

Task R5: Gestione dei Team di Utenti e degli Esercizi assegnati da parte di un Admin

Software Architecture Design, Anno 2024/2025

Team R5: Perillo Giulio, Del Vecchio Federica, Esposito Francesco, Mauriello Valentina

Matricola: M63001726, M63001587, M63001830, M63001631

Email: giulio.perillo@studenti.unina.it, feder.delvecchio@studenti.unina.it, francesco.esposito158@studenti.unina.it ,
valen.mauriello@studenti.unina.it

Repository GitHub: <https://github.com/DelVecchio-Esposito-Mauriello-Perillo/A13>

Sommario

1. INTRODUZIONE	4
1.1. Punto di partenza	4
1.2. Nuove funzionalità	6
1.3. Processo di sviluppo	7
1.4. Strumenti di supporto	8
2. ANALISI E SPECIFICA DEI REQUISITI	11
2.1. Glossario dei termini	11
2.2. Requisiti funzionali	12
2.2.1. Storie utente	13
2.2.2. Diagramma dei casi d'uso	15
2.2.3. Scenari dei casi d'uso	17
2.2.3.1. CreateTeam	18
2.2.3.2. DeleteTeam	18
2.2.3.3. ViewTeamList	18
2.2.3.4. SelectAndViewTeam	19
2.2.3.5. AddStudent	19
2.2.3.6. DeleteStudent	20
2.2.3.7. EmailNotification	21
2.2.3.8. AddExerciseToTeam	21
2.2.3.9. ViewUserExercises	22
1.2. Requisiti non funzionali	22
2. PROGETTAZIONE	24
2.2. Dominio dei dati	24
2.3. Architettura a microservizi	26
2.4. Deployment Diagram	27
2.5. Gateway Pattern	29
2.6. Diagrammi di attività	30
2.6.1. CreateTeam	31
2.6.2. DeleteTeam	32
2.6.3. AddStudent	33
2.6.4. DeleteStudent	34

2.7.	Diagrammi di sequenza	35
2.7.1.	ViewStudentTeams.....	36
2.7.2.	CreateTeam.....	37
2.7.3.	DeleteTeam.....	38
2.7.4.	AddStudent	39
2.7.5.	DeleteStudent	40
3.	IMPLEMENTAZIONE	41
3.1.	Servizio T23	41
3.1.1.	Team	43
3.1.2.	TeamRepository.....	44
3.1.3.	TeamRestController.....	45
3.1.3.1.	Gestione dei Team	46
3.1.3.2.	Gestione degli Studenti nei Team.....	47
3.1.3.3.	Gestione degli Esercizi assegnati ai Team	47
3.1.3.4.	Funzioni Ausiliarie	48
3.2.	Servizio T5	50
3.2.1.	Gli esercizi.....	51
3.2.2.	Gli obiettivi degli esercizi.....	52
3.3.	Front-end (T1 e T5).....	55
3.3.1.	Teams.....	56
3.3.2.	Team's Dashboard	57
3.3.3.	Student's teams and missions	58
4.	SVILUPPI FUTURI	59

1. INTRODUZIONE

Il presente documento descrive il lavoro svolto nell'ambito del corso di Software Architecture Design, con particolare attenzione all'analisi e all'implementazione del Task R5, dedicato alla gestione dei team di utenti e degli esercizi assegnati da un amministratore.

Inserito nel quadro del progetto ERASMUS ENACTEST (European iNnovative AllianCe for TESTing educaTion), questo lavoro si propone di valorizzare l'importanza del software testing attraverso una strategia innovativa di gamification.

ENACTEST mira a risolvere il divario di conoscenze nel testing, spesso sottovalutato nel contesto accademico e aziendale. Il gioco educativo "Man vs Automated Testing Tools challenges" è stato ideato per coinvolgere gli studenti nella progettazione di casi di test, sfidandoli a competere contro robot come Randoop ed EvoSuite, capaci di generare automaticamente test JUnit. Questa iniziativa non solo promuove la competenza tecnica ma stimola anche un approccio pratico e competitivo tra gli studenti, affrontando le sfide del testing di software con creatività e innovazione.

1.1. Punto di partenza

Il progetto si basa su un'architettura a microservizi, una scelta motivata dalla necessità di garantire flessibilità, scalabilità e facilità di manutenzione. Questo approccio consente ai vari team di sviluppo di lavorare in modo autonomo su servizi specifici, riducendo al minimo le dipendenze reciproche. Ogni servizio è progettato per svolgere una funzione ben definita, rendendo possibile l'introduzione di nuove funzionalità e tecnologie senza impatti significativi sul resto del sistema.

Il sistema si basa sul Gateway Pattern, che centralizza la gestione delle richieste in un unico punto d'accesso, semplificando così la comunicazione tra i client e i microservizi sottostanti. In questo modello, il gateway è suddiviso in due componenti principali. Il primo è l'UI Gateway, il punto d'ingresso esclusivo per tutte le richieste, capace di distinguere tra richieste di pagine web e richieste di API, instradandole rispettivamente verso il frontend o verso il secondo componente, l'API Gateway. Quest'ultimo si occupa di inoltrare le richieste ai servizi appropriati.

In relazione al Task R5, l'attenzione si focalizza sull'analisi di tre servizi specifici dell'architettura, in quanto si ritiene che saranno quelli maggiormente impattati dalle modifiche da apportare.

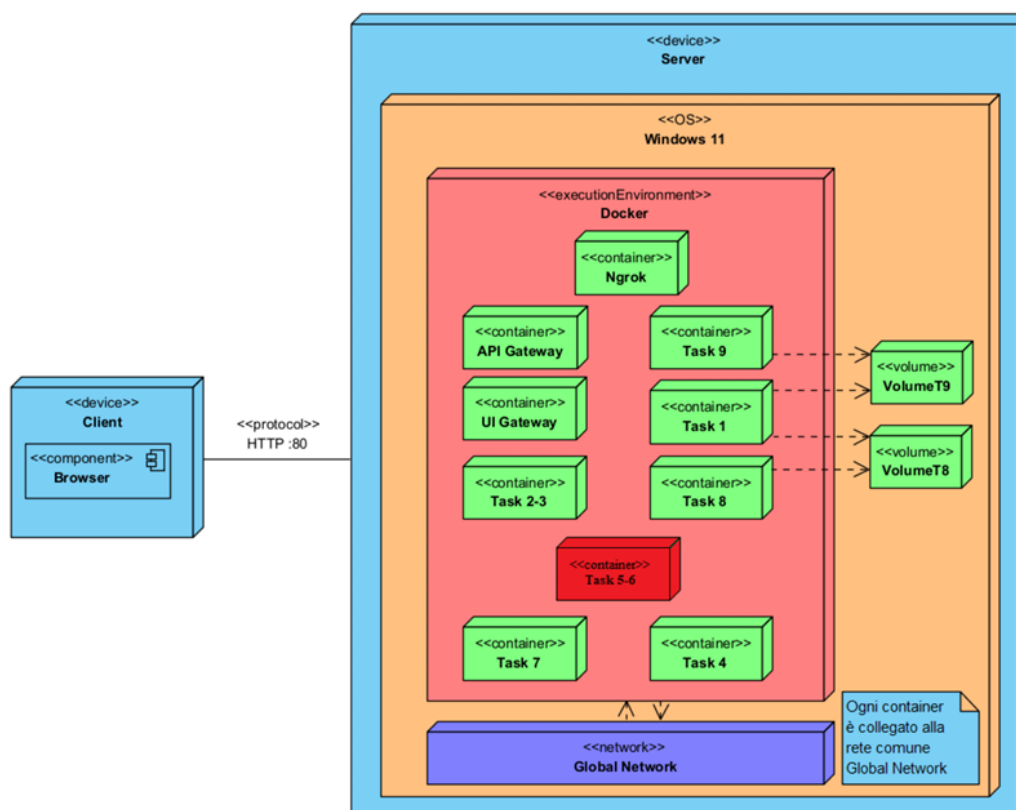
Il primo servizio, T1, si occupa della gestione delle classi Java da parte degli amministratori. Gli amministratori hanno la possibilità registrarsi e autenticarsi tramite nome, cognome, username e password. Una volta autenticati, gli amministratori possono utilizzare un apposito cruscotto per caricare, modificare, ordinare, filtrare, ricercare, scaricare o eliminare classi di programmazione. Inoltre, il servizio permette loro di visualizzare una classifica rudimentale dei giocatori. Dal lato degli studenti, il servizio consente di accedere, previa autenticazione, all'elenco delle classi caricate dagli amministratori.

Il secondo servizio, T23, è responsabile della registrazione e dell'autenticazione dei giocatori. Gli studenti possono registrarsi fornendo informazioni personali come nome, cognome, email e corso di studi. In caso di smarrimento della password, è prevista la possibilità di reimpostarla. Una volta autenticati, gli utenti hanno accesso a un'area riservata dove possono selezionare i parametri di gioco o effettuare il logout.

Infine, il terzo servizio, T5, gestisce l'avvio della partita. I giocatori autenticati possono accedere alla loro area personale, dove possono visualizzare le classi Java e i robot disponibili. In questa fase, gli studenti configurano i parametri necessari e avviano la partita, accedendo infine all'arena di gioco.

Un altro servizio che potrebbe essere impattato è il T4. Questo servizio gestisce diverse operazioni relative alle partite di gioco, come la creazione, l'aggiornamento e l'eliminazione delle partite. Si occupa anche della generazione e dell'aggiornamento dei round, nonché della gestione dei turni per i partecipanti. Inoltre, consente di recuperare i punteggi dei robot durante ogni round, specifici per ciascuna classe di gioco, migliorando così il flusso e la sincronizzazione delle informazioni.

La versione di riferimento del progetto è la A13, su cui il lavoro descritto si basa. La repository attuale è disponibile al seguente link: [Testing-Game-SAD-2023/A13](#).



1.2. Nuove funzionalità

La traccia iniziale è stata interpretata come la richiesta di dare agli amministratori del sistema pieno controllo sulla gestione dei team e degli esercizi. Da questa considerazione è emersa la necessità di sviluppare una serie di funzionalità per gestire in modo semplice ed efficace i team di utenti e tutte le informazioni ad essi associate.

Task da svolgere –R5 (Features Amministratore- Classi di studenti)

Task	Descrizione	Team
Task R5	<p>L'amministratore deve avere la possibilità di gestire /creare una classe di studenti</p> <ul style="list-style-type: none"> - Di assegnare sfide/compiti agli studenti della classe (a tempo) - verificarne/ analizzare i risultati della classe attraverso una dashboard che mostri ad esempio: <ul style="list-style-type: none"> ▪ Tempo medio necessario per completare una sfida ▪ Percentuale di copertura media raggiunta dai giocatori ▪ Etc.. 	

In primo luogo, è stato ritenuto necessario che l'amministratore potesse creare nuovi team, specificando un nome e una descrizione. Inoltre, l'amministratore deve poter modificare i dettagli di un team in qualsiasi momento e, se necessario, eliminarlo. Deve anche avere la possibilità di aggiungere o rimuovere utenti da un team.

Allo stesso tempo, si è ritenuto che l'amministratore dovesse poter assegnare esercizi ai team di utenti. Durante la creazione, l'amministratore può definire gli obiettivi che costituiscono l'esercizio. Questi obiettivi sono stati progettati per essere abbastanza generici, così da poter essere adattati facilmente a diversi contesti. Ad esempio, un obiettivo potrebbe essere del tipo “testa questa classe con l’x% di copertura” oppure “vinci una sfida N volte”.

A ogni esercizio è associata una data di scadenza, la quale rappresenta il termine entro il quale i membri del team devono completare gli obiettivi stabiliti. Per dare all'amministratore un controllo completo sulla gestione degli esercizi, è stato previsto che possa modificare la descrizione o la data di scadenza di un esercizio anche dopo la creazione e, se necessario, eliminarlo anche prima della scadenza.

Per monitorare i progressi, è stata implementata una dashboard che permette agli amministratori di visualizzare lo stato di avanzamento degli utenti del team rispetto agli esercizi assegnati. La dashboard offre una panoramica chiara dei progressi, degli obiettivi completati e di quelli ancora da affrontare, permettendo all'amministratore di intervenire tempestivamente in caso di ritardi o problemi.

Gli utenti devono poter accedere alle informazioni sui team a cui appartengono e sugli esercizi assegnati, con i relativi obiettivi. In questo modo, possono avere una visione chiara delle proprie responsabilità e dei tempi da rispettare.

Queste considerazioni iniziali hanno dato vita alla formulazione di specifici requisiti che sono dettagliati nel capitolo successivo.

1.3. Processo di sviluppo

Per il processo di sviluppo, è stata adottata una filosofia agile, mantenendola il più possibile flessibile e adattabile alle esigenze del team. I modelli agili, come Scrum o Extreme Programming (XP), sono progettati per favorire la collaborazione continua, l'adattamento ai cambiamenti (specialmente nei requisiti) e un approccio iterativo e incrementale. Questa metodologia, anche

se in sua versione semplificata, ha permesso al team di reagire rapidamente alle problematiche e di apportare modifiche tempestive al progetto. Inoltre, l'approccio agile prevede cicli di sviluppo brevi, che permettono di ottenere feedback continui e identificare rapidamente gli errori e correggerli, migliorando così l'efficienza del processo complessivo.

Il team ha adottato un approccio incrementale che alternava fasi di lavoro individuale e momenti di collaborazione per raggruppare il lavoro attorno a macro-funzionalità. Definite le interfacce, ogni membro ha potuto concentrarsi autonomamente sui propri compiti, senza interferenze costanti. Tuttavia, per mantenere il lavoro allineato e affrontare puntualmente decisioni strutturali, sono stati organizzati incontri regolari, principalmente tramite videochiamate. Questi incontri non seguivano una cadenza fissa, né per frequenza né per durata. La programmazione degli incontri dipendeva dalle esigenze riscontrate.

Durante le videochiamate, ogni membro del team spiegava cosa aveva fatto, le difficoltà incontrate e come le aveva superate. Questo processo ha permesso una continua condivisione delle informazioni e una collaborazione attiva da parte di tutti i partecipanti. Le discussioni si concentravano anche sulla pianificazione delle prossime fasi del progetto, dividendosi le attività da svolgere e stabilendone le priorità.

Il flusso di lavoro è stato molto dinamico e privo di formalismi rigidi, offrendo così libertà nello sviluppo del software, ma mantenendo comunque una forte coesione tra i membri del team. La flessibilità del modello adottato è stata efficiente nell'affrontare le problematiche emerse lungo il percorso, considerando le dimensioni del gruppo e la complessità del lavoro. Tuttavia, con l'aumentare di queste due variabili, sarebbe stato necessario utilizzare strumenti più strutturati per organizzare il lavoro in modo più efficace

1.4. Strumenti di supporto

Per lo sviluppo di questo progetto, è stato scelto un insieme di tool: ogni strumento è stato selezionato per rispondere alle esigenze specifiche del progetto, migliorando la produttività e semplificando le operazioni quotidiane. Di seguito sono elencati gli strumenti principali utilizzati, con una breve descrizione di come ciascuno di essi ha contribuito al successo del progetto.

