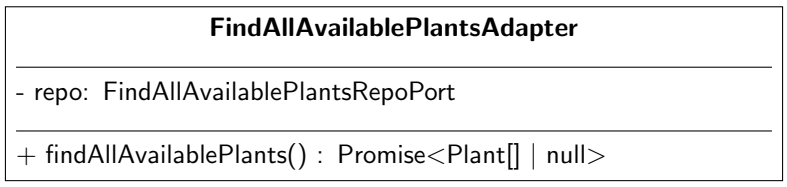

Figura 293: Diagramma della classe FindAllAvailablePlantsAdapter

Descrizione: Adapter che implementa FindAllAvailablePlantsPort delegando la ricerca al repository

Descrizione dei metodi della classe:

- `findAllAvailablePlants() : Promise<Plant[] | null>`: Recupera le entità dal repository e le converte nei modelli di dominio

4.1.8.17 FindAllPlantsAdapter

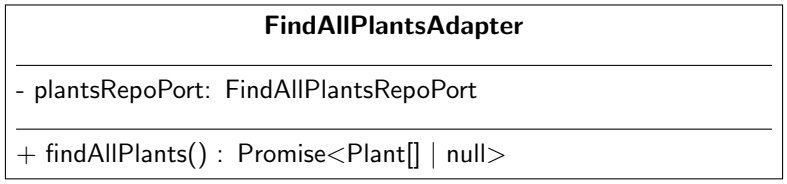

Figura 294: Diagramma della classe FindAllPlantsAdapter

Descrizione: Adapter che implementa FindAllPlantsPort delegando la ricerca al repository

Descrizione dei metodi della classe:

- `findAllPlants() : Promise<Plant[] | null>`: Recupera le entità dal repository e le converte nei modelli di dominio

4.1.8.18 FindPlantByIdRepoPort

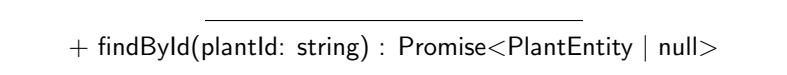

Figura 295: Diagramma dell'interfaccia FindPlantByIdRepoPort

Descrizione: Contratto del repository di persistenza per cercare un impianto per ID

Descrizione dei metodi dell'interfaccia:

- `findById(plantId: string) : Promise<PlantEntity | null>`: Cerca nel database un impianto tramite il suo ID univoco

4.1.8.19 FindAllAvailablePlantsRepoPort

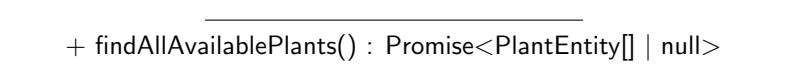

Figura 296: Diagramma dell'interfaccia FindAllAvailablePlantsRepoPort

Descrizione: Contratto del repository di persistenza per ottenere gli impianti disponibili

Descrizione dei metodi dell'interfaccia:

- `findAllAvailablePlants() : Promise<PlantEntity[] | null>`: Recupera dal database tutti gli impianti disponibili per l'utente corrente

4.1.8.20 FindAllPlantsRepoPort

FindAllPlantsRepoPort

+ findAllPlants() : Promise<PlantEntity[] | null>

Figura 297: Diagramma dell'interfaccia FindAllPlantsRepoPort

Descrizione: Contratto del repository di persistenza per ottenere tutti gli impianti

Descrizione dei metodi dell'interfaccia:

- `findAllPlants() : Promise<PlantEntity[] | null>`: Recupera dal database tutti gli impianti presenti nel sistema

4.1.8.21 PlantRepositoryImpl

PlantRepositoryImpl

+ findById(plantId: string) : Promise¡PlantEntity | null>
+ findAllAvailablePlants() : Promise¡PlantEntity[] | null>
+ findAllPlants() : Promise¡PlantEntity[] | null>

Figura 298: Diagramma della classe PlantRepositoryImpl

Descrizione: Implementazione del repository di persistenza per le operazioni di lettura sugli impianti

Descrizione dei metodi della classe:

- `findById(plantId: string) : Promise<PlantEntity | null>`: Cerca nel database un impianto tramite il suo ID univoco
- `findAllAvailablePlants() : Promise<PlantEntity[] | null>`: Recupera dal database tutti gli impianti disponibili per l'utente corrente
- `findAllPlants() : Promise<PlantEntity[] | null>`: Recupera dal database tutti gli impianti presenti nel sistema

4.1.9 Subscriptions

4.1.9.1 SubscriptionAttributesDto

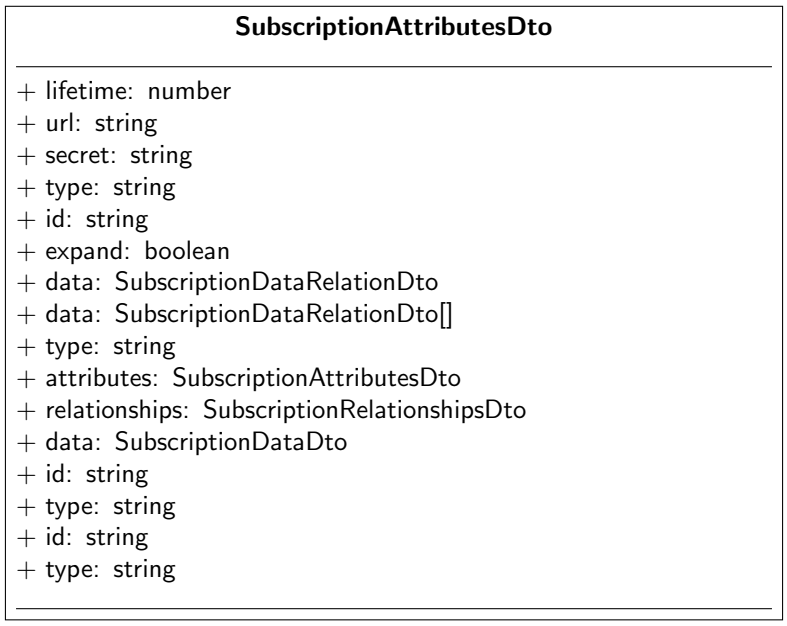

Figura 299: Diagramma della classe SubscriptionAttributesDto

Descrizione: DTO che struttura il payload per la creazione di una sottoscrizione verso l'API esterna

4.1.9.2 EventSubscriptionController

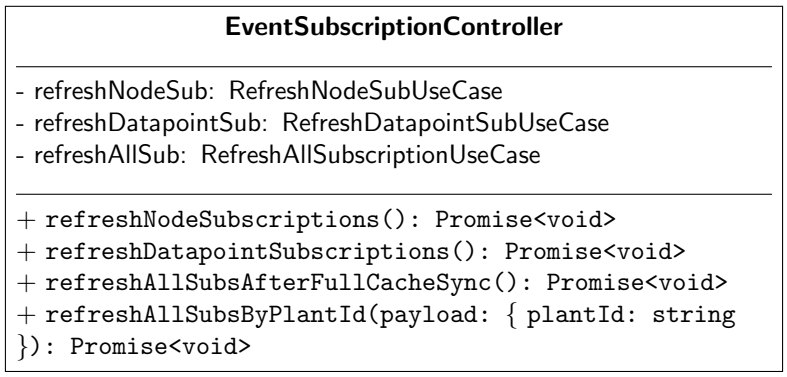

Figura 300: Diagramma della classe EventSubscriptionController

Descrizione: Controller eventi che ascolta gli eventi interni e delega il refresh delle sottoscrizioni ai relativi casi d'uso

Descrizione dei metodi della classe:

- `refreshNodeSubscriptions(): Promise<void>`: Gestisce l'evento e avvia il refresh delle sottoscrizioni sui nodi per tutti gli impianti
- `refreshDatapointSubscriptions(): Promise<void>`: Gestisce l'evento e avvia il refresh delle sottoscrizioni sui datapoint per tutti gli impianti
- `refreshAllSubsAfterFullCacheSync(): Promise<void>`: Avvia il refresh di tutte le sottoscrizioni al termine di una sincronizzazione completa della cache
- `refreshAllSubsByPlantId(payload: { plantId: string }): Promise<void>`: Avvia il refresh di tutte le sottoscrizioni per un impianto specifico

4.1.9.3 RefreshNodeSubCmd

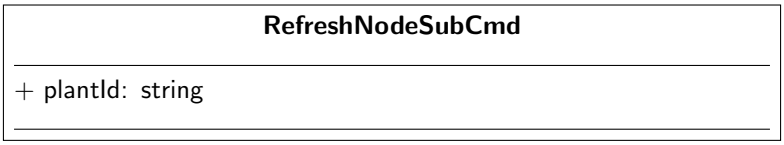

Figura 301: Diagramma della classe RefreshNodeSubCmd

Descrizione: Comando che trasporta i dati per aggiornare le sottoscrizioni sui nodi di un impianto

4.1.9.4 RefreshDatapointSubCmd

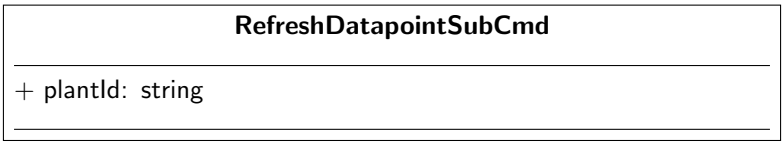

Figura 302: Diagramma della classe RefreshDatapointSubCmd

Descrizione: Comando che trasporta i dati per aggiornare le sottoscrizioni sui datapoint di un impianto

4.1.9.5 RefreshAllSubCmd

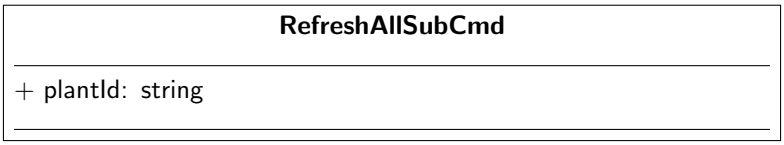

Figura 303: Diagramma della classe RefreshAllSubCmd

Descrizione: Comando che trasporta i dati per aggiornare tutte le sottoscrizioni di un impianto

4.1.9.6 RefreshNodeSubUseCase

Figura 304: Diagramma dell'interfaccia RefreshNodeSubUseCase

Descrizione: Porta in ingresso per aggiornare le sottoscrizioni sui nodi

Descrizione dei metodi dell'interfaccia:

- `refreshSub(): Promise<boolean>`: Aggiorna le sottoscrizioni sui nodi per tutti gli impianti disponibili

4.1.9.7 RefreshDatapointSubUseCase

RefreshDatapointSubUseCase

+ `refreshDatapointSub(): Promise<boolean>`

Figura 305: Diagramma dell'interfaccia RefreshDatapointSubUseCase

Descrizione: Porta in ingresso per aggiornare le sottoscrizioni sui datapoint

Descrizione dei metodi dell'interfaccia:

- `refreshDatapointSub(): Promise<boolean>`: Aggiorna le sottoscrizioni sui datapoint per tutti gli impianti disponibili

4.1.9.8 RefreshAllSubscriptionUseCase

RefreshAllSubscriptionUseCase

+ `refreshAllSubscription(cmd: RefreshAllSubCmd): Promise<boolean>`

Figura 306: Diagramma dell'interfaccia RefreshAllSubscriptionUseCase

Descrizione: Porta in ingresso per aggiornare tutte le sottoscrizioni di un impianto

Descrizione dei metodi dell'interfaccia:

- `refreshAllSubscription(cmd: RefreshAllSubCmd): Promise<boolean>`: Aggiorna tutte le sottoscrizioni per l'impianto indicato nel comando

4.1.9.9 SubscriptionService

SubscriptionService

- refreshPort: RefreshNodeSubscriptionPort
- refreshDatapointPort: RefreshDatapointSubPort
- getAllPlantIdsPort: GetAllPlantIdsPort

+ `refreshSub(): Promise<boolean>`
+ `refreshDatapointSub(): Promise<boolean>`
+ `refreshAllSubscription(cmd: RefreshAllSubCmd): Promise<boolean>`

Figura 307: Diagramma della classe SubscriptionService

Descrizione: Servizio applicativo che orchestra il refresh delle sottoscrizioni recuperando gli impianti e delegando alle porte di uscita

Descrizione dei metodi della classe:

- `refreshSub(): Promise<boolean>`: Recupera tutti gli ID impianto e aggiorna le sottoscrizioni sui nodi per ciascuno
- `refreshDatapointSub(): Promise<boolean>`: Recupera tutti gli ID impianto e aggiorna le sottoscrizioni sui datapoint per ciascuno
- `refreshAllSubscription(cmd: RefreshAllSubCmd): Promise<boolean>`: Aggiorna tutte le sottoscrizioni (nodi e datapoint) per l'impianto indicato nel comando

4.1.9.10 RefreshNodeSubscriptionPort

RefreshNodeSubscriptionPort

+ `refreshSub(cmd: RefreshNodeSubCmd): Promise<boolean>`

Figura 308: Diagramma dell'interfaccia RefreshNodeSubscriptionPort

Descrizione: Porta di uscita per aggiornare le sottoscrizioni sui nodi di un impianto

Descrizione dei metodi dell'interfaccia:

- `refreshSub(cmd: RefreshNodeSubCmd): Promise<boolean>`: Aggiorna le sottoscrizioni sui nodi per l'impianto indicato nel comando

4.1.9.11 RefreshDatapointSubPort

RefreshDatapointSubPort

+ `refreshDatapointSub(cmd: RefreshDatapointSubCmd): Promise<boolean>`

Figura 309: Diagramma dell'interfaccia RefreshDatapointSubPort

Descrizione: Porta di uscita per aggiornare le sottoscrizioni sui datapoint di un impianto

Descrizione dei metodi dell'interfaccia:

- `refreshDatapointSub(cmd: RefreshDatapointSubCmd): Promise<boolean>`: Aggiorna le sottoscrizioni sui datapoint per l'impianto indicato nel comando

4.1.9.12 RefreshNodeSubscriptionAdapter

RefreshNodeSubscriptionAdapter

- nodeRepo: RefreshNodeSubscriptionRepoPort
- getValidTokenPort: GetValidTokenPort

+ `refreshSub(cmd: RefreshNodeSubCmd): Promise<boolean>`

Figura 310: Diagramma della classe RefreshNodeSubscriptionAdapter

Descrizione: Adapter che implementa RefreshNodeSubscriptionPort coordinando autenticazione e aggiornamento delle sottoscrizioni sui nodi

Descrizione dei metodi della classe:

- `refreshSub(cmd: RefreshNodeSubCmd): Promise<boolean>`: Ottiene un token valido e delega il refresh delle sottoscrizioni sui nodi al repository

4.1.9.13 RefreshDatapointSubAdapter

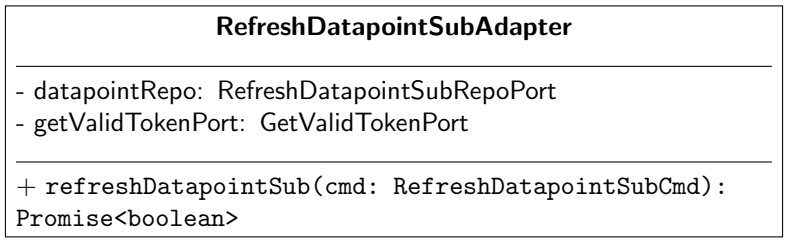

Figura 311: Diagramma della classe RefreshDatapointSubAdapter

Descrizione: Adapter che implementa RefreshDatapointSubPort coordinando autenticazione e aggiornamento delle sottoscrizioni sui datapoint

Descrizione dei metodi della classe:

- `refreshDatapointSub(cmd: RefreshDatapointSubCmd): Promise<boolean>`: Ottiene un token valido e delega il refresh delle sottoscrizioni sui datapoint al repository

4.1.9.14 RefreshNodeSubscriptionRepoPort

RefreshNodeSubscriptionRepoPort

+ `refreshSub(validToken: string, plantId: string): Promise<boolean>`

Figura 312: Diagramma dell'interfaccia RefreshNodeSubscriptionRepoPort

Descrizione: Contratto del repository HTTP per aggiornare le sottoscrizioni sui nodi di un impianto

Descrizione dei metodi dell'interfaccia:

- `refreshSub(validToken: string, plantId: string): Promise<boolean>`: Crea o rinnova le sottoscrizioni sui nodi per l'impianto tramite API esterna

4.1.9.15 RefreshDatapointSubRepoPort

RefreshDatapointSubRepoPort

+ `refreshDatapointSub(validToken: string, plantId: string): Promise<boolean>`

Figura 313: Diagramma dell'interfaccia RefreshDatapointSubRepoPort

Descrizione: Contratto del repository HTTP per aggiornare le sottoscrizioni sui datapoint di un impianto

Descrizione dei metodi dell'interfaccia:

- `refreshDatapointSub(validToken: string, plantId: string): Promise<boolean>`: Crea o rinnova le sottoscrizioni sui datapoint per l'impianto tramite API esterna

4.1.9.16 SubscriptionRepoImpl

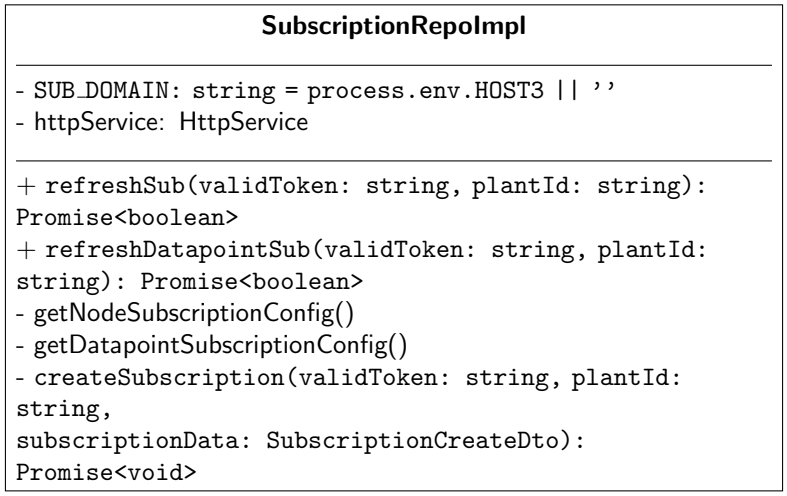

Figura 314: Diagramma della classe SubscriptionRepoImpl

Descrizione: Implementazione HTTP che crea e rinnova le sottoscrizioni su nodi e datapoint tramite API esterna

Descrizione dei metodi della classe:

- `refreshSub(validToken: string, plantId: string): Promise<boolean>`: Crea o rinnova le sottoscrizioni sui nodi per l'impianto tramite API esterna
- `refreshDatapointSub(validToken: string, plantId: string): Promise<boolean>`: Crea o rinnova le sottoscrizioni sui datapoint per l'impianto tramite API esterna
- `getNodeSubscriptionConfig()`: Restituisce la configurazione statica per le sottoscrizioni sui nodi
- `getDatapointSubscriptionConfig()`: Restituisce la configurazione statica per le sottoscrizioni sui datapoint
- `createSubscription(validToken: string, plantId: string, subscriptionData: SubscriptionCreateDto): Promise<void>`: Esegue la chiamata HTTP per creare una singola sottoscrizione sull'API esterna

4.1.10 Users

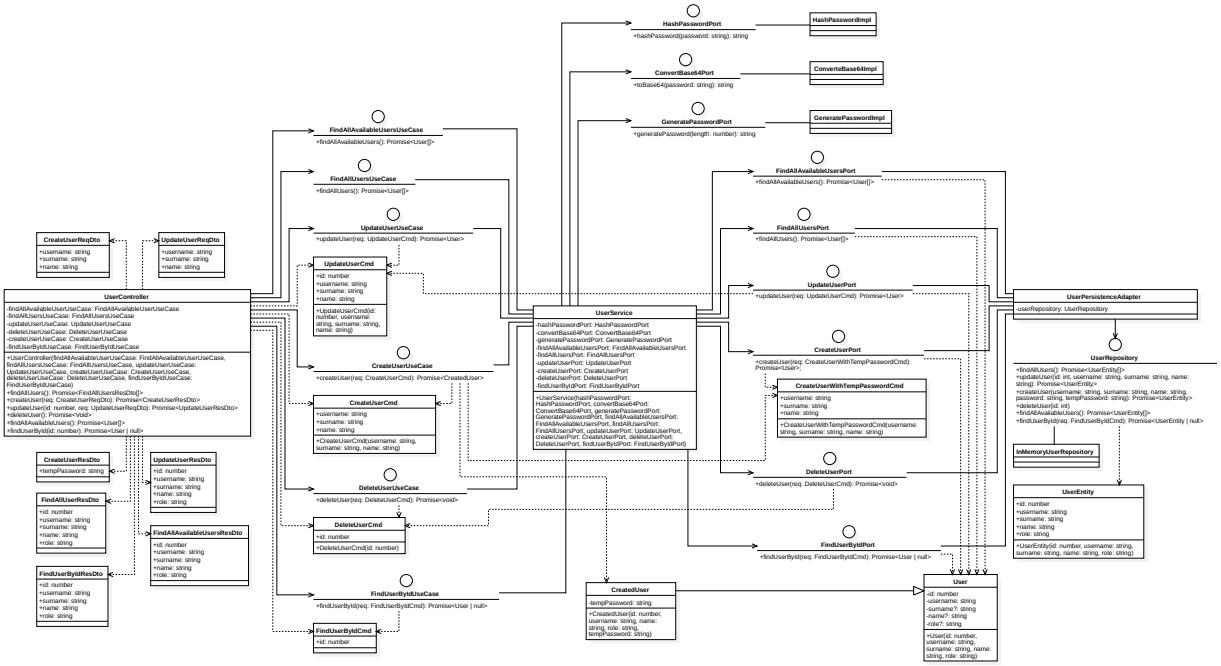

Figura 315: Diagramma delle classi del modulo Users

4.1.10.1 CreateUserWithTempPasswordCmd

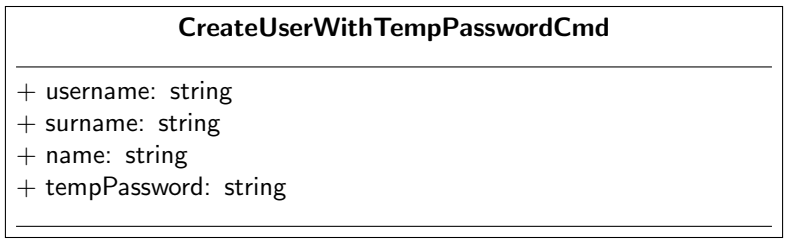

Figura 316: Diagramma della classe CreateUserWithTempPasswordCmd

Descrizione: Comando applicativo per la creazione di un utente con password temporanea già generata.

4.1.10.2 CreateUserReqDto

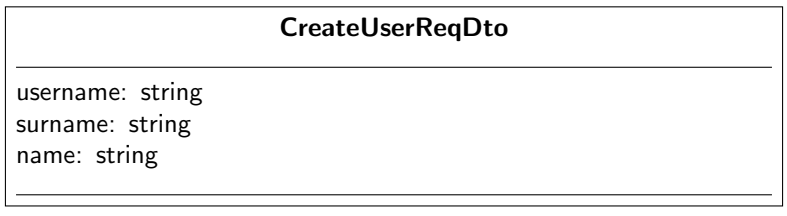

Figura 317: Diagramma della classe CreateUserReqDto

Descrizione: DTO di richiesta per la creazione di un nuovo utente

4.1.10.3 UpdateUserReqDto

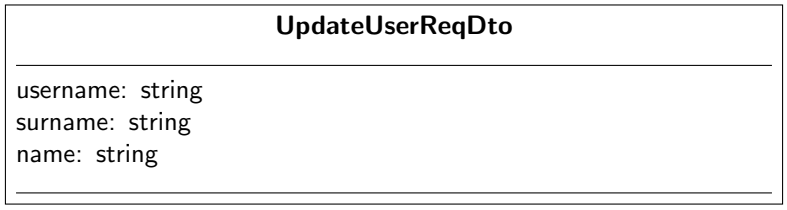

Figura 318: Diagramma della classe UpdateUserReqDto

Descrizione: DTO di richiesta per l'aggiornamento dei dati di un utente esistente

4.1.10.4 UpdateUserCmd

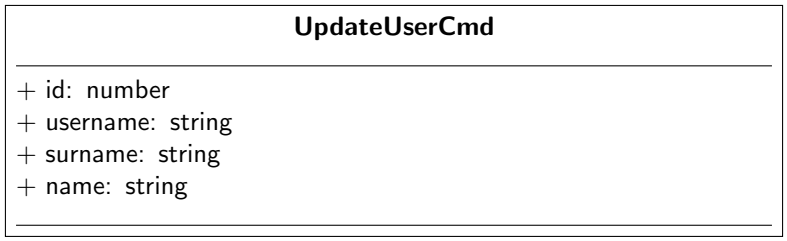

Figura 319: Diagramma della classe UpdateUserCmd

Descrizione: Comando applicativo per l'aggiornamento dei dati di un utente.

4.1.10.5 FindAllAvailableUsersResDto

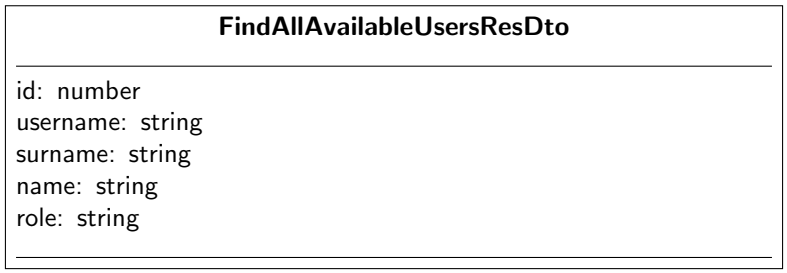

Figura 320: Diagramma della classe FindAllAvailableUsersResDto

Descrizione: DTO di risposta per un elemento della lista degli utenti disponibili per l'assegnazione

4.1.10.6 FindAllUserResDto

Figura 321: Diagramma della classe FindAllUserResDto

Descrizione: DTO di risposta per un elemento della lista di tutti gli utenti del sistema

4.1.10.7 FindUserByIdResDto

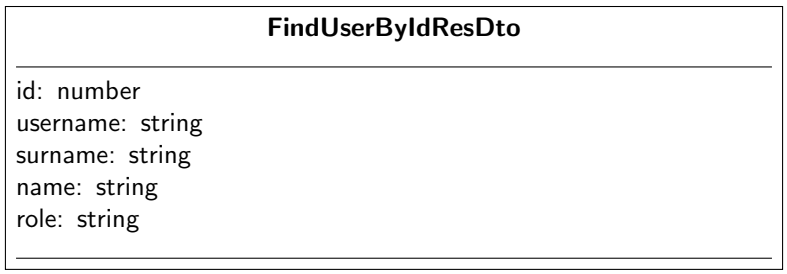

Figura 322: Diagramma della classe FindUserByIdResDto

Descrizione: DTO di risposta contenente i dettagli di un singolo utente

4.1.10.8 UpdateUserResDto

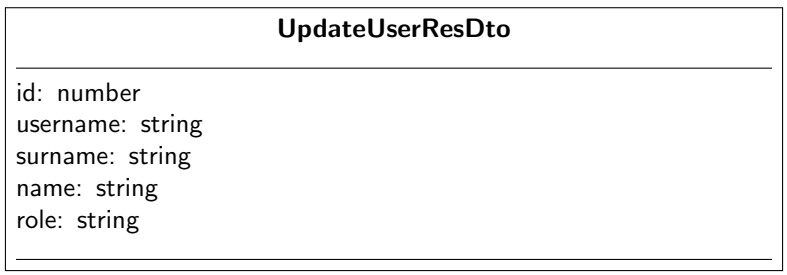

Figura 323: Diagramma della classe UpdateUserResDto

Descrizione: DTO di risposta restituito dopo l'aggiornamento di un utente

4.1.10.9 CreateUserResDto

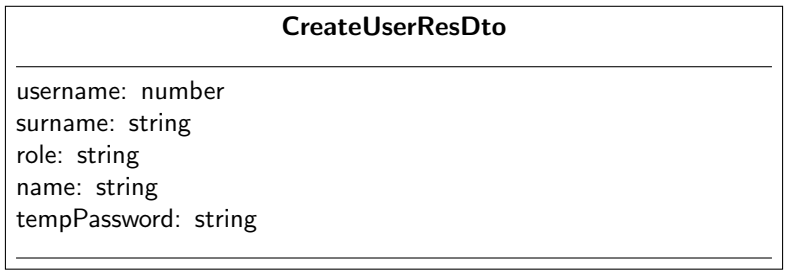

Figura 324: Diagramma della classe CreateUserResDto

Descrizione: DTO di risposta restituito dopo la creazione di un nuovo utente, include la password temporanea

4.1.10.10 CreatedUser

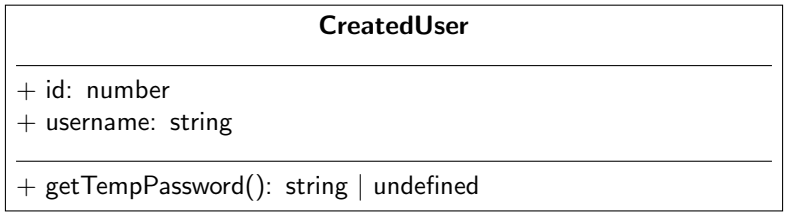

Figura 325: Diagramma della classe CreatedUser

Descrizione: Modello di dominio che rappresenta un utente appena creato, con la relativa password temporanea

Descrizione dei metodi della classe:

- `getTempPassword(): string | undefined`: Restituisce la password temporanea generata, undefined se non disponibile

4.1.10.11 User

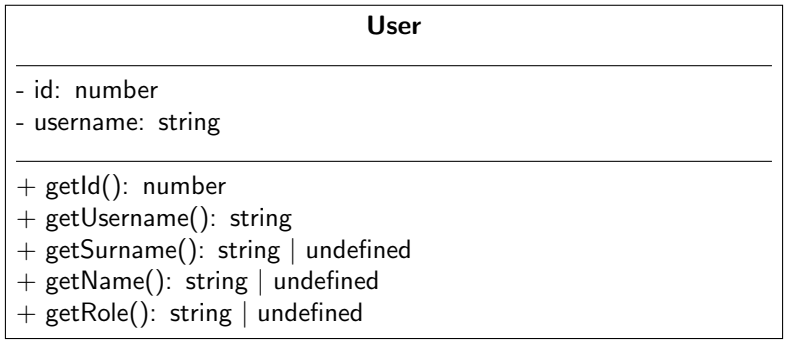

Figura 326: Diagramma della classe User

Descrizione: Modello di dominio che rappresenta un utente del sistema

Descrizione dei metodi della classe:

- `getId(): number`: Restituisce l'identificativo univoco dell'utente

- `getUsername(): string`: Restituisce lo username dell'utente
- `getSurname(): string | undefined`: Restituisce il cognome dell'utente, undefined se non impostato
- `getName(): string | undefined`: Restituisce il nome dell'utente, undefined se non impostato
- `getRole(): string | undefined`: Restituisce il ruolo dell'utente, undefined se non impostato

4.1.10.12 UsersController

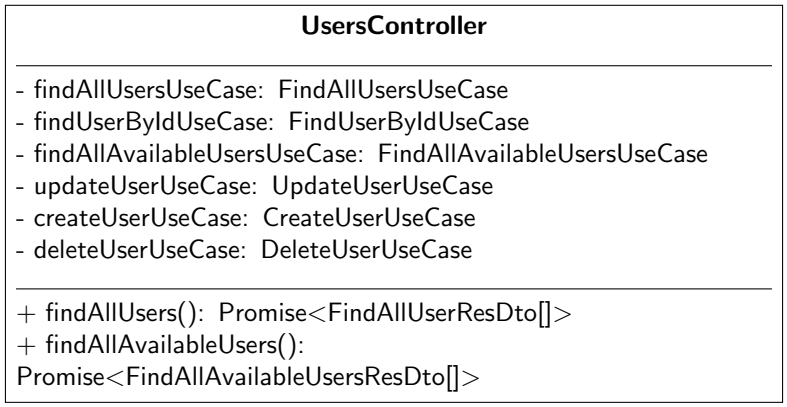

Figura 327: Diagramma della classe UsersController

Descrizione: Controller REST che gestisce le richieste HTTP relative agli utenti

Descrizione dei metodi della classe:

- `findAllUsers(): Promise<FindAllUserResDto[]>`: Espone l'endpoint per il recupero di tutti gli utenti del sistema
- `findAllAvailableUsers(): Promise<FindAllAvailableUsersResDto[]>`: Espone l'endpoint per il recupero degli utenti disponibili per l'assegnazione

4.1.10.13 CreateUserCmd

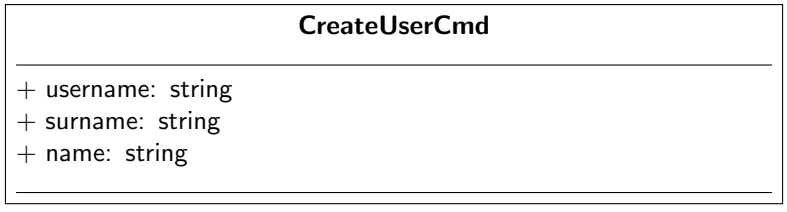

Figura 328: Diagramma della classe CreateUserCmd

Descrizione: Comando per la creazione di un nuovo utente

4.1.10.14 CreateUserWithTempPasswordCmd

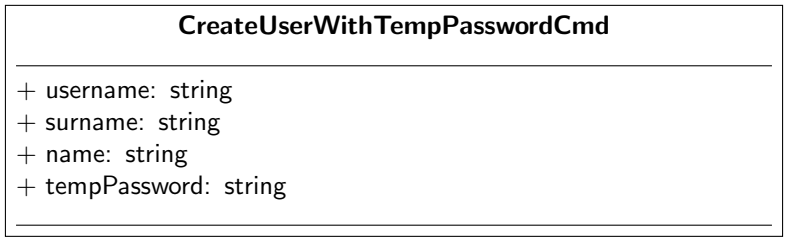

Figura 329: Diagramma della classe CreateUserWithTempPasswordCmd

Descrizione: Comando per la creazione di un utente comprensivo di password temporanea già generata

4.1.10.15 DeleteUserCmd

Figura 330: Diagramma della classe DeleteUserCmd

Descrizione: Comando per l'eliminazione di un utente

4.1.10.16 FindUserByIdCmd

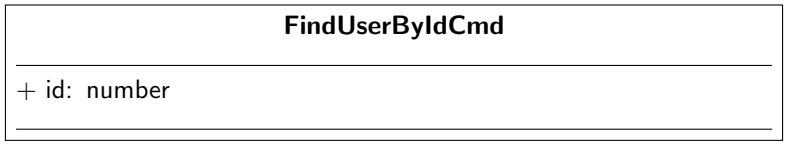

Figura 331: Diagramma della classe FindUserByIdCmd

Descrizione: Comando per il recupero di un utente tramite identificativo

4.1.10.17 UpdateUserCmd

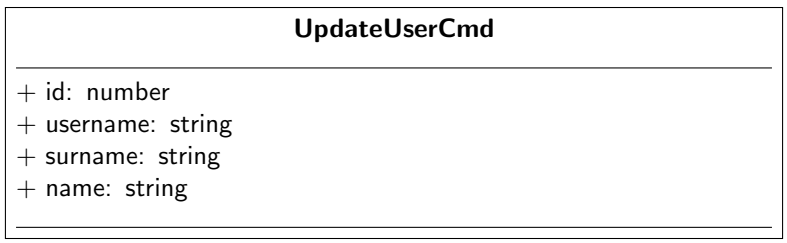

Figura 332: Diagramma della classe UpdateUserCmd

Descrizione: Comando per l'aggiornamento dei dati anagrafici di un utente

4.1.10.18 CreateUserUseCase

CreateUserUseCase

+ createUser(req: CreateUserCmd): Promise<CreatedUser>

Figura 333: Diagramma dell'interfaccia CreateUserUseCase

Descrizione: Interfaccia del use case per la creazione di un nuovo utente

Descrizione dei metodi dell'interfaccia:

- `createUser(req: CreateUserCmd): Promise<CreatedUser>`: Crea un nuovo utente generando una password temporanea e restituisce il modello dell'utente creato

4.1.10.19 DeleteUserUseCase

DeleteUserUseCase

+ deleteUser(req: DeleteUserCmd): Promise<void>

Figura 334: Diagramma dell'interfaccia DeleteUserUseCase

Descrizione: Interfaccia del use case per l'eliminazione di un utente

Descrizione dei metodi dell'interfaccia:

- `deleteUser(req: DeleteUserCmd): Promise<void>`: Elimina l'utente identificato dal comando

4.1.10.20 FindAllAvailableUsersUseCase

FindAllAvailableUsersUseCase

+ findAllAvailableUsers(): Promise<User[]>

Figura 335: Diagramma dell'interfaccia FindAllAvailableUsersUseCase

Descrizione: Interfaccia del use case per il recupero degli utenti disponibili

Descrizione dei metodi dell'interfaccia:

- `findAllAvailableUsers(): Promise<User[]>`: Recupera tutti gli utenti disponibili per l'assegnazione a un ward

4.1.10.21 FindAllUsersUseCase

FindAllUsersUseCase

+ findAllUsers(): Promise<User[]>

Figura 336: Diagramma dell'interfaccia FindAllUsersUseCase

Descrizione: Interfaccia del use case per il recupero di tutti gli utenti

Descrizione dei metodi dell'interfaccia:

- `findAllUsers(): Promise<User[]>`: Recupera tutti gli utenti registrati nel sistema

4.1.10.22 FindUserByIdUseCase

FindUserByIdUseCase

+ findUserById(req: FindUserByIdCmd): Promise<User | null>

Figura 337: Diagramma dell'interfaccia FindUserByIdUseCase

Descrizione: Interfaccia del use case per il recupero di un singolo utente

Descrizione dei metodi dell'interfaccia:

- `findUserById(req: FindUserByIdCmd): Promise<User | null>`: Recupera un utente tramite identificativo, restituisce null se non trovato

4.1.10.23 UpdateUserUseCase

UpdateUserUseCase

+ updateUser(req: UpdateUserCmd): Promise<User>

Figura 338: Diagramma dell'interfaccia UpdateUserUseCase

Descrizione: Interfaccia del use case per l'aggiornamento dei dati di un utente

Descrizione dei metodi dell'interfaccia:

- `updateUser(req: UpdateUserCmd): Promise<User>`: Aggiorna i dati anagrafici dell'utente e restituisce il modello aggiornato

4.1.10.24 UsersService

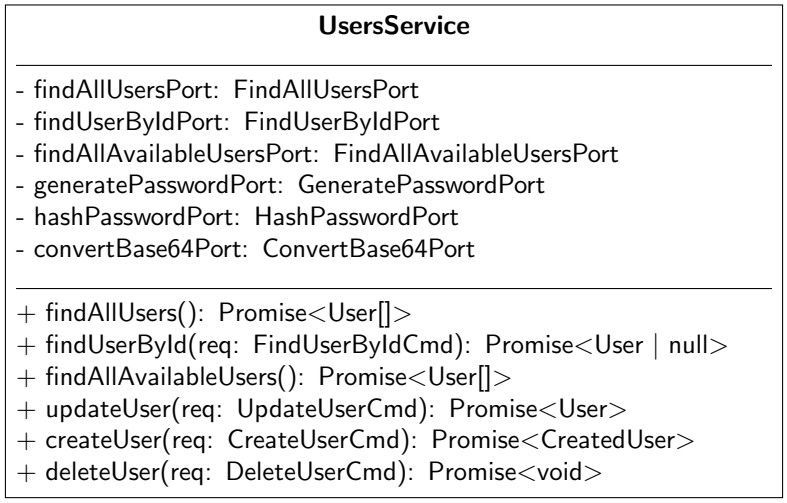

Figura 339: Diagramma della classe UsersService

Descrizione: Servizio applicativo che implementa i use case relativi alla gestione degli utenti

Descrizione dei metodi della classe:

- `findAllUsers(): Promise<User[]>`: Recupera tutti gli utenti delegando alla porta di persistenza
- `findUserById(req: FindUserByIdCmd): Promise<User | null>`: Recupera un utente per identificativo delegando alla porta di persistenza
- `findAllAvailableUsers(): Promise<User[]>`: Recupera gli utenti disponibili delegando alla porta di persistenza
- `updateUser(req: UpdateUserCmd): Promise<User>`: Aggiorna i dati di un utente delegando alla porta di persistenza
- `createUser(req: CreateUserCmd): Promise<CreatedUser>`: Genera una password temporanea, la codifica e crea il nuovo utente tramite le porte di persistenza
- `deleteUser(req: DeleteUserCmd): Promise<void>`: Elimina un utente delegando alla porta di persistenza

4.1.10.25 ConvertBase64Port

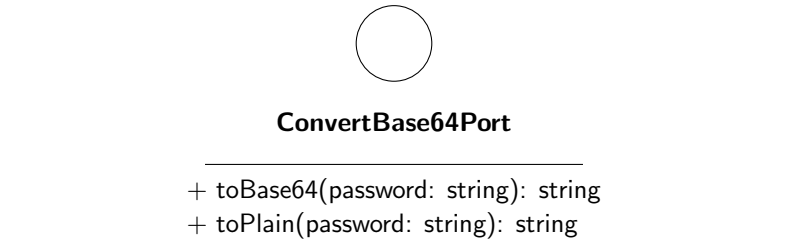

Figura 340: Diagramma dell'interfaccia ConvertBase64Port

Descrizione: Porta di uscita per la conversione delle password da e verso il formato Base64

Descrizione dei metodi dell'interfaccia:

- `toBase64(password: string): string`: Codifica la password fornita in formato Base64
- `toPlain(password: string): string`: Decodifica una password da formato Base64 al testo in chiaro

4.1.10.26 CreateUserPort

CreateUserPort

+ createUser(req: CreateUserWithTempPasswordCmd): Promise<User>

Figura 341: Diagramma dell'interfaccia CreateUserPort

Descrizione: Porta di uscita per la creazione e persistenza di un nuovo utente

Descrizione dei metodi dell'interfaccia:

- `createUser(req: CreateUserWithTempPasswordCmd): Promise<User>`: Persiste il nuovo utente con la password temporanea e restituisce il modello creato

4.1.10.27 DeleteUserPort

DeleteUserPort

+ deleteUser(req: DeleteUserCmd): Promise<void>

Figura 342: Diagramma dell'interfaccia DeleteUserPort

Descrizione: Porta di uscita per l'eliminazione di un utente

Descrizione dei metodi dell'interfaccia:

- `deleteUser(req: DeleteUserCmd): Promise<void>`: Elimina l'utente specificato dal livello di persistenza

4.1.10.28 FindAllAvailableUsersPort

FindAllAvailableUsersPort

+ findAllAvailableUsers(): Promise<User[]>

Figura 343: Diagramma dell'interfaccia FindAllAvailableUsersPort

Descrizione: Porta di uscita per il recupero degli utenti disponibili per l'assegnazione

Descrizione dei metodi dell'interfaccia:

- `findAllAvailableUsers(): Promise<User[]>`: Recupera dal livello di persistenza gli utenti non ancora assegnati a un ward

4.1.10.29 FindAllUsersPort

FindAllUsersPort

+ findAllUsers(): Promise<User[]>

Figura 344: Diagramma dell'interfaccia FindAllUsersPort

Descrizione: Porta di uscita per il recupero di tutti gli utenti

Descrizione dei metodi dell'interfaccia:

- `findAllUsers(): Promise<User[]>`: Recupera dal livello di persistenza tutti gli utenti del sistema

4.1.10.30 FindUserByIdPort

FindUserByIdPort

+ findUserById(req: FindUserByIdCmd): Promise<User | null>

Figura 345: Diagramma dell'interfaccia FindUserByIdPort

Descrizione: Porta di uscita per il recupero di un singolo utente tramite identificativo

Descrizione dei metodi dell'interfaccia:

- `findUserById(req: FindUserByIdCmd): Promise<User | null>`: Recupera dal livello di persistenza un utente per identificativo, restituisce null se non trovato

4.1.10.31 HashPasswordPort

HashPasswordPort

+ hashPassword(password: string): string

Figura 346: Diagramma dell'interfaccia HashPasswordPort

Descrizione: Porta di uscita per l'hashing sicuro delle password

Descrizione dei metodi dell'interfaccia:

- `hashPassword(password: string): string`: Applica una funzione di hashing alla password e restituisce il digest risultante

4.1.10.32 GeneratePasswordPort

GeneratePasswordPort

+ generatePassword(length: number): string

Figura 347: Diagramma dell'interfaccia GeneratePasswordPort

Descrizione: Porta di uscita per la generazione di password temporanee casuali

Descrizione dei metodi dell'interfaccia:

- `generatePassword(length: number): string`: Genera una password casuale della lunghezza specificata

4.1.10.33 UpdateUserPort

UpdateUserPort

+ updateUser(req: UpdateUserCmd): Promise<User>

Figura 348: Diagramma dell'interfaccia UpdateUserPort

Descrizione: Porta di uscita per l'aggiornamento dei dati di un utente

Descrizione dei metodi dell'interfaccia:

- `updateUser(req: UpdateUserCmd): Promise<User>`: Aggiorna i dati dell'utente nel livello di persistenza e restituisce il modello aggiornato

4.1.10.34 UserEntity

UserEntity

+ id: number
+ username: string
+ surname: string
+ name: string
+ role: string

Figura 349: Diagramma della classe UserEntity

Descrizione: Entità di persistenza che rappresenta un utente nel database

4.1.10.35 UserPersistenceAdapter

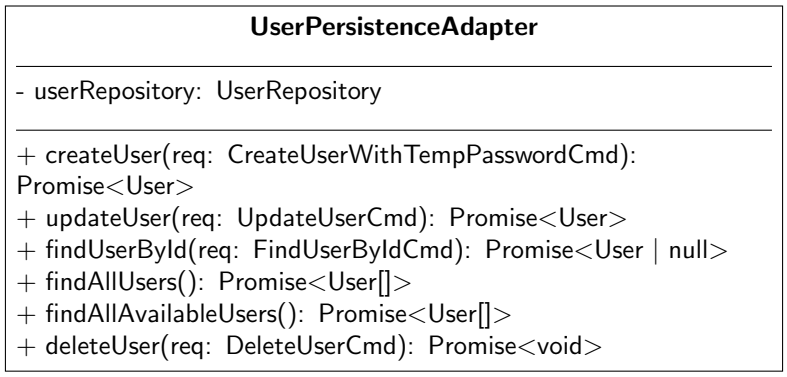

Figura 350: Diagramma della classe UserPersistenceAdapter

Descrizione: Adattatore di persistenza che implementa le porte relative agli utenti

Descrizione dei metodi della classe:

- `createUser(req: CreateUserWithTempPasswordCmd): Promise<User>`: Delega al repository la creazione dell'utente e mappa il risultato nel modello di dominio
- `updateUser(req: UpdateUserCmd): Promise<User>`: Delega al repository l'aggiornamento dell'utente e mappa il risultato nel modello di dominio
- `findUserById(req: FindUserByIdCmd): Promise<User | null>`: Recupera e mappa un utente per identificativo dal repository
- `findAllUsers(): Promise<User[]>`: Recupera e mappa tutti gli utenti dal repository
- `findAllAvailableUsers(): Promise<User[]>`: Recupera e mappa gli utenti disponibili dal repository
- `deleteUser(req: DeleteUserCmd): Promise<void>`: Delega al repository l'eliminazione dell'utente specificato

4.1.10.36 UserRepository

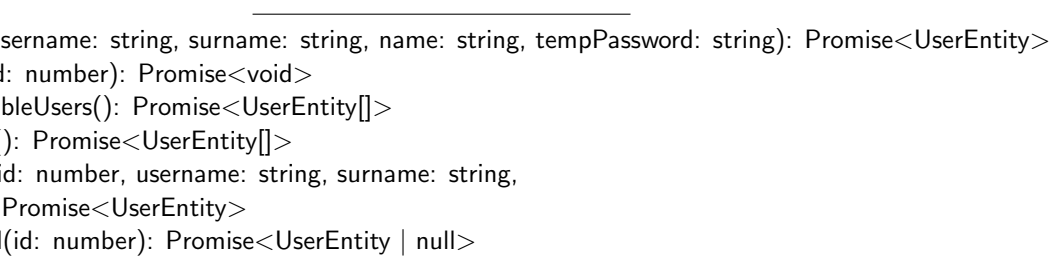

Figura 351: Diagramma dell'interfaccia UserRepository

Descrizione: Interfaccia del repository per le operazioni di persistenza relative agli utenti

Descrizione dei metodi dell'interfaccia:

- `createUser(username: string, surname: string, name: string, tempPassword: string): Promise<UserEntity>`: Inserisce un nuovo utente nel database con i dati e la password temporanea forniti
- `deleteUser(id: number): Promise<void>`: Elimina dal database l'utente con l'identificativo specificato
- `findAllAvailableUsers(): Promise<UserEntity[]>`: Recupera dal database gli utenti non ancora assegnati a un ward
- `findAllUsers(): Promise<UserEntity[]>`: Recupera dal database tutti gli utenti del sistema
- `updateUser(id: number, username: string, surname: string, name: string): Promise<UserEntity>`: Aggiorna nel database i dati anagrafici dell'utente specificato
- `findUserById(id: number): Promise<UserEntity | null>`: Recupera dal database un utente per identificativo, restituisce null se non trovato

4.1.10.37 UsersRepositoryImpl

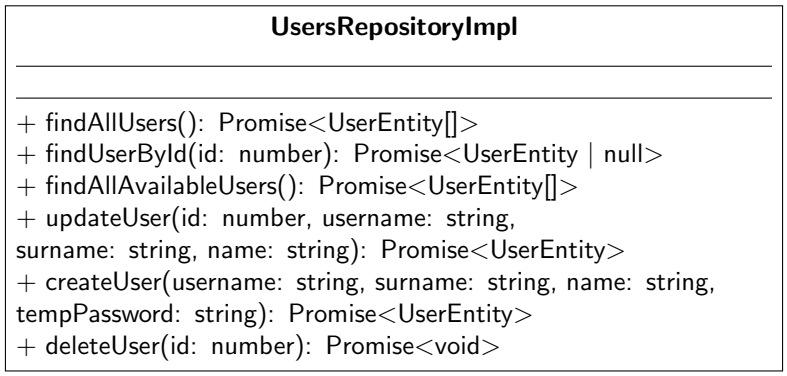

Figura 352: Diagramma della classe UsersRepositoryImpl

Descrizione: Implementazione concreta del repository per gli utenti

Descrizione dei metodi della classe:

- `findAllUsers(): Promise<UserEntity[]>`: Implementa il recupero di tutti gli utenti tramite query al database
- `findUserById(id: number): Promise<UserEntity | null>`: Implementa il recupero di un utente per identificativo tramite query al database
- `findAllAvailableUsers(): Promise<UserEntity[]>`: Implementa il recupero degli utenti disponibili tramite query al database
- `updateUser(id: number, username: string, surname: string, name: string): Promise<UserEntity>`: Implementa l'aggiornamento dei dati anagrafici di un utente tramite query al database

- `createUser(username: string, surname: string, name: string, tempPassword: string): Promise<UserEntity>`: Implementa l'inserimento di un nuovo utente nel database
- `deleteUser(id: number): Promise<void>`: Implementa l'eliminazione di un utente tramite query al database

4.1.11 Wards

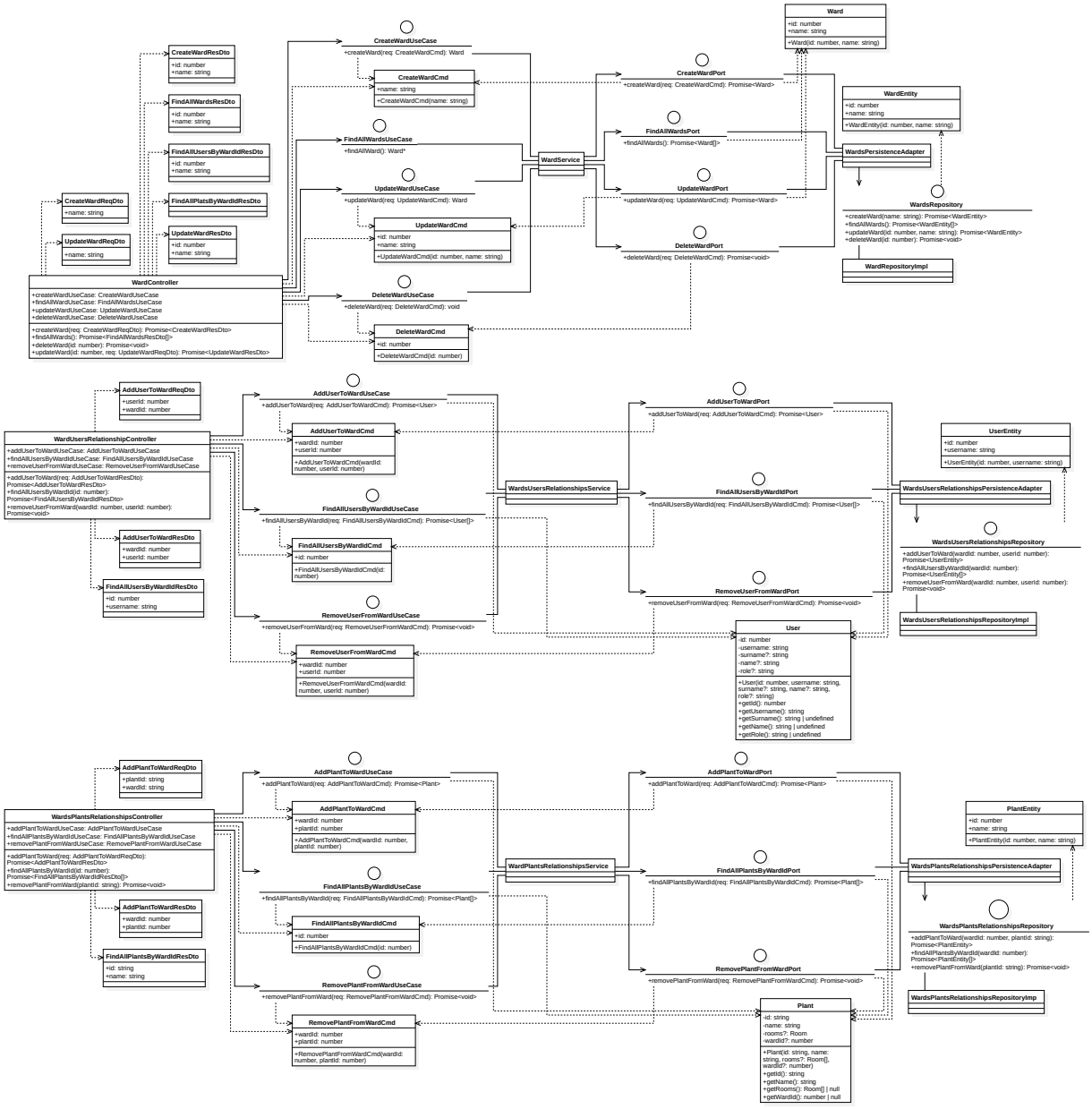

Figura 353: Diagramma delle classi del modulo Wards

4.1.11.1 AddPlantToWardReqDto

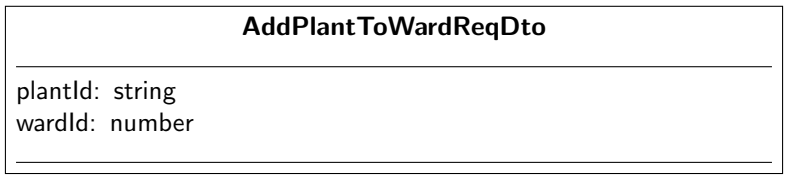

Figura 354: Diagramma della classe AddPlantToWardReqDto

Descrizione: DTO di richiesta per l'associazione di un impianto a un ward

4.1.11.2 AddUserToWardReqDto

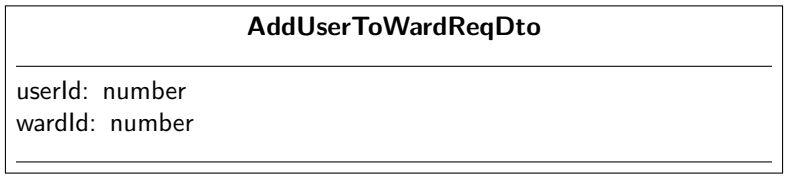

Figura 355: Diagramma della classe AddUserToWardReqDto

Descrizione: DTO di richiesta per l'associazione di un utente a un ward

4.1.11.3 CreateWardReqDto

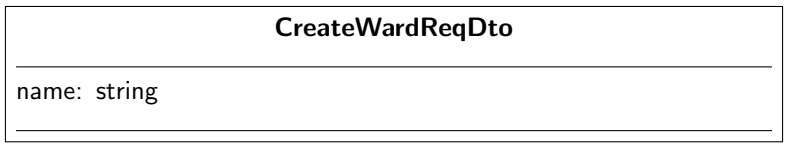

Figura 356: Diagramma della classe CreateWardReqDto

Descrizione: DTO di richiesta per la creazione di un nuovo ward

4.1.11.4 UpdateWardReqDto

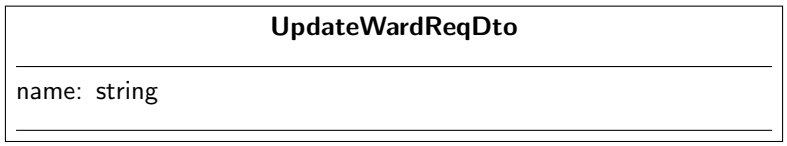

Figura 357: Diagramma della classe UpdateWardReqDto

Descrizione: DTO di richiesta per l'aggiornamento del nome di un ward

4.1.11.5 AddPlantToWardResDto

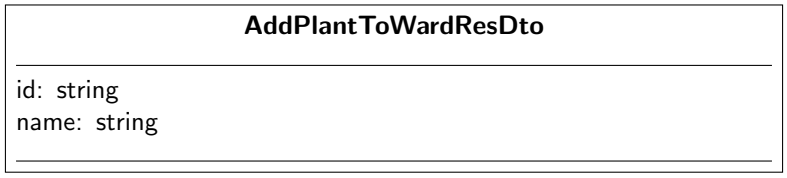

Figura 358: Diagramma della classe AddPlantToWardResDto

Descrizione: DTO di risposta restituito dopo l'associazione di un impianto a un ward

