

**Università degli Studi di Salerno**

**Corso di Ingegneria del Software**

**AstroMark**  
**Object Design**  
**Versione 1.1**



Data: 30/01/2025

Progetto: AstroMark	Versione: 1.1
Documento: Object Design	Data: 30/01/2025

**Partecipanti:**

Nome	Matricola
Giuseppe Cavallaro	0512116926
Mario Cosenza	0512116320
Mario Fasolino	0512116965
Giulio Sacrestano	0512116812

<b>Scritto da:</b>	Giuseppe Cavallaro
	Mario Cosenza
	Mario Fasolino
	Giulio Sacrestano

**Revision History**

Data	Versione	Descrizione	Autore
15/12/2024	1.0	Inizio stesura ODD, con specifica delle interfacce delle classi Spring e dei Componenti React	Giuseppe Cavallaro, Mario Cosenza, Mario Fasolino, Giulio Sacrestano
30/01/2025	1.1	Riscrittura delle interfacce e aggiornamento Componenti React	Mario Cosenza, Mario Fasolino, Giulio Sacrestano

## Indice

1.	Introduzione .....	5
1.1.	Object design trade-offs .....	5
1.2.	Linee guida documentazione di interfaccia .....	7
1.3.	Design Pattern .....	11
1.4.	Definizioni, acronimi e abbreviazioni .....	13
1.5.	Riferimenti .....	14
2.	Packages .....	15
2.1	Back-End .....	15
2.1.1	Elenco Classi e Interfacce di tipo Service e Controller .....	21
2.1.2	Elenco Classi e Interfacce di tipo Entity e JpaRepository .....	26
2.2	Front-End .....	27
4.	Interfaccia Classi .....	29
3.1	Interfacce .....	29
3.1.1	CrudService<T, R, RS, ID> .....	30
3.1.2	AuthenticationService .....	31
3.1.3	SchoolUserService .....	35
3.1.4	StudentService .....	40
3.1.5	ParentService extends CrudService .....	42
3.1.6	SecretaryService extends CrudService .....	43
3.1.7	TeacherService extends CrudService .....	43
3.1.8	SchoolService extends CrudService .....	44
3.1.9	ReceptionAgendaService .....	45
3.1.10	ClassAgendaService .....	51
3.1.11	MarkService .....	55
3.1.12	NoteService extends CrudService .....	59
3.1.13	JustifiableService .....	60
3.1.14	AttendanceService .....	62
3.1.15	ClassworkService .....	64
3.1.16	CommunicationService extends CrudService .....	66
3.1.17	MessageService .....	67
3.1.18	HomeworkChatService .....	68
3.1.19	TicketChatService .....	73
3.1.20	ClassManagementService .....	77

3.1.21	OrientationService .....	83
3.1.22	FileService .....	83
3.	Specifica DTO e Documentazione REST .....	84
3.1.	Componenti React.....	85
4.	Glossario.....	90

# 1. Introduzione

Questo documento di **Object Design** descrive l'architettura logica e fisica del sistema **AstroMark**, delineando la struttura delle principali classi, interfacce e componenti che costituiscono l'applicazione. L'obiettivo è fornire una guida chiara per l'implementazione, mantenendo un equilibrio tra modularità, scalabilità e facilità di manutenzione. Il sistema è stato progettato seguendo un approccio orientato agli oggetti, che enfatizza la separazione delle responsabilità tra i diversi componenti e promuove la riusabilità del codice. Il documento illustra le decisioni progettuali chiave e i trade-off considerati, dettagliando le scelte tecnologiche che soddisfano i requisiti funzionali e non funzionali. Sono inclusi anche i vincoli e le linee guida per garantire la coerenza dell'implementazione e l'aderenza agli standard di settore.

## 1.1. Object design trade-offs

Il design degli oggetti per il sistema **AstroMark** richiede un'attenta valutazione dei compromessi tra diverse priorità progettuali, che includono scalabilità, affidabilità, usabilità, prestazioni e compatibilità. Ogni decisione di design è stata guidata da requisiti funzionali e non funzionali, cercando di bilanciare le esigenze operative con i vincoli economici e tecnici. Questo paragrafo analizza i principali **trade-off** considerati durante la fase di progettazione, evidenziando come le scelte adottate abbiano influito sull'architettura del sistema, con l'obiettivo di garantire un equilibrio ottimale tra flessibilità, efficienza e sostenibilità nel lungo periodo.

### Reliability vs Cost-Effectiveness

L'affidabilità è una priorità centrale per il sistema **AstroMark**, che deve garantire continuità operativa anche in presenza di un carico massimo di 1500 utenti contemporanei. Per soddisfare questo requisito, l'architettura include infrastrutture cloud scalabili, bilanciamento del carico e strumenti di monitoraggio delle prestazioni in tempo reale. Queste scelte, pur aumentando significativamente la resilienza e la stabilità del sistema, comportano costi operativi elevati, sia per l'infrastruttura che per la manutenzione. Per mitigare questi costi, il sistema è stato progettato con un approccio scalabile che consente configurazioni iniziali più economiche, come server condivisi o hosting di base. Sebbene ciò possa comportare una riduzione temporanea dell'affidabilità, la struttura del sistema consente un'espansione rapida e graduale verso configurazioni più performanti. Questo compromesso garantisce un equilibrio tra affidabilità e sostenibilità economica, consentendo un adattamento alle esigenze operative man mano che la piattaforma evolve.

	Ingegneria del Software	Pagina 5 di 92
--	-------------------------	----------------

## Scalability vs Rapid Development

La scalabilità è stata ottenuta adottando un'architettura REST stateless, con autenticazione basata su token JWT. Questo approccio permette una gestione efficiente delle sessioni, facilitando l'espansione orizzontale e verticale del sistema. Tuttavia, l'implementazione di un'autenticazione JWT comporta una maggiore complessità rispetto ai tradizionali meccanismi basati su sessione e cookie, richiedendo una gestione precisa dei token e delle loro validità. Per bilanciare la necessità di scalabilità con la rapidità di sviluppo, il sistema è stato progettato in modo iterativo, permettendo di affrontare le complessità dell'autenticazione stateless senza rallentare significativamente il rilascio iniziale. Questo compromesso garantisce che il sistema sia pronto per un'eventuale crescita, mantenendo una base solida per ulteriori miglioramenti.

## Reliability vs Performance

L'affidabilità è un requisito essenziale per il sistema **AstroMark**, che deve garantire stabilità e continuità operativa anche in condizioni di carico elevato. Per raggiungere questo obiettivo, vengono adottate soluzioni come il monitoraggio continuo, il bilanciamento del carico e meccanismi di failover, che assicurano il recupero rapido in caso di guasti. Tuttavia, queste scelte possono introdurre un leggero overhead nelle prestazioni, poiché funzionalità, come la replica dei dati e la registrazione dettagliata dei log, richiedono risorse aggiuntive.

Un design orientato esclusivamente alle prestazioni potrebbe sacrificare queste misure di sicurezza per ottimizzare la velocità e ridurre la latenza delle operazioni. Per bilanciare questi aspetti, sono stati implementati meccanismi di caching e ottimizzazioni delle query, mantenendo un livello accettabile di affidabilità, senza compromettere significativamente i tempi di risposta.

## Reusability vs Simplicity

La riusabilità dei componenti è un principio chiave del sistema **AstroMark**, volto a garantire una manutenzione efficiente e a ridurre la duplicazione del codice. La progettazione di componenti riutilizzabili, come moduli per form generici e viste configurabili, consente di risparmiare tempo durante lo sviluppo e di mantenere coerenza tra le diverse funzionalità del sistema. Tuttavia, questa scelta aumenta la complessità del design, poiché i componenti devono essere sufficientemente flessibili per adattarsi a molteplici contesti.

## 1.2. Linee guida documentazione di interfaccia

Per garantire coerenza, manutenibilità e qualità del codice nell'applicazione **AstroMark**, che utilizza **Spring Boot** per il back-end e **React** con **TypeScript** per il front-end, si seguono le seguenti linee guida per la specifica delle interfacce.

Queste linee guida sono basate sui principali standard di stile e best practices, assicurando un design uniforme e l'adozione di pattern di progettazione efficaci.

### Convenzioni di Nomenclatura

#### Camel Case:

##### Java:

- Utilizzare **camelCase** per nomi di variabili, metodi e proprietà.
- Utilizzare **PascalCase** per nomi di classi e interfacce.

##### TypeScript/React:

- Adottare **camelCase** per variabili e funzioni.
- Utilizzare **PascalCase** per componenti React e interfacce.

#### Consistenza:

- Mantenere una nomenclatura coerente tra back-end e front-end per facilitare la comprensione e la manutenzione del codice.
- Preferire nomi descrittivi e chiari che riflettano lo scopo e la funzionalità dell'elemento, evitando abbreviazioni non standard.

### Naming Convention per DTO, Controller e Service

#### Controller:

##### Java:

- I nomi dei controller devono terminare con Controller per indicare chiaramente il loro ruolo.
- Utilizzare **PascalCase** per i nomi delle classi.

**Esempio:** StudentController, CourseController, EnrollmentController

##### React:

- Non esistono controller nel front-end React, ma per coerenza, i componenti che fungono da controller (gestori della logica) dovrebbero seguire una convenzione simile, terminando con Container.
- Utilizzare **PascalCase** per i nomi dei componenti.

**Esempio:** StudentContainer, CourseContainer, EnrollmentContainer

## Service:

### Java:

- I nomi dei servizi devono terminare con Service per chiarire la loro funzione di gestione della logica di business.
- Utilizzare **PascalCase** per i nomi delle classi.

**Esempio:** StudentService, CourseService, EnrollmentService

### TypeScript:

- Nei servizi front-end, utilizzare il suffisso Service per mantenere la coerenza con il back-end.
- Utilizzare **PascalCase** per le classi o moduli di servizio.

**Esempio:** StudentService, CourseService, EnrollmentService

## Definizione delle Interfacce

### Chiarezza e Semplicità:

- Le interfacce devono definire contratti chiari e comprensibili tra le diverse componenti del sistema, evitando complessità inutili.
- Utilizzare nomi che descrivano esattamente il ruolo e le responsabilità dell'interfaccia.

### Separazione delle Responsabilità:

- Ogni interfaccia dovrebbe avere una singola responsabilità, facilitando l'implementazione e il testing.
- Evitare interfacce sovraccariche che gestiscono troppe funzionalità diverse.

## Stile delle Parentesi

### Posizionamento delle Parentesi:

#### Java:

- Aprire la parentesi graffa "{" alla fine della dichiarazione della classe, metodo o blocco di controllo, sulla stessa linea.
- Chiudere la parentesi graffa "}" su una nuova linea.

#### TypeScript/React:

- Seguire lo stesso stile utilizzato in Java per mantenere la coerenza.
- Aprire la parentesi graffa "{" alla fine della dichiarazione della funzione o componente, sulla stessa linea.
- Chiudere la parentesi graffa "}" su una nuova linea.



### Spaziatura:

- Inserire uno spazio tra il nome del metodo e l'apertura della parentesi (.
- Non inserire spazi all'interno delle parentesi stesse.

### Esempi:

- **Corretto:** `getStudentById(Long id)`
- **Errato:** `getStudentById( Long id)`

## Struttura del Progetto

### Organizzazione Modulare:

#### Java:

- Strutturare il codice in pacchetti coerenti, separando le diverse responsabilità (es. controller, service, repository, dto, model).

#### React:

- Organizzare i componenti in cartelle basate sulle funzionalità o sulle pagine, distinguendo tra componenti presentazionali e container.

## Consistenza e Standardizzazione

### Aderenza agli Style Guides:

- **Java:** Seguire le linee guida del [Google Java Style Guide](#) per mantenere uno stile di codice coerente.
- **TypeScript:** Adottare le pratiche suggerite nel [TypeScript Style Guide](#) per garantire una codifica uniforme.
- **React:** Implementare le raccomandazioni del [React Style Guide](#) per sviluppare componenti React chiari e mantenibili.
- **Spring Framework:** Allinearsi alle [Code Style](#) del Spring Framework per assicurare coerenza nelle implementazioni.

### Documentazione:

- Documentare le interfacce e le componenti utilizzando **Javadoc** per Java e **JSDoc** per TypeScript, fornendo descrizioni chiare delle funzionalità e delle responsabilità.

## Utilizzo di Lombok

### Riduzione del Boilerplate:

- Sfruttare **Lombok** per generare automaticamente getter, setter, costruttori, `toString()`, `equals()` e `hashCode()`, riducendo la quantità di codice ripetitivo e migliorando la leggibilità.
- Utilizzare annotazioni Lombok come `@Data`, `@Getter`, `@Setter`, `@Builder` per semplificare la definizione delle classi DTO e dei modelli di dati.

## Gestione degli Errori e Validazioni

### Java:

- Implementare una gestione globale degli errori utilizzando `@ControllerAdvice` e `@ExceptionHandler` per centralizzare il trattamento delle eccezioni.
- Utilizzare annotazioni di validazione come `@NotNull`, `@Size` per garantire l'integrità dei dati.

### React:

- Gestire gli errori a livello di componenti utilizzando state e props.
- Utilizzare librerie di validazione come **Formik** e **Yup** per gestire le validazioni dei form in modo efficiente.

### 1.3. Design Pattern

Nel progetto Astromark, implementato in Spring Boot, sono utilizzati diversi design pattern per strutturare il codice in modo modulare, riutilizzabile e facilmente manutenibile. I design pattern aiutano a risolvere problemi comuni di progettazione software e a semplificare la gestione di complessità. Tra questi, il Singleton garantisce un'unica istanza globale per risorse condivise, il Facade semplifica l'accesso a sistemi complessi, e l'Adapter consente l'integrazione di componenti con interfacce incompatibili. Altri pattern, come il Bridge e il Builder, offrono flessibilità nella gestione delle implementazioni e nella costruzione di oggetti complessi. Inoltre, pattern come l'Abstract Factory e il Chain of Responsibility consentono di gestire la creazione di oggetti e il flusso delle richieste in modo efficiente, mentre il DTO e il DAO separano la logica di business dalla gestione dei dati, semplificando l'interazione tra i vari livelli dell'applicazione.

#### Singleton

Un oggetto creato in un'unica istanza globale e condivisa, utile quando è necessario un punto di accesso unificato a una risorsa. In Spring, i bean per default sono singleton, garantendo che i servizi condivisi come i DataSource vengano istanziati una sola volta.

#### Facade

Fornisce un'interfaccia semplificata per un insieme complesso di classi o funzionalità, agevolando l'uso di sistemi complessi. In Spring, i servizi possono fungere da facciata verso i repository e le altre logiche, offrendo un unico punto di accesso alle operazioni sul dominio. In React invece una componente raccogliere dati da diverse API e organizzarli per la presentazione.

#### Adapter

Permette a classi con interfacce incompatibili di lavorare insieme, convertendo l'interfaccia di una classe in un'altra attesa dal client. In Spring Boot può essere usato per integrare servizi esterni che forniscono dati con formati diversi, adattandoli a DAO o DTO esistenti.

#### Bridge

Separa un'astrazione dalla sua implementazione, permettendo loro di variare indipendentemente. In Spring, si può avere un'astrazione di servizio e varie implementazioni iniettabili tramite bean, facilitando la sostituzione e l'espansione del comportamento.

## Builder

Fornisce un modo flessibile per costruire oggetti complessi passo dopo passo, mantenendo il codice client pulito. In Spring Boot, può essere sfruttato ad esempio per costruire entità o DTO complessi a partire da informazioni parziali, senza incorrere in costruttori enormi. In React, per creare set di proprietà o configurazioni di routing complesse in modo fluido e leggibile.

## Abstract Factory

Fornisce un'interfaccia per creare famiglie di oggetti correlati tra loro, senza specificare le classi concrete. In Spring Boot si possono configurare bean differenti a seconda del profilo attivo.

## Chain of Responsibility

Delega la richiesta lungo una catena di handler, dove ognuno può gestire la richiesta o passarla avanti. In Spring Boot, può essere implementata in filtri per processare richieste HTTP in sequenza.

## DTO (Data Transfer Object)

Strutture dati semplici, senza logica di business, usate per trasferire informazioni tra livelli o servizi. In Spring Boot, i DTO sono comunemente utilizzati nei controller per scambiare dati con il client React, mantenendo separata la logica dal modello di dominio.

