Università degli Studi di Salerno

Corso di Ingegneria del Software

AstroMark
Object Design
Versione 1.0

Data: 15/12/2024

Partecipanti:

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

Scritto da:	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

Indice

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.

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

authentication

- → **userAuthentication**
- → **jwtService**

user

→ **commons**

- → **entity**
- → **dto**
- → **service**
- → **controller**

→ **student**

- → **entity**
- → **dto**
- → **service**
- → **exception**
- → **repository**
- → **controller**

→ **teacher**

- → **entity**
- → **dto**
- → **service**
- → **exception**
- → **repository**
- → **controller**

→ **secretary**

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

→ **parent**

- → **entity**
- → **dto**
- → **service**
- → **config**
- → **repository**
- → **controller**

school

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

agenda

- → **commons**
  - → **entity**
  - → **dto**
- → **reception**
  - → **entity**
  - → **dto**
  - → **service**
  - → **exception**
  - → **repository**
  - → **controller**
- → **schoolClass**
  - → **entity**
  - → **dto**
  - → **service**
  - → **exception**
  - → **repository**
  - → **controller**

behavior

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

attendance

- → **entity**
- → **dto**
- → **service**
- → **exception**
- → **repository**
- → **controller**

rating

- → **entity**
- → **dto**
- → **service**
- → **exception**
- → **repository**
- → **controller**

classwork

- → **entity**
- → **dto**
- → **service**
- → **exception**
- → **repository**
- → **controller**

communication

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

classManagement

- → **entity**
- → **dto**
- → **service**
- → **exception**
- → **repository**
- → **controller**

chat

- → **entity**
- → **dto**
- → **service**
- → **exception**
- → **repository**
- → **config**
- → **controller**

orientation

- → **service**
- → **controller**

commons

- → **controller**
- → **service**
- → **config**

2.1.1 Elenco Classi e Interfacce di tipo Service e Controller

File	Package	Descrizione
CrudController<T, R, RS, ID>	it.astromark.commons.controller	Controller generico per gestire operazioni CRUD su entità, integrando repository e servizi. Facilita la gestione uniforme di dati e richieste HTTP.
CrudService<T, R, RS, ID>	it.astromark.commons.service	Servizio generico per la logica delle operazioni CRUD, integrato con repository e servizi personalizzati.
UserController<T>	it.astromark.user.commons.controller	Controller specifico per la gestione delle operazioni CRUD relative agli utenti, estendibile per esigenze personalizzate.
UserService<T>	it.astromark.user.commons.service	Servizio dedicato alla logica per la gestione degli utenti, supportando operazioni CRUD e funzionalità specifiche.
AuthController	it.astromark.userAuthentication.controller	Controller per la gestione delle operazioni di autenticazione e autorizzazione degli utenti, come login e registrazione.
StudentController	it.astromark.user.student.controller	Controller per la gestione delle operazioni relative agli studenti, incluse funzionalità specifiche e CRUD.
StudentService	it.astromark.user.student.service	Servizio per la gestione della logica relativa agli studenti, con supporto per operazioni CRUD e funzionalità personalizzate.
SecretaryController	it.astromark.user.secretary.controller	Controller per la gestione delle operazioni relative ai segretari, incluse funzionalità specifiche e CRUD.
SecretaryService	it.astromark.user.secretary.service	Servizio per la gestione della logica relativa alla segreteria, con supporto per operazioni CRUD e funzionalità personalizzate.

ParentController	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.
ParentService	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.
TeacherController	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.
TeacherService	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.
SchoolController	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.
SchoolService	It.astromark.school.service	Servizio per la gestione della logica della scuola.
ReceptionAgendaController	it.astromark.agenda.reception.controller	Controller per la gestione delle operazioni relative all'agenda dei ricevimenti, incluse funzionalità CRUD e pianificazione degli appuntamenti.
ReceptionAgendaService	it.astromark.agenda.reception.service	Servizio per la gestione della logica dell'agenda dei ricevimenti, supportando operazioni CRUD e pianificazione personalizzata.
ClassAgendaController	it.astromark.agenda.schoolClass.controller	Controller per la gestione delle operazioni relative all'agenda delle classi, incluse funzionalità CRUD e organizzazione degli eventi

		scolastici.
ClassAgendaService	it.astromark.agenda.schoolClass.service	Servizio per la logica relativa all'agenda delle classi, gestendo operazioni CRUD e organizzazione degli eventi scolastici.
TeachingTimeslotController	it.astromark.agenda.schoolClass.controller	Controller per la gestione delle operazioni sulle fasce orarie di insegnamento, incluse funzionalità CRUD e pianificazione delle lezioni
TeachingTimeslotService	it.astromark.agenda.schoolClass.service	Servizio per la gestione della logica delle fasce orarie di insegnamento, supportando operazioni CRUD e pianificazione delle lezioni.
MarkController	it.astromark.mark.controller	Controller per la gestione delle operazioni relative ai voti, incluse funzionalità CRUD e visualizzazione dei risultati degli studenti.
MarkService	it.astromark.mark.service	Servizio per la logica relativa ai voti, gestendo operazioni CRUD e calcoli statistici sui risultati degli studenti.
NoteController	it.astromark.behavior.controller	Controller per la gestione delle operazioni relative alle note disciplinari, incluse funzionalità CRUD e visualizzazione.
NoteService	it.astromark.behavior.service	Servizio per la gestione della logica delle note disciplinari, supportando operazioni CRUD e analisi comportamentali.
JustifiableController<T extends JustifiableEntity, R, RS, ID>	it.astromark.attendance.controller	Controller generico per gestire operazioni su entità giustificabili, come assenze, con supporto a operazioni CRUD e logica personalizzata.
JustifiableService<T extends JustifiableEntity, R, RS, ID>	it.astromark.attendance.service	Servizio generico per la gestione della logica delle entità giustificabili, supportando operazioni CRUD e funzionalità personalizzate per giustificazioni.

AbsenceController	it.astromark.attendance.controller	Controller per la gestione delle operazioni relative alle assenze, incluse funzionalità CRUD e la gestione delle giustificazioni.
AbsenceService	it.astromark.attendance.service	Servizio per la logica relativa alle assenze, gestendo operazioni CRUD e la gestione delle giustificazioni.
DelayController	it.astromark.attendance.controller	Controller per la gestione delle operazioni relative ai ritardi, incluse funzionalità CRUD e la registrazione delle giustificazioni.
DalayService	it.astromark.attendance.service	Servizio per la gestione della logica dei ritardi, supportando operazioni CRUD e la gestione delle giustificazioni.
HomeworkController	it.astromark.classwork.controller	Controller per la gestione delle operazioni relative ai compiti, incluse funzionalità CRUD e assegnazione degli esercizi agli studenti.
HomeworkService	it.astromark.classwork.service	Servizio per la gestione della logica dei compiti, supportando operazioni CRUD e l'assegnazione e la valutazione degli esercizi.
ClassActivityController	it.astromark.classwork.controller	Controller per la gestione delle operazioni relative alle attività in classe, incluse funzionalità CRUD e pianificazione delle sessioni didattiche.
ClassActivityService	it.astromark.classwork.service	Servizio per la gestione della logica delle attività in classe, supportando operazioni CRUD e l'organizzazione delle sessioni didattiche.
ClassCommunicationController	it.astromark.communication.controller	Controller per la gestione delle operazioni relative alla comunicazione tra classe e famiglia, incluse funzionalità CRUD e invio di notifiche o messaggi.
ClassCommunicationService	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.
ChatController<ID>	it.astromark.chat.controller	Controller per la gestione delle operazioni relative alle chat, inclusi invio, ricezione e gestione dei messaggi, con supporto per identificatori univoci.
ChatService<ID>	it.astromark.chat.service	Servizio per la gestione della logica delle chat, supportando l'invio, la ricezione e l'elaborazione dei messaggi, con identificatori univoci per le conversazioni.
HomeworkChatController	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.
HomeworkChatService	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.
TicketChatController	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.
TicketChatService	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.
ClassManagementController	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.

ClassManagementService	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.
AcademicController	it.astromark.classManagement.controller	Controller per la gestione delle operazioni relative alla parte accademica delle classi, incluse funzionalità CRUD e gestione dei programmi e delle valutazioni.
AcademicService	it.astromark.classManagement.service	Servizio per la gestione della logica relativa agli aspetti accademici delle classi, supportando operazioni CRUD e l'organizzazione di programmi e valutazioni.
OrientationController	it.astromark.orientation.controller	Controller per la gestione delle operazioni relative all'orientamento scolastico, incluse funzionalità CRUD e supporto per le attività di orientamento per studenti.
PythonService	it.astromark.orientation.service	Servizio per la gestione della logica relativa all'orientamento scolastico, con un focus specifico su corsi o attività legate alla programmazione in Python.

2.1.2 Elenco Classi e Interfacce di tipo Entity e JpaRepository

Le classi JpaRepository saranno inserite nel package repository dello 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 delle saranno inserite nel package "entity" corrispondente. Per la descrizione completa delle classi riferirsi al RAD.

File	Package
User	it.astromark.user.commons.entity
SchoolUser	it.astromark.user.commons.entity
Student	it.astromark.user.student.entity
Parent	it.astromark.user.parent.entity
Teacher	it.astromark.user.teacher.entity
Secretary	it.astromark.user.secretary.entity
School	it.astromark.school.entity
Timetable	it.astromark.agenda.commons.entity
ClassTimetable	it.astromark.agenta.schoolClass.entity
ReceptionTimetable	it.astromark.agenda.reception.entity
Timeslot	it.astromark.agenda.commons.entity
ReceptionTimeslot	it.astromark.agenda.reception.entity
ReceptionBooking	it.astromark.agenda.reception.entity
TeachingTimeslot	it.astromark.agenda.schoolClass.entity
SignHour	it.astromark.agenda.schoolClass.entity
ClassActivity	it.astromark.classwork.entity
Homework	it.astromark.classwork.entity
TeacherClass	it.astromark.classManagement.entity
Teaching	it.astromark.classManagement.entity
Subject	it.astromark.classManagement.entity
StudyPlan	it.astromark.classManagement.entity
SchoolClass	it.astromark.classManagement.entity
Communication	it.astromark.communication.entity
Chat	it.astromark.chat.entity
Ticket	it.astromark.chat.entity
HomeworkChat	it.astromark.chat.entity
Message	it.astromark.chat.entity
JustifiableEntity	it.astromark.attendance.entity

Delay	it.astromark.attendance.entity
Absence	it.astromark.attendance.entity
Note	it.astromark.behavior.entity
Mark	it.astromark.mark.entity
SemesterReport	it.astromark.mark.entity

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.

User Interface
- → **assets**
- → **components**
  - → **form**
  - → **ui**
- → **configs**
- → **context**
- → **hooks**
- → **pages**
- → **services**
  - → **authentication**
  - → **userManagement**
  - → **schoolManagement**
  - → **attendance**
  - → **rating**
  - → **classwork**
  - → **communication**
  - → **chat**
  - → **agenda**
  - → **classManagement**
  - → **orientation**
- → **types**

3. 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 declered query usate dai @Service e quindi spesso mappate direttamente in questo layer.
Per inviare le password di primo accesso verrà usato JavaMail 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 Oracle e da CloudFlare.
Le classi TypeScript utilizzate per l'implementazione del sottosistema di Interfaccia Utente sono omesse perché fortemente legate 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 CrudController<T, R, RS, ID>

Nome	Descrizione	Precondizione	Post condizione
+create (R request): ResponseEntity<RS>	Crea una nuova entità nel sistema.	**context** CrudController::create (R request): ResponseEntity<RS> **pre** self.getById(request.id) = null	**context** CrudController::create (R request): ResponseEntity<RS> **post** self.getById(request.id) <> null **post** result.status = HttpStatus.OK
+ getById(ID id): ResponseEntity<RS>	Recupera un'entità esistente tramite l'ID.	**context** CrudController::create (R request): ResponseEntity<RS> **pre** -	**context** CrudController::create (R request): ResponseEntity<RS> **post** getById(request.id).id = request.id **post** result.status = HttpStatus.OK
+update(R request): ResponseEntity<RS>	Aggiorna un'entità esistente nel sistema.	**context** CrudController::create (R request): ResponseEntity<RS> **pre** self.getById(request.id).id = request.id	**context** CrudController::create (R request): ResponseEntity<RS> **post** result.status = HttpStatus.OK
+ delete(ID id)	Elimina un'entità esistente dal sistema.	**context** CrudController::create (R request): ResponseEntity<RS> **pre** self.getById(request.id).id = request.id	**context** CrudController::create (R request): ResponseEntity<RS> **post** self.getById(request.id).id = null