- **Microsoft Teams:** È stato utilizzato come piattaforma principale per la comunicazione,

sia per le videochiamate che per la condivisione di documenti. Teams ha rappresentato il repository di tutte le risorse del progetto, assicurando una gestione centralizzata e un facile accesso.

- **GitHub:** È stato impiegato come repository per il codice. GitHub ha permesso di mantenere traccia delle modifiche apportate nel codice sorgente, garantendo un flusso di lavoro organizzato e trasparente.
- **Editor di codice:** Ogni sviluppatore ha utilizzato l'editor di codice preferito, con una maggiore diffusione di **Visual Studio Code** e **Visual Studio**. Entrambi offrono funzionalità avanzate, come il debugging, il completamento automatico del codice e l'integrazione con il controllo della versione, rendendo così la programmazione più rapida ed efficiente.
- **Visual Paradigm:** È stato utilizzato per la creazione di diagrammi UML, fondamentali per rappresentare visivamente l'architettura del sistema e i flussi di lavoro. Ha permesso di creare immagini chiare e facilmente comprensibili, utili per progettare e documentare.
- **Draw.io:** È un altro strumento utilizzato per creare diagrammi e schemi, che semplifica la visualizzazione dei processi e delle strutture.
- **Microsoft Word:** È stato impiegato per redigere e organizzare la documentazione testuale del progetto.
- **Docker Desktop:** Ha permesso di eseguire ciascun servizio in ambienti isolati, garantendo che ogni componente funzionasse indipendentemente ma in modo coordinato. Docker ha semplificato il deployment locale, consentendo di eseguire e testare facilmente l'applicazione su diverse macchine, senza problemi di compatibilità tra i vari servizi o le loro dipendenze.
- **Spring Boot:** Si è scelto come framework principale per lo sviluppo del backend. Grazie alla sua configurazione automatica, ha semplificato la creazione di servizi web e API, riducendo il tempo di sviluppo e migliorando la manutenibilità del codice. La sua architettura modulare e le numerose funzionalità integrate hanno permesso di concentrarsi sullo sviluppo delle logiche applicative senza preoccuparsi troppo della configurazione.
- **Postman:** È stato utilizzato per testare le API REST del sistema, consentendo di eseguire richieste HTTP, provare gli endpoint e visualizzare facilmente le risposte del server. Questo strumento è stato essenziale per verificare il corretto funzionamento delle API.

- **JavaMail e Gmail SMTP Server:** Per gestire l'invio delle email, è stata utilizzata la libreria JavaMail insieme al Gmail SMTP Server. Questa combinazione ha reso possibile inviare notifiche e comunicazioni agli utenti, in particolare riguardo ai team a cui appartengono e agli esercizi loro assegnati.
- **MongoDB e MySQL:** Oltre al classico MySQL, è stato utilizzato anche MongoDB, un database NoSQL che ha offerto maggiore flessibilità nella gestione dei dati non strutturati, in particolare per quanto riguarda gli esercizi e i relativi obiettivi. Quindi MongoDB si è rivelato utile per archiviare dati complessi e variabili, mentre MySQL è stato usato per gestire i dati relazionali e strutturati, come quelli relativi ai team di utenti.
- **Jackson:** È una libreria per la serializzazione e deserializzazione degli oggetti JSON. In pratica, questo strumento consente di convertire i dati tra il formato JSON e gli oggetti Java.
- **Lombok:** È una libreria che riduce il boilerplate nel codice Java. Grazie alle sue annotazioni, è stato possibile generare automaticamente metodi come i getter e setter, il costruttore e il metodo toString(), riducendo così la quantità di codice da scrivere e migliorando la leggibilità e la manutenibilità del codice.
- **Thymeleaf:** È stato utilizzato per gestire le pagine web dell'applicazione. Come motore di template, ha permesso di creare interfacce utente dinamiche, facilmente collegate al backend. La sintassi semplice e la perfetta compatibilità con Spring Boot lo hanno reso uno strumento ideale.

2. ANALISI E SPECIFICA DEI REQUISITI

L'analisi dei requisiti è una fase fondamentale nel ciclo di vita di un progetto software, perché rappresenta la base per sviluppare un sistema che soddisfi le reali esigenze degli utenti e degli stakeholder. In questo capitolo si analizzerà il requisito assegnato, spiegando come è stato interpretato e suddiviso in base alle sue caratteristiche principali.

Lo scopo di questa fase è quello di trasformare le richieste iniziali in specifiche chiare e strutturate, utili a guidare le fasi successive di progettazione e sviluppo. Tale processo non si limita alla comprensione delle esigenze esplicite indicate dai clienti, ma include anche l'individuazione di eventuali requisiti impliciti o latenti che possono emergere dall'utilizzo del sistema.

Il requisito principale assegnato è riportato nella tabella seguente:

ID	Descrizione
R5	Si desidera che l'amministratore abbia la possibilità di gestire e creare una classe di studenti. All'interno di questa funzione, dovrebbe poter assegnare sfide o compiti agli studenti, con l'opzione di impostare un limite di tempo per il completamento. Inoltre, è fondamentale che l'amministratore possa verificare e analizzare i risultati ottenuti dalla classe tramite una dashboard intuitiva, che fornisca metriche utili come il tempo medio necessario per completare una sfida e la percentuale media di copertura raggiunta dai giocatori.

Sulla base di questa descrizione, sono stati individuati requisiti funzionali e non funzionali che orienteranno le modifiche da apportare al sistema di gioco. Nei paragrafi successivi verranno analizzati nel dettaglio i requisiti identificati, con un approfondimento sulle implicazioni architetturali.

2.1. Glossario dei termini

Il glossario dei termini fornisce le definizioni dei concetti chiave del progetto, assicurando che tutti i membri del team e i clienti abbiano una comprensione comune. Aiuta a prevenire ambiguità

e fraintendimenti, migliorando la comunicazione e la collaborazione tra persone con competenze diverse. La tabella sottostante elenca i principali termini rilevanti con le relative definizioni.

Termine	Sinonimo	Definizione
Task	Requisito principale	Il lavoro assegnato al team di sviluppo per la realizzazione di una specifica funzionalità.
Servizio	Microservizio, Modulo, Tx	Un componente dell'architettura del sistema software che fornisce una funzionalità specifica all'interno dell'applicazione.
User	Utente, giocatore, studente, partecipante	Un giocatore registrato sulla piattaforma.
Team	Gruppo	Un insieme di utenti, gestibile da un qualsiasi amministratore.
Admin	Amministratore, responsabile, insegnante	Un responsabile che gestisce i team e assegna loro esercizi.
Exercise	Esercizio, Missione	Una serie di compiti assegnati a un team da un amministratore.
Goal	Obiettivo	Un risultato che si deve raggiungere come parte di un esercizio.
Assignment	Assegnazione	Assegnazione di un obiettivo a un utente, in quanto membro del team a cui è stato assegnato l'esercizio.

Alcune scelte terminologiche sono state fatte per garantire chiarezza e coerenza:

- **Team:** È stato scelto questo termine perché è il più generico e versatile, facilmente comprensibile e utilizzabile in diversi contesti. Inizialmente si era pensato a “corso”, ma questo termine non richiamava immediatamente l'idea di un insieme di utenti. Il termine “classe” è stato scartato per evitare confusione, sia con le classi da testare all'interno del gioco, sia con le classi del software stesso. Inoltre, questi due ultimi termini richiamano fortemente l'ambiente scolastico, mentre si cercava un termine con maggiore applicabilità.
- **Esercizi:** Si è scelto di utilizzare questo termine invece di “missioni” o “task” per evitare sovrapposizioni con requisiti assegnati ad altri team di sviluppo o con componenti dell'architettura a microservizi.

2.2. Requisiti funzionali

I requisiti funzionali definiscono le operazioni che il sistema deve eseguire per soddisfare le esigenze degli utenti. Questi requisiti sono essenziali per guidare lo sviluppo del sistema, assicurandosi che tutte le funzionalità richieste siano correttamente implementate.

In questa sezione, vengono descritti i requisiti funzionali del progetto, evidenziando le operazioni e le interazioni con gli utenti. Di seguito è riportata una tabella che riassume i requisiti principali, con le relative descrizioni.

ID	Descrizione
RF01	L'amministratore deve poter creare un team, specificando il nome e la descrizione.
RF02	L'amministratore deve poter modificare le informazioni relative a un team (nome e descrizione).
RF03	L'amministratore deve poter eliminare un team.
RF04	L'amministratore deve poter aggiungere un utente a un team.
RF05	L'amministratore deve poter rimuovere uno studente da un team.
RF06	L'amministratore deve poter visualizzare un elenco di tutti i team.
RF07	L'amministratore deve poter visualizzare la lista degli utenti che fanno parte di un team.
RF08	Il sistema deve inviare una notifica tramite e-mail a un utente quando viene aggiunto a un team.
RF09	Il sistema deve inviare una notifica tramite e-mail a un utente quando viene rimosso da un team.
RF10	Il sistema deve inviare una notifica tramite e-mail a uno utente quando un team a cui appartiene viene eliminato.
RF11	L'utente deve poter visualizzare la lista dei team a cui appartiene.
RF12	L'amministratore deve poter assegnare un esercizio a un team, specificando la data di inizio e fine, una descrizione e gli obiettivi.
RF13	L'amministratore deve poter modificare la data di scadenza e la descrizione di un esercizio di un team.
RF14	L'amministratore deve poter eliminare un esercizio assegnato a un team.
RF15	L'amministratore deve poter visualizzare un elenco degli esercizi di un team.
RF16	L'amministratore deve poter visualizzare lo stato di completamento di un esercizio di un team.
RF17	L'utente deve poter visualizzare gli esercizi assegnati ai team di cui fa parte.

2.2.1. Storie utente

L'analisi e la specifica dei requisiti possono essere condotte attraverso diversi approcci, che variano in base al contesto del progetto, al modello di sviluppo adottato e al livello di dettaglio richiesto. Un metodo ampiamente utilizzato nei contesti agili è quello delle user stories (storie utente), che permettono di descrivere i requisiti in modo chiaro, orientato all'utente e accessibile a tutti i membri del team.

Le storie utente sono brevi descrizioni che illustrano ciò che un utente desidera ottenere dal sistema, raccontate dal suo stesso punto di vista. Questo formato permette di focalizzarsi sui bisogni reali degli utenti, favorendo un approccio collaborativo alla progettazione, che si basa su miglioramenti e adattamenti continui in risposta al feedback ricevuto.

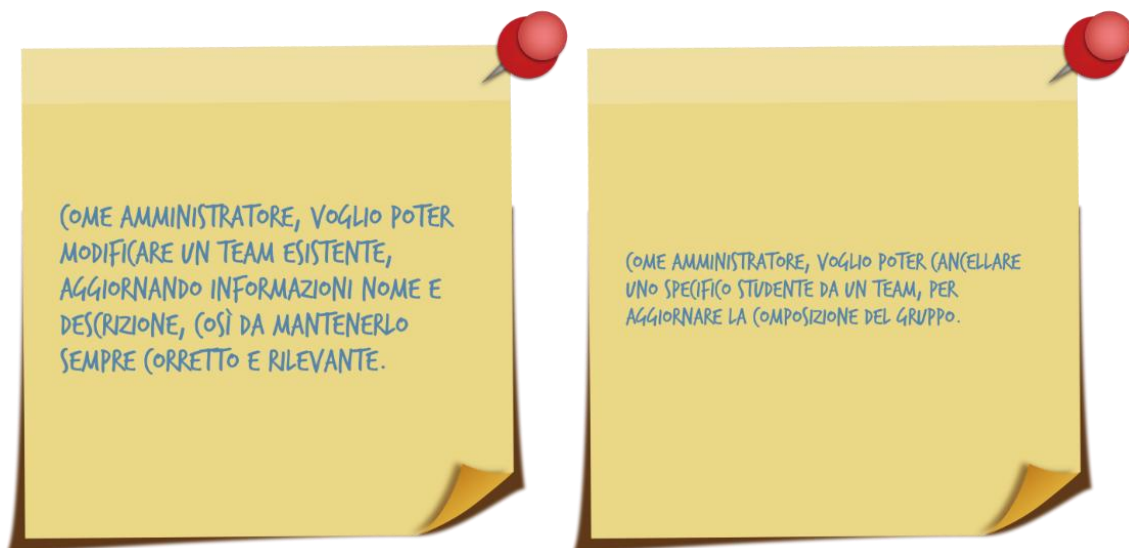
Ogni storia segue generalmente la struttura:

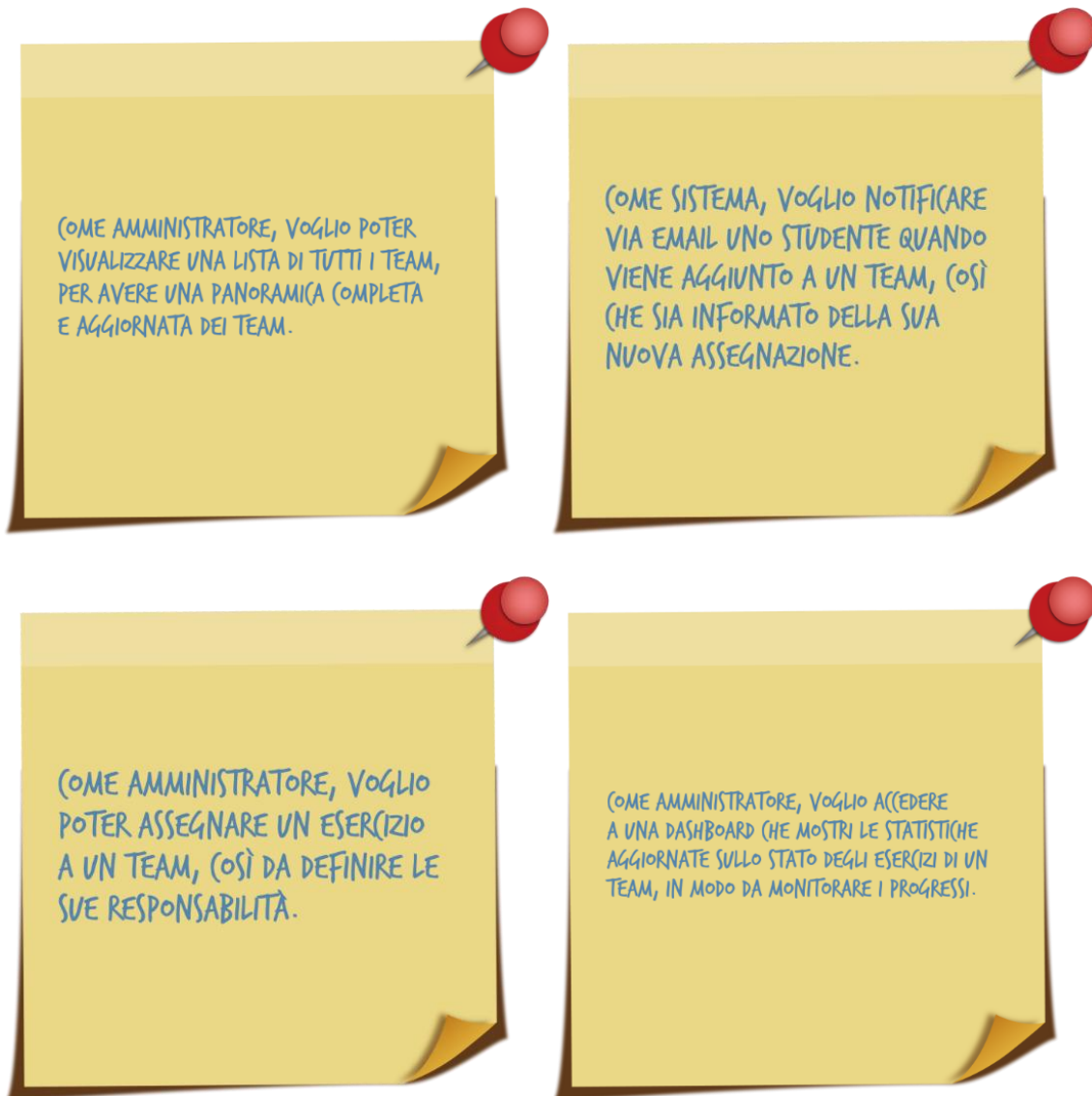
"Come [ruolo], voglio [obiettivo] così che [beneficio]."