4.1.11.6 AddUserToWardResDto

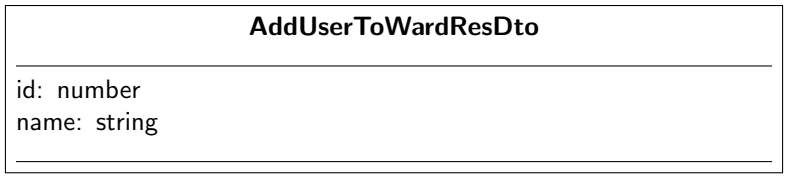

Figura 359: Diagramma della classe AddUserToWardResDto

Descrizione: DTO di risposta restituito dopo l'associazione di un utente a un ward

4.1.11.7 CreateWardResDto

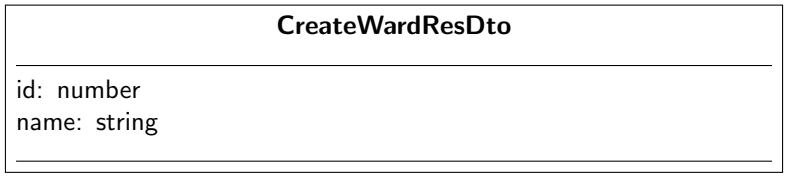

Figura 360: Diagramma della classe CreateWardResDto

Descrizione: DTO di risposta restituito dopo la creazione di un nuovo ward

4.1.11.8 FindAllPlantsByWardIdResDto

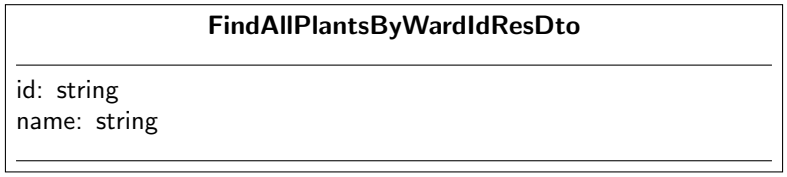

Figura 361: Diagramma della classe FindAllPlantsByWardIdResDto

Descrizione: DTO di risposta per un elemento della lista degli impianti associati a un ward

4.1.11.9 FindAllUsersByWardIdResDto

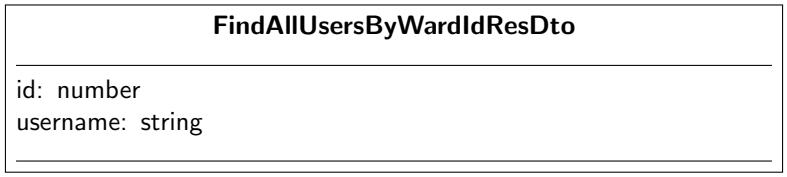

Figura 362: Diagramma della classe FindAllUsersByWardIdResDto

Descrizione: DTO di risposta per un elemento della lista degli utenti associati a un ward

4.1.11.10 FindAllWardsResDto

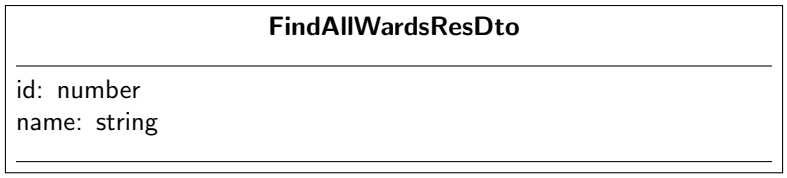

Figura 363: Diagramma della classe FindAllWardsResDto

Descrizione: DTO di risposta per un elemento della lista di tutti i ward del sistema

4.1.11.11 UpdateWardResDto

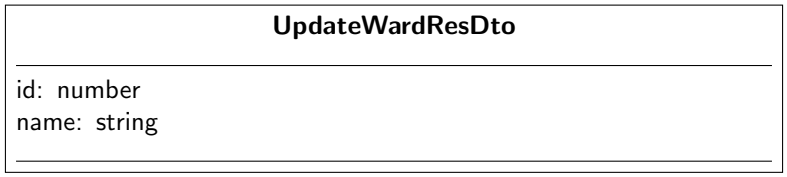

Figura 364: Diagramma della classe UpdateWardResDto

Descrizione: DTO di risposta restituito dopo l'aggiornamento di un ward

4.1.11.12 User

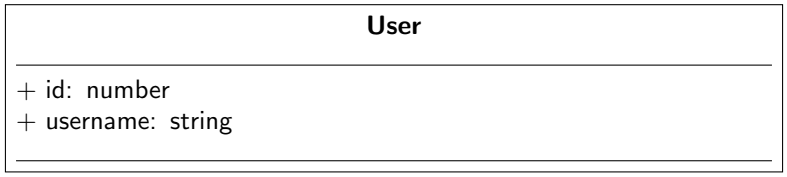

Figura 365: Diagramma della classe User

Descrizione: Modello di dominio che rappresenta un utente nel contesto del modulo Wards

4.1.11.13 Ward

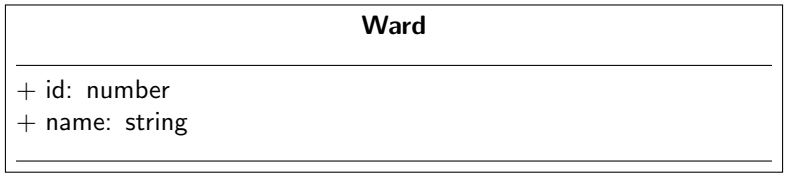

Figura 366: Diagramma della classe Ward

Descrizione: Modello di dominio che rappresenta un ward del sistema

4.1.11.14 WardsPlantsRelationshipsController

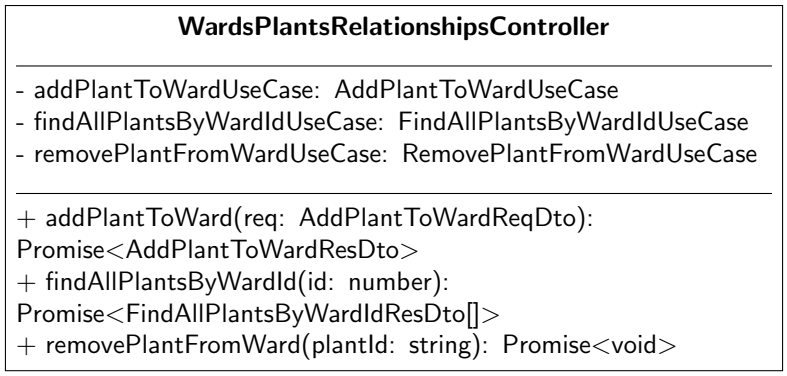

Figura 367: Diagramma della classe WardsPlantsRelationshipsController

Descrizione: Controller REST che gestisce le richieste HTTP relative alle associazioni tra ward e impianti

Descrizione dei metodi della classe:

- `addPlantToWard(req: AddPlantToWardReqDto): Promise<AddPlantToWardResDto>`: Espone l'endpoint per l'associazione di un impianto a un ward
- `findAllPlantsByWardId(id: number): Promise<FindAllPlantsByWardIdResDto[]>`: Espone l'endpoint per il recupero di tutti gli impianti associati a un ward
- `removePlantFromWard(plantId: string): Promise<void>`: Espone l'endpoint per la rimozione di un impianto da un ward

4.1.11.15 WardsUsersRelationshipsController

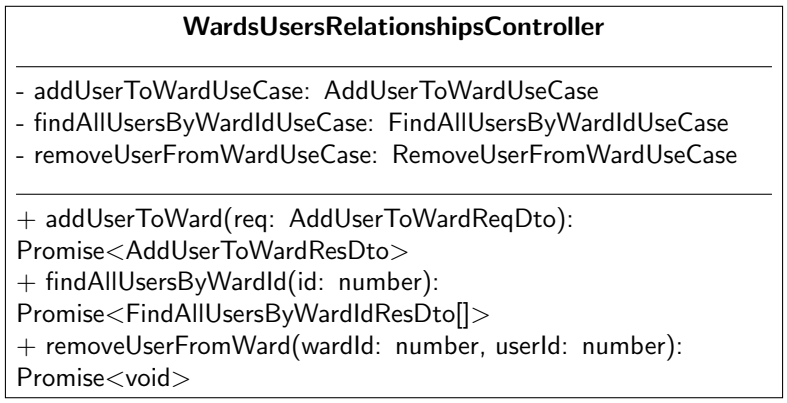

Figura 368: Diagramma della classe WardsUsersRelationshipsController

Descrizione: Controller REST che gestisce le richieste HTTP relative alle associazioni tra ward e utenti

Descrizione dei metodi della classe:

- `addUserToWard(req: AddUserToWardReqDto): Promise<AddUserToWardResDto>`: Espone l'endpoint per l'associazione di un utente a un ward
- `findAllUsersByWardId(id: number): Promise<FindAllUsersByWardIdResDto[]>`: Espone l'endpoint per il recupero di tutti gli utenti associati a un ward
- `removeUserFromWard(wardId: number, userId: number): Promise<void>`: Espone l'endpoint per la rimozione di un utente da un ward

4.1.11.16 WardsController

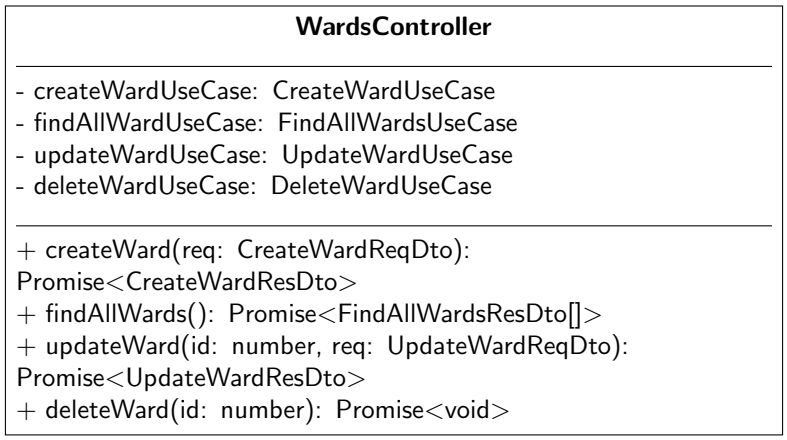

Figura 369: Diagramma della classe WardsController

Descrizione: Controller REST che gestisce le richieste HTTP relative alle operazioni CRUD sui ward

Descrizione dei metodi della classe:

- `createWard(req: CreateWardReqDto): Promise<CreateWardResDto>`: Espone l'endpoint per la creazione di un nuovo ward
- `findAllWards(): Promise<FindAllWardsResDto[]>`: Espone l'endpoint per il recupero di tutti i ward del sistema

- `updateWard(id: number, req: UpdateWardReqDto): Promise<UpdateWardResDto>`: Espone l'endpoint per l'aggiornamento del nome di un ward
- `deleteWard(id: number): Promise<void>`: Espone l'endpoint per l'eliminazione di un ward

4.1.11.17 AddPlantToWardCmd

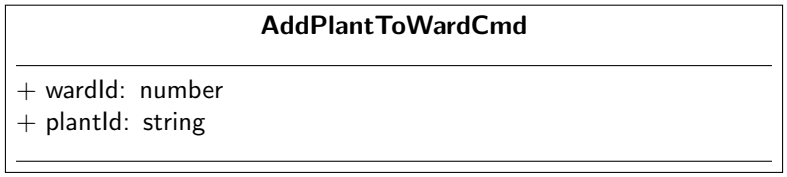

Figura 370: Diagramma della classe AddPlantToWardCmd

Descrizione: Comando per l'associazione di un impianto a un ward

4.1.11.18 AddUserToWardCmd

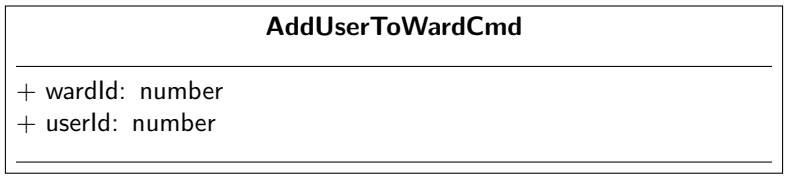

Figura 371: Diagramma della classe AddUserToWardCmd

Descrizione: Comando per l'associazione di un utente a un ward

4.1.11.19 CreateWardCmd

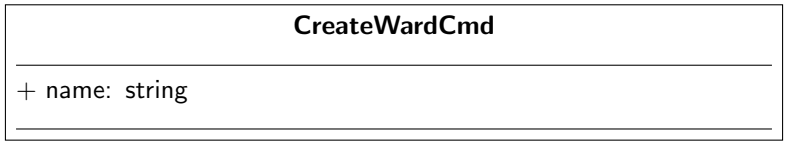

Figura 372: Diagramma della classe CreateWardCmd

Descrizione: Comando per la creazione di un nuovo ward

4.1.11.20 DeleteWardCmd

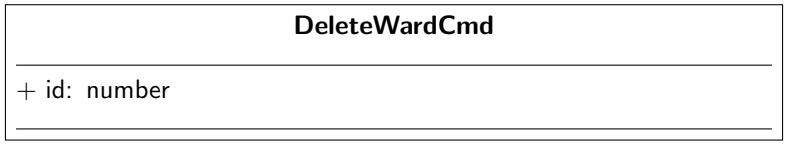

Figura 373: Diagramma della classe DeleteWardCmd

Descrizione: Comando per l'eliminazione di un ward

4.1.11.21 FindAllPlantsByWardIdCmd

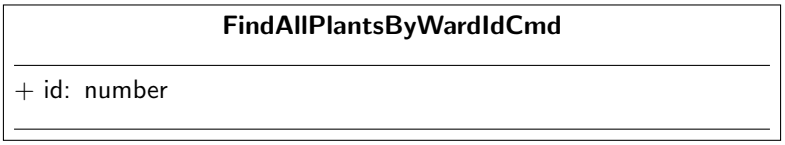

Figura 374: Diagramma della classe FindAllPlantsByWardIdCmd

Descrizione: Comando per il recupero di tutti gli impianti associati a un ward

4.1.11.22 FindAllUsersByWardIdCmd

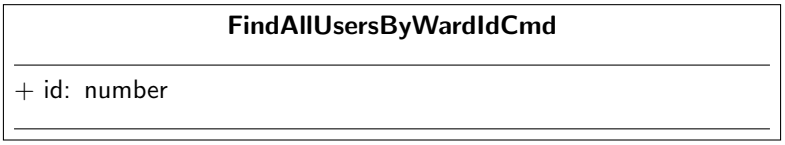

Figura 375: Diagramma della classe FindAllUsersByWardIdCmd

Descrizione: Comando per il recupero di tutti gli utenti associati a un ward

4.1.11.23 RemovePlantFromWardCmd

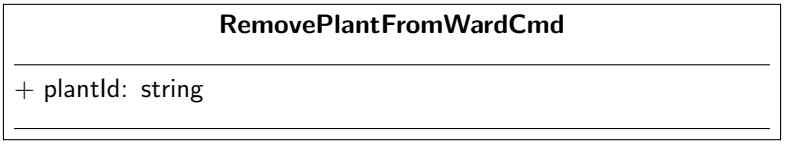

Figura 376: Diagramma della classe RemovePlantFromWardCmd

Descrizione: Comando per la rimozione di un impianto da un ward

4.1.11.24 RemoveUserFromWardCmd

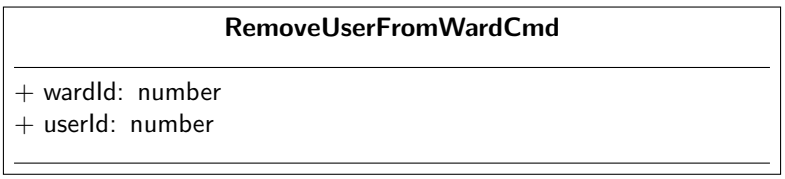

Figura 377: Diagramma della classe RemoveUserFromWardCmd

Descrizione: Comando per la rimozione di un utente da un ward

4.1.11.25 UpdateWardCmd

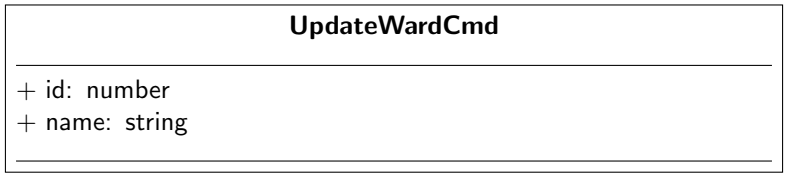

Figura 378: Diagramma della classe UpdateWardCmd

Descrizione: Comando per l'aggiornamento del nome di un ward

4.1.11.26 AddPlantToWardUseCase

Figura 379: Diagramma dell'interfaccia AddPlantToWardUseCase

Descrizione: Interfaccia del use case per l'associazione di un impianto a un ward

Descrizione dei metodi dell'interfaccia:

- `addPlantToWard(req: AddPlantToWardCmd): Promise<Plant>`: Associa un impianto al ward specificato e restituisce il modello dell'impianto aggiunto

4.1.11.27 AddUserToWardUseCase

Figura 380: Diagramma dell'interfaccia AddUserToWardUseCase

Descrizione: Interfaccia del use case per l'associazione di un utente a un ward

Descrizione dei metodi dell'interfaccia:

- `addUserToWard(req: AddUserToWardCmd): Promise<User>`: Associa un utente al ward specificato e restituisce il modello dell'utente aggiunto

4.1.11.28 CreateWardUseCase

CreateWardUseCase

+ createWard(req: CreateWardCmd): Promise<Ward>

Figura 381: Diagramma dell'interfaccia CreateWardUseCase

Descrizione: Interfaccia del use case per la creazione di un nuovo ward

Descrizione dei metodi dell'interfaccia:

- `createWard(req: CreateWardCmd): Promise<Ward>`: Crea un nuovo ward e restituisce il modello creato

4.1.11.29 DeleteWardUseCase

DeleteWardUseCase

+ deleteWard(req: DeleteWardCmd): Promise<void>

Figura 382: Diagramma dell'interfaccia DeleteWardUseCase

Descrizione: Interfaccia del use case per l'eliminazione di un ward

Descrizione dei metodi dell'interfaccia:

- `deleteWard(req: DeleteWardCmd): Promise<void>`: Elimina il ward identificato dal comando

4.1.11.30 FindAllPlantsByWardIdUseCase

FindAllPlantsByWardIdUseCase

+ findAllPlantsByWardId(req: FindAllPlantsByWardIdCmd): Promise<Plant[]>

Figura 383: Diagramma dell'interfaccia FindAllPlantsByWardIdUseCase

Descrizione: Interfaccia del use case per il recupero degli impianti di un ward

Descrizione dei metodi dell'interfaccia:

- `findAllPlantsByWardId(req: FindAllPlantsByWardIdCmd): Promise<Plant[]>`: Recupera tutti gli impianti associati al ward specificato

4.1.11.31 FindAllUsersByWardIdUseCase

FindAllUsersByWardIdUseCase

+ findAllUsersByWardId(req: FindAllUsersByWardIdCmd): Promise<User[]>

Figura 384: Diagramma dell'interfaccia FindAllUsersByWardIdUseCase

Descrizione: Interfaccia del use case per il recupero degli utenti di un ward

Descrizione dei metodi dell'interfaccia:

- `findAllUsersByWardId(req: FindAllUsersByWardIdCmd): Promise<User[]>`: Recupera tutti gli utenti associati al ward specificato

4.1.11.32 FindAllWardsUseCase

FindAllWardsUseCase

+ findAllWards(): Promise<Ward[]>

Figura 385: Diagramma dell'interfaccia FindAllWardsUseCase

Descrizione: Interfaccia del use case per il recupero di tutti i ward

Descrizione dei metodi dell'interfaccia:

- `findAllWards(): Promise<Ward[]>`: Recupera tutti i ward configurati nel sistema

4.1.11.33 RemovePlantFromWardUseCase

RemovePlantFromWardUseCase

+ removePlantFromWard(req: RemovePlantFromWardCmd): Promise<void>

Figura 386: Diagramma dell'interfaccia RemovePlantFromWardUseCase

Descrizione: Interfaccia del use case per la rimozione di un impianto da un ward

Descrizione dei metodi dell'interfaccia:

- `removePlantFromWard(req: RemovePlantFromWardCmd): Promise<void>`: Rimuove l'associazione tra un impianto e il ward a cui appartiene

4.1.11.34 RemoveUserFromWardUseCase

RemoveUserFromWardUseCase

+ removeUserFromWard(req: RemoveUserFromWardCmd): Promise<void>

Figura 387: Diagramma dell'interfaccia RemoveUserFromWardUseCase

Descrizione: Interfaccia del use case per la rimozione di un utente da un ward

Descrizione dei metodi dell'interfaccia:

- `removeUserFromWard(req: RemoveUserFromWardCmd): Promise<void>`: Rimuove l'associazione tra un utente e il ward a cui è assegnato

4.1.11.35 UpdateWardUseCase

UpdateWardUseCase

+ updateWard(req: UpdateWardCmd): Promise<Ward>

Figura 388: Diagramma dell'interfaccia UpdateWardUseCase

Descrizione: Interfaccia del use case per l'aggiornamento di un ward

Descrizione dei metodi dell'interfaccia:

- `updateWard(req: UpdateWardCmd): Promise<Ward>`: Aggiorna il nome del ward e restituisce il modello aggiornato

4.1.11.36 WardService

WardService

- findAllWardsPort: FindAllWardsPort

+ createWard(req: CreateWardCmd): Promise<Ward>
+ findAllWards(): Promise<Ward[]>
+ updateWard(req: UpdateWardCmd): Promise<Ward>
+ deleteWard(req: DeleteWardCmd): Promise<void>

Figura 389: Diagramma della classe WardService

Descrizione: Servizio applicativo che implementa i use case relativi alle operazioni CRUD sui ward

Descrizione dei metodi della classe:

- `createWard(req: CreateWardCmd): Promise<Ward>`: Crea un nuovo ward delegando alla porta di persistenza
- `findAllWards(): Promise<Ward[]>`: Recupera tutti i ward delegando alla porta di persistenza
- `updateWard(req: UpdateWardCmd): Promise<Ward>`: Aggiorna un ward delegando alla porta di persistenza
- `deleteWard(req: DeleteWardCmd): Promise<void>`: Elimina un ward delegando alla porta di persistenza

4.1.11.37 WardsPlantsRelationshipsService

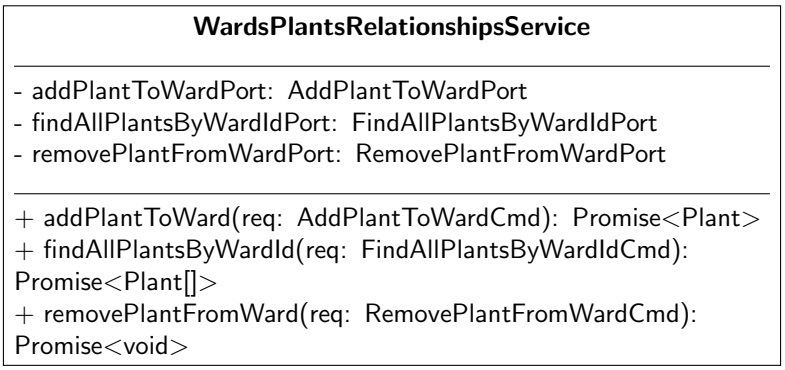

Figura 390: Diagramma della classe WardsPlantsRelationshipsService

Descrizione: Servizio applicativo che implementa i use case relativi alle associazioni tra ward e impianti

Descrizione dei metodi della classe:

- `addPlantToWard(req: AddPlantToWardCmd): Promise<Plant>`: Associa un impianto a un ward delegando alla porta di persistenza
- `findAllPlantsByWardId(req: FindAllPlantsByWardIdCmd): Promise<Plant[]>`: Recupera gli impianti di un ward delegando alla porta di persistenza
- `removePlantFromWard(req: RemovePlantFromWardCmd): Promise<void>`: Rimuove un impianto da un ward delegando alla porta di persistenza

4.1.11.38 WardsUsersRelationshipsService

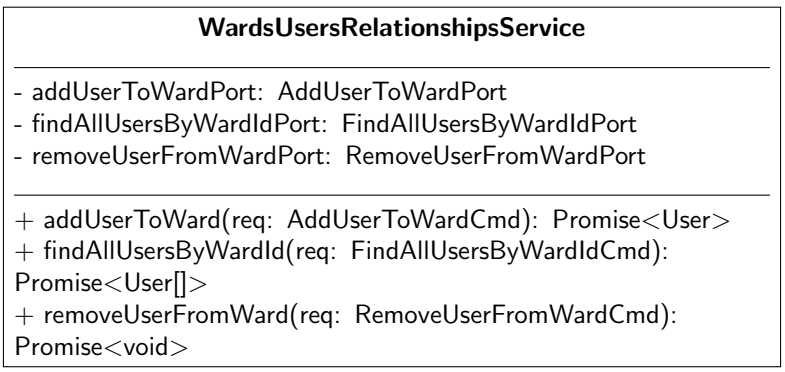

Figura 391: Diagramma della classe WardsUsersRelationshipsService

Descrizione: Servizio applicativo che implementa i use case relativi alle associazioni tra ward e utenti

Descrizione dei metodi della classe:

- `addUserToWard(req: AddUserToWardCmd): Promise<User>`: Associa un utente a un ward delegando alla porta di persistenza
- `findAllUsersByWardId(req: FindAllUsersByWardIdCmd): Promise<User[]>`: Recupera gli utenti di un ward delegando alla porta di persistenza
- `removeUserFromWard(req: RemoveUserFromWardCmd): Promise<void>`: Rimuove un utente da un ward delegando alla porta di persistenza

4.1.11.39 AddPlantToWardPort

AddPlantToWardPort

+ addPlantToWard(req: AddPlantToWardCmd): Promise<Plant>

Figura 392: Diagramma dell'interfaccia AddPlantToWardPort

Descrizione: Porta di uscita per l'associazione di un impianto a un ward

Descrizione dei metodi dell'interfaccia:

- `addPlantToWard(req: AddPlantToWardCmd): Promise<Plant>`: Persiste l'associazione tra un impianto e un ward e restituisce il modello dell'impianto

4.1.11.40 AddUserToWardPort

AddUserToWardPort

+ addUserToWard(req: AddUserToWardCmd): Promise<User>

Figura 393: Diagramma dell'interfaccia AddUserToWardPort

Descrizione: Porta di uscita per l'associazione di un utente a un ward

Descrizione dei metodi dell'interfaccia:

- `addUserToWard(req: AddUserToWardCmd): Promise<User>`: Persiste l'associazione tra un utente e un ward e restituisce il modello dell'utente

4.1.11.41 CreateWardPort

CreateWardPort

+ createWard(req: CreateWardCmd): Promise<Ward>

Figura 394: Diagramma dell'interfaccia CreateWardPort

Descrizione: Porta di uscita per la creazione e persistenza di un nuovo ward

Descrizione dei metodi dell'interfaccia:

- `createWard(req: CreateWardCmd): Promise<Ward>`: Persiste il nuovo ward e restituisce il modello creato

4.1.11.42 DeleteWardPort

DeleteWardPort

+ deleteWard(req: DeleteWardCmd): Promise<void>

Figura 395: Diagramma dell'interfaccia DeleteWardPort

Descrizione: Porta di uscita per l'eliminazione di un ward

Descrizione dei metodi dell'interfaccia:

- `deleteWard(req: DeleteWardCmd): Promise<void>`: Elimina il ward specificato dal livello di persistenza

4.1.11.43 FindAllPlantsByWardIdPort

FindAllPlantsByWardIdPort

+ findAllPlantsByWardId(req: FindAllPlantsByWardIdCmd): Promise<Plant[]>

Figura 396: Diagramma dell'interfaccia FindAllPlantsByWardIdPort

Descrizione: Porta di uscita per il recupero degli impianti di un ward

Descrizione dei metodi dell'interfaccia:

- `findAllPlantsByWardId(req: FindAllPlantsByWardIdCmd): Promise<Plant[]>`: Recupera dal livello di persistenza tutti gli impianti associati al ward specificato

4.1.11.44 FindAllUsersByWardIdPort

FindAllUsersByWardIdPort

+ findAllUsersByWardId(req: FindAllUsersByWardIdCmd): Promise<User[]>

Figura 397: Diagramma dell'interfaccia FindAllUsersByWardIdPort

Descrizione: Porta di uscita per il recupero degli utenti di un ward

Descrizione dei metodi dell'interfaccia:

- `findAllUsersByWardId(req: FindAllUsersByWardIdCmd): Promise<User[]>`: Recupera dal livello di persistenza tutti gli utenti associati al ward specificato

4.1.11.45 FindAllWardsPort

FindAllWardsPort

+ findAllWards(): Promise<Ward[]>

Figura 398: Diagramma dell'interfaccia FindAllWardsPort

Descrizione: Porta di uscita per il recupero di tutti i ward

Descrizione dei metodi dell'interfaccia:

- `findAllWards(): Promise<Ward[]>`: Recupera dal livello di persistenza tutti i ward del sistema

4.1.11.46 RemovePlantFromWardPort

RemovePlantFromWardPort

+ removePlantFromWard(req: RemovePlantFromWardCmd): Promise<void>

Figura 399: Diagramma dell'interfaccia RemovePlantFromWardPort

Descrizione: Porta di uscita per la rimozione di un impianto da un ward

Descrizione dei metodi dell'interfaccia:

- `removePlantFromWard(req: RemovePlantFromWardCmd): Promise<void>`: Elimina dal livello di persistenza l'associazione tra l'impianto e il suo ward

4.1.11.47 RemoveUserFromWardPort

RemoveUserFromWardPort

+ removeUserFromWard(req: RemoveUserFromWardCmd): Promise<void>

Figura 400: Diagramma dell'interfaccia RemoveUserFromWardPort

Descrizione: Porta di uscita per la rimozione di un utente da un ward

Descrizione dei metodi dell'interfaccia:

- `removeUserFromWard(req: RemoveUserFromWardCmd): Promise<void>`: Elimina dal livello di persistenza l'associazione tra l'utente e il ward

4.1.11.48 UpdateWardPort

UpdateWardPort

+ updateWard(req: UpdateWardCmd): Promise<Ward>

Figura 401: Diagramma dell'interfaccia UpdateWardPort

Descrizione: Porta di uscita per l'aggiornamento di un ward

Descrizione dei metodi dell'interfaccia:

- `updateWard(req: UpdateWardCmd): Promise<Ward>`: Aggiorna il ward nel livello di persistenza e restituisce il modello aggiornato

4.1.11.49 PlantEntity

PlantEntity

+ id: string
+ name: string

Figura 402: Diagramma della classe PlantEntity

Descrizione: Entità di persistenza che rappresenta un impianto nel contesto del modulo Wards

4.1.11.50 UserEntity

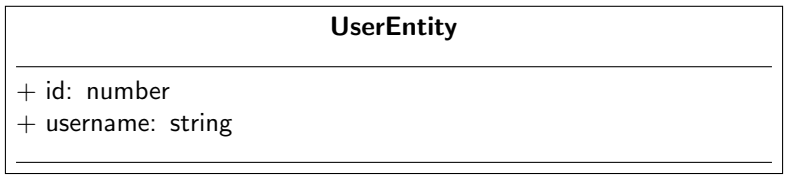

Figura 403: Diagramma della classe UserEntity

Descrizione: Entità di persistenza che rappresenta un utente nel contesto del modulo Wards

4.1.11.51 WardEntity

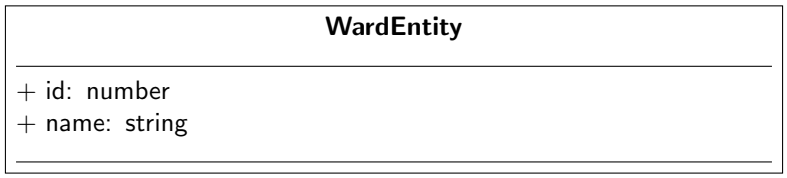

Figura 404: Diagramma della classe WardEntity

Descrizione: Entità di persistenza che rappresenta un ward nel database

4.1.11.52 WardsPersistenceAdapter

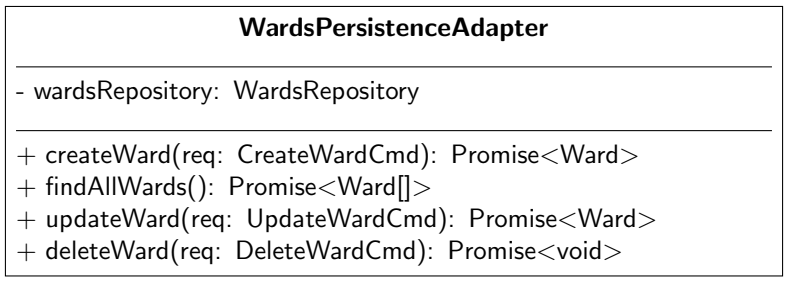

Figura 405: Diagramma della classe WardsPersistenceAdapter

Descrizione: Adattatore di persistenza che implementa le porte relative alle operazioni CRUD sui ward

Descrizione dei metodi della classe:

- `createWard(req: CreateWardCmd): Promise<Ward>`: Delega al repository la creazione del ward e mappa il risultato nel modello di dominio
- `findAllWards(): Promise<Ward[]>`: Recupera e mappa tutti i ward dal repository
- `updateWard(req: UpdateWardCmd): Promise<Ward>`: Delega al repository l'aggiornamento del ward e mappa il risultato nel modello di dominio
- `deleteWard(req: DeleteWardCmd): Promise<void>`: Delega al repository l'eliminazione del ward specificato

4.1.11.53 WardsUsersRelationshipsPersistenceAdapter

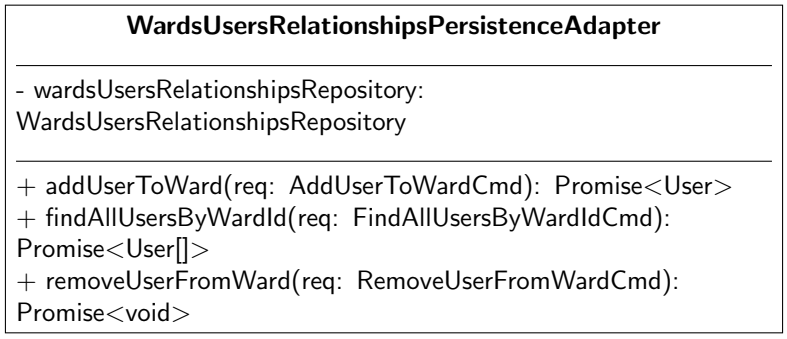

Figura 406: Diagramma della classe WardsUsersRelationshipsPersistenceAdapter

Descrizione: Adattatore di persistenza che implementa le porte relative alle associazioni tra ward e utenti

Descrizione dei metodi della classe:

- `addUserToWard(req: AddUserToWardCmd): Promise<User>`: Delega al repository la creazione dell'associazione e mappa il risultato nel modello di dominio
- `findAllUsersByWardId(req: FindAllUsersByWardIdCmd): Promise<User[]>`: Recupera e mappa gli utenti del ward dal repository
- `removeUserFromWard(req: RemoveUserFromWardCmd): Promise<void>`: Delega al repository la rimozione dell'associazione tra utente e ward

4.1.11.54 WardsPlantsRelationshipsPersistenceAdapter

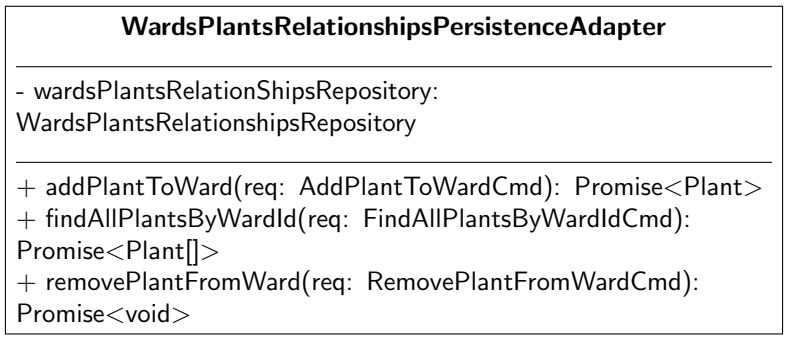

Figura 407: Diagramma della classe WardsPlantsRelationshipsPersistenceAdapter

Descrizione: Adattatore di persistenza che implementa le porte relative alle associazioni tra ward e impianti

Descrizione dei metodi della classe:

- `addPlantToWard(req: AddPlantToWardCmd): Promise<Plant>`: Delega al repository la creazione dell'associazione e mappa il risultato nel modello di dominio
- `findAllPlantsByWardId(req: FindAllPlantsByWardIdCmd): Promise<Plant[]>`: Recupera e mappa gli impianti del ward dal repository
- `removePlantFromWard(req: RemovePlantFromWardCmd): Promise<void>`: Delega al repository la rimozione dell'associazione tra impianto e ward

4.1.11.55 WardsUsersRelationshipsRepository

WardsUsersRelationshipsRepository

+ addUserToWard(wardId: number, userId: number): Promise<UserEntity>
+ findAllUsersByWardId(wardId: number): Promise<UserEntity[]>
+ removeUserFromWard(wardId: number, userId: number): Promise<void>

Figura 408: Diagramma dell'interfaccia WardsUsersRelationshipsRepository

Descrizione: Interfaccia del repository per le operazioni di persistenza sulle associazioni tra ward e utenti

Descrizione dei metodi dell'interfaccia:

- `addUserToWard(wardId: number, userId: number): Promise<UserEntity>`: Inserisce nel database l'associazione tra il ward e l'utente specificati
- `findAllUsersByWardId(wardId: number): Promise<UserEntity[]>`: Recupera dal database tutti gli utenti associati al ward specificato
- `removeUserFromWard(wardId: number, userId: number): Promise<void>`: Elimina dal database l'associazione tra il ward e l'utente specificati

4.1.11.56 WardsRepository

WardsRepository

+ createWard(name: string): Promise<WardEntity>
+ findAllWards(): Promise<WardEntity[]>
+ updateWard(id: number, name: string): Promise<WardEntity>
+ deleteWard(id: number): Promise<void>

Figura 409: Diagramma dell'interfaccia WardsRepository

Descrizione: Interfaccia del repository per le operazioni di persistenza sui ward

Descrizione dei metodi dell'interfaccia:

- `createWard(name: string): Promise<WardEntity>`: Inserisce un nuovo ward nel database con il nome fornito
- `findAllWards(): Promise<WardEntity[]>`: Recupera dal database tutti i ward del sistema
- `updateWard(id: number, name: string): Promise<WardEntity>`: Aggiorna nel database il nome del ward specificato
- `deleteWard(id: number): Promise<void>`: Elimina dal database il ward con l'identificativo specificato

4.1.11.57 WardsPlantsRelationshipsRepository

WardsPlantsRelationshipsRepository

+ addPlantToWard(wardId: number, plantId: string): Promise<PlantEntity>
+ findAllPlantsByWardId(wardId: number): Promise<PlantEntity[]>
+ removePlantFromWard(plantId: string): Promise<void>

Figura 410: Diagramma dell'interfaccia WardsPlantsRelationshipsRepository

Descrizione: Interfaccia del repository per le operazioni di persistenza sulle associazioni tra ward e impianti

Descrizione dei metodi dell'interfaccia:

- `addPlantToWard(wardId: number, plantId: string): Promise<PlantEntity>`: Inserisce nel database l'associazione tra il ward e l'impianto specificati
- `findAllPlantsByWardId(wardId: number): Promise<PlantEntity[]>`: Recupera dal database tutti gli impianti associati al ward specificato
- `removePlantFromWard(plantId: string): Promise<void>`: Elimina dal database l'associazione dell'impianto con il suo ward corrente

4.1.11.58 WardsPlantsRelationshipsRepositoryImpl

WardsPlantsRelationshipsRepositoryImpl

+ addPlantToWard(wardId: number, plantId: string): Promise<PlantEntity>
+ findAllPlantsByWardId(wardId: number): Promise<PlantEntity[]>
+ removePlantFromWard(plantId: string): Promise<void>

Figura 411: Diagramma della classe WardsPlantsRelationshipsRepositoryImpl

Descrizione: Implementazione concreta del repository per le associazioni tra ward e impianti

Descrizione dei metodi della classe:

- `addPlantToWard(wardId: number, plantId: string): Promise<PlantEntity>`: Implementa l'inserimento dell'associazione tra ward e impianto tramite query al database
- `findAllPlantsByWardId(wardId: number): Promise<PlantEntity[]>`: Implementa il recupero degli impianti di un ward tramite query al database
- `removePlantFromWard(plantId: string): Promise<void>`: Implementa la rimozione dell'associazione impianto-ward tramite query al database

4.1.11.59 WardsRepositoryImpl

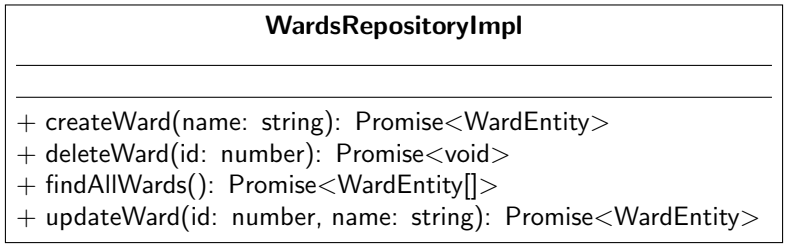

Figura 412: Diagramma della classe WardsRepositoryImpl

Descrizione: Implementazione concreta del repository per i ward

Descrizione dei metodi della classe:

- `createWard(name: string): Promise<WardEntity>`: Implementa l'inserimento di un nuovo ward nel database
- `deleteWard(id: number): Promise<void>`: Implementa l'eliminazione di un ward tramite query al database
- `findAllWards(): Promise<WardEntity[]>`: Implementa il recupero di tutti i ward tramite query al database
- `updateWard(id: number, name: string): Promise<WardEntity>`: Implementa l'aggiornamento del nome di un ward tramite query al database

4.1.11.60 WardsUsersRelationshipsRepositoryImpl

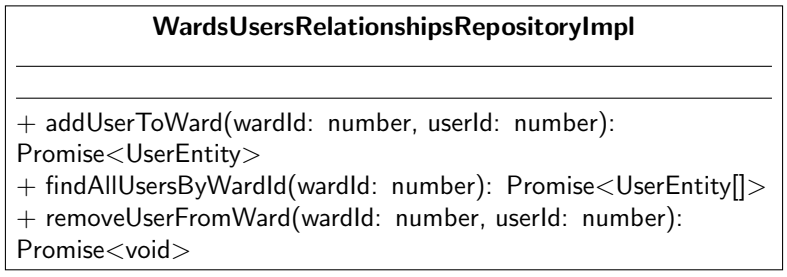

Figura 413: Diagramma della classe WardsUsersRelationshipsRepositoryImpl

Descrizione: Implementazione concreta del repository per le associazioni tra ward e utenti

Descrizione dei metodi della classe:

- `addUserToWard(wardId: number, userId: number): Promise<UserEntity>`: Implementa l'inserimento dell'associazione tra ward e utente tramite query al database
- `findAllUsersByWardId(wardId: number): Promise<UserEntity[]>`: Implementa il recupero degli utenti di un ward tramite query al database
- `removeUserFromWard(wardId: number, userId: number): Promise<void>`: Implementa la rimozione dell'associazione utente-ward tramite query al database

4.2 Frontend

Il *frontend* è composto dai seguenti moduli:

- **Alarm-configuration**: gestisce l'interfaccia per la creazione, la modifica, l'abilitazione, la disabilitazione e l'eliminazione delle regole di allarme, riservata all'Amministratore;
- **Alarm-history**: fornisce la visualizzazione dello storico degli allarmi risolti, con i dettagli relativi a orario di scatto, risoluzione e gestore;
- **Alarm-management**: espone la vista degli allarmi attivi, consentendo all'utente di visualizzarli e prenderne in carico la risoluzione;
- **Analytics**: espone i grafici statistici relativi all'appartamento selezionato, inclusi consumi energetici, presenze, temperature, allarmi e suggerimenti per il risparmio energetico;
- **Apartment-monitor**: fornisce la visualizzazione dello stato di un appartamento, con la panoramica delle stanze e dei dispositivi presenti;
- **Core**: mantiene elementi di base utilizzati dalle features;
- **Dashboard**: fornisce il cruscotto informativo principale, aggregando il modulo di gestione allarmi, le statistiche e i grafici di analisi consumi e la panoramica dell'appartamento monitorato.
- **Device-interaction**: gestisce l'interfaccia per la visualizzazione e la modifica dello stato dei dispositivi IoT presenti nelle stanze di un appartamento;
- **MainLayout**: definisce la struttura principale dell'applicativo, includendo il menu di navigazione e il contenitore delle viste;
- **MyVimar-integration**: gestisce l'interfaccia per il collegamento e la visualizzazione dello stato dell'account MyVimar associato al Sistema;
- **Notifications**: fornisce la visualizzazione e la gestione delle notifiche ricevute dal Sistema in seguito allo scatto di un allarme;
- **Shared**: contiene entità condivise più moduli;
- **User-authentication**: gestisce le schermate di accesso al Sistema, incluso il flusso di primo accesso con cambio della password temporanea;
- **User-management**: fornisce l'interfaccia per la creazione, la visualizzazione e l'eliminazione degli Operatori Sanitari, riservata all'Amministratore;
- **Ward-management**: gestisce la lista dei reparti le relazioni di appartenenza tra reparti, appartamenti e operatori sanitari, è riservata all'amministratore.

4.2.1 Alarm-configuration

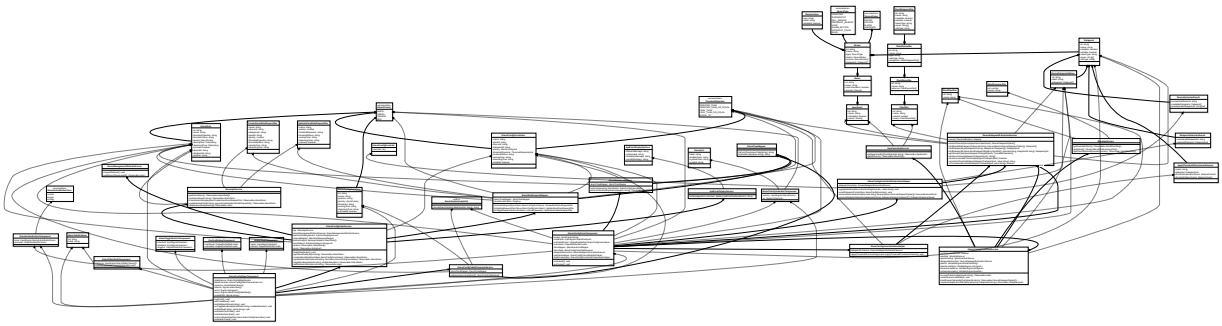

Figura 414: Diagramma delle classi del modulo Alarm-configuration

4.2.1.1 AlarmConfigFormComponent

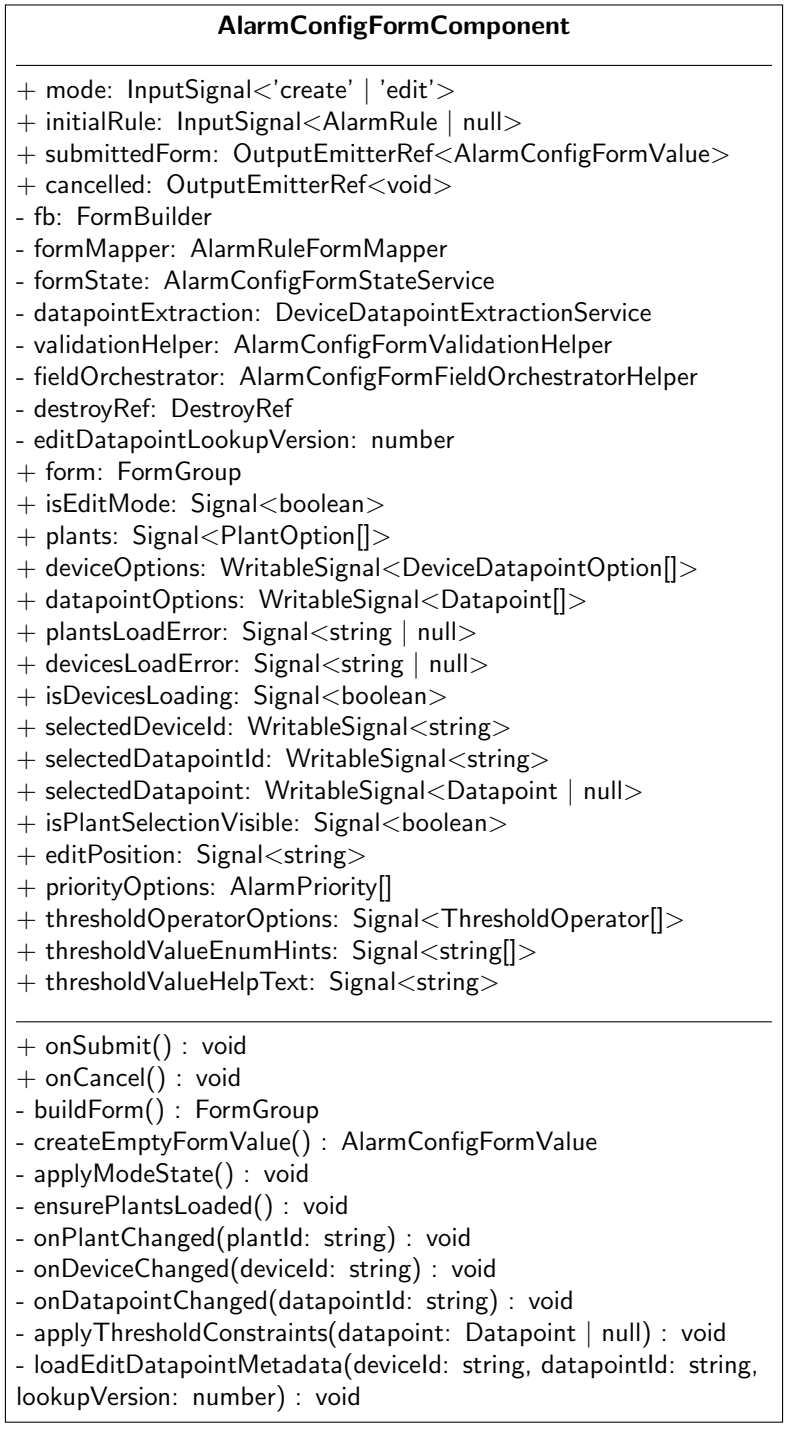

Figura 415: Diagramma della classe AlarmConfigFormComponent

Descrizione: Componente Angular che gestisce il form reattivo per la creazione e la modifica di una regola di allarme. Coordina la selezione a cascata di impianto, dispositivo e datapoint, delegando la validazione e l'orchestrazione dei campi ai relativi servizi helper.

Descrizione dei metodi della classe:

- `onSubmit() : void`: Valida il form, normalizza il valore di soglia se necessario ed emette i dati tramite l'evento submittedForm.
- `onCancel() : void`: Emette l'evento cancelled per segnalare la rinuncia alla compilazione del form.
- `buildForm() : FormGroup`: Costruisce e restituisce il form reattivo con tutti i campi e i relativi validatori.
- `createEmptyFormValue() : AlarmConfigFormValue`: Restituisce il valore iniziale vuoto del form da utilizzare in modalità creazione.
- `applyModeState() : void`: Delega all'orchestratore l'abilitazione o disabilitazione dei campi del form in base alla modalità corrente.
- `ensurePlantsLoaded() : void`: Delega al servizio di stato il caricamento degli impianti disponibili se non ancora caricati.
- `onPlantChanged(plantId: string) : void`: Gestisce il cambio di impianto selezionato, resettando le opzioni dispositivo e caricando quelle relative al nuovo impianto.
- `onDeviceChanged(deviceId: string) : void`: Gestisce il cambio di dispositivo selezionato, aggiornando le opzioni datapoint e resettando i vincoli di soglia.
- `onDatapointChanged(datapointId: string) : void`: Gestisce il cambio di datapoint selezionato, aggiornando il datapoint corrente e i vincoli di soglia.
- `applyThresholdConstraints(datapoint: Datapoint | null) : void`: Delega all'helper di validazione l'applicazione dei vincoli sui campi di soglia in base al datapoint selezionato.
- `loadEditDatapointMetadata(deviceId: string, datapointId: string, lookupVersion: number) : void`: Carica i metadati del datapoint associato alla regola in modalità modifica, aggiornando il datapoint selezionato e i vincoli di soglia.