## DAO (Data Access Object)

Isola i dettagli di accesso ai dati (query SQL, mapping) all'interno di classi dedicate, semplificando la logica di business. In Spring, i repository (basati su JPA o altri driver) ricalcano il pattern DAO, fornendo un'interfaccia pulita per le operazioni di persistenza

### 1.4. Definizioni, acronimi e abbreviazioni

Di seguito è fornito un elenco degli acronimi, abbreviazioni con le relative definizioni utilizzati in questo documento:

Termine	Definizione
<b>CRUD</b>	Create, Read, Update, Delete – Operazioni di base per la gestione dei dati in un'applicazione.
<b>DTO</b>	Data Transfer Object – Struttura dati utilizzata per trasferire informazioni tra diversi livelli o servizi dell'applicazione.
<b>DAO</b>	Data Access Object – Pattern che isola i dettagli di accesso ai dati, come query SQL e mapping, all'interno di classi dedicate.
<b>HTTP</b>	HyperText Transfer Protocol – Protocollo di trasferimento dati utilizzato per le comunicazioni web.
<b>JPA</b>	Java Persistence API – Specifica Java per la gestione della persistenza dei dati tra le applicazioni Java e i database relazionali.
<b>JSDoc</b>	JavaScript Documentation – Strumento di documentazione per il linguaggio JavaScript, simile a Javadoc per Java.
<b>Javadoc</b>	Strumento di documentazione per il linguaggio Java, utilizzato per generare documentazione API a partire dal codice sorgente.
<b>JWT</b>	JSON Web Token – Standard aperto per la trasmissione sicura di informazioni tra le parti come oggetti JSON.
<b>OCL</b>	Object Constraint Language – Linguaggio utilizzato per specificare restrizioni e vincoli nei modelli UML.
<b>REST</b>	Representational State Transfer – Stile architetturale per la progettazione di servizi web che utilizza le operazioni HTTP.
<b>UML</b>	Unified Modeling Language – Linguaggio di modellazione standardizzato utilizzato per specificare, visualizzare, costruire e documentare gli artefatti di sistemi software.
<b>API</b>	Application Programming Interface – Insieme di regole e specifiche che le applicazioni possono seguire per comunicare tra loro.
<b>CSS</b>	Cascading Style Sheets – Linguaggio utilizzato per descrivere la presentazione di documenti HTML o XML.

## 1.5. Riferimenti

Il presente progetto si basa sull'analisi e il confronto con piattaforme di gestione della didattica già consolidate e affermate nel settore, le quali hanno dimostrato notevole efficacia. Tra queste, un punto di riferimento significativo è rappresentato dalle soluzioni sviluppate da Argo per la gestione della didattica.

Di seguito si presenta un elenco dei documenti chiave del progetto a cui si fa esplicito riferimento:

- **Problem Statement:** Documento che definisce i problemi principali che il progetto intende affrontare e risolvere.
- **System Design Document:** Documento che descrive l'architettura del sistema e le componenti principali del progetto.
- **RAD Requirement Analysis Document:** Documento di analisi dei requisiti che dettaglia le esigenze funzionali e non funzionali del sistema.

Oltre ai documenti del progetto, si fa riferimento ad opere di letteratura tecnica e linee guida che hanno contribuito allo sviluppo metodologico e concettuale di questo lavoro:

- **Object-Oriented Software Engineering Using UML, Patterns, and Java™ Third Edition** di Bernd Bruegge & Allen H. Dutoit.
- **Google Java Style Guide:** Guida completa alle convenzioni di codifica Java di Google, che copre aspetti come la nomenclatura, la formattazione e le best practices per scrivere codice pulito e manutenibile.
- **TypeScript Style Guide:** Linee guida per scrivere codice TypeScript chiaro, consistente e conforme agli standard di settore, focalizzandosi su convenzioni di nomenclatura, strutturazione del codice e pratiche di tipizzazione.
- **React Style Guide:** Raccomandazioni e best practices per lo sviluppo di componenti React, includendo la gestione dello stato, la strutturazione dei componenti e l'ottimizzazione delle performance.
- **Spring Framework Code Style:** Standard di codifica e best practices per lo sviluppo con il framework Spring, comprendenti convenzioni di nomenclatura, formattazione del codice e strutturazione dei progetti per garantire coerenza e qualità.

## 2. Packages

L'applicazione **AstroMark** adotta una struttura di **package** organizzata per garantire una chiara separazione delle responsabilità e facilitare la manutenzione del codice.

### 2.1 Back-End

Nel **back-end**, i package principali includono elementi come i **controller**, che gestiscono le richieste HTTP esposte tramite endpoint REST e delegano la logica di business ai **service**, contenuti in package dedicati. I package **entity** definiscono le tabelle del database utilizzando mappature JPA, rappresentando la struttura dei dati persistenti. Per trasferire dati tra componenti, i **dto** offrono una rappresentazione sicura e ottimizzata, mentre i **repository**, anch'essi organizzati in package, forniscono l'accesso ai dati persistenti tramite Spring Data JPA, supportando operazioni CRUD e query personalizzate. I package **exception** centralizzano la gestione degli errori, mentre i package **config** includono le configurazioni per aspetti come la sicurezza e i servizi di comunicazione. Di seguito sono descritti i principali package individuati:

#### Authentication

Gestisce l'autenticazione e la sicurezza degli accessi al sistema, implementando meccanismi basati su JWT per garantire la scalabilità e la sicurezza delle sessioni.

#### User

Comprende tutte le funzionalità relative alla gestione degli utenti generici del sistema, inclusa la gestione dei dati personali e le interazioni comuni.

#### School

Raccoglie i componenti e le logiche legate alla gestione delle scuole, comprese le informazioni sulle classi e i dettagli relativi agli istituti.

#### Agenda

Focalizzato sulla gestione degli eventi scolastici, come ricevimenti, appuntamenti e attività pianificate, con strumenti per la visualizzazione e la modifica.

#### Attendance

Gestisce le funzionalità relative alla presenza, ritardi e assenze degli studenti, offrendo strumenti per il tracciamento e la giustificazione.

### **Rating**

Si occupa delle funzionalità legate alle valutazioni scolastiche, comprese l'inserimento, la modifica e la visualizzazione dei voti.

### **Classwork**

Riguarda la gestione dei compiti assegnati, dalle assegnazioni alla verifica delle attività svolte.

### **Communication**

Include le funzionalità per la gestione delle comunicazioni tra utenti, come notifiche, avvisi e messaggi diretti.

### **ClassManagement**

Contiene logiche e strumenti per la gestione delle classi, comprese la pianificazione e l'assegnazione di insegnanti e studenti.

### **Chat**

Gestisce le comunicazioni in tempo reale tra utenti tramite una chat interna, supportando anche allegati e notifiche.

### **Orientation**

Fornisce strumenti e risorse per l'orientamento degli studenti, supportando la pianificazione del percorso scolastico e professionale.

### **Commons**

Contiene classi generiche per effettuare operazioni CRUD e classi di configurazione come quelle necessarie per il mapping tra DTO e entità.

### **Behavior**

Contiene logiche e strumenti per la gestione delle note disciplinari.



## *Struttura package backend*

### **authentication**

- **controller**
- **dto**
- **entity**
- **repository**
- **service**
- **utils**

### **user**

#### → **commons**

- **model**
- **mapper**
- **dto**
- **service**
- **controller**

#### → **student**

- **entity**
- **dto**
- **service**
- **repository**
- **controller**

#### → **teacher**

- **entity**
- **dto**
- **service**
- **repository**
- **controller**

#### → **secretary**

- **controller**
- **entity**
- **dto**
- **service**
- **repository**

#### → **parent**

- **controller**

- **entity**
- **dto**
- **service**
- **repository**

#### **school**

- **entity**
- **dto**
- **service**
- **repository**
- **controller**

#### **agenda**

- **commons**
  - **entity**
  - **repository**
  - **mapper**
- **reception**
  - **entity**
  - **dto**
  - **service**
  - **mapper**
  - **repository**
  - **controller**
- **schoolClass**
  - **entity**
  - **dto**
  - **service**
  - **mapper**
  - **repository**
  - **controller**

#### **behavior**

- **entity**
- **dto**
- **service**
- **mapper**
- **repository**

→ controller

#### **attendance**

→ entity

→ dto

→ service

→ mapper

→ repository

→ controller

#### **rating**

→ entity

→ dto

→ service

→ mapper

→ repository

→ controller

#### **classwork**

→ entity

→ dto

→ service

→ mapper

→ repository

→ controller

#### **communication**

→ entity

→ dto

→ service

→ mapper

→ repository

→ controller

#### **classmanagement**

→ entity

→ dto

→ service

→ mapper

→ repository

→ **controller**

→ **didactic**

→ **entity**

→ **repository**

#### **chat**

→ **entity**

→ **dto**

→ **service**

→ **mapper**

→ **repository**

→ **controller**

#### **orientation**

→ **service**

#### **commons**

→ **dto**

→ **service**

→ **configuration**

→ **exception**

→ **security**

### 2.1.1 Elenco Classi e Interfacce di tipo Service e Controller

File	Package	Descrizione
<b>CrudService&lt;T, R, RS, ID&gt;</b>	it.astromark.common.service	Servizio generico per la logica delle operazioni CRUD, integrato con repository e servizi personalizzati.
<b>SchoolUserController</b>	it.astromark.user.common.controller	Controller specifico per la gestione delle operazioni CRUD relative agli utenti, estendibile per esigenze personalizzate.
<b>SchoolUserService</b>	it.astromark.user.common.service	Servizio dedicato alla logica per la gestione degli utenti, supportando operazioni CRUD e funzionalità specifiche.
<b>AuthController</b>	it.astromark.authentication.controller	Controller per la gestione delle operazioni di autenticazione e autorizzazione degli utenti, come login e registrazione.
<b>AuthenticationService</b>	it.astromark..authentication.service	Servizio responsabile della gestione delle operazioni di autenticazione degli utenti, come la verifica delle credenziali e la generazione dei token di accesso.
<b>JWTService</b>	it.astromark.authentication.service	Servizio specializzato nella creazione, validazione e gestione dei JSON Web Token (JWT) utilizzati per l'autenticazione.
<b>StudentController</b>	it.astromark.user.student.controller	Controller per la gestione delle operazioni relative agli studenti, incluse funzionalità specifiche e CRUD.
<b>StudentService</b>	it.astromark.user.student.service	Servizio per la gestione della logica relativa agli studenti, con supporto per operazioni CRUD e funzionalità personalizzate.
<b>SecretaryController</b>	it.astromark.user.secretary.controller	Controller per la gestione delle operazioni relative ai segretari, incluse funzionalità specifiche e CRUD.

<b>SecretaryService</b>	it.astromark.user.secretary.service	Servizio per la gestione della logica relativa alla segreteria, con supporto per operazioni CRUD e funzionalità personalizzate.
<b>ParentController</b>	it.astromark.user.parent.controller	Controller per la gestione delle operazioni relative ai genitori, inclusi CRUD e funzionalità specifiche per l'interazione con gli studenti.
<b>ParentService</b>	it.astromark.user.parent.service	Servizio per la logica relativa ai genitori, gestendo operazioni CRUD e funzionalità specifiche per l'interazione con gli studenti e altre entità.
<b>TeacherController</b>	it.astromark.user.teacher.controller	Controller per la gestione delle operazioni relative agli insegnanti, inclusi CRUD e funzionalità specifiche per l'interazione con gli studenti e il sistema.
<b>TeacherService</b>	it.astromark.user.teacher.service	Servizio per la logica relativa agli insegnanti, gestendo operazioni CRUD e funzionalità personalizzate per l'interazione con gli studenti e le materie.
<b>SchoolController</b>	it.astromark.school.controller	Controller per la gestione delle operazioni relative alle scuole, incluse funzionalità di CRUD e altre operazioni specifiche per la gestione degli istituti.
<b>SchoolService</b>	It.astromark.school.service	Servizio per la gestione della logica della scuola.
<b>ReceptionAgendaController</b>	it.astromark.agenda.reception.controller	Controller per la gestione delle operazioni relative all'agenda dei ricevimenti, incluse funzionalità CRUD e pianificazione degli appuntamenti.
<b>ReceptionAgendaService</b>	it.astromark.agenda.reception.service	Servizio per la gestione della logica dell'agenda dei ricevimenti, supportando operazioni CRUD e pianificazione personalizzata.

<b>ClassAgendaController</b>	it.astromark.agenda.schoolclass.controller	Controller per la gestione delle operazioni relative all'agenda delle classi, incluse funzionalità CRUD e organizzazione degli eventi scolastici.
<b>ClassAgendaService</b>	it.astromark.agenda.schoolClass.service	Servizio per la logica relativa all'agenda delle classi, gestendo operazioni CRUD e organizzazione degli eventi scolastici.
<b>MarkController</b>	it.astromark.mark.controller	Controller per la gestione delle operazioni relative ai voti, incluse funzionalità CRUD e visualizzazione dei risultati degli studenti.
<b>MarkService</b>	it.astromark.mark.service	Servizio per la logica relativa ai voti, gestendo operazioni CRUD e calcoli statistici sui risultati degli studenti.
<b>NoteController</b>	it.astromark.behavior.controller	Controller per la gestione delle operazioni relative alle note disciplinari.
<b>NoteService</b>	it.astromark.behavior.service	Servizio per la gestione della logica delle note disciplinari, supportando operazioni CRUD e analisi comportamentali.
<b>AttendanceController</b>	it.astromark.attendance.controller	Controller generico per la gestione della logica dell'appello della classe.
<b>AttendanceService</b>	it.astromark.attendance.service	Servizio generico per la gestione della logica dell'appello della classe.
<b>JustifiableController</b>	it.astromark.attendance.controller	Controller generico per gestire operazioni su entità giustificabili, come assenze, con supporto a operazioni CRUD e logica personalizzata.
<b>JustifiableService</b>	it.astromark.attendance.service	Servizio generico per la gestione della logica delle entità giustificabili, supportando operazioni CRUD e funzionalità personalizzate per giustificazioni.

<b>ClassworkController</b>	it.astromark.classwork.controller	Controller per la gestione delle operazioni relative ai compiti, incluse funzionalità CRUD e assegnazione degli esercizi agli studenti.
<b>ClassworkService</b>	it.astromark.classwork.service	Servizio per organizzare e gestire compiti, scadenze in ambito scolastico.
<b>CommunicationController</b>	it.astromark.communication.controller	Controller per la gestione delle operazioni relative alla comunicazione tra classe e famiglia.
<b>CommunicationService</b>	it.astromark.communication.service	Servizio per la gestione della logica delle comunicazioni tra la classe e la famiglia, supportando operazioni CRUD e invio di notifiche o aggiornamenti.
<b>ChatUploadController</b>	it.astromark.chat.controller	Permette agli utenti di condividere file durante le conversazioni.
<b>HomeworkChatController</b>	it.astromark.chat.controller	Controller per la gestione delle chat relative ai compiti, inclusi invio e ricezione di messaggi e discussioni tra studenti e insegnanti.
<b>TicketController</b>	it.astromark.chat.controller	Controller per la gestione delle chat relative ai ticket di supporto, inclusi invio e ricezione di messaggi e gestione delle conversazioni per risolvere richieste o problematiche.
<b>WebSocketChatController</b>	it.astromark.chat.controller	Gestisce le connessioni WebSocket per la chat, consentendo la comunicazione in tempo reale tra gli utenti.
<b>MessageService</b>	it.astromark.chat.service	Fornisce la logica di business per la gestione dei messaggi, come l'invio, la ricezione e la memorizzazione dei messaggi.
<b>HomeworkChatService</b>	it.astromark.chat.service	Servizio per la gestione della logica delle chat sui compiti,



		supportando l'invio di messaggi, la discussione tra studenti e insegnanti e la gestione delle conversazioni.
<b>TicketService</b>	it.astromark.chat.service	Servizio per la gestione della logica delle chat sui ticket di supporto, supportando la gestione dei messaggi e la risoluzione delle richieste o problematiche segnalate.
<b>ClassManagementController</b>	it.astromark.classmanagement.controller	Controller per la gestione delle operazioni relative all'amministrazione delle classi, incluse funzionalità CRUD e organizzazione delle informazioni sugli studenti e docenti.
<b>ClassManagementService</b>	it.astromark.classmanagement.service	Servizio per la gestione della logica relativa all'amministrazione delle classi, supportando operazioni CRUD e l'organizzazione delle informazioni su studenti, docenti e orari.
<b>OrientationService</b>	it.astromark.orientation.controller	Servizio per la gestione della logica relativa all'orientamento scolastico, con un focus specifico su corsi o attività legate alla programmazione in Python.
<b>FileService</b>	it.astromark.common.service	Gestisce le operazioni relative ai file, come il caricamento, il download, la modifica e la cancellazione di file.

### 2.1.2 Elenco Classi e Interfacce di tipo Entity e JpaRepository

Le classi JpaRepository saranno inserite nel package repository del livello superiore a quello della corrispondente Entity e per mantenere la consistenza la Repository avrà il nome della Entity con suffisso "Repository", eventuali classi Ebeddable relative agli ID saranno inserite nel package "entity" corrispondente. Per la descrizione completa delle classi riferirsi al RAD.