Le storie utente sono spesso rappresentate secondo il modello delle 3 C:

- Card: la storia utente viene scritta su una schedina fisica o digitale, che contiene la descrizione della funzionalità.
- Conversation: la storia utente stimola una conversazione tra sviluppatori e stakeholder per chiarire e rifinire i dettagli.
- Confirmation: vengono definiti i criteri di accettazione per confermare che la funzionalità è stata implementata correttamente.

In un primo momento, si è deciso di utilizzare le storie utente per definire i requisiti. Tuttavia, per garantire maggiore completezza e struttura, si è poi optato per un approccio più dettagliato tramite i casi d'uso. Questo aspetto verrà approfondito nei paragrafi successivi. Per completezza, vengono di seguito presentati alcuni esempi di storie utente relative al task specifico.

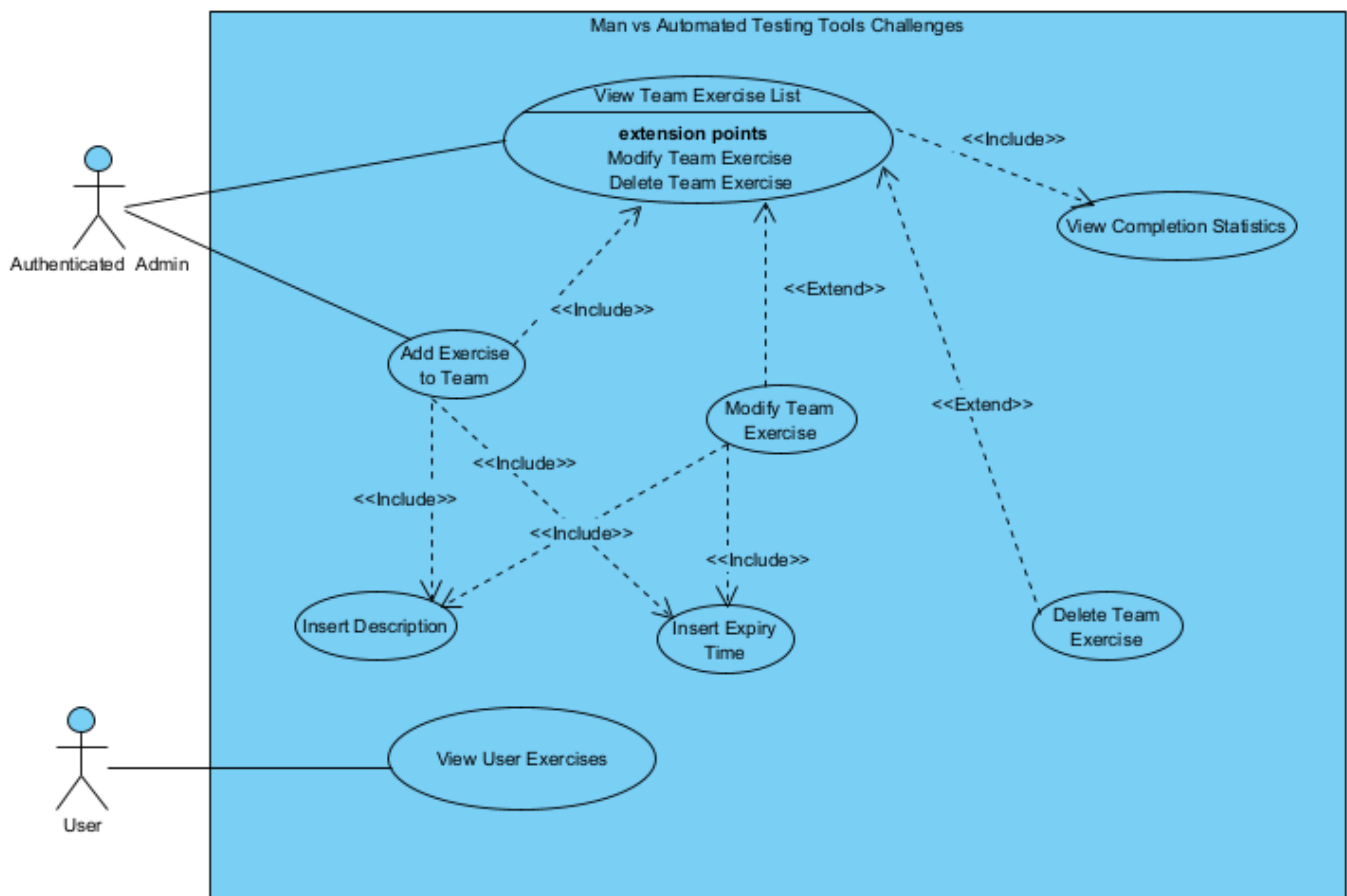
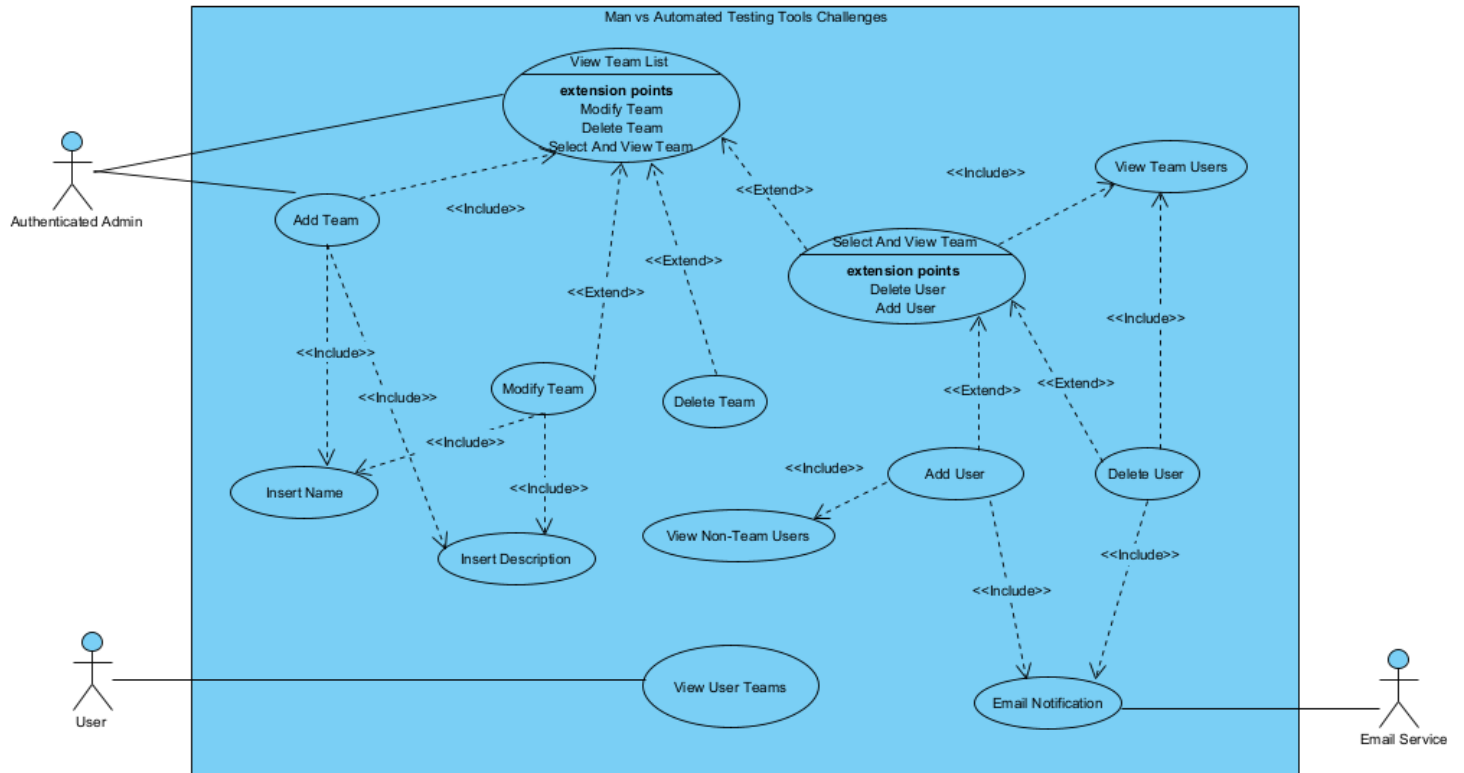




2.2.2. Diagramma dei casi d'uso

Il diagramma dei casi d'uso è uno strumento fondamentale per rappresentare le interazioni tra gli attori esterni e il sistema, mettendo in evidenza le funzionalità principali richieste. Grazie alla sua forma intuitiva, il diagramma consente di analizzare i requisiti funzionali in modo strutturato, favorendo una comprensione condivisa tra sviluppatori e utenti.

Di seguito vengono presentati due diagrammi dei casi d'uso in UML (Unified Modeling Language). Il primo riguarda le operazioni relative ai team di utenti, mentre il secondo si concentra sulla gestione degli esercizi. Questa suddivisione è stata adottata per migliorare la leggibilità.



2.2.3. Scenari dei casi d'uso

Gli scenari dei casi d'uso sono una rappresentazione dettagliata delle interazioni tra gli utenti e il sistema, necessarie per raggiungere specifici obiettivi. Descrivono in modo chiaro le operazioni che il sistema deve eseguire, aiutando il team di sviluppo a comprendere le esigenze degli utenti e progettare soluzioni adeguate. In altre parole, gli scenari definiscono in modo chiaro i flussi di lavoro del sistema, fornendo una guida per lo sviluppo e i test.

Se il diagramma dei casi d'uso offre una rappresentazione intuitiva delle interazioni principali, gli scenari forniscono invece una descrizione più approfondita di ciascun caso d'uso. Il diagramma, infatti, non è sempre necessario, poiché gli scenari dei casi d'uso stessi forniscono una spiegazione completa, dettagliando ogni fase del processo.

Per definire uno scenario di un caso d'uso, si adotta un template che assicura coerenza e organizzazione. Di seguito viene descritta ogni sezione del template.

- **Attori primari e secondari:** identificano le entità esterne che interagiscono con il sistema.
- **Descrizione:** fornisce una definizione sintetica del caso d'uso.
- **Pre-condizioni:** indica lo stato che deve essere soddisfatto prima dell'inizio del caso d'uso.
- **Sequenza di eventi principali:** descrive i passaggi che avvengono quando il caso d'uso si svolge correttamente.
- **Post-condizioni:** indica lo stato finale del sistema dopo il completamento del caso d'uso principale.
- **Casi d'uso correlati:** elenca altri casi d'uso che possono essere collegati a quello in esame.
- **Sequenza di eventi alternativi:** descrive gli scenari alternativi che possono verificarsi durante l'esecuzione del caso d'uso.

Non sono stati descritti tutti i possibili scenari dei casi d'uso in dettaglio, poiché alcuni risultano particolarmente semplici e altri sono molto simili a quelli che verranno descritti. Inoltre, gli scenari presentati in seguito riflettono i requisiti funzionali come interpretati all'inizio del processo di sviluppo e potrebbero subire modifiche durante le fasi successive, qualora emergano problematiche o opportunità di miglioramento. Infine, alcuni dettagli, come i controlli sugli input o la gestione degli errori, sono stati omessi in quanto sono considerati comuni a qualsiasi scenario.

2.2.3.1. CreateTeam

Caso d'uso: CreateTeam	
Attore primario	Authenticated Admin
Attore secondario	-
Descrizione	Un amministratore crea un nuovo team.
Pre-condizioni	L'amministratore è loggato nel sistema.
Sequenza di eventi principale	<ol style="list-style-type: none">1. L'amministratore accede alla sezione dedicata dei team.2. L'amministratore inserisce il nome e la descrizione del nuovo team, quindi conferma la creazione.3. Il sistema crea il nuovo team.
Post-condizioni	Un amministratore può visualizzare, modificare o eliminare il team.
Casi d'uso correlati	<i>ViewTeamList, InsertTeamName, InsertTeamDescription</i>
Sequenza di eventi alternativi	-

2.2.3.2. DeleteTeam

Caso d'uso: DeleteTeam	
Attore primario	Authenticated Admin
Attore secondario	-
Descrizione	Un amministratore elimina un team.
Pre-condizioni	L'amministratore è loggato nel sistema e il team esiste.
Sequenza di eventi principale	<ol style="list-style-type: none">1. L'amministratore accede alla sezione dedicata dei team.2. L'amministratore chiede di vedere la lista dei team.3. Il sistema recupera e mostra l'elenco completo dei team.4. L'amministratore seleziona il team da eliminare e conferma la scelta.5. Il sistema elimina il team e aggiorna l'elenco.
Post-condizioni	Il team non esiste più nel sistema.
Casi d'uso correlati	<i>ViewTeamList</i>
Sequenza di eventi alternativi	-

Inoltre, alla cancellazione di un team, il sistema invia automaticamente una notifica via e-mail a tutti gli utenti che ne fanno parte, informandoli dell'eliminazione del team e delle eventuali modifiche che potrebbero riguardare i loro esercizi.

2.2.3.3. ViewTeamList

Caso d'uso: ViewTeamList	
Attore primario	Authenticated Admin
Attore secondario	-
Descrizione	Un amministratore visualizza una lista di tutti i team.