3.1.2 CrudService<T, R, RS, ID>

Nome	Descrizione	Precondizione	Post condizione
+ create (R request): RS	Crea una nuova entità nel servizio.	**context** CrudService::create (R request): RS **pre** self.getById(request.id) = null	**context** CrudService::create (R request): RS **post** self.getById(request.id) <> null **post** result <> null
+ getById(ID id): RS	Recupera un'entità esistente tramite l'ID.	**context** CrudService::getById(ID id): RS **pre** -	**context** CrudService::getById(ID id): RS **post** getById(request.id).id = request.id **post** result <> null
+ update(R request): RS	Aggiorna un'entità esistente nel servizio.	**context** CrudService::update(R request): RS **pre** self.getById(request.id).id = request.id	**context** CrudService::update(R request): RS **post** result <> null
+ delete(ID id)	Elimina un'entità esistente dal servizio.	**context** CrudController::create (R request): ResponseEntity<RS> **pre** self.getById(request.id).id = request.id	**context** CrudController::create (R request): ResponseEntity<RS> **post** self.getById(request.id).id = null

3.1.3 UserController

Nome	Descrizione	Precondizione	Post condizione
+requestRemoval(UUID id, Role role): ResponseEntity< Boolean>	Gestisce la richiesta di rimozione di un utente, aggiornando lo stato.	**context** UserController:: requestRemoval(UUID id): ResponseEntity< Boolean> **pre** CrudService::getById(id) <> null **pre** CrudService::getById(id).pendingState <> “MarkRemoved”	**context** UserController:: requestRemoval(UUID id): ResponseEntity< Boolean> **post** result.status = HttpStatus.OK and result.body = true and CrudService::getById(id).pendingState = “MarkRemoved”

3.1.4 UserService

Nome	Descrizione	Precondizione	Post condizione
+requestRemoval(UUID id, Role role): Boolean	Gestisce la rimozione dell'utente, aggiornando il suo stato.	**context** UserService:: requestRemoval(UUID id): Boolean **pre** CrudService::getById(id) <> null **pre** CrudService::getById(id).pendingState <> “MarkRemoved”	**context** UserService:: requestRemoval(UUID id): Boolean **post** result.pendingState = “MarkRemoved”
+ getByRole(String code, String username, String password, Role role): SchoolUser	Recupera un utente scolastico in base al ruolo e alle credenziali.	**context** UserService:: getByRole(String code, String username, String password, Role role): SchoolUser **pre** -	**context** UserService:: getByRole(String code, String username, String password, Role role): SchoolUser **post** result = null or (result.code = code and result.username = username and result.password = password)
+	Restituisce lo studente attualmente loggato.	**context** UserService:: getLoggedStudent() :	**context** UserService:: getLoggedStudent() :

getLoggedStudent() : Student		Student **pre** -	Student **post** -
+ getLoggedParent() : Parent	Restituisce il genitore attualmente loggato.	**context** UserService:: getLoggedParent() : Parent **pre** -	**context** UserService:: getLoggedParent() : Parent **post** -
+getLoggedSecretary () : Secretary	Restituisce il segretario attualmente loggato.	**context** UserService:: getLoggedSecretary() : Secretary **pre** -	**context** UserService:: getLoggedSecretary() : Secretary **post** -
+ getLoggedTeacher() : Teacher	Restituisce il docente attualmente loggato.	**context** UserService:: getLoggedTeacher() : Teacher **pre** -	**context** UserService:: getLoggedTeacher() : Teacher **post** -

3.1.5 AuthController

Nome	Descrizione	Precondizione	Post condizione
+login(LoginRequest request): JwtToken	Autentica l'utente e genera un token JWT se le credenziali sono valide.	**context** AuthController:: login(LoginRequest request): JwtToken **pre** (request.role = “Parent” or request.role = “Student” or request.role = “Teacher” or request.role = “Secretary”) and request.username.size() > 4 and request.password.size() >= 8 and	**context** AuthController:: login(LoginRequest request): JwtToken **post** UserService::getByRole(request.code, request.username, request.password, request.role) <> null

		request.code.size() = 7 and self.logout() = false	
+ logout() : Boolean	Gestisce la disconnessione dell'utente.	**context** AuthController:: logout(): Boolean **pre** -	**context** AuthController:: logout(): Boolean **post** -

3.1.6 StudentController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+getByIdDetailedIfLogged(): ResponseEntity<StudentDetailsRespose>	Restituisce i dettagli completi dello studente loggato.	**context** StudentController:: getByIdDetailedIfLogged() : ResponseEntity<StudentDetailsRespose> **pre** -	**context** StudentController:: getByIdDetailedIfLogged() : ResponseEntity<StudentDetailsRespose> **post** let user = UserService::getLoggedStudent **post** result.body.id = user.id **post** result.body..username = user.body.username **post** result.body.name = user.name **post** result.body.surname = user.surname **post** result.body.email = user.email **post** result.body.birthDate = user. birthDate **post** result.body.taxID = user.taxID
+ getAverege():	Calcola e restituisce la	**context**	**context**

ResponseEntity<Double>	media delle valutazioni dello studente.	StudentController::getAverege(): ResponseEntity<Double> **pre** let marks : Set = MarkService::getStudentMark(UserService::getLoggedStudent.id) mark <> null	StudentController::getAverege(): ResponseEntity<Double> **post** result.body >= 0
+hasBeenPromoted() : ResponseEntity<Boolean>	Verifica se lo studente è stato promosso in base alla media del semestre.	**context** StudentController::hasBeenPromoted(): ResponseEntity<Boolean> **pre** MarkService::getAllReport()<> null	**context** StudentController::hasBeenPromoted(): ResponseEntity<Boolean> **post** result.body = MarkService::getSemesterAverage(MarkService::getAllReport().asOrderedSet().get(0)) > 6
+ getAttitute(): ResponseEntity<String>	Restituisce il comportamento dello studente (ad esempio, positivo o negativo).	**context** StudentController::getAttitute(): ResponseEntity<String> **pre** -	**context** StudentController::getAttitute(): ResponseEntity<Boolean> **post** -

3.1.7 StudentService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+getByIdDetailed():StudentDetailsRespose	Restituisce i dettagli completi dello studente loggato.	**context** StudentService::getByIdDetailed():StudentDetailsRespose **pre** -	**context** StudentService::getByIdDetailed():StudentDetailsRespose **post** let user = UserService::getLoggedStudent **post** result.id = user.id **post** result..username =

			user.username **post** result.name = user.name **post** result.surname = user.surname **post** result.email = user.email **post** result.birthDate = user.birthDate **post** result.taxID = user.taxID
+ getAverege():Double	Calcola la media delle valutazioni dello studente.	**context** StudentService::getAverege():Double **pre** let marks : Set = MarkService::getStudentMark(UserService::getLoggedStudent.id) mark <> null	**context** StudentService::getAverege():Double **post** result >= 0
+hasBeenPromoted():Boolean	Verifica se lo studente è stato promosso in base alla media del semestre.	**context** StudentService::hasBeenPromoted():Boolean **pre** MarkService::getAllReport()<> null	**context** StudentService::hasBeenPromoted():Boolean **post** result = MarkService::getSemesterAverage(MarkService::getAllReport().asOrderedSet().get(0)) > 6
+ getAttitute():Boolean	Restituisce l'atteggiamento dello studente (ad esempio, positivo o negativo).	**context** StudentService::getAttitute():Boolean **pre** -	**context** StudentService::getAttitute():Boolean **post** -
+ create(StudentRequest request): StudentResponse	Crea un nuovo studente basato sulla richiesta e restituisce i dettagli.	**context** StudentService::create(StudentRequest request): StudentResponse **pre** request.name.size() > 0	**context** StudentService::create(StudentRequest request): StudentResponse **post** result.username =

		pre request.surname.size() > 0 **pre** request.taxId.size() = 16 **pre** DateYear::now - request.birthDay.year >= 12	@pre.request.concat(self.name, "." self.surname, SchoolService::getNumberStudents())

3.1.8 ParentController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+ addStudent(UUID studentId, UUID parentId):ResponseEntity< ParentResponse>	Aggiunge uno studente a un genitore, associando i rispettivi ID.	**context** ParentController:: addStudent(UUID studentId, UUIDparentId):ResponseEntity< ParentResponse> **pre** -	**context** ParentController:: addStudent(UUID studentId, UUIDparentId):ResponseEntity< ParentResponse> **post** StudentService::getById(@pre.studentId) <> null **post** result.body = ParentService::getById(@pre.parentId) <> null

3.1.9 ParentService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+addStudent(UUID studentId, UUID parentId): ParentResponse	Associa uno studente a un genitore, verificando l'esistenza di entrambi nel sistema.	**context** ParentService:: addStudent(UUID studentId, UUIDparentId): ParentResponse **pre** -	**context** ParentService:: addStudent(UUID studentId, UUIDparentId): ParentResponse **post** StudentService::getById(@pre.studentId) <> null **post** result = ParentService::getById(@pre.parentId) <> null

3.1.10SecretaryController extends CrudController

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

Nome	Descrizione	Precondizione	Post condizione
-	-	-	-

3.1.11SecretaryService 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
-	-	-	-

3.1.12TeacherController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+ getSchoolClasses(): ResponseEntity<List<SchoolClass>	Restituisce la lista delle classi scolastiche associate al docente loggato.	**context** TeacherController:: getSchoolClasses(): ResponseEntity<List<SchoolClass> **pre** -	**context** TeacherController:: getSchoolClasses(): ResponseEntity<List<SchoolClass> **post** result.body = TeacherService::getLoggedTeacher().schoolClasses

3.1.13TeacherService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+getSchoolClasses():List<SchoolClass>	Restituisce la lista delle classi scolastiche associate al docente loggato.	**context** TeacherService:: getSchoolClasses():List<SchoolClass> **pre** -	**context** TeacherService:: getSchoolClasses():List<SchoolClass> **post** result = TeacherService::getLoggedTeacher().schoolClasses

3.1.14SchoolController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+addClass(SchoolClassRequest request) : ResponseEntity<SchoolClassResponse>	Aggiunge una nuova classe scolastica associata alla scuola dell'utente segretario loggato.	**context** SchoolController::addClass(SchoolClassRequest request) : ResponseEntity<SchoolClassResponse> **pre** request.code = UserService::getLoggedSecretary.code **pre** SchoolService::getById(request.id) <> null	**context** SchoolController::addClass(SchoolClassRequest request) : ResponseEntity<SchoolClassResponse> **post** result = SchoolService::getById(request.id) <> null
+ addStudent(UUID studentId) : ResponseEntity <StudentResponse>	Aggiunge uno studente alla scuola associata all'utente segretario loggato.	**context** SchoolController::addStudent(UUID studentId) : ResponseEntity <StudentResponse> **pre** StudentService::getById(studentId) <> null **pre** StudentService::getById(studentId).code = UserService::getLoggedSecretary.code	**context** SchoolController::addStudent(UUID studentId) : ResponseEntity <StudentResponse> **post** result = StudentService::getById(studentId) <> null
+ addSecretary(UUID secretaryId) : ResponseEntity <SecretaryResponse>	Aggiunge un segretario alla scuola.	**context** SchoolController::addSecretary(UUID secretaryId) : ResponseEntity <SecretaryResponse> **pre** SecretaryService::getById(secretaryId) <> null **pre** SecretaryService::getById(secretaryId).code = UserService::getLoggedSecretary.code	**context** SchoolController::addSecretary(UUID secretaryId) : ResponseEntity <SecretaryResponse> **post** result.body = Stud SecretaryService::getById(studentId) <> null

+ addTeacher(UUID teacherId) : ResponseEntity <TeacherResponse>	Aggiunge un insegnante alla scuola.	**context** SchoolController::addTeacher(UUID teacherId) : ResponseEntity <TeacherResponse> **pre** TeacherService::getById(studentId) <> null **pre** TeacherService::getById(studentId).code = UserService::getLoggedTeacher.code	**context** SchoolController::addTeacher(UUID teacherId) : ResponseEntity <TeacherResponse> **post** result.body = TeacherService::getById(teacherId) <> null
+ addParent(UUID parentID) : ResponseEntity<ParentResponse>	Aggiunge un genitore alla scuola.	**context** SchoolController::addParent(UUID parentID) : ResponseEntity<ParentResponse> **pre** ParentService::getById(parentId) <> null **pre** ParentService::getById(parentId).code = UserService::getLoggedParent.code	**context** SchoolController::addParent(UUID parentID) : ResponseEntity<ParentResponse> **post** result.body = ParentService::getById(parentId) <> null