File	Package
<b>SchoolUser</b>	it.astromark.user.commons.model
<b>Student</b>	it.astromark.user.student.entity
<b>Parent</b>	it.astromark.user.parent.entity
<b>Teacher</b>	it.astromark.user.teacher.entity
<b>Secretary</b>	it.astromark.user.secretary.entity
<b>School</b>	it.astromark.school.entity
<b>Timetable</b>	it.astromark.agenda.commons.entity
<b>RedDate</b>	it.astromark.agenda.commons.entity
<b>ClassTimetable</b>	it.astromark.agenda.schoolclass.entity
<b>ReceptionTimetable</b>	it.astromark.agenda.reception.entity
<b>Timeslot</b>	it.astromark.agenda.commons.entity
<b>ReceptionTimeslot</b>	it.astromark.agenda.reception.entity
<b>ReceptionBooking</b>	it.astromark.agenda.reception.entity
<b>TeachingTimeslot</b>	it.astromark.agenda.schoolclass.entity
<b>SignHour</b>	it.astromark.agenda.schoolclass.entity
<b>ClassActivity</b>	it.astromark.classwork.entity
<b>Homework</b>	it.astromark.classwork.entity
<b>TeacherClass</b>	it.astromark.classmanagement.entity
<b>Teaching</b>	it.astromark.classmanagement.didactic.entity
<b>Subject</b>	it.astromark.classmanagement.didactic.entity
<b>StudyPlan</b>	it.astromark.classmanagement.didactic.entity
<b>SchoolClass</b>	it.astromark.classmanagement.entity
<b>Communication</b>	it.astromark.communication.entity
<b>Chat</b>	it.astromark.chat.entity
<b>Ticket</b>	it.astromark.chat.entity
<b>HomeworkChat</b>	it.astromark.chat.entity
<b>Message</b>	it.astromark.chat.entity
<b>JustifiableEntity</b>	it.astromark.attendance.entity

<b>Delay</b>	it.astromark.attendance.entity
<b>Absence</b>	it.astromark.attendance.entity
<b>Note</b>	it.astromark.behavior.entity
<b>Mark</b>	it.astromark.mark.entity
<b>SemesterReport</b>	it.astromark.mark.model
<b>SemesterReportMark</b>	it.astromark.mark.model

## 2.2 Front-End

Nel **front-end**, la struttura dei package è pensata per supportare la modularità e la riutilizzabilità del codice. I package **components** contengono elementi dell'interfaccia utente, suddivisi tra componenti specifici per i **form**, come campi di input e validazioni, e componenti di interfaccia generale, come pulsanti e modali. Le risorse statiche, come immagini e file CSS, sono organizzate all'interno del package **assets** per una gestione efficiente.

### src

- **assets**
- **components**
  - **route**
- **entities**
- **theme**
- **styles**
- **pages**
  - **parents**
  - **secretary**
  - **studentsParents**
  - **teacher**
- **services**

### 3. Invarianti Classi

Di seguito sono riportate le invarianti, dove necessarie, per le entità individuate in fase di analisi dei requisiti. Non sono previste invarianti per le classi service.

Nome Classe (context)	Invariante
User	birthDate <= Date.now() - 10
Parent	students.forAll(s   s.school = self.school)
Student	students.schoolClasses.forAll(c   c.school = self.school)
SchoolClass	teachers.forAll(t   t.school = self.school)
TeachingTimeslot	teaching.teacher.schoolClasses .includes(classTimetable.schoolClass)
SignedHour	teacher.school = self.teachingTimeslot .classTimetable.schoolClass.school)
ReceptionBooking	parent.strudents.exist(s   s.schoolClasses.exist(c   c.teacherClass.exist(tc   tc.teacher = receptionTimeslot.receptionTimetable.teacher)
Homework	<b>inv:</b> homeworkChats.forAll(s   s.schoolClasses.exist(c   c = homework.signedHour.teachingTimeslot.classTimetable.schoolClass) <b>inv:</b> dueDate > self.signedHour.teachingTimeslot.date
Ticket	(closed = false and soleved = false) or closed = true
SignedHour	timeSign.date >= teachingTimeslot.date
JustificationEntity	(needsJustification = false and justified = true) or (needsJustification = true and justified = false and justificationText.size() = 0) or (needsJustification = true and justified = true)
SemesterReport	(viewed = false and publish = false) or publish = true

## 4. Interfaccia Classi

### 3.1 Interfacce

In questo paragrafo si fa riferimento a degli oggetti **Controller** questi sono dei **@RestController** di **Spring** e dovranno essere decorati con altre annotazioni specifiche in fase di implementazione. **SchoolManager** non sarà implementato nella prime versioni di AstroMark pertanto non è stato realizzato ancora il design. Gli oggetti **Entity**, individuati in fase di analisi dei requisiti, sono stati affinati e alcune operazioni estratte come operazione degli oggetti **@Service** separando la logica di business dalle Entity per la persistenza di **Jakarta EE**. Per molte delle entità verranno realizzati delle classi record **DTO** per ridurre l'utilizzo di banda e nascondere le informazioni in base ai permessi di accesso individuati nel **SDD**.

Tutte i repositories estendono **JpaRepository<T, ID>** e i metodi aggiunti sono delle declared query usate dai **@Service** e quindi spesso mappate direttamente in questo layer.

Per inviare le password di primo accesso verrà usato **SendGrid** e per la memorizzazione degli allegati ai messaggi verrà utilizzato **CloudFlare R2** quindi saranno creati dei Bean di configurazione per entrambi i servizi che verranno implementate seguendo le documentazioni fornite da **AWS** e da **CloudFlare**.

Le classi **TypeScript** utilizzate per l'implementazione del sottosistema di Interfaccia Utente sono omesse perché fortemente legata al layout delle pagine web, in seguito, è comunque riportato un elenco dei componenti. Verrà realizzato un livello **service** anche per il client per ridurre l'accoppiamento tra i componenti e la logica per interrogare l'**API Rest**, la validazione sarà realizzata anche lato **front-end** con il supporto della libreria **yup**, ogni campo sarà validato per rispettare le precondizioni espresse per gli endpoint **REST** e riportate in **OCL** di seguito.

### 3.1.1 CrudService<T, R, RS, ID>

Interfaccia generica per operazioni CRUD (Create, Read, Update, Delete) su un'entità di tipo T, utilizzando un Request DTO di tipo R, un Response DTO di tipo RS e un ID di tipo ID.

Nome	Descrizione	Precondizione	Post condizione
<b>+ create (request: R): RS</b>	Crea una nuova entità nel servizio.	<b>context</b> CrudService::create (request: R): RS  <b>pre</b> self.getByld(request.id) = null	<b>context</b> CrudService::create (request: R): RS  <b>post</b> self.getByld(request.id) <> null <b>post</b> result <> null
<b>+ getByld(id: Id): T</b>	Recupera un'entità esistente tramite l'ID.	<b>context</b> CrudService::getByld(id: Id): T  <b>pre</b> -	<b>context</b> CrudService::getByld(id: Id): T  <b>post</b> getByld(request.id).id = request.id <b>post</b> result <> null
<b>+ update(id: ID, request: R): RS</b>	Aggiorna un'entità esistente nel servizio.	<b>context</b> CrudService::update(id: ID, request: R): RS  <b>pre</b> self.getByld(request.id).id = request.id	<b>context</b> CrudService::update(id: ID, request: R): RS  <b>post</b> result <> null
<b>+ delete(id: ID) : Boolean</b>	Elimina un'entità esistente dal servizio.	<b>Context</b> CrudService::delete(id: ID) : Boolean  <b>pre</b> self.getByld(request.id).id = request.id	<b>context</b> CrudService::delete(id: ID) : Boolean  <b>post</b> self.getByld(request.id).id = null && result = true    result = false

### 3.1.2 AuthenticationService

Servizio per la gestione dell'autenticazione degli utenti.

Nome	Descrizione	Precondizione	Post condizione
<b>+ login</b> (user: UserLoginRequest): SchoolUser	Autentica un utente e gli permette di loggarsi	<b>context</b> AuthenticationService:: login(user: UserLoginRequest): SchoolUser <b>pre</b> user.username().size() >= 5 and user.username().size() <= 256 <b>pre</b> user.password().regExpMatch('^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@\$%^&*~]).{8,}\$') <b>pre</b> user.schoolCode().regExpMatch('SS\\d{5}\$') <b>pre</b> user.role().size() > 0	<b>context</b> AuthenticationService:: login(user: UserLoginRequest): SchoolUser <b>post</b> if not result.isNullUndefined() then result.getUsername() = user.username() and result.getSchoolCode() = user.schoolCode() and result.getRole() = user.role()
<b>+ getUser</b> (id: UUID, role: String) : SchoolUser	Recupera un utente della scuola	<b>context</b> AuthenticationService:: getUser(UUID id, String role) : SchoolUser <b>pre</b> id <> null And (role = "Teacher" or role = "Student" or role = "Parent" or role = "Secretary")	<b>context</b> AuthenticationService:: getUser(UUID id, String role) : SchoolUser <b>post</b> result.getId() = id and result.getRole() = role
<b>+ verify</b> (username: String, password: String, schoolCode: String, role: String): String	Verifica le credenziali di un utente	<b>context</b> AuthenticationService:: verify(username: String, password: String, schoolCode: String, role: String): String	<b>context</b> AuthenticationService:: verify(username: String, password: String, schoolCode: String, role: String): String

		<b>pre</b> not username .ocllsUndefined() and not password .ocllsUndefined() and not schoolCode .ocllsUndefined() and not role.ocllsUndefined()	<b>post</b> result.ocllsTypeOf(String)
<b>+ firstLogin</b> (user: UserLoginRequest): SchoolUser	Gestisce il processo della prima autenticazione di un utente	<b>context</b> AuthenticationService:: firstLogin (user: UserLoginRequest): SchoolUser <b>pre</b> user.username().size() >= 5 and user.username().size() <= 256 <b>pre</b> user.newPassword().regExpMatch('^?(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[#!@#\$%^&*~]).{8,}\$') <b>pre</b> user.schoolCode().regExpMatch('SS\\d{5}\$') <b>pre</b> user.role().size() > 0est user) : SchoolUser	<b>context</b> AuthenticationService:: firstLogin (user: UserLoginRequest): SchoolUser <b>post</b> result.getUsername() = user.username() and result.getSchoolCode() = user.schoolCode() and result.getRole() = user.role() and result.getPendingState() = PendingState.NORMAL
<b>+ getRole(user:</b> SchoolUser): GrantedAuthorirty	Restituisce il ruolo di un utente	<b>context</b> AuthenticationService:: getRole(user: SchoolUser): GrantedAuthorirty <b>pre</b> not user.ocllsUndefined()	<b>context</b> AuthenticationService:: getRole(user: SchoolUser): GrantedAuthorirty <b>post</b> if user.ocllsTypeOf(Parent) then result.getAuthority() = 'ROLE_PARENT'



			<pre> else if user.ocllsTypeOf(Teacher) then result.getAuthority() = 'ROLE_TEACHER' else if user.ocllsTypeOf(Student) then result.getAuthority() = 'ROLE_STUDENT' else if user.ocllsTypeOf(Secretary) then result.getAuthority() = 'ROLE_SECRETARY' endif endif endif endif </pre>
<b>+ isStudent() :</b> Boolean	Restituisce vero se lo studente è loggato, falso altrimenti.	<b>context</b> AuthenticationService:: isStudent() : Student <b>pre</b> -	<b>context</b> AuthenticationService:: isStudent() : Student <b>post</b> -
<b>+ isParent() :</b> Boolean	Restituisce vero se lo studente è loggato, falso altrimenti.	<b>context</b> AuthenticationService:: isParent() : Parent <b>pre</b> -	<b>context</b> AuthenticationService:: isParent() : Parent <b>post</b> -
<b>+isSecretary() :</b> Boolean	Restituisce vero se lo studente è loggato, falso altrimenti.	<b>context</b> AuthenticationService:: isSecretary() : Secretary <b>pre</b> -	<b>context</b> AuthenticationService:: isSecretary() : Secretary <b>post</b> -
<b>+isTeacher() :</b> Boolean	Restituisce vero se lo studente è loggato, falso altrimenti.	<b>context</b> AuthenticationService:: isTeacher() : Teacher <b>pre</b> -	<b>context</b> AuthenticationService:: isTeacher() : Teacher <b>post</b> -

<b>+getParent() : Parent</b>	Restituisce il genitore attualmente loggato.	<b>context</b> AuthenticationService::getParent() :Parent <b>pre</b> -	<b>context</b> AuthenticationService::getParent() :Parent <b>post</b> result = getUser(self.id, "Parent")
<b>+getStudent() : Student</b>	Restituisce lo studente attualmente loggato.	<b>context</b> AuthenticationService::getStudent() : Student <b>pre</b> -	<b>context</b> AuthenticationService::getStudent() : Student <b>post</b> getUser(self.id, "Student")
<b>+getTeacher() : Teacher</b>	Restituisce il docente attualmente loggato.	<b>context</b> AuthenticationService::getTeacher() : Teacher <b>pre</b> -	<b>context</b> AuthenticationService::getTeacher() : Teacher <b>post</b> getUser(self.id, "Teacher")
<b>+getSecretary() : Secretary</b>	Restituisce l'utente della segreteria attualmente loggato.	<b>context</b> AuthenticationService::getSecretary() : Secretary <b>pre</b> -	<b>context</b> AuthenticationService::getSecretary() : Secretary <b>post</b> result = getUser(self.id, "Secretary")

### 3.1.3 SchoolUserService

Servizio per la gestione degli utenti all'interno della scuola. Fornisce specifici per la gestione di informazioni comuni a tutti i tipi di utenti scolastici.

Nome	Descrizione	Precondizione	Post condizione
<b>+ requestRemoval(id: UUID): Boolean</b>	Gestisce la rimozione dell'utente, aggiornando il suo stato.	<b>context</b> UserService:: requestRemoval(id: UUID): Boolean <b>pre</b> -	<b>context</b> UserService:: requestRemoval(id: UUID): Boolean <b>post</b> result.pendingState = "REMOVE"
<b>+isStudentParent(parent: Parent, studentId: UUID): Boolean</b>	Controlla se uno specifico genitore è padre di un dato studente	<b>context</b> SchoolUserService::isStudentParent(parent: Parent, studentId: UUID): Boolean <b>pre</b> not parent.ocIsUndefined() and not studentId.ocIsUndefined()	<b>context</b> SchoolUserService::isStudentParent(parent: Parent, studentId: UUID): Boolean <b>post</b> result = parent.getStudents()->exists(s   s.getId() = studentId)
<b>+isLoggedUserParent(studentId: UUID): Boolean</b>	Controlla se l'utente loggato è il padre di un dato studente	<b>context</b> SchoolUserService:: isLoggedUserParent(studentId: UUID): Boolean <b>pre</b> not studentId .ocIsUndefined()	<b>context</b> SchoolUserService:: isLoggedUserParent(studentId: UUID): Boolean <b>post</b> if not AuthenticationService:: isStudent() then result = true else let parent: Parent = AuthenticationService:: getParent() in result = parent .getStudents()->exists(s   s.getId() = studentId) endif

<b>+isLoggedTeacherClass</b> (classId: Integer): Boolean	Controlla se l'insegnante loggato è il responsabile di una data classe	<b>context</b> SchoolUserService::isLoggedTeacherClass(classId: Integer): Boolean  <b>pre</b> not teacher .ocllsUndefined() and not classId .ocllsUndefined()	<b>context</b> SchoolUserService::isLoggedTeacherClass(classId: Integer): Boolean  <b>post</b> -
<b>+isTeacherClass</b> (teacher: Teacher, classId: Integer): Boolean	Controlla se uno specifico insegnante insegna in una classe	<b>context</b> SchoolUserService::isTeacherClass(teacher: Teacher, classId: Integer): Boolean  <b>pre</b> not teacher .ocllsUndefined() and not classId .ocllsUndefined()	<b>context</b> SchoolUserService::isTeacherClass(teacher: Teacher, classId: Integer): Boolean  <b>post</b> -
<b>+isLoggedParentStudentClass</b> (classId: Integer): Boolean	Controlla se se il genitore loggato ha uno student nella classe	<b>context</b> SchoolUserService::isLoggedParentStudentClass(classId: Integer): Boolean  <b>pre</b> not classId .ocllsUndefined()	<b>context</b> SchoolUserService::isLoggedParentStudentClass(classId: Integer): Boolean  <b>post</b> if AuthenticationService::isParent() then let parent: Parent = AuthenticationService::getParent() in result = parent.getStudents()->exists(s   s.getSchoolClasses()->exists(c   c.getId() = classId) ) else result = true endif
<b>+updatePreferences</b> (schoolUserUpdate: SchoolUserUpdate):	Aggiorna la password di un utente della scuola	<b>context</b> SchoolUserService::updatePreferences(schoolUserUpdate)	<b>context</b> SchoolUserService::updatePreferences(schoolUserUpdate)

SchoolUserResponse		IUserUpdate: SchoolUserUpdate): SchoolUserResponse <b>pre</b> not schoolUserUpdate .ocllsUndefined() and not schoolUserUpdate .password() .ocllsUndefined() and schoolUserUpdate .password().size() >= 8 and schoolUserUpdate .password().size() <= 512	IUserUpdate: SchoolUserUpdate): SchoolUserResponse <b>post</b> let initialUser: SchoolUser = if AuthenticationService ::isStudent() then AuthenticationService e::getStudent() else if AuthenticationService ::isParent() then AuthenticationService ::getParent() else if AuthenticationService ::isTeacher() then AuthenticationService ::getTeacher() else AuthenticationService ::getSecretary() endif endif endif in result.getId() = initialUser.getId() and result.getPassword() = PasswordUtils::hashPass word(schoolUserUpdate.p assword())
+updateAddress(addr ess: String): SchoolUserResponse	Aggiorna l'indirizzo dell'utente loggato	<b>context</b> SchoolUserService:: updateAddress(address: String): SchoolUserResponse	<b>context</b> SchoolUserService:: updateAddress(address: String): SchoolUserResponse

		<b>pre</b> not address.isNullUndefined() and address.size() >= 5	<b>post</b> let initialUser: SchoolUser = if AuthenticationService ::isStudent() then AuthenticationService:: getStudent() else if AuthenticationService ::isParent() then AuthenticationService ::getParent() else if AuthenticationService ::isTeacher() then AuthenticationService ::getTeacher() else AuthenticationService ::getSecretary() endif endif endif in result.getId() = initialUser.getId() and result.getResidentialAddr ess() = address
+isLoggedTeacherStudent(studentId: UUID): Boolean	Contratlla se un insegnate insegna ad uno specifico studente	<b>context</b> SchoolUserService:: isLoggedTeacherStudent(studentId: UUID): Boolean <b>pre</b> not studentId .isNullUndefined()	<b>context</b> SchoolUserService:: isLoggedTeacherStudent(studentId: UUID): Boolean <b>post</b> -
+isLoggedStudent(studentId: UUID): Boolean	Controlla che lo student rispecchi l'id dato	<b>context</b> SchoolUserService:: isLoggedStudent(studentId: UUID): Boolean <b>pre</b> not studentId .isNullUndefined()	<b>context</b> SchoolUserService:: isLoggedStudent(studentId: UUID): Boolean <b>post</b> if not AuthenticationService ::isStudent() then result =

			<pre> true else let student: Student =   AuthenticationService ::getStudent() in result =   student.getId() =   studentId endif </pre>
<pre> +getByldDetailed(): SchoolUserDetailed </pre>	Restituisce tutte le informazioni di uno utente della scuola	<pre> <b>context</b> SchoolUserService:: getByldDetailed(): SchoolUserDetailed <b>pre</b> - </pre>	<pre> <b>context</b> SchoolUserService:: getByldDetailed(): SchoolUserDetailed <b>post</b> let user: SchoolUser = if   AuthenticationService ::isStudent() then   AuthenticationService ::getStudent() else if   AuthenticationService ::isParent() then   AuthenticationService ::getParent() else if   AuthenticationService ::isTeacher() then   AuthenticationService ::getTeacher() else   AuthenticationService ::getSecretary() endif endif in result.getId() = user.getId() </pre>

### 3.1.4 StudentService

Servizio per la gestione degli studenti. Include metodi per la gestione delle informazioni personali.