Pre-condizioni	L'amministratore è loggato nel sistema e il team esiste.
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'amministratore accede alla sezione dedicata dei team. 2. L'amministratore chiede di vedere la lista dei team. 3. Il sistema recupera e mostra l'elenco completo dei team.
Post-condizioni	L'amministratore può eseguire operazioni come, ad esempio, la modifica o l'eliminazione del team.
Casi d'uso correlati	<i>ModifyTeam, DeleteTeam, SelectAndViewTeam</i>
Sequenza di eventi alternativi	-

2.2.3.4. SelectAndViewTeam

Caso d'uso: SelectAndViewTeam	
Attore primario	Authenticated Admin
Attore secondario	-
Descrizione	Un amministratore visualizza le informazioni di un team.
Pre-condizioni	L'amministratore è loggato nel sistema e il team esiste.
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'amministratore accede alla sezione dedicata dei team. 2. L'amministratore chiede di vedere la lista dei team. 3. Il sistema recupera e mostra l'elenco completo dei team. 4. L'amministratore seleziona il team di cui visualizzare i dettagli. 5. Il sistema mostra la sezione dedicata al team, permettendo di visualizzare il nome, la descrizione e, eventualmente, gli utenti che ne fanno parte.
Post-condizioni	Tra le altre cose, l'amministratore può aggiungere o rimuovere un utente dal team.
Casi d'uso correlati	<i>ViewTeamList, DeleteUser, AddUser</i>
Sequenza di eventi alternativi	-

2.2.3.5. AddStudent

Caso d'uso: AddStudent	
Attore primario	Authenticated Admin
Attore secondario	User
Descrizione	Un amministratore inserisce un utente in un team.
Pre-condizioni	L'amministratore è loggato nel sistema, il team è stato creato e l'utente è registrato.
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'amministratore accede alla sezione dedicata dei team. 2. L'amministratore chiede di vedere la lista dei

	<p>team.</p> <ol style="list-style-type: none"> Il sistema recupera e mostra l'elenco completo dei team. L'amministratore seleziona il team di cui visualizzare i dettagli. Il sistema mostra la sezione dedicata al team, permettendo di visualizzare l'elenco degli utenti che attualmente fanno parte del team. L'amministratore richiede di aggiungere un nuovo utente al team. Il sistema mostra una lista di utenti registrati che non sono ancora assegnati al team. L'amministratore seleziona un utente da inserire nel team. Il sistema aggiunge l'utente selezionato e aggiorna l'elenco degli utenti del team.
Post-condizioni	Gli utenti del team vengono aggiornati nel sistema. L'utente può visualizzare il team nella sezione dedicata.
Casi d'uso correlati	<i>SelectAndViewTeam, EmailNotification, ViewNonTeamUsers, ViewUserTeams</i>
Sequenza di eventi alternativi	-

2.2.3.6. DeleteStudent

Caso d'uso: DeleteStudent	
Attore primario	Authenticated Admin
Attore secondario	User
Descrizione	Un amministratore inserisce un utente in un team.
Pre-condizioni	L'amministratore è loggato nel sistema, il team esiste, l'utente è registrato e fa parte del team.
Sequenza di eventi principale	<ol style="list-style-type: none"> L'amministratore accede alla sezione dedicata dei team. L'amministratore chiede di vedere la lista dei team. Il sistema recupera e mostra l'elenco completo dei team. L'amministratore seleziona il team di cui visualizzare i dettagli. Il sistema mostra la sezione dedicata al team, permettendo di visualizzare l'elenco degli utenti che attualmente fanno parte del team. L'amministratore seleziona un utente da rimuovere dal team e conferma la sua scelta.

	7. Il sistema elimina l'utente dal team e aggiorna la lista visualizzata.
Post-condizioni	Gli utenti del team vengono aggiornati nel sistema. L'utente non può visualizzare il team nella sezione dedicata ai team a cui appartiene.
Casi d'uso correlati	<i>SelectAndViewTeam, EmailNotification, ViewNonTeamUsers</i>
Sequenza di eventi alternativi	-

2.2.3.7. EmailNotification

Caso d'uso: EmailNotification	
Attore primario	Email Service
Attore secondario	User
Descrizione	Il sistema invia una notifica via e-mail al momento dell'aggiunta o della rimozione di un utente da un team.
Pre-condizioni	Il team è stato creato e l'utente è registrato.
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il sistema rileva l'aggiunta di un utente a un team. 2. Il sistema recupera l'indirizzo e-mail dell'utente e il nome del team. 3. Il sistema prepara il messaggio. 4. Il sistema invia l'e-mail all'utente tramite il servizio di posta elettronica.
Post-condizioni	L'utente visualizza la notifica nella propria casella di posta elettronica.
Casi d'uso correlati	<i>AddUser, DeleteUser</i>
Sequenza di eventi alternativi	<ol style="list-style-type: none"> 1. Il sistema rileva la rimozione di un utente da un team. 2. Il sistema recupera l'indirizzo e-mail dell'utente e il nome del team. 3. Il sistema prepara il messaggio. 4. Il sistema invia l'e-mail all'utente tramite il servizio di posta elettronica.

2.2.3.8. AddExerciseToTeam

Caso d'uso: AddExerciseToTeam	
Attore primario	Authenticated Admin
Attore secondario	-
Descrizione	L'amministratore assegna un esercizio a un team.
Pre-condizioni	L'amministratore si trova nella sezione del sistema dedicata allo specifico team.
Sequenza di eventi principale	1. L'amministratore indica di voler assegnare un

	nuovo esercizio al team. 2. L'amministratore inserisce uno o più obiettivi, la data di inizio e di fine e la descrizione. 3. Il sistema crea e aggiunge l'esercizio al team di studenti.
Post-condizioni	Un amministratore può visualizzare, modificare o eliminare l'esercizio dal team. L'utente può visualizzare l'esercizio nella sezione dedicata allo specifico team.
Casi d'uso correlati	<i>InsertDescription, InsertExpiryTime, ViewTeamExerciseList</i>
Sequenza di eventi alternativi	-

2.2.3.9. ViewUserExercises

Caso d'uso: ViewUserExercises	
Attore primario	User
Attore secondario	-
Descrizione	L'utente visualizza gli esercizi assegnatogli.
Pre-condizioni	L'utente deve essere loggato nel sistema.
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente accede alla sua area personale. 2. L'utente seleziona la sezione relativa ai team di cui è membro. 3. Il sistema consente all'utente di visualizzare le liste degli esercizi assegnati, organizzate per team.
Post-condizioni	-
Casi d'uso correlati	-
Sequenza di eventi alternativi	-

1.2. Requisiti non funzionali

I requisiti non funzionali sono quei requisiti che definiscono le caratteristiche di qualità del sistema e che non sono direttamente legati alle funzionalità specifiche che il sistema deve mettere a disposizione. Piuttosto, si concentrano su come il sistema dovrebbe comportarsi durante l'uso.

I requisiti non funzionali identificati includono:

- **Usabilità:** Il sistema deve essere facile da usare per l'amministratore, senza richiedere una formazione complessa. L'interfaccia deve essere chiara e intuitiva, per semplificare l'interazione con la dashboard e la gestione dei team di studenti.
- **Integrità:** Il sistema deve garantire l'integrità dei dati e dei relativi processi, assicurando che le informazioni sui team siano registrate e recuperate correttamente, senza errori. Ogni operazione sui dati, come l'aggiornamento o la visualizzazione, deve essere eseguita in

modo accurato, evitando qualsiasi perdita o corruzione delle informazioni.

- **Estensibilità:** Il sistema deve essere facilmente modificabile, in modo che future funzionalità possano essere aggiunte senza compromettere la stabilità dell'applicazione esistente. L'architettura deve consentire l'introduzione di nuovi requisiti o modifiche con il minimo impatto sulle funzioni già implementate.

In questo progetto, i requisiti non funzionali non sono stati definiti in modo rigoroso e misurabile, ma trattati in maniera più informale. Tuttavia, questi aspetti di qualità sono stati presi in considerazione durante tutte le fasi del lavoro, per garantire che il sistema rispondesse adeguatamente alle aspettative degli utenti e funzionasse in modo stabile e sostenibile.

2. PROGETTAZIONE

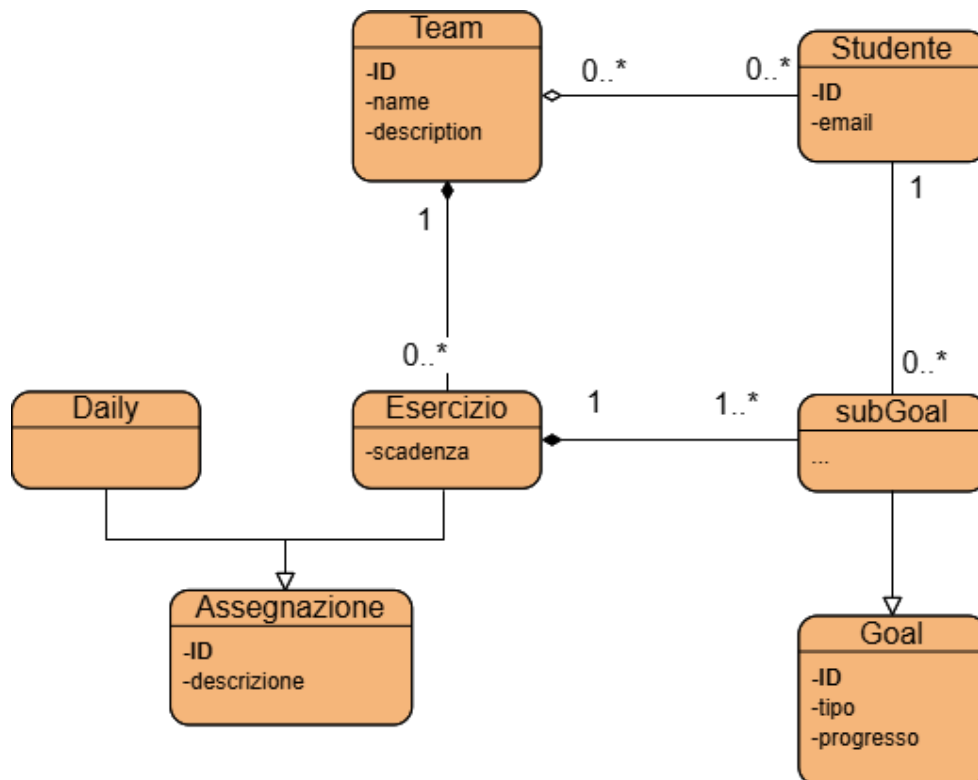
Questo capitolo descrive nel dettaglio le modifiche da apportare all'applicazione in base al task assegnato, offrendo una visione chiara e strutturata dell'architettura e dei flussi operativi.

Innanzitutto, viene fornita una modellazione del dominio della soluzione attraverso un diagramma entità-relazione, accompagnata da un'analisi delle scelte effettuate.

La soluzione proposta, successivamente, viene analizzata e particolareggiata sotto diversi aspetti, primo dei quali il Deployment Diagram, che mostra come i componenti software sono distribuiti sui nodi della rete in un ambiente di produzione, evidenziandone le interazioni.

Successivamente, vengono esaminati i diagrammi delle sequenze, che descrivono il flusso delle operazioni nei diversi casi d'uso, mostrando l'ordine e la tempistica delle comunicazioni tra le varie parti del sistema. Questi diagrammi sono essenziali per comprendere il funzionamento dell'applicazione durante l'esecuzione dei processi.

2.2. Dominio dei dati



Il Team è composto da una raccolta di studenti ed esercizi. Gli studenti sono già presenti nel sistema, mentre gli esercizi devono essere modellati. Nell'interfaccia utente, il Team verrà presentato con un nome e una descrizione.

La modellazione degli esercizi è affetta dalla genericità della richiesta. In un contesto di sviluppo reale, il key stakeholder avrebbe guidato le scelte implementative sul dominio dei dati, fornendo feedback su mock di interfacce e simulazioni di casi d'uso.

Invece, durante l'analisi, sono emerse tre possibili interpretazioni:

- Un esercizio può essere completato attraverso la sottomissione volontaria di una soluzione.
- L'esercizio potrebbe configurarsi come una modalità di gioco esclusiva, che estende quelle esistenti e include una sonda per restituire i risultati all'amministratore.
- Il sistema di esercizi potrebbe basarsi su un meccanismo di obiettivi che rileva gli eventi generati quando il giocatore compie determinate azioni, come completare una determinata modalità di gioco.

L'ultima interpretazione è apparsa moderatamente complessa, ma ideale per introdurre un'infrastruttura capace di supportare future evoluzioni, come un sistema migliorato di achievement e missioni giornaliere.

Mentre l'esercizio è un oggetto del team e descrive gli obiettivi che lo compongono, l'obiettivo effettivo presenta le seguenti proprietà:

- Tracciamento del progresso: Il goal contiene l'avanzamento del giocatore a cui fa riferimento. In particolare, per ogni giocatore e per ogni prototipo di obiettivo in un esercizio, esiste un goal corrispondente. I goal attivi di uno studente (non scaduti e non completati) vengono controllati contro gli eventi generati per aggiornare, eventualmente, il loro stato.
- Polimorfismo: Potrebbero esistere vari tipi di goal, ognuno con specifiche caratteristiche, tra cui:
 - Ambito (ad esempio obiettivo di gioco o sociale).
 - Filtro (ad esempio modalità di gioco, classe under test o punteggio).
 - Meccanica di progresso (ad esempio ripetizione di un'azione per un certo numero di volte o aggiornamento progressivo dello stato a ogni evento).

Infine, un requisito fondamentale per l'implementazione del goal vuole essere la modularità, poiché non si esclude che nuovi obiettivi possano essere aggiunti continuamente anche a sistema installato e in esecuzione. Per questo motivo, si vuole minimizzare l'impatto di ogni nuova aggiunta sul database e sul codice.

2.3. Architettura a microservizi

Il progetto si basa su un'architettura a microservizi, un modello in cui un'applicazione viene suddivisa in piccoli servizi indipendenti che comunicano tra loro tramite API. Ogni microservizio è progettato per eseguire una funzione specifica o un insieme limitato di funzioni e può essere sviluppato, distribuito e scalato autonomamente rispetto agli altri.

Tra i principali vantaggi di questa architettura vi è la modularità, che consente lo sviluppo e la manutenzione dei microservizi da parte di team di lavoro indipendenti. Un altro punto di forza è la scalabilità, poiché ogni microservizio può essere scalato individualmente in base alle esigenze di carico. Ad esempio, un servizio che riceve un numero elevato di richieste può essere scalato orizzontalmente senza coinvolgere l'intera applicazione, ottimizzando così l'uso delle risorse e riducendo i costi operativi.

Tuttavia, questa architettura presenta anche alcune sfide. La gestione di un numero elevato di microservizi può aumentare la complessità operativa, mentre la comunicazione tra di essi può introdurre latenza, rendendo necessaria un'infrastruttura di rete robusta per garantire prestazioni affidabili.