3.1.15SchoolService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+addClass(SchoolClassRequest request) : SchoolClassResponse	Aggiunge una nuova classe scolastica alla scuola.	**context** SchoolService::addClass(SchoolClassRequest request) : SchoolClassResponse **pre** request.code = UserService::getLoggedSecretary.code **pre** SchoolService::getById(request.id) <> null	**context** SchoolService::addClass(SchoolClassRequest request) : SchoolClassResponse **post**
+ addStudent(UUID studentId request) : StudentResponse	Aggiunge uno studente alla scuola.	**context** SchoolController::addStudent(UUID studentId request) : StudentResponse **pre** StudentService::getById(studentId) <> null **pre** StudentService::getById(studentId).code = UserService::getLoggedSecretary.code	**context** SchoolService::addStudent(UUID studentId request) : StudentResponse **post**
+ addSecretary(UUID secretaryId request) : SecretaryResponse	Aggiunge un segretario alla scuola.	**context** SchoolService::addSecretary(UUID secretaryId request) : SecretaryResponse> **pre** SecretaryService::getById(secretaryId) <> null **pre** SecretaryService::getById(secretaryId).code = UserService::getLoggedSecretary.code	**context** SchoolService::addSecretary(UUID secretaryId reques,) : SecretaryResponse **post**
+ addTeacher(UUID	Aggiunge un	**context**	**context**

teacherId request) : TeacherResponse	insegnante alla scuola.	SchoolService::addTeacher(UUID teacherId request) : TeacherResponse **pre** TeacherService::getById(studentId) <> null **pre** TeacherService::getById(studentId).code = UserService::getLoggedTeacher.code	SchoolService::addTeacher(UUID teacherId request) : TeacherResponse **post** result = TeacherService::getById(teacherId) <> null
+ addParent(UUID parentID) : ParentResponse	Aggiunge un genitore alla scuola.	**context** SchoolService::addParent(UUID parentID) : ParentResponse **pre** ParentService::getById(parentId) <> null **pre** ParentService::getById(parentId).code = UserService::getLoggedParent.code	**context** SchoolService::addParent(UUID parentID) : ParentResponse **post** result = ParentService::getById(parentId) <> null
+getNumberStudents(String code) : Integer	Restituisce il numero di studenti iscritti alla scuola identificata dal codice.	**context** SchoolService::getNumberStudents(String code) : Integer **pre** request.code.size() = 7	**context** SchoolService::getNumberStudents(String code) : Integer **post** result >= 0

3.1.16ReceptionAgendaController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+ book(Long receptionTimeslotID, Date date) : ResponseEntity<Boolean>	Prenota un orario di ricevimento per un genitore.	**context** ReceptionAgendaController::book(UUID parentID, Long receptionTimeslotID, Date date) : ResponseEntity<Boolean> **pre** UserService::getLoggedParent().students.schoolClass.teachers.include(self.getById(receptionTimeslotId).teacher)	**context** ReceptionAgendaController::book(UUID parentID, Long receptionTimeslotID, Date date) : ResponseEntity<Boolean> **post** result.body = self.getById(receptionTimeslotId).capacity > self.getById(receptionTimeslotId).bookedSlots::filter(r | r.date = date).count()
+ addTimeslot (ReceptionTimeslotRequest request): ResponseEntity<ReceptionTimeslotRespose>	Aggiunge un nuovo orario di ricevimento.	**context** ReceptionAgendaController::addTimeslot (ReceptionTimeslotRequest request): ResponseEntity<ReceptionTimeslotRespose> **pre** UserService::getLoggedParent<>null	**context** ReceptionAgendaController::addTimeslot (ReceptionTimeslotRequest request): ResponseEntity<ReceptionTimeslotRespose> **post** -
+refuse(Long bookedId) : ResponseEntity<Boolean>	Rifiuta una prenotazione di orario di ricevimento.	**context** ReceptionAgendaController::refuse(Long bookedId) : ResponseEntity<Boolean> **pre** exist(s | self.getBookedSlots.id = s and s.receptionSlot.ReceptionAgenda.teacher = UserService::getLoggedTeacher())	**context** ReceptionAgendaController::refuse(Long bookedId) : ResponseEntity<Boolean> **post** result.body = true

		pre not exist(s | self.getBookedSlots.id = s and s.receptionSlot.ReceptionAgenda.teacher = UserService::getLoggedTeacher)() and s.confirmed)	
+getNotConfirmed(Long tableId) : ResponseEntity<List< ReceptionTimeslotRespose>>	Ottiene gli orari di ricevimento non confermati per una data e un tavolo specifico.	**context** ReceptionAgendaControlller::getNotConfirmed(Long tableId, Date date) : ResponseEntity<List< ReceptionTimeslotRespose>> **pre** self.getById(tableId) <> null **pre** self.getById(tableId).teacher = UserService::getLoggedTeacher() or UserService::getLoggedParent().students.schoolClass.teachers.exist(u | u.receptionTable.tableId = tableId)	**context** ReceptionAgendaControlller::getNotConfirmed(Long tableId, Date date) : ResponseEntity<List< ReceptionTimeslotRespose>> **post** -
+ confirm(Long id) : ResponseEntity<Boolean>	Conferma una prenotazione di orario di ricevimento.	**context** ReceptionAgendaControlller::confirm(Long id) : ResponseEntity<Boolean> **pre** exist(s | self.getBookedSlots.id = s and s.receptionSlot.ReceptionAgenda.teacher = UserService::getLoggedTeacher)()) **pre**	**context** ReceptionAgendaControlller::confirm(Long id) : ResponseEntity<Boolean> **post** result.body = true

		not exist(s | self.getBookedSlots.id = s and s.receptionSlot.ReceptionAgenda.teacher = UserService::getLoggedTeacher)() and s.refused)	
+ getRefused(Long tableId) : ResponseEntity<List<BookedSlots>	Ottiene l'esito del rifiuto di una prenotazione di orario di ricevimento per un tavolo specifico.	**context** ReceptionAgendaControlller::getRefused(Long tableId) : ResponseEntity<List<BookedSlots> **pre** self.getById(tableId) <> null **pre** self.getById(tableId).teacher = UserService::getLoggedTeacher()	**context** ReceptionAgendaControlller::getRefused(Long tableId) : ResponseEntity<List<BookedSlots> **post** -
+getBookedSlots(Long tableId) : ResponseEntity<List< ReceptionTimeslotRespose>>	Ottiene la lista degli orari di ricevimento prenotati per un tavolo specifico.	**context** ReceptionAgendaControlller::getBookedSlots(Long tableId) : ResponseEntity<List< ReceptionTimeslotRespose>> **pre** self.getById(tableId) <> null **pre** self.getById(tableId).teacher = UserService::getLoggedTeacher() or UserService::getLoggedParent().students.schoolClass.teachers.exist(u | u.receptionTable.tableId = tableId)	**context** ReceptionAgendaControlller::getBookedSlots(Long tableId) : ResponseEntity<List< ReceptionTimeslotRespose>> **post** -

3.1.17ReceptionAgendaService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+ book(UUID parentID, Long receptionTimeslotID) : Boolean	Prenota un orario di ricevimento per un genitore.	**context** ReceptionAgendaService::book(UUID parentID, Long receptionTimeslotID, Date date) : ResponseEntity<Boolean> **pre** UserService::getLoggedParent().students.schoolClass.teachers.include(self.getById(receptionTimeslotId).teacher)	**context** ReceptionAgendaService::book(UUID parentID, Long receptionTimeslotID) : Boolean **post** result = self.getById(receptionTimeslotId).capacity > self.getById(receptionTimeslotId).bookedSlots::filter(r | r.date = date).count()
+ addTimeslot (Long id, ReceptionTimeslotRequest request): ResponseEntity<ReceptionTimeslotResponse>	Aggiunge un nuovo orario di ricevimento.	**context** ReceptionAgendaService::addTimeslot (ReceptionTimeslotRequest request): ReceptionTimeslotResponse **pre** UserService::getLoggedParent<>null	**context** ReceptionAgendaService::addTimeslot (Long id, ReceptionTimeslotRequest request): ResponseEntity<ReceptionTimeslotRespose> **post** -
+ refuse(Long id) : Boolean	Rifiuta una prenotazione di orario di ricevimento.	**context** ReceptionAgendaService::refuse(Long bookedId) : Boolean **pre** exist(s | self.getBookedSlots.id = s and s.receptionSlot.ReceptionAgenda.teacher = UserService::getLoggedTeacher()()) **pre** not exist(s | self.getBookedSlots.id = s	**context** ReceptionAgendaService::refuse(Long id) : Boolean **post** result.body = true

		and s.receptionSlot.ReceptionAgenda.teacher = UserService::getLoggedTeacher)() and s.confirmed)	
+getNotConfirmed(Long tableId, Date date) : List< ReceptionTimeslotRespose>	Ottiene gli orari di ricevimento non confermati per una data e un tavolo specifico.	**context** ReceptionAgendaService::getNotConfirmed(Long tableId, Date date) : List< ReceptionTimeslotRespose> **pre** self.getById(tableId) <> null **pre** self.getById(tableId).teacher = UserService::getLoggedTeacher() or UserService::getLoggedParent().students.schoolClass.teachers.exist(u | u.receptionTable.tableId = tableId)	**context** ReceptionAgendaService::getNotConfirmed(Long tableId, Date date) : List< ReceptionTimeslotRespose> **post** -
+ confirm(Long id) : Boolean	Conferma una prenotazione di orario di ricevimento.	**context** ReceptionAgendaController::confirm(Long id) : ResponseEntity<Boolean> **pre** exist(s | self.getBookedSlots.id = s and s.receptionSlot.ReceptionAgenda.teacher = UserService::getLoggedTeacher)()) **pre** not exist(s | self.getBookedSlots.id = s and s.receptionSlot.Reception	**context** ReceptionAgendaService::confirm(Long id) : Boolean **post** result.body = true

		Agenda.teacher = UserService::getLoggedTeacher)() and s.refused)	
+ getRefused(Long tableId) : Boolean	Ottiene l'esito del rifiuto di una prenotazione di orario di ricevimento per un tavolo specifico.	**context** ReceptionAgendaController::getRefused(Long tableId) : ResponseEntity<List<BookedSlots> **pre** self.getById(tableId) <> null **pre** self.getById(tableId).teacher = UserService::getLoggedTeacher()	**context** ReceptionAgendaService::getRefused(Long tableId) : Boolean **post** -
+getBookedSlots(Long tableId) : List< ReceptionTimeslotRespose>	Ottiene la lista degli orari di ricevimento prenotati per un tavolo specifico.	**context** ReceptionAgendaController::getBookedSlots(Long tableId) : ResponseEntity<List< ReceptionTimeslotRespose>> **pre** self.getById(tableId) <> null **pre** self.getById(tableId).teacher = UserService::getLoggedTeacher() or UserService::getLoggedParent().students.schoolClass.teachers.exist(u | u.receptionTable.tableId = tableId)	**context** ReceptionAgendaService::getBookedSlots(Long tableId) : List< ReceptionTimeslotRespose> **post** -

3.1.18ClassAgendaController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+ getTotalHour(Long id) : ResponseEntity<Integer>	Ottiene il totale delle ore di lezione per una classe o un insegnante identificato da un ID specifico.	**context** ClassAgendaController::getTotalHour(Long id) : ResponseEntity<Integer> **pre** ClassAgendaService::getById(id) <> null	**context** ClassAgendaController::getTotalHour(Long id) : ResponseEntity<Integer> **post** result.body = ClassAgendaService::getById(id).teachingTimeslots.count()
+ addTimeslot(Long id, TeachingTimeslotRequest request): ResponseEntity<TeachingTimeslotResponse>	Aggiunge un orario di lezione (timeslot) per una classe o insegnante identificato dall'ID.	**context** ClassAgendaController::addTimeslot (Long id, TeachingTimeslotRequest request): ResponseEntity<TeachingTimeslotResponse> **pre** ClassAgendaService::getById(id).schoolClass.school.code = UserService::getLoggedSecretary().code **pre** request.day <> Day.Sunday **pre** request.hour > 0 and request.hour < 9	**context** ClassAgendaController::addTimeslot (Long id, TeachingTimeslotRequest request): ResponseEntity<TeachingTimeslotResponse> **post** report.body.day = @pre.request.day **post** report.body.hour = @pre.request.hour