4.2.1.2 AlarmConfigPageComponent

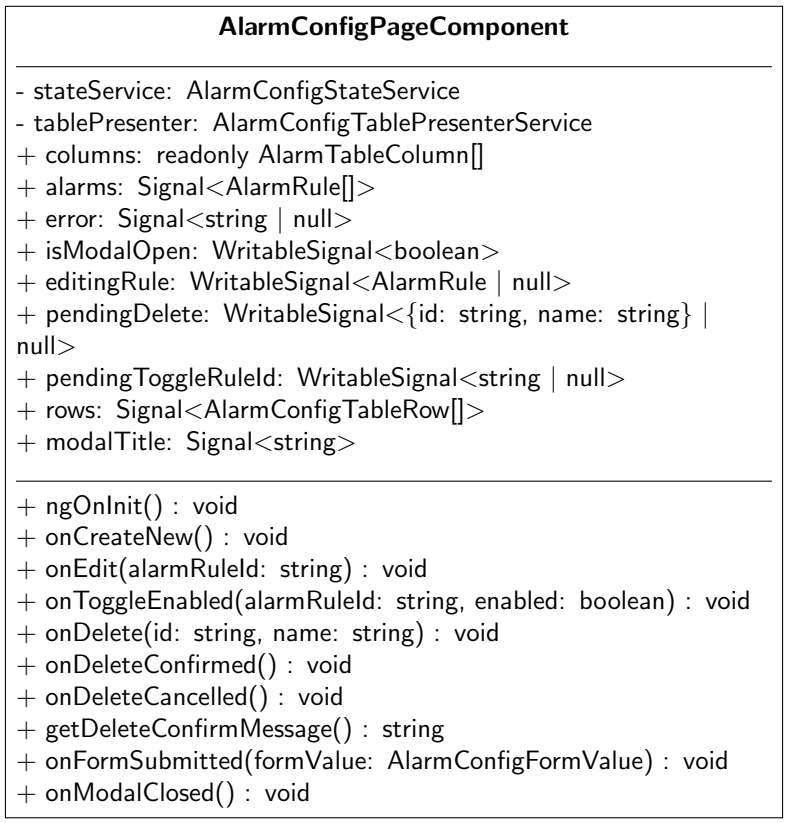

Figura 416: Diagramma della classe AlarmConfigPageComponent

Descrizione: Componente Angular che rappresenta la pagina di configurazione delle regole di allarme, con supporto alla creazione, modifica, eliminazione e attivazione delle regole. Coordina l'apertura del modal di form e i dialog di conferma tramite segnali reattivi.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Avvia il caricamento delle regole di allarme tramite il servizio di stato.
- `onCreateNew() : void`: Reimposta la regola in modifica e apre il modal per la creazione di una nuova regola.
- `onEdit(alarmRuleId: string) : void`: Recupera la regola corrispondente all'identificativo specificato e apre il modal in modalità modifica.
- `onToggleEnabled(alarmRuleId: string, enabled: boolean) : void`: Attiva o disattiva la regola specificata tramite il servizio di stato, prevenendo richieste concorrenti.
- `onDelete(id: string, name: string) : void`: Imposta la regola candidata all'eliminazione per la richiesta di conferma.
- `onDeleteConfirmed() : void`: Conferma l'eliminazione della regola in attesa e reimposta il segnale pendingDelete.
- `onDeleteCancelled() : void`: Annulla l'eliminazione in corso reimpostando il segnale pendingDelete a null.
- `getDeleteConfirmMessage() : string`: Restituisce il messaggio di conferma per l'eliminazione della regola in attesa.
- `onFormSubmitted(formValue: AlarmConfigFormValue) : void`: Crea o aggiorna la regola di allarme in base alla modalità corrente del form, chiudendo il modal al termine.

- `onModalClosed() : void`: Chiude il modal e reimposta la regola in modifica a null.

4.2.1.3 AlarmConfigFormFieldOrchestratorHelper

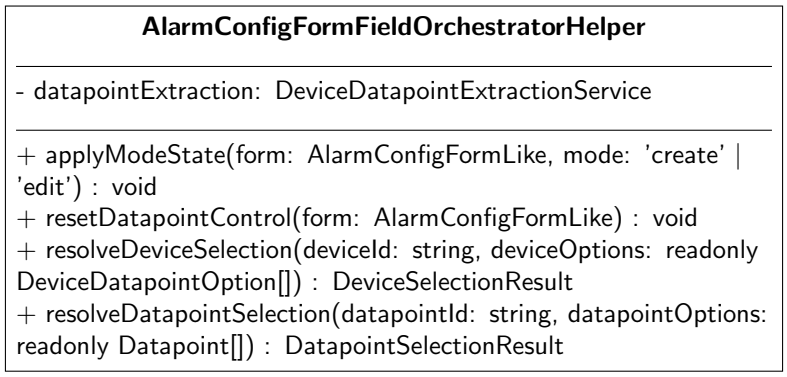

Figura 417: Diagramma della classe AlarmConfigFormFieldOrchestratorHelper

Descrizione: Helper Angular per l'orchestrazione dei campi del form di configurazione allarme. Gestisce l'abilitazione e la validazione dei controlli in base alla modalità e risolve le selezioni a cascata di dispositivo e datapoint.

Descrizione dei metodi della classe:

- `applyModeState(form: AlarmConfigFormLike, mode: 'create' | 'edit') : void`: Abilita o disabilita i campi del form e aggiorna i validatori in base alla modalità corrente.
- `resetDatapointControl(form: AlarmConfigFormLike) : void`: Reimposta il controllo datapointId a stringa vuota e ne aggiorna la validità.
- `resolveDeviceSelection(deviceId: string, deviceOptions: readonly DeviceDatapointOption[]) : DeviceSelectionResult`: Risolve la selezione del dispositivo restituendo i datapoint leggibili e l'eventuale datapoint auto-selezionato se unico.
- `resolveDatapointSelection(datapointId: string, datapointOptions: readonly Datapoint[]) : DatapointSelectionResult`: Risolve la selezione del datapoint restituendo l'identificativo normalizzato e il datapoint corrispondente se trovato.

4.2.1.4 AlarmConfigFormValidationHelper

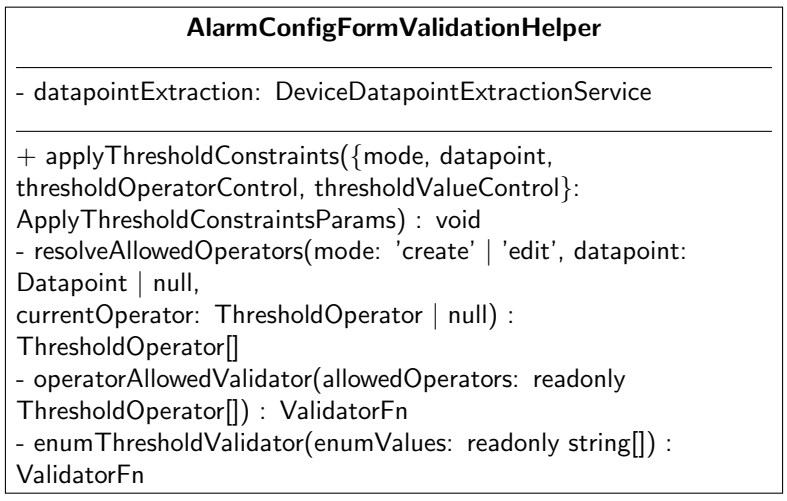

Figura 418: Diagramma della classe AlarmConfigFormValidationHelper

Descrizione: Helper Angular per la gestione della validazione dei campi di soglia nel form di configurazione allarme. Applica dinamicamente i validatori appropriati in base al datapoint selezionato e alla modalità del form.

Descrizione dei metodi della classe:

- `applyThresholdConstraints({mode, datapoint, thresholdOperatorControl, thresholdValueControl}: ApplyThresholdConstraintsParams) : void`: Applica i validatori ai controlli di operatore e valore di soglia in base alla modalità e al datapoint selezionato, resettando l'operatore se non più consentito.
- `resolveAllowedOperators(mode: 'create' | 'edit', datapoint: Datapoint | null, currentOperator: ThresholdOperator | null) : ThresholdOperator[]`: Restituisce la lista degli operatori consentiti in base al datapoint selezionato e alla modalità corrente.
- `operatorAllowedValidator(allowedOperators: readonly ThresholdOperator[]) : ValidatorFn`: Restituisce un validatore che verifica se l'operatore selezionato è tra quelli consentiti.
- `enumThresholdValidator(enumValues: readonly string[]) : ValidatorFn`: Restituisce un validatore che verifica se il valore di soglia inserito appartiene ai valori enumerati previsti dal datapoint.

4.2.1.5 AlarmRuleFormMapper

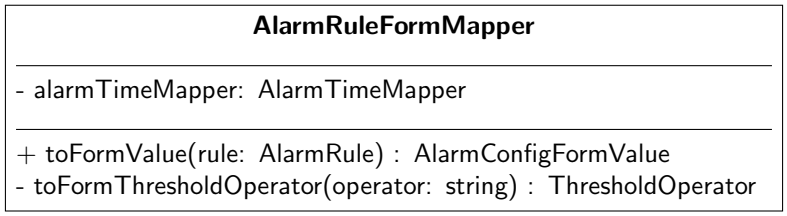

Figura 419: Diagramma della classe AlarmRuleFormMapper

Descrizione: Mapper Angular per la trasformazione di una regola di allarme nel formato atteso dal form di configurazione. Delega la conversione degli orari all'AlarmTimeMapper e gestisce la mappatura degli operatori di soglia.

Descrizione dei metodi della classe:

- `toFormValue(rule: AlarmRule) : AlarmConfigFormValue`: Trasforma una regola di allarme nel corrispondente valore del form di configurazione.
- `toFormThresholdOperator(operator: string) : ThresholdOperator`: Converte una stringa rappresentante un operatore di soglia nell'enumerato ThresholdOperator corrispondente, lanciando un errore se non riconosciuto.

4.2.1.6 AlarmRuleRequestMapper

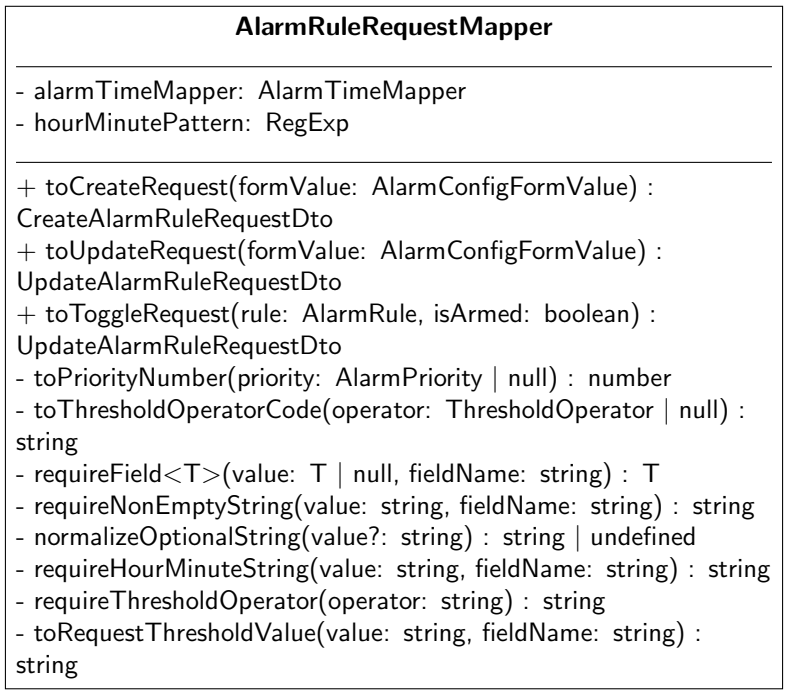

Figura 420: Diagramma della classe AlarmRuleRequestMapper

Descrizione: Mapper Angular per la costruzione dei DTO di creazione, aggiornamento e toggle delle regole di allarme a partire dai valori del form o da una regola esistente. Gestisce la validazione e la normalizzazione di tutti i campi obbligatori prima della serializzazione.

Descrizione dei metodi della classe:

- `toCreateRequest(formValue: AlarmConfigFormValue) : CreateAlarmRuleRequestDto`: Trasforma il valore del form nel DTO di richiesta per la creazione di una nuova regola di allarme.
- `toUpdateRequest(formValue: AlarmConfigFormValue) : UpdateAlarmRuleRequestDto`: Trasforma il valore del form nel DTO di richiesta per l'aggiornamento di una regola di allarme esistente.
- `toToggleRequest(rule: AlarmRule, isArmed: boolean) : UpdateAlarmRuleRequestDto`: Costruisce il DTO di aggiornamento per la sola modifica dello stato di armamento di una regola esistente.
- `toPriorityNumber(priority: AlarmPriority | null) : number`: Estrae il valore numerico della priorità, lanciando un errore se null.
- `toThresholdOperatorCode(operator: ThresholdOperator | null) : string`: Converte l'enumerato ThresholdOperator nel codice stringa corrispondente, lanciando un errore se non riconosciuto.
- `requireField<T>(value: T | null, fieldName: string) : T`: Verifica che il valore non sia null, lanciando un errore con il nome del campo se mancante.
- `requireNonEmptyString(value: string, fieldName: string) : string`: Verifica che la stringa non sia vuota dopo il trim, lanciando un errore con il nome del campo se vuota.
- `normalizeOptionalString(value?: string) : string | undefined`: Normalizza una stringa opzionale restituendo undefined se vuota dopo il trim.
- `requireHourMinuteString(value: string, fieldName: string) : string`: Verifica che la stringa rispetti il formato HH:MM, lanciando un errore se non conforme.

- `requireThresholdOperator(operator: string) : string`: Verifica che la stringa sia un operatore di soglia valido, lanciando un errore altrimenti.
- `toRequestThresholdValue(value: string, fieldName: string) : string`: Normalizza il valore di soglia convertendo le varianti booleane nei valori canonici on/off.

4.2.1.7 AlarmTimeMapper

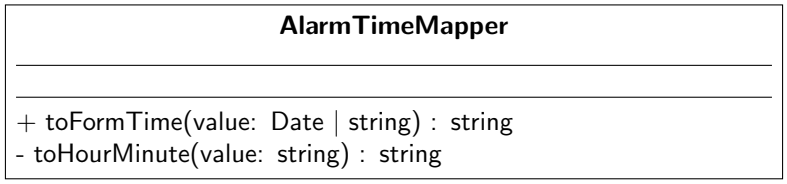

Figura 421: Diagramma della classe AlarmTimeMapper

Descrizione: Mapper Angular per la conversione di valori temporali nel formato HH:MM richiesto dal form di configurazione allarme. Gestisce diversi formati di input tra cui Date, ISO 8601, HH:MM e HH:MM:SS.

Descrizione dei metodi della classe:

- `toFormTime(value: Date | string) : string`: Converte un valore di tipo Date o stringa nel formato orario HH:MM utilizzato dal form, gestendo i formati ISO, HH:MM e HH:MM:SS.
- `toHourMinute(value: string) : string`: Estrae i componenti ora e minuto da una stringa in formato ISO o la tronca ai primi cinque caratteri se il pattern non corrisponde.

4.2.1.8 AlarmConfigFormValue

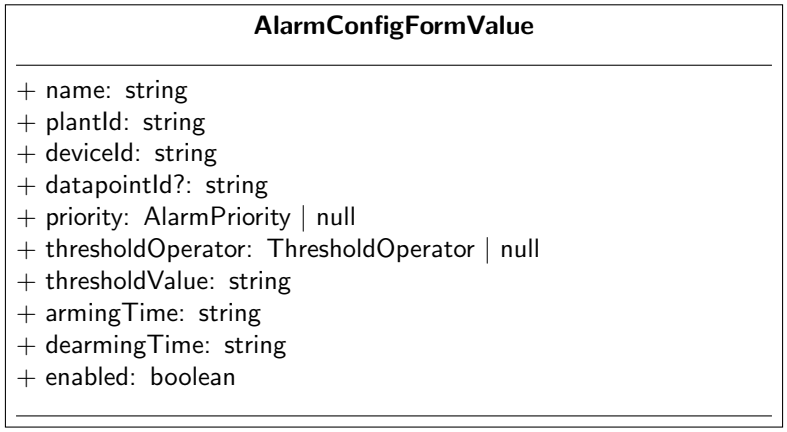

Figura 422: Diagramma della classe AlarmConfigFormValue

Descrizione: Interfaccia che rappresenta il valore del form di configurazione di una regola di allarme. Raccoglie tutti i campi necessari per la creazione e la modifica di una regola, inclusi dispositivo, datapoint, soglia e orari di armamento.

4.2.1.9 AlarmConfigTableRow

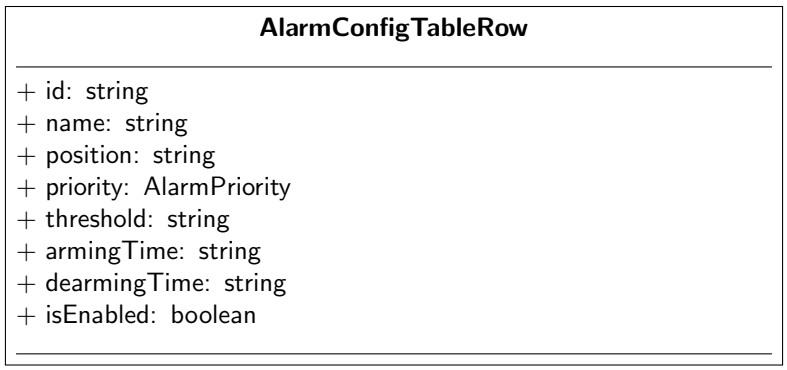

Figura 423: Diagramma della classe AlarmConfigTableRow

Descrizione: Tipo che rappresenta una riga della tabella di configurazione delle regole di allarme. Raccoglie le informazioni di posizione, priorità, soglia e orari di armamento da visualizzare nella tabella.

4.2.1.10 AlarmConfigFormStateService

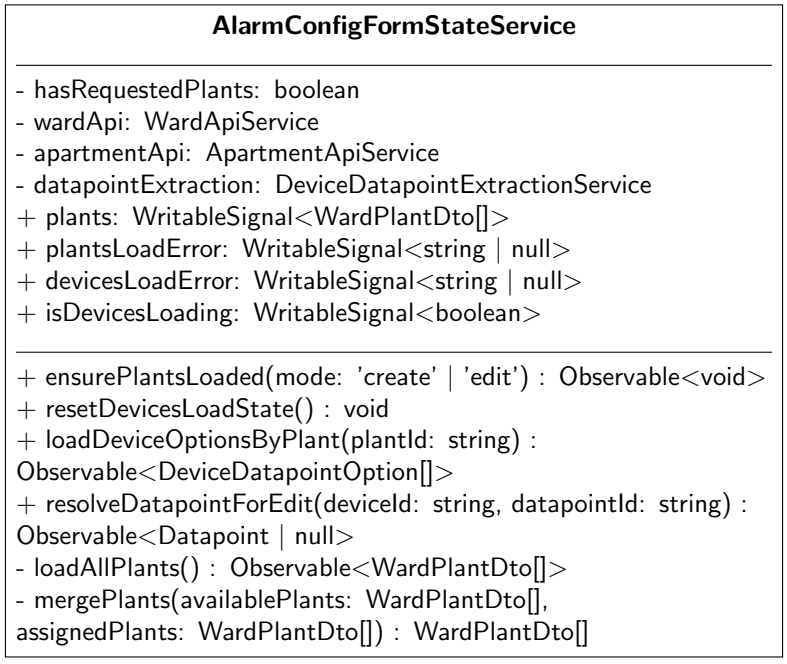

Figura 424: Diagramma della classe AlarmConfigFormStateService

Descrizione: Servizio Angular per la gestione dello stato del form di configurazione allarme, con scope a livello di componente. Gestisce il caricamento degli impianti e delle opzioni dispositivo, esponendo segnali reattivi per lo stato di caricamento e gli eventuali errori.

Descrizione dei metodi della classe:

- `ensurePlantsLoaded(mode: 'create' | 'edit') : Observable<void>`: Carica la lista degli impianti disponibili se non ancora caricata e la modalità è di creazione, aggiornando il segnale plants o l'errore in caso di fallimento.

- `resetDevicesLoadState() : void`: Reimposta l'errore di caricamento dispositivi e il flag di caricamento in corso.
- `loadDeviceOptionsByPlant(plantId: string) : Observable<DeviceDatapointOption[]>`: Carica le opzioni dispositivo per l'impianto specificato, aggiornando lo stato di caricamento e restituendo un array vuoto in caso di errore.
- `resolveDatapointForEdit(deviceId: string, datapointId: string) : Observable<Datapoint | null>`: Ricerca il datapoint corrispondente agli identificativi forniti tra tutti gli impianti disponibili, restituendo null se non trovato o in caso di errore.
- `loadAllPlants() : Observable<WardPlantDto[]>`: Recupera tutti gli impianti disponibili combinando quelli dei ward e quelli generici, deduplicando e ordinando il risultato per nome.
- `mergePlants(availablePlants: WardPlantDto[], assignedPlants: WardPlantDto[]) : WardPlantDto[]`: Unisce due liste di impianti deduplicando per identificativo e ordinando il risultato alfabeticamente per nome.

4.2.1.11 AlarmConfigStateService

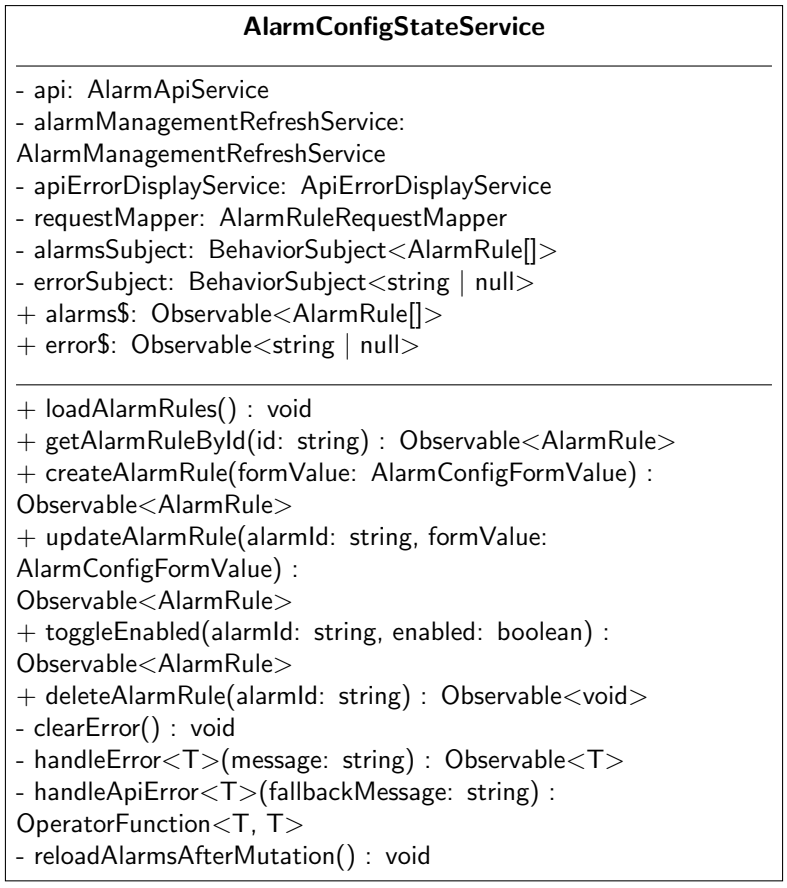

Figura 425: Diagramma della classe AlarmConfigStateService

Descrizione: Servizio Angular con scope a livello di componente per la gestione dello stato delle regole di allarme nella pagina di configurazione. Coordina le operazioni CRUD delegando le chiamate API al servizio dedicato e mantenendo la lista aggiornata tramite BehaviorSubject.

Descrizione dei metodi della classe:

- `loadAlarmRules() : void`: Carica la lista delle regole di allarme dal backend e aggiorna il subject interno, impostando l'errore in caso di fallimento.
- `getAlarmRuleById(id: string) : Observable<AlarmRule>`: Recupera la singola regola di allarme corrispondente all'identificativo specificato.
- `createAlarmRule(formValue: AlarmConfigFormValue) : Observable<AlarmRule>`: Mappa il valore del form nel DTO di creazione e invia la richiesta al backend, ricaricando la lista al termine.
- `updateAlarmRule(alarmId: string, formValue: AlarmConfigFormValue) : Observable<AlarmRule>`: Mappa il valore del form nel DTO di aggiornamento e invia la richiesta al backend, ricaricando la lista al termine.
- `toggleEnabled(alarmId: string, enabled: boolean) : Observable<AlarmRule>`: Aggiorna lo stato di armamento della regola specificata, costruendo il DTO a partire dallo stato locale corrente.
- `deleteAlarmRule(alarmId: string) : Observable<void>`: Elimina la regola di allarme specificata e aggiorna la lista locale rimuovendo l'elemento eliminato.
- `clearError() : void`: Reimposta il subject degli errori a null se contiene un valore.
- `handleError<T>(message: string) : Observable<T>`: Imposta il messaggio di errore nel subject e restituisce EMPTY per interrompere il flusso.
- `handleApiError<T>(fallbackMessage: string) : OperatorFunction<T, T>`: Restituisce un operatore RxJS che intercetta gli errori API, formattando il messaggio tramite il servizio di display degli errori.
- `reloadAlarmsAfterMutation() : void`: Ricarica la lista delle regole di allarme e notifica il servizio di refresh dopo una mutazione.

4.2.1.12 AlarmConfigTablePresenterService

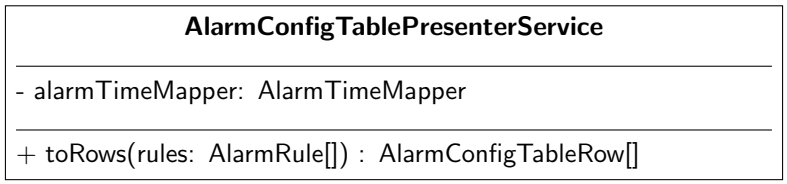

Figura 426: Diagramma della classe AlarmConfigTablePresenterService

Descrizione: Servizio Angular per la trasformazione delle regole di allarme nel formato di presentazione della tabella di configurazione. Delega la conversione degli orari all'AlarmTimeMapper e formatta la soglia combinando operatore e valore.

Descrizione dei metodi della classe:

- `toRows(rules: AlarmRule[]) : AlarmConfigTableRow[]`: Trasforma una lista di regole di allarme nelle corrispondenti righe della tabella di configurazione, formattando posizione, soglia e orari.

4.2.1.13 DeviceDatapointExtractionService

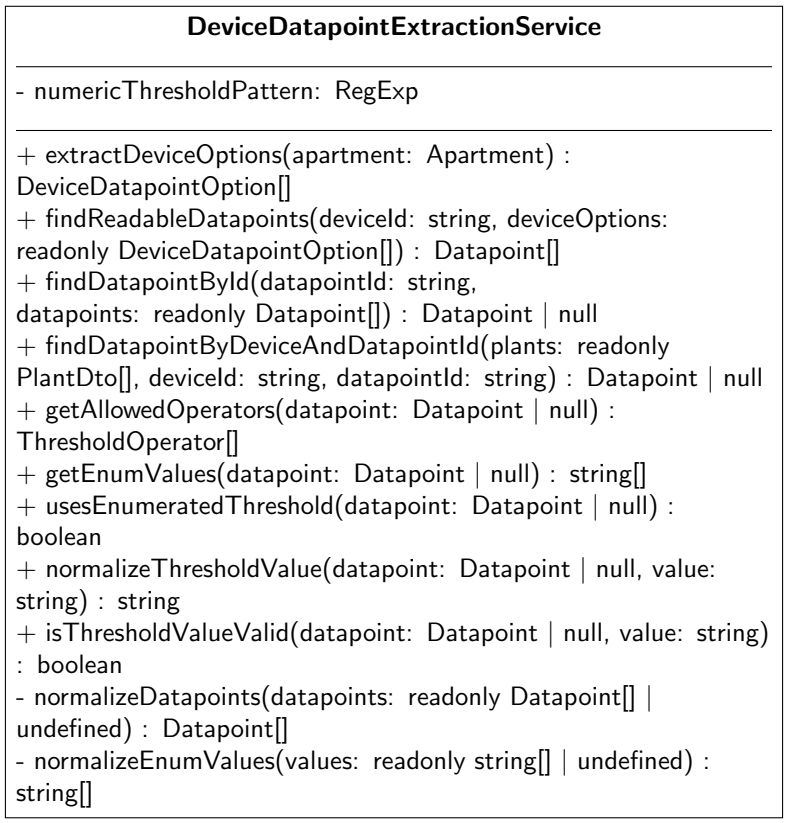

Figura 427: Diagramma della classe DeviceDatapointExtractionService

Descrizione: Servizio Angular per l'estrazione e la normalizzazione dei datapoint dei dispositivi, con supporto alla ricerca per identificativo e alla validazione dei valori di soglia. Determina gli operatori consentiti e il tipo di soglia in base alle caratteristiche del datapoint selezionato.

Descrizione dei metodi della classe:

- `extractDeviceOptions(apartment: Apartment) : DeviceDatapointOption[]`: Estrae le opzioni dispositivo da un appartamento, costruendo l'etichetta combinando nome stanza e nome dispositivo.
- `findReadableDatapoints(deviceId: string, deviceOptions: readonly DeviceDatapointOption[]) : Datapoint[]`: Restituisce i datapoint leggibili del dispositivo corrispondente all'identificativo specificato.
- `findDatapointById(datapointId: string, datapoints: readonly Datapoint[]) : Datapoint | null`: Cerca e restituisce il datapoint corrispondente all'identificativo specificato, o null se non trovato.
- `findDatapointByDeviceAndDatapointId(plants: readonly PlantDto[], deviceId: string, datapointId: string) : Datapoint | null`: Ricerca il datapoint corrispondente agli identificativi di dispositivo e datapoint tra tutti gli impianti forniti.
- `getAllowedOperators(datapoint: Datapoint | null) : ThresholdOperator[]`: Restituisce gli operatori di soglia consentiti per il datapoint specificato, limitando a EQUAL_TO per i datapoint enumerati.
- `getEnumValues(datapoint: Datapoint | null) : string[]`: Restituisce i valori enumerati normalizzati del datapoint specificato, o un array vuoto se assente.

- `usesEnumeratedThreshold(datapoint: Datapoint | null) : boolean`: Verifica se il datapoint richiede una soglia enumerata anziché numerica.
- `normalizeThresholdValue(datapoint: Datapoint | null, value: string) : string`: Normalizza il valore di soglia cercando una corrispondenza case-insensitive tra i valori enumerati del datapoint.
- `isThresholdValueValid(datapoint: Datapoint | null, value: string) : boolean`: Verifica se il valore di soglia è valido rispetto ai valori enumerati o al pattern numerico del datapoint.
- `normalizeDatapoints(datapoints: readonly Datapoint[] | undefined) : Datapoint[]`: Normalizza la lista di datapoint applicando la normalizzazione dei valori enumerati a ciascuno.
- `normalizeEnumValues(values: readonly string[] | undefined) : string[]`: Normalizza i valori enumerati rimuovendo spazi, filtrando i vuoti e deduplicando.

4.2.2 Alarm-history

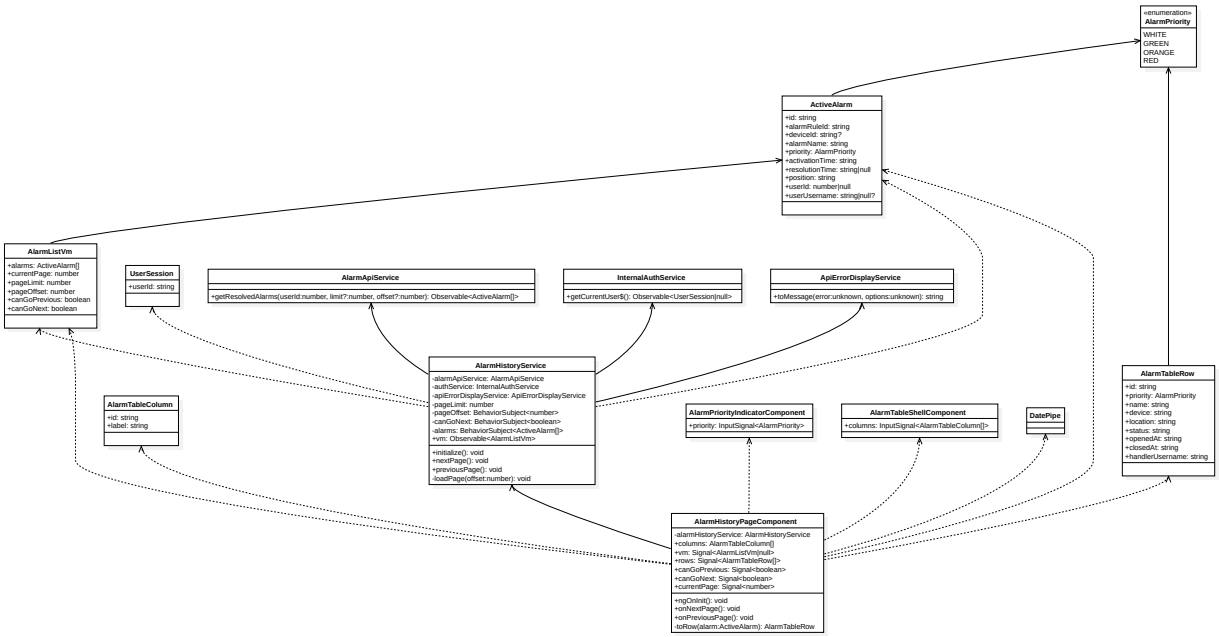

Figura 428: Diagramma delle classi del modulo Alarm-history

4.2.2.1 AlarmHistoryPageComponent

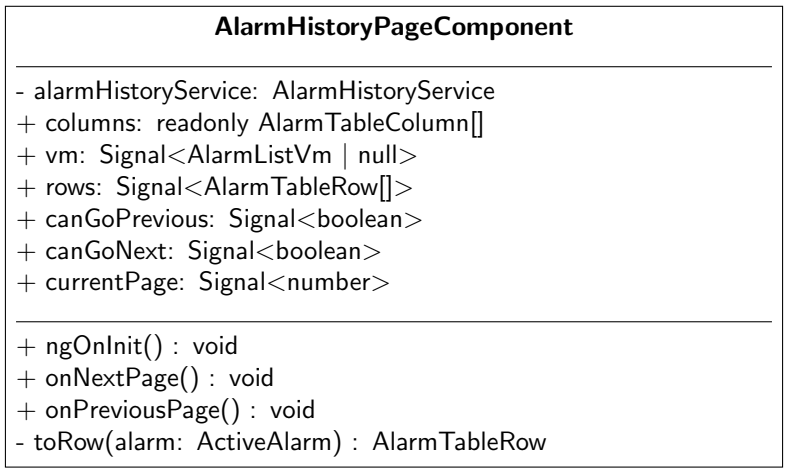

Figura 429: Diagramma della classe AlarmHistoryPageComponent

Descrizione: Componente Angular che rappresenta la pagina dello storico degli allarmi, con supporto alla paginazione e visualizzazione delle informazioni di risoluzione. Le righe vengono ordinate per data di risoluzione decrescente e i valori assenti vengono sostituiti con valori di fallback.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il servizio storico allarmi al momento della creazione del componente.
- `onNextPage() : void`: Delega al servizio storico la navigazione alla pagina successiva degli allarmi.
- `onPreviousPage() : void`: Delega al servizio storico la navigazione alla pagina precedente degli allarmi.
- `toRow(alarm: ActiveAlarm) : AlarmTableRow`: Trasforma un allarme attivo nella corrispondente riga della tabella, normalizzando i valori assenti con valori di fallback.

4.2.2.2 AlarmListVm

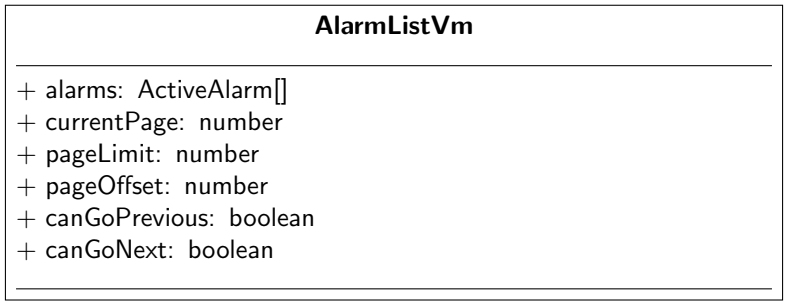

Figura 430: Diagramma della classe AlarmListVm

Descrizione: Interfaccia che rappresenta il view model della lista degli allarmi, contenente i dati della pagina corrente e lo stato di navigazione della paginazione.

4.2.2.3 AlarmTableRow

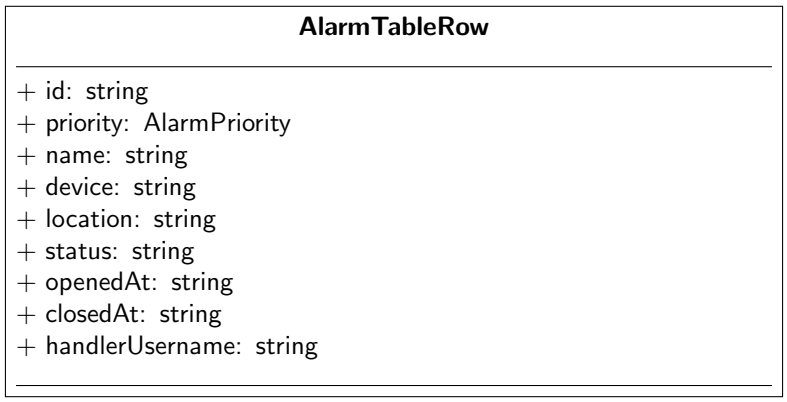

Figura 431: Diagramma della classe AlarmTableRow

Descrizione: Tipo che rappresenta una riga della tabella dello storico degli allarmi. Raccoglie le informazioni di priorità, posizione, stato e dati di risoluzione da visualizzare nella tabella.

4.2.2.4 AlarmHistoryService

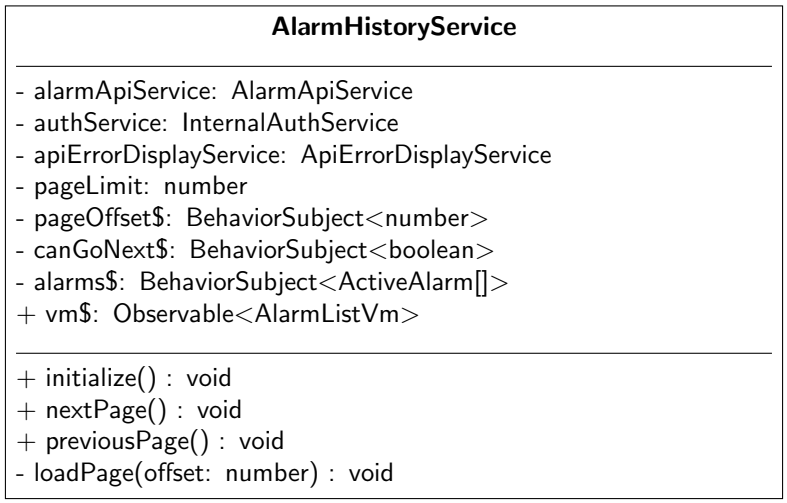

Figura 432: Diagramma della classe AlarmHistoryService

Descrizione: Servizio Angular per il caricamento e la gestione paginata dello storico degli allarmi risolti. Espone un view model reattivo che combina la lista degli allarmi e lo stato di paginazione per il consumo da parte dei componenti.

Descrizione dei metodi della classe:

- `initialize() : void`: Reimposta la paginazione e carica la prima pagina degli allarmi risolti.
- `nextPage() : void`: Carica la pagina successiva degli allarmi risolti se disponibile.
- `previousPage() : void`: Carica la pagina precedente degli allarmi risolti se l'offset corrente è maggiore di zero.
- `loadPage(offset: number) : void`: Recupera gli allarmi risolti per l'utente corrente all'offset specificato, aggiornando la lista e lo stato di paginazione.

4.2.3 Alarm-management

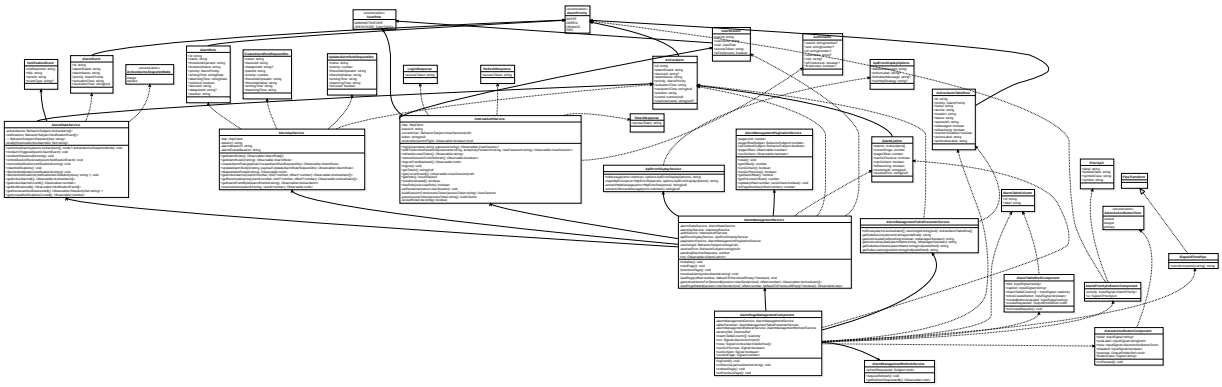

Figura 433: Diagramma delle classi del modulo Alarm-management

4.2.3.1 AlarmPageManagementComponent

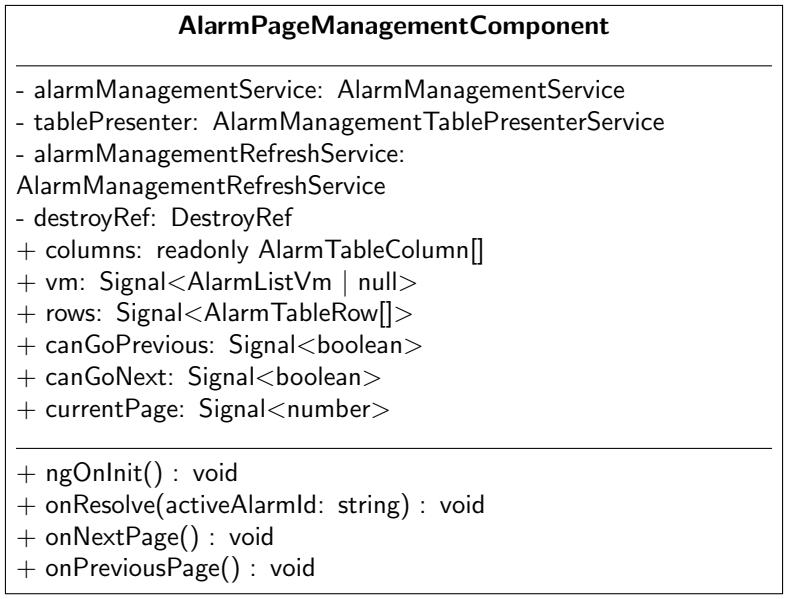

Figura 434: Diagramma della classe AlarmPageManagementComponent

Descrizione: Componente Angular che rappresenta la pagina di gestione degli allarmi attivi, con supporto alla paginazione e alla risoluzione. Si aggiorna automaticamente in risposta alle notifiche di refresh provenienti dal servizio dedicato.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il servizio di gestione allarmi e sottoscrive le notifiche di refresh per reinizializzare i dati.
- `onResolve(activeAlarmId: string) : void`: Delega al servizio di gestione la risoluzione dell'allarme con l'identificativo specificato.
- `onNextPage() : void`: Delega al servizio di gestione la navigazione alla pagina successiva degli allarmi.
- `onPreviousPage() : void`: Delega al servizio di gestione la navigazione alla pagina precedente degli allarmi.

4.2.3.2 ActiveAlarmTableRow

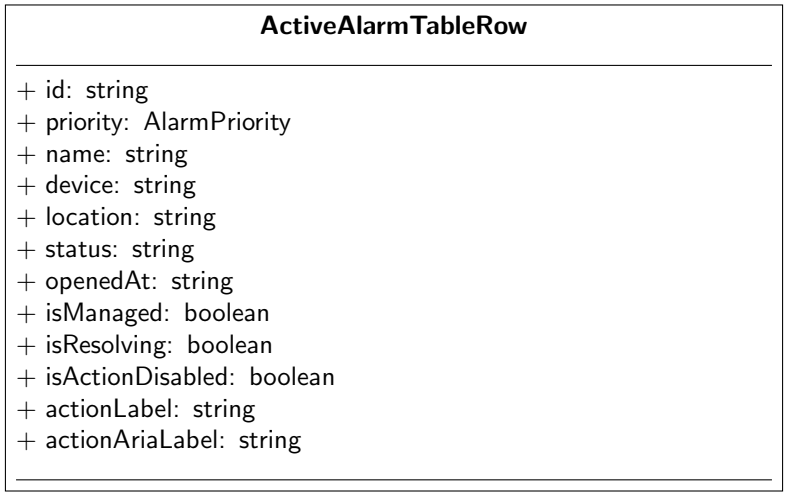

Figura 435: Diagramma della classe ActiveAlarmTableRow

Descrizione: Tipo che rappresenta una riga della tabella degli allarmi attivi. Raccoglie le informazioni di priorità, posizione, stato e i flag necessari alla gestione dell'azione di risoluzione.

4.2.3.3 AlarmListVm

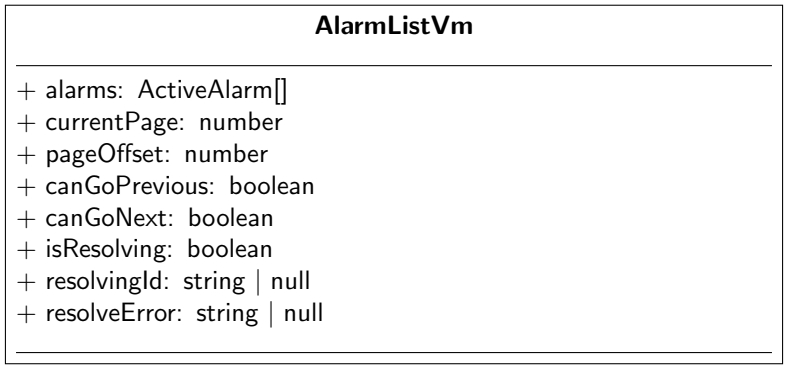

Figura 436: Diagramma della classe AlarmListVm

Descrizione: Interfaccia che rappresenta il view model della lista degli allarmi attivi, contenente i dati della pagina corrente, lo stato di paginazione e le informazioni sull'eventuale risoluzione in corso.

4.2.3.4 AlarmManagementPaginationService

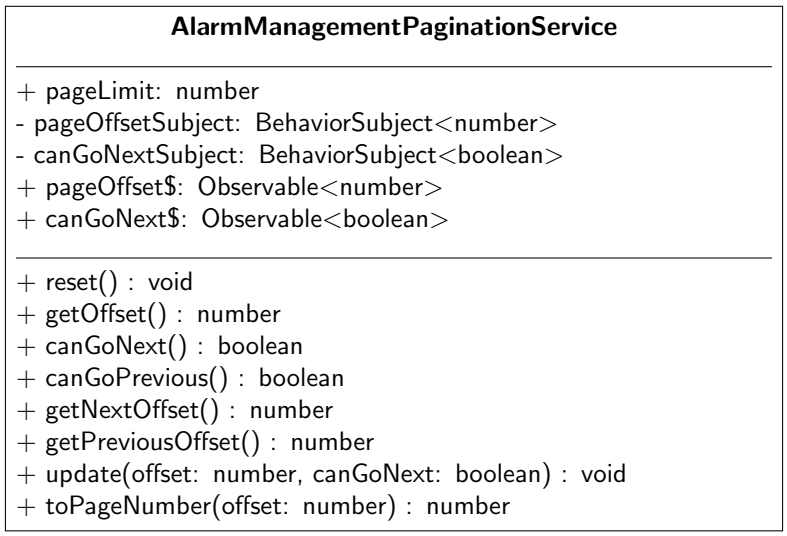

Figura 437: Diagramma della classe AlarmManagementPaginationService

Descrizione: Servizio Angular per la gestione dello stato di paginazione della lista allarmi. Espone stream reattivi e metodi di utilità per il calcolo degli offset e la navigazione tra le pagine.

Descrizione dei metodi della classe:

- `reset() : void`: Reimposta l'offset a zero e la disponibilità della pagina successiva a false.
- `getOffset() : number`: Restituisce il valore corrente dell'offset di paginazione.
- `canGoNext() : boolean`: Restituisce true se è disponibile una pagina successiva.
- `canGoPrevious() : boolean`: Restituisce true se l'offset corrente è maggiore di zero, indicando la presenza di una pagina precedente.
- `getNextOffset() : number`: Calcola e restituisce l'offset della pagina successiva.
- `getPreviousOffset() : number`: Calcola e restituisce l'offset della pagina precedente, con un minimo di zero.
- `update(offset: number, canGoNext: boolean) : void`: Aggiorna l'offset corrente e la disponibilità della pagina successiva.
- `toPageNumber(offset: number) : number`: Converte un offset nel corrispondente numero di pagina a partire da uno.

4.2.3.5 AlarmManagementTablePresenterService

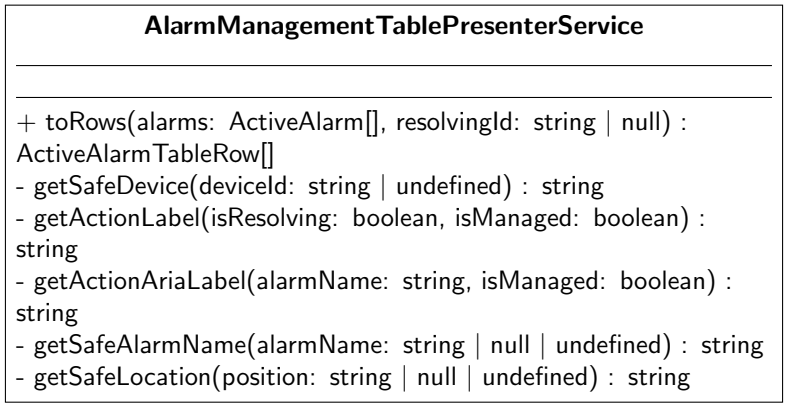

Figura 438: Diagramma della classe AlarmManagementTablePresenterService

Descrizione: Servizio Angular che si occupa della trasformazione dei dati degli allarmi attivi nelle righe del modello di presentazione della tabella. Gestisce la normalizzazione dei valori e il calcolo delle etichette e dei flag di stato per ciascuna riga.

Descrizione dei metodi della classe:

- `toRows(alarms: ActiveAlarm[], resolvingId: string | null) : ActiveAlarmTableRow[]`: Trasforma una lista di allarmi attivi nelle corrispondenti righe della tabella, calcolando stato, etichette e flag di interazione.
- `getSafeDevice(deviceId: string | undefined) : string`: Restituisce l'identificativo del dispositivo normalizzato, o un trattino se assente o vuoto.
- `getActionLabel(isResolving: boolean, isManaged: boolean) : string`: Restituisce l'etichetta del pulsante di azione in base allo stato di risoluzione e gestione dell'allarme.
- `getActionAriaLabel(alarmName: string, isManaged: boolean) : string`: Restituisce l'etichetta aria del pulsante di azione per l'accessibilità, differenziando tra allarme già gestito e da gestire.
- `getSafeAlarmName(alarmName: string | null | undefined) : string`: Restituisce il nome dell'allarme normalizzato, o la stringa senza nome se assente o vuoto.
- `getSafeLocation(position: string | null | undefined) : string`: Restituisce la posizione dell'allarme normalizzata, o un trattino se assente o vuota.

4.2.3.6 AlarmManagementService

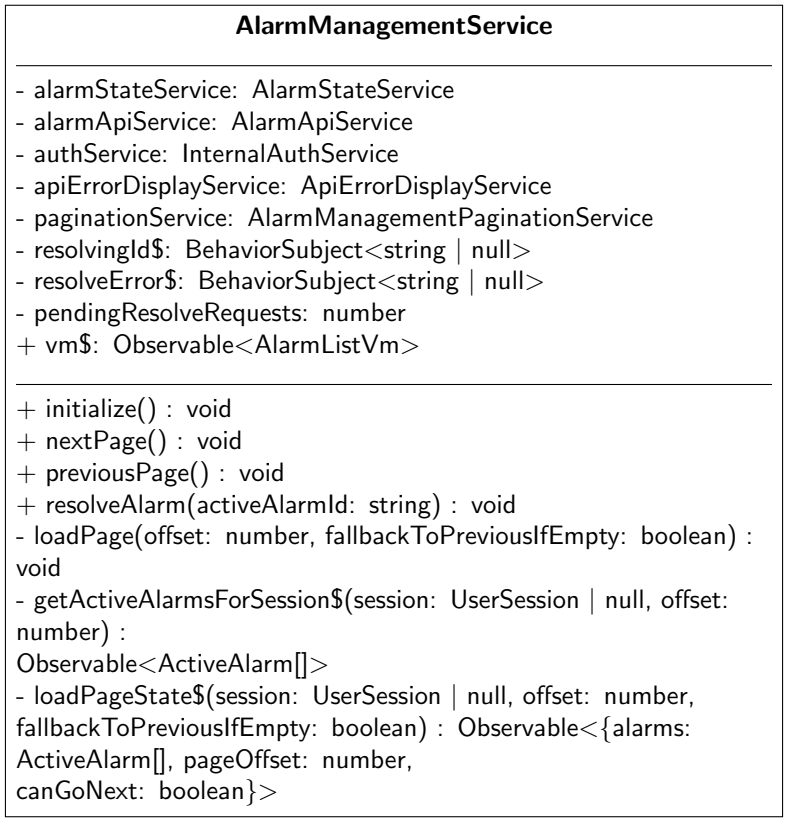

Figura 439: Diagramma della classe AlarmManagementService

Descrizione: Servizio Angular per la gestione degli allarmi attivi, che coordina il caricamento paginato, la risoluzione e l'aggiornamento del view model reattivo. Integra i servizi di stato, API, autenticazione e paginazione per esporre un unico stream di dati ai componenti.

Descrizione dei metodi della classe:

- `initialize() : void`: Reimposta la paginazione e carica la prima pagina degli allarmi attivi.
- `nextPage() : void`: Carica la pagina successiva degli allarmi se disponibile.
- `previousPage() : void`: Carica la pagina precedente degli allarmi se disponibile.
- `resolveAlarm(activeAlarmId: string) : void`: Avvia la risoluzione dell'allarme specificato, aggiornando lo stato e ricaricando la pagina corrente al termine.
- `loadPage(offset: number, fallbackToPreviousIfEmpty: boolean) : void`: Carica la pagina di allarmi corrispondente all'offset specificato, con fallback alla pagina precedente se la pagina risulta vuota.
- `getActiveAlarmsForSession$(session: UserSession | null, offset: number) : Observable<ActiveAlarm[]>`: Recupera gli allarmi attivi per la sessione utente corrente all'offset specificato, lanciando un errore se l'utente non è valido.
- `loadPageState$(session: UserSession | null, offset: number, fallbackToPreviousIfEmpty: boolean) : Observable<{alarms: ActiveAlarm[], pageOffset: number, canGoNext: boolean}>`: Calcola lo stato completo della pagina da caricare, gestendo il fallback alla pagina precedente e la verifica della disponibilità della pagina successiva.