Inoltre, ogni servizio può essere progettato seguendo un modello architetturale specifico, in base alle sue esigenze funzionali e non. Ad esempio, il servizio T1 è stato implementato utilizzando l'architettura MVC (Model-View-Controller), che suddivide l'applicazione in tre componenti distinti: il Model, responsabile della gestione dei dati e della logica di business; la View, che si occupa della presentazione delle informazioni all'utente; e il Controller, che gestisce le richieste e coordina l'interazione tra Model e View. Questo approccio garantisce un codice modulare e di facile manutenzione, permettendo al servizio T1 di evolversi senza influenzare gli altri componenti dell'applicazione.

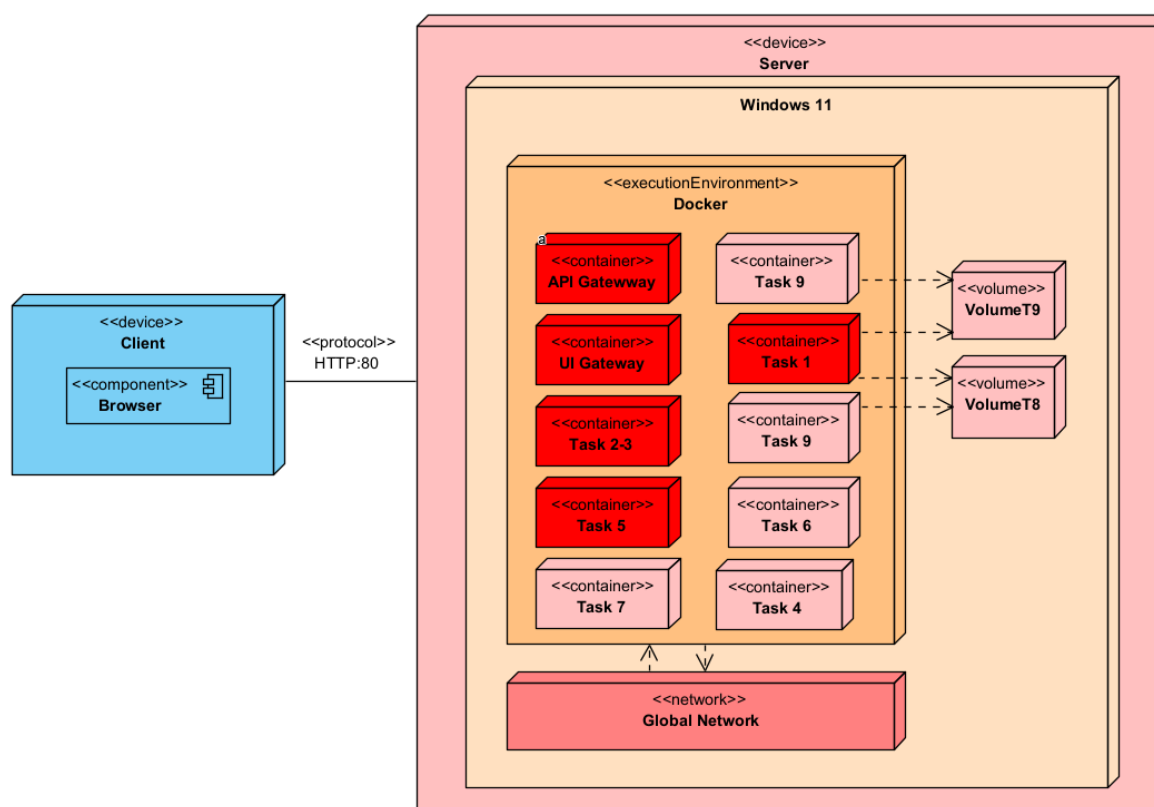
2.4. Deployment Diagram

Il Deployment Diagram è un tipo di diagramma UML utilizzato per rappresentare la distribuzione dei componenti software nell'ambiente di esecuzione; è fondamentale per comprendere le interazioni tra le varie sotto-parti del sistema.

Nel diagramma di deployment sono individuabili diversi elementi chiave:

- **Device:** rappresenta l'hardware su cui il sistema è eseguito.
- **Execution environment:** definisce l'ambiente software in cui operano i componenti.
- **Container:** indica i microservizi eseguiti all'interno di un ambiente virtualizzato.
- **Component:** rappresenta i moduli software specifici, come il browser nel client.
- **Network:** mostra le connessioni tra i vari elementi del sistema.
- **Volume:** rappresenta lo storage persistente utilizzato da alcuni microservizi.

Il diagramma successivo fornisce una panoramica dell'architettura del sistema, che utilizza Windows 11 come ambiente server e Docker come piattaforma di esecuzione dei microservizi. L'accesso al sistema avviene tramite un client browser, che comunica tramite il protocollo HTTP:80 con i gateway di sistema.



Come già menzionato, il sistema è composto da diversi microservizi in esecuzione su container Docker. I microservizi evidenziati in rosso sono quelli da modificare o espandere durante il corso del progetto.

Il microservizio T1 sarà modificato principalmente con l'aggiunta di nuove pagine web che consentiranno all'amministratore di accedere alle funzionalità aggiuntive a lui dedicate (descritte nel capitolo precedente dai requisiti RF01-RF07 e RF12-RF16). Questa modifica è necessaria poiché T1 gestisce l'interazione del sistema con gli amministratori e gestisce le informazioni relative alle classi Java da testare, che possono essere assegnate come parte di un esercizio. Per questi motivi, T1 rappresenta il punto di partenza nella progettazione delle nuove funzionalità.

Il microservizio T23 sarà ampliato per diventare il principale gestore dei team di studenti, poiché esso già si occupa delle informazioni degli utenti registrati: questa scelta, oltre a garantire una utile affinità semantica, ottimizza le prestazioni nelle richieste di liste di studenti. Di conseguenza, il T1 fungerà da client del server T23, interagendo con esso per l'effettiva gestione dei team, ma sempre sotto la supervisione dell'admin. Inoltre, il T23 già ospita un servizio di invio e-mail che sarà sfruttato per soddisfare i requisiti RF08-RF10.

Il microservizio T5 sarà espanso per gestire anche gli esercizi assegnati ai team. Questa scelta è stata fatta considerando la sua vicinanza al motore di gioco: T5 gestisce già la logica delle partite, rendendolo il servizio più adatto a trattare i dettagli degli esercizi, come gli obiettivi. Sarà, però, il T23 (client) a richiamare il T5 (server) per effettuare le operazioni di creazione, modifica ed eliminazione sugli esercizi e i relativi obiettivi. Inoltre, nel T5 verrà creata una pagina web che permetterà agli utenti di visualizzare i propri team ed esercizi (RF11 e RF17). Questa soluzione è dovuta al fatto che la pagina temporanea del profilo personale dello studente risiede direttamente nel T5. In questo caso il T5 funge da cliente del server T23.

È stato necessario apportare modifiche anche ai gateway UI e API per abilitare la comunicazione tra i microservizi precedentemente riportati, garantendo che tutti i componenti possano interagire in modo fluido e senza interruzioni.

In generale, i microservizi, durante la loro esecuzione, interagiscono con volumi di dati persistenti (come il VolumeT8 e il VolumeT9) e sono connessi tramite la rete globale, che assicura la comunicazione e la scalabilità tra i diversi servizi.

Un'attenzione particolare va al T5, dove si è deciso di integrare un database. Questa scelta è dovuta alle difficoltà di modificare il T4, che in futuro potrebbe essere sostituito dal solo T5 per gestire l'intera logica di gioco. In particolare, è stato scelto MongoDB come database.

MongoDB è un database NoSQL orientato ai documenti, progettato per gestire grandi volumi di dati in modo scalabile e flessibile. A differenza dei tradizionali database relazionali (SQL), MongoDB utilizza documenti in formato BSON (una variante binaria di JSON), senza la necessità di definire schemi rigidi. La scelta di MongoDB è stata motivata dalla necessità di un modello dati flessibile, fondamentale per rappresentare in modo efficace gli esercizi e i loro obiettivi.

2.5. Gateway Pattern

Nell'ambito dell'applicazione, è essenziale comprendere come i diversi microservizi dell'architettura interagiscono tra loro, poiché, come evidenziato nel paragrafo precedente, il task assegnato coinvolge più servizi nel sistema software.

Come accennato in passato, per l'integrazione dei microservizi, il sistema adotta l'approccio del Gateway Pattern, un modello architetturale che introduce un livello intermedio sia tra i microservizi sia tra i microservizi e il frontend. Il gateway è suddiviso in due componenti principali

- **UI Gateway:** questo componente è stato implementato utilizzando Nginx con funzionalità di reverse proxy. L'UI Gateway uniforma tutte le richieste indirizzandole al porto 80, semplificando così la gestione della comunicazione. Garantisce elevata scalabilità e distribuzione, consentendo l'instradamento delle richieste su più server e la gestione di grandi volumi di traffico. Gestisce tutte le richieste in arrivo, fornendo un'unica interfaccia verso l'esterno. In altre parole, l'UI Gateway è il punto di ingresso principale del sistema, poiché l'utente interagisce esclusivamente con esso. Le richieste vengono elaborate e, se necessario, inoltrate all'API Gateway. La connessione dell'applicazione con l'utente è quindi logica, poiché l'UI Gateway smista le richieste senza esporre direttamente i microservizi.
- **API Gateway:** questo componente è stato realizzato utilizzando Spring Boot insieme a Netflix Zuul, implementando anch'esso come un reverse proxy. In particolare, l'API Gateway si occupa di gestire le richieste API, consentendo il

routing delle richieste in base alla loro destinazione finale. Gestisce inoltre l'autenticazione e l'autorizzazione tramite token, assicurando che solo gli utilizzatori autenticati possano accedere alle risorse. Inoltre, centralizza l'accesso alle API tramite un percorso comune, organizzato sotto l'URL `"/api/"`, per garantire uniformità e facilità di gestione.

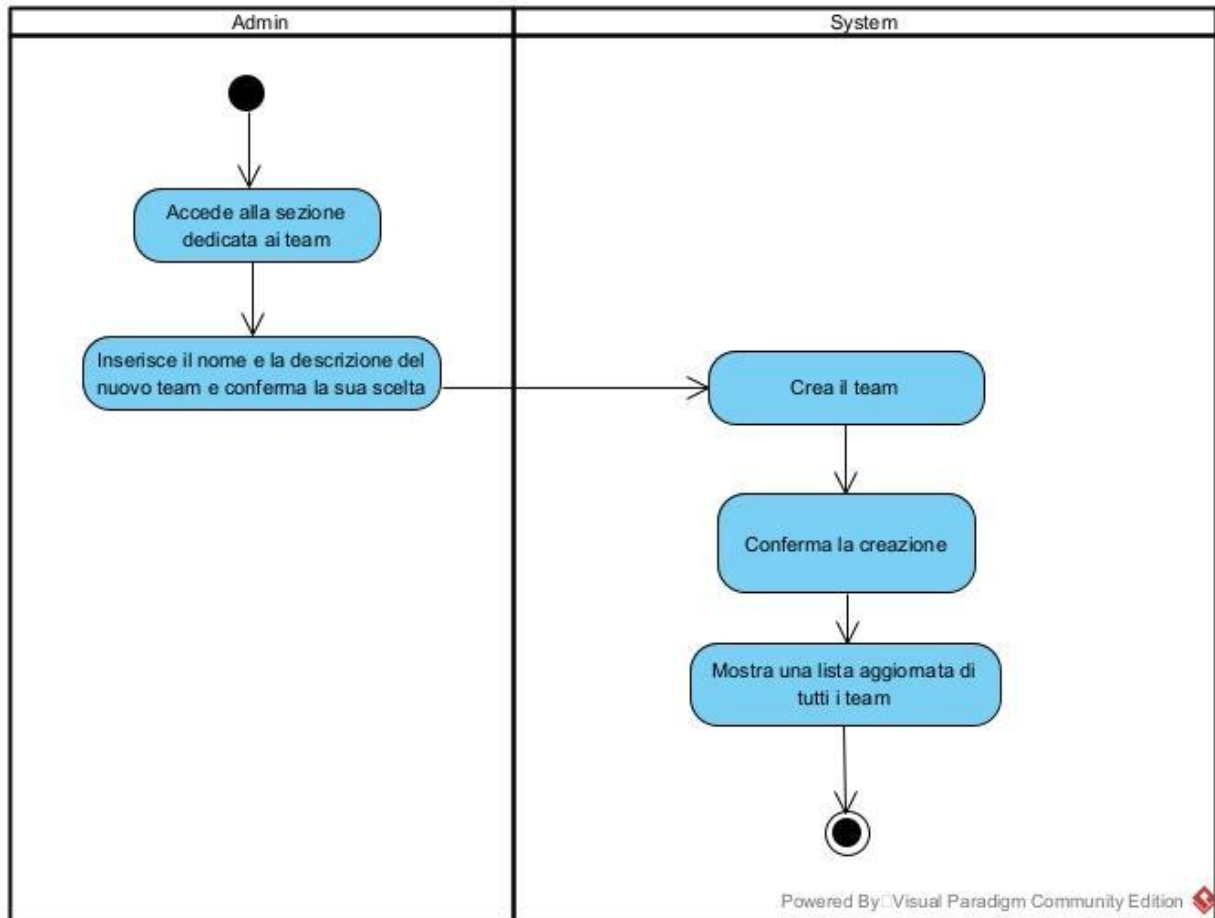
In poche parole, l'UI Gateway rappresenta l'unico punto di accesso al sistema da parte di un utente, mentre l'API Gateway gestisce le richieste API backend, verificando la presenza e validità dei token di autenticazione. A differenza dell'API Gateway, l'UI Gateway dispone di una lista di pagine escluse dal controllo del token, come quelle di login e registrazione. Se la richiesta riguarda una pagina non inclusa nella lista, viene richiesta l'autenticazione e verificato il token dell'utente, seguendo un flusso simile a quello dell'API Gateway.