3.1.19ClassAgendaService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+ getTotalHour(Long id) : Integer	Restituisce il totale delle ore di lezione per una classe o un insegnante identificato da un ID specifico.	**context** ClassAgendaService::getTotalHour(Long id) : Integer **pre** self.getById(id) <> null	**context** ClassAgendaService::getTotalHour(Long id) : Integer **post** result = self.getById(id).teachingTimeslots.count()
+ addTimeslot (Long id, TeachingTimeslotRequest request): TeachingTimeslotResponse	Aggiunge un orario di lezione (timeslot) per una classe o insegnante identificato dall'ID.	**context** ClassAgendaService::addTimeslot (Long id, TeachingTimeslotRequest request): TeachingTimeslotResponse **pre** self.getById(id).schoolClass.school.code = UserService::getLoggedSecretary().code **pre** request.day <> Day.Sunday **pre** request.hour > 0 and request.hour < 9	**context** ClassAgendaService::addTimeslot (Long id, TeachingTimeslotRequest request): TeachingTimeslotResponse **post** report.day = @pre.request.day **post** report.hour = @pre.request.hour

3.1.20TeachingTimeslotController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+sign(Long id, SignHourRequest request): ResponseEntity<SignHourResponse>	Registra la firma di un orario di lezione per una determinata classe o insegnante identificato da un ID.	**context** TeachingTimeslotController:: sign(Long id, SignHourRequest request): ResponseEntity<SignHourResponse> **pre** UserService::getLoggedTeacher() .teacherClasses.schoolClass = TeachingTimeslotService::getById(id).classTimetable.schoolClass **pre** TeachingTimeslotService::getById(request.id).teacher = UserService::getLoggedTeacher() **pre** not TeachingTimeslotService::getById(request.id).signedTimeslots.exist(t | t.id = id and t.day = request.day and t.date = DateDay::now())	**context** TeachingTimeslotController:: sign(Long id, SignHourRequest request): ResponseEntity<SignHourResponse> **post** result.body.date = @pre.request.dataTime = Date::now()
+ isSigned(Long id, Date date): ResponseEntity<Boolean>	Verifica se un orario di lezione è stato firmato per una determinata classe o insegnante in una data specifica.	**context** TeachingTimeslotController:: isSigned(Long id, Date date): ResponseEntity<Boolean> **pre** -	**context** TeachingTimeslotController:: isSigned(Long id, Date date): ResponseEntity<Boolean> **post** result.body = TeachingTimeslotService:: getById(id).signedHours.exist(s | s.date = date)

3.1.21TeachingTimeslotService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+sign(Long id, SignHourRequest request): SignHourResponse	Gestisce la registrazione della firma di un orario di lezione per una determinata classe o insegnante.	**context** TeachingTimeslotService::sign(Long id, SignHourRequest request): SignHourResponse **pre** UserService::getLoggedTeacher().teacherClasses.schoolClass = TeachingTimeslotService::getById(id).classTimetable.schoolClass **pre** TeachingTimeslotService::getById(request.id).teacher = UserService::getLoggedTeacher() **pre** not TeachingTimeslotService::getById(request.id).signedTimeslots.exist(t | t.id = id and t.day = request.day and t.date = DateDay::now())	**context** TeachingTimeslotService::sign(Long id, SignHourRequest request): SignHourResponse **post** result.date = @pre.request.dataTime = Date::now()
+ isSigned(Long id, Date date): Boolean	Verifica se un orario di lezione è stato firmato per una determinata classe o insegnante in una data specifica.	**context** TeachingTimeslotService::isSigned(Long id, Date date): Boolean **pre**	**context** TeachingTimeslotService::isSigned(Long id, Date date): Boolean **post** result = TeachingTimeslotService::getById(id).signedHours.exist(s | s.date = date)

3.1.22 MarkController extends CrudController<Mark, MarkRequest, MarkReponse, MarkID>

Nome	Descrizione	Precondizione	Post condizione
+createReport(SemesterReportRequest request): ResponseEntity<SemesterReportResponse>	Crea un nuovo report semestrale.	**context** MarkController:: createReport(SemesterReportRequest request): ResponseEntity<SemesterReportResponse> **pre** not MarkService::getAllReport().exist(r | r.id = request.id and year = request.year and request.semester = r.semester) **pre** request.semester = 1 or request.semester = 2 **pre** UserService::getLoggedTeacher().teachersClass.exist(t | t.schoolClass.exist(c | exist(s | s.id = request.id) and c.teacher.include(UserService::getLoggedTeacher())))	**context** MarkController:: createReport(SemesterReportRequest request): ResponseEntity<SemesterReportResponse> **post** result.body.year = request.year **post** result.body.semester = request.semester **post** result.body.passed = MarkService::getSemesterAverage(self.id) **post** result.body.viewed = false **post** result.body.student = request.id **post** result.body.public =false
+ viewReport(Long id): ResponseEntity<SemesterReportResponse>	Visualizza un report semestrale esistente	**context** MarkController:: viewReport(Long id): ResponseEntity<SemesterReportResponse> **pre** UserService::getLoggedParent().students.exist(s | s.semesterReports.exist(r | r.id = id)	**context** MarkController:: viewReport(Long id): ResponseEntity<SemesterReportResponse> **post** report.body = MarkService::getById(@pre.id) **post** MarkService::getById(@pre.id).viewed = true
+publishReport(Long id): ResponseEntity<SemesterReportResponse>	Pubblica un report semestrale.	**context** MarkController:: publishReport(Long id): ResponseEntity<SemesterReportResponse> **pre** UserService::getLoggedTeache	**context** MarkController:: publishReport(Long id): ResponseEntity<SemesterReportResponse> **post** result.body =

		r() <> null	MarkService::getReportById(@pre.id) **post** MarkService::getReportById(@pre.id).public = true
+addReportMark(Long id, Subject subject, Integer mark): ResponseEntity<SemesterReportResponse>	Aggiunge un voto al report semestrale per una specifica materia.	**context** MarkController::addReportMark(Long id, Subject subject, Integer mark): ResponseEntity<SemesterReportResponse> **pre** UserService::getLoggedTeacher() <> null **pre** mark >= 0 and mark <= 10 **pre** not MarkService::getReportById(@pre.id).marks.exist(m | m.subject = subject)	**context** MarkController::addReportMark(Long id, Subject subject, Integer mark): ResponseEntity<SemesterReportResponse> **post** result.body = MarkService::getReportById(@pre.id) **post** result.body.passed = MarkService::getSemesterAverage(self.id)
+ deleteReport(Long id)	Elimina un report semestrale esistente.	**context** MarkController::deleteReport(Long id) **pre** -	**context** MarkController::deleteReport(Long id) **post** -
+getStudentMarkByDate(UUID studentId, Date date): ResponseEntity<List<MarkResponse>>	Ottiene i voti di uno studente per una specifica data.	**context** MarkController::getStudentMarkByDate(UUID studentId, Date date): ResponseEntity<List<MarkResponse>> **pre** StudentService::getById(studentId)<> null **pre** date < Date::now + 1	**context** MarkController::getStudentMarkByDate(UUID studentId, Date date): ResponseEntity<List<MarkResponse>> **post** -
+getStudentMarks(UUID studentId): ResponseEntity<List<MarkResponse>	Ottiene tutti i voti di uno studente.	**context** MarkController::getStudentMarks(UUID studentId): ResponseEntity<List<MarkResponse> **pre** StudentService::getById(studentId)<> null	**context** MarkController::getStudentMarkByDate(UUID studentId): ResponseEntity<List<MarkResponse> **post** -

3.1.23 MarkService extends CrudService<Mark, MarkRequest, MarkReponse, MarkID>

Nome	Descrizione	Precondizione	Post condizione
+createReport(SemesterReportRequest request): SemesterReportResponse	Crea un nuovo report semestrale.	**context** MarkService:: createReport(SemesterReportRequest request): SemesterReportResponse **pre** not MarkService::getAllReport().exist(r | r.id = request.id and year = request.year and request.semester = r.semester) **pre** request.semester = 1 or request.semester = 2 **pre** UserService::getLoggedTeacher().teachersClass.exist(t | t.schoolClass.exist(c | exist(s | s.id = request.id) and c.teacher.include(UserService::getLoggedTeacher())))	**context** MarkService:: createReport(SemesterReportRequest request): SemesterReportResponse **post** result.year = request.year **post** result.semester = request.semester **post** result.passed = MarkService::getSemesterAverage(self.id) **post** result.viewed = false **post** result.student = request.id **post** result.public =false
+ viewReport(Long id): SemesterReportResponse	Visualizza un report semestrale esistente.	**context** MarkService:: viewReport(Long id): SemesterReportResponse **pre** UserService::getLoggedParent().students.exist(s | s.semesterReports.exist(r | r.id = id)	**context** self. viewReport(Long id): SemesterReportResponse **post** report = self.getById(@pre.id) **post** self.getById(@pre.id).viewed = true
+publishReport(Long id): SemesterReportResp	Pubblica un report semestrale.	**context** MarkService:: publishReport(Long id): SemesterReportResponse	**context** MarkService:: publishReport(Long id): SemesterReportResponse

onse		**pre** UserService::getLoggedTeacher() <> null	**post** result = self.getReportById(@pre.id) **post** self.getReportById(@pre.id).public = true
+getReportById(Long id): List<SemesterReportResponse>	Ottiene tutti i report semestrali.	**context** MarkService::getAllReport(): List<SemesterReportResponse> **pre** UserService::getLoggedTeacher() <> null or UserService::getLoggedStudent.semesterReports.exist(r | r.id = id) or UserService::getLoggedParent().students.exist(s | s.semesterReports.exist(r | r.id = id)	**context** MarkService::getAllReport(): List<SemesterReportResponse> **post** **-**
+getAllReport(): List<SemesterReportResponse>	Ottiene tutti i report semestrali.	**context** MarkService::getAllReport(): List<SemesterReportResponse> **pre** **-**	**context** MarkService::getAllReport(): List<SemesterReportResponse> **post** **-**
+getSemesterAverage(SemesterReport report): Double	Calcola la media dei voti per un report semestrale.	**context** MarkService::getSemesterAverage(SemesterReport report): Double **pre** **-**	**context** MarkService::getSemesterAverage(SemesterReport report): Double **post** result = report.marks.sum() / report.marks.count()
+ addReportMark(Long id, Subject subject, Integer mark): SemesterReportResponse	Aggiunge un voto a un report semestrale per una materia specifica.	**context** MarkService::addReportMark(Long id, Subject subject, Integer mark): SemesterReportResponse **pre**	**context** MarkService::addReportMark(Long id, Subject subject, Integer mark): SemesterReportResponse **post**

		UserService::getLoggedTeacher() <> null **pre** mark >= 0 and mark <= 10 **pre** not self.getReportById(@pre.id).marks.exist(m | m.subject = subject)	result = self.getReportById(@pre.id) **post** result.passed = self.getSemesterAverage(self.id)
+deleteReport(Long id)	Elimina un report semestrale esistente.	**context** MarkService::deleteReport(Long id) **pre**	**context** MarkService::deleteReport(Long id) **post**
+getStudentMarkByDate(UUID studentId, Date date): List<MarkResponse>	Ottiene i voti di uno studente per una specifica data.	**context** MarkService::getStudentMarkByDate(UUID studentId, Date date): List<MarkResponse> **pre** self.getById(studentId)<> null **pre** date < Date::now + 1	**context** MarkService::getStudentMarkByDate(UUID studentId, Date date): List<MarkResponse> **post** -
+getStudentMarks(UUID studentId): List<MarkResponse>	Ottiene tutti i voti di uno studente.	**context** MarkController::getStudentMarks(UUID studentId): List<MarkResponse> **pre** self.getById(studentId)<> null	**context** MarkController::getStudentMarkByDate(UUID studentId): List<MarkResponse> **post** -

3.1.24NoteController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+ view(UUID id): ResponseEntity<NoteResponse>	Conferma la visione di una nota specifica in base all'ID.	**context** NoteController::view(UUID id): ResponseEntity<NoteResponse> **pre** -	**context** NoteController::view(UUID id): ResponseEntity<NoteResponse> **post** self.getById(@pre.id).viewed = true **post** result.body = self.getById(@pre.id)
+ getAll(): ResponseEntity<List<NoteResponse>>	Recupera tutte le note disponibili.	**context** NoteController::getAll(): ResponseEntity<List<NoteResponse>> **pre** -	**context** NoteController::getAll(): ResponseEntity<List<NoteResponse>> **post** -