4.2.4 Analytics

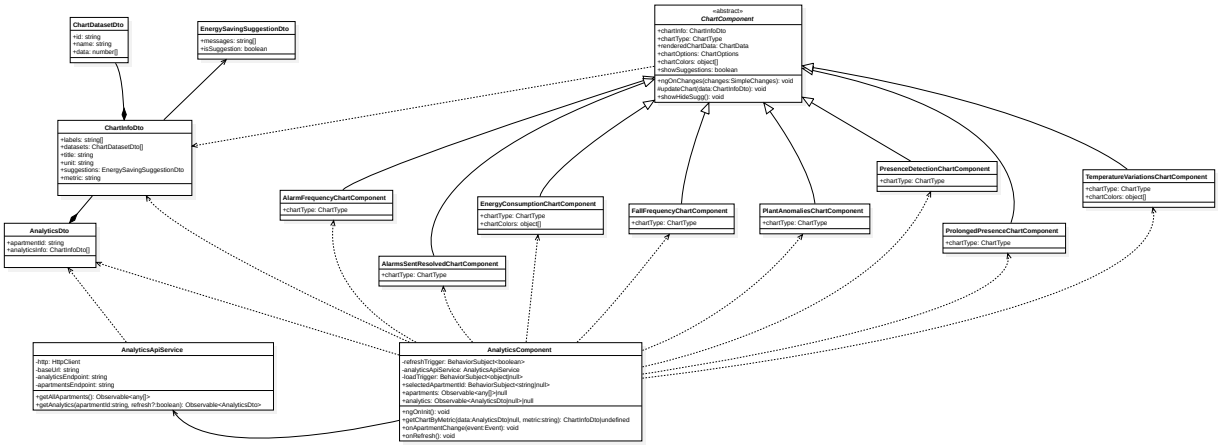

Figura 440: Diagramma delle classi del modulo Analytics

4.2.4.1 ChartDatasetDto

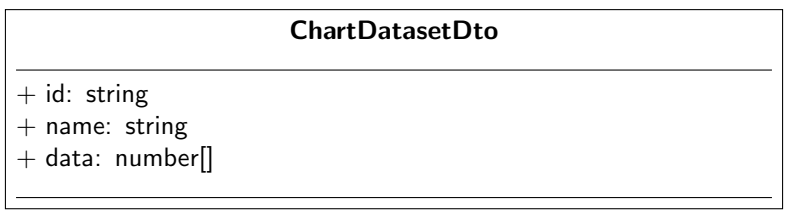

Figura 441: Diagramma della classe ChartDatasetDto

Descrizione: DTO che rappresenta una singola serie di dati all'interno di un grafico Chart.js

4.2.4.2 EnergySavingSuggestionDto

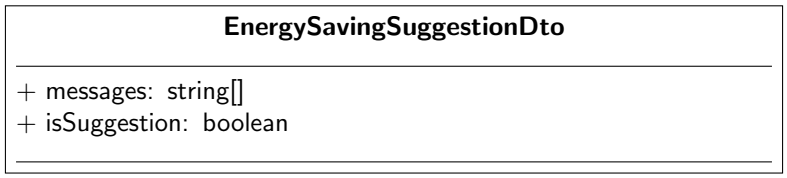

Figura 442: Diagramma della classe EnergySavingSuggestionDto

Descrizione: DTO che trasporta il suggerimento energetico generato dal backend, visualizzato nel pannello suggerimenti di `ChartComponent`

4.2.4.3 ChartInfoDto

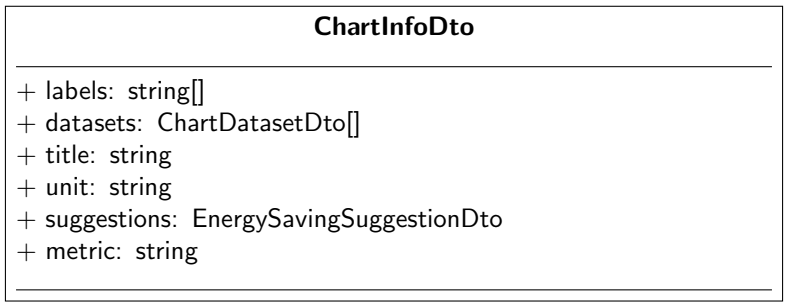

Figura 443: Diagramma della classe ChartInfoDto

Descrizione: DTO che trasporta tutti i dati necessari al rendering di un singolo grafico analytics: etichette, serie, metadati e suggerimento energetico

4.2.4.4 AnalyticsDto

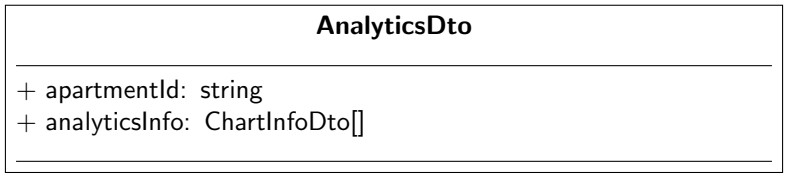

Figura 444: Diagramma della classe AnalyticsDto

Descrizione: DTO che trasporta l'insieme delle analytics di un appartamento dal backend al frontend

4.2.4.5 AnalyticsApiService

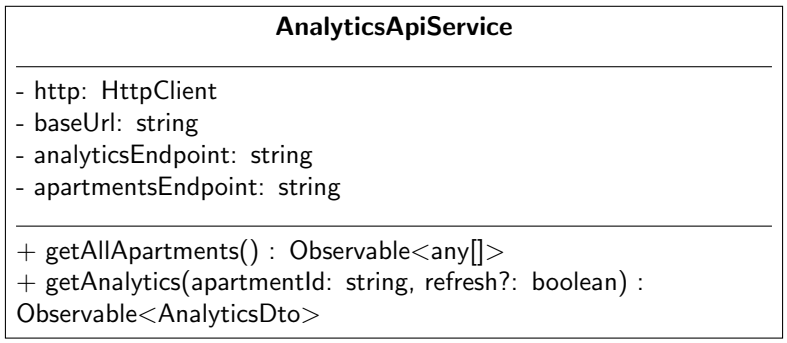

Figura 445: Diagramma della classe AnalyticsApiService

Descrizione: Servizio Angular che gestisce le chiamate HTTP verso le API REST del modulo analytics. Espone metodi reattivi basati su `Observable` per il recupero degli appartamenti e delle analytics

Descrizione dei metodi della classe:

- `getAllApartments() : Observable<any[]>`: Recupera dal backend la lista di tutti gli appartamenti disponibili per la selezione
- `getAnalytics(apartmentId: string, refresh?: boolean) : Observable<AnalyticsDto>`: Recupera dal backend le analytics dell'appartamento identificato da `apartmentId`; il parametro opzionale `refresh` forza il ricaricamento bypassando la cache

4.2.4.6 ChartComponent

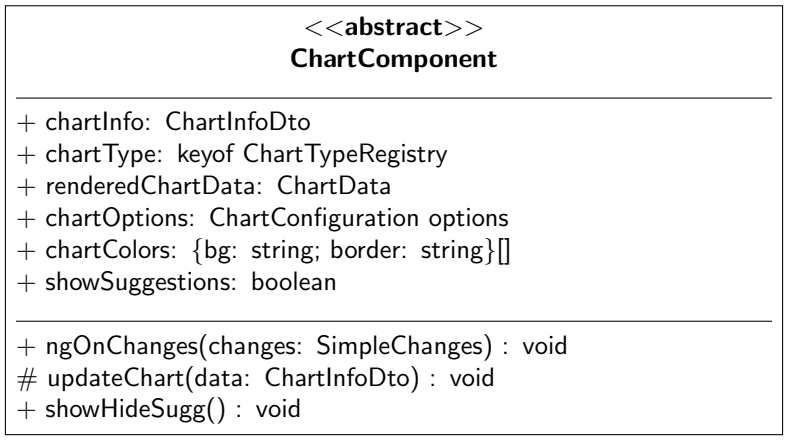

Figura 446: Diagramma della classe astratta ChartComponent

Descrizione: Classe base astratta per tutti i componenti grafici analytics. Gestisce il ciclo di vita del grafico Chart.js, l'aggiornamento dei dati al cambio degli input e la visibilità dei suggerimenti energetici. Tutte le sottoclassi ereditano da questa classe e ne specializzano il `chartType`

Descrizione dei metodi della classe:

- `ngOnChanges(changes: SimpleChanges) : void`: Intercetta i cambiamenti degli input Angular e aggiorna il grafico di conseguenza
- `updateChart(data: ChartInfoDto) : void`: Metodo protetto che effettua il rendering del grafico a partire dai dati ricevuti; può essere sovrascritto dalle sottoclassi
- `showHideSugg() : void`: Alterna la visibilità del pannello dei suggerimenti energetici

4.2.4.7 AlarmFrequencyChartComponent

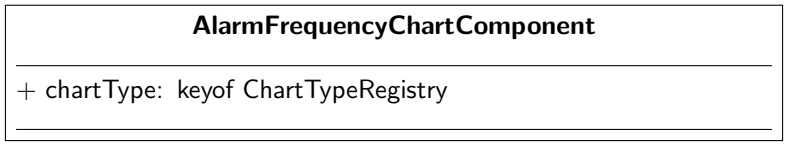

Figura 447: Diagramma della classe AlarmFrequencyChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza la frequenza degli allarmi del reparto come grafico a linee

4.2.4.8 AlarmsSentResolvedChartComponent

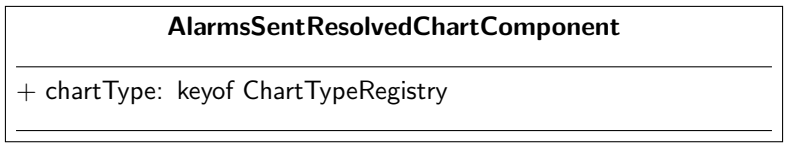

Figura 448: Diagramma della classe AlarmsSentResolvedChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza il confronto tra allarmi inviati e allarmi risolti nel reparto come grafico a linee

4.2.4.9 EnergyConsumptionChartComponent

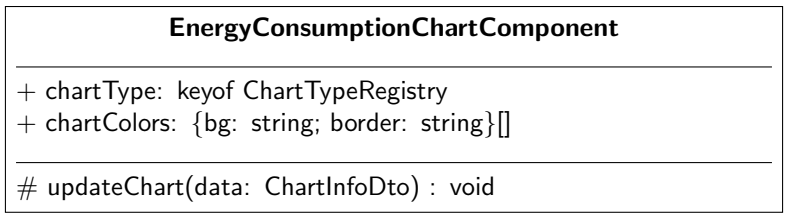

Figura 449: Diagramma della classe EnergyConsumptionChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza il consumo energetico dell'impianto come grafico a linee. Sovrascrive `updateChart` e definisce una palette colori personalizzata

Descrizione dei metodi della classe:

- `updateChart(data: ChartInfoDto) : void`: Sovrascrive il metodo base per applicare logica di rendering specifica al consumo energetico

4.2.4.10 FallFrequencyChartComponent

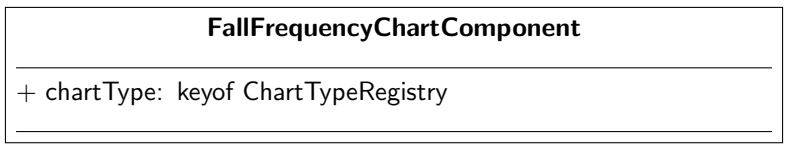

Figura 450: Diagramma della classe FallFrequencyChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza la frequenza degli eventi di caduta nel reparto come grafico a barre

4.2.4.11 PlantAnomaliesChartComponent

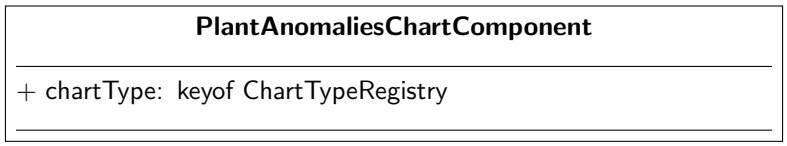

Figura 451: Diagramma della classe PlantAnomaliesChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza le anomalie di consumo energetico dell'impianto come grafico a barre

4.2.4.12 PresenceDetectionChartComponent

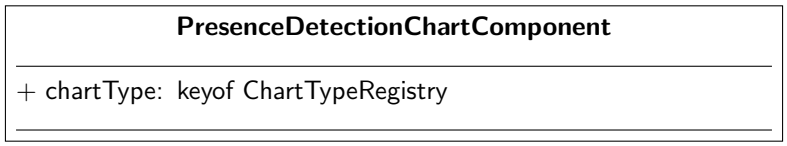

Figura 452: Diagramma della classe PresenceDetectionChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza le presenze rilevate dai sensori come grafico a barre

4.2.4.13 ProlongedPresenceChartComponent

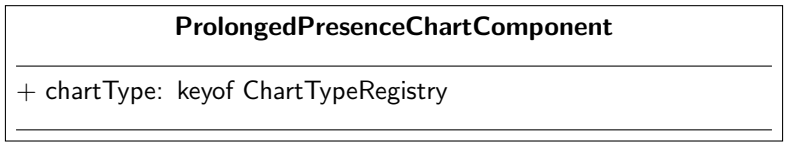

Figura 453: Diagramma della classe ProlongedPresenceChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza le presenze prolungate (oltre 30 minuti) rilevate dai sensori come grafico a barre

4.2.4.14 TemperatureVariationsChartComponent

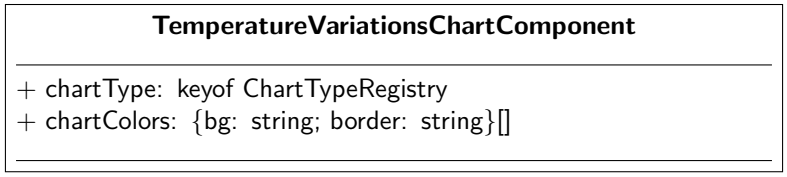

Figura 454: Diagramma della classe TemperatureVariationsChartComponent

Descrizione: Componente figlio di `ChartComponent` che visualizza le variazioni di temperatura rilevate dai termostati come grafico a linee, con palette colori personalizzata

4.2.4.15 AnalyticsComponent

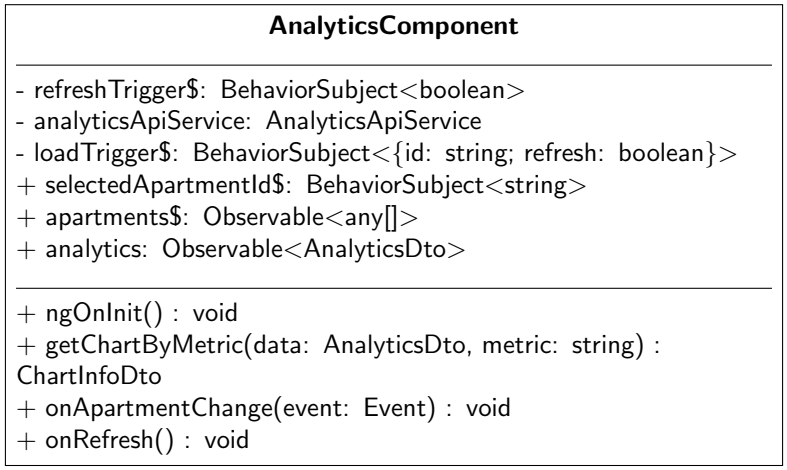

Figura 455: Diagramma della classe AnalyticsComponent

Descrizione: Componente principale della sezione analytics. Implementa `OnInit` e orchestra il caricamento reattivo degli appartamenti e delle analytics tramite `BehaviorSubject` e `AnalyticsApiService`

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il componente al momento del mount, avviando il caricamento degli appartamenti e delle analytics
- `getChartByMetric(data: AnalyticsDto, metric: string) : ChartInfoDto`: Restituisce il `ChartInfoDto` corrispondente alla metrica indicata a partire dai dati analytics
- `onApartmentChange(event: Event) : void`: Gestisce il cambio dell'appartamento selezionato dall'utente, aggiornando il `loadTrigger$`
- `onRefresh() : void`: Forza il ricaricamento delle analytics per l'appartamento corrente emettendo un nuovo valore sul `loadTrigger$`

4.2.4.16 DeviceAction

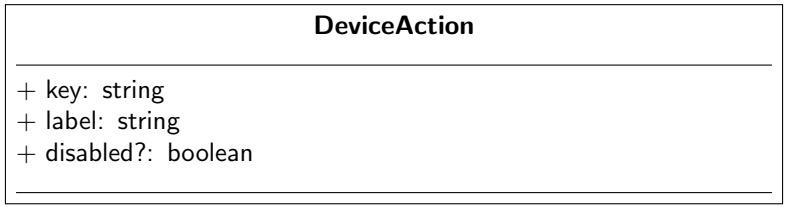

Figura 456: Diagramma della classe DeviceAction

Descrizione: Interfaccia che rappresenta un'azione disponibile su un dispositivo IoT, con chiave identificativa, etichetta e flag di disabilitazione opzionale

4.2.4.17 Datapoint

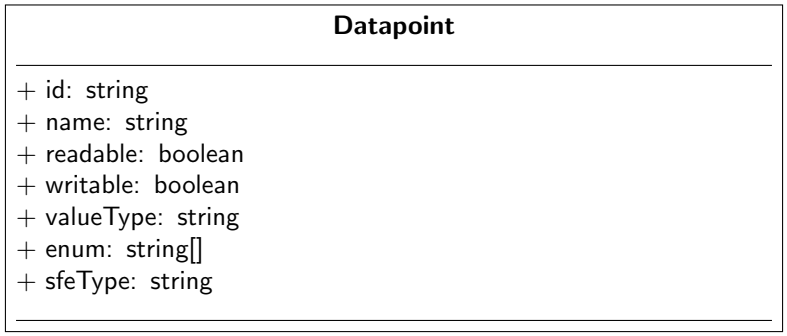

Figura 457: Diagramma della classe Datapoint

Descrizione: Interfaccia che rappresenta un singolo datapoint di un dispositivo IoT, con le sue proprietà di lettura, scrittura e tipo

4.2.4.18 Device

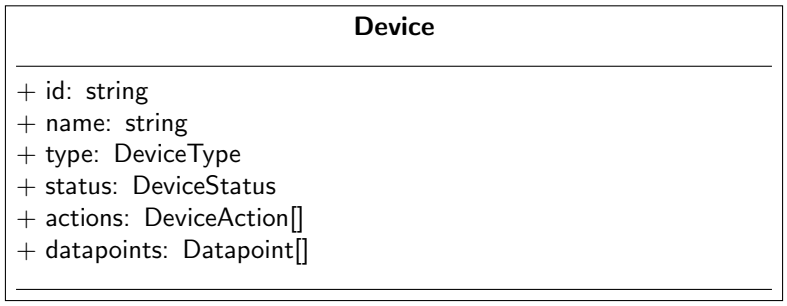

Figura 458: Diagramma della classe Device

Descrizione: Interfaccia che rappresenta un dispositivo IoT all'interno di un appartamento, aggregando tipo, stato, azioni disponibili e datapoint

4.2.4.19 Room

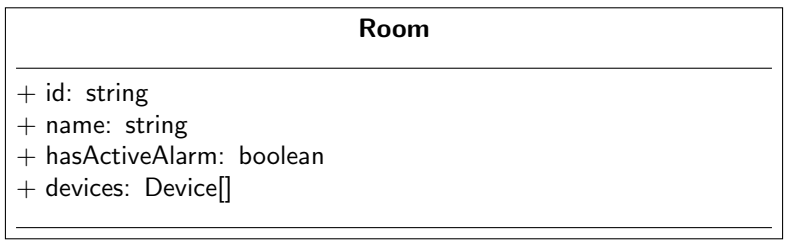

Figura 459: Diagramma della classe Room

Descrizione: Interfaccia che rappresenta una stanza all'interno di un appartamento, con i suoi dispositivi e lo stato degli allarmi

4.2.4.20 Apartment

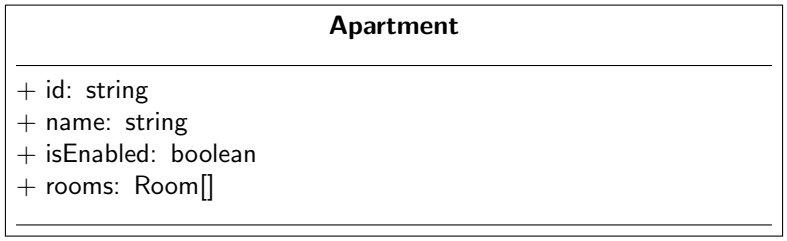

Figura 460: Diagramma della classe Apartment

Descrizione: Interfaccia che rappresenta un appartamento monitorato, aggregando le stanze e i relativi dispositivi

4.2.4.21 UserSession

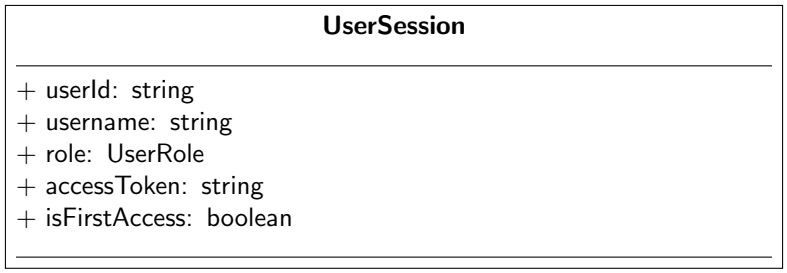

Figura 461: Diagramma della classe UserSession

Descrizione: Interfaccia che rappresenta la sessione utente attiva, contenente le informazioni di autenticazione e il ruolo

4.2.4.22 LoginResponse

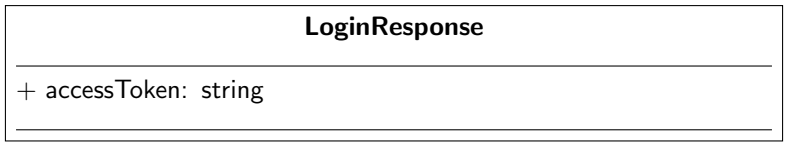

Figura 462: Diagramma della classe LoginResponse

Descrizione: DTO di risposta al login, contenente il token JWT di accesso

4.2.4.23 RefreshResponse

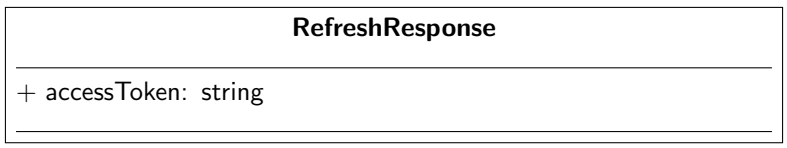

Figura 463: Diagramma della classe RefreshResponse

Descrizione: DTO di risposta al refresh del token, contenente il nuovo token JWT di accesso

4.2.4.24 AuthClaims

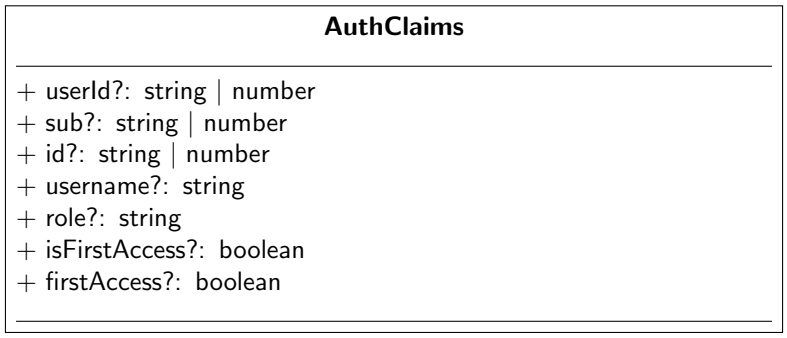

Figura 464: Diagramma della classe AuthClaims

Descrizione: Interfaccia che modella i claims del payload JWT, con campi opzionali per gestire varianti nei nomi dei campi

4.2.4.25 TokenResponse

Figura 465: Diagramma della classe TokenResponse

Descrizione: DTO generico di risposta contenente un token JWT, usato per operazioni di autenticazione

4.2.4.26 InternalAuthService

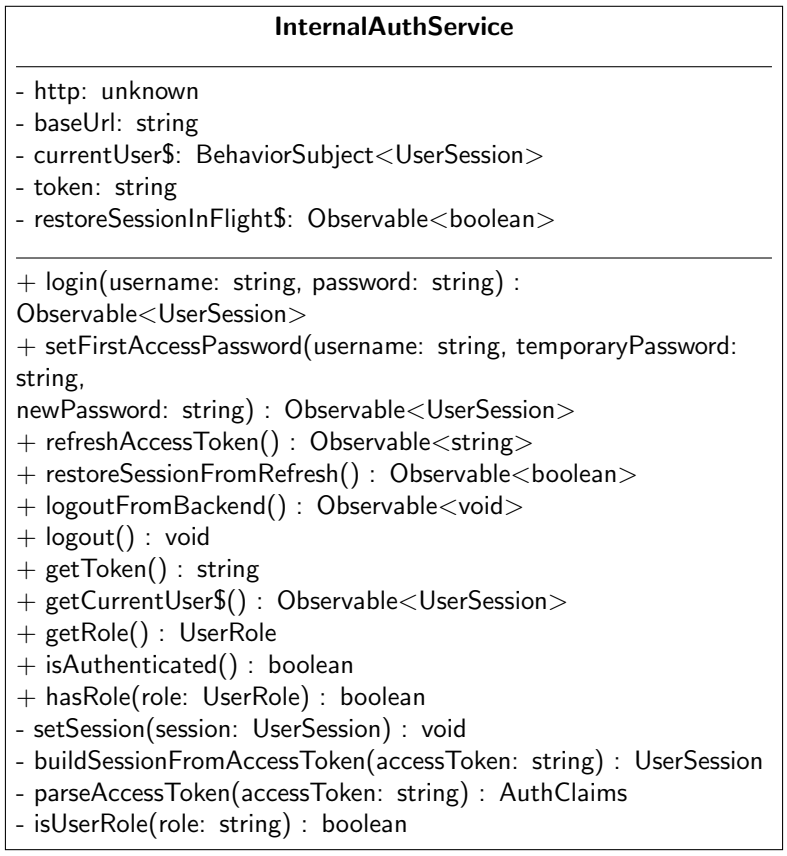

Figura 466: Diagramma della classe InternalAuthService

Descrizione: Servizio Angular che gestisce l'intera logica di autenticazione: login, logout, refresh del token, ripristino della sessione e controllo dei ruoli. Mantiene la sessione corrente tramite un `BehaviorSubject` e la espone come stream reattivo

Descrizione dei metodi della classe:

- `login(username: string, password: string) : Observable<UserSession>`: Autentica l'utente con le credenziali fornite e restituisce uno stream con la sessione creata
- `setFirstAccessPassword(username: string, temporaryPassword: string, newPassword: string) : Observable<UserSession>`: Imposta la nuova password al primo accesso, sostituendo quella temporanea, e restituisce la sessione aggiornata
- `refreshAccessToken() : Observable<string>`: Rinnova il token di accesso tramite il refresh token e restituisce il nuovo JWT
- `restoreSessionFromRefresh() : Observable<boolean>`: Tenta di ripristinare la sessione utente a partire dal refresh token; restituisce true se il ripristino è avvenuto con successo
- `logoutFromBackend() : Observable<void>`: Invalida la sessione lato backend notificando il server del logout
- `logout() : void`: Esegue il logout locale, azzerando la sessione e il token in memoria
- `getToken() : string`: Restituisce il token JWT di accesso correntemente in uso
- `getCurrentUser$() : Observable<UserSession>`: Restituisce lo stream della sessione utente corrente

- `getRole() : UserRole`: Restituisce il ruolo dell'utente della sessione corrente
- `isAuthenticated() : boolean`: Restituisce true se l'utente è autenticato con un token valido
- `hasRole(role: UserRole) : boolean`: Restituisce true se l'utente della sessione corrente possiede il ruolo indicato
- `setSession(session: UserSession) : void`: Aggiorna la sessione corrente in memoria e nel `BehaviorSubject`
- `buildSessionFromAccessToken(accessToken: string) : UserSession`: Costruisce un oggetto `UserSession` a partire dal token JWT, estraendone i claims
- `parseAccessToken(accessToken: string) : AuthClaims`: Decodifica e restituisce il payload JWT come oggetto `AuthClaims`
- `isUserRole(role: string) : boolean`: Verifica se la stringa fornita corrisponde a un valore valido dell'enumerazione `UserRole`

4.2.5 Apartment-monitor

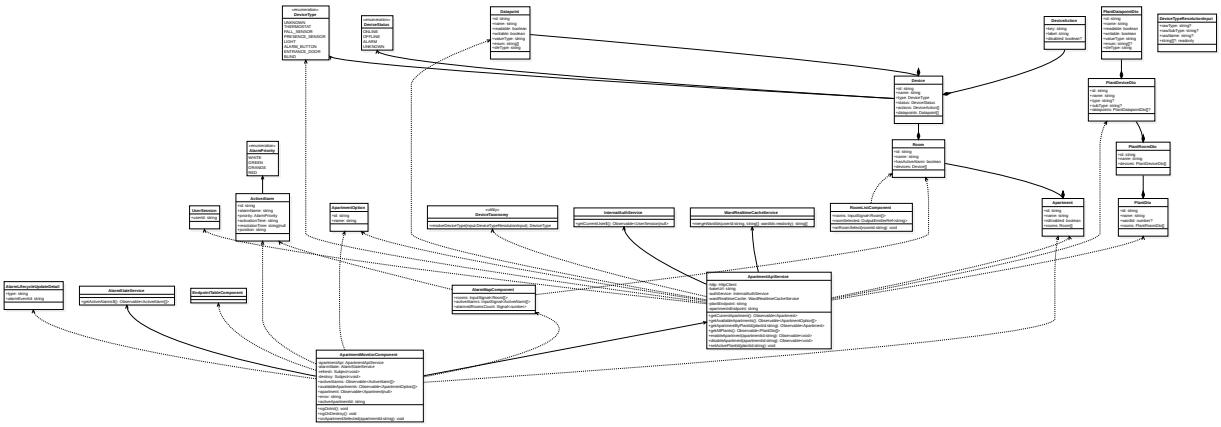

Figura 467: Diagramma delle classi del modulo Apartment monitor

4.2.5.1 ApartmentMonitorComponent

ApartmentMonitorComponent

- apartmentApi: ApartmentApiService
- alarmState: AlarmStateService
- refresh$: Subject<void>
- destroy$: Subject<void>
+ activeAlarms$: Observable<Alarm[]>
+ availableApartments$: Observable<ApartmentOption[]>
+ apartment$: Observable<Apartment | null>
+ error: string
+ activeApartmentId: string

+ ngOnInit() : void
+ ngOnDestroy() : void
+ onApartmentSelected(apartmentId: string) : void

Figura 468: Diagramma della classe ApartmentMonitorComponent

Descrizione: Componente Angular per il monitoraggio di un appartamento, che visualizza gli allarmi attivi e i dati dell'appartamento corrente. Gestisce la selezione dell'appartamento attivo e si aggiorna in risposta agli eventi real-time di allarme.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza la sottoscrizione agli eventi real-time di aggiornamento del ciclo di vita degli allarmi, innescando il refresh dei dati al loro arrivo.
- `ngOnDestroy() : void`: Completa il Subject `destroy$` per terminare tutte le sottoscrizioni attive e prevenire memory leak.
- `onApartmentSelected(apartmentId: string) : void`: Gestisce la selezione di un nuovo appartamento, impostando l'appartamento attivo tramite l'API e innescando il refresh dei dati.

4.2.5.2 AlarmMapComponent

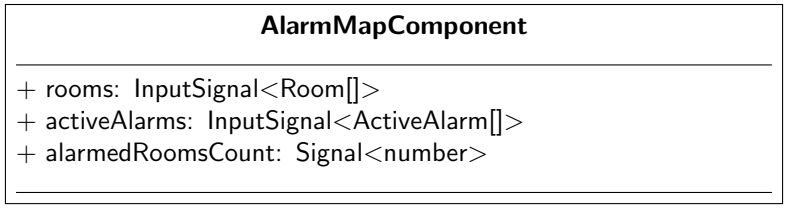

Figura 469: Diagramma della classe AlarmMapComponent

Descrizione: Componente Angular che visualizza una mappa delle stanze con evidenza di quelle in stato di allarme. Calcola dinamicamente il numero di stanze con allarme attivo tramite un segnale derivato.

4.2.5.3 RoomListComponent

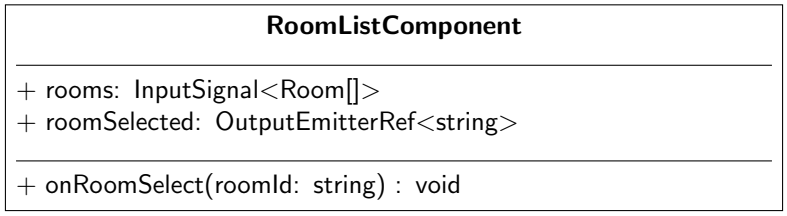

Figura 470: Diagramma della classe RoomListComponent

Descrizione: Componente Angular che visualizza la lista delle stanze disponibili. Emette un evento alla selezione di una stanza, comunicando il relativo identificativo al componente padre.

Descrizione dei metodi della classe:

- `onRoomSelect(roomId: string) : void`: Emette l'evento `roomSelected` con l'identificativo della stanza selezionata.

4.2.5.4 Apartment

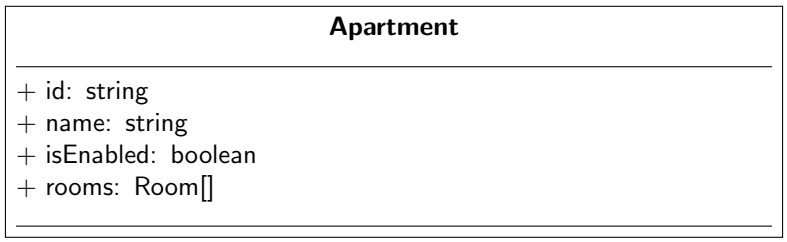

Figura 471: Diagramma della classe Apartment

Descrizione: Interfaccia che rappresenta il modello di dominio di un appartamento. Aggrega le informazioni identificative, lo stato di abilitazione e la lista delle stanze.

4.2.5.5 Datapoint

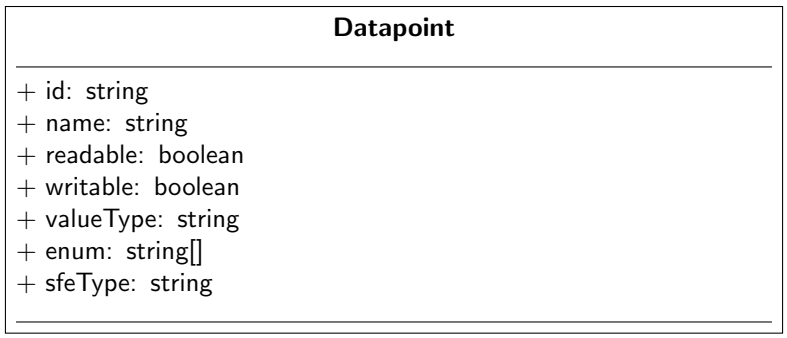

Figura 472: Diagramma della classe Datapoint

Descrizione: Interfaccia che rappresenta un datapoint di un dispositivo. Definisce le proprietà di lettura, scrittura, tipo di valore e valori enumerati disponibili.

4.2.5.6 DeviceAction

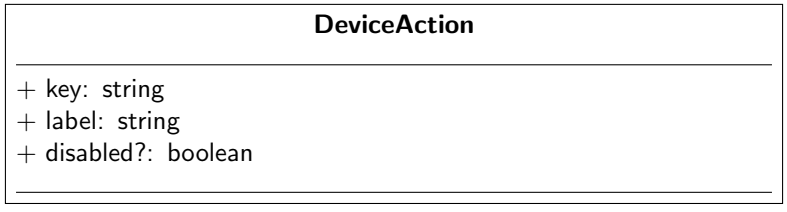

Figura 473: Diagramma della classe DeviceAction

Descrizione: Interfaccia che rappresenta un'azione eseguibile su un dispositivo. Associa una chiave identificativa a un'etichetta visualizzabile e a un flag opzionale di disabilitazione.

4.2.5.7 Device

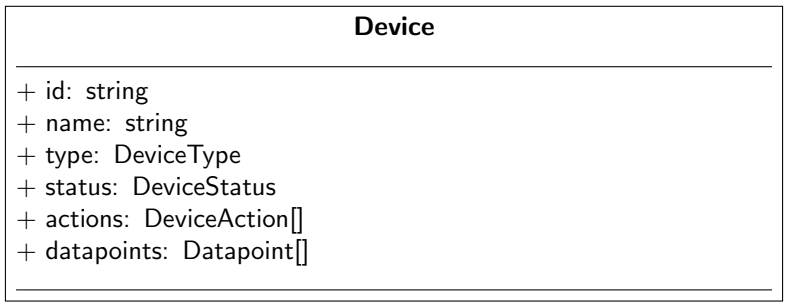

Figura 474: Diagramma della classe Device

Descrizione: Interfaccia che rappresenta il modello di dominio di un dispositivo. Raccoglie tipo, stato, azioni disponibili e datapoint associati.

4.2.5.8 PlantDatapointDto

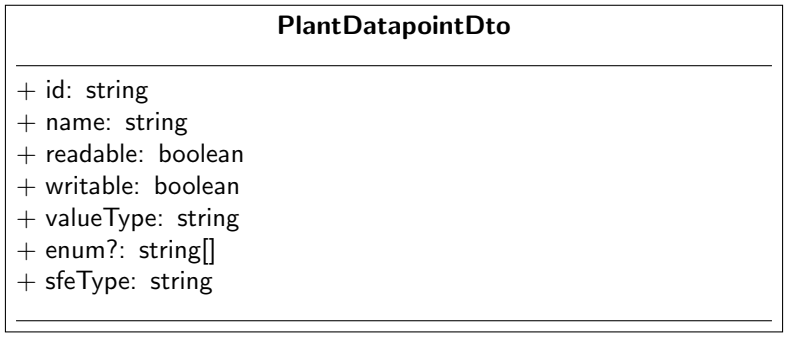

Figura 475: Diagramma della classe PlantDatapointDto

Descrizione: Interfaccia che rappresenta il DTO di un datapoint restituito dal backend nella risposta dell'impianto.

4.2.5.9 PlantDeviceDto

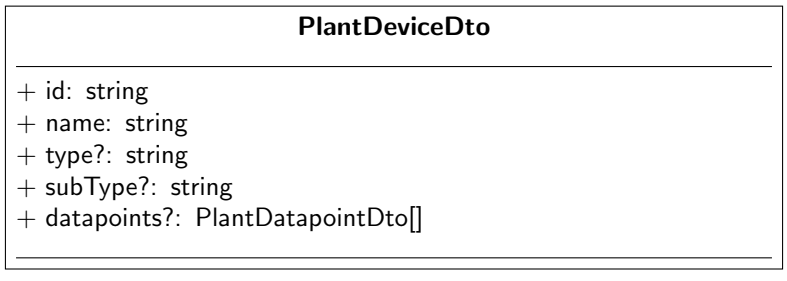

Figura 476: Diagramma della classe PlantDeviceDto

Descrizione: Interfaccia che rappresenta il DTO di un dispositivo restituito dal backend nella risposta dell'impianto.

4.2.5.10 PlantRoomDto

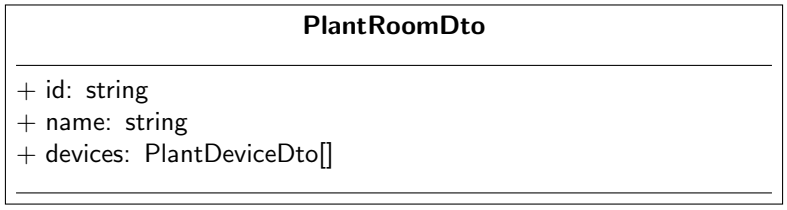

Figura 477: Diagramma della classe PlantRoomDto

Descrizione: Interfaccia che rappresenta il DTO di una stanza restituita dal backend nella risposta dell'impianto.

4.2.5.11 PlantDto

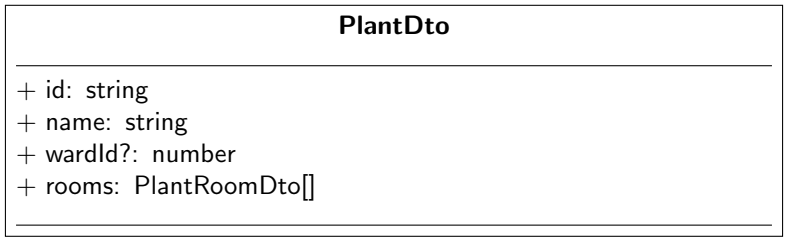

Figura 478: Diagramma della classe PlantDto

Descrizione: Interfaccia che rappresenta il DTO di un impianto restituito dal backend, contenente le stanze e l'eventuale identificativo del ward associato.

4.2.5.12 Room

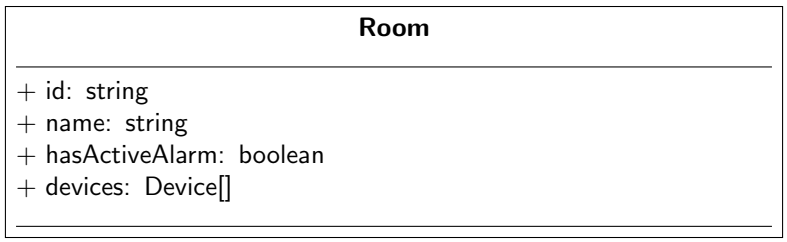

Figura 479: Diagramma della classe Room

Descrizione: Interfaccia che rappresenta il modello di dominio di una stanza. Aggrega le informazioni identificative, lo stato di allarme attivo e la lista dei dispositivi presenti.

4.2.5.13 ApartmentApiService

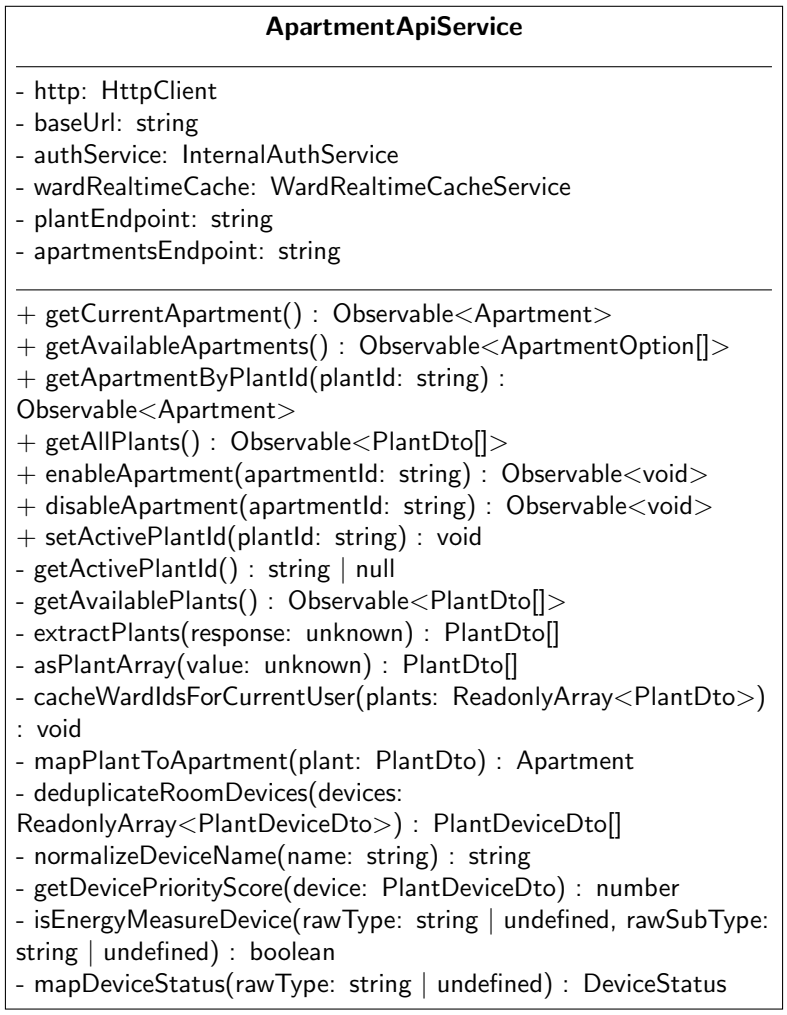

Figura 480: Diagramma della classe ApartmentApiService

Descrizione: Servizio Angular per la gestione delle chiamate API relative agli appartamenti e agli impianti. Espone operazioni per il recupero, la selezione e l'abilitazione degli appartamenti, gestendo internamente la mappatura dei dati e la cache dei ward.

Descrizione dei metodi della classe:

- `getCurrentApartment() : Observable<Apartment>`: Recupera l'appartamento corrente selezionando l'impianto attivo tra quelli disponibili e caricandone i dettagli.
- `getAvailableApartments() : Observable<ApartmentOption[]>`: Restituisce la lista degli appartamenti disponibili ordinata alfabeticamente per nome.
- `getApartmentByPlantId(plantId: string) : Observable<Apartment>`: Recupera i dettagli dell'appartamento associato all'identificativo di impianto specificato.
- `getAllPlants() : Observable<PlantDto[]>`: Recupera tutti gli impianti disponibili dal backend e aggiorna la cache dei ward per l'utente corrente.
- `enableApartment(apartmentId: string) : Observable<void>`: Invia una richiesta PATCH per abilitare l'appartamento con l'identificativo specificato.
- `disableApartment(apartmentId: string) : Observable<void>`: Invia una richiesta PATCH per disabilitare l'appartamento con l'identificativo specificato.

- `setActivePlantId(plantId: string) : void`: Salva l'identificativo dell'impianto attivo nel localStorage e dispatcha un evento custom se il valore è cambiato.
- `getActivePlantId() : string | null`: Recupera l'identificativo dell'impianto attivo dal localStorage, restituendo `null` se non disponibile.
- `getAvailablePlants() : Observable<PlantDto[]>`: Delega il recupero degli impianti disponibili al metodo `getAllPlants`.
- `extractPlants(response: unknown) : PlantDto[]`: Estrae l'array di impianti da una risposta generica del backend, gestendo sia array diretti che oggetti con proprietà `data` o `plants`.
- `asPlantArray(value: unknown) : PlantDto[]`: Converte un valore generico in un array di `PlantDto` validi, deduplicando per identificativo e normalizzando il nome.
- `cacheWardIdsForCurrentUser(plants: ReadonlyArray<PlantDto>) : void`: Estrae gli identificativi dei ward dagli impianti forniti e li memorizza nella cache per l'utente corrente.
- `mapPlantToApartment(plant: PlantDto) : Apartment`: Trasforma un `PlantDto` nel modello di dominio `Apartment`, mappando stanze e dispositivi e filtrando i dispositivi di misura energetica.
- `deduplicateRoomDevices(devices: ReadonlyArray<PlantDeviceDto>) : PlantDeviceDto[]`: Deduplica i dispositivi di una stanza per nome normalizzato, mantenendo quello con il punteggio di priorità più alto.
- `normalizeDeviceName(name: string) : string`: Normalizza il nome di un dispositivo rimuovendo spazi e convertendo in minuscolo.
- `getDevicePriorityScore(device: PlantDeviceDto) : number`: Calcola un punteggio di priorità per un dispositivo in base al tipo, sottotipo e numero di datapoint.
- `isEnergyMeasureDevice(rawType: string | undefined, rawSubType: string | undefined) : boolean`: Verifica se un dispositivo è di tipo misura energetica in base al tipo e sottotipo forniti.
- `mapDeviceStatus(rawType: string | undefined) : DeviceStatus`: Mappa il tipo grezzo di un dispositivo nel corrispondente stato, restituendo `ONLINE` se il tipo è definito, `UNKNOWN` altrimenti.

4.2.6 Core

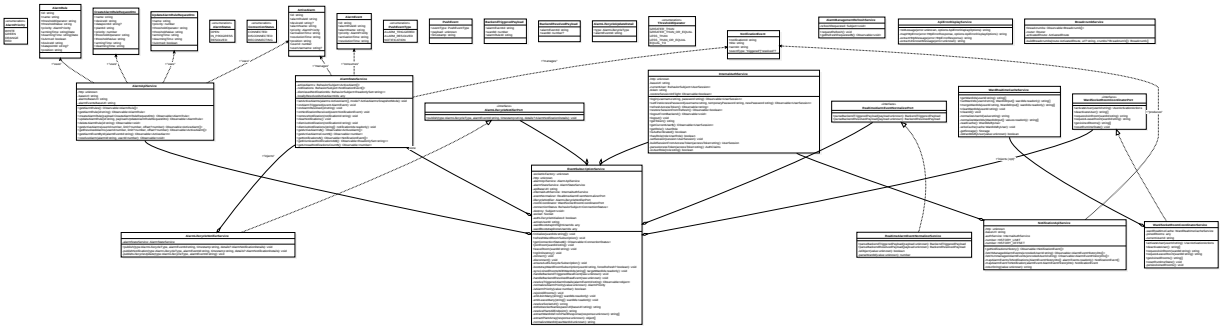

Figura 481: Diagramma delle classi del modulo Core

4.2.6.1 CreateAlarmRuleRequestDto

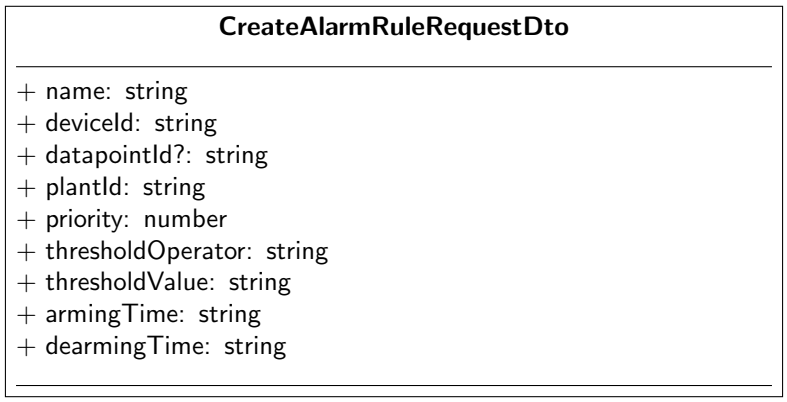

Figura 482: Diagramma della classe CreateAlarmRuleRequestDto

Descrizione: Interfaccia che rappresenta il DTO per la creazione di una nuova regola di allarme. Raccoglie tutti i campi necessari alla definizione della regola, inclusi dispositivo, soglia e orari di armamento.

4.2.6.2 UpdateAlarmRuleRequestDto

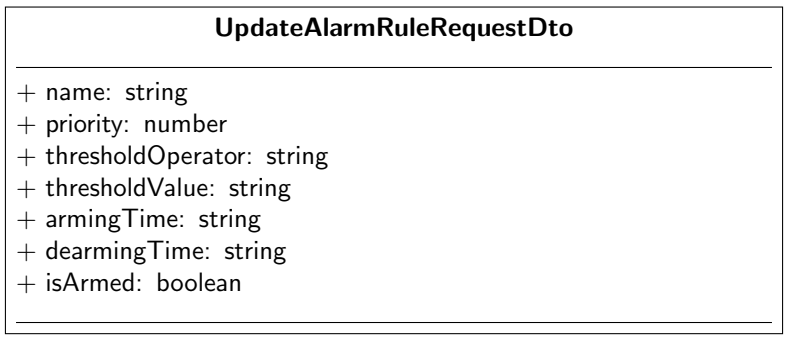

Figura 483: Diagramma della classe UpdateAlarmRuleRequestDto

Descrizione: DTO che definisce la struttura dei dati necessaria per aggiornare i parametri di una regola di allarme esistente. Permette di modificare i criteri di soglia, le tempistiche di attivazione e lo stato di armamento del sistema.

4.2.6.3 ActiveAlarm

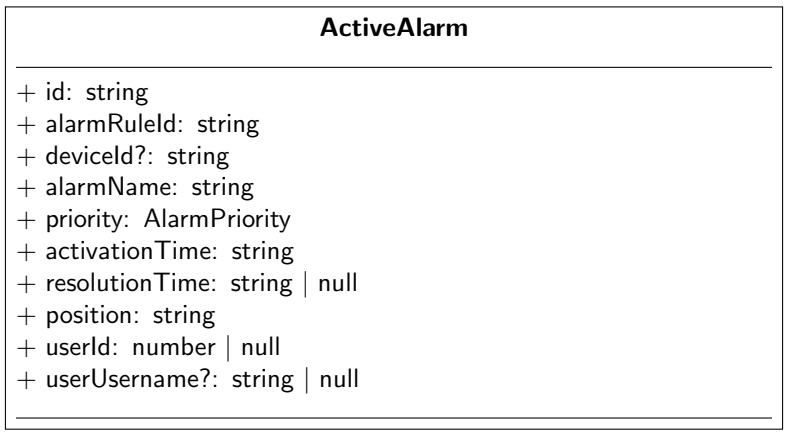

Figura 484: Diagramma della classe ActiveAlarm

Descrizione: Questa interfaccia definisce la struttura di un allarme attivo all'interno del sistema, raggruppando informazioni identificative, temporali e di localizzazione. Viene utilizzata per tracciare lo stato degli allarmi in corso, includendo i dettagli sulla priorità e l'eventuale utente assegnato alla risoluzione.

4.2.6.4 AlarmEvent

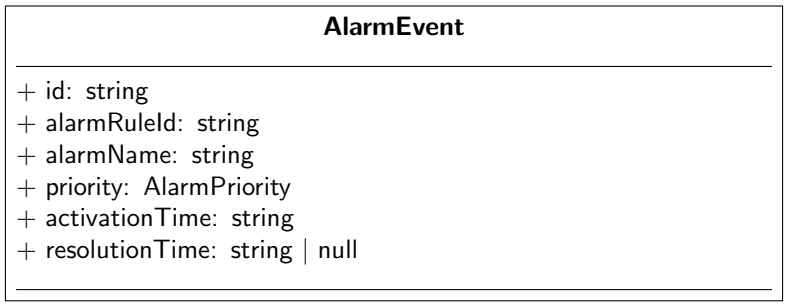

Figura 485: Diagramma della classe AlarmEvent

Descrizione: Questa interfaccia rappresenta un evento di allarme specifico, contenendo i dati essenziali relativi alla sua attivazione, risoluzione e livello di criticità. Fornisce una struttura standardizzata per tracciare lo storico e lo stato temporale delle notifiche generate dal sistema di monitoraggio.

4.2.6.5 AlarmRule

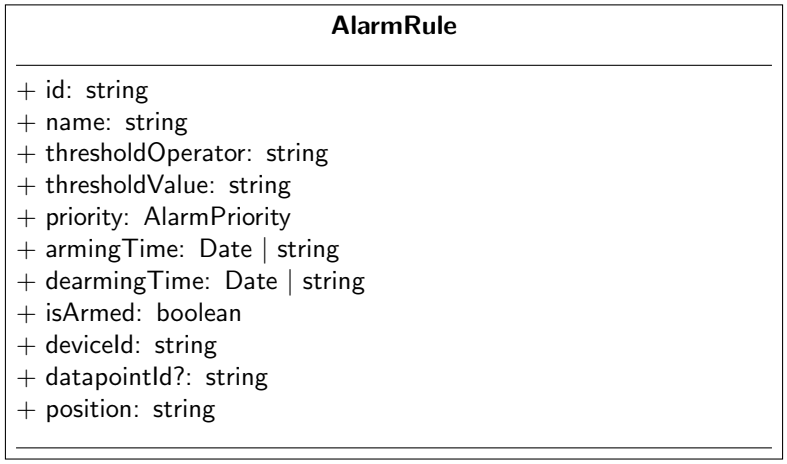

Figura 486: Diagramma della classe AlarmRule

Descrizione: Questa interfaccia definisce i criteri e le configurazioni di una regola di allarme applicata a un dispositivo o datapoint specifico. Include i parametri operativi come le soglie di intervento, i tempi di attivazione e il livello di priorità per il monitoraggio dei sistemi.

4.2.6.6 AlarmApiService

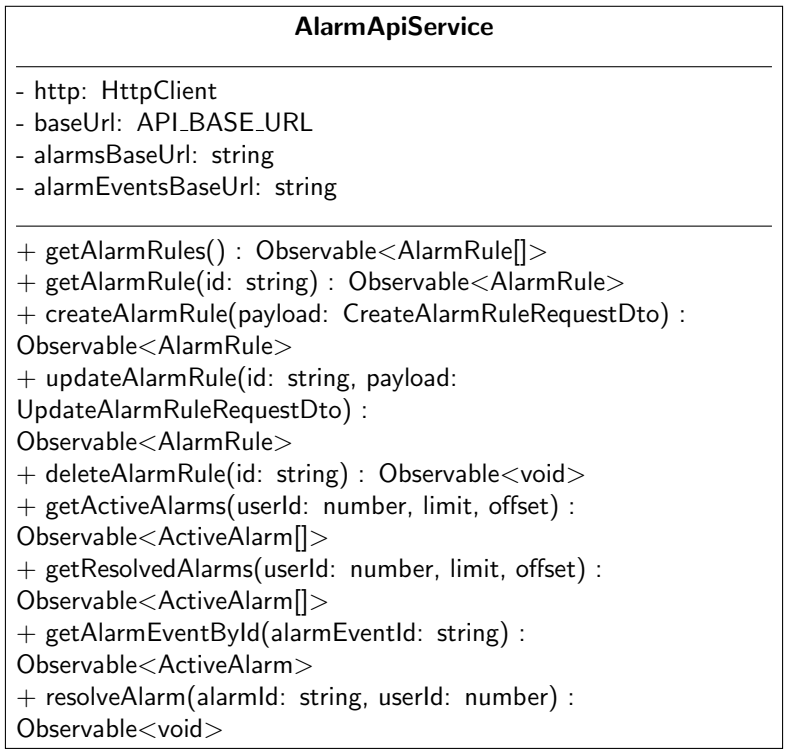

Figura 487: Diagramma della classe AlarmApiService

Descrizione: Questa classe di servizio gestisce tutte le comunicazioni HTTP relative alle regole di allarme e agli eventi di allarme verso l'API di backend. Centralizza le operazioni di CRUD per le configurazioni e il monitoraggio degli stati di attivazione e risoluzione degli allarmi.