Nome	Descrizione	Precondizione	Post condizione
<b>+getStudentYears</b> (studentId: UUID): Bag(Integer)	Restituisce una lista di anni in cui è stato in una scuola associati ad uno studente	<b>context</b> StudentService::getStudentYears(studentId: UUID): Bag(Integer) <b>pre</b> not studentId .ocllsUndefined() and SchoolUserService::isLoggedInUserParent(studentId) and AuthenticationService::isStudent()	<b>context</b> StudentService::getStudentYears(studentId: UUID): Bag(Integer) <b>post</b> if not result .ocllsUndefined() then result->forAll(year   year > 0) and result = SchoolUserService::getStudentById(studentId).getSchoolClasses@pre()->collect(c   c.getYear())->asBag() endif
<b>+getSchoolClassByYear</b> (studentId: UUID, year: Year): Bag(SchoolClassResponse)	Recupera la lista di classi di uno student in uno specifico anno	<b>context</b> StudentService::getSchoolClassByYear(studentId: UUID, year: Year): Bag(SchoolClassResponse) <b>pre</b> not studentId .ocllsUndefined() and not year .ocllsUndefined() and SchoolUserService::isLoggedInUserParent(studentId) and SchoolUserService::isLoggedInStudent(studentId)	<b>context</b> StudentService::getSchoolClassByYear(studentId: UUID, year: Year): Bag(SchoolClassResponse) <b>post</b> if not result .ocllsUndefined() then result->forAll(c   c.getYear() = year.getValue()) endif



<b>+getById(uuid: UUID):SchoolUserDetailed</b>	Restituisce le informazioni di uno studente dal suo id	<b>context</b> StudentService:: getById(uuid: UUID):SchoolUserDetailed <b>pre</b> -	<b>context</b> StudentService:: getById(uuid: UUID):SchoolUserDetailed <b>post</b> -
<b>+ attitude(studentId: UUID):String</b>	Restituisce l'atteggiamento dello studente (ad esempio, positivo o negativo).	<b>context</b> StudentService:: attitude(studentId: UUID):String <b>pre</b> not studentId .oclIsUndefined() and (SchoolUserService::isLoggedInUserParent(studentId) or AuthenticationService::isStudent())	<b>context</b> StudentService attitude(studentId: UUID):String <b>post</b> result = OrientationService::attitude
<b>+create(studentRequest: StudentRequest): SchoolUserDetailed</b>	Crea un nuovo studente basato sulla richiesta e restituisce i dettagli.	<b>context</b> StudentService:: create(studentRequest: StudentRequest): SchoolUserDetailed <b>pre</b> studentRequest.name().size() > 0 <b>pre</b> request.surname.size() > 0 <b>pre</b> request.taxId.size() = 16 <b>pre</b> DateYear::now - request.birthDay.year >= 12 <b>pre</b> AuthenticationService::isSecretary() and not AuthenticationService::getSecretary().oclIsUndefined()	<b>context</b> StudentService: create(studentRequest: StudentRequest): SchoolUserDetailed <b>post</b> if not result .oclIsUndefined() then result.getName() = studentRequest.name() and result.getSurname() = studentRequest.surname() and result.getEmail() = studentRequest.email() and result.getUsername() = studentRequest.name().toLowerCase() + "." +

		<b>pre</b> not studentRequest.email().oclIsUndefined() and not studentRequest.classId().oclIsUndefined()	studentRequest .surname().toLowerCase() + "unique" and result .getSchoolClasses()- >exists(c   c.getId() = studentRequest.classId()) and result .getPendingState() = PendingState .FIRST_LOGIN
--	--	---	--

### 3.1.5 ParentService extends CrudService

Servizio per la gestione dei genitori, estende CrudService, eredita le operazioni CRUD di base.

Nome	Descrizione	Precondizione	Post condizione
<b>+addStudent(studentId: UUID, parentId: UUID): ParentResponse</b>	Associa uno studente a un genitore, verificando l'esistenza di entrambi nel sistema.	<b>context</b> ParentService:: addStudent(UUID studentId, UUIDparentId): ParentResponse <b>pre</b> not StudentService ::getById(studentId) .oclIsUndefined() <b>pre</b> ParentService ::getById(studentId).oclIsUndefined()	<b>context</b> ParentService:: addStudent(UUID studentId, UUIDparentId): ParentResponse <b>post</b> students .includes(StudentService:: getById(@pre.studentId))
<b>+ getTeachers(): Bag(SchoolUserResponse)</b>	Tutti gli insegnanti che insegnano ai figli	<b>context</b> ParentService ::getTeachers(): Bag (SchoolUserResponse) <b>pre</b> AuthenticationService ::isParent() and not AuthenticationService ::getParent() .oclIsUndefined()	<b>context</b> ParentService ::getTeachers(): Bag (SchoolUserResponse) <b>post</b> -

<b>+ getStudents():</b> Bag(SchoolUserDetailed)	Recupera tutti gli studenti associati al genitore	<b>context</b> ParentService::getStudents(): Bag(SchoolUserDetailed) <b>pre:</b> AuthenticationService::isParent() and not AuthenticationService::getParent() .ocllsUndefined()	<b>context</b> ParentService::getStudents(): Bag(SchoolUserDetailed) <b>post</b> -
--	---	--	--

### 3.1.6 SecretaryService extends CrudService

Questa classe è necessaria per definire il percorso REST per la gestione degli account Segreteria. Tutti i metodi sono ereditati da CrudService.

Nome	Descrizione	Precondizione	Post condizione
-	-	-	-

### 3.1.7 TeacherService extends CrudService

Servizio per la gestione degli insegnanti, estende CrudService.

Nome	Descrizione	Precondizione	Post condizione
<b>+ getSchoolClasses():</b> Bag(SchoolClassResponse)	Restituisce la lista delle classi scolastiche associate al docente loggato.	<b>context</b> TeacherService::getSchoolClasses(): Bag(SchoolClassResponse) <b>pre</b> AuthenticationService::isTeacher() and not AuthenticationService::getTeacher() .ocllsUndefined()	<b>context</b> TeacherService::getSchoolClasses(): Bag(SchoolClassResponse) <b>post</b> -
<b>+ getTeaching():</b> Bag(String)	Restituisce una lista di insegnamenti associati all'insegnante	<b>context</b> TeacherService::getTeaching(): Bag(String)	<b>context</b> TeacherService::getTeaching(): Bag(String)

		<b>pre</b> AuthenticationService ::isTeacher() and not AuthenticationService ::getTeacher() .ocllsUndefined()	<b>post</b> if not result.ocllsUndefined() then result = AuthenticationService::ge tTeacher().getTeachings() - >collect(teaching   teaching.getSubjectTitle(). getTitle()) ->asBag() - >sortedBy(title   title) endif
<b>+ getTeachers():</b> Bag(TeacherRespons e)	Restituisce tutti gli insegnanti	<b>context</b> TeacherService::getTeache rs(): Bag(TeacherResponse) <b>pre</b> AuthenticationService ::isSecretary ()	<b>context</b> TeacherService::getTeache rs(): Bag(TeacherResponse) <b>post</b> result.forAll(r   r.getSchool() = AuthenticationService::ge tSecretary())
<b>+getTeacherTeaching(</b> teacheruuid: String): TeacherDetailsRespo nse	Recupera tutte le informazioni di uno specifico insegnante	<b>context</b> TeacherService ::getTeacherTeaching (teacheruuid: String): TeacherDetailsResponse <b>pre</b> not teacheruuid .ocllsUndefined()	<b>context</b> TeacherService ::getTeacherTeaching (teacheruuid: String): TeacherDetailsResponse <b>post</b> -

### 3.1.8 SchoolService extends CrudService

Servizio per la gestione delle scuole, estende CrudService.

Nome	Descrizione	Precondizione	Post condizione
<b>+getNumberStudent</b> s(code: String): Integer	Restituisce il numero di studenti iscritti alla scuola identificata dal codice.	<b>context</b> SchoolService:: getNumberStudents(Strin g code) : Integer <b>pre</b> request.code.size() = 7	<b>context</b> SchoolService:: getNumberStudents(Strin g code) : Integer <b>post</b> result >= 0

### 3.1.9 ReceptionAgendaService

Servizio per la gestione dell'agenda del ricevimento, per prenotazioni e appuntamenti.

Nome	Descrizione	Precondizione	Post condizione
<b>+book(receptionTime slotID: Integer): Boolean</b>	Prenota un orario di ricevimento per un genitore.	<b>context</b> ReceptionAgendaService::book(receptionTimeslotID: Integer): Boolean <b>pre</b> not receptionTimeslotID .ocllsUndefined() and AuthenticationService ::isParent() and not AuthenticationService ::getParent() .ocllsUndefined()	<b>context</b> ReceptionAgendaService::book(receptionTimeslotID: Integer): Boolean <b>post</b> if result then let parent = AuthenticationService ::getParent() in let students = parent.getStudents() in let slot = ReceptionTimeslot ::allInstances()->any(s   s.getId() = receptionTimeslotID and s.getDate() > Date::now()) in slot .getReceptionTimetable(). getTeacher() .getTeacherClasses() ->exists(tc   tc.getSchoolClass() .getStudents()- >exists(student   students ->includes(student))) and slot.getBooked() = slot.getBooked() @pre + 1 else true endif
<b>+addTimeslot(request: ReceptionTimeslotRequest):</b>	Aggiunge un nuovo orario di ricevimento.	<b>context</b> ReceptionAgendaService ::addTimeslot(request: ReceptionTimeslotRequest)	<b>context</b> ReceptionAgendaService::addTimeslot(request: ReceptionTimeslotRequest)

ReceptionTimeslotResponse		t):ReceptionTimeslotResponse <b>pre</b> not request .ocllsUndefined() and AuthenticationService ::isTeacher() and not AuthenticationService ::getTeacher() .ocllsUndefined()	t): ReceptionTimeslotResponse <b>post</b> let teacher = AuthenticationService ::getTeacher() Result .getReceptionTimetable(). getTeacher() = teacher and result.getDate() = request.date() and result.getHour() = request.hour() and result.getCapacity() = request.capacity() endif
+refuse(receptionTimeslotID: ReceptionBookingId): Boolean	Rifiuta una prenotazione di orario di ricevimento.	<b>context</b> ReceptionAgendaService ::refuse(receptionTimeslotID: ReceptionBookingId): Boolean <b>pre</b> not receptionTimeslotID .ocllsUndefined() and AuthenticationService ::isTeacher() and not AuthenticationService ::getTeacher() .ocllsUndefined() and ReceptionBooking ::allInstances()->exists(rb   rb.getId() = receptionTimeslotID and rb .getReceptionTimeslot() .getReceptionTimetable() .getTeacher().getId() = AuthenticationService ::getTeacher().getId())	<b>context</b> ReceptionAgendaService:: refuse(receptionTimeslotID: ReceptionBookingId): Boolean <b>post</b> result and ReceptionBooking ::allInstances()->any(rb   rb.getId() = receptionTimeslotID) .getRefused() = true

<b>+getNotConfirmed(tableId: Integer) :</b> Bag( ReceptionBookingResponse)	Ottiene gli orari di ricevimento non confermati per una data e un tavolo specifico.	<b>context</b> ReceptionAgendaService::getNotConfirmed(tableId: Integer) : Bag( ReceptionBookingResponse) <b>pre</b> self.getById(tableId) <> null <b>pre</b> self.getById(tableId).teacher = UserService::getLoggedTeacher() or AuthenticationService::getParent().students.schoolClass.teachers.exist(u   u.receptionTable.tableId = tableId)	<b>context</b> ReceptionAgendaService::getNotConfirmed(tableId: Integer) : Bag( ReceptionBookingResponse) <b>post</b> -
<b>+ confirm(id :Long) :</b> Boolean	Conferma una prenotazione di orario di ricevimento.	<b>context</b> ReceptionAgendaController::confirm(id :Long) : Boolean <b>pre</b> not receptionTimeslotID.isUndefined() and AuthenticationService::isTeacher() and not AuthenticationService::getTeacher().isUndefined() and ReceptionBooking::allInstances()->exists(rb   rb.getId() = receptionTimeslotID and rb.getReceptionTimeslot().getReceptionTimetable().getTeacher().getId() = AuthenticationService::getTeacher().getId())	<b>context</b> ReceptionAgendaService::confirm(id: Long) : Boolean <b>post</b> result and ReceptionBooking::allInstances()->any(rb   rb.getId() = receptionTimeslotID).getConfirmed() = true

<b>+ getRefused(tableId: Integer) :</b> Bag(ReceptionBookingResponse)	Ottiene l'esito del rifiuto di una prenotazione di orario di ricevimento per un tavolo specifico.	<b>context</b> ReceptionAgendaController:: getRefused(tableId: Integer): Bag(ReceptionBookingResponse) <b>pre</b> self.getByld(tableId) <> null <b>pre</b> self.getByld(tableId).teacher = UserService::getLoggedTeacher()	<b>context</b> ReceptionAgendaService:: getRefused(tableId: Integer) : Bag(ReceptionBookingResponse) <b>post</b> result = self.getAll().forall(r   r.refused = false)
<b>+ getBookedSlots():</b> Sequence(ReceptionBookingResponse)	Ottiene la lista degli orari di ricevimento prenotati per un tavolo specifico.	<b>context</b> ReceptionAgendaService:: getBookedSlots():Sequence(ReceptionBookingResponse) <b>pre</b> AuthenticationService::isTeacher() or AuthenticationService::isParent()	<b>context</b> ReceptionAgendaService:: getBookedSlots():Sequence(ReceptionBookingResponse) <b>post</b> if AuthenticationService::isTeacher() then result = ReceptionBooking::allInstances() ->select(rb   rb .getReceptionTimeslot() .getReceptionTimetable().getTeacher() = AuthenticationService::getTeacher()) ->sortedBy(b   b.getReceptionTimeslot().getDate().toString() .concat(b.getReceptionTimeslot().getHour().toString()) .concat(b.getBookingOrder().toString())) else



			<pre> result = ReceptionBooking ::allInstances() -&gt;select(rb   rb .getParent() = AuthenticationService ::getParent()) -&gt;sortedBy(b   b .getReceptionTimeslot() .getDate().toString() .concat(b .getReceptionTimeslot() .getHour().toString()) .concat(b.getBookingOrd er().toString())) endif </pre>
<b>+getSlots(teacherID: UUID) : Bag ReceptionTimeslotResponse)</b>	Ottiene tutte gli slot di ore di uno specific insegnante	<b>context</b> ReceptionAgendaService::getSlots(teacherID: UUID): Bag(ReceptionTimeslotResponse) <b>pre</b> not teacherID .ocllsUndefined() and (AuthenticationService::isTeacher() or AuthenticationService::isParent())	<b>context</b> ReceptionAgendaService::getSlots(teacherID: UUID): Bag(ReceptionTimeslotResponse) <b>post</b> -
<b>+createReceptionTimeTable(teacherId: UUID, textInfo: String): ReceptionTimetable</b>		<b>context</b> ReceptionAgendaService::createReceptionTimetable(teacherId: UUID, textInfo: String): ReceptionTimetable <b>pre</b> not teacherId .ocllsUndefined() andnot textInfo .ocllsUndefined() and (AuthenticationService	<b>context</b> ReceptionAgendaService::createReceptionTimetable(teacherId: UUID, textInfo: String): ReceptionTimetable <b>post</b> result.getTeacher().getId() = teacherId andresult .getStartValidity() = Date::now() and result

		::isTeacher() and AuthenticationService ::getTeacher() .getId() = teacherId) or (AuthenticationService::is Secretary() and AuthenticationService::ge tSecretary().getSchool().g etTeachers()->exists(t   t.getId() = teacherId))	.getTextInfoReception() = textInfo
--	--	---	---------------------------------------

### 3.1.10 Class AgendaService

Servizio per la gestione dell'agenda di classe.