2.6. Diagrammi di attività

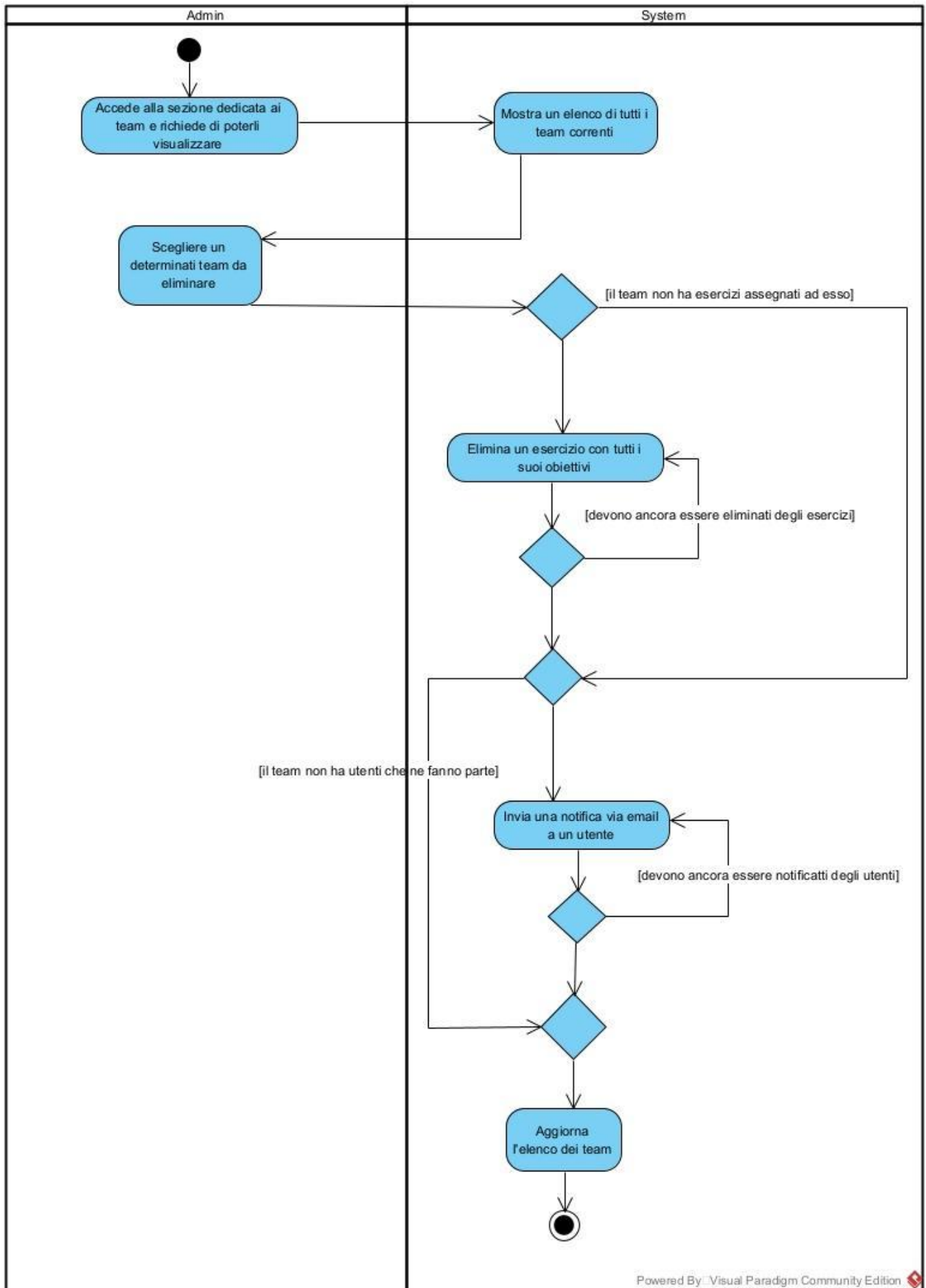
I diagrammi delle attività UML sono uno strumento utile per rappresentare il flusso di lavoro all'interno di un sistema, evidenziando le azioni compiute, le decisioni prese e le interazioni tra le varie componenti. Questi diagrammi sono particolarmente adatti per modellare complesse sequenze di operazioni che devono essere eseguite in risposta a eventi o azioni degli utenti. Prevedono l'utilizzo di simboli standard quali attività, flussi di controllo, nodi decisionali ed elementi di parallelismo per descrivere il comportamento del sistema software.

Nel progetto si è inizialmente proposto l'uso dei diagrammi delle attività per progettare gli step dei processi operativi e mappare le interazioni tra le principali componenti del sistema. Tuttavia, a seguito di una riflessione più approfondita, si è deciso di adottare i diagrammi di sequenza per definire il funzionamento dinamico del sistema, poiché consentono di rappresentare in maniera dettagliata i flussi e le comunicazioni tra gli oggetti dell'applicazione. Per completezza, di seguito sono riportati alcuni diagrammi di attività a alto livello esemplificativi.

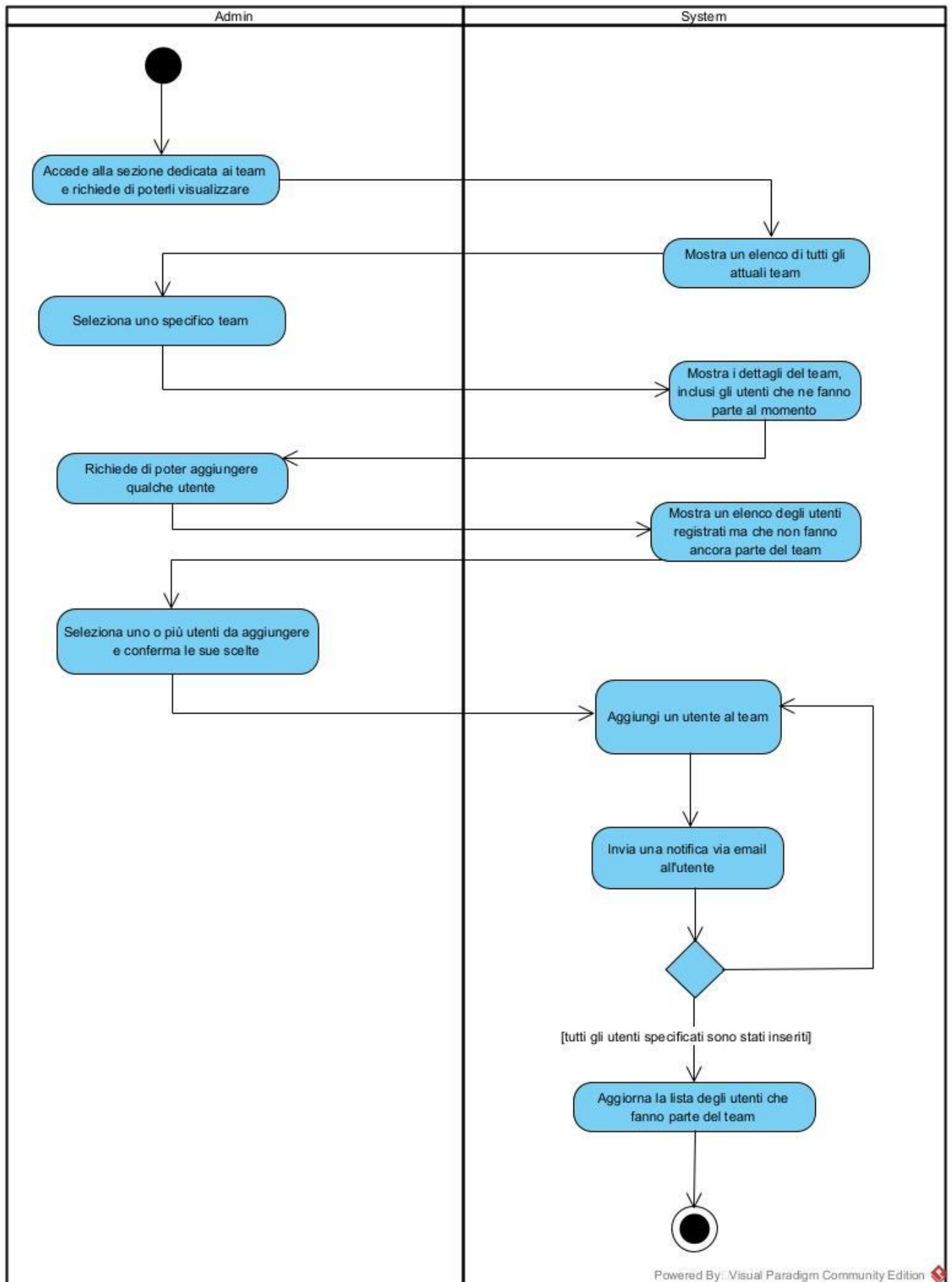
2.6.1. CreateTeam



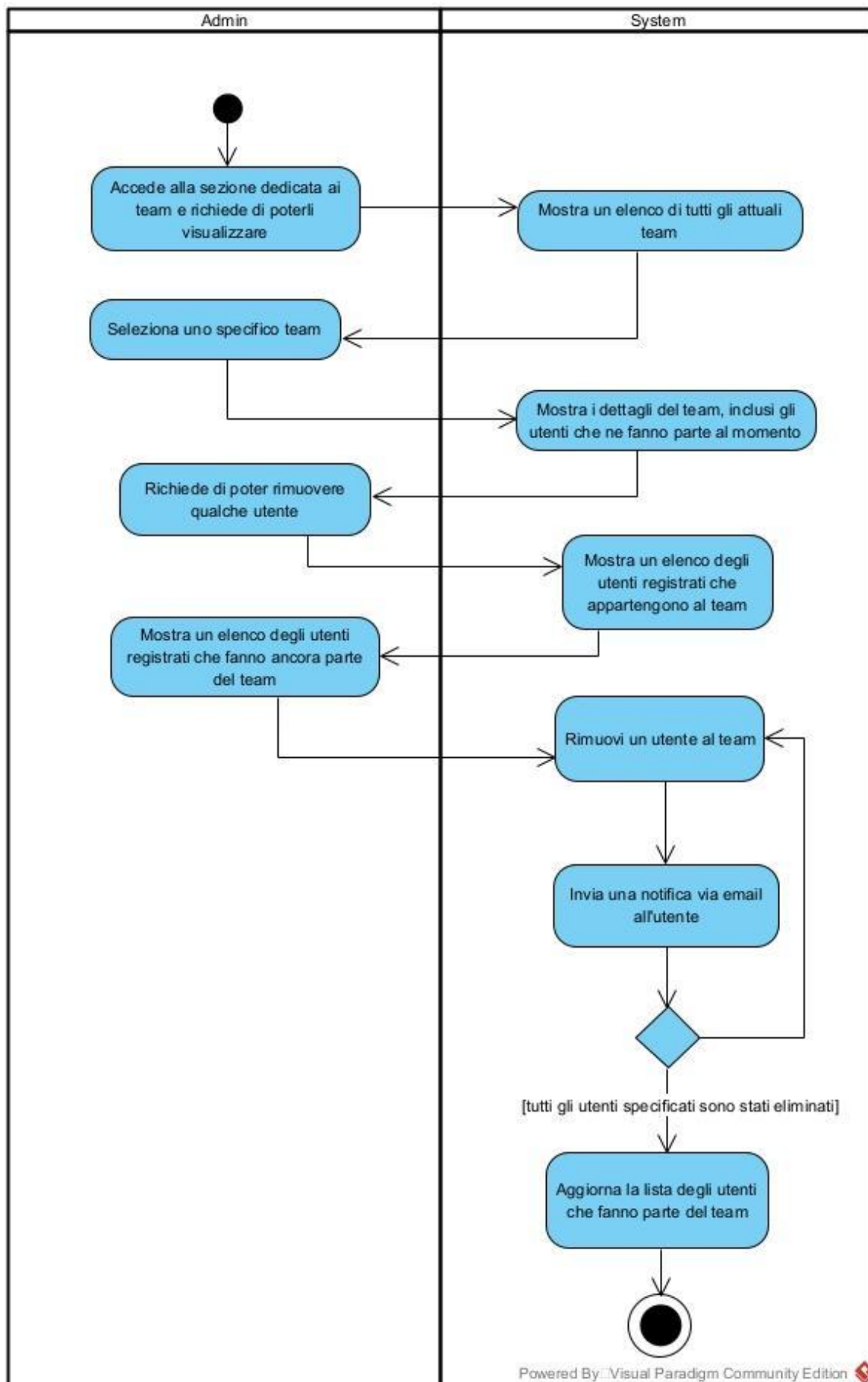
2.6.2. DeleteTeam



2.6.3. AddStudent



2.6.4. DeleteStudent



2.7. Diagrammi di sequenza

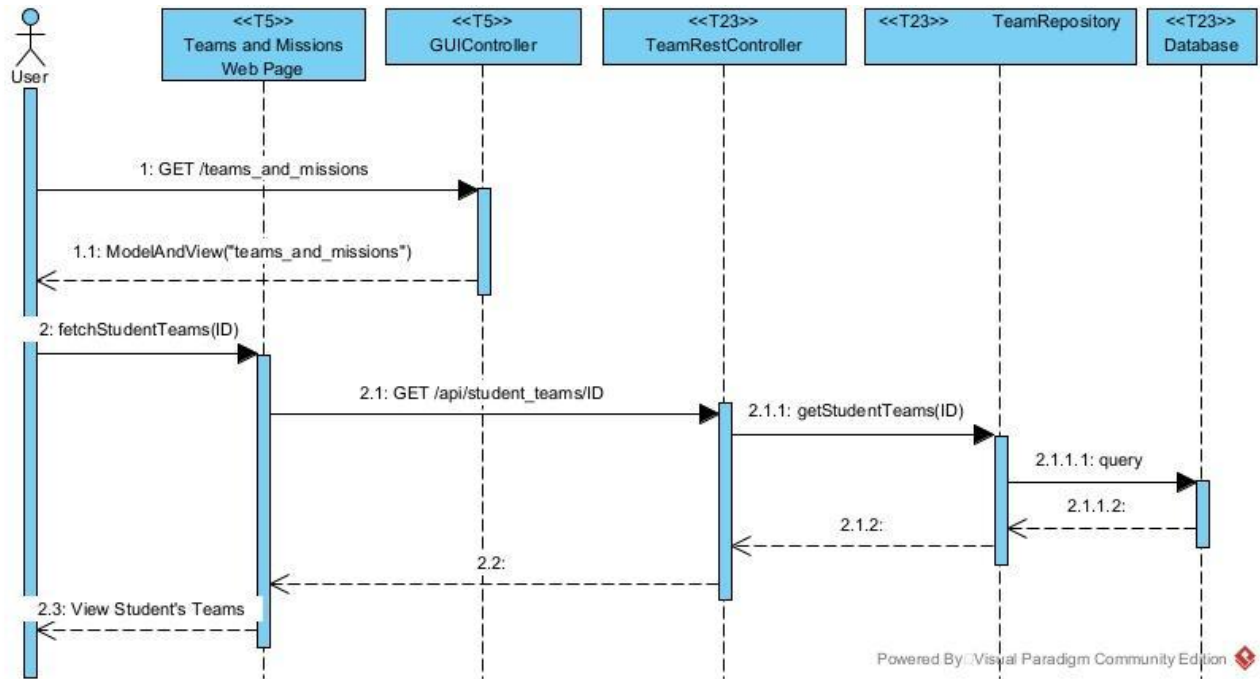
I diagrammi delle sequenze UML forniscono una rappresentazione dettagliata e dinamica dell'interazione tra oggetti e componenti all'interno di un sistema, evidenziando l'ordine temporale dei messaggi scambiati e le collaborazioni necessarie per l'esecuzione di specifici processi.

Questi diagrammi sono particolarmente utili per visualizzare il flusso di controllo e le dipendenze tra i vari attori, come utenti, gateway e microservizi, mettendo in rilievo i passaggi critici come l'autenticazione, il routing e l'interazione con database o altri servizi esterni.