3.1.25NoteService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+ view(UUID id): NoteResponse	Conferma la visione di una nota specifica in base all'ID.	**context** NoteService::view(UUID id): NoteResponse **pre** -	**context** NoteService::view(UUID id): NoteResponse **post** self.getById(@pre.id).viewed = true **post** result.body = self.getById(@pre.id)
+getAll():List<NoteResponse>	Recupera tutte le note disponibili.	**context** NoteService::getAll():List<NoteResponse> **pre** -	**context** NoteService::getAll():List<NoteResponse> **post** -

3.1.26JustifiableController<T extends JustifiableEntity, R, RS>

Nome	Descrizione	Precondizione	Post condizione
+ justify(UUID idStudent, R justification):ResponseEntity<RS>	Giustifica l'assenza di uno studente tramite una richiesta di giustificazione.	**context** JustifiableController::justify(UUID idStudent, R justification):ResponseEntity<RS> **pre** UserService::getLoggedStudent().id = idStudent or UserService::getLoggedParent().students.exist(s | s.id = idStudent)	**context** JustifiableController::justify(UUID idStudent, R justification):ResponseEntity<RS> **post** result = null or result.body.justified = true
+getByDateRange (UUID idStudent, JustifiableDateRange range) :ResponseEntity<List<RS>>	Recupera le giustificazioni di uno studente in un intervallo di date.	**context** JustifiableController:getByDateRange (UUID idStudent, JustifiableDateRange range) :ResponseEntity<List<RS>> **pre** range.date <= Date::now() **pre** UserService::getLoggedStudent().id = idStudent or UserService::getLoggedParent().students.exist(s | s.id = idStudent) or UserService::getLoggedTeacher().schoolClasses.exist(c | c = StudentService::getById(idStudent).schoolClass)	**context** JustifiableController:getByDateRange (UUID idStudent, JustifiableDateRange range) :ResponseEntity<List<RS>> **post** -
+delete(R id)	Elimina una giustificazione specifica tramite l'ID.	**context** JustifiableController::delete(R id) **pre** -	**context** JustifiableController::delete(R id) **post** -

3.1.27JustifiableService<T extends JustifiableEntity, R, RS>

Nome	Descrizione	Precondizione	Post condizione
+ justify(UUID idStudent, R justification): RS	Giustifica l'assenza di uno studente tramite una richiesta di giustificazione.	**context** JustifiableService::justify(UUID idStudent, R justification): RS **pre** range.date <= Date::now() **pre** UserService::getLoggedStudent().id = idStudent or UserService::getLoggedParent().students.exist(s | s.id = idStudent)	**context** JustifiableService::justify(UUID idStudent, R justification): RS **post** result = null or result.body.justified = true
+getByDateRange (UUID idStudent, JustifiableDateRange range) : List<RS>	Recupera le giustificazioni di uno studente in un intervallo di date.	**context** JustifiableService:getByDateRange (UUID idStudent, JustifiableDateRange range) : List<RS> **pre** range.date <= Date::now() **pre** UserService::getLoggedStudent().id = idStudent or UserService::getLoggedParent().students.exist(s | s.id = idStudent) or UserService::getLoggedTeacher().schoolClasses.exist(c | c = StudentService::getById(idStudent).schoolClass)	**context** JustifiableService:getByDateRange (UUID idStudent, JustifiableDateRange range) : List<RS> **post**
+delete(R id)	Elimina una giustificazione specifica tramite l'ID.	**context** JustifiableService::delete(R id) **pre** -	**context** JustifiableService::delete(R id) **post** -

3.1.28AbsenceController extends JustifiableController

Nome	Descrizione	Precondizione	Post condizione
+createAbsence(UUID studentId, Date date) :ResponseEntity<Absence>	Crea un'assenza per uno studente in una data specifica.	**context** AbsenceController::createAbsence(UUID studentId, Date date) :ResponseEntity<Absence> **pre** date <= Date::now **pre** AbsenceService::getByDate(studentId, date) = null **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** AbsenceController::createAbsence(UUID studentId, Date date) :ResponseEntity<Absence> **post** result.body.date = @pre.date **post** result.body.student = StudentService::getById(@pre.studentId)
+replaceAbsence(Date date, Long absenceId, UUID studentId) :ResponseEntity<Absence>	Sostituisce una giustificazione esistente con una nuova in una data specifica.	**context** AbsenceController::replaceAbsence(Date date, Long absenceId, UUID studentId) :ResponseEntity<Absence> **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId) **pre** AbsenceService::getByDate(studentId, date).id = absenceId	**context** AbsenceController::replaceAbsence(Date date, Long absenceId, UUID studentId) :ResponseEntity<Absence> **post** result.body.date = @pre.date **post** result.body.student = StudentService::getById(@pre.studentId)
+getById (Long id) :ResponseEntity<Absence>	Recupera una giustificazione specifica tramite l'ID.	**context** AbsenceController::getById (Long id) :ResponseEntity<Absence> **pre** -	**context** AbsenceController::getById (Long id) :ResponseEntity<Absence> **post** -
+getByDate (UUID idStudent, LocalDate	Recupera le giustificazioni di	**context** AbsenceController::getByDat	**context** AbsenceController::getByDat

date) :ResponseEntity<List<Absence>>	uno studente per una specifica data.	e (UUID idStudent, LocalDate date) :ResponseEntity<List< Absence >> **pre** date <= Date::now **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	e (UUID idStudent, LocalDate date) :ResponseEntity<List< Absence >> **post** -
+getByStudent(UUID idStudent) :ResponseEntity<List<Absence>>	Recupera tutte le giustificazioni di uno studente.	**context** AbsenceController::getByStudent(UUID idStudent) :ResponseEntity<List<Absence>> **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** AbsenceController::getByStudent(UUID idStudent) :ResponseEntity<List<Absence>> **post** -
+getTotal(UUID idStudent) :ResponseEntity<Integer>	Recupera il totale delle giustificazioni per uno studente.	**context** AbsenceController::getTotal(UUID idStudent) :ResponseEntity<Integer> **pre** -	**context** AbsenceController::getTotal(UUID idStudent) :ResponseEntity<Integer> **post** StudentService::getById(idStudent).absences.count()

3.1.29AbsenceService extends JustifiableService

Nome	Descrizione	Precondizione	Post condizione
+createAbsence(UUID studentId, Date date) : Absence	Crea un'assenza per uno studente in una data specifica.	**context** AbsenceService::createAbsence(UUID studentId, Date date) : Absence **pre** date <= Date::now **pre** self.getByDate(studentId, date) = null **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** AbsenceService::createAbsence(UUID studentId, Date date) : Absence **post** result.body.date = @pre.date **post** result.body.student = StudentService::getById(@pre.studentId)
+replaceAbsence(Date date, Long absenceId, UUID studentId) :Absence	Sostituisce una giustificazione esistente con una nuova in una data specifica.	**context** AbsenceService::replaceAbsence(Date date, Long absenceId, UUID studentId) :Absence **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId) **pre** self.getByDate(studentId, date).id = absenceId	**context** AbsenceService::replaceAbsence(Date date, Long absenceId, UUID studentId) :Absence **post** result.body.date = @pre.date **post** result.body.student = StudentService::getById(@pre.studentId)
+getById (Long id) :Absence	Recupera una giustificazione specifica tramite l'ID.	**context** AbsenceService::getById (Long id) :Absence **pre** -	**context** AbsenceService::getById (Long id) :Absence **post** -
+getByDate (UUID	Recupera le	**context**	**context**

idStudent, LocalDate date) :List<Absence>	giustificazioni di uno studente per una specifica data.	AbsenceService::getByDate (UUID idStudent, LocalDate date) : List< Absence > **pre** date <= Date::now **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	AbsenceService::getByDate (UUID idStudent, LocalDate date) : List< Absence > **post** -
+getByStudent(UUID idStudent) :List<Absence>	Recupera tutte le giustificazioni di uno studente.	**context** AbsenceService::getByStudent(UUID idStudent) : List<Absence> **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** AbsenceService::getByStudent(UUID idStudent) : List<Absence> **post** -
+getTotal(UUID idStudent) :Integer	Recupera il totale delle giustificazioni per uno studente.	**context** AbsenceService::getTotal(UUID idStudent) : Integer **pre** -	**context** AbsenceService::getTotal(UUID idStudent) : Integer **post** StudentService::getById(idStudent).absences.count()

3.1.30DelayController extends JustifiableController

Nome	Descrizione	Precondizione	Post condizione
+createDelay(UUID studentId, DateTime date) :ResponseEntity<Delay>	Crea un ritardo per uno studente in una data e ora specifica.	**context** DelayController::createDelay(UUID studentId, DateTime date) :ResponseEntity<Delay> **pre** date <= Date::now **pre** DelayService::getByDate(studentId, date) = null **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** DelayController::createDelay(UUID studentId, DateTime date) :ResponseEntity<Delay> **post** result.body.date = @pre.date **post** result.body.student = StudentService::getById(@pre.studentId)
+updateDelay(Time time, Long dalayId) :ResponseEntity<Delay>	Aggiorna un ritardo esistente, sostituendo l'orario specificato.	**context** DelayController::updateDelay(Time time, Long dalayId) :ResponseEntity<Delay> **pre** DelayService::getById(delayId) <> null **Pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = DelayService::getById(delayId).student.id)	**context** DelayController::updateDelay(Time time, Long dalayId) :ResponseEntity<Delay> **post** result.body.time = time
+getById (Long id) :ResponseEntity<Delay>	Recupera una giustificazione specifica tramite l'ID.	**context** DelayController::getById (Long id) :ResponseEntity< Delay > **pre** -	**context** DelayController::getById (Long id) :ResponseEntity< Delay > **post** -
+getByDate (UUID idStudent, LocalDate	Recupera le giustificazioni di uno	**context** DelayController::getByDat	**context** DelayController::getByDat

date) :ResponseEntity<List <Delay>>	studente per una specifica data.	e (UUID idStudent, LocalDate date) :ResponseEntity<List< Delay >> **pre** date <= Date::now **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId	e (UUID idStudent, LocalDate date) :ResponseEntity<List< Delay >> **post** -
+getByStudent(UUID idStudent) :ResponseEntity<List <Delay>>	Recupera tutte le giustificazioni di uno studente.	**context** DelayController::getByStudent(UUID idStudent) :ResponseEntity<List<Delay>> **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** DelayController::getByStudent(UUID idStudent) :ResponseEntity<List< Delay >> **post** -
+getTotal(UUID idStudent) :ResponseEntity<Integer>	Recupera il totale delle giustificazioni per uno studente.	**context** DelayController::getTotal(UUID idStudent) :ResponseEntity<Integer> **pre** -	**context** DelayController::getTotal(UUID idStudent) :ResponseEntity<Integer> **post** StudentService::getById(idStudent).dealys.count()

3.1.31DelayService extends JustifiableService

Nome	Descrizione	Precondizione	Post condizione
+createDelay(UUID studentId, DateTime date) : Delay	Crea un ritardo per uno studente in una data e ora specifica.	**context** DelayService::createDelay (UUID studentId, DateTime date) : Delay **pre** date <= Date::now **pre** self.getByDate(studentId, date) = null **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** DelayService::createDelay (UUID studentId, DateTime date) : Delay **post** result.body.date = @pre.date **post** result.body.student = StudentService::getById(@pre.studentId)
+updateDelay(Time time, Long dalayId) : Delay	Aggiorna un ritardo esistente, sostituendo l'orario specificato.	**context** DelayService::updateDelay(Time time, Long dalayId) : Delay **pre** self.getById(delayId) <> null **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = self.getById(delayId).student.id)	**context** DelayService::updateDelay(Time time, Long dalayId) : Delay **post** result.body.time = time
+getById (Long id) : Delay	Recupera una giustificazione specifica tramite l'ID.	**context** DelayService::getById (Long id) : Delay **pre** -	**context** DelayService::getById (Long id) :Delay **post** -
+getByDate (UUID idStudent, LocalDate date)	Recupera le giustificazioni di uno studente per una	**context** DelayService::getByDate (UUID idStudent,	**context** DelayService::getByDate (UUID idStudent,