Nome	Descrizione	Precondizione	Post condizione
<b>+ getTotalHour(id: Integer) : Integer</b>	Restituisce il totale delle ore di lezione per una classe o un insegnante identificato da un ID specifico.	<b>context</b> ClassAgendaService::getTotalHour(id: Integer) : Integer <b>pre</b> not classId.ocllsUndefined()	<b>context</b> ClassAgendaService::getTotalHour(id: Integer) : Integer <b>post</b> result = self.getById(id).teachingTimeslots.count()
<b>+ addTimeslot(classId: Integer, request: TeachingTimeslotRequest)</b>	Aggiunge un orario di lezione (timeslot) per una classe o insegnante identificato dall'ID.	<b>context</b> ClassAgendaService::addTimeslot(classId: Integer, request: TeachingTimeslotRequest) <b>pre</b> not classId.ocllsUndefined() and not request.ocllsUndefined() and not request timetableId().ocllsUndefined() and not request.subject().ocllsUndefined() and not request.username().ocllsUndefined() and not request.hour().ocllsUndefined()	<b>context</b> ClassAgendaService::addTimeslot(classId: Integer, request: TeachingTimeslotRequest) <b>post</b> if request.dayWeek() > 0 then let timetable = ClassTimetable::allInstances() ->any(ct   ct.getId() = request.timetableId()) in timetable .getTeachingTimeslots() ->exists(ts   ts.getDate() >= timetable.getStartValidity() and ts.getDate() <= timetable.getEndValidity() and ts.getHour() = request.hour() and ts.getTeaching().getSubjectTitle().getTitle()=request.subject() and ts.getTeaching().getTeacher().getUsername() = request.username()) endif

<b>+ sign(classId: Integer, request: SignHourRequest): TeachingTimeslotDetailedResponse</b>	Gestisce la registrazione della firma di un orario di lezione per una determinata classe o insegnante.	<b>context</b> ClassAgendaService::sign(classId: Integer, request: SignHourRequest): TeachingTimeslotDetailedResponse  <b>pre</b> not classId .oclIsUndefined() and not request .oclIsUndefined() and not request.date() .oclIsUndefined() and SchoolUserService::isLoggedInTeacherClass(classId)	<b>context</b> ClassAgendaService::sign(classId: Integer, request: SignHourRequest): TeachingTimeslotDetailedResponse  <b>post</b> if not result.oclIsUndefined() then result .getTeachingTimeslot() .getClassTimetable() .getSchoolClass() .getId() = classId and result .getTeachingTimeslot() .getDate() = request.date() and result .getTeachingTimeslot() .getHour() = request.hour() and result .getTeachingTimeslot() .getTeaching() .getTeacher().getId() = AuthenticationService::getTeacher().getId() endif
<b>+ isSigned(signedId: Integer, date: Date): Boolean</b>	Verifica se un orario di lezione è stato firmato per una determinata classe o insegnante in una data specifica.	<b>context</b> ClassAgendaService::isSigned(signedId: Integer, date: Date): Boolean  <b>pre</b> not signedId .oclIsUndefined() and not date.oclIsUndefined()	<b>context</b> ClassAgendaService::isSigned(signedId: Integer, date: Date): Boolean  <b>post</b> -

<b>+getWeekTimeslot(classID: Integer, date: Date):</b> Bag(TeachingTimeslotResponse)	Recupera tutte le ore di una settimana	<b>context</b> ClassAgendaService ::getWeekTimeslot(classID: Integer, date: Date): Bag(TeachingTimeslotResponse)  <b>pre</b> not classID .ocllsUndefined() and not date .ocllsUndefined() and (SchoolUserService ::isLoggedInParentStudent Class(classID) or (AuthenticationService ::isStudent() and AuthenticationService ::getStudent() .getSchoolClasses() ->exists(sc   sc.getId() = classID)))	<b>context</b> ClassAgendaService ::getWeekTimeslot(classID: Integer, date: Date): Bag(TeachingTimeslotResponse)  <b>post</b> result->forAll(timeslot   timeslot.getDate() >= date.with(DayOfWeek ::MONDAY) and timeslot .getDate() <= date.with(DayOfWeek ::SUNDAY) and timeslot .getTeachingTimeslot() .getClassTimetable() .getSchoolClass() .getId() = classID)
<b>+getTeachingTimeslot(classId: Integer, date: Date):</b> Bag(TeachingTimeslotDetailedResponse)	Recupera tutte le ore di insegnamento in una settimana	<b>context</b> ClassAgendaService::getTeachingTimeslot(classId: Integer, Date: Date): Bag(TeachingTimeslotDetailedResponse)  <b>pre</b> not classId .ocllsUndefined() and not date.ocllsUndefined() and SchoolUserService ::isLoggedInTeacherClass(classId)	<b>context</b> ClassAgendaService::getTeachingTimeslot(classId: Integer, Date: Date): Bag(TeachingTimeslotDetailedResponse)  <b>post</b> result->forAll(timeslot   timeslot.getDate() = date and timeslot .getTeachingTimeslot() .getClassTimetable() .getSchoolClass() .getId() = classId)
<b>+createTimeTable(request: ClassTimeTableRequest)</b>	Crea un calendario per una classe	<b>context</b> ClassAgendaService ::createTimeTable(request: ClassTimeTableRequest)	<b>context</b> ClassAgendaService ::createTimeTable(request: ClassTimeTableRequest)

		<b>pre</b> not request .ocllsUndefined() and not request .schoolClassId() .ocllsUndefined() and not request.startDate() .ocllsUndefined()	<b>post</b> ClassTimetable ::allInstances()->exists(ct   ct.getSchoolClass() .getId() = request.schoolClassId() and ct.getStartValidity() = request.startDate() and (request.endDate() .ocllsUndefined() or ct.getEndValidity() = request.endDate()))
<b>+getClassTimeslot(classId: Integer, now: Date): Bag(TeachingTimeslotResponse)</b>	Recupera tutte le ore di insegnamento di una classe in un giorno specifico	<b>context</b> ClassAgendaService::getClassTimeslot(classId: Integer, now: Date): Bag(TeachingTimeslotResponse) <b>pre</b> not classId.ocllsUndefined() and not now.ocllsUndefined()	<b>context</b> ClassAgendaService ::getClassTimeslot(classId: Integer, now: Date): Bag(TeachingTimeslotResponse) <b>post</b> result->forAll(timeslot   timeslot .getTeachingTimeslot() .getClassTimetable() .getSchoolClass().getId() = classId and timeslot .getDate() >= now)
<b>+getTimeTable(classId: Integer): Bag(TimeTableResponse)</b>	Recupera tutto il calendario di una specifica classe	<b>context</b> ClassAgendaService ::getTimeTable(classId: Integer): Bag(TimeTableResponse) <b>pre</b> not classId.ocllsUndefined()	<b>context</b> ClassAgendaService ::getTimeTable(classId: Integer): Bag(TimeTableResponse) <b>post</b> result->forAll(timetable   timetable .getSchoolClass() .getId() = classId)

### 3.1.11 MarkService

Servizio per la gestione dei voti degli studenti.

Nome	Descrizione	Precondizione	Post condizione
<b>+ viewReport(id: Integer): SemesterReportResponse</b>	Imposta un report semestrale come visualizzato.	<b>context</b> MarkService:: viewReport(id: Integer): SemesterReportResponse <b>pre</b> not id.isNull() and AuthenticationService::getParent().students.exists(s   s.semesterReports.exists(r   r.id = id))	<b>context</b> MarkService:: viewReport(id: Integer): SemesterReportResponse <b>post</b> result = AuthenticationService::getParent().students.any(s   s.semesterReports.exists(r   r.id = id and r.viewed = true))
<b>+ getReport(studentId: UUID, year: Short, semester: Boolean): SemesterReportResponse</b>	Ottiene il report semestrale di un dato anno e semestre.	<b>context</b> MarkService:: getReport(studentId: UUID, year: Short, semester: Boolean): SemesterReportResponse <b>pre</b> not studentId.isNull() and (SchoolUserService::isLoggedInUserParent(studentId) or SchoolUserService::isLoggedInStudent(studentId) and AuthenticationService::getParent().semesterReports.exists(r   r.year = year and r.firstSemester = semester and r.publicField = true) or SchoolUserService::isLoggedInTeacherStudent(studentId))	<b>context</b> MarkService:: getReport(studentId: UUID, year: Short, semester: Boolean): SemesterReportResponse <b>post</b> if result.isNull() then result = SemesterReport::allInstances()->any(r   r.student.id = studentId and r.year = year and r.firstSemester = semester) endif
<b>+ getRatings(classId: Integer, teaching: String, date: Date): Sequence(RatingResponse)</b>	Recupera tutti i voti di una classe di una materia in un dato giorno.	<b>context</b> MarkService:: getRatings(classId: Integer, teaching: String, date: Date): Sequence(RatingResponse)	<b>context</b> MarkService:: getRatings(classId: Integer, teaching: String, date: Date): Sequence(RatingResponse)

ponse)		e) <b>pre</b> not classId.ocllsUndefined() and SchoolUserService::isLoggedTeacherClass(classId) and AuthenticationService::getTeacher().teachings.exists(t   t.subjectTitle.title = teaching) and date > Date::now()	e) <b>pre</b> result = Mark::allInstances()->select(m   m.student.schoolClasses.exists(c   c.id = classId) and m.teaching.subjectTitle.title = teaching and m.date = date
<b>+getEveryRatings(classId: Integer, teaching: String): Sequence(RatingResponse)</b>	Recupera tutti i voti di una materia.	<b>context</b> MarkService::getEveryRatings(classId: Integer, teaching: String): Sequence(RatingResponse) <b>pre</b> not classId.ocllsUndefined() and SchoolUserService::isLoggedTeacherClass(classId) and AuthenticationService::getTeacher().teachings.exists(t   t.subjectTitle.title = teaching)	<b>context</b> MarkService::getEveryRatings(classId: Integer, teaching: String): Sequence(RatingResponse) <b>pre</b> result = Mark::allInstances()->select(m   m.student.schoolClasses.exists(c   c.id = classId) and m.teaching.subjectTitle.title = teaching)
<b>+getAverage(studentId: UUID, year: Year): Double</b>	Calcola la media dei voti per un anno scolastico di uno studente.	<b>context</b> MarkService::getAverage(studentId: UUID, year: Year): Double <b>pre</b> not studentId.ocllsUndefined() and (SchoolUserService::isLoggedUserParent(studentId) or SchoolUserService::isLoggedStudent(studentId)) and year <= DateYear::now()	<b>context</b> MarkService::getAverage(studentId: UUID, year: Year): Double <b>post</b> let marks = MarkService::getMarkByYear(studentId, year) in result = marks.sum().div(marks.size())
<b>+addMark(reportId:</b>	Aggiunge un voto a un	<b>context</b> MarkService::	<b>context</b> MarkService::



Integer, subject: Subject, mark: Integer): SemesterReportResponse	report semestrale per una materia specifica.	addMark(reportId: Integer, subject: Subject, mark: Integer): SemesterReportResponse <b>pre</b> AuthenticationService::isTeacher() and mark >= 0 and mark <= 10 and SemesterReport::allInstances()->exists(r   r.id = reportId and not r.marks.exists(m   m.subject = subject))	addMark(reportId: Integer, subject: Subject, mark: Integer): SemesterReportResponse <b>post</b> result = SemesterReport::allInstances()->any(r   r.id = reportId and r.marks.includes(m   m.subject = subject and m.mark = mark) and r.passed = (r.marks.sum().div(r.marks.size()) > 6))
+getMarkByYear(studentId: UUID, year: Year): Sequence(MarkResponse)	Ottiene i voti di uno studente in un anno scolastico specificato.	<b>context</b> MarkService:: getMarkByYear(studentId: UUID, year: Year): Sequence(MarkResponse) <b>pre</b> not studentId.isNull() and (SchoolUserService::isLoggedInUserParent(studentId) or SchoolUserService::isLoggedInStudent(studentId)) and year <= DateYear::now()	<b>context</b> MarkService:: getMarkByYear(studentId: UUID, year: Year): Sequence(MarkResponse) <b>post</b> result = Mark::allInstances()->select(m   m.student.id = studentId and m.date.year = year)
+ create(mark: MarkRequest): MarkResponse	Crea una nuova valutazione.	<b>context</b> MarkService:: create(mark: MarkRequest): MarkResponse <b>pre</b> not mark.isNull() and mark.date <= Date::now() and mark.mark >= 0 and mark.mark <= 10 and	<b>context</b> MarkService:: create(mark: MarkRequest): MarkResponse <b>post</b> result.mark = mark.mark and result.date = mark.date and result.studentId = mark.studentId and

		mark.type.ocllsUndefined() ) and AuthenticationService::getTeacher().teachings.exists(t   t.teachingId = mark.teachingId) and SchoolUserService::isLoggedInTeacherStudent(mark.studentId)	result.teaching = mark.teaching and result.type = mark.type and result.description = mark.description
<b>+ update</b> (mark: MarkUpdateRequest, studentId: UUID): MarkResponse	Aggiorna la valutazione di uno studente.	<b>context</b> MarkService::update(mark: MarkUpdateRequest, studentId: UUID): MarkResponse <b>pre</b> not mark.ocllsUndefined() and mark.mark >= 0 and mark.mark <= 10 and SchoolUserService::isLoggedInTeacherStudent(studentId) and Mark::allInstances().exists(m   m.id = mark.id and m.teaching.teacher = AuthenticationService::getTeacher() and m.student.id = studentId)	<b>context</b> MarkService::update(mark: MarkUpdateRequest, studentId: UUID): MarkResponse <b>post</b> result.id = mark.id and result.studentId = studentId and result.mark = mark.mark and result.type = mark.type and result.description = mark.description
<b>+ delete</b> (id: Integer): Boolean	Cancella la valutazione da uno student.	<b>context</b> MarkService::delete(id: Integer): Boolean <b>pre</b> Mark::allInstances()->exists(m   m.id = id and m.teaching.teacher = AuthenticationService::getTeacher())	<b>context</b> MarkService::delete(id: Integer): Boolean <b>post</b> not Mark::allInstances().exists(m   m.id = id)

### 3.1.12 NoteService extends CrudService

Servizio per la gestione delle note disciplinari.

Nome	Descrizione	Precondizione	Post condizione
<b>+ view</b> (studentId: UUID, noteId: UUID):	Conferma la visione di una nota specifica in base all'ID.	<b>context</b> NoteService::view(studentId: UUID, noteId: UUID) <b>pre</b> not studentId.isNull() and not noteId.isNull() and (SchoolUserService::isLoggedInUserParent(studentId) or SchoolUserService::isLoggedInStudent(studentId)) and Note::allInstances()->exists(n   n.id = noteId and n.student.id = studentId)	<b>context</b> NoteService::view(studentId: UUID, noteId: UUID) <b>post</b> Note::allInstances()->exists(n   n.id = noteId and n.student.id = studentId and n.viewed = true)
<b>+ getNoteByStudentId</b> (studentId: UUID, classId: Integer): Sequence(NoteResponse)	Recupera tutte le note disponibili per uno studente.	<b>context</b> NoteService::getNoteByStudentId(studentId: UUID, classId: Integer): Sequence(NoteResponse) <b>pre</b> not studentId.isNull() and not classId.isNull() and (SchoolUserService::isLoggedInUserParent(studentId) or SchoolUserService::isLoggedInStudent(studentId) or SchoolUserService::isLoggedInTeacherStudent(studentId))	<b>context</b> NoteService::getNoteByStudentId(studentId: UUID, classId: Integer): Sequence(NoteResponse) <b>post</b> result = Note::allInstances()->select(n   n.student.id = studentId and n.student.schoolClasses.exists(c   c.id = classId))

### 3.1.13 JustifiableService

Servizio per la gestione delle giustificazioni di assenza e ingressi in ritardo.