Descrizione dei metodi della classe:

- `getAlarmRules() : Observable<AlarmRule[]>`: Recupera l'elenco completo di tutte le regole di allarme configurate nel sistema.
- `getAlarmRule(id: string) : Observable<AlarmRule>`: Ottiene i dettagli specifici di una singola regola di allarme tramite il suo identificativo univoco.
- `createAlarmRule(payload: CreateAlarmRuleRequestDto) : Observable<AlarmRule>`: Crea una nuova regola di allarme inviando i dati di configurazione al server.
- `updateAlarmRule(id: string, payload: UpdateAlarmRuleRequestDto) : Observable<AlarmRule>`: Aggiorna i parametri di una regola di allarme esistente identificata dall'ID fornito.
- `deleteAlarmRule(id: string) : Observable<void>`: Rimuove definitivamente una regola di allarme dal sistema.
- `getActiveAlarms(userId: number, limit, offset) : Observable<ActiveAlarm[]>`: Recupera la lista degli allarmi attivi e non ancora gestiti per un determinato utente con supporto alla paginazione.
- `getResolvedAlarms(userId: number, limit, offset) : Observable<ActiveAlarm[]>`: Ritorna lo storico degli allarmi già risolti e gestiti da uno specifico utente.
- `getAlarmEventById(alarmEventId: string) : Observable<ActiveAlarm>`: Recupera le informazioni dettagliate di uno specifico evento di allarme tramite il suo ID.
- `resolveAlarm(alarmId: string, userId: number) : Observable<void>`: Invia la richiesta per marcare un allarme specifico come risolto dall'utente indicato.

4.2.6.7 AlarmNotificationDetails

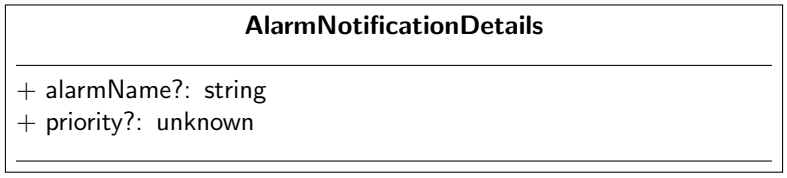

Figura 488: Diagramma della classe AlarmNotificationDetails

Descrizione: Questa interfaccia definisce i dettagli opzionali necessari per la costruzione del contenuto informativo di una notifica di allarme. Permette di trasportare il nome della regola e il suo livello di priorità per una visualizzazione corretta all'utente.

4.2.6.8 AlarmLifecycleNotifierPort

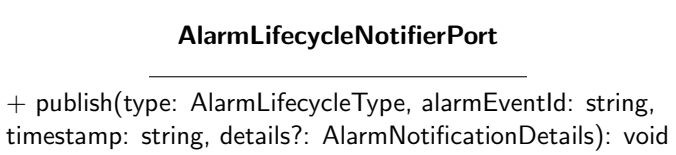

Figura 489: Diagramma dell'interfaccia AlarmLifecycleNotifierPort

Descrizione: Interfaccia di porta che astrae il meccanismo di notifica del ciclo di vita degli allarmi, permettendo l'invio di aggiornamenti su attivazioni e risoluzioni. Facilita il disaccoppiamento tra la logica di business e l'implementazione specifica della distribuzione degli eventi.

Descrizione dei metodi dell'interfaccia:

- `publish(type: AlarmLifecycleType, alarmEventId: string, timestamp: string, details?: AlarmNotificationDetails): void`: Definisce il contratto per la pubblicazione degli eventi legati al ciclo di vita di un allarme verso i sistemi di notifica.

4.2.6.9 AlarmLifecycleNotifierService

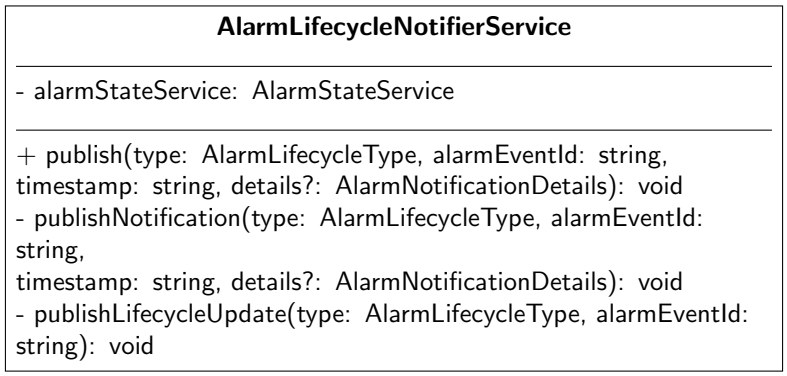

Figura 490: Diagramma della classe AlarmLifecycleNotifierService

Descrizione: Questo servizio implementa la logica per la gestione delle notifiche in tempo reale e degli aggiornamenti di stato relativi agli allarmi. Centralizza la creazione dei titoli delle notifiche e l'invio degli eventi tramite il dispatcher globale dell'applicazione.

Descrizione dei metodi della classe:

- `publish(type: AlarmLifecycleType, alarmEventId: string, timestamp: string, details?: AlarmNotificationDetails): void`: Coordina il processo di notifica globale attivando sia la creazione della notifica visiva che l'aggiornamento dello stato del ciclo di vita.
- `publishNotification(type: AlarmLifecycleType, alarmEventId: string, timestamp: string, details?: AlarmNotificationDetails): void`: Costruisce e invia una notifica formattata al servizio di stato degli allarmi in base al tipo di evento ricevuto.
- `publishLifecycleUpdate(type: AlarmLifecycleType, alarmEventId: string): void`: Emette un evento personalizzato a livello globale per informare i componenti dell'applicazione dell'avvenuta modifica di un allarme.

4.2.6.10 AlarmManagementRefreshService

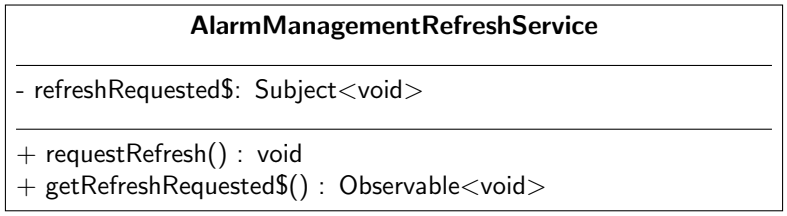

Figura 491: Diagramma della classe AlarmManagementRefreshService

Descrizione: Questo servizio fornisce un meccanismo centralizzato per coordinare l'aggiornamento dei dati relativi alla gestione degli allarmi all'interno dell'applicazione. Permette la comunicazione disaccoppiata tra diversi componenti che necessitano di sincronizzare il rinfresco delle interfacce.

Descrizione dei metodi della classe:

- `requestRefresh() : void`: Innesca una nuova richiesta di aggiornamento emettendo un segnale attraverso il relativo stream.
- `getRefreshRequested$() : Observable<void>`: Restituisce un Observable che permette ai componenti di sottoscriversi per reagire alle richieste di refresh.

4.2.6.11 AlarmStateService

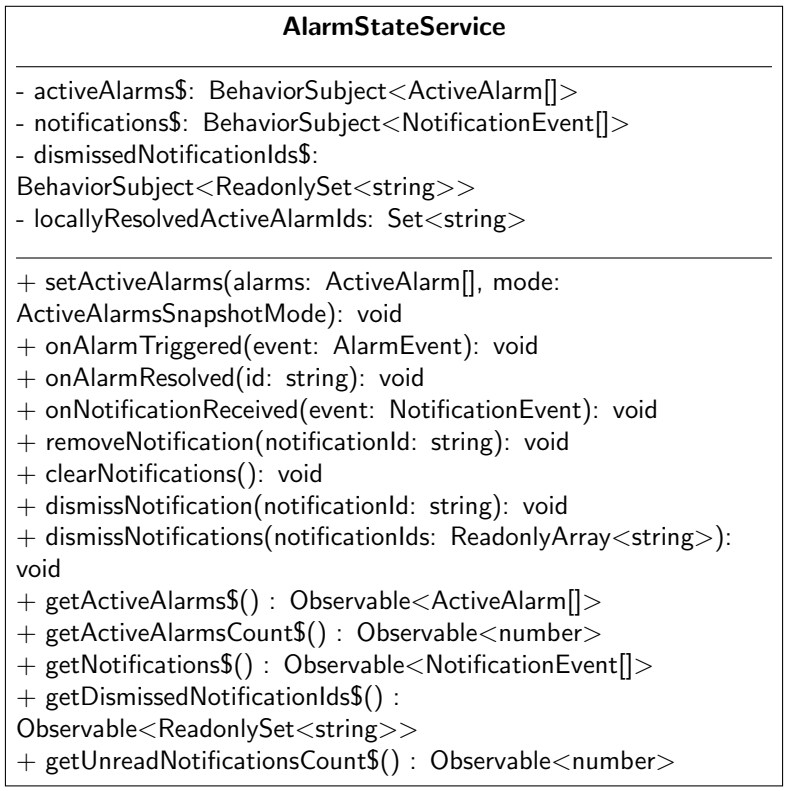

Figura 492: Diagramma della classe AlarmStateService

Descrizione: Questo servizio centralizza la gestione dello stato reattivo degli allarmi e delle notifiche all'interno del frontend dell'applicazione. Si occupa di mantenere la sincronia tra gli eventi ricevuti in tempo reale, le azioni dell'utente e la visualizzazione corretta dei contatori e delle liste.

Descrizione dei metodi della classe:

- `setActiveAlarms(alarms: ActiveAlarm[], mode: ActiveAlarmsSnapshotMode): void`: Aggiorna lo stato degli allarmi attivi permettendo di unire i nuovi dati a quelli esistenti o di sostituirli completamente.
- `onAlarmTriggered(event: AlarmEvent): void`: Gestisce l'attivazione di un nuovo allarme aggiungendolo alla lista locale e resettando eventuali stati di risoluzione precedente.
- `onAlarmResolved(id: string): void`: Marca un allarme come risolto rimuovendolo dalla lista degli allarmi attivi e tenendo traccia dell'operazione a livello locale.
- `onNotificationReceived(event: NotificationEvent): void`: Registra la ricezione di una nuova notifica verificando che non sia stata precedentemente scartata dall'utente.
- `removeNotification(notificationId: string): void`: Elimina una specifica notifica dall'elenco visualizzato e la aggiunge a quelle rimosse definitivamente.
- `clearNotifications(): void`: Rimuove istantaneamente tutte le notifiche correnti e le marca come scartate nel sistema.
- `dismissNotification(notificationId: string): void`: Aggiunge l'identificativo di una singola notifica alla lista di quelle ignorate per evitarne la ricomparsa.
- `dismissNotifications(notificationIds: ReadonlyArray<string>): void`: Gestisce l'inserimento massivo di più notifiche nella lista di quelle ignorate tramite un'operazione atomica.

- `getActiveAlarms$() : Observable<ActiveAlarm[]>`: Fornisce uno stream osservabile per monitorare in tempo reale i cambiamenti nella lista degli allarmi attivi.
- `getActiveAlarmsCount$() : Observable<number>`: Restituisce un osservabile che emette il numero totale di allarmi attualmente attivi nel sistema.
- `getNotifications$() : Observable<NotificationEvent[]>`: Permette di sottoscriversi per ricevere aggiornamenti sulla lista delle notifiche correnti non ancora rimosse.
- `getDismissedNotificationIds$() : Observable<ReadonlySet<string>>`: Restituisce l'insieme degli identificativi delle notifiche che l'utente ha scelto di ignorare.
- `getUnreadNotificationsCount$() : Observable<number>`: Ritorna il conteggio delle notifiche presenti nell'elenco degli avvisi correnti.

4.2.6.12 EventSubscriptionService

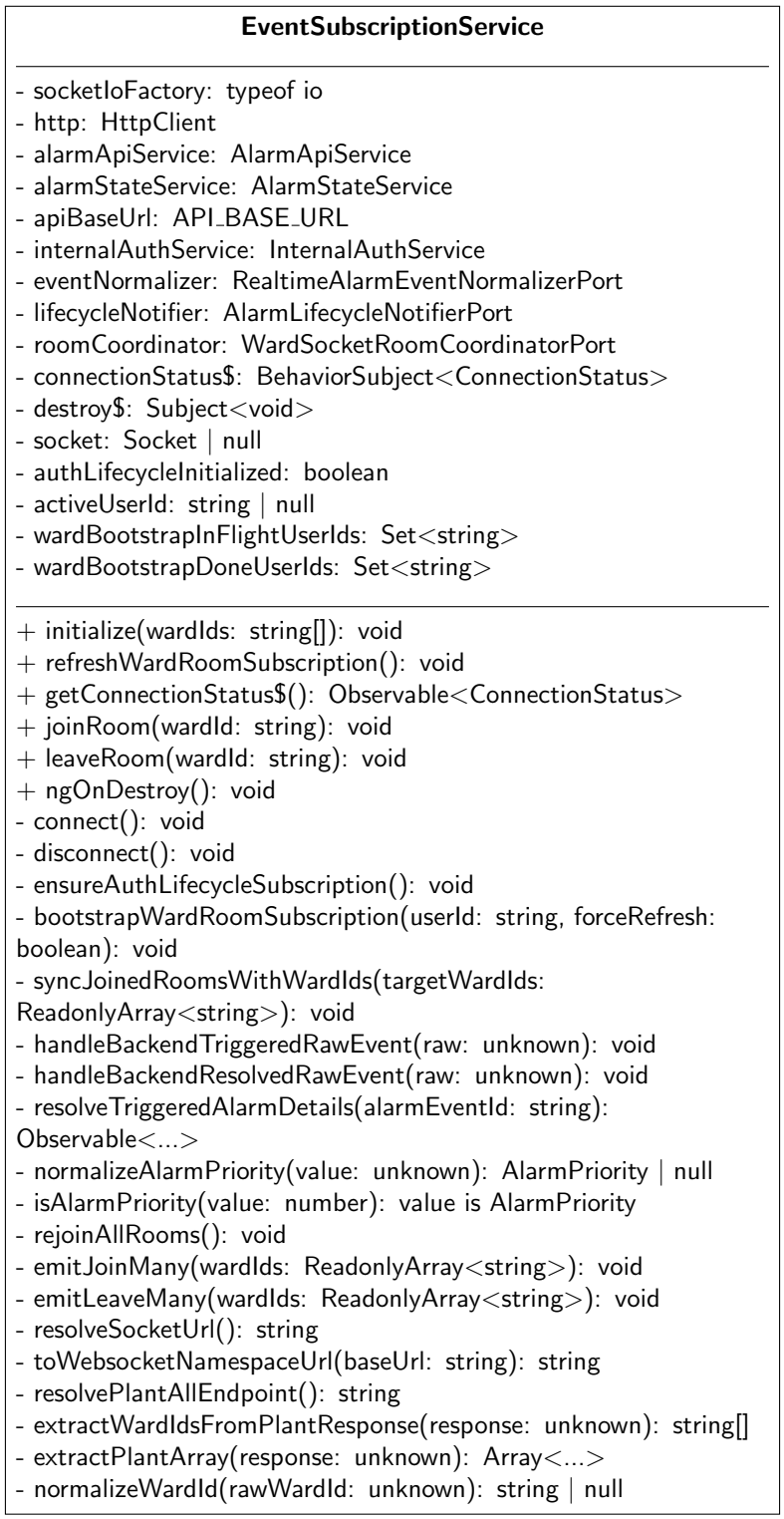

Figura 493: Diagramma della classe EventSubscriptionService

Descrizione: Questo servizio coordina la comunicazione in tempo reale tramite WebSocket, gestendo le sottoscrizioni alle stanze dei reparti e la ricezione degli eventi di allarme. Si occupa di mantenere la sincronia tra la sessione dell'utente, i dati del database e le notifiche live integrate nel sistema.

Descrizione dei metodi della classe:

- `initialize(wardIds: string[]): void`: Configura la connessione iniziale ai socket, sottoscrive il ciclo di vita dell'autenticazione e unisce l'utente alle stanze dei reparti specificati.
- `refreshWardRoomSubscription(): void`: Forza l'aggiornamento delle iscrizioni alle stanze dei reparti per l'utente attualmente attivo.
- `getConnectionStatus$(): Observable<ConnectionStatus>`: Restituisce uno stream osservabile per monitorare lo stato della connessione WebSocket in tempo reale.
- `joinRoom(wardId: string): void`: Richiede l'iscrizione a una specifica stanza di reparto tramite il protocollo socket.
- `leaveRoom(wardId: string): void`: Invia una richiesta al server per abbandonare la stanza associata a un determinato reparto.
- `ngOnDestroy(): void`: Gestisce la pulizia delle risorse chiudendo la connessione socket e completando gli stream alla distruzione del servizio.
- `connect(): void`: Inizializza la connessione WebSocket impostando i gestori degli eventi per la riconnessione e la ricezione dei messaggi dal backend.
- `disconnect(): void`: Chiude la connessione attiva, rimuove i listener e resetta lo stato interno del coordinatore delle stanze.
- `ensureAuthLifecycleSubscription(): void`: Sottoscrive i cambiamenti dello stato dell'utente per gestire automaticamente l'ingresso o l'uscita dalle stanze in base alla sessione.
- `bootstrapWardRoomSubscription(userId: string, forceRefresh: boolean): void`: Esegue il recupero iniziale dei reparti disponibili tramite API per configurare le sottoscrizioni socket dell'utente.
- `syncJoinedRoomsWithWardIds(targetWardIds: ReadonlyArray<string>): void`: Sincronizza le stanze attualmente seguite con un nuovo elenco di reparti target, abbandonando quelle non più necessarie.
- `handleBackendTriggeredRawEvent(raw: unknown): void`: Processa i dati grezzi di un allarme attivato, normalizzandoli e notificando il sistema di stato e il ciclo di vita.
- `handleBackendResolvedRawEvent(raw: unknown): void`: Gestisce la ricezione di un evento di risoluzione allarme, aggiornando lo stato locale e i notificatori.
- `resolveTriggeredAlarmDetails(alarmEventId: string): Observable<...>`: Recupera i dettagli informativi di un allarme dal server per arricchire l'evento ricevuto in tempo reale.
- `normalizeAlarmPriority(value: unknown): AlarmPriority | null`: Converte e valida i valori di priorità provenienti dal backend nel formato enumerato corretto.
- `isAlarmPriority(value: number): value is AlarmPriority`: Verifica se un valore numerico corrisponde a uno dei livelli di priorità validi definiti nell'applicazione.
- `rejoinAllRooms(): void`: Invia nuovamente i comandi di join per tutte le stanze precedentemente attive in seguito a una riconnessione.
- `emitJoinMany(wardIds: ReadonlyArray<string>): void`: Invia in blocco comandi di ingresso per una lista di identificativi di reparto.
- `emitLeaveMany(wardIds: ReadonlyArray<string>): void`: Invia in blocco comandi di uscita per una lista di identificativi di reparto.
- `resolveSocketUrl(): string`: Determina l'URL corretto per la connessione WebSocket basandosi sulla configurazione dell'API base.

- `toWebsocketNamespaceUrl(baseUrl: string): string`: Trasforma un URL API standard nel formato richiesto per il namespace dei WebSocket.
- `resolvePlantAllEndpoint(): string`: Costruisce l'endpoint URL necessario per recuperare l'elenco di tutti i reparti (plants).
- `extractWardIdsFromPlantResponse(response: unknown): string[]`: Analizza la risposta del server per estrarre una lista univoca di identificativi di reparto.
- `extractPlantArray(response: unknown): Array<...>`: Estrae in modo sicuro l'array di dati dei reparti da diverse possibili strutture di risposta JSON.
- `normalizeWardId(rawWardId: unknown): string | null`: Normalizza l'identificativo di un reparto convertendolo in una stringa valida o restituendo null.

4.2.6.13 RealtimeAlarmEventNormalizerPort

RealtimeAlarmEventNormalizerPort

+ parseBackendTriggeredPayload(payload: unknown): BackendTriggeredPayload | null
+ parseBackendResolvedPayload(payload: unknown): BackendResolvedPayload | null

Figura 494: Diagramma dell'interfaccia RealtimeAlarmEventNormalizerPort

Descrizione: Interfaccia di porta che definisce il contratto per la normalizzazione dei messaggi ricevuti in tempo reale tramite protocolli di comunicazione asincrona. Garantisce che i dati provenienti dall'esterno siano validati prima di essere processati dalla logica di business.

Descrizione dei metodi dell'interfaccia:

- `parseBackendTriggeredPayload(payload: unknown): BackendTriggeredPayload | null`: Valida e converte i dati grezzi ricevuti dal backend relativi all'attivazione di un nuovo allarme in un oggetto tipizzato.
- `parseBackendResolvedPayload(payload: unknown): BackendResolvedPayload | null`: Esegue il parsing dei dati grezzi riguardanti la risoluzione di un allarme, assicurando l'integrità dei campi obbligatori.

4.2.6.14 RealtimeAlarmEventNormalizerService

RealtimeAlarmEventNormalizerService

+ parseBackendTriggeredPayload(payload: unknown): BackendTriggeredPayload | null
+ parseBackendResolvedPayload(payload: unknown): BackendResolvedPayload | null
- isObject(value: unknown): value is Record<string, unknown>
- parseWardId(value: unknown): number | null

Figura 495: Diagramma della classe RealtimeAlarmEventNormalizerService

Descrizione: Servizio responsabile della trasformazione e validazione dei dati grezzi in arrivo dal backend via socket in strutture dati coerenti per l'applicazione. Centralizza i controlli di tipo e le conversioni di formato per prevenire errori di runtime dovuti a payload malformati.

Descrizione dei metodi della classe:

- `parseBackendTriggeredPayload(payload: unknown): BackendTriggeredPayload | null`: Implementa la logica di estrazione e validazione per i payload di attivazione allarme, gestendo la conversione degli identificativi di reparto.
- `parseBackendResolvedPayload(payload: unknown): BackendResolvedPayload | null`: Analizza il contenuto dei messaggi di risoluzione allarme, estraendo l'identificativo dell'evento e il relativo reparto associato.
- `isObject(value: unknown): value is Record<string, unknown>`: Verifica internamente se un valore sconosciuto è un oggetto non nullo compatibile con un record di chiavi e valori.
- `parseWardId(value: unknown): number | null`: Tenta di convertire un valore di input in un identificativo numerico intero valido per la gestione dei reparti.

4.2.6.15 WardRealtimeCacheService

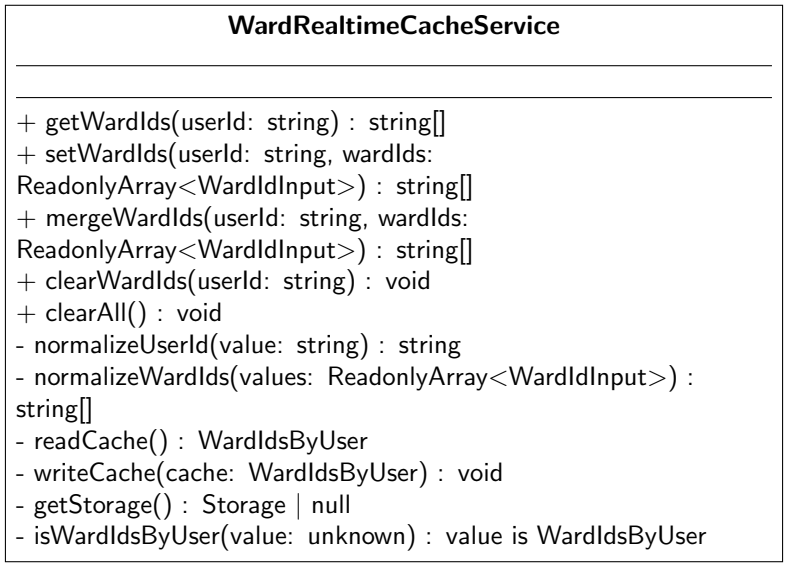

Figura 496: Diagramma della classe WardRealtimeCacheService

Descrizione: Questo servizio gestisce la persistenza temporanea degli identificativi di reparto su base utente utilizzando la session storage del browser. Assicura che le preferenze di monitoraggio in tempo reale siano mantenute durante la sessione, offrendo metodi per la normalizzazione, il recupero e la sincronizzazione dei dati.

Descrizione dei metodi della classe:

- `getWardIds(userId: string) : string[]`: Recupera dalla cache l'elenco degli identificativi di reparto associati a uno specifico utente.
- `setWardIds(userId: string, wardIds: ReadonlyArray<WardIdInput>) : string[]`: Sovrascrive la lista dei reparti per l'utente indicato, normalizzando i valori prima della memorizzazione.

- `mergeWardIds(userId: string, wardIds: ReadonlyArray<WardIdInput>) : string[]`: Unisce i nuovi identificativi di reparto a quelli già presenti nella cache per l'utente specificato.
- `clearWardIds(userId: string) : void`: Rimuove definitivamente dalla cache tutte le informazioni relative ai reparti di un singolo utente.
- `clearAll() : void`: Svuota completamente lo storage locale rimuovendo ogni dato salvato dal servizio di caching.
- `normalizeUserId(value: string) : string`: Rimuove gli spazi bianchi in eccesso dall'identificativo utente per garantire la coerenza delle chiavi.
- `normalizeWardIds(values: ReadonlyArray<WardIdInput>) : string[]`: Converte i valori di input in una lista di stringhe univoche e validate, eliminando duplicati e valori nulli.
- `readCache() : WardIdsByUser`: Legge e deserializza i dati dalla session storage, gestendo eventuali errori di parsing o formati non validi.
- `writeCache(cache: WardIdsByUser) : void`: Serializza e salva lo stato corrente della cache nella session storage del browser.
- `getStorage() : Storage | null`: Fornisce un riferimento sicuro alla session storage globale dell'ambiente di esecuzione.
- `isWardIdsByUser(value: unknown) : value is WardIdsByUser`: Esegue una validazione strutturale profonda per assicurarsi che i dati caricati corrispondano al formato atteso dal servizio.

4.2.6.16 UserActivationActions

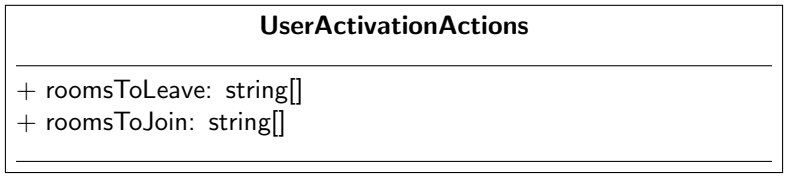

Figura 497: Diagramma della classe UserActivationActions

Descrizione: Questa interfaccia definisce la struttura degli insiemi di stanze socket da gestire durante la transizione di sessione di un utente. Specifica quali canali devono essere abbandonati e quali attivati per garantire la corretta ricezione degli eventi in tempo reale.

4.2.6.17 WardSocketRoomCoordinatorPort

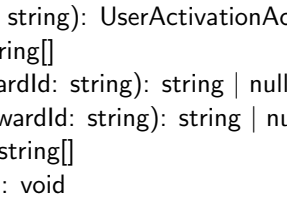

Figura 498: Diagramma dell'interfaccia WardSocketRoomCoordinatorPort

Descrizione: Interfaccia di porta che definisce la logica di coordinamento per la gestione delle stanze socket associate ai reparti ospedalieri. Astrae la complessità della sincronizzazione tra lo stato della sessione utente e le sottoscrizioni ai canali di comunicazione in tempo reale.

Descrizione dei metodi dell'interfaccia:

- `activateUser(userId: string): UserActivationActions`: Gestisce la transizione di un utente calcolando le stanze da abbandonare e quelle da sottoscrivere in base alla cache.
- `deactivateUser(): string[]`: Esegue il logout dell'utente corrente rimuovendo le informazioni dalla cache e restituendo l'elenco delle stanze da abbandonare.
- `requestJoinRoom(wardId: string): string | null`: Verifica se l'iscrizione a una nuova stanza di reparto è necessaria e, in caso positivo, la registra nello stato interno.
- `requestLeaveRoom(wardId: string): string | null`: Rimuove un reparto dallo stato di monitoraggio attivo e restituisce l'identificativo se l'operazione è andata a buon fine.
- `getJoinedRooms(): string[]`: Restituisce l'elenco completo di tutti gli identificativi di reparto attualmente monitorati dall'istanza attiva.
- `resetRuntimeState(): void`: Ripristina lo stato iniziale del coordinatore azzerando l'utente corrente e svuotando l'elenco delle stanze attive.

4.2.6.18 WardSocketRoomCoordinatorService

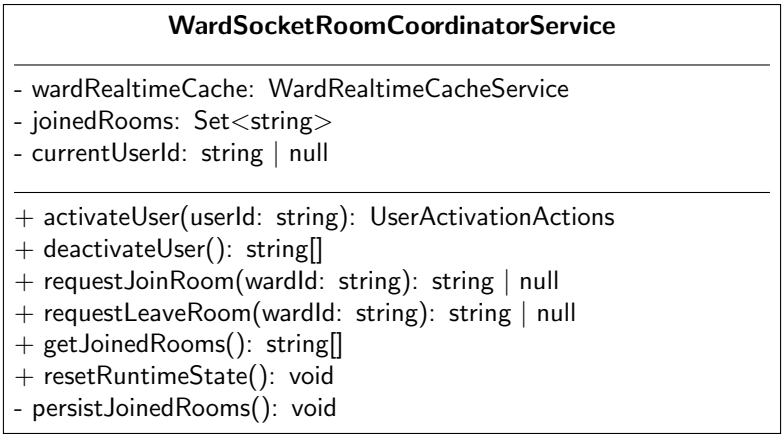

Figura 499: Diagramma della classe WardSocketRoomCoordinatorService

Descrizione: Questo servizio coordina in modo reattivo l'iscrizione e la disiscrizione dalle stanze socket relative ai reparti, basandosi sulla sessione utente corrente. Assicura la persistenza delle preferenze di monitoraggio tramite il servizio di cache e calcola le azioni necessarie per mantenere la sincronia con il server WebSocket.

Descrizione dei metodi della classe:

- `activateUser(userId: string): UserActivationActions`: Implementa il cambio utente sincronizzando lo stato dei socket con i dati persistiti nella cache per il nuovo identificativo fornito.
- `deactivateUser(): string[]`: Pulisce lo stato di sessione e la cache locale, fornendo l'elenco delle stanze che devono essere chiuse lato socket.
- `requestJoinRoom(wardId: string): string | null`: Aggiunge un reparto all'insieme delle stanze monitorate e persiste immediatamente la modifica nello storage.

- `requestLeaveRoom(wardId: string): string | null`: Elimina un reparto dal set di monitoraggio e aggiorna lo stato persistente della cache utente.
- `getJoinedRooms(): string[]`: Fornisce una copia array dell'insieme delle stanze attualmente gestite internamente dal servizio.
- `resetRuntimeState(): void`: Azzera le variabili di stato in memoria per prevenire perdite di dati o stati incoerenti tra diverse sessioni.
- `persistJoinedRooms(): void`: Aggiorna lo storage persistente con l'elenco attuale delle stanze monitorate per l'utente attivo.

—

4.2.6.19 ApiErrorDisplayOptions

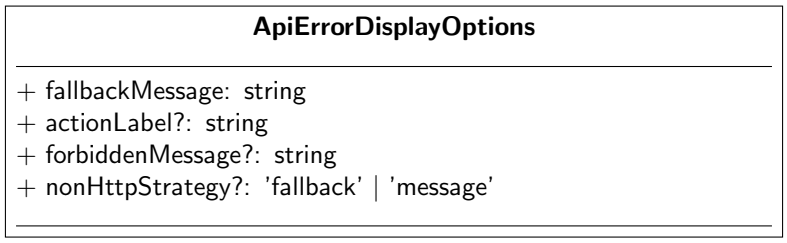

Figura 500: Diagramma della classe ApiErrorDisplayOptions

Descrizione: Questa interfaccia definisce le opzioni di configurazione per la generazione dei messaggi di errore rivolti all'utente. Permette di personalizzare i messaggi di fallback, le etichette delle azioni e le strategie di gestione per errori non strettamente legati al protocollo HTTP.

4.2.6.20 ApiErrorDisplayService

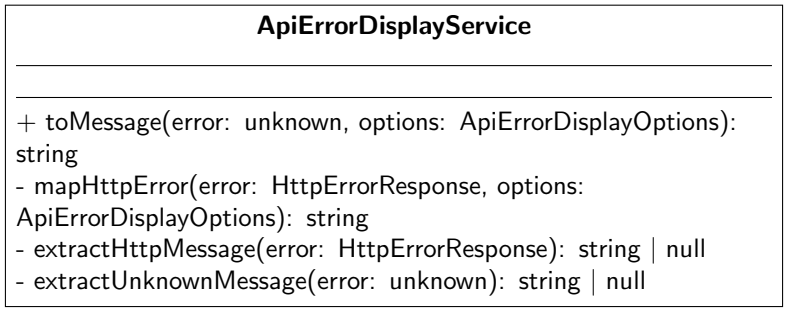

Figura 501: Diagramma della classe ApiErrorDisplayService

Descrizione: Questo servizio centralizza la logica di trasformazione degli errori tecnici in messaggi comprensibili per l'utente finale. Fornisce un'astrazione coerente per gestire le risposte del server e i fallimenti applicativi, migliorando l'esperienza utente attraverso feedback chiari e contestualizzati.

Descrizione dei metodi della classe:

- `toMessage(error: unknown, options: ApiErrorDisplayOptions): string`: Converte un errore di natura ignota in una stringa leggibile dall'utente, applicando logiche di mappatura differenti in base al tipo di eccezione catturata.

- `mapHttpError(error: HttpErrorResponse, options: ApiErrorDisplayOptions): string`: Analizza il codice di stato HTTP per restituire messaggi localizzati e contestualizzati, gestendo scenari come la perdita di connessione, sessioni scadute o permessi insufficienti.
- `extractHttpMessage(error: HttpErrorResponse): string | null`: Tenta di estrarre un messaggio di errore descrittivo direttamente dal corpo della risposta HTTP, gestendo formati testuali, oggetti JSON o array di messaggi.
- `extractUnknownMessage(error: unknown): string | null`: Cerca di recuperare una descrizione testuale da eccezioni generiche o oggetti JavaScript che non derivano dalla comunicazione di rete.

4.2.6.21 Breadcrumb

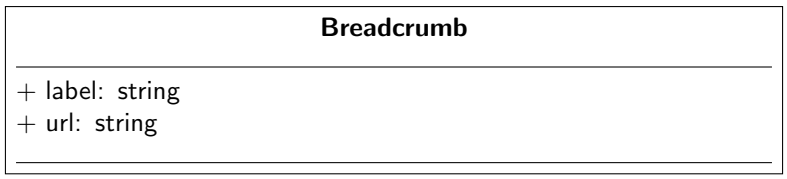

Figura 502: Diagramma della classe Breadcrumb

Descrizione: Questa interfaccia definisce la struttura di un singolo frammento del percorso di navigazione (breadcrumb). Associa un'etichetta testuale descrittiva all'URL corrispondente per permettere all'utente di risalire la gerarchia delle pagine.

4.2.6.22 BreadcrumbService

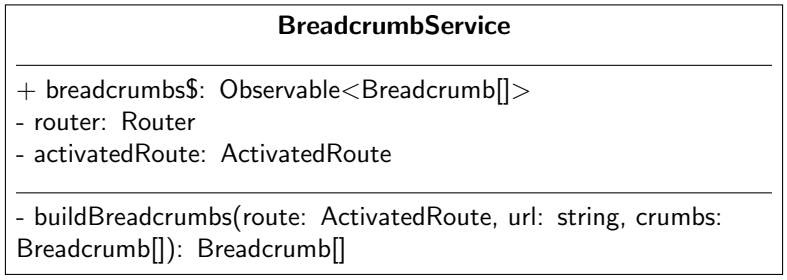

Figura 503: Diagramma della classe BreadcrumbService

Descrizione: Servizio responsabile della generazione dinamica del percorso di navigazione basato sullo stato corrente del router. Monitora gli eventi di navigazione per aggiornare automaticamente uno stream reattivo di oggetti Breadcrumb, facilitando l'orientamento dell'utente all'interno dell'applicazione.

Descrizione dei metodi della classe:

- `buildBreadcrumbs(route: ActivatedRoute, url: string, crumbs: Breadcrumb[]): Breadcrumb[]`: Analizza ricorsivamente l'albero delle rotte attive per costruire l'elenco dei breadcrumb, estraendo le etichette dai dati di configurazione delle rotte.

4.2.6.23 InternalAuthService

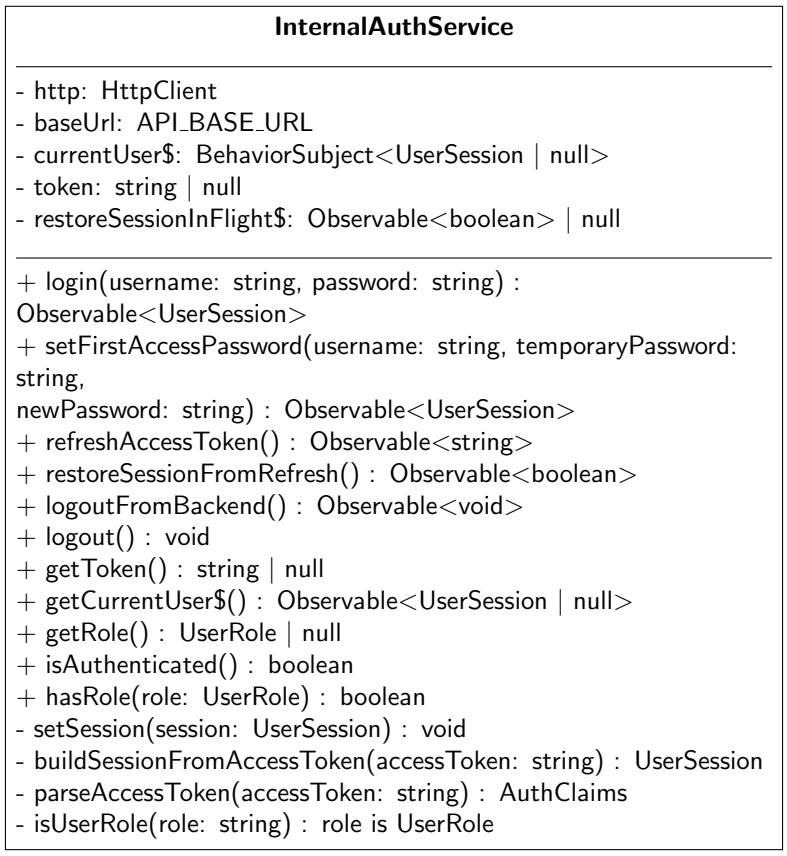

Figura 504: Diagramma della classe InternalAuthService

Descrizione: Servizio centrale per la gestione del ciclo di vita dell'autenticazione. Si occupa del login, del rinnovo dei token JWT, del ripristino delle sessioni e della decodifica delle autorizzazioni utente, garantendo che lo stato di sicurezza sia sincronizzato tra il backend e l'interfaccia frontend.

Descrizione dei metodi della classe:

- `login(username: string, password: string) : Observable<UserSession>`: Esegue la procedura di autenticazione inviando le credenziali al backend e inizializzando la sessione utente in caso di successo.
- `setFirstAccessPassword(username: string, temporaryPassword: string, newPassword: string) : Observable<UserSession>`: Gestisce la procedura di primo accesso permettendo all'utente di sostituire la password temporanea e ottenere un token definitivo.
- `refreshAccessToken() : Observable<string>`: Richiede al server un nuovo token di accesso utilizzando il meccanismo di refresh basato su cookie, aggiornando la sessione corrente.
- `restoreSessionFromRefresh() : Observable<boolean>`: Tenta di ripristinare una sessione esistente all'avvio dell'applicazione. Gestisce le richieste concorrenti tramite un meccanismo di caching dello stream attivo.
- `logoutFromBackend() : Observable<void>`: Informa il server della chiusura della sessione e procede alla pulizia dei dati di autenticazione locali.
- `logout() : void`: Esegue la pulizia immediata dello stato di autenticazione locale, rimuovendo il token e resettando l'utente corrente.
- `getToken() : string | null`: Restituisce il token JWT attualmente memorizzato per l'inclusione nelle chiamate HTTP protette.

- `getCurrentUser$() : Observable<UserSession | null>`: Fornisce uno stream reattivo per monitorare le informazioni dell'utente loggato in tutta l'applicazione.
- `getRole() : UserRole | null`: Recupera il ruolo dell'utente dalla sessione attiva, se presente.
- `isAuthenticated() : boolean`: Verifica se è presente una sessione valida e un token di accesso attivo.
- `hasRole(role: UserRole) : boolean`: Controlla se l'utente corrente possiede il ruolo specifico passato come parametro.
- `setSession(session: UserSession) : void`: Aggiorna internamente il token e il BehaviorSubject della sessione utente.
- `buildSessionFromAccessToken(accessToken: string) : UserSession`: Decodifica il token JWT e mappa i claim estratti nel modello di sessione applicativo, validando la presenza dei campi obbligatori.
- `parseAccessToken(accessToken: string) : AuthClaims`: Esegue il parsing della parte payload del token JWT gestendo la codifica Base64URL.
- `isUserRole(role: string) : role is UserRole`: Valida se la stringa del ruolo estratta dal token corrisponde a uno dei ruoli definiti nel sistema.

4.2.6.24 IVimarCloudApiService

IVimarCloudApiService

+ getLinkedAccount() : Observable<MyVimarAccount>
+ initiateOAuth() : void
+ unlinkAccount() : Observable<void>

Figura 505: Diagramma dell'interfaccia IVimarCloudApiService

Descrizione: Interfaccia di porta che definisce il contratto per l'integrazione con i servizi cloud di Vimar. Astrae le operazioni di gestione dell'account, autenticazione e sincronizzazione tra le diverse piattaforme, garantendo un'integrazione disaccoppiata.

Descrizione dei metodi dell'interfaccia:

- `getLinkedAccount() : Observable<MyVimarAccount>`: Recupera le informazioni relative all'account Vimar attualmente collegato al sistema.
- `initiateOAuth() : void`: Avvia la procedura di autenticazione delegata tramite protocollo OAuth per stabilire la connessione con il cloud Vimar.
- `unlinkAccount() : Observable<void>`: Rimuove l'associazione tra l'account locale e l'account Vimar Cloud, revocando l'integrazione.

4.2.7 Dashboard

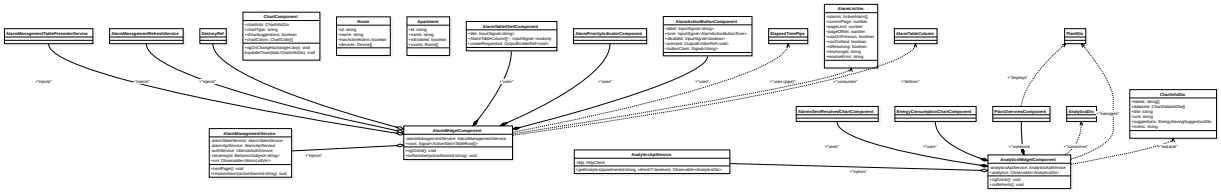

Figura 506: Diagramma delle classi del modulo Dashboard

4.2.7.1 AlarmWidgetComponent

AlarmWidgetComponent

- alarmManagementService: AlarmManagementService
- tablePresenter: AlarmManagementTablePresenterService
- alarmManagementRefreshService: AlarmManagementRefreshService
- destroyRef: DestroyRef
+ columns: readonly AlarmTableColumn[]
+ vm: Signal<AlarmListVm | null>
+ rows: Signal<ActiveAlarmTableRow[]>
+ canGoPrevious: Signal<boolean>
+ canGoNext: Signal<boolean>
+ currentPage: Signal<number>

+ ngOnInit() : void
+ onResolve(activeAlarmId: string) : void

Figura 507: Diagramma della classe AlarmWidgetComponent

Descrizione: Componente Angular che rappresenta il widget degli allarmi attivi nella dashboard, con supporto alla paginazione e alla risoluzione. Si aggiorna automaticamente in risposta alle notifiche di refresh provenienti dal servizio dedicato.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il servizio di gestione allarmi e sottoscrive le notifiche di refresh per reinizializzare i dati.
- `onResolve(activeAlarmId: string) : void`: Delega al servizio di gestione la risoluzione dell'allarme con l'identificativo specificato.

4.2.7.2 AnalyticsWidgetComponent

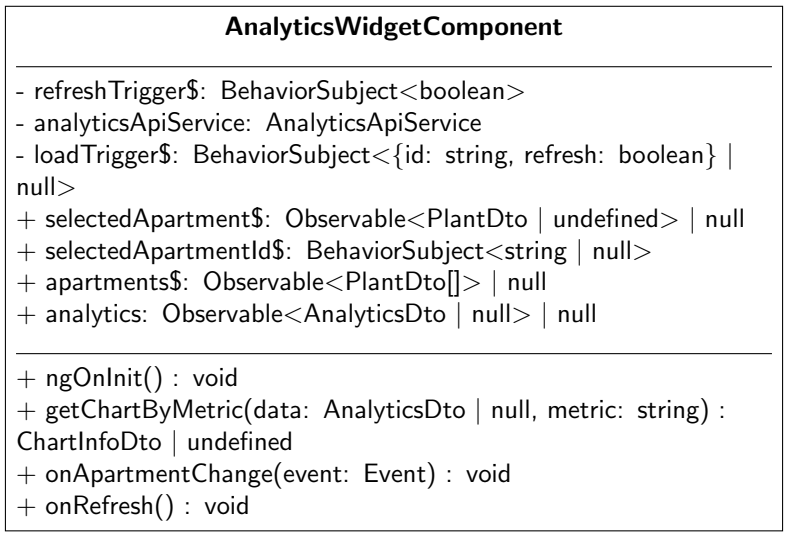

Figura 508: Diagramma della classe AnalyticsWidgetComponent

Descrizione: Componente Angular che rappresenta il widget di analisi nella dashboard, con supporto alla selezione dell'appartamento e al caricamento dei dati analitici. Gestisce gli stream reattivi per l'appartamento selezionato, i grafici e il refresh manuale dei dati.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il caricamento degli appartamenti, seleziona il primo disponibile e configura gli stream reattivi per l'appartamento selezionato e i dati analitici.
- `getChartByMetric(data: AnalyticsDto | null, metric: string) : ChartInfoDto | undefined`: Restituisce il grafico corrispondente alla metrica specificata dai dati analitici, o undefined se non trovato.
- `onApartmentChange(event: Event) : void`: Aggiorna l'appartamento selezionato e innesca il caricamento dei relativi dati analitici.
- `onRefresh() : void`: Innesca il ricaricamento forzato dei dati analitici per l'appartamento correntemente selezionato.

4.2.7.3 PlantOverviewComponent

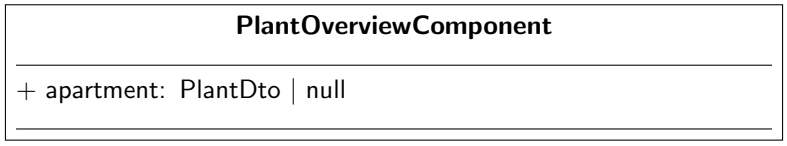

Figura 509: Diagramma della classe PlantOverviewComponent

Descrizione: Componente Angular che visualizza una panoramica delle informazioni di un impianto. Riceve i dati dell'impianto tramite input e li presenta nel template dedicato.

4.2.8 Device-interaction

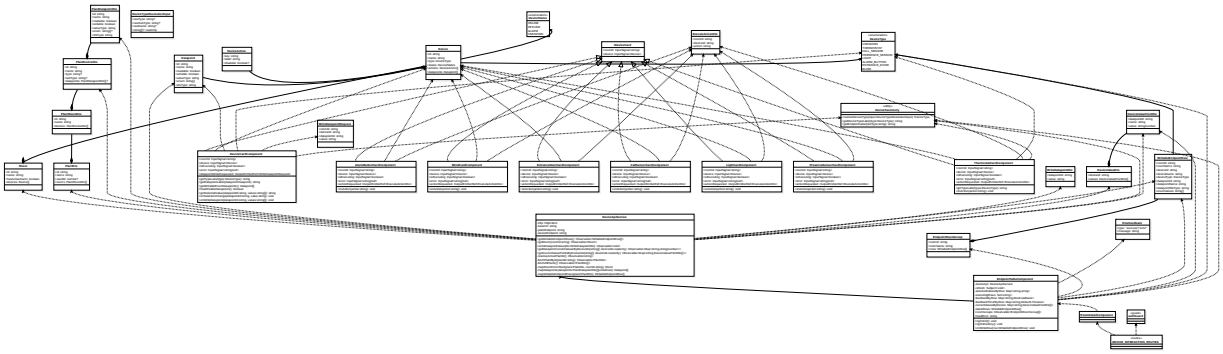

Figura 510: Diagramma delle classi del modulo Device interaction

4.2.8.1 AlarmButtonCardComponent

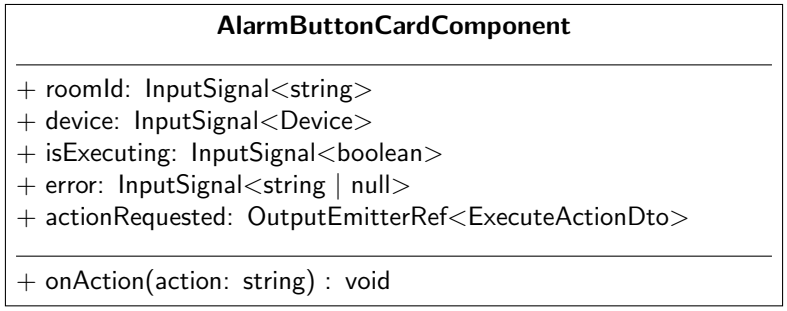

Figura 511: Diagramma della classe AlarmButtonCardComponent

Descrizione: Componente Angular che rappresenta una card per un dispositivo di tipo pulsante allarme. Implementa IDeviceCard ed emette le azioni richieste dall'utente verso il componente padre.

Descrizione dei metodi della classe:

- `onAction(action: string) : void`: Emette l'evento `actionRequested` con i dati necessari all'esecuzione dell'azione richiesta.

4.2.8.2 BlindCardComponent

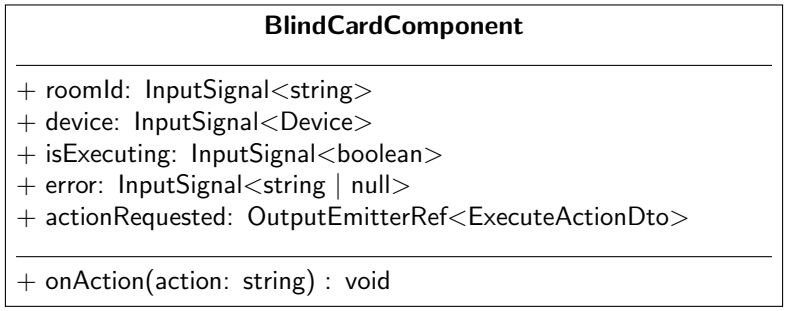

Figura 512: Diagramma della classe BlindCardComponent

Descrizione: Componente Angular che rappresenta una card per un dispositivo di tipo tapparella. Implementa IDeviceCard ed emette le azioni richieste dall'utente verso il componente padre.

Descrizione dei metodi della classe:

- `onAction(action: string) : void`: Emette l'evento `actionRequested` con i dati necessari all'esecuzione dell'azione richiesta.

4.2.8.3 DeviceCardComponent

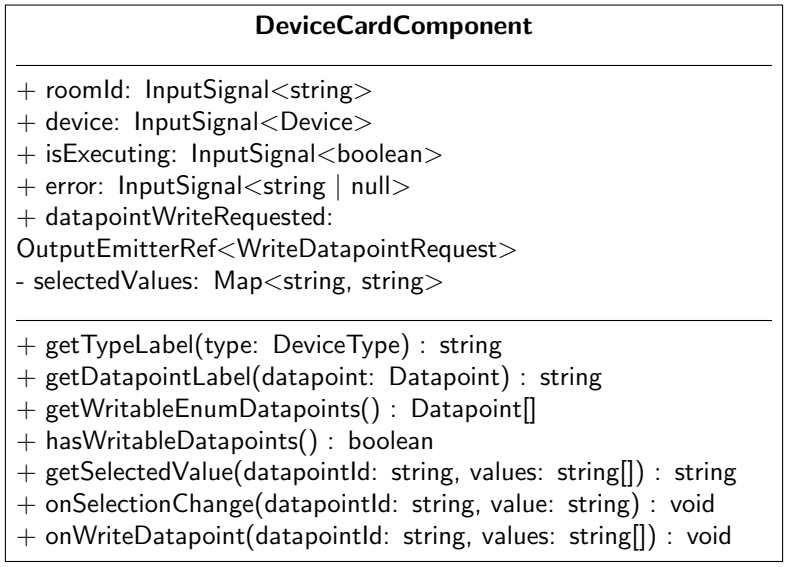

Figura 513: Diagramma della classe DeviceCardComponent

Descrizione: Componente Angular che rappresenta una card generica per un dispositivo, con supporto alla lettura e scrittura dei datapoint enumerati. Implementa IDeviceCard e gestisce internamente la selezione dei valori da inviare al componente padre tramite eventi.

Descrizione dei metodi della classe:

- `getTypeLabel(type: DeviceType) : string`: Restituisce l'etichetta localizzata corrispondente al tipo di dispositivo specificato.
- `getDatapointLabel(datapoint: Datapoint) : string`: Restituisce l'etichetta localizzata corrispondente al tipo SFE del datapoint specificato.
- `getWritableEnumDatapoints() : Datapoint[]`: Restituisce i datapoint del dispositivo che sono scrivibili e hanno valori enumerati disponibili.
- `hasWritableDatapoints() : boolean`: Verifica se il dispositivo possiede almeno un datapoint scrivibile con valori enumerati.
- `getSelectedValue(datapointId: string, values: string[]) : string`: Restituisce il valore selezionato per il datapoint specificato, inizializzandolo al primo valore disponibile se non ancora impostato.
- `onSelectionChange(datapointId: string, value: string) : void`: Aggiorna il valore selezionato per il datapoint specificato nella mappa interna.
- `onWriteDatapoint(datapointId: string, values: string[]) : void`: Emette l'evento datapointWriteRequested con il valore selezionato per il datapoint, se non è già in corso un'esecuzione.