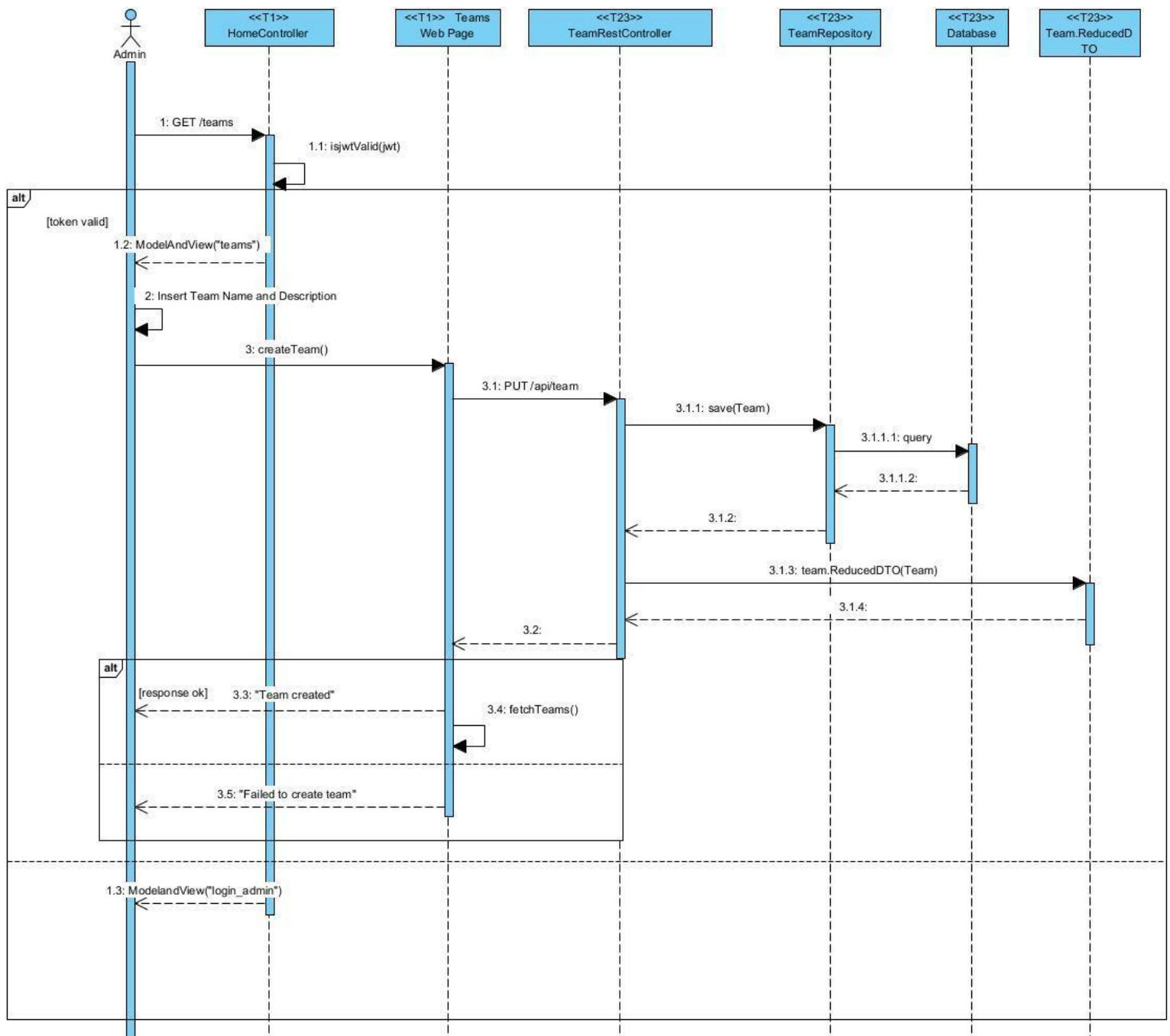
La loro struttura, che si sviluppa lungo una linea temporale verticale, consente di comprendere facilmente come le richieste vengano propagate, elaborate e restituite, semplificando il debugging e l'ottimizzazione delle performance del sistema. In contesti complessi, come quelli descritti nel progetto, l'utilizzo di diagrammi delle sequenze risulta fondamentale per analizzare e documentare il comportamento dinamico del sistema, supportando la collaborazione tra i membri del team di sviluppo e assicurando una visione chiara e condivisa delle interazioni tra i vari componenti.

Prima di presentare i diagrammi, è importante sottolineare che sono state adottate alcune semplificazioni per migliorare la leggibilità. In particolare, gran parte della gestione delle eccezioni è stata omessa e le interazioni tra l'utente e il sistema software (tramite l'UI Gateway) sono state semplificate, trattando la pagina web come una singola lifeline. Inoltre, nei diagrammi non è stata rappresentata la presenza dell'API Gateway nel processo di inoltro delle richieste e risposte HTTP tra i vari servizi, anche se il suo ruolo di intermediario è comunque evidente nelle chiamate che iniziano con `"/api"`. Queste scelte permettono di focalizzarsi esclusivamente sui meccanismi chiave, facilitando la comprensione del comportamento complessivo del sistema. Infine, i diagrammi di sequenza riportati di seguito sono stati creati per fare il design delle interazioni tra il T1 e il T23 e, pertanto, non includono interazioni con il T5.

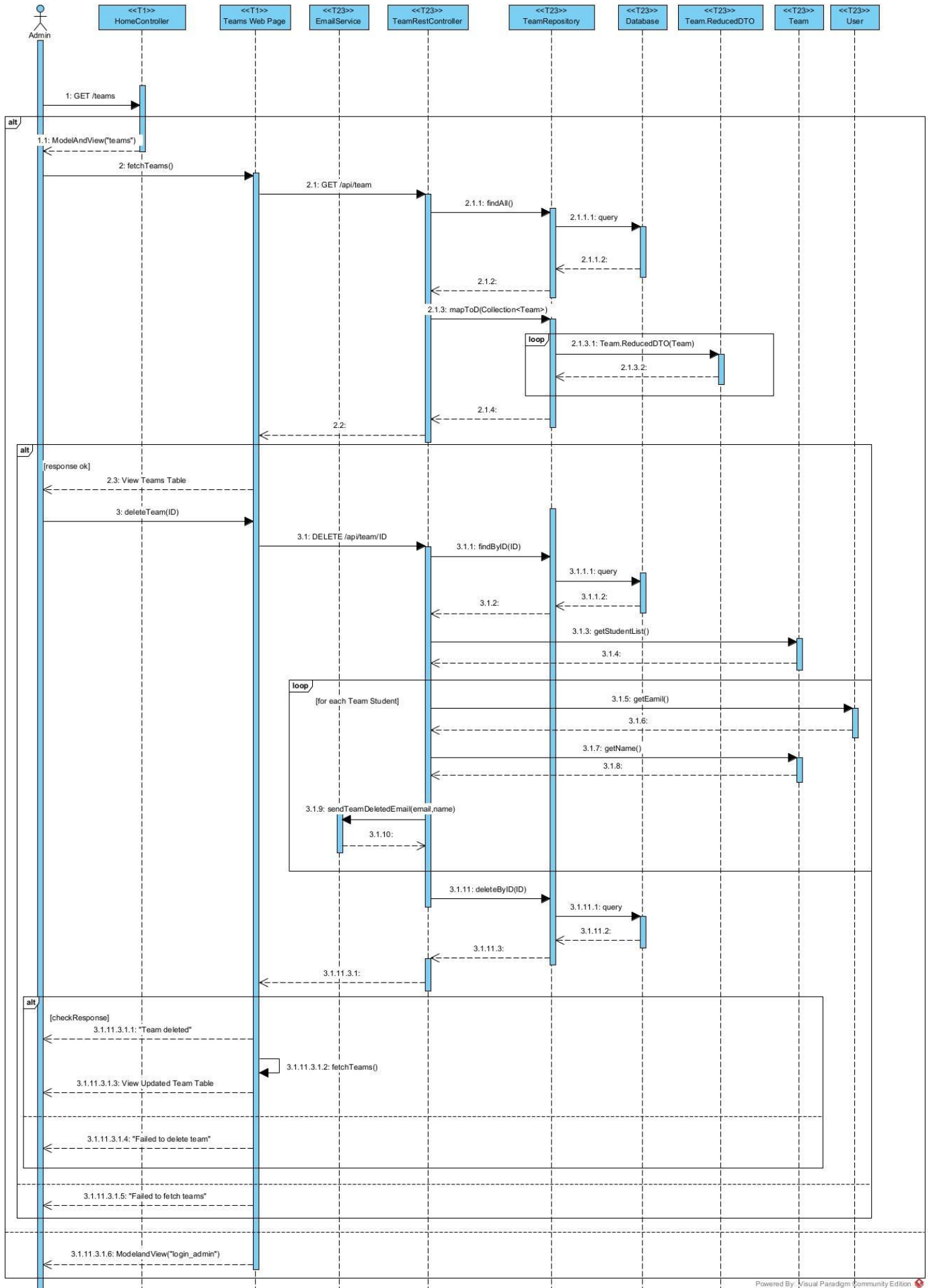
2.7.1. ViewStudentTeams



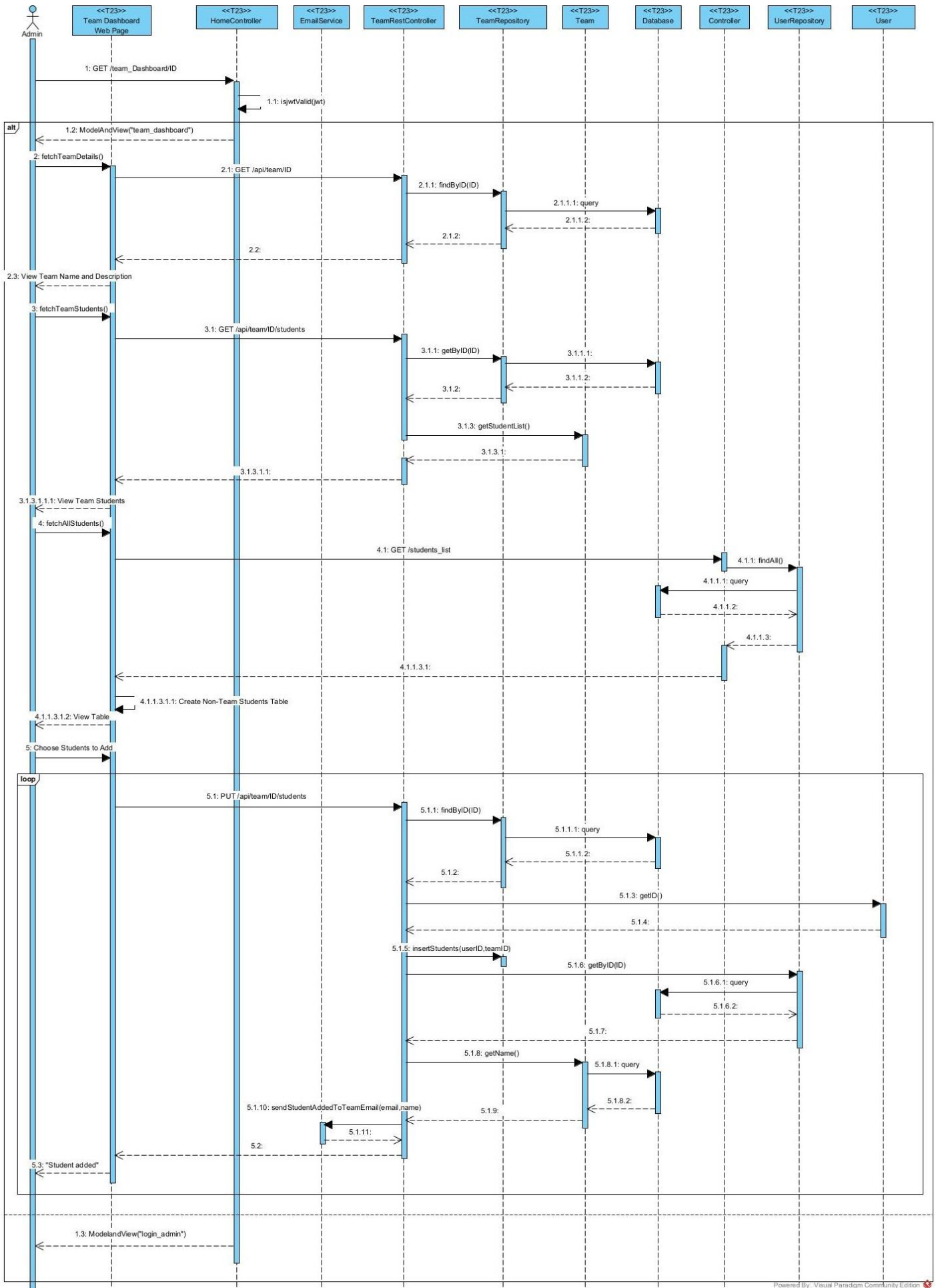
2.7.2. CreateTeam



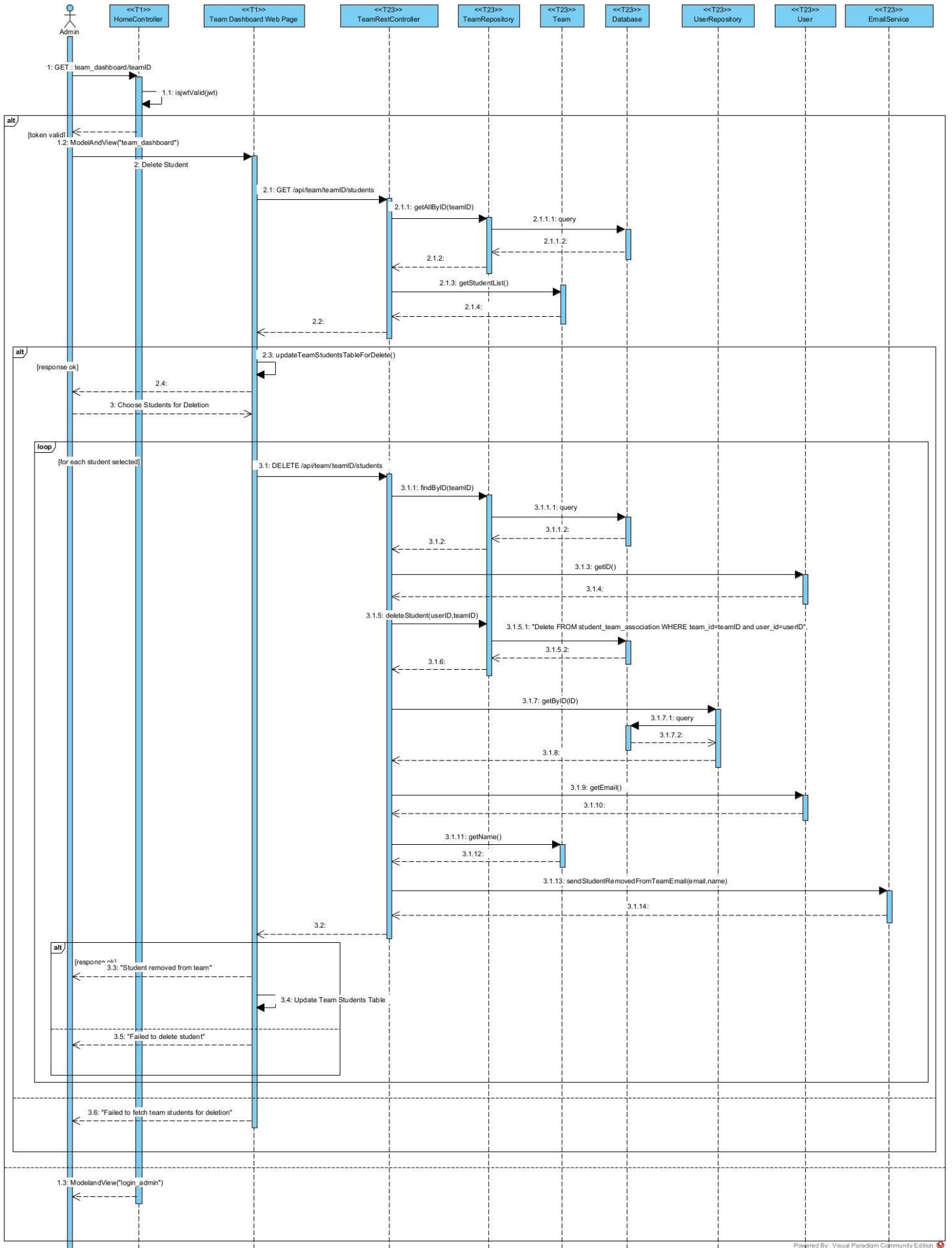
2.7.3. DeleteTeam



2.7.4. AddStudent



2.7.5. DeleteStudent



3. IMPLEMENTAZIONE

In questo capitolo verrà illustrata l'implementazione del lavoro proposto nel capitolo precedente sulla web application, con particolare attenzione ai cambiamenti apportati ai servizi T23 e T5. Nei paragrafi seguenti, ciascun servizio sarà analizzato nel dettaglio, descrivendo le modifiche effettuate per integrare le funzionalità proposte e garantire la compatibilità con il sistema esistente.