: List<Delay>	specifica data.	LocalDate date) : List< Delay > **pre** date <= Date::now **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId	LocalDate date) : List< Delay > **post** -
+getByStudent(UUID idStudent) :ResponseEntity<List<Delay>>	Recupera tutte le giustificazioni di uno studente.	**context** DelayService::getByStudent(UUID idStudent) : List<Delay> **pre** StudentService::getById(idStudent) = UserService::getLoggedStudent() **pre** UserService::getLoggedTeacher().schoolClasses.exist(c | c.studets.exist(s | s.id = studentId)	**context** DelayService::getByStudent(UUID idStudent) : List< Delay > **post** -
+getTotal(UUID idStudent) :ResponseEntity<Integer>	Recupera il totale delle giustificazioni per uno studente.	**context** DelayService::getTotal(UUID idStudent) : Integer **pre** -	**context** DelayService::getTotal(UUID idStudent) : Integer **post** StudentService::getById(idStudent).dealys.count()

3.1.32HomeworkController extends CrudController

Nome	Descrizione	Precondizione	Post condizione
+addChat(HomeworkID id): ResponseEntity<HomeworkResponse>	Aggiunge una chat associata a un compito specificato tramite ID.	**context** HomeworkController::addChat(HomeworkID id): ResponseEntity<HomeworkResponse> **pre** HomeworkService::getById(id).needChat = false	**context** HomeworkController::addChat(HomeworkID id): ResponseEntity<HomeworkResponse> **post** HomeworkService::getById(@pre.id).needChat = true **post** HomeworkService::getById(@pre.id).signHour.teachingTimeslot.classTimetable.schoolClass.students.forAll(s | HomeworkChatService::createHomeworkChat(HomeworkService::getById(@pre.id), HomeworkService::getById(@pre.id).signHour)
+updateDescription(HomeworkID id, String description): ResponseEntity<HomeworkResponse>	Aggiorna la descrizione di un compito specificato tramite ID.	**context** HomeworkController::updateDescription(HomeworkID id, String description): ResponseEntity<HomeworkResponse> **pre** description > 0 **pre** HomeworkService::getById(id) <> null	**context** HomeworkController::updateDescription(HomeworkID id, String description): ResponseEntity<HomeworkResponse> **post** result.body.description = @pre.description
+updateDueDate(HomeworkID id, Date date): ResponseEntity<HomeworkResponse>	Aggiorna la data di scadenza di un compito specificato tramite ID.	**context** HomeworkController::updateDueDate(HomeworkID id, Date date): ResponseEntity<HomeworkResponse> **pre** date > Date::now() **pre** HomeworkService::getById(id) <> null	**context** HomeworkController::updateDueDate(HomeworkID id, Date date): ResponseEntity<HomeworkResponse> **post** result.body.date = @pre.date

3.1.33HomeworkService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+ addChat(HomeworkID id): HomeworkResponse	Aggiunge una chat a un compito specificato tramite ID.	**context** HomeworkService:: addChat(HomeworkID id): HomeworkResponse **pre** self.getById(id).needChat = false	**context** HomeworkService:: addChat(HomeworkID id): HomeworkResponse **post** self.getById(@pre.id).needChat = true **post** self.getById(@pre.id).signHour.teachingTimeslot.classTimetable.schoolClass.studets.forAll(s | HomeworkChatService::createHomeworkChat(HomeworkService::getById(@pre.id), self.getById(@pre.id).signHour)
+updateDescription(HomeworkID id, String description): HomeworkResponse	Aggiorna la descrizione di un compito specificato tramite ID.	**context** HomeworkService:: updateDescription(HomeworkID id, String description): HomeworkResponse **pre** description > 0 **pre** HomeworkService::getById(id) <> null	**context** HomeworkService:: updateDescription(HomeworkID id, String description): HomeworkResponse **post** result.body.description = @pre.description
+updateDueDate(HomeworkID id, Date date): HomeworkResponse	Aggiorna la data di scadenza di un compito specificato tramite ID.	**context** HomeworkService:: updateDueDate(HomeworkID id, Date date): HomeworkResponse **pre** date > Date::now() **pre** self.getById(id) <> null	**context** HomeworkService:: updateDueDate(HomeworkID id, Date date): HomeworkResponse **post** result.body.date = @pre.date

3.1.34ClassActivityController extends CrudController

Questa classe è necessaria per definire il percorso REST per la gestione delle attività svolte in classe. Tutti i metodi sono ereditati da CrudController.

Nome	Descrizione	Precondizione	Post condizione
-	-	-	-

3.1.35ClassActivityService extends CrudService

Questa classe è necessaria per definire il percorso REST per la gestione delle attività svolte in classe. Tutti i metodi sono ereditati da CrudService.

Nome	Descrizione	Precondizione	Post condizione
-	-	-	-

3.1.36ClassCommunicationController extends CrudController

Questa classe è necessaria per definire il percorso REST per la gestione degli avvisi.
Tutti i metodi sono ereditati da CrudController.

Nome	Descrizione	Precondizione	Post condizione
-	-	-	-

3.1.37ClassCommunicationService 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
-	-	-	-

3.1.38ChatController<ID>

Nome	Descrizione	Precondizione	Post condizione
+ addMessage(String text, request, ID chatId):ResponseEntity<MessageResponse>	Aggiunge un messaggio a una chat specificata dall'ID.	**context** ChatController::addMessage(String text, request, ID chatId):ResponseEntity<MessageResponse> **pre** text.size() > 0	**context** ChatController::addMessage(String text, request, ID chatId):ResponseEntity<MessageResponse> **post** result.body = (ChatService::getById (chatId).getMessages().size() = @pre.ChatService::getById (chatId).getMessages().size() + 1)
+ getAllMessageById(ID id):ResponseEntity<MessageResponse>	Restituisce tutti i messaggi associati a una specifica chat identificata da id.	**context** ChatController::getAllMessageById(ID id):ResponseEntity<MessageResponse> **pre** -	**context** ChatController::getAllMessageById(ID id):ResponseEntity<MessageResponse> **post** result.body = ChatService::getById (id).getMessages
+sendMessageWithAttachment(String text, File file, ID chatId):ResponseEntity< MessageResponse >	Invia un messaggio con allegato a una chat specificata dall'ID.	**context** ChatController::sendMessageWithAttachment(String text, File file, ID chatId):ResponseEntity< MessageResponse > **pre** text.size() > 0 **pre** file.size() < 16Mb	**context** ChatController::sendMessageWithAttachment(String text, File file, ID chatId):ResponseEntity< MessageResponse > **post** result.body = (ChatService::getById (chatId).getMessages().size() = @pre.ChatService::getById (chatId).getMessages().size() + 1)
+ getSender(UUID messageId):Respons	Restituisce le informazioni sul	**context** ChatController::getSender(UUID	**context** ChatController::getSender(UUID

eEntity<SenderResp ose>	mittente di un messaggio specificato dall'ID del messaggio.	messageId):ResponseEnti ty<SenderRespose> **pre** -	messageId):ResponseEnti ty<SenderRespose> **post** -
+ getMessageList(ID id):ResponseEntity<Li st<Message>>	Restituisce la lista di tutti i messaggi per una chat specificata dall'ID.	**context** ChatController:: getMessageList(ID id):ResponseEntity<List< Message>> **pre** -	**context** ChatController:: getMessageList(ID id):ResponseEntity<List< Message>> **post** result = ChatService::getById (id).getMessages()

3.1.39ChatService<ID>

Nome	Descrizione	Precondizione	Post condizione
+addMessage(String text, request, ID chatId): MessageResponse	Aggiunge un messaggio alla chat identificata da chatId.	**context** ChatService:: addMessage(String text, request, ID chatId): MessageResponse **pre** text.size() > 0	**context** ChatService:: addMessage(String text, request, ID chatId): MessageResponse **post** result = (ChatService::getById (chatId).getMessages().siz e() = @pre.ChatService::getByI d (chatId).getMessages().siz e() + 1)
+ getAllMessageById(I D id): MessageResponse	Restituisce tutti i messaggi associati a una chat specificata da id.	**context** ChatService:: getAllMessageById(ID id): MessageResponse **pre** -	**context** ChatService:: getAllMessageById(ID id): MessageResponse **post** result = ChatService::getById (id).getMessages
+sendMessageWithA ttachment(String text, File file, ID chatId):	Invia un messaggio con allegato a una chat specificata dall'ID.	**context** ChatService:: sendMessageWithAttach ment(String text, File file, ID chatId): MessageResponse	**context** ChatService:: sendMessageWithAttach ment(String text, File file, ID chatId): MessageResponse

MessageResponse		**pre** text.size() > 0 **pre** file.size() < 16Mb	**post** result = (ChatService::getById (chatId).getMessages().size() = @pre.ChatService::getById (chatId).getMessages().size() + 1)
+ getSender(UUID messageId): SenderRespose	Restituisce le informazioni sul mittente di un messaggio identificato dall'ID del messaggio.	**context** ChatService:: getSender(UUID messageId): SenderRespose **pre** -	**context** ChatService:: getSender(UUID messageId): SenderRespose **post** -
+ getMessageList(ID id):List<Message>	Restituisce la lista di tutti i messaggi di una chat specificata dall'ID.	**context** ChatService:: getMessageList(ID id):List<Message> **pre** -	**context** ChatService:: getMessageList(ID id):List<Message> **post** result = ChatService::getById (id).getMessages()

3.1.40HomeworkChatController extends ChatController

Nome	Descrizione	Precondizione	Post condizione
+ createHomeworkChat(HomeworkId signHourId):ResponseEntity<HomeworkChatResponse>	Crea una chat associata a un compito e a uno specifico orario firmato.	**context** HomeworkChatController::createHomeworkChat(HomeworkId signHourId):ResponseEntity<HomeworkChatResponse> **pre** HomeworkService::getById(HomeworkId) <> null **pre** UserService::getLoggedTeacher().signedTimeslots.exist(t | t.signedHours.exist(s | s.id = signHourId))	**context** HomeworkChatController::createHomeworkChat(HomeworkId signHourId):ResponseEntity<HomeworkChatResponse> **post** result.body.homework = HomeworkService::getById(HomeworkId) **post** result.body.signHour = UserService::getLoggedTeacher().signedTimeslots.select(t | t.signedHours.exist(s | s.id = signHourId)).signedHours.select(s | s.id = signHourId)
+ complete(SignHourID id):ResponseEntity<HomeworkChatResponse>	Segna tutte le chat di un compito specifico come completate per l'orario firmato specificato.	**context** HomeworkChatController::complete(SignHourID signHourId):ResponseEntity<HomeworkChatResponse> **pre** UserService::getLoggedTeacher().signedTimeslots.exist(t | t.signedHours.exist(s | s.id = id))	**context** HomeworkChatController::complete(SignHourID signHourId):ResponseEntity<HomeworkChatResponse> **post** result = UserService::getLoggedTeacher().signedTimeslots.select(t | t.signedHours.exist(s | s.id = id)).signedHours.select(s | s.id = id).homeworks.forAll(h |

			h.homeworkChats.forAll(c | c.completed = True))
+ completeWithFeedback(SignHourID signHourId, String text):ResponseEntity < HomeworkChatResponse>	Segna tutte le chat come completate e aggiunge un messaggio finale con feedback.	**context** HomeworkChatController:: completeWithFeedback(SignHourID signHourId, String text):ResponseEntity< HomeworkChatResponse > **pre** text.size() > 0 **pre** UserService:: getLoggedTeacher().signedTimeslots.exist(t | t.signedHours.exist(s | s.id = id))	**context** HomeworkChatController:: completeWithFeedback(SignHourID signHourId, String text):ResponseEntity< HomeworkChatResponse > **post** let homeworkChats = UserService:: getLoggedTeacher().signedTimeslots.select(t | t.signedHours.exist(s | s.id = id)).signedHours.select(s | s.id = id).homeworks.collect(h | h. homeworkChats) **post** homeworkChats.forAll(c | c.completed = True) **post** homeworkChats.forAll(c | c.getMessageList().getLast().text = text)