4.2.8.4 EndpointTableComponent

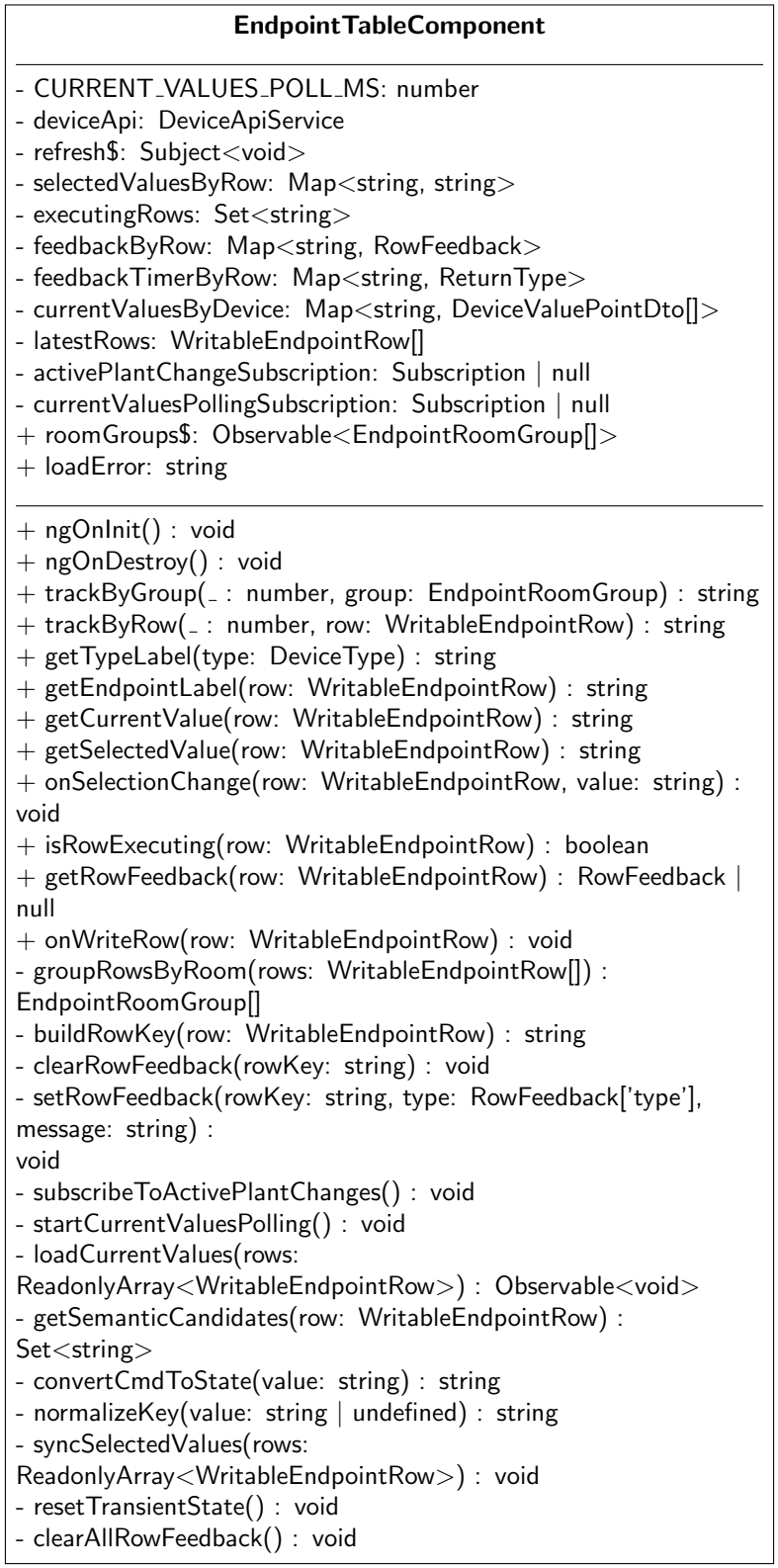

Figura 514: Diagramma della classe EndpointTableComponent

Descrizione: Componente Angular che visualizza una tabella di endpoint scrivibili raggruppati per stanza, con supporto alla lettura dei valori correnti tramite polling e all'invio di comandi con feedback visivo. Gestisce il refresh automatico al cambio di impianto attivo e il cleanup delle sottoscrizioni al momento della distruzione.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza la sottoscrizione agli eventi di cambio impianto e avvia il polling periodico dei valori correnti.
- `ngOnDestroy() : void`: Cancella le sottoscrizioni attive e i timer di feedback per prevenire memory leak.
- `trackByGroup(_ : number, group: EndpointRoomGroup) : string`: Restituisce l'identificativo della stanza per il tracking delle righe del gruppo nel template.
- `trackByRow(_ : number, row: WritableEndpointRow) : string`: Restituisce la chiave univoca della riga per il tracking nel template.
- `getTypeLabel(type: DeviceType) : string`: Restituisce l'etichetta localizzata corrispondente al tipo di dispositivo specificato.
- `getEndpointLabel(row: WritableEndpointRow) : string`: Restituisce l'etichetta localizzata corrispondente al tipo SFE del datapoint della riga.
- `getCurrentValue(row: WritableEndpointRow) : string`: Restituisce il valore corrente del datapoint della riga, cercando prima per id esatto, poi per corrispondenza semantica e infine per primo valore disponibile.
- `getSelectedValue(row: WritableEndpointRow) : string`: Restituisce il valore selezionato per la riga specificata, usando il primo valore enumerato come fallback.
- `onSelectionChange(row: WritableEndpointRow, value: string) : void`: Aggiorna il valore selezionato per la riga specificata nella mappa interna.
- `isRowExecuting(row: WritableEndpointRow) : boolean`: Verifica se è in corso l'invio di un comando per la riga specificata.
- `getRowFeedback(row: WritableEndpointRow) : RowFeedback | null`: Restituisce il feedback di esito operazione per la riga specificata, o `null` se non presente.
- `onWriteRow(row: WritableEndpointRow) : void`: Invia il valore selezionato per la riga al backend, aggiornando il feedback e innescando il refresh in caso di successo.
- `groupRowsByRoom(rows: WritableEndpointRow[]) : EndpointRoomGroup[]`: Raggruppa le righe endpoint per stanza, restituendo una lista di gruppi ordinati per ordine di inserimento.
- `buildRowKey(row: WritableEndpointRow) : string`: Costruisce una chiave univoca per una riga combinando gli identificativi di stanza, dispositivo e datapoint.
- `clearRowFeedback(rowKey: string) : void`: Rimuove il feedback e annulla il timer di autocancellazione per la riga specificata.
- `setRowFeedback(rowKey: string, type: RowFeedback['type'], message: string) : void`: Imposta il feedback per la riga specificata e pianifica la sua rimozione automatica dopo 5 secondi.
- `subscribeToActivePlantChanges() : void`: Sottoscrive l'evento di cambio impianto attivo per resettare lo stato e ricaricare le righe.

- `startCurrentValuesPolling() : void`: Avvia il polling periodico dei valori correnti dei dispositivi all'intervallo definito dalla costante statica.
- `loadCurrentValues(rows: ReadonlyArray<WritableEndpointRow>) : Observable<void>`: Recupera i valori correnti per tutti i dispositivi presenti nelle righe fornite e aggiorna la mappa interna.
- `getSemanticCandidates(row: WritableEndpointRow) : Set<string>`: Restituisce un insieme di chiavi normalizzate da usare per la ricerca semantica del valore corrente di un datapoint.
- `convertCmdToState(value: string) : string`: Converte una chiave di tipo comando nella corrispondente chiave di tipo stato sostituendo il suffisso `cmd` con `state`.
- `normalizeKey(value: string | undefined) : string`: Normalizza una stringa rimuovendo caratteri speciali e convertendo in minuscolo per confronti semantici.
- `syncSelectedValues(rows: ReadonlyArray<WritableEndpointRow>) : void`: Sincronizza la mappa dei valori selezionati con le righe correnti, aggiungendo nuove righe e rimuovendo quelle non più presenti.
- `resetTransientState() : void`: Azzera tutto lo stato transitorio del componente, inclusi valori selezionati, esecuzioni in corso, feedback e valori correnti.
- `clearAllRowFeedback() : void`: Cancella tutti i timer di feedback attivi e svuota le mappe di feedback.

4.2.8.5 EntranceDoorCardComponent

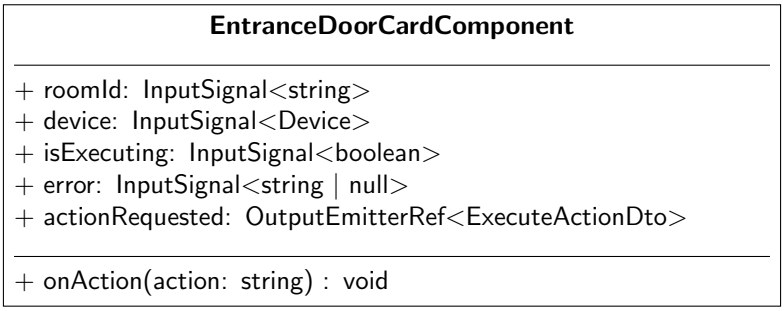

Figura 515: Diagramma della classe EntranceDoorCardComponent

Descrizione: Componente Angular che rappresenta una card per un dispositivo di tipo porta d'ingresso. Implementa IDeviceCard ed emette le azioni richieste dall'utente verso il componente padre.

Descrizione dei metodi della classe:

- `onAction(action: string) : void`: Emette l'evento actionRequested con i dati necessari all'esecuzione dell'azione richiesta.

4.2.8.6 FallSensorCardComponent

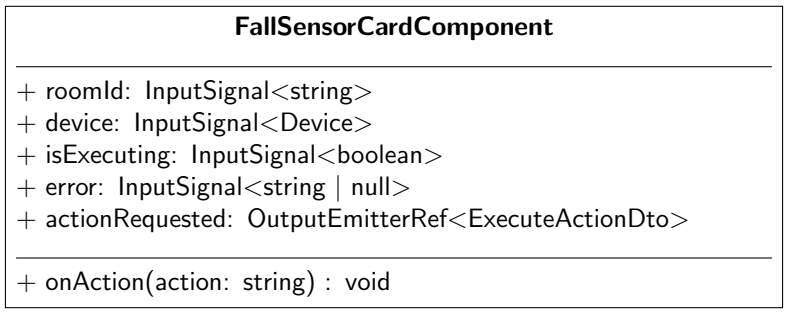

Figura 516: Diagramma della classe FallSensorCardComponent

Descrizione: Componente Angular che rappresenta una card per un dispositivo di tipo sensore di caduta. Implementa IDeviceCard ed emette le azioni richieste dall'utente verso il componente padre.

Descrizione dei metodi della classe:

- `onAction(action: string) : void`: Emette l'evento actionRequested con i dati necessari all'esecuzione dell'azione richiesta.

4.2.8.7 LightCardComponent

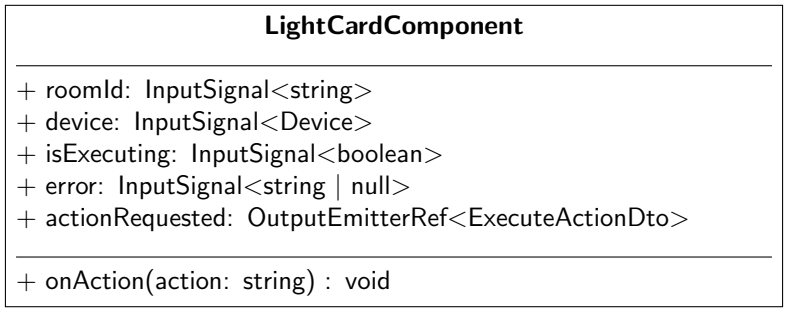

Figura 517: Diagramma della classe LightCardComponent

Descrizione: Componente Angular che rappresenta una card per un dispositivo di tipo luce. Implementa IDeviceCard ed emette le azioni richieste dall'utente verso il componente padre.

Descrizione dei metodi della classe:

- `onAction(action: string) : void`: Emette l'evento actionRequested con i dati necessari all'esecuzione dell'azione richiesta.

4.2.8.8 PresenceSensorCardComponent

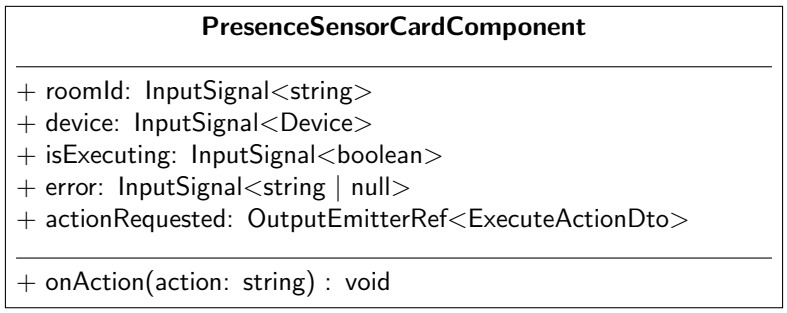

Figura 518: Diagramma della classe PresenceSensorCardComponent

Descrizione: Componente Angular che rappresenta una card per un dispositivo di tipo sensore di presenza. Implementa IDeviceCard ed emette le azioni richieste dall'utente verso il componente padre.

Descrizione dei metodi della classe:

- `onAction(action: string) : void`: Emette l'evento actionRequested con i dati necessari all'esecuzione dell'azione richiesta.

4.2.8.9 ThermostatCardComponent

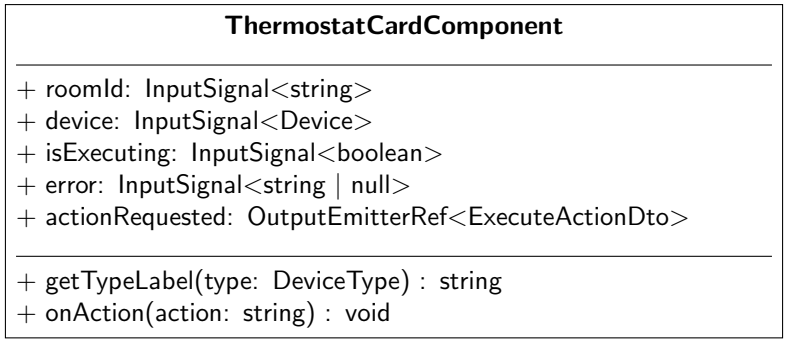

Figura 519: Diagramma della classe ThermostatCardComponent

Descrizione: Componente Angular che rappresenta una card per un dispositivo di tipo termostato. Implementa IDeviceCard ed emette le azioni richieste dall'utente verso il componente padre.

Descrizione dei metodi della classe:

- `getTypeLabel(type: DeviceType) : string`: Restituisce l'etichetta localizzata corrispondente al tipo di dispositivo specificato.
- `onAction(action: string) : void`: Emette l'evento actionRequested con i dati necessari all'esecuzione dell'azione richiesta.

4.2.8.10 IDeviceCard

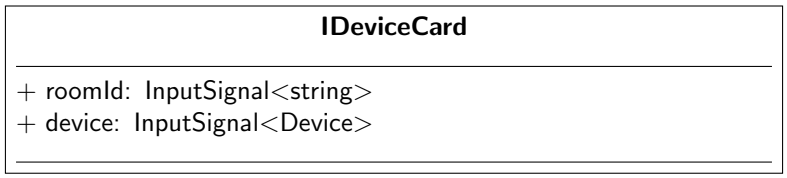

Figura 520: Diagramma della classe IDeviceCard

Descrizione: Interfaccia che definisce il contratto minimo per i componenti card di dispositivo. Garantisce la presenza dei segnali di input per l'identificativo della stanza e i dati del dispositivo.

4.2.8.11 EndpointRoomGroup

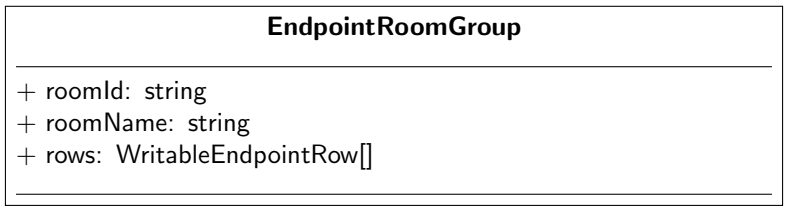

Figura 521: Diagramma della classe EndpointRoomGroup

Descrizione: Interfaccia che rappresenta un gruppo di righe endpoint appartenenti alla stessa stanza. Aggrega l'identificativo, il nome della stanza e la lista delle righe endpoint scrivibili associate.

4.2.8.12 ExecuteActionDto

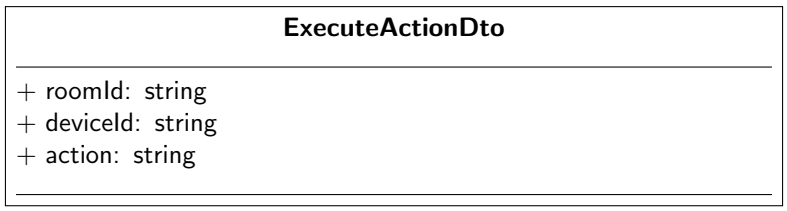

Figura 522: Diagramma della classe ExecuteActionDto

Descrizione: Interfaccia che rappresenta il DTO per l'esecuzione di un'azione su un dispositivo. Raccoglie gli identificativi della stanza e del dispositivo insieme al nome dell'azione da eseguire.

4.2.8.13 RowFeedback

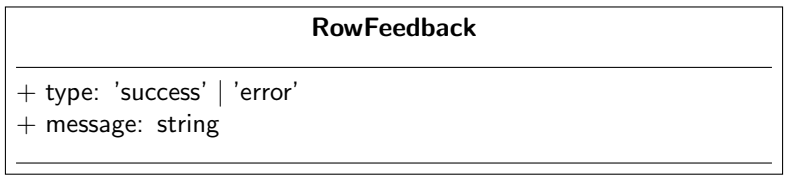

Figura 523: Diagramma della classe RowFeedback

Descrizione: Interfaccia che rappresenta il feedback visivo associato a una riga della tabella endpoint. Indica il tipo di esito dell'operazione e il messaggio da visualizzare all'utente.

4.2.8.14 WritableEndpointRow

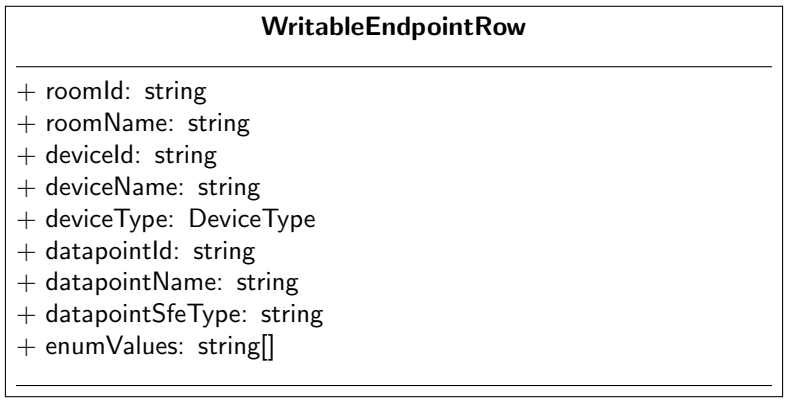

Figura 524: Diagramma della classe WritableEndpointRow

Descrizione: Interfaccia che rappresenta una riga della tabella degli endpoint scrivibili, aggregando le informazioni di stanza, dispositivo e datapoint. Include i valori enumerati disponibili per la scrittura del datapoint associato.

4.2.8.15 WriteDatapointRequest

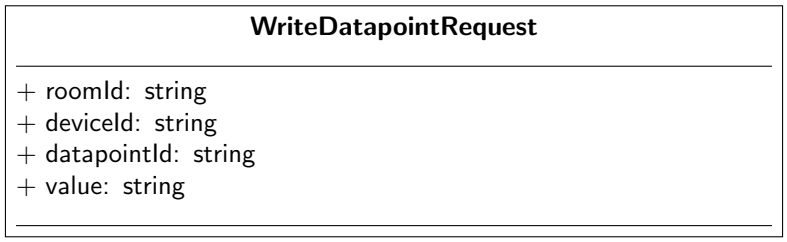

Figura 525: Diagramma della classe WriteDatapointRequest

Descrizione: Interfaccia che rappresenta la richiesta di scrittura di un valore su un datapoint, includendo gli identificativi di stanza, dispositivo e datapoint.

4.2.8.16 WriteDatapointDto

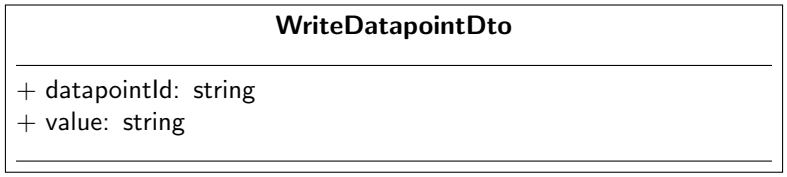

Figura 526: Diagramma della classe WriteDatapointDto

Descrizione: Interfaccia che rappresenta il DTO minimo per la scrittura di un valore su un datapoint, contenente solo l'identificativo del datapoint e il valore da scrivere.

4.2.8.17 DeviceValuePointDto

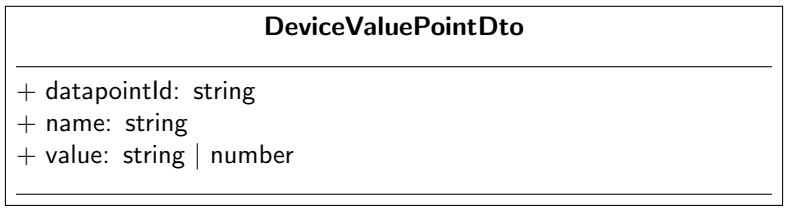

Figura 527: Diagramma della classe DeviceValuePointDto

Descrizione: Interfaccia che rappresenta il valore corrente di un singolo datapoint di un dispositivo, con identificativo, nome e valore.

4.2.8.18 DeviceValueDto

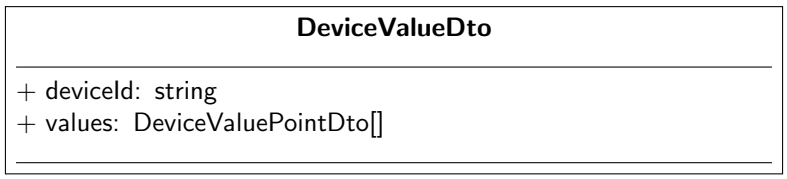

Figura 528: Diagramma della classe DeviceValueDto

Descrizione: Interfaccia che rappresenta la risposta del backend contenente i valori correnti di tutti i datapoint di un dispositivo.

4.2.8.19 DeviceApiService

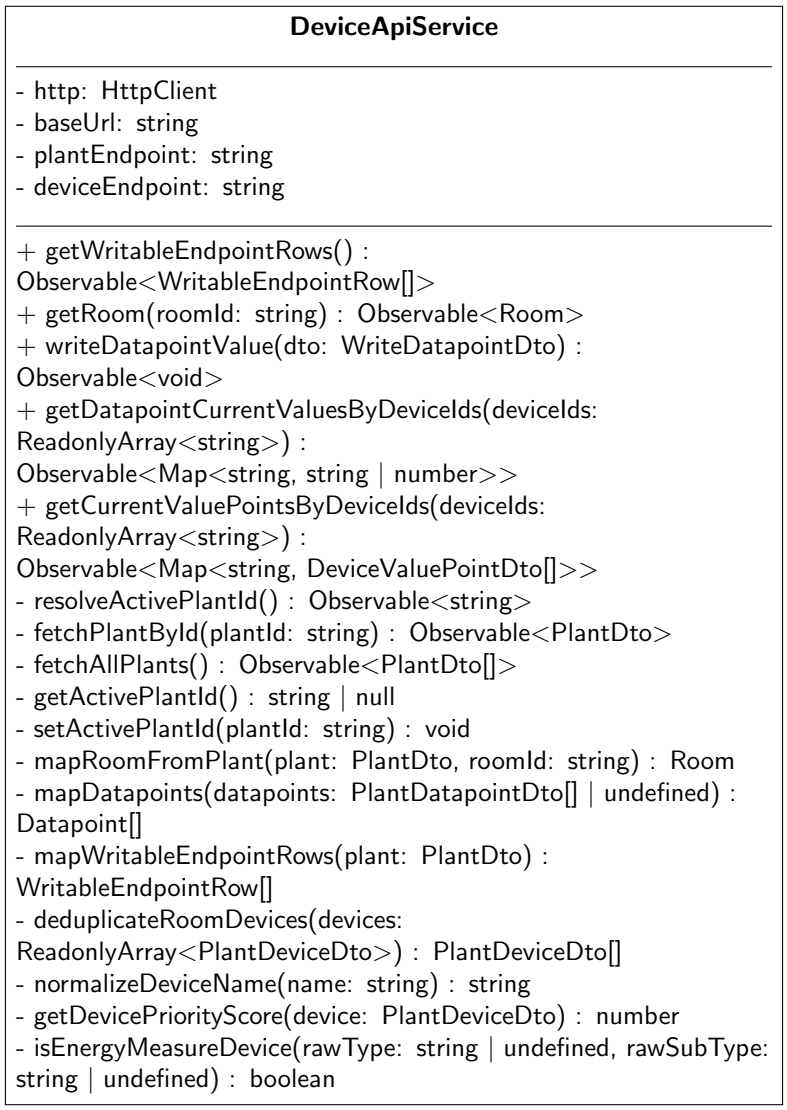

Figura 529: Diagramma della classe DeviceApiService

Descrizione: Servizio Angular per la comunicazione con le API REST relative ai dispositivi e agli impianti. Espone operazioni per la lettura e scrittura dei datapoint, il recupero delle righe endpoint scrivibili e la gestione dell'impianto attivo.

Descrizione dei metodi della classe:

- `getWritableEndpointRows() : Observable<WritableEndpointRow[]>`: Recupera le righe endpoint scrivibili dell'impianto attivo, mappandole a partire dai dati dell'impianto corrente.
- `getRoom(roomId: string) : Observable<Room>`: Recupera i dati di una stanza specifica a partire dall'impianto attivo, mappando dispositivi e datapoint.
- `writeDatapointValue(dto: WriteDatapointDto) : Observable<void>`: Invia una richiesta POST per scrivere un valore su un datapoint, verificando l'esito tramite codice di stato e messaggio di risposta.
- `getDatapointCurrentValuesByDeviceIds(deviceIds: ReadonlyArray<string>) : Observable<Map<string, string | number>>`: Recupera i valori correnti dei datapoint per i dispositivi specificati, restituendo una mappa indicizzata per identificativo di datapoint.

- `getCurrentValuePointsByDeviceIds(deviceIds: ReadonlyArray<string>) : Observable<Map<string, DeviceValuePointDto[]>>`: Recupera i valori correnti di tutti i datapoint per i dispositivi specificati, restituendo una mappa indicizzata per identificativo di dispositivo.
- `resolveActivePlantId() : Observable<string>`: Restituisce l'identificativo dell'impianto attivo dal localStorage, recuperandolo dal backend se non disponibile.
- `fetchPlantById(plantId: string) : Observable<PlantDto>`: Recupera i dati dell'impianto con l'identificativo specificato tramite una richiesta GET al backend.
- `fetchAllPlants() : Observable<PlantDto[]>`: Recupera tutti gli impianti disponibili dal backend, restituendo un array vuoto in caso di errore.
- `getActivePlantId() : string | null`: Legge l'identificativo dell'impianto attivo dal localStorage, restituendo null se non disponibile o vuoto.
- `setActivePlantId(plantId: string) : void`: Salva l'identificativo dell'impianto attivo nel localStorage.
- `mapRoomFromPlant(plant: PlantDto, roomId: string) : Room`: Estrae e mappa i dati di una stanza specifica da un PlantDto, includendo dispositivi e datapoint.
- `mapDatapoints(datapoints: PlantDatapointDto[] | undefined) : Datapoint[]`: Trasforma un array di PlantDatapointDto nel corrispondente array di Datapoint del modello di dominio.
- `mapWritableEndpointRows(plant: PlantDto) : WritableEndpointRow[]`: Trasforma i dati di un impianto in una lista di righe endpoint scrivibili, filtrando i dispositivi di misura energetica e deduplicando per nome.
- `deduplicateRoomDevices(devices: ReadonlyArray<PlantDeviceDto>) : PlantDeviceDto[]`: Deduplica i dispositivi di una stanza per nome normalizzato, mantenendo quello con il punteggio di priorità più alto.
- `normalizeDeviceName(name: string) : string`: Normalizza il nome di un dispositivo rimuovendo spazi e convertendo in minuscolo.
- `getDevicePriorityScore(device: PlantDeviceDto) : number`: Calcola un punteggio di priorità per un dispositivo in base al tipo, sottotipo e numero di datapoint.
- `isEnergyMeasureDevice(rawType: string | undefined, rawSubType: string | undefined) : boolean`: Verifica se un dispositivo è di tipo misura energetica in base al tipo e sottotipo forniti.

4.2.9 MainLayout

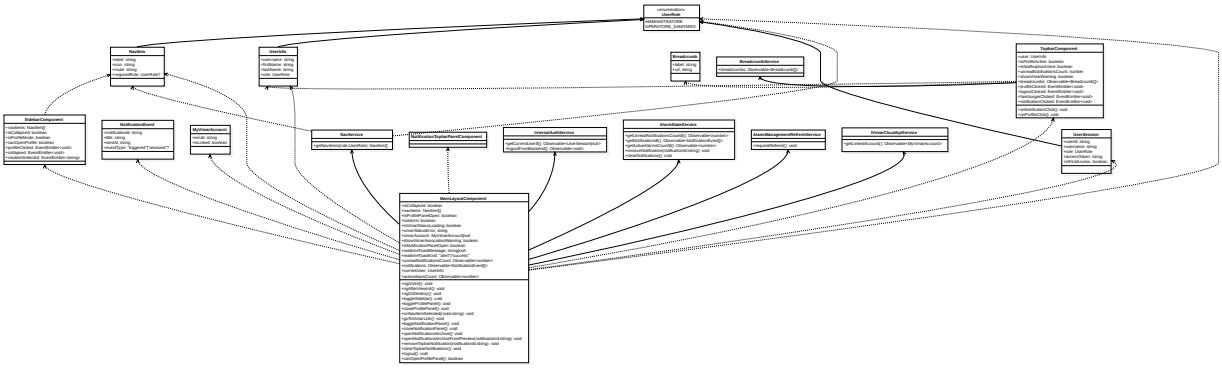

Figura 530: Diagramma delle classi del modulo MainLayout

4.2.9.1 MainLayoutComponent

MainLayoutComponent

+ topbarRef: ElementRef
+ isCollapsed: boolean
+ navItems: NavItem[]
+ isProfilePanelOpen: boolean
+ isAdmin: boolean
+ isVimarStatusLoading: boolean
+ vimarStatusError: string
+ vimarAccount: MyVimarAccount | null
+ showVimarAssociationWarning: boolean
+ isNotificationPanelOpen: boolean
+ realtimeToastMessage: string | null
+ realtimeToastKind: 'alert' | 'success'
- navService: NavService
- internalAuthService: InternalAuthService
- alarmStateService: AlarmStateService
- alarmManagementRefreshService: AlarmManagementRefreshService
- router: Router
- myVimarService: IVimarCloudApiService | null
+ unreadNotificationsCount$: Observable<number>
+ notifications$: Observable<NotificationEvent[]>
- cdr: ChangeDetectorRef
- destroy$: Subject<void>
- toastTimer: ReturnType<typeof setTimeout> | null
- latestRealtimeNotificationId: string | null
+ currentUser: UserInfo
+ activeAlarmCount$: Observable<number>
- resizeObserver: ResizeObserver

+ ngOnInit() : void
+ ngAfterViewInit() : void
+ ngOnDestroy() : void
+ toggleSidebar() : void
+ toggleProfilePanel() : void
+ closeProfilePanel() : void
+ onNavItemSelected(route: string) : void
+ goToVimarLink() : void
+ toggleNotificationPanel() : void
+ closeNotificationPanel() : void
+ openNotificationsArchive() : void
+ openNotificationsArchiveFromPreview(notificationId: string) : void
+ removeTopbarNotification(notificationId: string) : void
+ clearTopbarNotifications() : void
+ logout() : void
+ canOpenProfilePanel() : boolean
- bindRealtimeAlarmNotifications() : void
- openRealtimeNotificationPanel() : void
- showRealtimeToast(message: string, kind: 'alert' | 'success') : void
- clearToastTimer() : void
- loadVimarStatus() : void

Figura 531: Diagramma della classe MainLayoutComponent

Descrizione: Il componente MainLayoutComponent gestisce il layout principale dell'applicazione, includendo sidebar, topbar e pannelli per profilo e notifiche. Coordina servizi per autenticazione, allarmi e navigazione, supportando funzionalità specifiche per amministratori come l'integrazione Vimar.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il componente, legando le notifiche e caricando i dati utente.
- `ngAfterViewInit() : void`: Dopo l'inizializzazione della vista, osserva le dimensioni della topbar.
- `ngOnDestroy() : void`: Distrugge il componente, completando i subject e disconnettendo l'osservatore.
- `toggleSidebar() : void`: Alterna lo stato di collasso della sidebar.
- `toggleProfilePanel() : void`: Alterna l'apertura del pannello del profilo e carica lo stato Vimar se admin.
- `closeProfilePanel() : void`: Chiude il pannello del profilo.
- `onNavItemSelected(route: string) : void`: Gestisce la selezione di un elemento di navigazione, richiudendo pannelli e gestendo refresh.
- `goToVimarLink() : void`: Naviga alla pagina di collegamento Vimar.
- `toggleNotificationPanel() : void`: Alterna l'apertura del pannello delle notifiche.
- `closeNotificationPanel() : void`: Chiude il pannello delle notifiche.
- `openNotificationsArchive() : void`: Naviga all'archivio delle notifiche.
- `openNotificationsArchiveFromPreview(notificationId: string) : void`: Naviga all'archivio notifiche focalizzandosi su una specifica notifica.
- `removeTopbarNotification(notificationId: string) : void`: Rimuove una notifica dalla topbar tramite ID.
- `clearTopbarNotifications() : void`: Cancella tutte le notifiche dalla topbar.
- `logout() : void`: Effettua il logout, navigando alla pagina di login.
- `canOpenProfilePanel() : boolean`: Verifica se il pannello del profilo può essere aperto.
- `bindRealtimeAlarmNotifications() : void`: Collega le notifiche di allarme in tempo reale.
- `openRealtimeNotificationPanel() : void`: Apre il pannello delle notifiche in tempo reale.
- `showRealtimeToast(message: string, kind: 'alert' | 'success') : void`: Mostra un toast in tempo reale con messaggio e tipo.
- `clearToastTimer() : void`: Cancella il timer del toast in tempo reale.
- `loadVimarStatus() : void`: Carica lo stato del collegamento Vimar, gestendo errori e caricamenti.

4.2.9.2 SidebarComponent

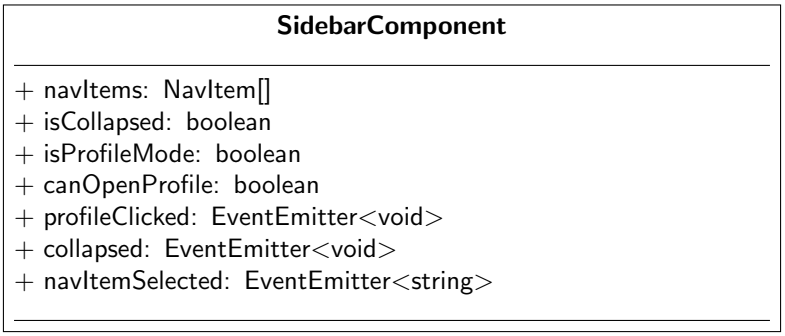

Figura 532: Diagramma della classe SidebarComponent

Descrizione: Il componente SidebarComponent rappresenta il componente presentazionale della sidebar, visualizzando gli elementi di navigazione e gestendo le interazioni utente. Comunica con il componente padre tramite Input per i dati e Output per emettere i principali eventi di interazione.

4.2.9.3 TopbarComponent

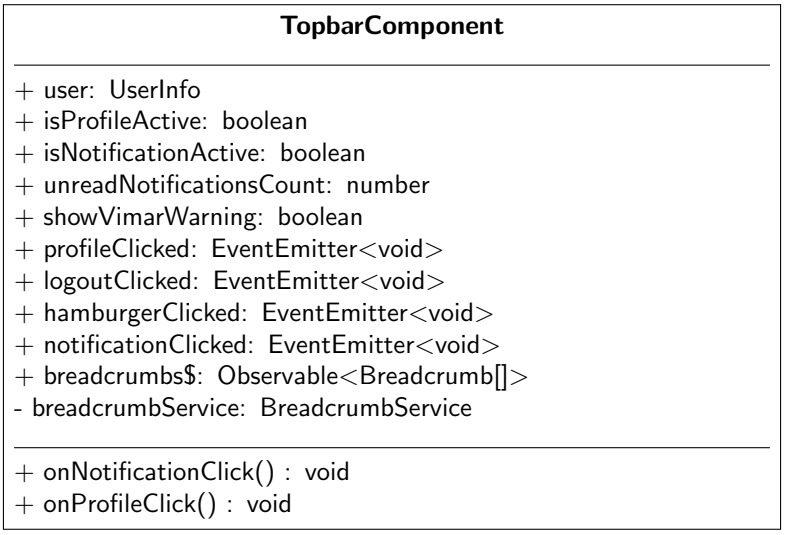

Figura 533: Diagramma della classe TopbarComponent

Descrizione: Il componente TopbarComponent rappresenta la barra superiore dell'applicazione, visualizzando informazioni utente, conteggio notifiche, breadcrumb e icone di interazione. Comunica con il componente padre tramite Input per i dati e Output per emettere gli eventi di interazione dell'utente.

Descrizione dei metodi della classe:

- `onNotificationClick() : void`: Emette un evento quando l'utente clicca sull'icona delle notifiche.
- `onProfileClick() : void`: Emette un evento quando l'utente clicca sull'icona del profilo.

4.2.9.4 NavService

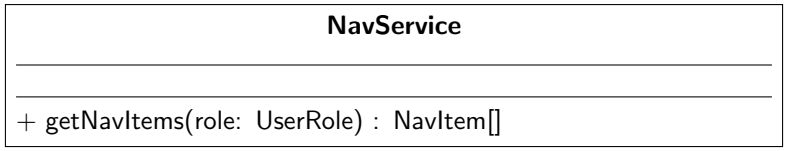

Figura 534: Diagramma della classe NavService

Descrizione: Il servizio NavService fornisce gli elementi di navigazione disponibili per l'utente, filtrando le opzioni in base al ruolo assegnato. Gestisce le autorizzazioni di accesso e le rotte associate a ciascun elemento di navigazione.

Descrizione dei metodi della classe:

- `getNavItems(role: UserRole) : NavItem[]`: Recupera la lista di elementi di navigazione filtrata in base al ruolo utente fornito, escludendo le opzioni riservate agli amministratori.

4.2.10 My-vimar-integration

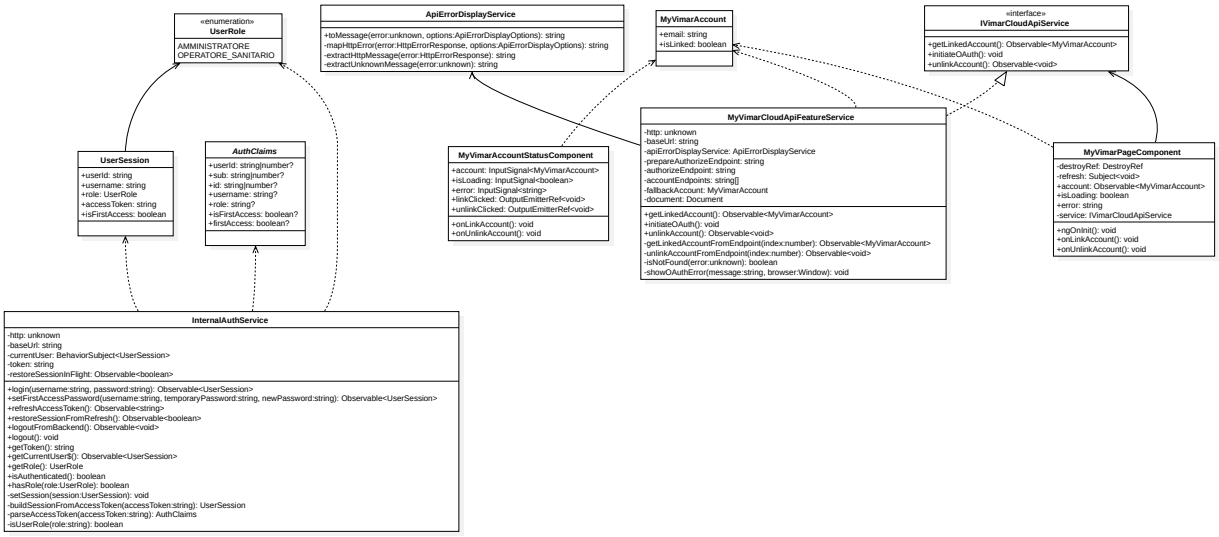

Figura 535: Diagramma delle classi del modulo MyVimar integration

4.2.10.1 MyVimarAccountStatusComponent

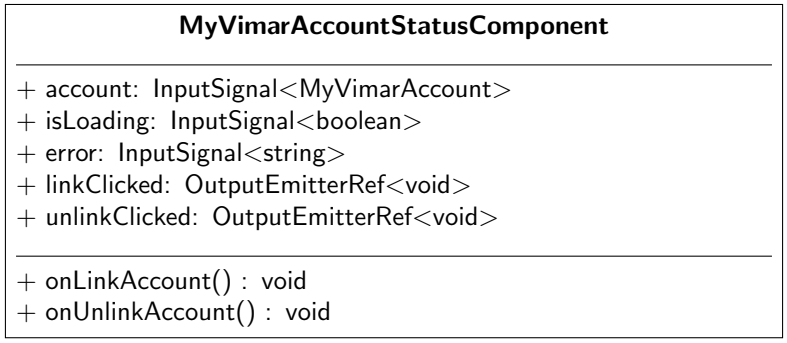

Figura 536: Diagramma della classe MyVimarAccountStatusComponent

Descrizione: Il componente MyVimarAccountStatusComponent visualizza lo stato dell'account MyVimar, gestendo input per dati e stati di caricamento. Fornisce output per eventi di collegamento e scollegamento, facilitando l'interazione utente con l'integrazione MyVimar.

Descrizione dei metodi della classe:

- `onLinkAccount() : void`: Emette l'evento linkClicked per avviare il collegamento dell'account MyVimar.
- `onUnlinkAccount() : void`: Emette l'evento unlinkClicked per avviare lo scollegamento dell'account MyVimar.

4.2.10.2 MyVimarPageComponent

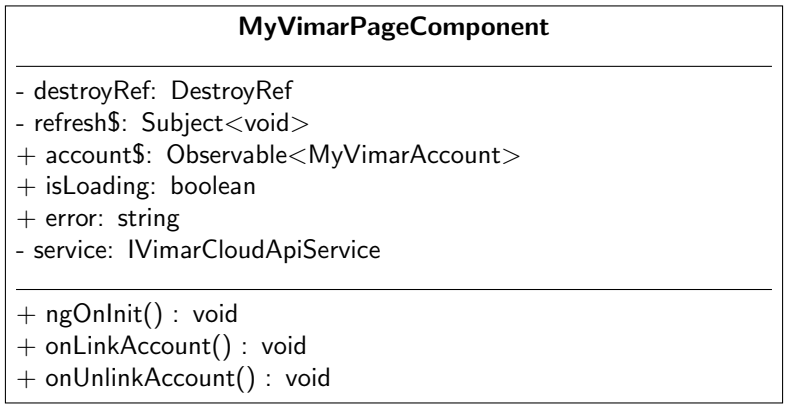

Figura 537: Diagramma della classe MyVimarPageComponent

Descrizione: Il componente MyVimarPageComponent gestisce la pagina di integrazione MyVimar, visualizzando lo stato dell'account e permettendo collegamento e scollegamento. Utilizza observable per gestire aggiornamenti reattivi e gestisce errori di comunicazione con l'API.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza l'observable per il recupero dello stato dell'account MyVimar con gestione errori.
- `onLinkAccount() : void`: Avvia il processo di collegamento dell'account MyVimar tramite OAuth.
- `onUnlinkAccount() : void`: Rimuove il collegamento dell'account MyVimar e aggiorna lo stato con gestione errori.

4.2.10.3 MyVimarAccount

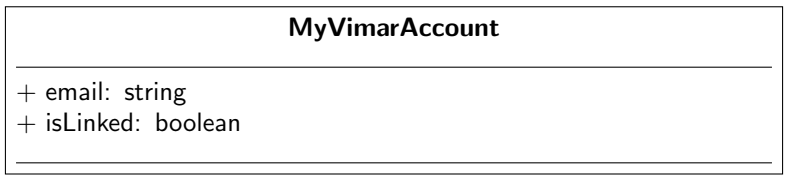

Figura 538: Diagramma della classe MyVimarAccount

Descrizione: Interfaccia che rappresenta le informazioni dell'account MyVimar collegato. Espone l'indirizzo email e lo stato di collegamento dell'account.

4.2.10.4 MyVimarCloudApiFeatureService

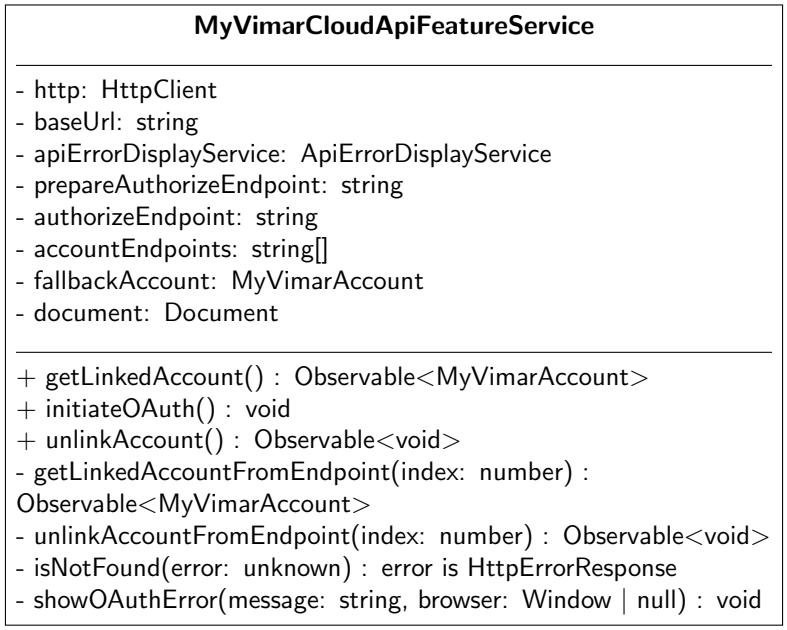

Figura 539: Diagramma della classe MyVimarCloudApiFeatureService

Descrizione: Servizio che implementa IVimarCloudApiService per la gestione dell'integrazione con il cloud MyVimar. Espone funzionalità per recuperare e scollegare l'account collegato e per avviare il flusso OAuth.

Descrizione dei metodi della classe:

- `getLinkedAccount() : Observable<MyVimarAccount>`: Restituisce le informazioni sull'account MyVimar collegato, interrogando gli endpoint disponibili in ordine.
- `initiateOAuth() : void`: Avvia il flusso OAuth preparando il ticket e reindirizzando il browser all'endpoint di autorizzazione.
- `unlinkAccount() : Observable<void>`: Rimuove il collegamento con l'account MyVimar tramite una richiesta DELETE agli endpoint disponibili.
- `getLinkedAccountFromEndpoint(index: number) : Observable<MyVimarAccount>`: Tenta di recuperare l'account collegato dall'endpoint all'indice specificato, passando al successivo in caso di risposta 404.
- `unlinkAccountFromEndpoint(index: number) : Observable<void>`: Tenta di scollegare l'account tramite l'endpoint all'indice specificato, passando al successivo in caso di risposta 404.
- `isNotFound(error: unknown) : error is HttpErrorResponse`: Verifica se l'errore ricevuto è una risposta HTTP con stato 404.
- `showOAuthError(message: string, browser: Window | null) : void`: Mostra un messaggio di errore OAuth tramite un alert del browser.

4.2.11 Notifications

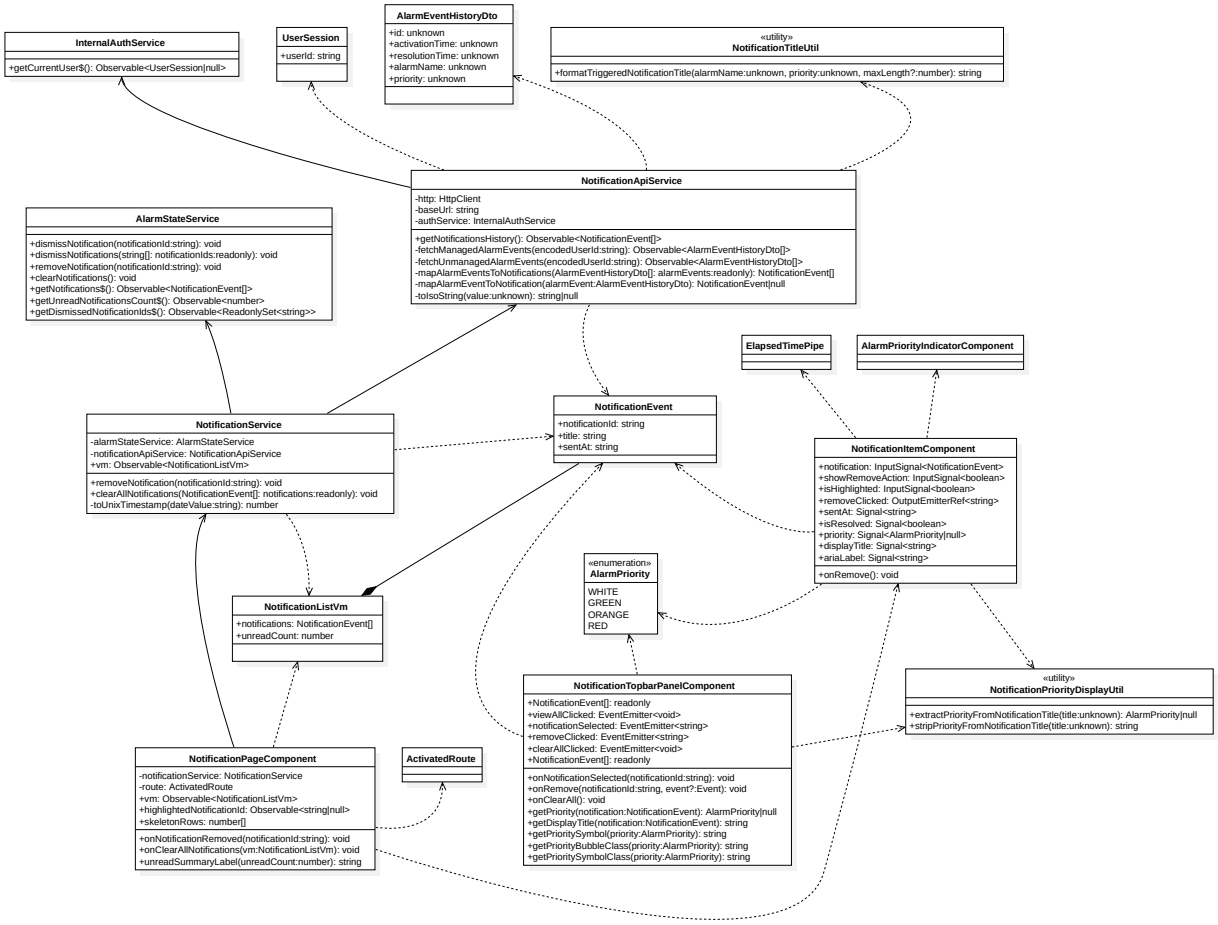

Figura 540: Diagramma delle classi del modulo Notifications

4.2.11.1 NotificationBadgeComponent

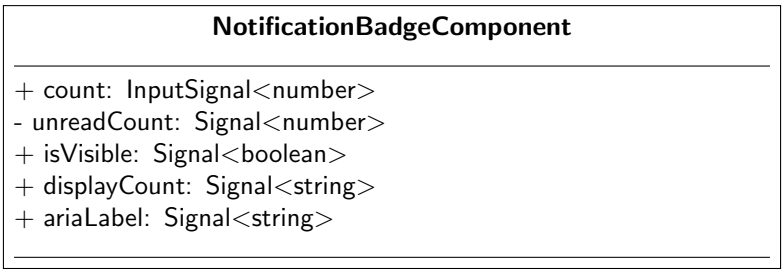

Figura 541: Diagramma della classe NotificationBadgeComponent

Descrizione: Componente Angular che visualizza un badge con il conteggio delle notifiche non lette. Limita il valore visualizzato a 99+ e si nasconde automaticamente quando il conteggio è zero.

4.2.11.2 NotificationItemComponent

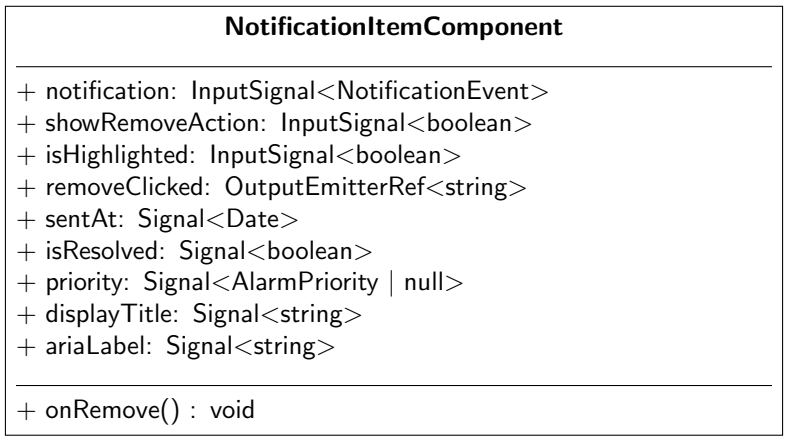

Figura 542: Diagramma della classe NotificationItemComponent

Descrizione: Componente Angular che visualizza una singola notifica con priorità, titolo, data e azione di rimozione opzionale. Calcola dinamicamente lo stato di risoluzione e la priorità a partire dal titolo della notifica.

Descrizione dei metodi della classe:

- `onRemove() : void`: Emette l'evento removeClicked con l'identificativo della notifica corrente.

4.2.11.3 NotificationPageComponent

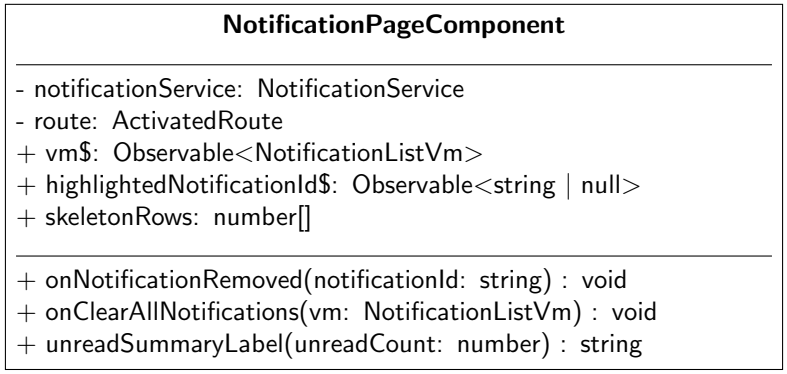

Figura 543: Diagramma della classe NotificationPageComponent

Descrizione: Componente Angular che rappresenta la pagina delle notifiche, con supporto alla visualizzazione, rimozione e cancellazione in blocco. Recupera l'identificativo della notifica da evidenziare tramite i parametri di query dell'URL.

Descrizione dei metodi della classe:

- `onNotificationRemoved(notificationId: string) : void`: Delega al servizio la rimozione della notifica con l'identificativo specificato.
- `onClearAllNotifications(vm: NotificationListVm) : void`: Delega al servizio la cancellazione di tutte le notifiche presenti nel view model.
- `unreadSummaryLabel(unreadCount: number) : string`: Restituisce l'etichetta testuale che riassume il numero di notifiche non lette, gestendo il caso singolare e plurale.

4.2.11.4 NotificationTopbarPanelComponent

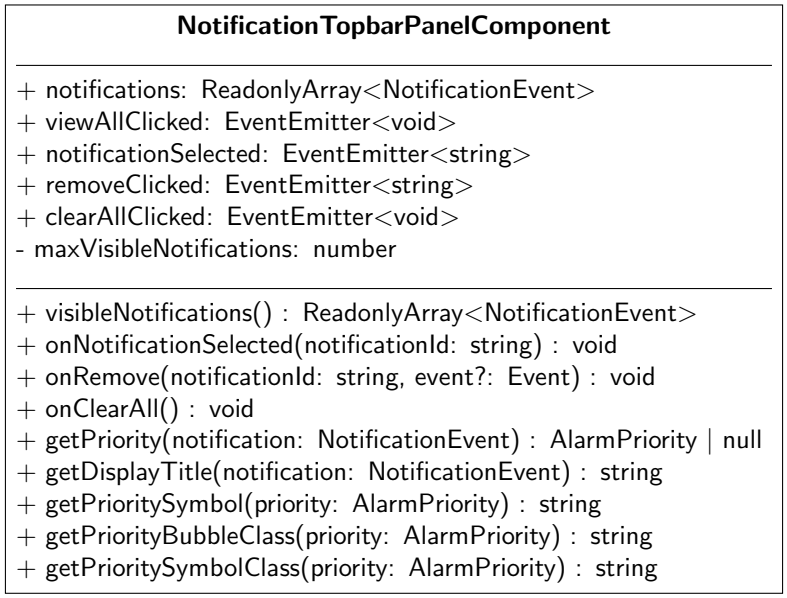

Figura 544: Diagramma della classe NotificationTopbarPanelComponent

Descrizione: Componente Angular che visualizza un pannello compatto delle notifiche nella topbar, limitando la lista alle prime notifiche visibili. Espone eventi per la selezione, rimozione e cancellazione delle notifiche e gestisce la presentazione visiva delle priorità.