Nome	Descrizione	Precondizione	Post condizione
+ justify(studentId: UUID, justificationId: UUID, justificationText: String, absence: Boolean): JustifiableResponse	Giustifica l'assenza o ingresso in ritardo di uno studente con un testo giustificazione.	<b>context</b> JustifiableService::justify(studentId: UUID, justificationId: UUID, justificationText: String, absence: Boolean) : JustifiableResponse <b>pre</b> not studentId.ocllsUndefined() and not justificationId.ocllsUndefined() and justificationText.size() > 0 and justificationText.size() < 512 and SchoolUserService::isLoggedInUserParent(studentId) and JustifiableEntity::allInstances()->exists(j   j.id = justificationId and (justified = false or needsJustification = true))	<b>context</b> JustifiableService::justify(studentId: UUID, justificationId: UUID, justificationText: String, absence: Boolean) : JustifiableResponse <b>post</b> if absence then result = Absence::allInstances()->any(a   a.id = justificationId and a.justificationText = justificationText and a.justified = true) else result = Delay::allInstances()->any(d   d.id = justificationId and d.justificationText = justificationText and d.justified = true)
+getAbsencesByYear(studentId: UUID, year: Year): Sequence(JustifiableResponse)	Recupera le assenze di uno studente in uno specific anno.	<b>context</b> JustifiableService::getAbsencesByYear(studentId: UUID, year: Year) : Sequence(JustifiableResponse) <b>pre</b> not studentId.ocllsUndefined() and (SchoolUserService::isLoggedInUserParent(studentId) or SchoolUserService::isLoggedInStudent(studentId))	<b>context</b> JustifiableService::getAbsencesByYear(studentId: UUID, year: Year) : <u>Sequence</u> (JustifiableResponse) <b>post</b> result = Absence::allInstances()->select(a   a.date.year = year and a.student.id = studentId)

<b>+getDelayByYear</b> (studentId: UUID, year: Year): Sequence(JustifiableResponse)	Recupera i ritardi di uno studente in uno specifico anno.	<b>context</b> JustifiableService::getDelayByYear(studentId: UUID, year: Year): Sequence(JustifiableResponse) <b>pre</b> not studentId .ocllsUndefined() and (SchoolUserService::isLoggedUserParent(studentId) or SchoolUserService::isLoggedStudent(studentId))	<b>context</b> JustifiableService::getDelayByYear(studentId: UUID, year: Year): Sequence(JustifiableResponse) <b>post</b> result = Delay::allInstances()->select(d   d.date.year = year and d.student.id = studentId)
<b>+getTotalAbsences</b> (studentId: UUID, year: Year): Integer	Recupera il numero totale delle assenze per uno studente.	<b>context</b> JustifiableService::getTotalAbsences(studentId: UUID, year: Year): Integer <b>pre</b> not studentId .ocllsUndefined() and (SchoolUserService::isLoggedUserParent(studentId) or SchoolUserService::isLoggedStudent(studentId) or SchoolUserService::isLoggedTeacherStudent(studentId))	<b>context</b> JustifiableService::getTotalAbsences(studentId: UUID, year: Year): Integer <b>post</b> result = Absence::allInstances()->select(a   a.date.year = year and a.student.id = studentId)->size()
<b>+getTotalDelays</b> (studentId: UUID, year: Year): Integer	Recupera il totale degli ingressi in ritardo per uno studente.	<b>context</b> JustifiableService::getTotalDelays(studentId: UUID, year: Year): Integer <b>pre</b> not studentId .ocllsUndefined() and (SchoolUserService::isLoggedUserParent(studentId) or SchoolUserService::isLoggedStudent(studentId))	<b>context</b> JustifiableService::getTotalDelays(studentId: UUID, year: Year): Integer <b>post</b> result = Delay::allInstances()->select(d   d.date.year = year and d.student.id = studentId)->size

### 3.1.14 AttendanceService

Servizio per la gestione delle presenze degli studenti.

Nome	Descrizione	Precondizione	Post condizione
<b>+getAttendance</b> (classId: Integer, date: Date): Sequence(AttendanceResponse)	Recupera i registri delle presenze per una classe specifica in una data specifica.	<b>context</b> AttendanceService::getAttendance(classId: Integer, date: Date): Sequence(AttendanceResponse) <b>pre</b> not classId.isNull() and SchoolUserService::isLoggedInTeacherClass(classId) and date <= Date::now()	<b>context</b> AttendanceService::getAttendance(classId: Integer, date: Date): Sequence(AttendanceResponse) <b>post</b> result->forall(response   Student::allInstances()->exists(s   s.id = response.id and s.schoolClass = classId) and response.isAbsent = Absence::allInstances()->exists(a   a.student.id = response.id and a.date = date) and response.isDelayed = Delay::allInstances()->exists(d   d.student.id = response.id and d.date = date) and response.delayTime = Delay::allInstances()->any(d   d.student.id = response.id and d.date = date)->getDateTime() and response.delayNeedJustification = Delay::allInstances()->any(d   d.student.id = response.id and d.date = date)->getNeedJustification() and response.totalAbsence = response.student.absences.size()

			and response.totalDelay = response.student.delays.size())
<b>+saveAttendance(classId: Integer, date: Date, attendanceRequests: Bag(AttendanceRequest))</b>	Salva i registri delle presenze per una classe specifica in una data specifica.	<b>context</b> AttendanceService:: saveAttendance(classId: Integer, date: Date, attendanceRequests: Bag(AttendanceRequest)) <b>pre</b> not classId.isNull() and SchoolUserService::isLoggedInTeacherClass(classId) and date <= Date::now() and attendanceRequests.forAll(a   Student::allInstances()->exists(s   s.id = a.id and s.schoolClass = classId)	<b>context</b> AttendanceService:: saveAttendance(classId: Integer, date: Date, attendanceRequests: Bag(AttendanceRequest)) <b>post</b> attendanceRequests.forAll(request   if request.isAbsent() then Absence::allInstances()->exists(a   a.student.id = request.id and a.date = date) else not Absence::allInstances()->exists(a   a.student.id = request.id and a.date = date) endif and if request.isDelayed() then Delay::allInstances()->exists(d   d.student.id = request.id and d.date = date and d.date.hour = request.delayTimeHour and d.date.minute = request.delayTimeMinute) else not Delay::allInstances()->exists(d   d.student.id = request.id and d.date = date and d.date.hour = request.delayTimeHour and d.date.minute = request.delayTimeMinute) endif)

### 3.1.15ClassworkService

Servizio per la gestione del lavoro in classe, compiti o attività.

Nome	Descrizione	Precondizione	Post condizione
<b>+getClassActivities(classId: Integer):</b> Bag(ClassworkResponse)	Recupera un elenco di attività di classe per una classe specifica.	<b>context</b> ClassworkService::getClassActivities(classId: Integer): Bag(ClassworkResponse) <b>pre</b> not classId.isNull() and (SchoolUserService::isLoggedInParentStudentClass(classId) or AuthenticationService::isStudent() and AuthenticationService::getStudent().schoolClasses.exists(c   c.id = classId))	<b>context</b> ClassworkService::getClassActivities(classId: Integer): Bag(ClassworkResponse) <b>post</b> result = ClassActivity::allInstances()->select(a   a.signedHour.teachingTimeslot.classTimetable.schoolClass.id = classId)
<b>+getHomework(classId: Integer):</b> Bag(HomeworkResponse)	Recupera un elenco di compiti assegnati per una classe specifica.	<b>context</b> ClassworkService::getHomework(classId: Integer): Bag(HomeworkResponse) <b>pre</b> not classId.isNull() and (SchoolUserService::isLoggedInParentStudentClass(classId) or AuthenticationService::isStudent() and AuthenticationService::getStudent().schoolClasses.exists(c   c.id = classId))	<b>context</b> ClassworkService::getHomework(classId: Integer): Bag(HomeworkResponse) <b>post</b> result = Homework::allInstances()->select(h   h.signedHour.teachingTimeslot.classTimetable.schoolClass.id = classId)
<b>+createActivity(request: ClassActivityRequest,</b>	Crea una nuova attività di classe.	<b>context</b> ClassworkService::createActivity(request: ClassActivityRequest,	<b>context</b> ClassworkService::createActivity(request: ClassActivityRequest,



signedHour: SignedHour)		signedHour: SignedHour) <b>pre</b> not request.ocllsUndefined() and not signedHour.ocllsUndefined() and signedHour.teacher = AuthenticationService::getTeacher() and request.title.size() <> 0	signedHour: SignedHour) <b>post</b> ClassActivity:: allInstances()->exists(a   a.id = result.id and a.signedHour = signedHour and a.title = request.title and a.description = request.description)
<b>+createHomework</b> (request: HomeworkRequest, signedHour: SignedHour)	Crea un nuovo compito assegnato.	<b>context</b> ClassworkService:: createHomework(request : HomeworkRequest, signedHour: SignedHour) <b>pre</b> not request.ocllsUndefined() and not signedHour.ocllsUndefined() and signedHour.teacher = AuthenticationService::getTeacher() and request.title.size() <> 0 and request.dueDate > Date::now()	<b>context</b> ClassworkService:: createHomework(request : HomeworkRequest, signedHour: SignedHour) <b>post</b> Homework:: allInstances()->exists(h   h.id = result.id and h.signedHour = signedHour and h.title = request.title and h.description = request.description and h.dueDate = request.dueDate and request.hasChat = (h.homeworkChats.size() > 0))

### 3.1.16 CommunicationService extends CrudService

Questa classe è necessaria per definire il percorso REST per la gestione degli avvisi.

Tutti i metodi sono ereditati da CrudService.

Nome	Descrizione	Precondizione	Post condizione
<b>+getCommunicationBySchoolClassId</b> (schoolClassId: Integer): Sequence(CommunicationResponse)	Recupera un elenco di comunicazioni per una classe specifica.	<b>context:</b> CommunicationService::getCommunicationBySchoolClassId(schoolClassId: Integer): Sequence(CommunicationResponse)  <b>pre:</b> SchoolUserService::isLoggedInTeacherClass(schoolClassId) or SchoolUserService::isLoggedInParentStudentClass(schoolClassId) or (AuthenticationService::isLoggedInStudent() and AuthenticationService::getStudent().schoolClasses->exists(sc   sc.id = schoolClassId))	<b>context:</b> CommunicationService::getCommunicationBySchoolClassId(schoolClassId: Integer): Sequence(CommunicationResponse)  <b>post:</b> result->forall(r   r.schoolClassId = schoolClassId) post: result->isOrderedByDescending(r   r.date)

### 3.1.17 MessageService

Servizio per la gestione dei messaggi delle chat.

Nome	Descrizione	Precondizione	Post condizione
<b>+addAttachment</b> (uuid: UUID, file: File): String	Aggiunge un allegato a un'entità specificata.	<b>context</b> MessageService::addAttachment(uuid: UUID, multipartFile: MultipartFile): String <b>pre</b> Message.allInstances()->exists(m   m.id = uuid) and ((AuthenticationService::isStudent() implies Message.allInstances()->any(m   m.id = uuid).student.id = AuthenticationService::getStudent().id) and (AuthenticationService::isTeacher() implies Message.allInstances()->any(m   m.id = uuid).teacher.id = AuthenticationService::getTeacher().id))	<b>context</b> MessageService::addAttachment(uuid: UUID, multipartFile: MultipartFile): String <b>post</b> let message: Message = Message.allInstances()->any(m   m.id = uuid) in message.attachment = FileService::uploadFile(multipartFile) and result = message.attachment
<b>+create</b> (message: String, chatId: UUID, isHomework: Boolean): MessageResponse	Crea un nuovo messaggio in una chat specifica.	<b>context</b> MessageService::create(message: String, chatId: UUID, isHomework: Boolean): MessageResponse <b>pre</b> (isHomework = true and HomeworkChat.allInstances()->exists(hc   hc.id = chatId and hc.completed = false)) or (isHomework = false and Ticket.allInstances()->exists(t   t.id = chatId and t.closed = false))	<b>context</b> MessageService::create(message: String, chatId: UUID, isHomework: Boolean): MessageResponse <b>post</b> result.text = message result.dateTime <> null implies result.dateTime.isValid()

<b>+getSender(message: Message): SchoolUser</b>	Restituisce le informazioni sul mittente di un messaggio identificato dall'ID del messaggio.	<b>context:</b> MessageService::getSender(message: Message): SchoolUser <b>pre</b> message.student <> null or message.parent <> null or message.teacher <> null or message.secretary <> null	<b>context:</b> MessageService::getSender(message: Message): SchoolUser <b>post</b> result = message.student or result = message.parent or result = message.teacher or result = message.secretary
---	--	---	--

### 3.1.18 HomeworkChatService

Servizio per la gestione della chat per i compiti a casa.

Nome	Descrizione	Precondizione	Post condizione
<b>+getChatWithMessagesSocket(chatId: UUID): HomeworkChatResponse</b>	Recupera una chat sui compiti insieme ai relativi messaggi utilizzando una connessione socket.	<b>context</b> HomeworkChatService::getChatWithMessagesSocket(chatId: UUID): HomeworkChatResponse <b>pre</b> HomeworkChat.allInstances()->exists(hc   hc.id = chatId and hc.completed = false)	<b>context</b> HomeworkChatService::getChatWithMessagesSocket(chatId: UUID): HomeworkChatResponse <b>post</b> result.id = chatId and result.title = HomeworkChat.allInstances()->select(hc   hc.id = chatId)->first().title and result.studentId = HomeworkChat.allInstances()->select(hc   hc.id = chatId)->first().student.id and result.completed = false and result.messages->forall(m   m.dateTime <> null)
<b>+sendMessage(chatId: UUID, text: String): UUID</b>	Invia un messaggio a una chat specifica dedicata ad un compito.	<b>context</b> HomeworkChatService::sendMessage(chatId: UUID, text: String): UUID	<b>context</b> HomeworkChatService::sendMessage(chatId: UUID, text: String): UUID

		<p><b>pre</b> HomeworkChat.allInstances()-&gt;exists(hc   hc.id = chatId and hc.completed = false)</p> <p><b>pre</b> AuthenticationService::isStudent() implies HomeworkChat.allInstances()-&gt;exists(hc   hc.id = chatId and hc.student = AuthenticationService::getStudent())</p> <p><b>pre</b> AuthenticationService::isTeacher() implies HomeworkChat.allInstances()-&gt;exists(hc   hc.id = chatId and hc.homeworkSignedHourTeachingTimeslot.signedHour.teacher = AuthenticationService::getTeacher())</p>	<p><b>post</b> Message.allInstances()-&gt;exists(m   m.text = text and m.chat.id = chatId) implies result = Message.allInstances()-&gt;select(m   m.text = text and m.chat.id = chatId)-&gt;first().id</p>
<b>+ findTeacher(chatId: UUID): Teacher</b>	Trova l'insegnante associato a una chat specifica.	<p><b>context</b> HomeworkChatService::findTeacher(chatId: UUID): Teacher</p> <p><b>pre</b> HomeworkChat.allInstances()-&gt;exists(hc   hc.id = chatId)</p>	<p><b>context</b> HomeworkChatService::findTeacher(chatId: UUID): Teacher</p> <p><b>post</b> result = HomeworkChat.allInstances()-&gt;select(hc   hc.id = chatId)-&gt;first().homeworkSignedHourTeachingTimeslot.signedHour.teacher</p>
<b>+addChat(homeworkId: Integer)</b>	Aggiunge una nuova chat per un compito specifico.	<p><b>context</b> HomeworkChatService::addChat(homeworkId: Integer)</p>	<p><b>context</b> HomeworkChatService::addChat(homeworkId: Integer)</p> <p><b>post</b> SchoolClass.allInstances()</p>

		<p><b>pre</b> Homework.allInstances()-&gt;exists(h   h.id = homeworkId)</p> <p><b>pre</b> SchoolUserService::isLoggedInTeacherClass(Homework.allInstances()-&gt;select(h   h.id = homeworkId)-&gt;first().signedHour.teachingTimeslot.classTimetable.schoolClass.id)</p>	<p>-&gt;select(sc   sc.id = Homework.allInstances()-&gt;select(h   h.id = homeworkId)-&gt;first().signedHour.teachingTimeslot.classTimetable.schoolClass.id).students-&gt;forAll(s   HomeworkChat.allInstances()-&gt;exists(hc   hc.student = s and hc.homeworkSignedHourTeachingTimeslot = Homework.allInstances()-&gt;select(h   h.id = homeworkId)-&gt;first()))</p> <p><b>post</b> HomeworkChat.allInstances()-&gt;forAll(hc   hc.homeworkSignedHourTeachingTimeslot = Homework.allInstances()-&gt;select(h   h.id = homeworkId)-&gt;first() and hc.completed = false)</p>
<p><b>+getMessageList(chatId: UUID): Sequence(MessageResponse)</b></p>	<p>Recupera l'elenco dei messaggi da una chat specifica.</p>	<p><b>context</b> HomeworkChatService::getMessageList(chatId: UUID): Sequence(MessageResponse)</p> <p><b>pre</b> HomeworkChat.allInstances()-&gt;exists(chat   chat.id = chatId) and (AuthenticationService::isLoggedInStudent() implies HomeworkChat.allInstances()-&gt;any(chat   chat.id = chatId).student = AuthenticationService::getStudent()) and</p>	<p><b>context</b> HomeworkChatService::getMessageList(chatId: UUID): Sequence(MessageResponse)</p> <p><b>post</b> result-&gt;forAll(m   m.homeworkChat.id = chatId) and result-&gt;forAll(i, j   i &lt; j implies result-&gt;at(i).dateTime &lt;= result-&gt;at(j).dateTime)</p>