3.1.41HomeworkChatService extends ChatService

Nome	Descrizione	Precondizione	Post condizione
+createHomeworkChat(HomeworkId signHourId):ResponseEntity<HomeworkChatResponse>	Crea una chat associata a un compito e a uno specifico orario firmato.	**context** HomeworkChatService::createHomeworkChat(HomeworkId signHourId):ResponseEntity<HomeworkChatResponse> **pre** HomeworkService::getById(HomeworkId) <> null **pre** UserService::getLoggedTeacher().signedTimeslots.exist(t | t.signedHours.exist(s | s.id = signHourId))	**context** HomeworkChatService::createHomeworkChat(HomeworkId signHourId):ResponseEntity<HomeworkChatResponse> **post** result.body.homework = HomeworkService::getById(HomeworkId) **post** result.body.signHour = UserService::getLoggedTeacher().signedTimeslots.select(t | t.signedHours.exist(s | s.id = signHourId)).signedHours.select(s | s.id = signHourId)
+ complete(SignHourID signHourId):ResponseEntity< HomeworkChatResponse>	Segna tutte le chat di un compito specifico come completate per l'orario firmato specificato.	**context** HomeworkChatService::complete(SignHourID signHourId):ResponseEntity< HomeworkChatResponse> **pre** UserService::getLoggedTeacher().signedTimeslots.exist(t | t.signedHours.exist(s | s.id = id))	**context** HomeworkChatService::complete(SignHourID signHourId):ResponseEntity< HomeworkChatResponse> **post** result = UserService::getLoggedTeacher().signedTimeslots.select(t | t.signedHours.exist(s | s.id = id)).signedHours.select(s | s.id = id).homeworks.forAll(h | h.homeworkChats.forAll(c | c.completed = True))

+completeWithFeedback(SignHourID signHourId, String text):ResponseEntity< HomeworkChatResponse>	Segna tutte le chat come completate e aggiunge un messaggio finale con feedback.	**context** HomeworkChatService:: completeWithFeedback(SignHourID signHourId, String text):ResponseEntity< HomeworkChatResponse > **pre** text.size() > 0 **pre** UserService:: getLoggedTeacher().signedTimeslots.exist(t | t.signedHours.exist(s | s.id = id))	**context** HomeworkChatService:: completeWithFeedback(SignHourID signHourId, String text):ResponseEntity< HomeworkChatResponse > **post** let homeworkChats = UserService:: getLoggedTeacher().signedTimeslots.select(t | t.signedHours.exist(s | s.id = id)).signedHours.select(s | s.id = id).homeworks.collect(h | h. homeworkChats) **post** result = homeworkChats.forAll(c | c.completed = True) **post** homeworkChats.forAll(c | c.getMessageList().getLast().text = text)

3.1.42TicketChatController extends ChatController

Nome	Descrizione	Precondizione	Post condizione
+createParentTicket(TicketRequest request):ResponseEntity<TicketResponse>	Crea un ticket associato a un genitore e una chat.	**context** TicketChatController::createParentTicket(TicketRequest request):ResponseEntity<TicketResponse> **pre** UserService::getLoggedParent().id = request.parentId	**context** TicketChatController::createParentTicket(TicketRequest request):ResponseEntity<TicketResponse> **pre** result = ChatService::getById(request.chatId) <> null
+ createTeacherTicket(TicketRequest request):ResponseEntity<TicketResponse>	Crea un ticket associato a un insegnante e una chat.	**context** TicketChatController::createTeacherTicket(TicketRequest request):ResponseEntity<TicketResponse> **pre** UserService::getLoggedTeacher().id = request.teacherId	**context** TicketChatController::createTeacherTicket(TicketRequest request):ResponseEntity<TicketResponse> **pre** result = ChatService::getById(request.chatId) <> null
+ deleteTicket(UUID id)	Elimina un ticket esistente.	**context** TicketChatController::deleteTicket(UUID id) **pre** ChatService::getById(id) <> null	**context** TicketChatController::deleteTicket(UUID id) **post** ChatService::getById(id) = null
+ closeUnresolved(UUID id):ResponseEntity<TicketResponse>	Chiude un ticket aperto impostandolo come "non risolto".	**context** TicketChatController::closeUnresolved(UUID id:ResponseEntity<TicketResponse> **pre** ChatService::getById(id) <> null **pre** ChatService::getById(id).status = “open”	**context** TicketChatController::closeUnresolved(UUID id:ResponseEntity<TicketResponse> **pre** result.body = ChatService::getById(id).status = “unresolved”
+	Chiude un ticket aperto	**context**	**context**

closeResolved(UUID id):ResponseEntity<TicketResponse>	impostandolo come "risolto".	TicketChatController::closeResolved(UUID id:ResponseEntity<TicketResponse> **pre** ChatService::getById(id) <> null **pre** ChatService::getById(id).status = “open”	TicketChatController::closeResolved(UUID id:ResponseEntity<TicketResponse> **pre** result.body = ChatService::getById(id).status = “close”
+ reopen(UUID id):ResponseEntity<TicketResponse>	Riapre un ticket precedentemente contrassegnato come "non risolto".	**context** TicketChatController::reopen(UUID id:ResponseEntity<TicketResponse> **pre** ChatService::getById(id) <> null **pre** ChatService::getById(id).status = “unresolved”	**context** TicketChatController::reopen(UUID id:ResponseEntity<TicketResponse> **pre** result.body = ChatService::getById(id).status = “open”

3.1.43TicketChatService extends ChatService

Nome	Descrizione	Precondizione	Post condizione
+ createParentTicket(TicketRequest request): TicketResponse	Crea un ticket associato a un genitore e una chat.	**context** TicketChatService:: createParentTicket(TicketRequest request): TicketResponse **pre** UserService:: getLoggedParent().id = request.parentId	**context** TicketChatService:: createParentTicket(TicketRequest request): TicketResponse **pre** result = ChatService::getById(request.chatId) <> null
+ createTeacherTicket(TicketRequest request): TicketResponse	Crea un ticket associato a un insegnante e una chat.	**context** TicketChatService:: createTeacherTicket(TicketRequest request): TicketResponse **pre** UserService:: getLoggedTeacher().id = request.teacherId	**context** TicketChatService:: createTeacherTicket(TicketRequest request): TicketResponse **pre** result = ChatService::getById(request.chatId) <> null
+ deleteTicket(UUID id)	Elimina un ticket esistente.	**context** TicketChatService:: deleteTicket(UUID id) **pre** ChatService::getById(id) <> null	**context** TicketChatService:: deleteTicket(UUID id) **post** ChatService::getById(id) = null
+ closeUnresolved(UUID id: TicketResponse	Chiude un ticket aperto impostandolo come "non risolto".	**context** TicketChatService::closeUnresolved(UUID id: TicketResponse **pre** ChatService::getById(id) <> null **pre** ChatService::getById(id).status = “open”	**context** TicketChatService::closeUnresolved(UUID id: TicketResponse **pre** result.body = ChatService::getById(id).status = “unresolved”
+ closeResolved(UUID id:ResponseEntity<TicketResponse>	Chiude un ticket aperto impostandolo come "risolto".	**context** TicketChatController::closeResolved(UUID id:ResponseEntity<Ticket	**context** TicketChatController::closeResolved(UUID id:ResponseEntity<Ticket

		Response> **pre** ChatService::getById(id) <> null **pre** ChatService::getById(id).status = “open”	Response> **pre** result.body = ChatService::getById(id).status = “close”
+ reopen(UUID id:ResponseEntity<TicketResponse>	Riapre un ticket precedentemente contrassegnato come "non risolto".	**context** TicketChatController::reopen(UUID id:ResponseEntity<TicketResponse> **pre** ChatService::getById(id) <> null **pre** ChatService::getById(id).status = “unresolved”	**context** TicketChatController::reopen(UUID id:ResponseEntity<TicketResponse> **pre** result.body = ChatService::getById(id).status = “open”

3.1.44ClassManagementController extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+ addStudyPlan(Long classId, Long studyPlanId)	Aggiunge un piano di studi a una specifica classe identificata da classId.	**context** ClassManagementController:: addStudyPlan(Long classId, Long studyPlanId)	**context** ClassManagementController:: addStudyPlan(Long classId, Long studyPlanId)
+ addStudent(UUID studentId, Long id)	Aggiunge uno studente a una classe specifica.	**context** ClassManagementController:: addStudent(UUID studentId, Long id) **pre** self.getById(id) <> null **pre** UserService::getById(studentId) <> null	**context** ClassManagementController:: addStudent(UUID studentId, Long id) **post** result.body = self.getById(id).students.exist(s | s.id = studentId)
+ addStudents(List<UUID> studentsId, Long id)	Aggiunge una lista di studenti a una classe specifica.	**context** ClassManagementController:: addStudents(List<UUID> studentsId, Long id) **pre** self.getById(id) <> null **pre** studentsId.forAll(studId | UserService::getById(studId) <> null)	**context** ClassManagementController:: addStudents(List<UUID> studentsId, Long id) **post** result.body = studentsId.forAll(studId | self.getById(id).students.exist(s | s.id = studId))
+addTeacher(TeacherClassRequest request)	Aggiunge un insegnante a una classe specifica in base ai dettagli forniti nella richiesta.	**context** ClassManagementController:: addTeacher(TeacherClassRequest request) **pre** self.getById(id) <> null **pre** UserService::getById(request.teacherId) <> null	**context** ClassManagementController:: addTeacher(TeacherClassRequest request) **post** result.body = self.getById(id).teachers.exist(t | t.id = request.teacherId)

3.1.45ClassManagementService extends CrudService

Nome	Descrizione	Precondizione	Post condizione
+ addStudyPlan(Long classId, Long studyPlanId)	Aggiunge un piano di studi a una specifica classe identificata da classId.	**context** ClassManagementService :: addStudyPlan(Long classId, Long studyPlanId)	**context** ClassManagementService :: addStudyPlan(Long classId, Long studyPlanId)
+ addStudent(UUID studentId, Long id)	Aggiunge uno studente a una classe specifica.	**context** ClassManagementService :: addStudent(UUID studentId, Long id) **pre** self.getById(id) <> null **pre** UserService::getById(studentId) <> null	**context** ClassManagementService :: addStudent(UUID studentId, Long id) **post** result.body = self.getById(id).students.exist(s | s.id = studentId)
+ addStudents(List<UUID> studentsId, Long id)	Aggiunge una lista di studenti a una classe specifica.	**context** ClassManagementService :: addStudents(List<UUID> studentsId, Long id) **pre** self.getById(id) <> null **pre** studentsId.forAll(studId | UserService::getById(studId) <> null)	**context** ClassManagementService :: addStudents(List<UUID> studentsId, Long id) **post** result.body = studentsId.forAll(studId | self.getById(id).students.exist(s | s.id = studId))
+addTeacher(TeacherClassRequest request)	Aggiunge un insegnante a una classe specifica in base ai dettagli forniti nella richiesta.	**context** ClassManagementService :: addTeacher(TeacherClassRequest request) **pre** self.getById(id) <> null **pre** UserService::getById(request.teacherId) <> null	**context** ClassManagementService :: addTeacher(TeacherClassRequest request) **post** result.body = self.getById(id).teachers.exist(t | t.id = request.teacherId)

3.1.46AcademincController

Nome	Descrizione	Precondizione	Post condizione
+ createTeaching (TeachingRequest request): ResponseEntity<TeachingResponse>	Crea un nuovo insegnamento associandolo a un docente e a una materia.	**context** AcademincController::createTeaching (TeachingRequest request): ResponseEntity<TeachingResponse> **pre** self.getById(request.subjectId) <> null **pre** UserService::getById(request.teacherId) <> null	**context** AcademincController::createTeaching (TeachingRequest request): ResponseEntity<TeachingResponse> **post** result.body = (UserService::getById (request.teacherId).teachings.size() = @pre.UserService::getById (request.teacherId).teachings.size() + 1)
+ replaceTeaching (TeachingRequest request): ResponseEntity<TeachingResponse>	Sostituisce un insegnamento esistente con uno nuovo.	**context** AcademincController::replaceTeaching (TeachingRequest request): ResponseEntity<TeachingResponse> **pre** UserService::getById(request.subjectId) <> null **pre** UserService::getById(request.teacherId).teachings.exists(t | t.id = request.oldTeachingId)	**context** AcademincController::replaceTeaching (TeachingRequest request): ResponseEntity<TeachingResponse> **post** result.body = UserService::getById(request.teacherId).teachings.exists(t | t.subject.id = request.subjectId) **post** UserService::getById(request.teacherId).teachings.forAll(t | t.id <> request.oldTeachingId)
+ deleteTeaching (TeachingRequest request)	Elimina un insegnamento specificato.	**context** AcademincController::deleteTeaching (TeachingRequest request) **pre** self.getById(request.tachingId) <> null	**context** AcademincController::deleteTeaching (TeachingRequest request) **post** result = self.getById(request.tachingId) = null
+ replaceTeaching (TeachingRequest request): ResponseEntity<Teac		**context** AcademincController::replaceTeaching (TeachingRequest request): ResponseEntity<TeachingR	**context** AcademincController::replaceTeaching (TeachingRequest request): ResponseEntity<TeachingRespo

hingResponse>		esponse>	nse>
+ deleteTeaching (TeachingRequest request)		**context** AcademincController::deleteTeaching (TeachingRequest request)	**context** AcademincController::deleteTeaching (TeachingRequest request)
+createSubject(String subject): ResponseEntity<Subject>	Crea una nuova materia con il nome specificato.	**context** AcademincController::createSubject(String subject): ResponseEntity<Subject> **pre** subject.size() > 0	**context** AcademincController::createSubject(String subject): ResponseEntity<Subject> **post** result.body.subject = subject
+ replaceSubject (Subject request): ResponseEntity<Subject>	Sostituisce una materia esistente con una nuova definizione.	**context** AcademincController::replaceSubject (Subject request): ResponseEntity<Subject> **pre** self.getById(request.oldSubjectId) <> null **pre** request.text.size() > 0	**context** AcademincController::replaceSubject (Subject request): ResponseEntity<Subject> **post** result.body = self.getById(request.oldSubjectId) = null
+ deleteSubject(Subject request)	Elimina una materia specificata.	**context** AcademincController::deleteSubject(Subject request) **pre** self.getById(request.subjectId) <> null	**context** AcademincController::deleteSubject(Subject request) **post** result.body = self.getById(request.subjectId) = null
+ createStudyPlan(String title):ResponseEntity<StudyPlanResponse>	Crea un nuovo piano di studi con il titolo specificato.	**context** AcademincController::createStudyPlan(String title):ResponseEntity<StudyPlanResponse> **pre** title.size() > 0	**context** AcademincController::createStudyPlan(String title):ResponseEntity<StudyPlanResponse> **post** request.body.title = title