Descrizione dei metodi della classe:

- `visibleNotifications() : ReadonlyArray<NotificationEvent>`: Restituisce le prime notifiche fino al limite massimo di visibilità configurato.
- `onNotificationSelected(notificationId: string) : void`: Emette l'evento notificationSelected con l'identificativo della notifica selezionata.
- `onRemove(notificationId: string, event?: Event) : void`: Emette l'evento removeClicked con l'identificativo della notifica, interrompendo la propagazione dell'evento DOM se fornito.
- `onClearAll() : void`: Emette l'evento clearAllClicked per richiedere la cancellazione di tutte le notifiche.
- `getPriority(notification: NotificationEvent) : AlarmPriority | null`: Estrae la priorità dalla notifica a partire dal titolo tramite la funzione di utilità dedicata.
- `getDisplayTitle(notification: NotificationEvent) : string`: Restituisce il titolo della notifica privo del prefisso di priorità, usando il titolo originale come fallback.
- `getPrioritySymbol(priority: AlarmPriority) : string`: Restituisce il simbolo testuale associato al livello di priorità specificato.
- `getPriorityBubbleClass(priority: AlarmPriority) : string`: Restituisce le classi CSS per il bubble di priorità in base al livello specificato.
- `getPrioritySymbolClass(priority: AlarmPriority) : string`: Restituisce le classi CSS per il simbolo di priorità in base al livello specificato.

4.2.11.5 NotificationEvent

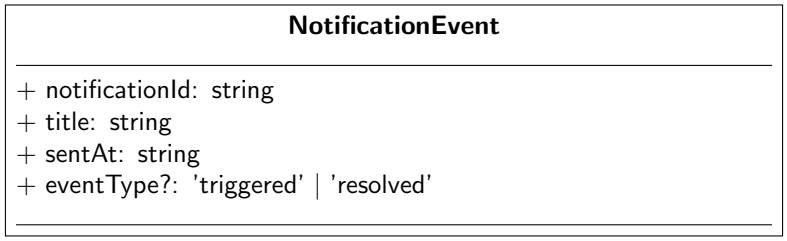

Figura 545: Diagramma della classe NotificationEvent

Descrizione: Interfaccia che rappresenta una singola notifica di sistema, condivisa tra lo storico HTTP e gli eventi push real-time. La simmetria tra le due sorgenti consente a NotificationService di unirle senza trasformazioni intermedie.

4.2.11.6 NotificationListVm

Figura 546: Diagramma della classe NotificationListVm

Descrizione: Interfaccia che rappresenta il view model della pagina delle notifiche, prodotto da NotificationService tramite combineLatest. Aggrega la lista unificata e deduplicata delle notifiche e il contatore delle non lette in un unico snapshot atomicamente coerente.

4.2.11.7 NotificationService

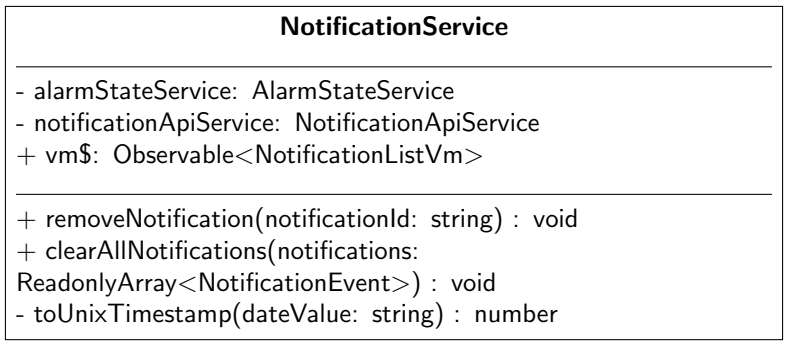

Figura 547: Diagramma della classe NotificationService

Descrizione: Servizio Angular con scope a livello di componente che compone le notifiche storiche HTTP e quelle in-session push in un unico view model reattivo. Gestisce la deduplicazione, il filtraggio delle notifiche dismesse e l'ordinamento per data decrescente.

Descrizione dei metodi della classe:

- `removeNotification(notificationId: string) : void`: Rimuove la notifica specificata dallo stato locale e la marca come letta tramite il servizio di stato degli allarmi.
- `clearAllNotifications(notifications: ReadonlyArray<NotificationEvent>) : void`: Marca come lette e rimuove tutte le notifiche fornite dallo stato locale.
- `toUnixTimestamp(dateValue: string) : number`: Converte una stringa data in timestamp Unix, restituendo zero se il valore non è parsabile.

4.2.11.8 NotificationApiService

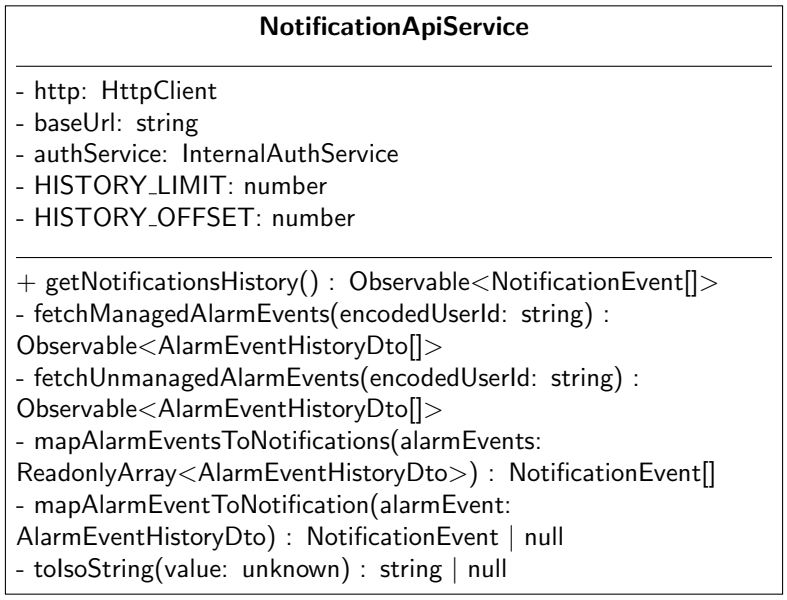

Figura 548: Diagramma della classe NotificationApiService

Descrizione: Questo servizio centralizza il recupero della cronologia delle notifiche tramite API REST, integrando i dati provenienti dal sistema di gestione allarmi. Si occupa della mappatura dei dati grezzi in modelli di notifica fruibili dai componenti UI.

Descrizione dei metodi della classe:

- `getNotificationsHistory() : Observable<NotificationEvent[]>`: Recupera la cronologia completa delle notifiche per l'utente corrente combinando gli eventi di allarme gestiti e non gestiti.
- `fetchManagedAlarmEvents(encodedUserId: string) : Observable<AlarmEventHistoryDto[]>`: Esegue una chiamata API per recuperare gli eventi di allarme che sono già stati presi in carico o risolti.
- `fetchUnmanagedAlarmEvents(encodedUserId: string) : Observable<AlarmEventHistoryDto[]>`: Effettua una richiesta HTTP per ottenere l'elenco degli eventi di allarme ancora attivi o non gestiti.
- `mapAlarmEventsToNotifications(alarmEvents: ReadonlyArray<AlarmEventHistoryDto>) : NotificationEvent[]`: Trasforma un insieme di oggetti DTO provenienti dal backend in una lista di eventi di notifica pronti per l'interfaccia.
- `mapAlarmEventToNotification(alarmEvent: AlarmEventHistoryDto) : NotificationEvent | null`: Converte un singolo evento di allarme nel formato standard di notifica, determinando il titolo e il timestamp corretti.
- `toIsoString(value: unknown) : string | null`: Normalizza diversi formati di data in una stringa conforme allo standard ISO per garantire la coerenza temporale.

4.2.12 Shared

4.2.12.1 AlarmActionButtonComponent

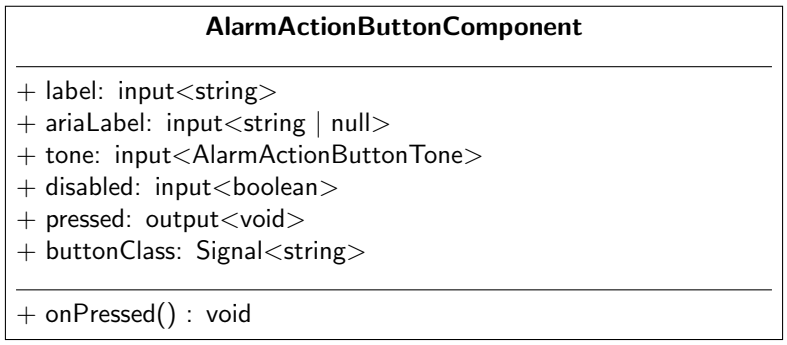

Figura 549: Diagramma della classe AlarmActionButtonComponent

Descrizione: Componente UI riutilizzabile che rappresenta un pulsante d'azione contestuale per la gestione degli allarmi. Utilizza le Signal API di Angular per una gestione reattiva delle proprietà e calcola dinamicamente le classi CSS in base al tono visivo (neutral, danger, primary) impostato.

Descrizione dei metodi della classe:

- `onPressed() : void`: Gestisce l'evento di interazione dell'utente emettendo il segnale di pressione solo se il componente non si trova in uno stato disabilitato.

4.2.12.2 AlarmPriorityIndicatorComponent

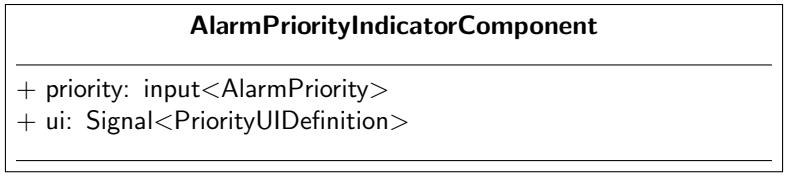

Figura 550: Diagramma della classe AlarmPriorityIndicatorComponent

Descrizione: Componente visivo preposto alla rappresentazione grafica della priorità di un allarme. Attraverso l'uso di segnali reattivi (Signals), il componente mappa il valore di priorità ricevuto in input alle corrispondenti definizioni stilistiche (colori, icone, etichette) per garantire una comunicazione visiva immediata e coerente.

4.2.12.3 AlarmTableShellComponent

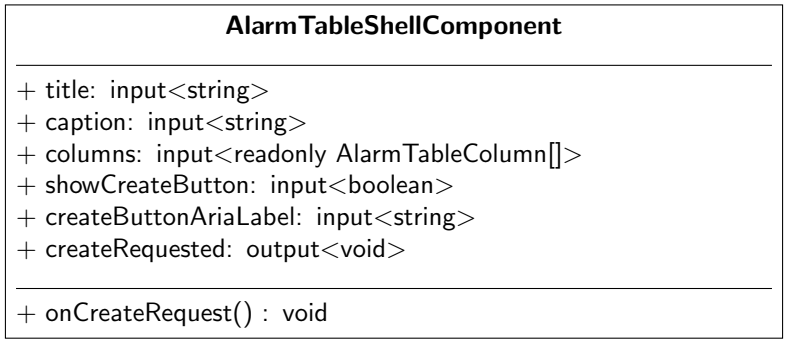

Figura 551: Diagramma della classe AlarmTableShellComponent

Descrizione: Componente strutturale (shell) progettato per avvolgere e fornire un contesto visivo alle tabelle di gestione allarmi. Si occupa della visualizzazione del titolo, della didascalia per l'accessibilità e della gestione opzionale del punto di ingresso per la creazione di nuovi elementi, promuovendo una disposizione coerente delle interfacce tabellari.

Descrizione dei metodi della classe:

- `onCreateRequest() : void`: Innesca l'emissione dell'evento di creazione verso il componente genitore quando l'utente interagisce con il pulsante di aggiunta.

4.2.12.4 AlarmToggleSwitchComponent

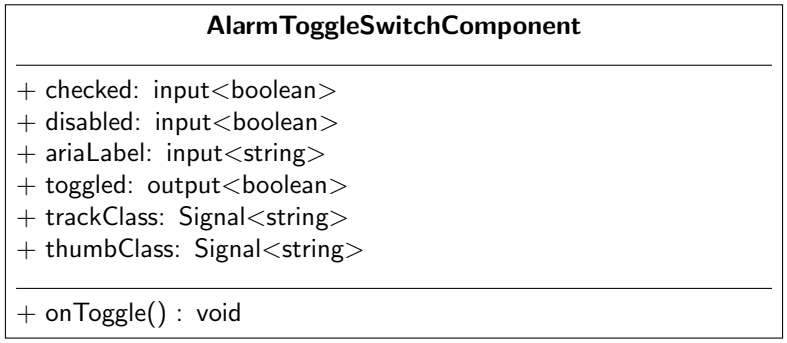

Figura 552: Diagramma della classe AlarmToggleSwitchComponent

Descrizione: Componente UI personalizzato che implementa uno switch a scorrimento (toggle) per l'attivazione o disattivazione di stati. Utilizza le Signal API per gestire in modo reattivo le classi CSS del tracciato e del cursore, garantendo feedback visivi immediati e il supporto per l'accessibilità tramite etichette ARIA.

Descrizione dei metodi della classe:

- `onToggle() : void`: Gestisce l'evento di commutazione dello switch, invertendo lo stato corrente ed emettendo il nuovo valore tramite l'output, a condizione che il componente non sia disabilitato.

4.2.12.5 ConfirmDialogComponent

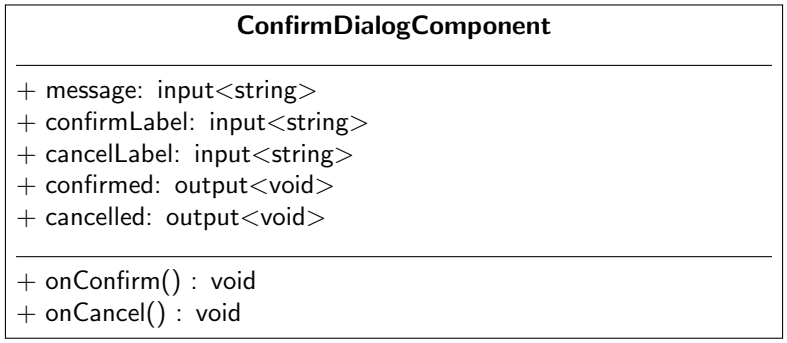

Figura 553: Diagramma della classe ConfirmDialogComponent

Descrizione: Componente per la visualizzazione di una finestra di dialogo modale di conferma. Offre un'interfaccia semplice per richiedere l'approvazione dell'utente prima di procedere con operazioni potenzialmente distruttive o significative, permettendo la personalizzazione del messaggio e delle etichette dei pulsanti.

Descrizione dei metodi della classe:

- `onConfirm() : void`: Innesca l'emissione dell'evento di conferma verso il componente genitore quando l'utente accetta l'operazione.
- `onCancel() : void`: Innesca l'emissione dell'evento di annullamento quando l'utente decide di chiudere il dialogo senza procedere.

4.2.12.6 ModalShellComponent

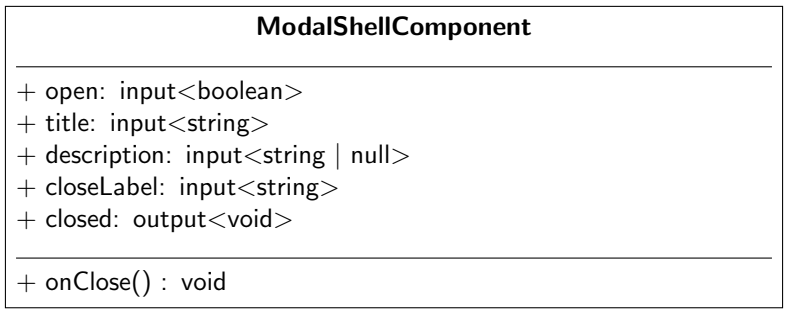

Figura 554: Diagramma della classe ModalShellComponent

Descrizione: Componente strutturale che funge da guscio per la creazione di finestre modali. Gestisce l'impalcatura visiva standard, inclusi il titolo, la descrizione opzionale e il pulsante di chiusura, permettendo di incapsulare contenuti dinamici in un contenitore coerente e accessibile.

Descrizione dei metodi della classe:

- `onClose() : void`: Emette il segnale di chiusura del componente per notificare al chiamante l'intenzione dell'utente di abbandonare la vista modale.

4.2.12.7 ElapsedTimePipe

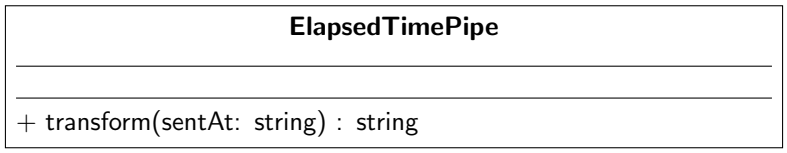

Figura 555: Diagramma della classe ElapsedTimePipe

Descrizione: Pipe personalizzata per la trasformazione di timestamp in indicatori di tempo relativo. Gestisce diverse unità di misura (secondi, minuti, ore, giorni) e include una logica di fallback per date future o non valide, facilitando la lettura dei tempi di ricezione degli eventi nell'interfaccia utente.

Descrizione dei metodi della classe:

- `transform(sentAt: string) : string`: Calcola la differenza temporale tra il timestamp fornito e l'istante attuale, restituendo una stringa formattata che descrive il tempo trascorso (es. "5m fa", "2h 15m fa").

4.2.13 User-authentication

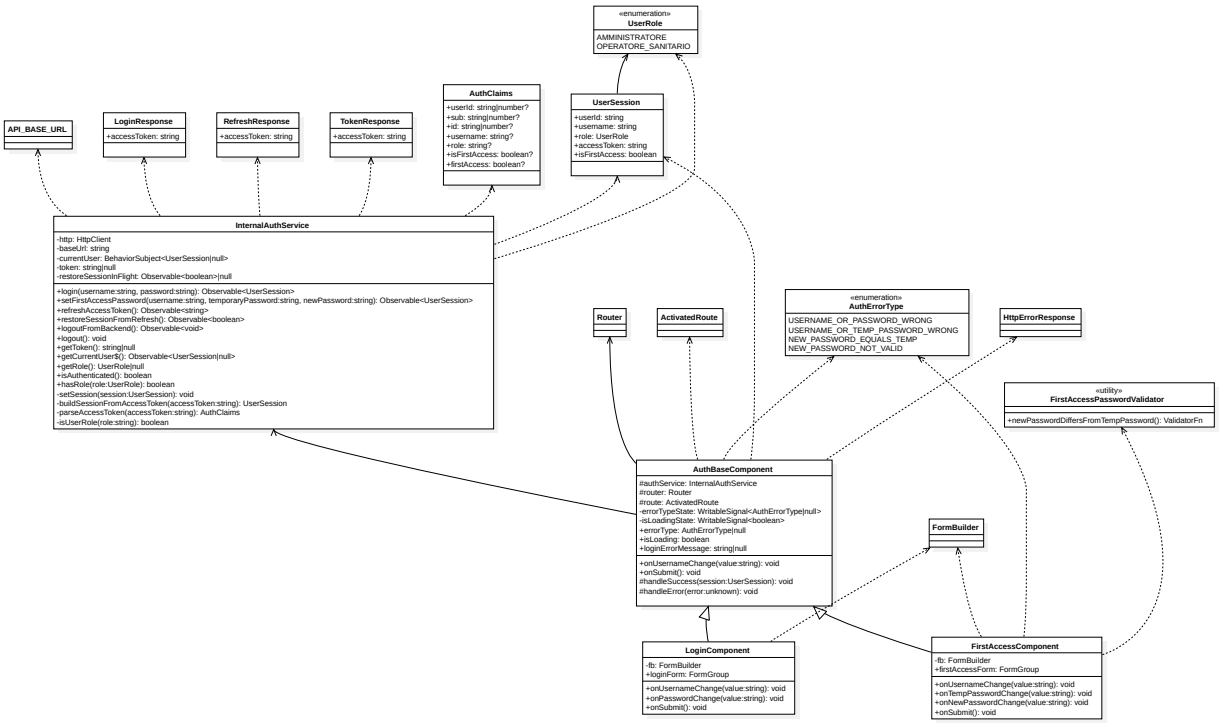

Figura 556: Diagramma delle classi del modulo User authentication

4.2.13.1 AuthBaseComponent

AuthBaseComponent

---

authService: InternalAuthService
router: Router
route: ActivatedRoute | null
- errorTypeState: WritableSignal<AuthErrorType | null>
- isLoadingState: WritableSignal<boolean>

+ errorType() : AuthErrorType | null
+ errorType(value: AuthErrorType | null) : void
+ isLoading() : boolean
+ isLoading(value: boolean) : void
+ loginErrorMessage() : string | null
+ *onUsernameChange(value: string) : void*
+ *onSubmit() : void*
# handleSuccess(session: UserSession) : void
# handleError(error: unknown) : void

Figura 557: Diagramma della classe AuthBaseComponent

Descrizione: Classe astratta base per i componenti di autenticazione, che gestisce lo stato di caricamento, il tipo di errore e la navigazione post-login tramite segnali reattivi. Definisce il contratto per i metodi di submit e cambio username che le sottoclassi devono implementare.

Descrizione dei metodi della classe:

- `errorType() : AuthErrorType | null`: Getter che restituisce il tipo di errore corrente leggendo il segnale interno.
- `errorType(value: AuthErrorType | null) : void`: Setter che aggiorna il tipo di errore corrente nel segnale interno.
- `isLoading() : boolean`: Getter che restituisce lo stato di caricamento corrente leggendo il segnale interno.
- `isLoading(value: boolean) : void`: Setter che aggiorna lo stato di caricamento nel segnale interno.
- `loginErrorMessage() : string | null`: Getter che restituisce il messaggio di errore localizzato corrispondente al tipo di errore corrente.
- *`onUsernameChange(value: string) : void`*: Metodo astratto che gestisce il cambiamento del valore del campo username.
- *`onSubmit() : void`*: Metodo astratto che gestisce il submit del form di autenticazione.
- `handleSuccess(session: UserSession) : void`: Gestisce il successo dell'autenticazione navigando verso la dashboard o l'URL di ritorno, con supporto al primo accesso.
- `handleError(error: unknown) : void`: Gestisce gli errori di autenticazione mappando i codici HTTP nei corrispondenti tipi di errore.

4.2.13.2 FirstAccessComponent

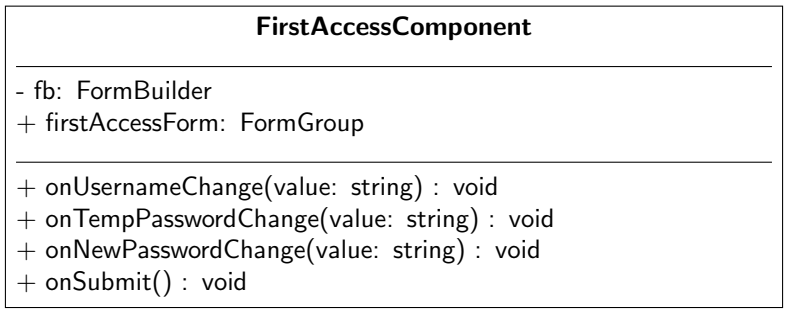

Figura 558: Diagramma della classe FirstAccessComponent

Descrizione: Componente Angular per la gestione del primo accesso, che consente all'utente di sostituire la password temporanea con una nuova. Estende AuthBaseComponent e aggiunge la validazione che la nuova password sia diversa da quella temporanea.

Descrizione dei metodi della classe:

- `onUsernameChange(value: string) : void`: Aggiorna il valore del campo username nel form di primo accesso.
- `onTempPasswordChange(value: string) : void`: Aggiorna il valore del campo password temporanea nel form di primo accesso.
- `onNewPasswordChange(value: string) : void`: Aggiorna il valore del campo nuova password nel form di primo accesso.
- `onSubmit() : void`: Valida il form e avvia la procedura di impostazione della nuova password, gestendo il successo e l'errore di autenticazione.

4.2.13.3 LoginComponent

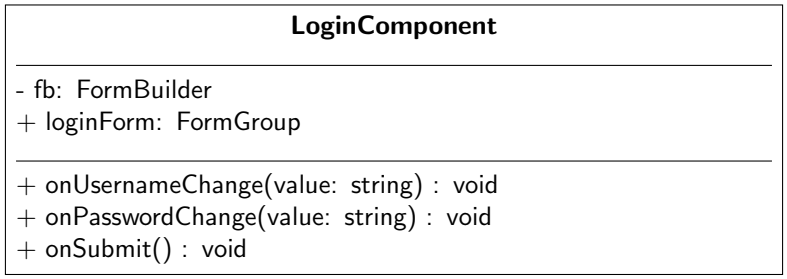

Figura 559: Diagramma della classe LoginComponent

Descrizione: Componente Angular per la gestione del login, che consente all'utente di autenticarsi tramite username e password. Estende AuthBaseComponent delegando la navigazione post-login e la gestione degli errori alla classe base.

Descrizione dei metodi della classe:

- `onUsernameChange(value: string) : void`: Aggiorna il valore del campo username nel form di login.
- `onPasswordChange(value: string) : void`: Aggiorna il valore del campo password nel form di login.
- `onSubmit() : void`: Valida il form e avvia la procedura di autenticazione, gestendo il successo e gli errori tramite i metodi ereditati da AuthBaseComponent.

4.2.13.4 UserSession

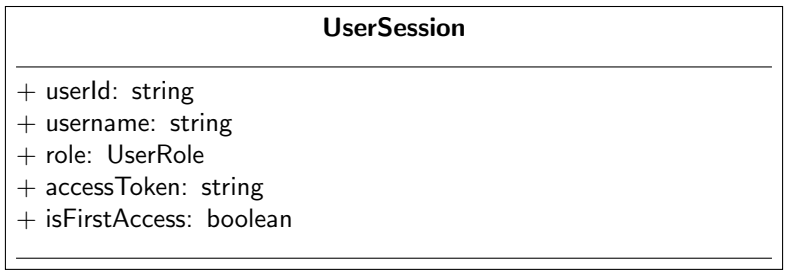

Figura 560: Diagramma della classe UserSession

Descrizione: Interfaccia che rappresenta la sessione di un utente autenticato. Raccoglie le informazioni identificative, il ruolo, il token di accesso e il flag di primo accesso.

4.2.13.5 LoginResponse

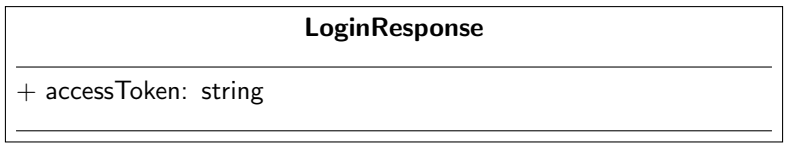

Figura 561: Diagramma della classe LoginResponse

Descrizione: Interfaccia che rappresenta la risposta del backend alla richiesta di login. Espone il token di accesso JWT da utilizzare per le richieste autenticate.

4.2.13.6 RefreshResponse

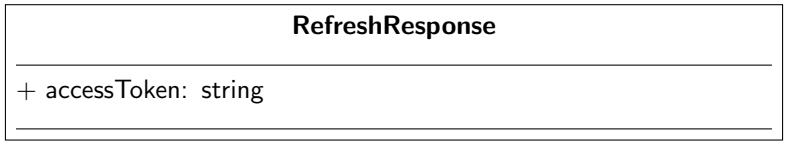

Figura 562: Diagramma della classe RefreshResponse

Descrizione: Interfaccia che rappresenta la risposta del backend alla richiesta di refresh del token. Espone il nuovo token di accesso JWT.

4.2.13.7 AuthClaims

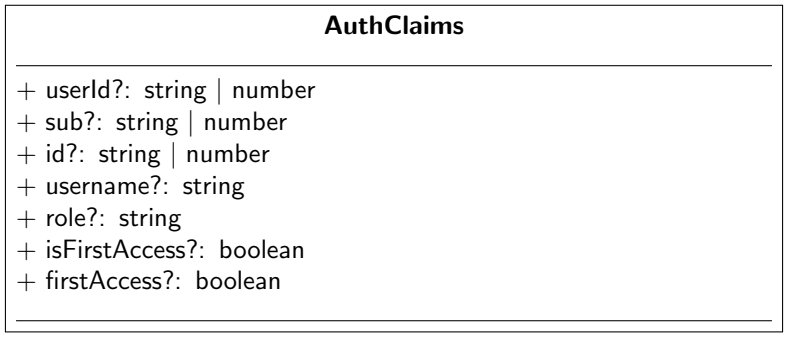

Figura 563: Diagramma della classe AuthClaims

Descrizione: Interfaccia che rappresenta i claims estratti dal payload di un token JWT di autenticazione. Gestisce le varianti di nomenclatura dei campi identificativi e del flag di primo accesso.

4.2.13.8 TokenResponse

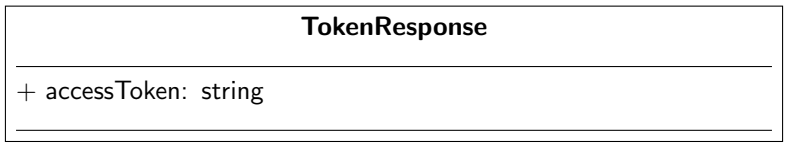

Figura 564: Diagramma della classe TokenResponse

Descrizione: Interfaccia generica che rappresenta una risposta del backend contenente un token di accesso JWT.

4.2.13.9 InternalAuthService

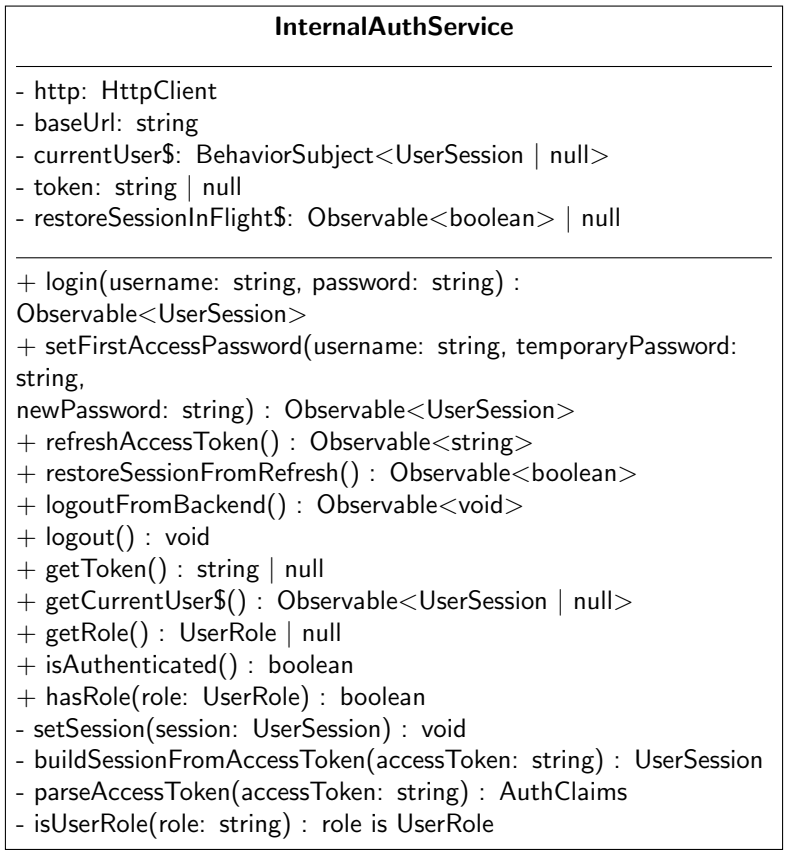

Figura 565: Diagramma della classe InternalAuthService

Descrizione: Servizio Angular per la gestione dell'autenticazione, che coordina login, logout, refresh del token e ripristino della sessione. Mantiene la sessione utente corrente tramite un BehaviorSubject e gestisce la decodifica e la validazione dei token JWT.

Descrizione dei metodi della classe:

- `login(username: string, password: string) : Observable<UserSession>`: Effettua il login inviando le credenziali al backend e costruisce la sessione utente a partire dal token di accesso ricevuto.
- `setFirstAccessPassword(username: string, temporaryPassword: string, newPassword: string) : Observable<UserSession>`: Invia la richiesta di impostazione della nuova password al primo accesso e costruisce la sessione utente dal token ricevuto.
- `refreshAccessToken() : Observable<string>`: Rinnova il token di accesso tramite il cookie di refresh e aggiorna la sessione corrente.
- `restoreSessionFromRefresh() : Observable<boolean>`: Tenta di ripristinare la sessione utente tramite refresh token, condividendo la richiesta in volo per evitare chiamate concorrenti.
- `logoutFromBackend() : Observable<void>`: Invia la richiesta di logout al backend e cancella la sessione locale indipendentemente dall'esito.
- `logout() : void`: Cancella il token e la sessione utente corrente dallo stato locale.
- `getToken() : string | null`: Restituisce il token di accesso corrente.

- `getCurrentUser$() : Observable<UserSession | null>`: Restituisce lo stream reattivo della sessione utente corrente.
- `getRole() : UserRole | null`: Restituisce il ruolo dell'utente corrente, o null se non autenticato.
- `isAuthenticated() : boolean`: Verifica se l'utente è autenticato controllando la presenza del token e della sessione.
- `hasRole(role: UserRole) : boolean`: Verifica se il ruolo dell'utente corrente corrisponde a quello specificato.
- `setSession(session: UserSession) : void`: Aggiorna il token e il BehaviorSubject della sessione con i dati della nuova sessione.
- `buildSessionFromAccessToken(accessToken: string) : UserSession`: Costruisce l'oggetto sessione estraendo e validando i claims dal payload del token JWT.
- `parseAccessToken(accessToken: string) : AuthClaims`: Decodifica il payload Base64 del token JWT e lo deserializza nei claims di autenticazione.
- `isUserRole(role: string) : role is UserRole`: Verifica se la stringa fornita corrisponde a un ruolo utente valido dell'enumerato UserRole.

4.2.14 User-management

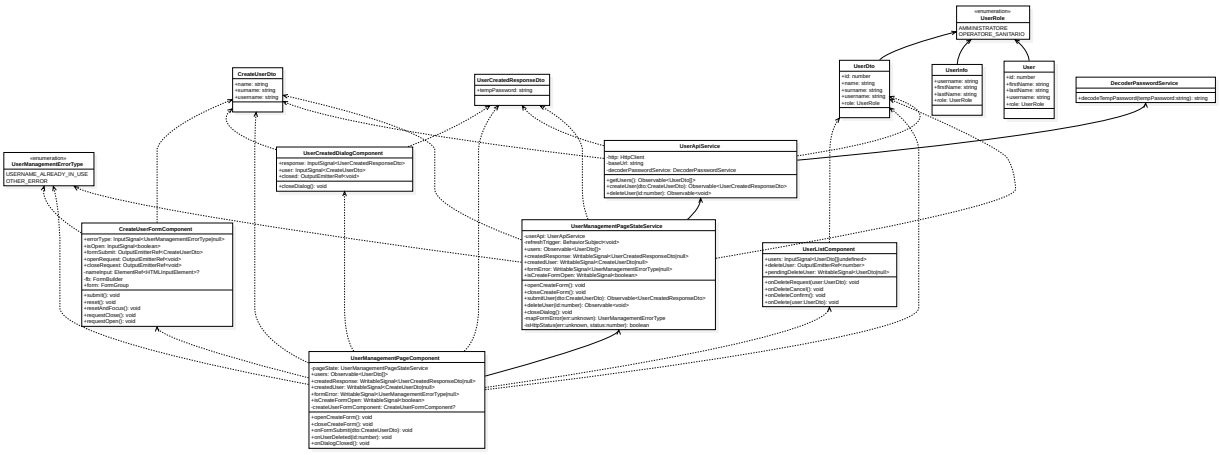

Figura 566: Diagramma delle classi del modulo User management

4.2.14.1 CreateUserFormComponent

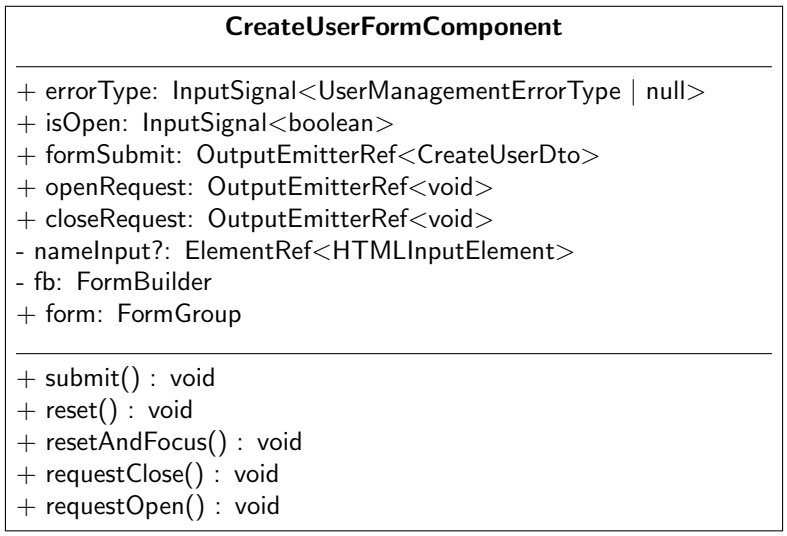

Figura 567: Diagramma della classe CreateUserFormComponent

Descrizione: Componente Angular che gestisce il form reattivo per la creazione di un nuovo utente. Espone segnali di input per lo stato e gli errori, ed eventi di output per le azioni di submit, apertura e chiusura.

Descrizione dei metodi della classe:

- `submit() : void`: Emette i dati del form se valido, altrimenti marca tutti i campi come toccati per mostrare gli errori di validazione.
- `reset() : void`: Reimposta tutti i campi del form ai valori vuoti iniziali.
- `resetAndFocus() : void`: Reimposta il form, ne ripristina lo stato pristine e untouched, e sposta il focus sul campo del nome.
- `requestClose() : void`: Emette l'evento `closeRequest` per segnalare la richiesta di chiusura del form.
- `requestOpen() : void`: Emette l'evento `openRequest` per segnalare la richiesta di apertura del form.

4.2.14.2 UserCreatedDialogComponent

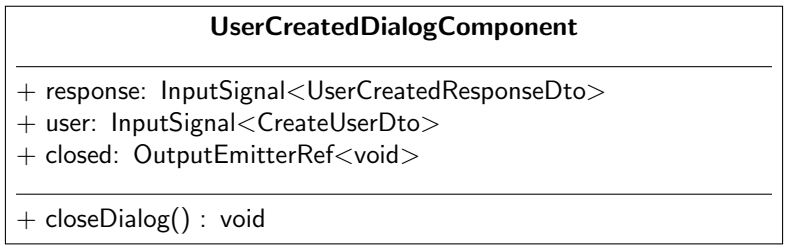

Figura 568: Diagramma della classe UserCreatedDialogComponent

Descrizione: Componente Angular che visualizza un dialog di conferma al termine della creazione di un utente. Viene mostrato solo quando i dati di risposta e dell'utente sono disponibili, grazie all'uso di input obbligatori.

Descrizione dei metodi della classe:

- `closeDialog() : void`: Emette l'evento `closed` per segnalare la chiusura del dialog.

4.2.14.3 UserListComponent

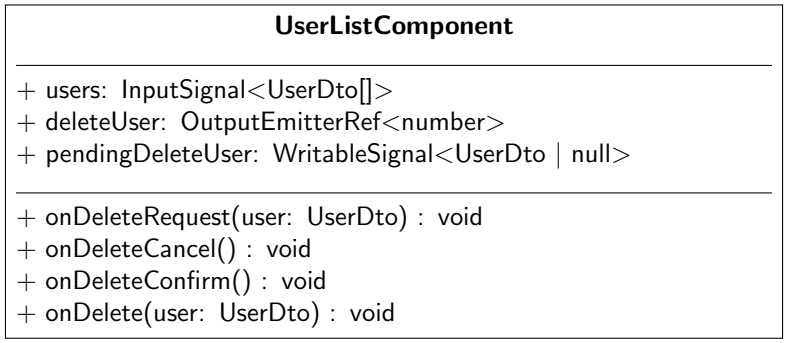

Figura 569: Diagramma della classe UserListComponent

Descrizione: Componente Angular che visualizza la lista degli utenti e gestisce il flusso di eliminazione con richiesta di conferma. Utilizza un segnale interno per tracciare l'utente in attesa di eliminazione prima di emettere l'evento definitivo.

Descrizione dei metodi della classe:

- `onDeleteRequest(user: UserDto) : void`: Imposta l'utente specificato come candidato all'eliminazione, avviando il flusso di conferma.
- `onDeleteCancel() : void`: Annulla l'eliminazione in corso reimpostando il segnale `pendingDeleteUser` a `null`.
- `onDeleteConfirm() : void`: Conferma l'eliminazione dell'utente in attesa, emette l'evento `deleteUser` con il suo id e reimposta il segnale a `null`.
- `onDelete(user: UserDto) : void`: Gestisce la richiesta di eliminazione di un utente delegando a `onDeleteRequest`.

4.2.14.4 UserManagementPageComponent

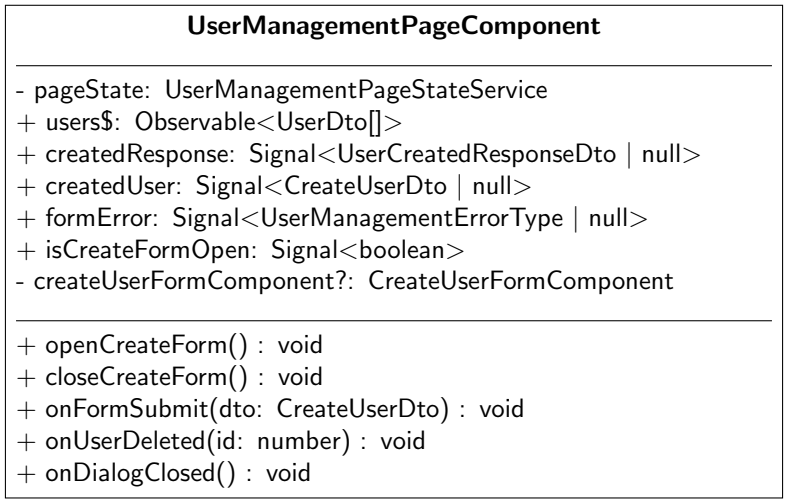

Figura 570: Diagramma della classe UserManagementPageComponent

Descrizione: Componente Angular che funge da pagina principale per la gestione degli utenti, orchestrando i sottocomponenti di lista, form di creazione e dialog di conferma. Delega tutta la logica di stato al servizio UserManagementPageStateService.

Descrizione dei metodi della classe:

- `openCreateForm() : void`: Delega al servizio di stato l'apertura del form di creazione utente.
- `closeCreateForm() : void`: Delega al servizio di stato la chiusura del form di creazione utente.
- `onFormSubmit(dto: CreateUserDto) : void`: Invia i dati del form al servizio di stato e, in caso di successo, reimposta il focus sul form.
- `onUserDeleted(id: number) : void`: Delega al servizio di stato l'eliminazione dell'utente con l'id specificato.
- `onDialogClosed() : void`: Delega al servizio di stato la chiusura del dialog di conferma creazione.

4.2.14.5 UserCreatedResponseDto

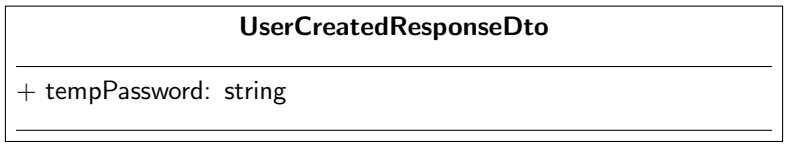

Figura 571: Diagramma della classe UserCreatedResponseDto

Descrizione: Interfaccia che rappresenta la risposta minima restituita dal backend alla creazione di un utente. Espone esclusivamente la password temporanea assegnata all'utente appena creato.

4.2.14.6 UserDto

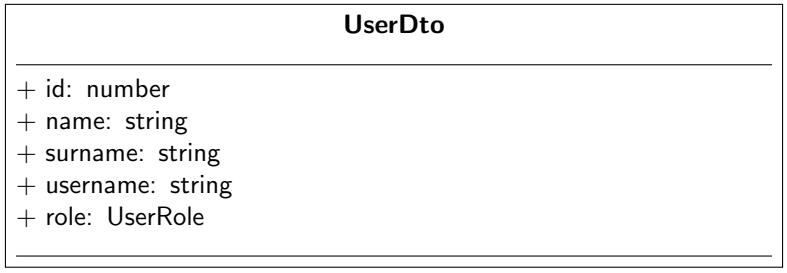

Figura 572: Diagramma della classe UserDto

Descrizione: Interfaccia che rappresenta i dati di un utente restituiti dal backend nelle chiamate GET. Raccoglie le informazioni anagrafiche e il ruolo dell'utente.

4.2.14.7 CreateUserDto

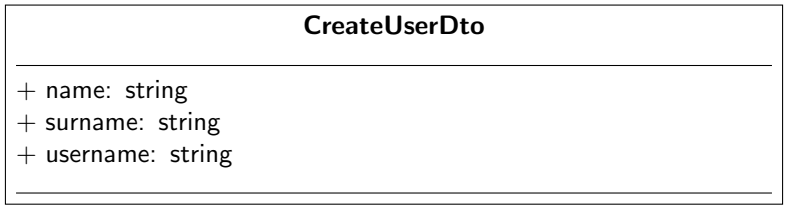

Figura 573: Diagramma della classe CreateUserDto

Descrizione: Interfaccia che rappresenta i dati necessari per la creazione di un nuovo utente. Raccoglie le informazioni anagrafiche e il nome utente da inviare al backend.

4.2.14.8 UserInfo

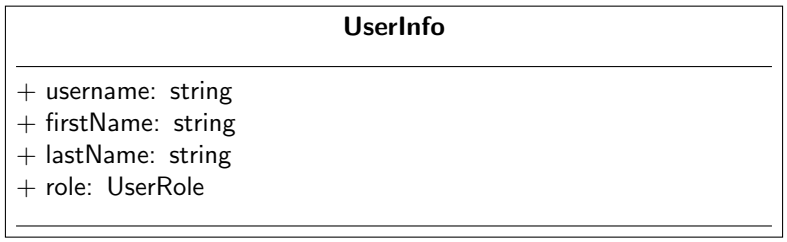

Figura 574: Diagramma della classe UserInfo

Descrizione: Interfaccia che rappresenta le informazioni di base di un utente autenticato. Raccoglie nome utente, dati anagrafici e ruolo.

4.2.14.9 User

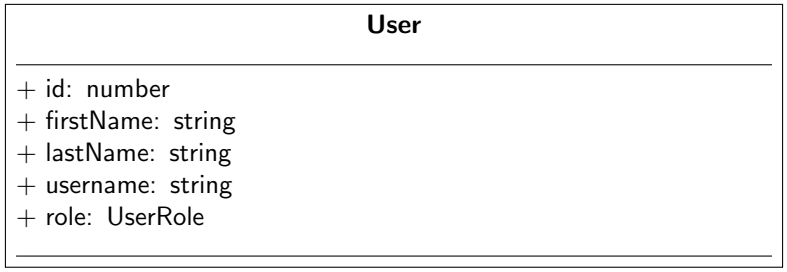

Figura 575: Diagramma della classe User

Descrizione: Interfaccia che rappresenta il modello di dominio di un utente. Raccoglie le informazioni anagrafiche, il nome utente e il ruolo assegnato.

4.2.14.10 DecoderPasswordService

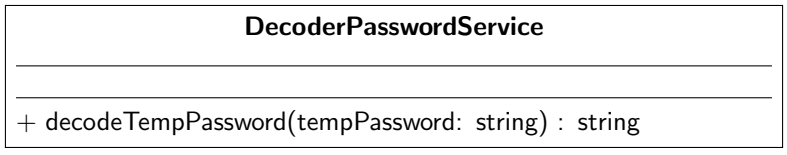

Figura 576: Diagramma della classe DecoderPasswordService

Descrizione: Servizio Angular per la decodifica delle password temporanee codificate in Base64. Verifica preventivamente che la stringa rispetti il formato Base64 prima di tentare la decodifica.

Descrizione dei metodi della classe:

- `decodeTempPassword(tempPassword: string) : string`: Decodifica una password temporanea da Base64, restituendo la stringa originale se il valore non è un Base64 valido o se la decodifica fallisce.

4.2.14.11 UserApiService

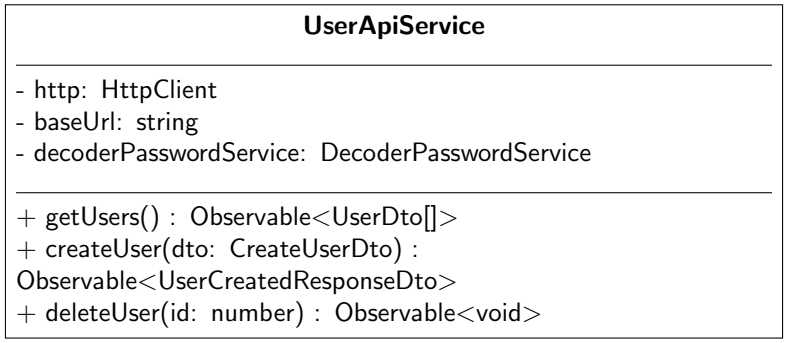

Figura 577: Diagramma della classe UserApiService

Descrizione: Servizio Angular per la comunicazione con le API REST relative alla gestione degli utenti. Espone operazioni di lettura, creazione ed eliminazione utenti, gestendo la decodifica della password temporanea restituita alla creazione.

Descrizione dei metodi della classe:

- `getUsers() : Observable<UserDto[]>`: Recupera la lista completa degli utenti tramite una richiesta GET al backend.
- `createUser(dto: CreateUserDto) : Observable<UserCreatedResponseDto>`: Invia una richiesta POST per la creazione di un nuovo utente e decodifica la password temporanea contenuta nella risposta.
- `deleteUser(id: number) : Observable<void>`: Invia una richiesta DELETE per eliminare l'utente con l'id specificato.

4.2.14.12 UserManagementPageStateService

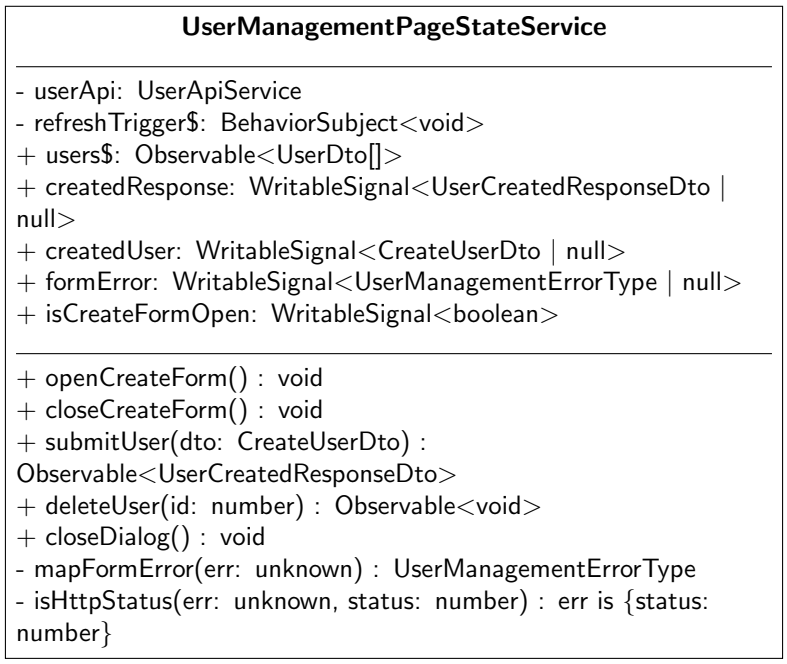

Figura 578: Diagramma della classe UserManagementPageStateService

Descrizione: Servizio di stato per la pagina di gestione utenti, responsabile del coordinamento tra le chiamate API e i segnali reattivi esposti ai componenti. Gestisce il ciclo di vita delle operazioni di creazione, eliminazione e refresh della lista utenti.

Descrizione dei metodi della classe:

- `openCreateForm() : void`: Imposta il segnale `isCreateFormOpen` a `true` per aprire il form di creazione.
- `closeCreateForm() : void`: Imposta il segnale `isCreateFormOpen` a `false` per chiudere il form di creazione.
- `submitUser(dto: CreateUserDto) : Observable<UserCreatedResponseDto>`: Invia i dati del nuovo utente all'API, aggiorna i segnali di stato e innesca il refresh della lista in caso di successo.
- `deleteUser(id: number) : Observable<void>`: Elimina l'utente con l'id specificato tramite l'API e innesca il refresh della lista in caso di successo.
- `closeDialog() : void`: Reimposta i segnali `createdResponse` e `createdUser` a `null` per chiudere il dialog di conferma.
- `mapFormError(err: unknown) : UserManagementErrorType`: Mappa un errore HTTP nel corrispondente tipo di errore del form, distinguendo il caso di username già in uso dagli altri errori.
- `isHttpStatus(err: unknown, status: number) : err is {status: number}`: Verifica se l'errore fornito è un oggetto con il codice di stato HTTP specificato.

4.2.15 Ward-management

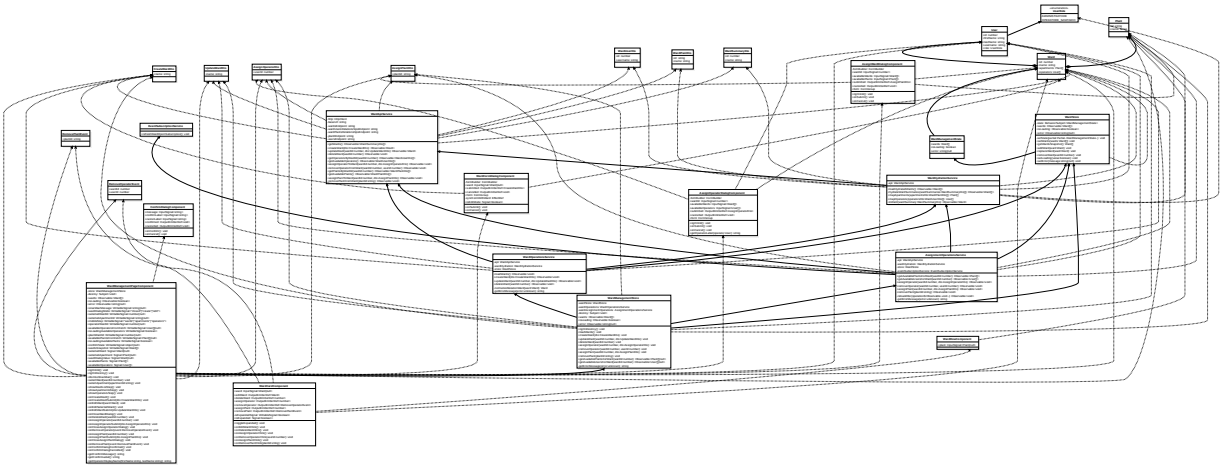

Figura 579: Diagramma delle classi del modulo Ward management

4.2.15.1 AssignOperatorDialogComponent

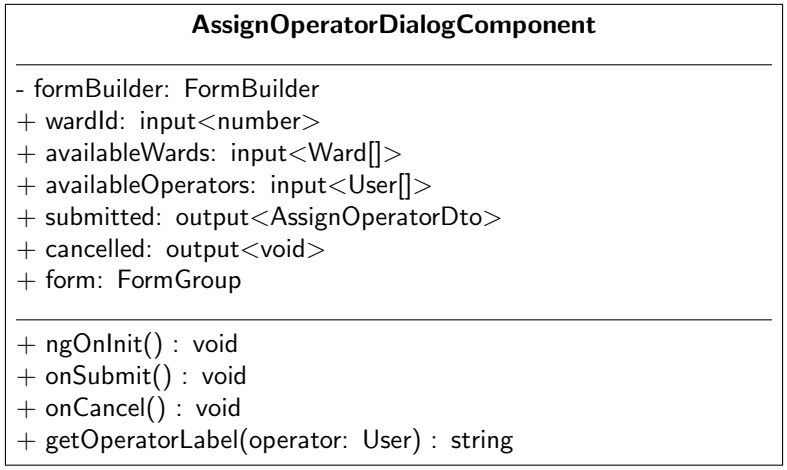

Figura 580: Diagramma della classe AssignOperatorDialogComponent

Descrizione: Componente che gestisce l'interfaccia di dialogo per l'assegnazione di un operatore sanitario a un reparto specifico. Utilizza i Reactive Forms di Angular per la gestione della selezione e della validazione dell'input utente.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il componente resettando lo stato del modulo reattivo per garantire un punto di partenza pulito a ogni apertura del dialogo.
- `onSubmit() : void`: Valida i dati del modulo e, in caso di successo, emette l'evento di assegnazione con l'identificativo dell'operatore selezionato.
- `onCancel() : void`: Notifica al componente genitore l'annullamento dell'operazione di assegnazione.
- `getOperatorLabel(operator: User) : string`: Costruisce un'etichetta leggibile per l'operatore, privilegiando il nome completo rispetto allo username.