		(AuthenticationService::isTeacher()) implies HomeworkChat.allInstances()->any(chat   chat.id = chatId).homeworkSignedHourTeachingTimeslot.signedHour.teacher = AuthenticationService::getTeacher())	
<b>+hasUncompletedHomeworkChat</b> (homeworkId: Integer): UUID	Controlla se è presente una chat di compiti incompiuti per un compito specifico.	<b>context</b> HomeworkChatService::hasUncompletedHomeworkChat(homeworkId: Integer): UUID <b>pre</b> Homework.allInstances()->select(h   h.id = homeworkId)->size() = 1 <b>pre</b> AuthenticationService::isStudent() implies Homework.allInstances()->select(h   h.id = homeworkId)->first().signedHour.teachingTimeslot.classTimetable.schoolClass.students->includes(AuthenticationService::getStudent())	<b>context</b> HomeworkChatService::hasUncompletedHomeworkChat(homeworkId: Integer): UUID <b>post</b> result = HomeworkChat.allInstances()->select(hc   hc.homeworkSignedHourTeachingTimeslot.id = homeworkId and hc.student = AuthenticationService::getStudent())->first().id
<b>+getStudentHomeworkChatId</b> (homeworkId: Integer, studentId: UUID): UUID	Recupera l'ID della chat relativa ai compiti di uno studente specifico.	<b>context</b> HomeworkChatService::getStudentHomeworkChatId(homeworkId: Integer, studentId: UUID): UUID <b>pre</b> SchoolUserService::isLoggedTeacherStudent(studentId) <b>pre</b> Homework.allInstances()->select(h   h.id = homeworkId)->size() = 1	<b>context</b> HomeworkChatService::getStudentHomeworkChatId(homeworkId: Integer, studentId: UUID): UUID <b>post</b> result = HomeworkChat.allInstances()->select(hc   hc.homeworkSignedHourTeachingTimeslot.id = homeworkId and hc.student.id = studentId)->first().id

<b>+complete(signHourId: Integer): HomeworkChatResponse</b>	Segna tutte le chat di un compito specifico come completate per l'orario firmato specificato.	<b>context</b> HomeworkChatService:: complete(signHourId: Integer): HomeworkChatResponse <b>pre</b> UserService:: getLoggedTeacher().signedTimeslots.exists(t   t.signedHours.exists(s   s.id = id))	<b>context</b> HomeworkChatService:: complete(signHourId: Integer): HomeworkChatResponse <b>post</b> result = UserService:: getLoggedTeacher().signedTimeslots.select(t   t.signedHours.exists(s   s.id = id)).signedHours.select(s   s.id = id).homeworks.forAll(h   h.homeworkChats.forAll(c   c.completed = true))
<b>+completeWithFeedback(signHourId: Integer, text: String): HomeworkChatResponse</b>	Segna tutte le chat come completate e aggiunge un messaggio finale con feedback.	<b>context</b> HomeworkChatService:: completeWithFeedback(signHourId: Integer, text: String): HomeworkChatResponse <b>pre</b> text.size() > 0 <b>pre</b> UserService:: getLoggedTeacher().signedTimeslots.exists(t   t.signedHours.exists(s   s.id = id))	<b>context</b> HomeworkChatService:: completeWithFeedback(signHourId: Integer, text: String): HomeworkChatResponse <b>post</b> let homeworkChats = UserService:: getLoggedTeacher().signedTimeslots.select(t   t.signedHours.exists(s   s.id = id)).signedHours.select(s   s.id = id).homeworks.collect(h   h.homeworkChats) <b>post</b> result = homeworkChats.forAll(c   c.completed = True) <b>post</b> homeworkChats.forAll(c   c.getMessageList().getLast().text = text)



### 3.1.19 TicketChatService

Servizio per la gestione della chat di supporto o ticket.

Nome	Descrizione	Precondizione	Post condizione
<b>+deleteTicket(id: UUID)</b>	Elimina un ticket esistente.	<b>context</b> TicketChatService::deleteTicket(id: UUID) <b>pre</b> -	<b>context</b> TicketChatService::deleteTicket(id: UUID) <b>post</b> -
<b>+getTickets(): Sequence(TicketResponse)</b>	Recupera un elenco di tutti i ticket.	<b>context</b> TicketService::getTickets(): Sequence(TicketResponse) <b>pre</b> AuthenticationService::isTeacher() implies Ticket.allInstances()->exists(t   t.teacher = AuthenticationService::getTeacher() and t.closed = false and t.solved = false) <b>pre</b> AuthenticationService::isParent() implies Ticket.allInstances()->exists(t   t.parent = AuthenticationService::getParent() and t.closed = false and t.solved = false) <b>pre</b> AuthenticationService::isSecretary() implies School.allInstances()->exists(s   s.secretaries->includes(AuthenticationService::getSecretary())) and Ticket.allInstances()->exists(t  (t.teacher.school = School.allInstances()->select(s   s.secretaries->includes(AuthenticationService::getSecretary()))-	<b>context</b> TicketService::getTickets(): Sequence(TicketResponse) <b>post</b> result->forall(t   t.datetime <> null) <b>post</b> result->forall(i, j   i < j implies result->at(i).datetime <= result->at(j).datetime)

		<pre> &gt;first() or t.parent.school = School.allInstances()- &gt;select(s   s.secretaries- &gt;includes(Authentication Service::getSecretary()))- &gt;first()) </pre>	
<b>+sendMessage(ticketId: UUID, text: String): UUID</b>	Invia un messaggio relativo a un ticket specifico.	<p><b>context</b>          MessageService::sendMessage(ticketId: UUID, text: String): UUID</p> <p><b>pre</b>          Ticket.allInstances()-&gt;exists(t   t.id = ticketId and not t.closed and not t.solved)</p> <p><b>post</b>          AuthenticationService::isTeacher() or          AuthenticationService::isParent() or          AuthenticationService::isSecretary()</p>	<p><b>context</b>          MessageService::sendMessage(ticketId: UUID, text: String): UUID</p> <p><b>post</b>          let createdMessage: Message =          Message.allInstances()-&gt;any(m   m.ticket.id = ticketId and m.text = text)          in Message.allInstances()-&gt;exists(m   m = createdMessage and m.ticket.id = ticketId and m.text = text and (AuthenticationService::isTeacher() implies m.teacher.id = AuthenticationService::getTeacher().id) and (AuthenticationService::isParent() implies m.parent.id = AuthenticationService::getParent().id) and (AuthenticationService::isSecretary() implies m.secretary.id = AuthenticationService::getSecretary().id))          post: result =          Message.allInstances()-&gt;any(m   m.ticket.id = ticketId and m.text = text).id</p>

<b>+ createTicket(title: String)</b>	Crea un nuovo ticket con il titolo.	<b>context</b> TicketService::createTicket(title: String) <b>pre</b> AuthenticationService::isTeacher() or AuthenticationService::isParent()	<b>context</b> TicketService::createTicket(title: String) <b>post</b> Ticket.allInstances()->exists(t   t.title = title.substring(1,title.size() - 1) and t.category = "Category"and t.datetime <> null and (AuthenticationService::isTeacher() implies t.teacher.id = AuthenticationService::getTeacher().id) and (AuthenticationService::isParent() implies t.parent.id = AuthenticationService::getParent().id))
<b>+getMessageList(ticket: Ticket): Sequence(MessageResponse)</b>	Recupera i messaggi associati a un ticket specifico.	<b>context</b> TicketService::getMessageList (ticket: Ticket): Sequence(MessageResponse) <b>pre</b> Ticket.allInstances()->exists(t   t = ticket)	<b>context:</b> TicketService::getMessageList (ticket: Ticket): Sequence(MessageResponse) <b>post</b> let messages: Sequence(Message) = Message.allInstances()->select(m   m.ticket = ticket)->sortedBy(m   m.dateTime) in result->forAll(r   messages->exists(m   r.messageId = m.id and r.content = m.content and r.dateTime = m.dateTime)) and result->forAll(i, j   i < j implies result->at(i).dateTime <= result->at(j).dateTime)

+ closeUnsolved(id: UUID): TicketResponse	Chiude un ticket aperto impostandolo come "non risolto".	<b>context</b> TicketChatService::closeUnsolved(id: UUID): TicketResponse <b>pre</b> ChatService::getById(id) <> null <b>pre</b> ChatService::getById(id).status = "open"	<b>context</b> TicketChatService::closeUnsolved(id: UUID): TicketResponse <b>pre</b> result.body = ChatService::getById(id).status = "unresolved"
+ closeAndSolve(id: UUID): TicketResponse	Chiude un ticket aperto impostandolo come "risolto".	<b>context</b> TicketChatService::closeAndSolve(id: UUID): TicketResponse <b>pre</b> ChatService::getById(id) <> null <b>pre</b> ChatService::getById(id).status = "open"	<b>context</b> TicketChatService::closeAndSolve(id: UUID): TicketResponse <b>pre</b> result.body = ChatService::getById(id).status = "close"

### 3.1.20ClassManagementService

Servizio per la gestione delle classi e della didattica.

Nome	Descrizione	Precondizione	Post condizione
<b>+addStudyPlan(classId: Long, studyPlanId: Long)</b>	Aggiunge un piano di studi a una specifica classe identificata da classId.	<b>context</b> ClassManagementService :: addStudyPlan(classId: Long, studyPlanId: Long) <b>pre</b> self.getByld(classId) <> null	<b>context</b> ClassManagementService :: addStudyPlan(classId: Long, studyPlanId: Long) <b>post</b> result.body = self.getByld(classId).studyPlan.id = studyPlanId
<b>+addStudents(studentsId: Bag(UUID), id: Long)</b>	Aggiunge una lista di studenti a una classe specifica.	<b>context</b> ClassManagementService ::AddStudents(studentsId: Bag(UUID), id: Long) <b>pre</b> self.getByld(id) <> null <b>pre</b> studentsId.forAll(studId   UserService::getByld(studId) <> null)	<b>context</b> ClassManagementService ::AddStudents(studentsId: Bag(UUID), id: Long) <b>post</b> result.body = studentsId.forAll(studId   self.getByld(id).students.exists(s   s.id = studId))
<b>+addTeacher(request: TeacherClassRequest)</b>	Aggiunge un insegnante a una classe specifica in base ai dettagli forniti nella richiesta.	<b>context</b> ClassManagementService :: addTeacher(request: TeacherClassRequest) <b>pre</b> self.getByld(id) <> null <b>pre</b> UserService::getByld(request.teacherId) <> null	<b>context</b> ClassManagementService :: addTeacher(request: TeacherClassRequest) <b>post</b> result.body = self.getByld(id).teachers.exists(t   t.id = request.teacherId)
<b>+getYear():Year</b>	Recupera l'anno scolastico corrente.	<b>context</b> ClassManagementService ::getYear(): Year <b>pre</b> -	<b>context</b> ClassManagementService ::getYear(): Year <b>post</b> let currentMonth: Integer = self.getCurrentMonth() in let currentYear: Integer = self.getCurrentYear() in (currentMonth <= 9 implies result.value =

			currentYear) and (currentMonth > 9 implies result.value = currentYear - 1)
<b>+ getClasses():</b> Bag(SchoolClassResponse)	Recupera un elenco di tutte le classi.	<b>context</b> ClassManagementService::getClasses(): Bag(SchoolClassResponse) <b>pre</b> AuthenticationService::getSecretary().isPresent() or AuthenticationService::getTeacher().isPresent()	<b>context</b> ClassManagementService::getClasses(): Bag(SchoolClassResponse) <b>post</b> let user: SchoolUser = if AuthenticationService::getSecretary().isPresent() then AuthenticationService::getSecretary() else AuthenticationService::getTeacher() endif inresult->forAll(c   SchoolClass.allInstances()->exists(sc  sc.school = user.school and c.id = sc.id and c.year = sc.year and c.letter = sc.letter and c.number = sc.number))
<b>+getStudents(classID : Integer):</b> Bag(SchoolClassStudentResponse)	Recupera un elenco di studenti in una classe specifica.	<b>context</b> ClassManagementService::getStudents(classId: Integer): Bag(SchoolClassStudentResponse) <b>pre</b> SchoolUserService::isLoggedInTeacherClass(classId)	<b>context</b> ClassManagementService::getStudents(classId: Integer): Bag(SchoolClassStudentResponse) <b>post</b> let students: Sequence(SchoolUser) = SchoolClass.allInstances()->select(sc   sc.id = classId).students->sortedBy(s   s.surname) in result->forAll(r   students->exists(s   r.id = s.id and r.name = s.name

			and r.surname = s.surname and r.schoolId = s.school.id))
<b>+ getTeachings():</b> Bag(TeachingResponse)	Recupera un elenco di tutti gli insegnamenti.	<b>context</b> ClassManagementService::getTeachings(): Bag(TeachingResponse) <b>pre</b> -	<b>context</b> ClassManagementService::getTeachings(): Bag(TeachingResponse) <b>post</b> result->size() <= 20 and result->forall(r   Teaching.allInstances()->exists(t   r.teacherUsername = t.teacher.username and r.subjectTitle = t.subjectTitle.title))
<b>+schoolClassResponse(request: SchoolClassRequest):</b> SchoolClassResponse	Crea o recupera una classe scolastica in base alla richiesta fornita.	<b>context</b> ClassManagementService::schoolClassResponse(request: SchoolClassRequest): SchoolClassResponse <b>pre</b> AuthenticationService::getSecretary().isPresent()	<b>context</b> ClassManagementService::schoolClassResponse(request: SchoolClassRequest): SchoolClassResponse <b>post</b> let newClass: SchoolClass = SchoolClass.allInstances()->any(sc   sc.number = request.number() and sc.letter = request.letter() and sc.year = self.getYear() and sc.school = AuthenticationService::getSecretary().school) in result.id = newClass.id and result.number = newClass.number and result.letter = newClass.letter and result.year =

			newClass.year and result.schoolId = newClass.school.id
<b>+addTeaching</b> (teacheruid: UUID, teachingRequest: TeachingRequest)	Aggiunge una nuova attività didattica per un insegnante specifico.	<b>context</b> ClassManagementService ::addTeaching(teacheruid : UUID, teachingRequest: TeachingRequest) <b>pre</b> Teacher.allInstances()->exists(t   t.id = teacheruid) <b>pre</b> Subject.allInstances()->exists(s   s.title = teachingRequest.subjectTitle())	<b>context</b> ClassManagementService ::addTeaching(teacheruid : UUID, teachingRequest: TeachingRequest) <b>post</b> Teaching.allInstances()->exists(te   te.id.teacherId = Teacher.allInstances()->any(t   t.id = teacheruid).id and te.id.subjectTitle = teachingRequest.subjectTitle() and te.teacher.id = teacheruid and te.subjectTitle.title = teachingRequest.subjectTitle() and te.typeOfActivity =teachingRequest.activityType())
<b>+ getSubject</b> (): Bag(String)	Recupera un elenco di tutti le materie.	<b>context</b> ClassManagementService ::getSubject(): Bag(String) <b>pre</b> -	<b>context</b> ClassManagementService ::getSubject(): Bag(String) <b>post</b> result = Subject.allInstances()->collect(s   s.title)
<b>+getTeacherClasses</b> (teacherid: UUID): Bag(SchoolClassResponse)	Recupera un elenco delle classi assegnate a un insegnante specifico.	<b>context</b> ClassManagementService ::getTeacherClasses(teacherId: UUID): Bag(SchoolClassResponse) <b>pre:</b> Teacher.allInstances()->exists(t   t.id = teacherId)	<b>context</b> ClassManagementService ::getTeacherClasses(teacherId: UUID): Bag(SchoolClassResponse) <b>post</b> let teacher: Teacher = Teacher.allInstances()->any(t   t.id = teacherId) in



			let classes: Sequence(SchoolClass) = TeacherClass.allInstances( )->select(tc   tc.teacher = teacher).schoolClass in result->forAll(r   classes- >exists(c   r.id = c.id and r.year = c.year and r.letter = c.letter and r.number = c.number and r.additionalInfo = ""))
<b>+addTeacherToClass</b> ( uuid: UUID, request: AddToClassRequest)	Aggiunge un insegnante a una classe specifica.	<b>context</b> ClassManagementService ::addTeacherToClass(uuid: UUID, addToClassRequest: AddToClassRequest) <b>pre:</b> Teacher.allInstances()- >exists(t   t.id = uuid) <b>post:</b> SchoolClass.allInstances() ->exists(sc   sc.id = addToClassRequest.classId())	<b>context:</b> ClassManagementService ::addTeacherToClass(uuid: UUID, addToClassRequest: AddToClassRequest) <b>post:</b> TeacherClass.allInstances( )->exists(tc   tc.id.teacherId = uuid and tc.id.schoolClassId = addToClassRequest.classId() and tc.teacher.id = uuid and tc.schoolClass.id = addToClassRequest.classId() and tc.isCoordinator =addToClassRequest.isCoordinator())
<b>+removeClass</b> (teacher uuid: String, schoolClassId: Integer)	Rimuove una classe associata a un insegnante specifico.	<b>context</b> ClassManagementService ::removeClass(teacheruuid: String, schoolClassId: Integer) <b>pre</b> TeacherClass.allInstances( )->exists(tc   tc.id.teacherId = UUID::fromString(teacher uuid) and tc.id.schoolClassId = schoolClassId)	<b>context</b> ClassManagementService ::removeClass(teacheruuid: String, schoolClassId: Integer) <b>post</b> not TeacherClass .allInstances()->exists(tc   tc.id.teacherId = UUID::fromString(teacher uuid) and tc.id.schoolClassId = schoolClassId)

<b>+getClassTeachings(classID: Integer): Bag(TeachingResponse)</b>	Recupera l'elenco degli insegnamenti associati a una classe specifica.	<b>context</b> ClassManagementService ::getClassTeachings(classID: Integer): Bag(TeachingResponse) <b>pre</b> SchoolClass.allInstances() ->exists(sc   sc.id = classID)	<b>context</b> ClassManagementService ::getClassTeachings(classID: Integer): Bag(TeachingResponse) <b>post</b> let teacherList: Sequence(Teacher) = TeacherClass.allInstances() ->select(tc   tc.schoolClass.id = classID).teacher in result- >forall(r   teacherList- >exists(t   r.teacherUsername = t.username and r.teachings ->forall(teachingString  t.teachings- >exists(teaching   teachingString = teaching.subjectTitle.title. concat('-' ').concat(teaching.typeOfActivity))))))
--	--	---	---

### 3.1.21 OrientationService

Servizio per l'orientamento degli studenti per comunicare con un modulo Python.