3.1.47AcademicService

Nome	Descrizione	Precondizione	Post condizione
+ createTeaching (TeachingRequest request): TeachingResponse	Crea un nuovo insegnamento associandolo a un docente e a una materia.	**context** AcademicService::createTeaching (TeachingRequest request): TeachingResponse **pre** self.getById(request.subjectId) <> null **pre** UserService::getById(request.teacherId) <> null	**context** AcademicService::createTeaching (TeachingRequest request): TeachingResponse **post** result.body = (UserService::getById (request.teacherId).teachings.size() = @pre.UserService::getById (request.teacherId).teachings.size() + 1)
+ replaceTeaching (TeachingRequest request): TeachingResponse	Sostituisce un insegnamento esistente con uno nuovo.	**context** AcademicService::replaceTeaching (TeachingRequest request): TeachingResponse **pre** UserService::getById(request.subjectId) <> null **pre** UserService::getById(request.teacherId).teachings.exists(t | t.id = request.oldTeachingId)	**context** AcademicService::replaceTeaching (TeachingRequest request): TeachingResponse **post** result.body = UserService::getById(request.teacherId).teachings.exists(t | t.subject.id = request.subjectId) **post** UserService::getById(request.teacherId).teachings.forAll(t | t.id <> request.oldTeachingId)
+ deleteTeaching (TeachingRequest request)	Elimina un insegnamento specificato.	**context** AcademicService::deleteTeaching (TeachingRequest request) **pre** self.getById(request.tachingId) <> null	**context** AcademicService::deleteTeaching (TeachingRequest request) **post** result = self.getById(request.tachingId) = null
+ createTeaching (TeachingRequest request): TeachingResponse		**context** AcademicService::createTeaching (TeachingRequest request):	**context** AcademicService::createTeaching (TeachingRequest request): TeachingResponse

		TeachingResponse	
+ replaceTeaching (TeachingRequest request): TeachingResponse		**context** AcademincService::replaceTeaching (TeachingRequest request): TeachingResponse	**context** AcademincService::replaceTeaching (TeachingRequest request): TeachingResponse
+ deleteTeaching (TeachingRequest request)		**context** AcademincService::deleteTeaching (TeachingRequest request)	**context** AcademincService::deleteTeaching (TeachingRequest request)
+createSubject(String subject): Subject	Crea una nuova materia con il nome specificato.	**context** AcademincService::createSubject(String subject): Subject **pre** subject.size() > 0	**context** AcademincService::createSubject(String subject): Subject **post** result.body.subject = subject
+ replaceSubject (Subject request): Subject	Sostituisce una materia esistente con una nuova definizione.	**context** AcademincService::replaceSubject (Subject request): Subject **pre** self.getById(request.oldSubjectId) <> null **pre** request.text.size() > 0	**context** AcademincService::replaceSubject (Subject request): Subject **post** result.body = self.getById(request.oldSubjectId) = null
+deleteSubject(Subject request)	Elimina una materia specificata.	**context** AcademincService::deleteSubject(Subject request) **pre** self.getById(request.subjectId) <> null	**context** AcademincService::deleteSubject(Subject request) **post** result.body = self.getById(request.subjectId) = null
+createStudyPlan(String title): StudyPlanResponse	Crea un nuovo piano di studi con il titolo specificato.	**context** AcademincService:: createStudyPlan(String title): StudyPlanResponse **pre** title.size() > 0	**context** AcademincService:: createStudyPlan(String title): StudyPlanResponse **post** request.body.title = title

3.1.48OrientationController

Nome	Descrizione	Precondizione	Post condizione
+attitude(UUID idStudent):ResponseEntity<String>	Restituisce un'indicazione sul comportamento di uno studente specificato dal suo idStudent.	**context** OrientationController::attitude(UUID idStudent):ResponseEntity<String>	**context** OrientationController::attitude(UUID idStudent):ResponseEntity<String>

3.1.49PythonService

Nome	Descrizione	Precondizione	Post condizione
+attitude(UUID idStudent): String	Restituisce una valutazione dell'atteggiamento di uno studente.	**context** PythonService::attitude(UUID idStudent): String **pre** StudetService::getById(idStudent) <> null	**context** PythonService::attitude(UUID idStudent): String **post** -
+getCategory(String text): Ticket.Category	Determina la categoria del ticket in base al testo fornito.	**context** PythonService::getCategory(String text): Ticket.Category **pre** text.size() > 10	**context** PythonService::getCategory(String text): Ticket.Category **post** -

3.2Specifica DTO

Class Diagram dei DTO utilizzati:
(prodotta successivamente)

3.3Documentazione REST

Durante le successive fasi di design sarà realizzata una documentazione completa di tutti gli endpoint REST esposti dall'applicazione con il support di tool automatici come Spring REST Docs.
(prodotta successivamente)

Nome metodo	Verbo HTTP	API endpoint
-	-	-

3.4 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 aggiornata dei componenti React, con ulteriori opportunità di riuso identificate.

Comuni a più utenti

- *FieldComponent*: Campo configurabile per l'inserimento di testo o dati.
- Form: Componente generico per gestire form complessi come login, registrazione o modifica dati.
- UserSetting: Schermata per visualizzare e modificare le impostazioni dell'utente.
- Login: Schermata di login con selezione del ruolo.
- HomePage: Homepage generica per utenti non registrati o non loggati.
- Dashboard: Vista generica della dashboard personalizzata per ogni utente.
- Modal: Componente per dialoghi, conferme o notifiche.
- Button: Pulsante riutilizzabile per diverse azioni, configurabile per stili e comportamenti specifici.

Utenti di una scuola

- FirstLogin: Schermata per il cambio password al primo accesso.
- Timeslot: Componente configurabile per rappresentare slot orari.
- UserInfoForm: Modulo per la modifica o inserimento di informazioni dell'utente.

Chat e ticket

- TicketCreationForm: Form generico per aprire ticket o richieste.
- Chat: Schermata per conversazioni private tra utenti o con la segreteria.
- MessageCard: Componente per raggruppare testo e allegati nei messaggi.
- FileUpload: Pulsante per caricare file, riutilizzabile anche in altri moduli (es. compiti o allegati).
- ConversationList: Lista generica di conversazioni o ticket con stato visibile

Didattica per genitori e studenti

- Orientation: Sezione dedicata all'orientamento in uscita.
- YearChooser: Componente per selezionare l'anno scolastico o storico.
- Absence: Schermata per riepilogo assenze e ritardi.
- Communication: Lista di avvisi o comunicazioni scolastiche.
- Homework: Lista di compiti assegnati, configurabile per diverse viste.
- Mark: Schermata riepilogativa dei voti, riutilizzabile per studenti e genitori.

- AverageProgress: Barra di avanzamento configurabile per statistiche (es. media o completamento).
- AnalyticChart: Grafico configurabile per mostrare andamenti o trend.
- DatePicker: Componente per selezionare intervalli di date, utilizzabile anche per altre funzionalità.
- Report: Vista generica per riepiloghi come pagelle o scrutini.
- SearchField: Campo di ricerca generico per selezionare elementi come professori o classi.
- ClassTimetable: Schermata per visualizzare orari delle lezioni.
- Activity: Schermata per attività svolte in classe.
- LadChooser: Componente per selezionare uno studente o un altro elemento dall'elenco.
- CommunicationForm: Form configurabile per creare comunicazioni o avvisi.

Insegnamento

- TeacherDashboard: Dashboard generica per i professori.
- TeacherActivityForm: Form per inserire dettagli su attività o compiti svolti.
- SignHour: Pulsante per firmare ore o presenze, riutilizzabile per altre conferme.
- ClassAppeal: Schermata per inserire o visualizzare assenze e ritardi.
- TeacherReportForm: Form configurabile per gestire voti o scrutini.
- TeacherMark: Vista per la gestione e visualizzazione delle valutazioni.
- PublishMark: Pulsante configurabile per inserire voti o approvare elementi.
- ReceptionManagement: Gestione ricevimenti con genitori, riutilizzabile anche per altre interazioni programmate.

Genitore

- ParentDashboard: Dashboard per genitori, personalizzabile con moduli aggiuntivi.
- ParentHomework: Vista compiti, riutilizzabile con altri dati come attività o voti.
- ReceptionTimetableForm: Form generico per prenotazioni o richieste programmate.
- ParentTicket: Vista per la gestione di ticket aperti.

Segreteria

- SecretaryDashboard: Dashboard della segreteria con pannelli configurabili.
- EditTimetableForm: Form per modificare dati complessi come orari o assegnazioni.
- SecretaryClassDashboard: Gestione classi, riutilizzabile per diverse tipologie di utenti.
- SecretaryUserCreationForm: Form generico per la creazione di nuovi account o ruoli.
- SecretaryStudentList: Lista studenti con funzioni di filtro e ricerca, utilizzabile anche per genitori o docenti.

Gestione scuola

- SchoolManagerDashboard: Schermata per gestire scuole o istituti.
- SchoolCreationForm: Form generico per creare o modificare entità complesse.
- SchoolInfoForm: Modulo per aggiungere o aggiornare informazioni sull'istituto.

Tabella orario

- Timetable: Vista configurabile per tabelle orarie di classi o docenti.
- TeachingChooser: Componente per selezionare insegnamenti o materie, utilizzabile in più contesti.

4. Glossario

Termine	Definizione
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, un componente raccoglie dati da diverse API e li organizza 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.
CRUD	Create, Read, Update, Delete – Operazioni di base per la gestione dei dati in un'applicazione.
HTTP	HyperText Transfer Protocol – Protocollo di trasferimento dati utilizzato per le comunicazioni web.
JPA	Java Persistence API – Specifica Java per la gestione della persistenza dei dati tra le applicazioni Java e i database relazionali.
JSDoc	JavaScript Documentation – Strumento di documentazione per il linguaggio JavaScript, simile a Javadoc per Java.
Javadoc	Strumento di documentazione per il linguaggio Java, utilizzato per generare documentazione API a partire dal codice sorgente.
JWT	JSON Web Token – Standard aperto per la trasmissione sicura di informazioni tra le parti come oggetti JSON.
OCL	Object Constraint Language – Linguaggio utilizzato per specificare restrizioni e vincoli nei modelli UML.
REST	Representational State Transfer – Stile architetturale per la progettazione di servizi web che utilizza le operazioni HTTP.
UML	Unified Modeling Language – Linguaggio di modellazione standardizzato utilizzato per specificare, visualizzare, costruire e documentare gli artefatti di sistemi software.
API	Application Programming Interface – Insieme di regole e specifiche che le applicazioni possono seguire per comunicare tra loro.
CSS	Cascading Style Sheets – Linguaggio utilizzato per descrivere la presentazione di documenti HTML o XML.
Spring Boot	Framework Java per lo sviluppo di applicazioni Spring con configurazioni automatiche e componenti pronti all'uso.
React	Libreria JavaScript per la costruzione di interfacce utente interattive e component-based.

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