4.2.15.2 AssignWardDialogComponent

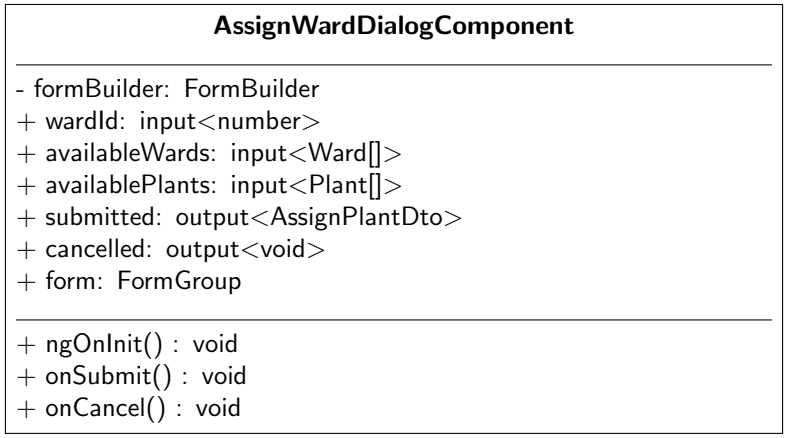

Figura 581: Diagramma della classe AssignWardDialogComponent

Descrizione: Componente che gestisce l'interfaccia di dialogo per l'assegnazione di un impianto a un determinato reparto. Sfrutta i Reactive Forms di Angular per gestire la logica di selezione e validazione dell'identificativo impianto.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza lo stato del componente resettando il controllo del modulo reattivo dedicato alla selezione dell'impianto.
- `onSubmit() : void`: Valida il modulo e, se i dati sono corretti, emette l'evento di sottomissione con l'identificativo dell'impianto (plantId) scelto per l'assegnazione al reparto.
- `onCancel() : void`: Emette un segnale di annullamento per notificare la chiusura del dialogo senza apportare modifiche.

4.2.15.3 WardCardComponent

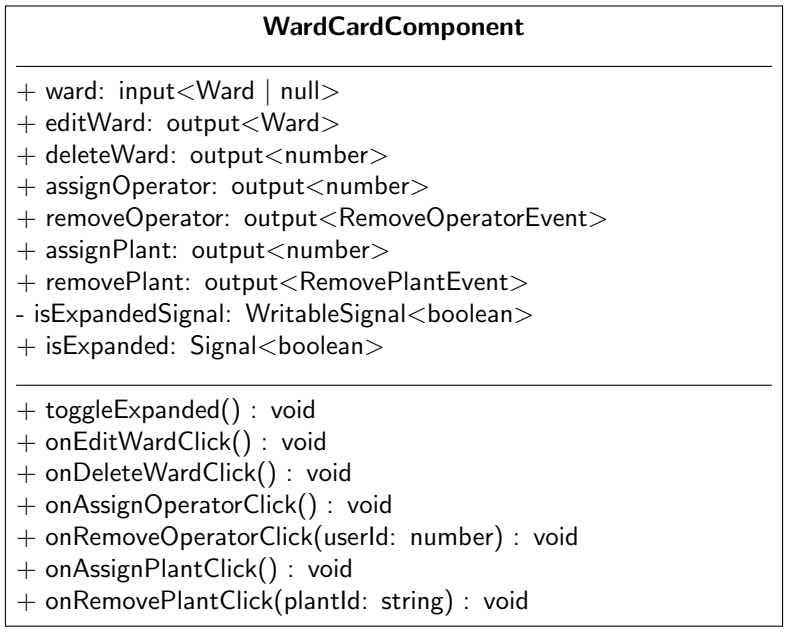

Figura 582: Diagramma della classe WardCardComponent

Descrizione: Componente di presentazione che visualizza le informazioni sintetiche di un reparto sotto forma di scheda espandibile. Coordina le interazioni dell'utente per la gestione del personale sanitario e degli impianti associati, delegando le operazioni di business tramite una serie di eventi in output.

Descrizione dei metodi della classe:

- `toggleExpanded() : void`: Inverte lo stato di espansione della scheda per mostrare o nascondere i dettagli aggiuntivi del reparto.
- `onEditWardClick() : void`: Emette l'evento di modifica passando l'oggetto reparto corrente se presente.
- `onDeleteWardClick() : void`: Invia l'identificativo del reparto per richiederne la rimozione dal sistema.
- `onAssignOperatorClick() : void`: Sollecita l'apertura della logica di assegnazione di un nuovo operatore per il reparto attivo.
- `onRemoveOperatorClick(userId: number) : void`: Emette un evento strutturato per dissociare uno specifico operatore sanitario dal reparto.
- `onAssignPlantClick() : void`: Innesca la procedura per collegare un nuovo impianto tecnologico al reparto.
- `onRemovePlantClick(plantId: string) : void`: Emette l'evento di rimozione dell'associazione tra l'impianto specificato e il reparto.

4.2.15.4 WardFormDialogComponent

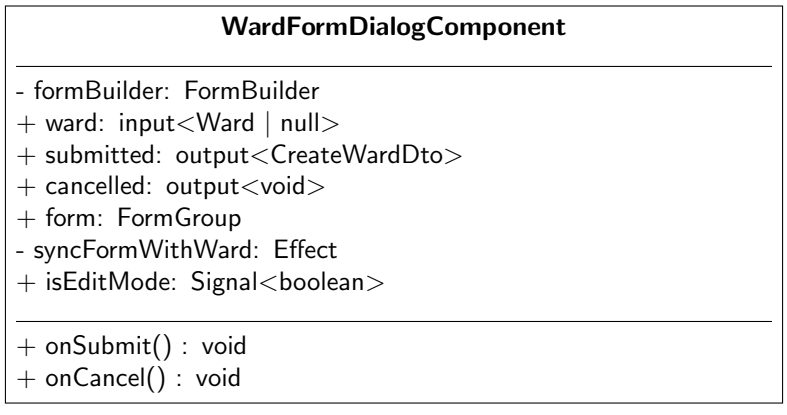

Figura 583: Diagramma della classe WardFormDialogComponent

Descrizione: Componente di interfaccia per la creazione e la modifica dei reparti. Utilizza un approccio reattivo basato su **Signals** ed **Effects** per sincronizzare automaticamente i dati del modulo con l'input esterno, gestendo dinamicamente lo stato di editing e le regole di validazione del nome.

Descrizione dei metodi della classe:

- `onSubmit() : void`: Valida il contenuto del modulo e, in caso di successo, emette i dati del reparto (DTO) per la creazione o l'aggiornamento, applicando il trimming al nome.
- `onCancel() : void`: Notifica la volontà dell'utente di chiudere il dialogo e annullare l'inserimento o la modifica dei dati.

4.2.15.5 WardManagementPageComponent

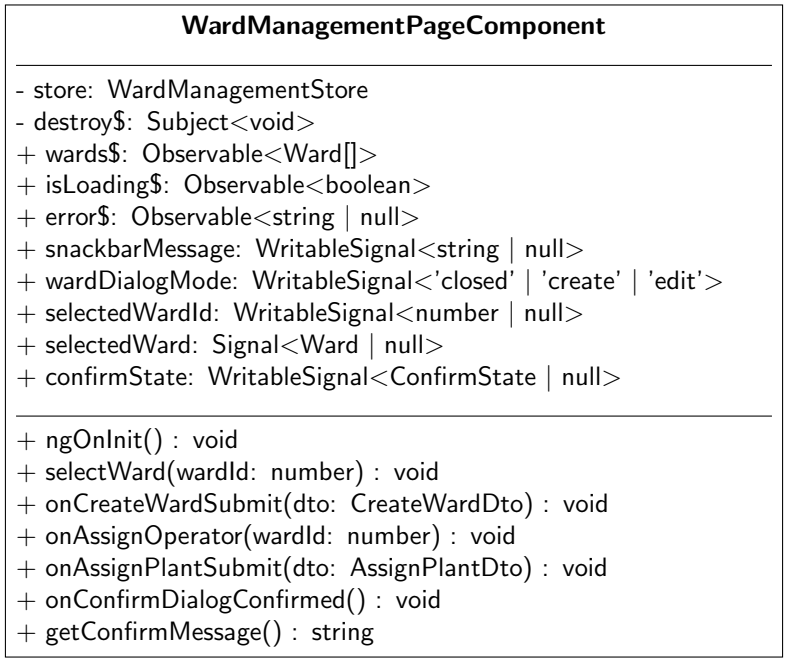

Figura 584: Diagramma della classe WardManagementPageComponent

Descrizione: Componente "Smart" che funge da pagina principale per la gestione dei reparti. Coordina l'interazione tra lo store dei dati e i vari componenti di dialogo (creazione, assegnazione operatori, impianti), gestendo inoltre la navigazione a step per i dispositivi mobili e il feedback utente tramite snackbar.

Descrizione dei metodi della classe:

- `ngOnInit() : void`: Inizializza il componente caricando i reparti dallo store e sottoscrivendo gli stream per aggiornare lo stato locale e la gestione degli errori.
- `selectWard(wardId: number) : void`: Gestisce la selezione di un reparto, aggiornando la vista e resettando la selezione degli appartamenti correlati.
- `onCreateWardSubmit(dto: CreateWardDto) : void`: Invia la richiesta di creazione di un nuovo reparto allo store e chiude la finestra di dialogo.
- `onAssignOperator(wardId: number) : void`: Avvia il processo di assegnazione operatore, recuperando la lista degli utenti disponibili per lo specifico reparto.
- `onAssignPlantSubmit(dto: AssignPlantDto) : void`: Finalizza l'associazione di un impianto al reparto selezionato tramite lo store.
- `onConfirmDialogConfirmed() : void`: Esegue l'azione di business (cancellazione reparto o rimozione associazione) basandosi sulla tipologia di conferma attiva.
- `getConfirmMessage() : string`: Restituisce un messaggio testuale contestualizzato per il dialogo di conferma in base all'operazione che l'utente sta eseguendo.

4.2.15.6 WardRowComponent

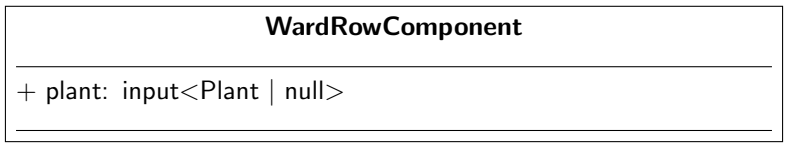

Figura 585: Diagramma della classe WardRowComponent

Descrizione: Componente di presentazione granulare utilizzato per visualizzare i dettagli di un singolo impianto (appartamento) all'interno della lista di un reparto. Riceve i dati tramite un input reattivo e adotta la strategia di **OnPush** per ottimizzare le prestazioni di rendering.

4.2.15.7 Ward

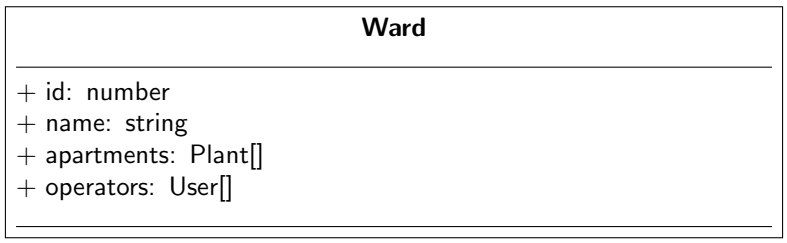

Figura 586: Diagramma della classe Ward

Descrizione: Questa interfaccia definisce il modello dati di un reparto all'interno del sistema. Rappresenta l'entità organizzativa principale, collegando un identificativo unico e un nome a collezioni di impianti (appartamenti) e operatori sanitari assegnati.

4.2.15.8 Plant

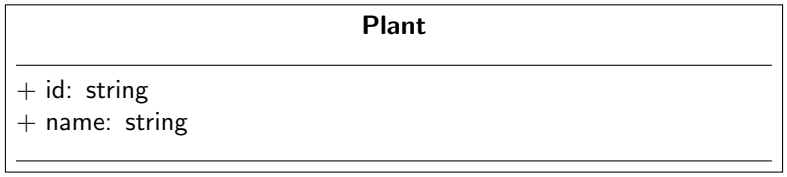

Figura 587: Diagramma della classe Plant

Descrizione: Questa interfaccia definisce il modello dati di un impianto (spesso riferito a un appartamento) all'interno del sistema. Rappresenta l'unità tecnologica fondamentale, identificata da un codice univoco e un nome descrittivo, che viene associata ai reparti per il monitoraggio.

4.2.15.9 CreateWardDto

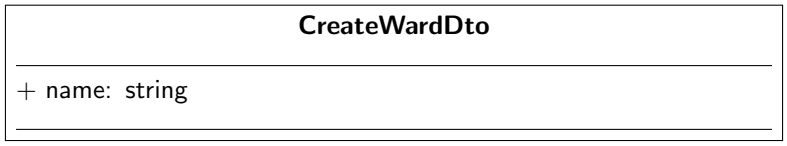

Figura 588: Diagramma della classe CreateWardDto

Descrizione: Data Transfer Object utilizzato per l'invio dei dati necessari alla creazione di un nuovo reparto, contenente esclusivamente il nome descrittivo.

4.2.15.10 UpdateWardDto

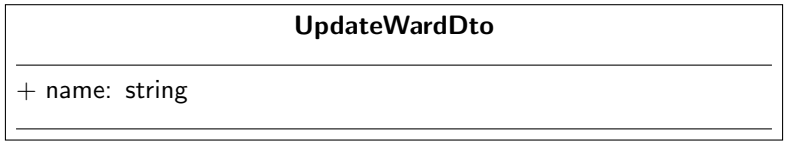

Figura 589: Diagramma della classe UpdateWardDto

Descrizione: Data Transfer Object per l'aggiornamento delle informazioni di un reparto esistente.

4.2.15.11 AssignOperatorDto

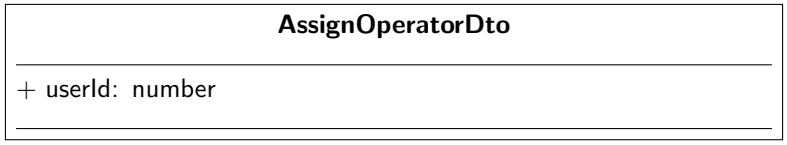

Figura 590: Diagramma della classe AssignOperatorDto

Descrizione: Oggetto per il trasferimento dei dati relativi all'associazione di un operatore sanitario a un reparto tramite il suo identificativo utente.

4.2.15.12 AssignPlantDto

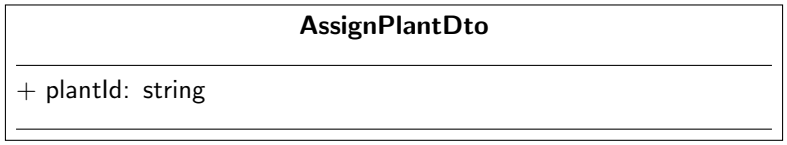

Figura 591: Diagramma della classe AssignPlantDto

Descrizione: Oggetto per il trasferimento dei dati relativi al collegamento di un impianto a un reparto tramite l'identificativo dell'impianto.

4.2.15.13 WardUserDto

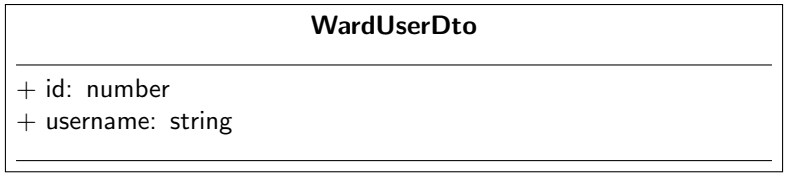

Figura 592: Diagramma della classe WardUserDto

Descrizione: Rappresentazione semplificata di un utente all'interno del contesto di gestione reparti, focalizzata su ID e nome utente.

4.2.15.14 WardPlantDto

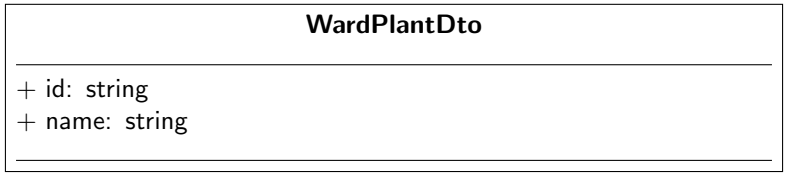

Figura 593: Diagramma della classe WardPlantDto

Descrizione: Rappresentazione sintetica di un impianto associato a un reparto, utilizzata per il trasferimento dati tra client e server.

4.2.15.15 WardSummaryDto

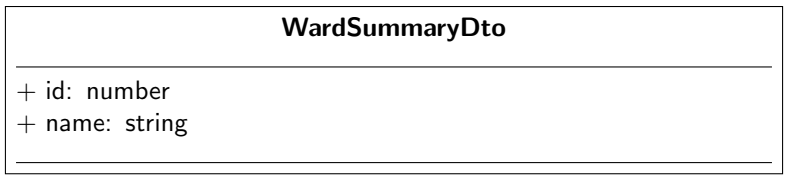

Figura 594: Diagramma della classe WardSummaryDto

Descrizione: Modello dati minimale che fornisce un riepilogo delle informazioni essenziali di un reparto.

4.2.15.16 WardManagementState

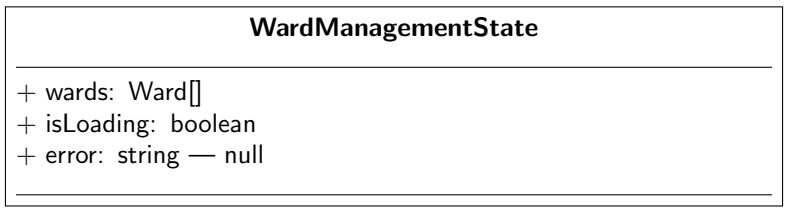

Figura 595: Diagramma della classe WardManagementState

Descrizione: Interfaccia che definisce lo stato globale del modulo di gestione reparti, includendo la lista dei reparti, lo stato di caricamento e la gestione degli errori.

4.2.15.17 RemoveOperatorEvent

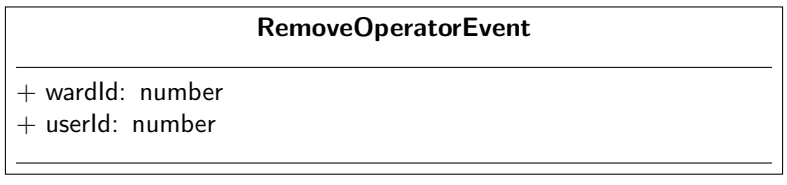

Figura 596: Diagramma della classe RemoveOperatorEvent

Descrizione: Interfaccia che definisce la struttura dell'evento di rimozione di un operatore, specificando le chiavi esterne del reparto e dell'utente.

4.2.15.18 RemovePlantEvent

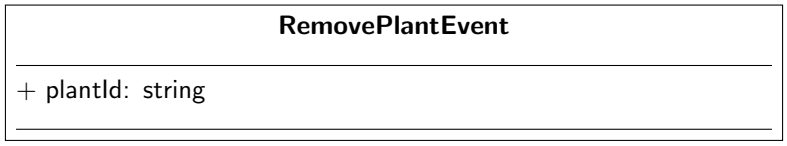

Figura 597: Diagramma della classe RemovePlantEvent

Descrizione: Interfaccia per l'evento di rimozione di un impianto, contenente l'identificativo univoco dell'asset da dissociare.

4.2.15.19 AssignmentOperationsService

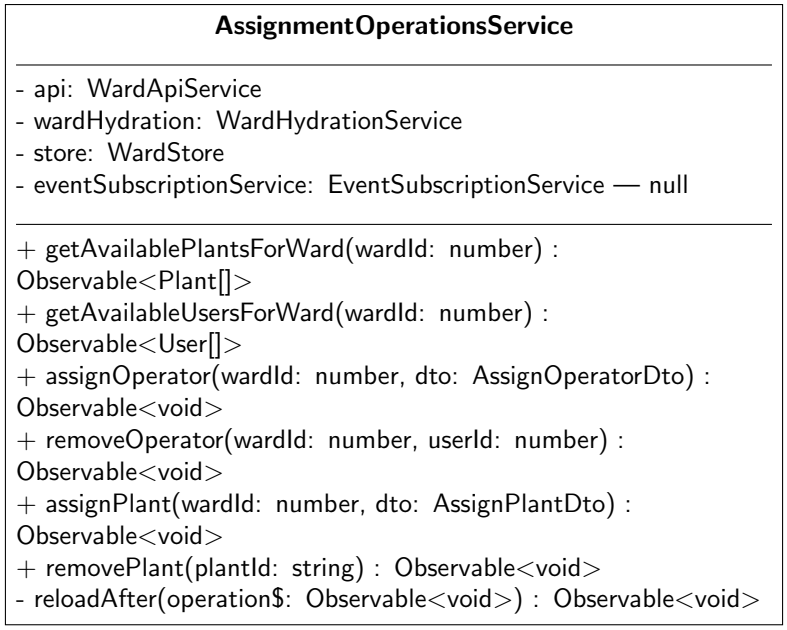

Figura 598: Diagramma della classe AssignmentOperationsService

Descrizione: Servizio di dominio preposto alla gestione delle relazioni tra reparti, operatori e impianti. Centralizza la logica di assegnazione e rimozione delle risorse, garantendo che lo stato dell'applicazione e le sottoscrizioni agli eventi (WebSocket) rimangano sincronizzati con i cambiamenti effettuati sul backend.

Descrizione dei metodi della classe:

- `getAvailablePlantsForWard(wardId: number) : Observable<Plant[]>`: Recupera gli impianti disponibili filtrando quelli già assegnati ad altri reparti e ne esegue l'idratazione dei dati.
- `getAvailableUsersForWard(wardId: number) : Observable<User[]>`: Fornisce l'elenco degli operatori che possono essere assegnati a un reparto, escludendo quelli già presenti nel reparto specificato.

- `assignOperator(wardId: number, dto: AssignOperatorDto) : Observable<void>`: Esegue l'associazione di un operatore a un reparto e innesca il ricaricamento dello stato.
- `removeOperator(wardId: number, userId: number) : Observable<void>`: Rimuove l'associazione di un operatore e aggiorna i dati sincronizzati nel sistema.
- `assignPlant(wardId: number, dto: AssignPlantDto) : Observable<void>`: Assegna un impianto tecnologico a un reparto, gestendo la successiva propagazione degli aggiornamenti nello store.
- `removePlant(plantId: string) : Observable<void>`: Dissocia un impianto dal relativo reparto e aggiorna le sottoscrizioni agli eventi di allarme.
- `reloadAfter(operation$: Observable<void>) : Observable<void>`: Metodo di utilità che incapsula la logica di post-esecuzione: ricarica i reparti idratati, aggiorna lo store globale e rinfresca le sottoscrizioni agli eventi.

4.2.15.20 WardApiService

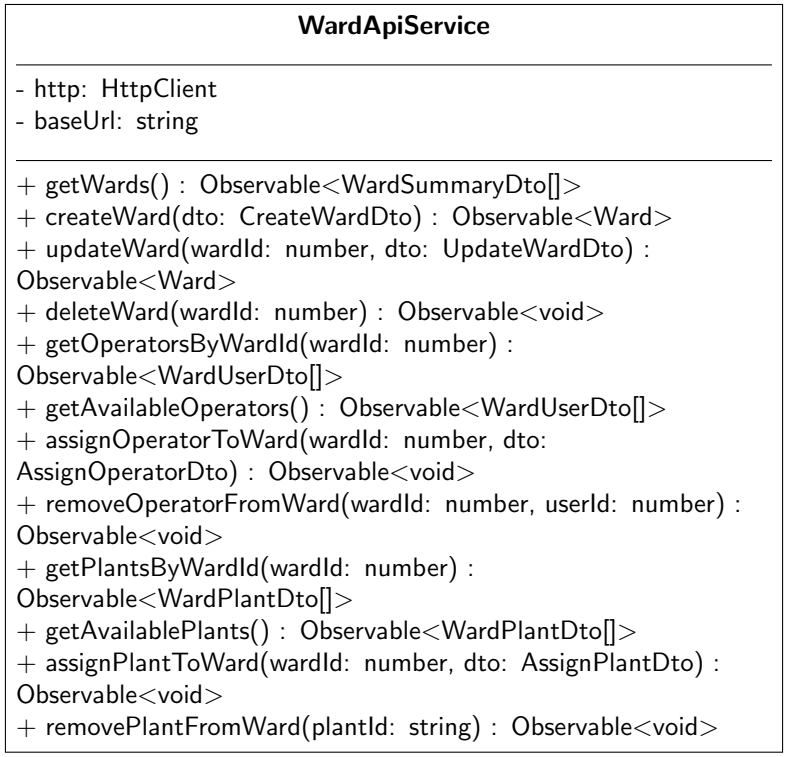

Figura 599: Diagramma della classe WardApiService

Descrizione: Servizio di comunicazione dati di basso livello che incapsula tutte le chiamate HTTP verso le API REST dedicate alla gestione dei reparti (Wards). Gestisce le operazioni CRUD e le relazioni molti-a-molti con operatori e impianti, garantendo la corretta formattazione degli URL e la gestione dei parametri tramite encodeURIComponent.

Descrizione dei metodi della classe:

- `getWards() : Observable<WardSummaryDto[]>`: Recupera la lista sintetica di tutti i reparti censiti nel sistema.
- `createWard(dto: CreateWardDto) : Observable<Ward>`: Invia una richiesta POST per la creazione di un nuovo reparto.

- `updateWard(wardId: number, dto: UpdateWardDto) : Observable<Ward>`: Aggiorna i dati di un reparto esistente tramite una chiamata PUT.
- `deleteWard(wardId: number) : Observable<void>`: Elimina definitivamente un reparto dal sistema.
- `getOperatorsByWardId(wardId: number) : Observable<WardUserDto[]>`: Ottiene l'elenco dei soli operatori associati a uno specifico reparto.
- `getAvailableOperators() : Observable<WardUserDto[]>`: Recupera la lista globale di tutti gli utenti/operatori potenzialmente assegnabili.
- `assignOperatorToWard(wardId: number, dto: AssignOperatorDto) : Observable<void>`: Crea una nuova relazione nel database tra un reparto e un operatore.
- `removeOperatorFromWard(wardId: number, userId: number) : Observable<void>`: Rimuove il legame associativo tra un reparto e un operatore specifico.
- `getPlantsByWardId(wardId: number) : Observable<WardPlantDto[]>`: Recupera gli impianti (plant) attualmente collegati a un determinato reparto.
- `getAvailablePlants() : Observable<WardPlantDto[]>`: Interroga il sistema per ottenere la lista completa di tutti gli impianti disponibili.
- `assignPlantToWard(wardId: number, dto: AssignPlantDto) : Observable<void>`: Stabilisce un'associazione formale tra un reparto e un impianto tecnologico.
- `removePlantFromWard(plantId: string) : Observable<void>`: Rimuove l'associazione di un impianto, svincolandolo dal reparto di appartenenza.

4.2.15.21 WardHydrationService

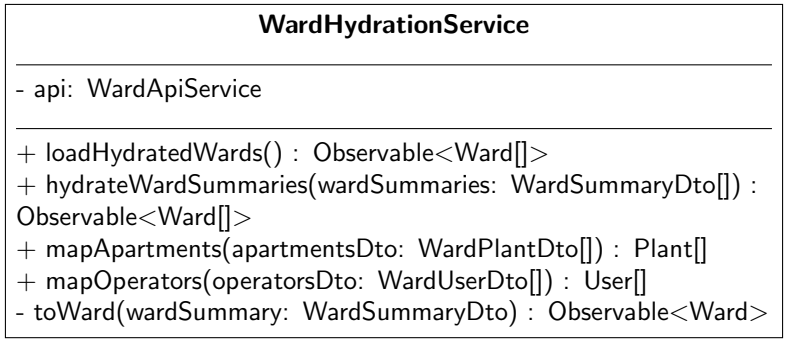

Figura 600: Diagramma della classe WardHydrationService

Descrizione: Servizio di orchestrazione responsabile del processo di "idratazione" dei dati. Il suo compito principale è trasformare oggetti DTO parziali in entità di dominio complete, aggregando informazioni provenienti da endpoint diversi (reparti, relazioni utenti, relazioni impianti) per fornire una vista coerente e pronta all'uso per l'interfaccia utente.

Descrizione dei metodi della classe:

- `loadHydratedWards() : Observable<Ward[]>`: Innesca il processo di caricamento completo dei reparti, recuperando prima le sintesi e poi arricchendo ogni elemento con i dati di dettaglio.
- `hydrateWardSummaries(wardSummaries: WardSummaryDto[]) : Observable<Ward[]>`: Coordina la risoluzione parallela delle dipendenze per una lista di riepiloghi reparti utilizzando **forkJoin** per gestire le richieste asincrone simultanee.

- `mapApartments(apartmentsDto: WardPlantDto[]) : Plant[]`: Esegue la mappatura strutturale tra i DTO degli impianti provenienti dal backend e il modello dati interno dell'applicazione.
- `mapOperators(operatorsDto: WardUserDto[]) : User[]`: Trasforma i dati sintetici degli utenti in oggetti operatore completi, assegnando ruoli predefiniti e normalizzando i campi anagrafici.
- `toWard(wardSummary: WardSummaryDto) : Observable<Ward>`: Metodo privato che aggrega le chiamate API per impianti e operatori di un singolo reparto, componendo l'oggetto di dominio finale **Ward**.

4.2.15.22 WardManagementStore

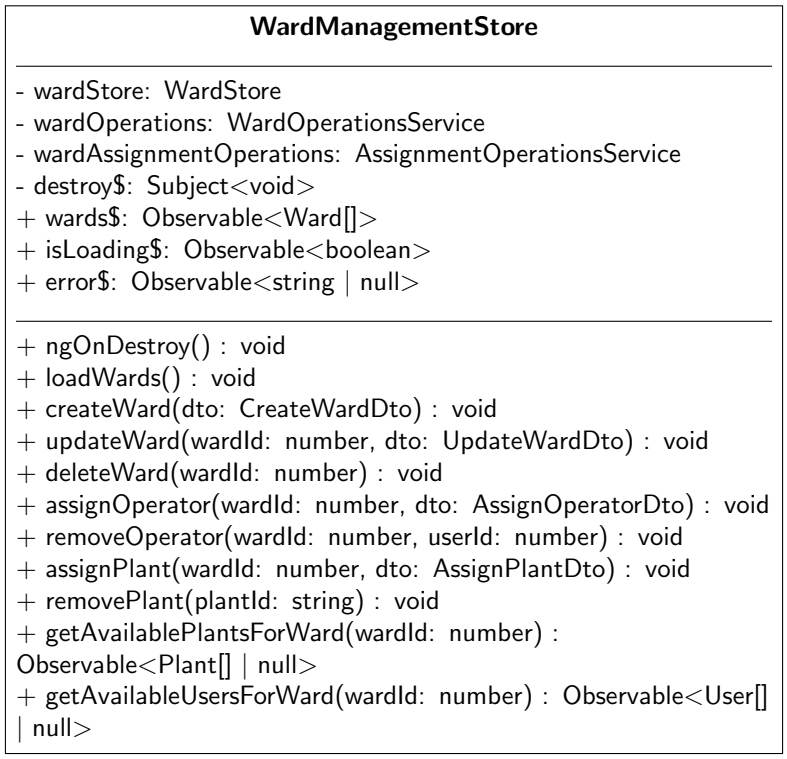

Figura 601: Diagramma della classe WardManagementStore

Descrizione: Classe Facade che funge da punto di ingresso unico per la gestione dello stato e delle operazioni del modulo reparti. Coordina l'interazione tra i servizi operativi (WardOperations, AssignmentOperations) e lo storage dei dati (WardStore), esponendo stream reattivi per l'interfaccia utente e gestendo il ciclo di vita delle operazioni asincrone.

Descrizione dei metodi della classe:

- `ngOnDestroy() : void`: Gestisce la pulizia delle sottoscrizioni attive tramite l'emissione del segnale sul Subject di distruzione.
- `loadWards() : void`: Innesca il caricamento dei reparti aggiornando lo stato di loading e gestendo la sottoscrizione allo stream operativo.
- `createWard(dto: CreateWardDto) : void`: Coordina la creazione di un nuovo reparto delegando l'operazione al servizio dedicato.
- `updateWard(wardId: number, dto: UpdateWardDto) : void`: Gestisce l'aggiornamento dei dati di un reparto esistente.

- `deleteWard(wardId: number) : void`: Esegue la rimozione di un reparto dal sistema.
- `assignOperator(wardId: number, dto: AssignOperatorDto) : void`: Avvia la procedura di associazione di un operatore sanitario a un reparto.
- `removeOperator(wardId: number, userId: number) : void`: Gestisce la dissociazione di un operatore da un reparto.
- `assignPlant(wardId: number, dto: AssignPlantDto) : void`: Innesca l'assegnazione di un impianto tecnologico a un reparto specifico.
- `removePlant(plantId: string) : void`: Esegue la rimozione di un impianto dal reparto a cui è associato.
- `getAvailablePlantsForWard(wardId: number) : Observable<Plant[] | null>`: Recupera gli impianti disponibili per l'assegnazione, gestendo eventuali errori di recupero dati.
- `getAvailableUsersForWard(wardId: number) : Observable<User[] | null>`: Recupera l'elenco degli utenti assegnabili al reparto, fornendo una gestione degli errori integrata.

4.2.15.23 WardOperationsService

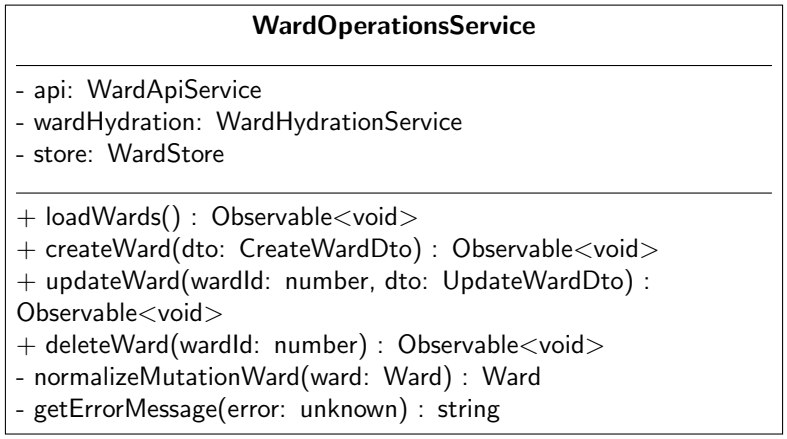

Figura 602: Diagramma della classe WardOperationsService

Descrizione: Servizio operativo di dominio che orchestra le operazioni CRUD sui reparti. Funge da ponte tra i servizi di comunicazione (API), i servizi di trasformazione (Hydration) e la gestione dello stato (Store), garantendo che ogni transazione sui dati sia riflessa correttamente nell'interfaccia utente.

Descrizione dei metodi della classe:

- `loadWards() : Observable<void>`: Innesca il caricamento asincrono dei reparti idratati, aggiornando lo store globale e gestendo eventuali errori di rete o di parsing.
- `createWard(dto: CreateWardDto) : Observable<void>`: Invia la richiesta di creazione di un nuovo reparto; gestisce specificamente l'errore di conflitto (409) in caso di nome duplicato e aggiorna lo store con l'entità normalizzata.
- `updateWard(wardId: number, dto: UpdateWardDto) : Observable<void>`: Esegue l'aggiornamento dei dati anagrafici di un reparto e sostituisce la versione precedente nello store mantenendo l'integrità delle relazioni esistenti.
- `deleteWard(wardId: number) : Observable<void>`: Rimuove un reparto dal sistema tramite API e aggiorna coerentemente lo stato locale eliminando l'entità corrispondente.

- `normalizeMutationWard(ward: Ward) : Ward`: Assicura che, a seguito di una mutazione (creazione o modifica), l'oggetto reparto mantenga le liste di appartamenti e operatori recuperandole dallo stato corrente se non fornite dal server.
- `getErrorMessage(error: unknown) : string`: Analizza l'eccezione catturata per estrarre un messaggio di errore leggibile, privilegiando il contenuto della risposta HTTP o il messaggio di sistema.

4.2.15.24 WardStore

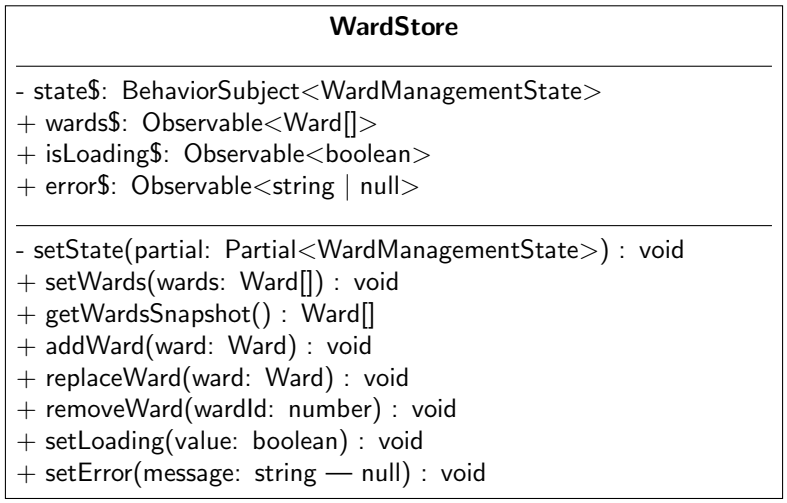

Figura 603: Diagramma della classe WardStore

Descrizione: Classe di gestione dello stato a basso livello (Data Store) basata sul pattern **BehaviorSubject**. Funge da "Single Source of Truth" locale per il modulo dei reparti, esponendo flussi di dati reattivi e immutabili per garantire la coerenza della UI durante le operazioni di manipolazione dei dati.

Descrizione dei metodi della classe:

- `setState(partial: Partial<WardManagementState>) : void`: Metodo privato per l'aggiornamento atomico dello stato locale, garantendo l'immutabilità tramite la sovrascrittura selettiva delle proprietà.
- `setWards(wards: Ward[]) : void`: Sovrascrive l'intera collezione dei reparti nello stato corrente.
- `getWardsSnapshot() : Ward[]`: Restituisce il valore sincronizzato attuale della lista reparti senza attivare una sottoscrizione allo stream.
- `addWard(ward: Ward) : void`: Aggiunge un nuovo reparto alla collezione esistente e resetta contemporaneamente gli indicatori di caricamento ed errore.
- `replaceWard(ward: Ward) : void`: Aggiorna un reparto specifico all'interno della lista, mappando l'ID fornito con quello esistente per sostituirne i dati.
- `removeWard(wardId: number) : void`: Elimina un reparto dallo stato filtrando la collezione in base all'identificativo fornito.
- `setLoading(value: boolean) : void`: Aggiorna lo stato di attesa dell'interfaccia durante le operazioni asincrone.
- `setError(message: string | null) : void`: Configura il messaggio di errore da visualizzare all'utente, forzando la disattivazione dello stato di caricamento.

5 Stato dei Requisiti Funzionali

5.1 Stato per requisito

Codice	Descrizione	Stato
RF1-OB	L'Utente deve potersi autenticare presso il Sistema	Soddisfatto
RF2-OB	L'Utente deve inserire il proprio username per autenticarsi presso il Sistema	Soddisfatto
RF3-OB	L'Utente deve inserire la propria password per autenticarsi presso il Sistema	Soddisfatto
RF4-OB	L'Utente deve ricevere un errore se l'username non è registrato nel Sistema o la password non è corretta per l'Utente inserito	Soddisfatto
RF5-OB	L'Utente deve potersi autenticare presso il Sistema e impostare una nuova password	Soddisfatto
RF6-OB	L'Utente deve inserire la password temporanea che gli è stata fornita dall'amministratore per autenticarsi presso il Sistema e impostare una nuova password	Soddisfatto
RF7-OB	L'Utente deve inserire la nuova password per registrarsi presso il Sistema	Soddisfatto
RF8-OB	L'Utente deve ricevere un errore se l'username non è registrato nel Sistema o la password temporanea non è corretta per l'Utente inserito	Soddisfatto
RF9-OB	L'Utente deve ricevere un errore se la nuova password è uguale alla password temporanea	Soddisfatto
RF10-OB	L'Utente deve ricevere un errore se la nuova password non rispetta i criteri richiesti	Soddisfatto
RF11-OB	L'Amministratore deve poter visualizzare se un account MyVimar è configurato nel Sistema	Soddisfatto
RF12-OB	L'Amministratore visualizzando l'account MyVimar collegato deve visualizzare l'email dell'account	Soddisfatto
RF13-OB	L'Amministratore deve poter collegare un account MyVimar nel Sistema	Soddisfatto
RF14-OB	L'Amministratore deve poter rimuovere un account MyVimar dal Sistema	Soddisfatto
RF15-DE	L'Amministratore deve poter visualizzare l'elenco degli utenti del Sistema	Soddisfatto
RF16-OB	L'Amministratore visualizzando l'elenco degli utenti del sistema deve visualizzare un utente del Sistema nel dettaglio	Soddisfatto
RF17-OB	L'Amministratore visualizzando un utente del sistema nel dettaglio deve visualizzare il nome	Soddisfatto
RF18-OB	L'Amministratore visualizzando un utente del sistema nel dettaglio deve visualizzare il cognome	Soddisfatto
RF19-OB	L'Amministratore visualizzando un utente del sistema nel dettaglio deve visualizzare l'username	Soddisfatto
RF20-DE	L'Amministratore deve poter creare un utente Operatore Sanitario	Soddisfatto
RF21-DE	L'Amministratore deve inserire il nome dell'Operatore Sanitario per creare un utente Operatore Sanitario	Soddisfatto
RF22-DE	L'Amministratore deve inserire il cognome dell'Operatore Sanitario per creare un utente Operatore Sanitario	Soddisfatto

Codice	Descrizione	Stato
RF23-DE	L'Amministratore deve inserire un username per creare un utente Operatore Sanitario	Soddisfatto
RF24-DE	L'Amministratore deve generare la password temporanea per creare un utente Operatore Sanitario	Soddisfatto
RF25-DE	L'Amministratore deve ricevere un errore se l'username è già in uso	Soddisfatto
RF26-DE	L'Amministratore deve poter eliminare un utente Operatore Sanitario	Soddisfatto
RF27-OB	L'Amministratore deve poter visualizzare tutti i reparti del Sistema	Soddisfatto
RF28-OB	L'Amministratore deve poter visualizzare un reparto del Sistema nel dettaglio	Soddisfatto
RF29-OB	L'Amministratore visualizzando un reparto del Sistema deve poter visualizzare il nome del reparto	Soddisfatto
RF30-OB	L'Amministratore deve poter creare un nuovo reparto nel Sistema	Soddisfatto
RF31-OB	L'Amministratore deve inserire il nome del nuovo reparto per crearlo nel Sistema	Soddisfatto
RF32-OB	L'Amministratore deve ricevere un errore se il nome del nuovo reparto è già in uso	Soddisfatto
RF33-OB	L'Amministratore deve poter modificare il nome di un reparto del Sistema	Soddisfatto
RF34-OB	L'Amministratore deve poter eliminare un reparto del Sistema	Soddisfatto
RF35-OB	L'Amministratore deve poter assegnare un reparto a un utente Operatore Sanitario	Soddisfatto
RF36-OB	L'Amministratore deve selezionare un utente Operatore Sanitario per assegnare a un utente Operatore Sanitario un reparto	Soddisfatto
RF37-OB	L'Amministratore deve selezionare un reparto per assegnare un reparto a un utente Operatore Sanitario	Soddisfatto
RF38-OB	L'Amministratore deve poter eliminare l'assegnazione utente Operatore Sanitario – reparto	Soddisfatto
RF39-OB	L'Amministratore deve poter assegnare un reparto a un appartamento	Soddisfatto
RF40-OB	L'Amministratore deve selezionare un appartamento per assegnare a un appartamento un reparto	Soddisfatto
RF41-OB	L'Amministratore deve selezionare un reparto per assegnare un reparto a un appartamento	Soddisfatto
RF42-OB	L'Amministratore deve poter eliminare l'assegnazione appartamento – reparto	Soddisfatto
RF43-OB	L'Utente deve poter visualizzare una dashboard che rappresenta lo stato del Sistema	Soddisfatto
RF44-OB	L'Utente visualizzando la dashboard deve poter visualizzare un modulo gestione allarmi	Soddisfatto
RF45-OB	L'Utente visualizzando il modulo gestione allarmi deve poter visualizzare ogni singolo allarme	Soddisfatto
RF46-OB	L'Utente visualizzando un singolo allarme deve poter visualizzare il segnale di pericolo	Soddisfatto
RF47-OB	L'Utente visualizzando un singolo allarme deve poter visualizzare il nome	Soddisfatto

Codice	Descrizione	Stato
RF48-OB	L'Utente visualizzando un singolo allarme deve poter visualizzare il tempo trascorso dallo scatto dell'allarme	Soddisfatto
RF49-OB	L'Utente visualizzando la dashboard deve poter visualizzare un modulo statistiche allarmi	Soddisfatto
RF50-OB	L'Utente visualizzando il modulo statistiche allarmi deve poter visualizzare il numero di allarmi risolti	Soddisfatto
RF51-OB	L'Utente visualizzando il modulo statistiche allarmi deve poter visualizzare il numero di allarmi attivi	Soddisfatto
RF52-OB	L'Utente visualizzando la dashboard deve poter visualizzare un modulo informazioni Utente	Soddisfatto
RF53-OB	L'Utente visualizzando il modulo informazioni Utente deve poter visualizzare il nome Utente	Soddisfatto
RF54-OB	L'Utente visualizzando il modulo informazioni Utente deve poter visualizzare il cognome Utente	Soddisfatto
RF55-OB	L'Utente visualizzando la dashboard deve poter visualizzare un modulo analisi clima	Soddisfatto
RF56-OB	L'Utente visualizzando la dashboard deve poter visualizzare un modulo analisi consumi	Soddisfatto
RF57-OB	L'Utente visualizzando la dashboard deve poter visualizzare un modulo analisi presenze	Soddisfatto
RF58-DE	L'Amministratore deve poter aggiungere alla visualizzazione della dashboard un modulo tra i seguenti: statistiche allarmi; informazioni Utente; analisi clima; analisi consumi	Soddisfatto
RF59-DE	L'Amministratore deve poter rimuovere dalla visualizzazione della dashboard un modulo tra i seguenti: statistiche allarmi; informazioni Utente; analisi clima; analisi consumi	Soddisfatto
RF60-OB	L'Utente deve poter risolvere un allarme	Soddisfatto
RF61-OB	L'Utente deve poter visualizzare le analytics	Soddisfatto
RF62-OB	L'Utente visualizzando le analytics deve visualizzare l'elenco dei suggerimenti risparmio energetico	Soddisfatto
RF63-OB	L'Utente visualizzando l'elenco dei suggerimenti risparmio energetico deve visualizzare un suggerimento risparmio energetico	Soddisfatto
RF64-OB	L'Utente visualizzando le analytics deve visualizzare il grafico dedicato al consumo energetico	Soddisfatto
RF65-OB	L'Utente visualizzando le analytics deve visualizzare il grafico dedicato alle anomalie dell'impianto	Soddisfatto
RF66-OB	L'Utente visualizzando le analytics deve visualizzare il grafico relativo al rilevamento di presenza	Soddisfatto
RF67-OB	L'Utente visualizzando le analytics deve visualizzare il grafico relativo alla presenza prolungata nello stesso ambiente	Soddisfatto
RF68-OB	L'Utente visualizzando le analytics deve visualizzare il grafico relativo alle variazioni di temperatura	Soddisfatto
RF69-OB	L'Utente visualizzando le analytics deve visualizzare il grafico relativo agli allarmi inviati e risolti	Soddisfatto
RF70-OP	L'Utente visualizzando le analytics deve visualizzare il grafico relativo alla frequenza degli allarmi	Soddisfatto
RF71-OP	L'Utente visualizzando le analytics deve visualizzare il grafico relativo alla frequenza delle cadute	Soddisfatto

Codice	Descrizione	Stato
RF72-OB	L'Utente deve poter visualizzare un appartamento del Sistema	Soddisfatto
RF73-OB	L'Utente visualizzando un appartamento del Sistema deve visualizzare il nome dell'appartamento	Soddisfatto
RF74-OP	L'Utente visualizzando un appartamento del Sistema deve visualizzare la mappa degli allarmi dell'appartamento	Soddisfatto
RF75-OB	L'Utente visualizzando un appartamento del Sistema deve visualizzare le stanze dell'appartamento	Soddisfatto
RF76-OB	L'Utente deve visualizzare una stanza nel dettaglio	Soddisfatto
RF77-OB	L'Utente visualizzando una stanza deve visualizzare il nome della stanza	Soddisfatto
RF78-OB	L'Utente visualizzando una stanza deve visualizzare l'elenco dei dispositivi della stanza	Soddisfatto
RF79-OB	L'Utente visualizzando l'elenco dei dispositivi deve visualizzare un singolo dispositivo di tipo: termostato; sensore di caduta; sensore di presenza; punto luce; pulsante di allarme; porta di ingresso; tapparella	Soddisfatto
RF80-OB	L'Utente visualizzando un dispositivo deve vedere il nome	Soddisfatto
RF81-OB	L'Utente visualizzando un dispositivo deve vedere lo stato	Soddisfatto
RF82-OP	L'Utente visualizzando un dispositivo deve vedere le azioni eseguibili	Soddisfatto
RF83-OP	L'Amministratore deve poter abilitare un appartamento nel Sistema	Non soddisfatto
RF84-OP	L'Amministratore deve poter disabilitare un appartamento nel Sistema	Non soddisfatto
RF85-OB	L'Amministratore deve poter creare un allarme nel Sistema	Soddisfatto
RF86-OB	L'Amministratore deve selezionare l'appartamento per creare un allarme nel Sistema	Soddisfatto
RF87-OB	L'Amministratore deve selezionare il sensore per creare un allarme nel Sistema	Soddisfatto
RF88-OB	L'Amministratore deve selezionare il livello di priorità tra: priorità 1 (bianco); priorità 2 (verde); priorità 3 (arancione); priorità 4 (rosso) per creare un allarme nel Sistema	Soddisfatto
RF89-OB	L'Amministratore deve selezionare una soglia di intervento per creare un allarme nel Sistema	Soddisfatto
RF90-OB	L'Amministratore deve selezionare un orario di attivazione per creare un allarme nel Sistema	Soddisfatto
RF91-OB	L'Amministratore deve selezionare un orario di disattivazione per creare un allarme nel Sistema	Soddisfatto
RF92-OB	L'Amministratore deve ricevere un errore se non ha selezionato alcun sensore	Soddisfatto
RF93-OB	L'Amministratore deve ricevere un errore se non ha selezionato alcun livello di priorità	Soddisfatto
RF94-OB	L'Amministratore deve ricevere un errore se non ha selezionato alcuna soglia di intervento	Soddisfatto
RF95-OB	L'Amministratore deve poter modificare la priorità di un allarme del Sistema	Soddisfatto
RF96-OB	L'Amministratore deve poter modificare la soglia d'intervento di un allarme del Sistema	Soddisfatto

Codice	Descrizione	Stato
RF97-OB	L'Amministratore deve poter modificare l'orario di attivazione di un allarme del Sistema	Soddisfatto
RF98-OB	L'Amministratore deve poter modificare l'orario di disattivazione di un allarme del Sistema	Soddisfatto
RF99-OB	L'Amministratore deve poter abilitare un allarme del Sistema	Soddisfatto
RF100-OB	L'Amministratore deve poter disabilitare un allarme del Sistema	Soddisfatto
RF101-OB	L'Amministratore deve poter eliminare un allarme nel Sistema	Soddisfatto
RF102-OP	L'Utente deve poter visualizzare le notifiche inviate dal Sistema	Soddisfatto
RF103-OP	L'Utente visualizzando le notifiche deve poter visualizzare una notifica nel dettaglio	Soddisfatto
RF104-OP	L'Utente visualizzando una notifica deve poter visualizzare il titolo della notifica	Soddisfatto
RF105-OP	L'Utente visualizzando una notifica deve poter visualizzare il tempo trascorso dall'invio della notifica	Soddisfatto

5.2 Grafici riassuntivi

A seguire vengono presentati i grafici riassuntivi del soddisfacimento dei requisiti funzionali, suddivisi per tipologia di classificazione: obbligatori (OB), desiderabili (DE) e opzionali (OP). Per ciascuna categoria viene mostrata la percentuale di requisiti soddisfatti rispetto al totale previsto, al fine di fornire una visione immediata e sintetica del grado di completamento del Sistema rispetto a quanto definito nell'Analisi dei Requisiti.

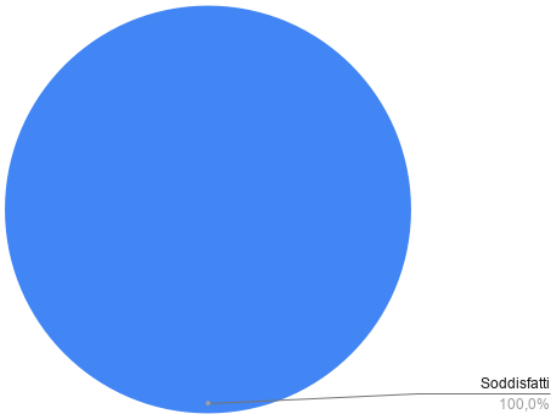

Figura 604: Percentuale di Requisiti Obbligatori Soddisfatti rispetto al totale

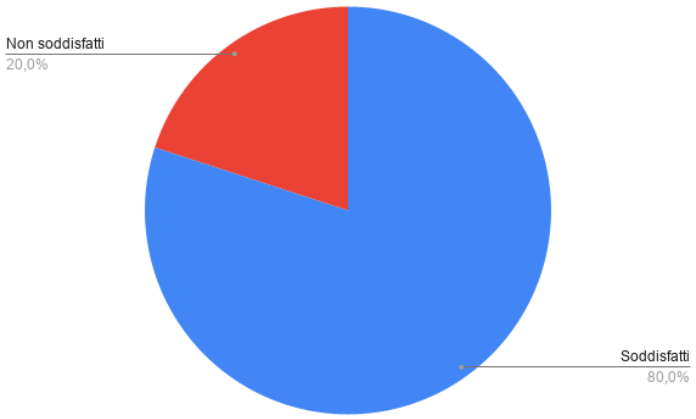

Figura 605: Percentuale di Requisiti Opzionali Soddisfatti rispetto al totale

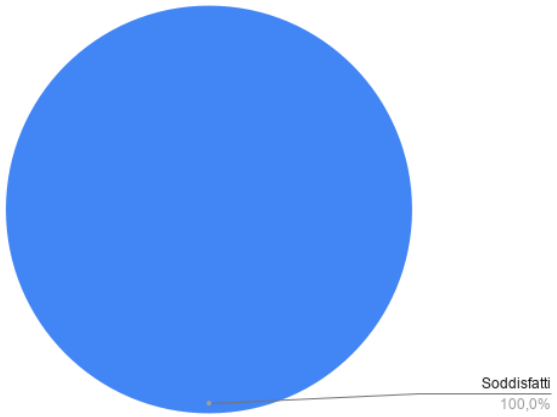

Figura 606: Percentuale di Requisiti Desiderabili Soddisfatti rispetto al totale