Nome	Descrizione	Precondizione	Post condizione
<b>+attitude(student: Student): String</b>	Restituisce una valutazione dell'atteggiamento di uno studente.	<b>context</b> OrientationService::attitude(student: Student): String <b>pre</b> not student.ocllsUndefined()	<b>context</b> OrientationService::attitude(student: Student): String <b>post</b> -
<b>+getCategory(text: String): Ticket.Category</b>	Determina la categoria del ticket in base al testo fornito.	<b>context</b> OrientationService::getCategory(text: String): Ticket.Category <b>pre</b> text.size() > 10	<b>context</b> OrientationService::getCategory(text: String): Ticket.Category <b>post</b> -

### 3.1.22 FileService

Servizio per la gestione dei file, probabilmente per caricamento degli allegati.

Nome	Descrizione	Precondizione	Post condizione
<b>+uploadFile(file: File): String</b>	Carica un file e restituisce il suo identificatore o percorso.	<b>context</b> FileService::uploadFile(file: File): String <b>pre</b> not file.size() > 16 Mb	<b>context</b> FileService::uploadFile(file: File): String <b>pre</b> result = file.name()
<b>+ delete(filename: String): Boolean</b>	Elimina un file in base al suo nome.	<b>context</b> FileService::: delete(filename: String): Boolean <b>pre</b> not student.ocllsUndefined()	<b>context</b> FileService::: delete(filename: String): Boolean <b>pre</b> not student.ocllsUndefined()

### 3. Specifica DTO e Documentazione REST

L'API RESTful del sistema è documentata utilizzando Swagger UI, uno strumento interattivo che consente di esplorare e testare le API direttamente dal browser.

La documentazione è accessibile tramite l'interfaccia utente di Swagger al seguente percorso:

/swagger-ui/index.html

The screenshot displays the Swagger UI interface for an OpenAPI definition. At the top, the Swagger logo and the path `/v3/api-docs` are visible, along with an "Explore" button. Below this, the "OpenAPI definition" section shows the version `v0` and the specification `OAS 3.0`. A "Servers" section indicates the base URL: `https://astromark.it - Generated server url`.

The main content area lists the endpoints for two controllers:

- ticket-controller**
  - POST** `/api/tickets/{ticketId}/addMessage`: Send a message to a ticket
  - POST** `/api/tickets/newTicket`: Create a new ticket
  - PATCH** `/api/tickets/{ticketId}/closeUnsolved`: Close an unsolved ticket
  - PATCH** `/api/tickets/{ticketId}/closeAndSolve`: Close and solve a ticket
  - GET** `/api/tickets/{ticketId}/messages`: Get messages for a ticket
  - GET** `/api/tickets/ticket`: Get all tickets
- teacher-controller**
  - POST** `/api/teachers/create`: Create a teacher account
  - GET** `/api/teachers/{teacheruuid}/teachings`: Retrieve teacher details
  - GET** `/api/teachers/teachings`: Retrieve teaching subjects
  - GET** `/api/teachers/schoolClasses`: Retrieve school classes
  - GET** `/api/teachers/all`: Retrieve all teachers

### 3.1. Componenti React

L'architettura modulare di **AstroMark** consente un ampio riutilizzo di componenti React per ottimizzare lo sviluppo e ridurre la duplicazione del codice. I componenti sono progettati per essere generici e configurabili, in modo da soddisfare diverse esigenze funzionali mantenendo una coerenza visiva e strutturale. Di seguito è riportata una lista parziale dei componenti React, con ulteriori opportunità di riuso identificate.

#### Componenti Comuni

- **HomePageFooter:** footer della pagina principale.
- **HomePageNavbar:** barra di navigazione della pagina principale del sito.
- **Help:** pagina di aiuto e supporto con informazioni utili e guide per l'utilizzo dell'applicazione.
- **Login:** pagina di accesso per gli utenti, con campi per inserire le credenziali.
- **Settings:** pagina per gestire le impostazioni dell'account utente, come password, notifiche o preferenze.
- **Accordion(NotViewable/Viewable):** componenti "a fisarmonica" che permettono di mostrare/nascondere contenuti. La versione "NotViewable" li nasconde inizialmente, mentre la versione "Viewable" li mostra subito.
- **CustomTableComponents:** componenti personalizzati per la creazione di tabelle.
- **GridList:** componente per visualizzare elementi in una griglia, utile per presentare dati in modo organizzato e compatto.
- **ListGeneric:** componente versatile per visualizzare qualsiasi tipo di lista di elementi.
- **ListBooked:** componente specifico per visualizzare elementi che sono stati prenotati, come i colloqui con i docenti.
- **ChatComponent:** componente principale per la chat, che gestisce l'interfaccia della conversazione tra gli utenti.
- **ChatHomework:** componente specifico per la chat dedicata ai compiti, per facilitare la comunicazione tra studenti e docenti.

- **TicketComp:** componente per visualizzare e gestire i ticket di supporto, con informazioni dettagliate e possibilità di interazione.
- **TicketCreation:** componente per creare nuovi ticket di supporto, con campi per descrivere il problema e richiedere assistenza.
- **TicketList:** componente per visualizzare una lista di ticket di supporto.

### Componenti Specifici per la Segreteria

- **SecretaryDashboard:** dashboard specifica per la segreteria, con informazioni utili per il loro lavoro.
- **SecretaryDashboardNavbar:** barra di navigazione principale della dashboard specifica alla segreteria, che permette di accedere alle diverse sezioni.
- **SecretarySideNav:** barra di navigazione laterale specifica per la segreteria, per accedere alle funzionalità dedicate.
- **ManageClass:** pagina per la gestione delle classi, con possibilità di modificare informazioni, aggiungere/rimuovere studenti o docenti.
- **ManageTeacher:** pagina per la gestione dei docenti, con possibilità di visualizzare informazioni, modificare dati o assegnare classi.
- **ManageTimetable:** pagina per la gestione dell'orario, con possibilità di definire orari, assegnare docenti e aule.
- **ClassSchedule:** pagina per la gestione dell'orario delle classi, con possibilità di definire orari, assegnare docenti e aule.
- **CreateClass:** pagina per la creazione di nuove classi, con campi per inserire informazioni su nome, descrizione e studenti.
- **DetailsSchoolClass:** pagina di dettaglio di una classe specifica, con informazioni su studenti, docenti, materie e orario.
- **TeacherDetails:** pagina di dettaglio di un docente specifico, con informazioni su classi, materie e orario.
- **SecretaryTicket:** pagina per la gestione dei ticket di supporto da parte della segreteria.

## Componenti Specifici per Studenti e Genitori

- **Dashboard:** pagina principale con una panoramica delle informazioni più importanti per lo studente.
- **StudentParentSideNav:** barra di navigazione laterale specifica per studenti e genitori, per accedere alle funzionalità dedicate.
- **DashboardNavbar:** barra di navigazione principale della dashboard specifica per studenti e genitori, che permette di accedere alle diverse sezioni.
- **AccountMenu:** menu a tendina per selezionare l'account degli studenti associati
- **ArchiveMenu:** menu a tendina per selezionare l'anno scolastico.
- **HomeworkList:** componente per visualizzare una lista di compiti, con dettagli su scadenza, materia e stato.
- **ClassActivity:** pagina per visualizzare le attività svolte e i compiti assegnati della classe.
- **AbsenceDelay:** pagina per la gestione di assenze e ritardi, con possibilità di giustificarli o visualizzarne lo storico.
- **JustifiableList:** componente per visualizzare liste di elementi che possono essere giustificati, come assenze e ritardi.
- **Report:** pagina per la visualizzazione delle pagelle scolastiche.
- **Mark:** pagina per la visualizzazione dei voti e della media, con la possibilità di filtrarli per materia.
- **ClassTimeslot:** componente per visualizzare un singolo slot orario di lezione, con dettagli su materia, orario e docente.
- **Timetable:** pagina per la visualizzazione dell'orario, con informazioni su materia, orario e docente per ogni slot.
- **Communication:** pagina dedicata alla visualizzazione delle comunicazioni e avvisi della classe.
- **Note:** pagina per visualizzare le note disciplinari.

## Componenti Specifici per Genitori

- **Reception:** pagina per la gestione del ricevimento, con informazioni su orari, prenotazioni e contatti.
- **Ticket:** pagina per la gestione dei ticket di supporto, con possibilità di visualizzare, rispondere o chiudere le richieste.

## Componenti Specifici per Docenti

- **TeacherDashboard:** dashboard specifica per il docente, con informazioni utili per il loro lavoro.
- **TeacherDashboardNavbar:** barra di navigazione principale della dashboard specifica per il docente, che permette di accedere alle diverse sezioni.
- **TeacherSideNav:** barra di navigazione laterale specifica per il docente, per accedere alle funzionalità dedicate.
- **TeachingMenu:** menu a tendina per selezionare l'insegnamento.
- **SchoolClass:** pagina per la selezione della classe.
- **Attendance:** pagina per la gestione delle presenze, con possibilità di registrare e visualizzare assenze.
- **DelayComponent:** componente per visualizzare e gestire ritardi, con possibilità di inserirli.
- **ClassAgenda:** pagina per la gestione dell'agenda di classe, con la possibilità di vedere l'orario scolastico.
- **SignHour:** pagina per la registrazione delle ore di lezione, con la possibilità di inserire le attività svolte e assegnare i compiti.
- **Ratings:** pagina per visualizzare valutazioni date in una data specificata.
- **RatingComponent:** componente per visualizzare e inserire le valutazioni.
- **EveryRatings:** pagina per visualizzare tutte le valutazioni date in una specifica materia.
- **HomeworkChat:** pagina dedicata alla chat con gli studenti.



- **Note:** pagina per visualizzare e/o inserire note disciplinari.
- **Communication:** pagina dedicata alla gestione delle comunicazioni e avvisi della classe.
- **CommunicationComponent:** componente dedicato a inserire nuove comunicazioni e avvisi ad una classe.
- **Reception:** pagina per la gestione del ricevimento, con informazioni su orari, prenotazioni e contatti.

## 4. Glossario

Termine	Definizione
<b>Singleton</b>	Un oggetto creato in un'unica istanza globale e condivisa, utile quando è necessario un punto di accesso unificato a una risorsa. In Spring, i bean per default sono singleton, garantendo che i servizi condivisi come i DataSource vengano istanziati una sola volta.
<b>Facade</b>	Fornisce un'interfaccia semplificata per un insieme complesso di classi o funzionalità, agevolando l'uso di sistemi complessi. In Spring, i servizi possono fungere da facciata verso i repository e le altre logiche, offrendo un unico punto di accesso alle operazioni sul dominio. In React, invece, un componente raccoglie dati da diverse API e li organizza per la presentazione.
<b>Adapter</b>	Permette a classi con interfacce incompatibili di lavorare insieme, convertendo l'interfaccia di una classe in un'altra attesa dal client. In Spring Boot può essere usato per integrare servizi esterni che forniscono dati con formati diversi, adattandoli a DAO o DTO esistenti.
<b>Bridge</b>	Separa un'astrazione dalla sua implementazione, permettendo loro di variare indipendentemente. In Spring, si può avere un'astrazione di servizio e varie implementazioni iniettabili tramite bean, facilitando la sostituzione e l'espansione del comportamento.
<b>Builder</b>	Fornisce un modo flessibile per costruire oggetti complessi passo dopo passo, mantenendo il codice client pulito. In Spring Boot, può essere sfruttato ad esempio per costruire entità o DTO complessi a partire da informazioni parziali senza incorrere in costruttori enormi. In React, per creare set di proprietà o configurazioni di routing complesse in modo fluido e leggibile.
<b>Abstract Factory</b>	Fornisce un'interfaccia per creare famiglie di oggetti correlati tra loro, senza specificare le classi concrete. In Spring Boot si possono configurare bean differenti a seconda del profilo attivo.
<b>Chain of Responsibility</b>	Delega la richiesta lungo una catena di handler, dove ognuno può gestire la richiesta o passarla avanti. In Spring Boot, può essere implementata in filtri per processare richieste HTTP in sequenza.

<b>DTO (Data Transfer Object)</b>	Strutture dati semplici, senza logica di business, usate per trasferire informazioni tra livelli o servizi. In Spring Boot, i DTO sono comunemente utilizzati nei controller per scambiare dati con il client React, mantenendo separata la logica dal modello di dominio.
<b>DAO (Data Access Object)</b>	Isola i dettagli di accesso ai dati (query SQL, mapping) all'interno di classi dedicate, semplificando la logica di business. In Spring, i repository (basati su JPA o altri driver) ricalcano il pattern DAO, fornendo un'interfaccia pulita per le operazioni di persistenza.
<b>CRUD</b>	Create, Read, Update, Delete – Operazioni di base per la gestione dei dati in un'applicazione.
<b>HTTP</b>	HyperText Transfer Protocol – Protocollo di trasferimento dati utilizzato per le comunicazioni web.
<b>JPA</b>	Java Persistence API – Specifica Java per la gestione della persistenza dei dati tra le applicazioni Java e i database relazionali.
<b>JSDoc</b>	JavaScript Documentation – Strumento di documentazione per il linguaggio JavaScript, simile a Javadoc per Java.
<b>Javadoc</b>	Strumento di documentazione per il linguaggio Java, utilizzato per generare documentazione API a partire dal codice sorgente.
<b>JWT</b>	JSON Web Token – Standard aperto per la trasmissione sicura di informazioni tra le parti come oggetti JSON.
<b>OCL</b>	Object Constraint Language – Linguaggio utilizzato per specificare restrizioni e vincoli nei modelli UML.
<b>REST</b>	Representational State Transfer – Stile architetturale per la progettazione di servizi web che utilizza le operazioni HTTP.
<b>UML</b>	Unified Modeling Language – Linguaggio di modellazione standardizzato utilizzato per specificare, visualizzare, costruire e documentare gli artefatti di sistemi software.
<b>API</b>	Application Programming Interface – Insieme di regole e specifiche che le applicazioni possono seguire per comunicare tra loro.
<b>CSS</b>	Cascading Style Sheets – Linguaggio utilizzato per descrivere la presentazione di documenti HTML o XML.
<b>Spring Boot</b>	Framework Java per lo sviluppo di applicazioni Spring con configurazioni automatiche e componenti pronti all'uso.
<b>React</b>	Libreria JavaScript per la costruzione di interfacce utente interattive e component-based.

<b>TypeScript</b>	Superinsieme tipizzato di JavaScript che aggiunge tipi statici e altre funzionalità al linguaggio.
<b>Lombok</b>	Libreria Java che riduce il boilerplate di codice generando automaticamente getter, setter, costruttori, e altri metodi comuni attraverso annotazioni.
<b>@RestController</b>	Annotazione di Spring per definire controller REST che gestiscono le richieste HTTP e producono risposte JSON o XML.
<b>@ControllerAdvice</b>	Annotazione di Spring che permette di gestire globalmente le eccezioni e aggiungere comportamenti trasversali ai controller.
<b>@ExceptionHandler</b>	Annotazione di Spring per definire metodi che gestiscono specifiche eccezioni lanciate dai controller.
<b>Formik</b>	Libreria per la gestione dei form in React, semplificando la gestione dello stato e delle validazioni dei form.
<b>Yup</b>	Libreria JavaScript per la validazione degli schemi dei dati, spesso utilizzata insieme a Formik per la validazione dei form in React.
<b>Jakarta EE</b>	Progetto open-source che fornisce un set di specifiche per lo sviluppo di applicazioni enterprise in Java.