Per ogni servizio, sarà presentata una tabella riepilogativa delle nuove API esposte, con le relative specifiche tecniche e funzionali. Infine, verrà fornita una descrizione del frontend, evidenziando le scelte di design adottate per garantire un'interfaccia utente intuitiva ed efficiente.

Si sorvolerà sulla struttura del dato nel database perché è totalmente descritta dalla classe Java e, a meno di ottimizzazioni che verranno esplicitamente illustrate, soggetta ai vincoli dei framework utilizzati.

3.1. Servizio T23

Di seguito viene fornita una descrizione dettagliata delle modifiche implementate al servizio T23, responsabile della gestione dei team di utenti e degli esercizi ad essi assegnati. L'aggiornamento introduce tre classi principali, ognuna con un ruolo specifico nell'architettura del servizio T23.

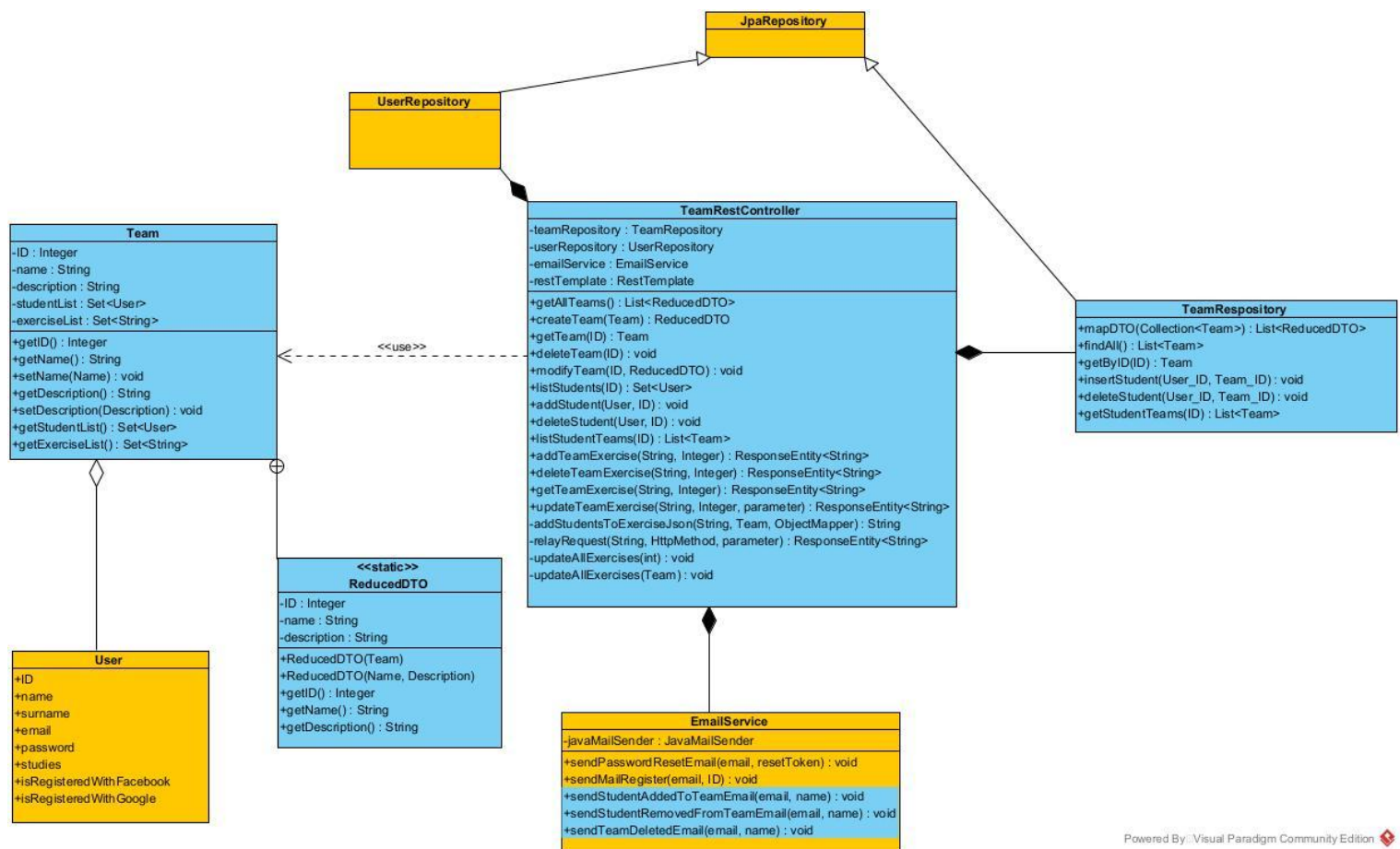
La classe Team è l'entità rappresentativa dei gruppi di utenti nel sistema. Gli attributi della classe descrivono le caratteristiche tipiche di un team, mentre i metodi consentono di accedere e modificare questi dati.

La classe TeamRestController si occupa della gestione delle richieste HTTP. Al suo interno sono definite le API che consentono di eseguire operazioni sui team, come la gestione degli studenti associati e l'assegnazione degli esercizi.

Infine, la classe TeamRepository si occupa dell'interazione con il database del servizio. Estendendo JpaRepository, eredita automaticamente le sue operazioni di base. Tuttavia, sono state necessarie implementazioni personalizzate per creare query più specifiche.

Inoltre, in T23 sono state implementate nuove funzionalità nell'E-mail Service, che ora gestisce l'invio di notifiche via e-mail agli utenti quando vengono aggiunti o rimossi da un team, o quando un team di cui fanno parte viene cancellato.

Questa struttura garantisce un codice ben organizzato, promuovendo un'architettura modulare, scalabile e facilmente manutenibile. La separazione delle responsabilità tra le classi consente inoltre di aggiungere nuove funzionalità in futuro senza compromettere l'integrità del sistema.



L'immagine superiore riporta un dettagliato diagramma delle classi che illustra i cambiamenti introdotti in T23. Le classi evidenziate in giallo erano già presenti e non sono state modificate, mentre quelle in blu rappresentano le nuove aggiunte.

I diagrammi di classi sono uno strumento dell'Unified Modeling Language utilizzato per rappresentare le classi di un sistema, le loro caratteristiche (attributi) e i comportamenti (metodi), nonché le relazioni tra di esse. Mostrano l'architettura statica del sistema, con associazioni come l'ereditarietà, la composizione e l'aggregazione.

3.1.1. Team

La classe Team rappresenta l'entità dei team all'interno del sistema, ossia gruppi di studenti che vengono trattati nello stesso modo (ad esempio, a cui vengono assegnati gli stessi esercizi o a cui è associata una determinata descrizione). È mappata su una tabella del database chiamata Teams, che si trova nello schema studentsrepo. Grazie all'annotazione @Entity, questa classe viene riconosciuta come un'entità JPA, il che significa che può essere facilmente gestita con il framework Spring per la persistenza dei dati. Anche nelle fasi successive, si fa ampio uso di annotazioni, che semplificano la programmazione dei vari aspetti del sistema software, dalla definizione delle entità alle operazioni, riducendo così la complessità del codice e migliorando l'efficienza complessiva.

Ogni team ha un ID univoco, che viene generato automaticamente, un nome e una descrizione. Questi attributi sono semplici e servono a identificare il team e fornire un breve riassunto.

Un aspetto più complesso è la gestione degli studenti che fanno parte del team. Per questo viene utilizzata una relazione Many-to-Many tra Team e User. In pratica, un team può avere più studenti e ogni studente può appartenere a più team. Per rappresentare questa relazione nel database, viene creata una tabella di associazione chiamata Student_Team_Association. In questa tabella, ogni riga contiene un team_id e un user_id, collegando così gli studenti ai team in modo efficiente. La relazione è impostata con FetchType.LAZY, il che significa che la lista di studenti viene caricata solo quando effettivamente necessaria, evitando sprechi di risorse. Nella classe Team, l'attributo che rappresenta gli studenti è un Set<User>.

Oltre agli studenti, ogni team ha anche una lista di esercizi assegnati. Questa informazione viene gestita in modo diverso rispetto agli studenti: invece di essere una relazione con un'altra entità, è semplicemente un insieme di stringhe (Set<String>) che rappresentano gli ID degli esercizi. Questa scelta è legata al fatto che gli esercizi non sono gestiti in questo servizio, come nel caso degli utenti, ma risiedono in un altro servizio, in particolare nel T5. Gli ID degli esercizi vengono salvati in una tabella chiamata Student_Exercise_Association, dove ogni riga contiene un team_id e un exercise_id. A differenza della lista di studenti, qui viene usato FetchType.EAGER, quindi gli identificatori degli esercizi vengono caricati immediatamente quando si recupera un team dal database.

Per gestire i dati, la classe Teams include metodi getter e setter per ogni attributo, permettendo di leggere ed eventualmente anche modificare il nome, la descrizione, gli studenti e gli esercizi associati a un gruppo. Nel dettaglio, gli studenti e gli esercizi sono accessibili tramite getter, ma non è previsto un setter per questi attributi.

Infine, c'è una classe interna e statica chiamata ReducedDTO, che serve per creare una versione "ridotta" dell'oggetto Team. Quando si deve restituire un team in una risposta API, a volte non è necessario inviare tutte le informazioni (ad esempio, la lista degli studenti potrebbe essere pesante da caricare). ReducedDTO permette di ottenere solo i campi essenziali: ID, name e description. In questa classe, ci sono due costruttori: uno che riceve un oggetto Team e ne estrae i dati principali e uno che permette di creare un DTO direttamente da un nome e una descrizione.

3.1.2. TeamRepository

La classe TeamRepository estende l'interfaccia JpaRepository, il che gli consente di utilizzare i metodi predefiniti di Spring Data JPA per interagire con il database in modo semplice ed efficiente. Questo repository si occupa della gestione dei dati relativi ai team, come l'inserimento, la modifica, la rimozione e il recupero delle informazioni dal database. Inoltre, offre metodi personalizzati per eseguire operazioni specifiche non coperte dai metodi generici di JPA.

Uno dei metodi principali è mapDto(Collection), un metodo statico che converte una collezione di oggetti Team in una lista di Team.ReducedDTO. I DTO (Data Transfer Object) servono a restituire solo le informazioni necessarie su un team, evitando di caricare dati non rilevanti dal database. Questo approccio permette di ridurre il carico e migliorare le performance delle API.

Il metodo findAll(), predefinito di JpaRepository, restituisce tutti i record della tabella teams nel database come una lista di oggetti Team. Un altro metodo, getById(ID), restituisce invece il team con l'ID specificato. A differenza di findById(), che ritorna un Optional, getById() restituisce direttamente il team e lancia un'eccezione se il team con quell'identificativo non esiste.

Il repository include anche altri metodi, in particolare per quanto riguarda le associazioni tra gli studenti e i team. In questi casi si è ritenuto opportuno specificare la query SQL adatta per agire nella maniera più efficiente, mediante l'annotazione @Query. Ad esempio, insertStudent(userID, teamID) esegue una query SQL personalizzata per assegnare uno studente a un team tramite l'inserimento di una riga nella tabella student_team_association (necessita solo degli id delle due

entità). L'alternativa gestita da JPARespository avrebbe richiesto: l'istanziamento dell'oggetto Team; l'accesso alla lista di studenti, che avrebbe scatenato un join sul database; l'inserimento dello studente nella lista di studenti in Java; la chiamata al metodo di update, il quale avrebbe risolto le differenze tra il database e la lista java prima di fare un inserimento. Ulteriormente, grazie alle annotazioni @Modifying e @Transactional, si garantisce che la modifica venga salvata nel database e che la transazione venga gestita automaticamente. Allo stesso modo, deleteStudent(userID, teamID) rimuove un'associazione tra uno studente e un team, eliminando la riga corrispondente dalla tabella student_team_association.

Infine, il metodo getStudentTeams(ID) esegue una query SQL per recuperare tutti i team a cui uno specifico studente appartiene. La query personalizzata esegue un JOIN tra la tabella teams e la tabella di associazione student_team_association, utilizzando l'ID dello studente per ottenere i team a cui appartiene.

3.1.3. TeamRestController

Il TeamRestController è una classe fondamentale per la gestione delle operazioni relative ai team di studenti all'interno del sistema. Ogni metodo di questa classe gestisce un'azione specifica e interagisce con altre componenti dell'applicazione, come il servizio di invio e-mail, il repository per l'accesso al database e gli altri servizi dell'applicazione, esterni al T23.

Le funzionalità del TeamRestController sono progettate per essere utilizzate come parte di un'API (Application Programming Interface). Quando un client, come un'interfaccia utente o un altro servizio, invia una richiesta HTTP, il controller risponde esponendo metodi che interagiscono con il database tramite il TeamRepository o che si interfacciano con il servizio T5, responsabile della gestione degli esercizi.

Ad esempio, una richiesta GET per recuperare tutti i team (/team) attiva il metodo getAllTeams() del controller, che a sua volta invoca il metodo findAll() del repository per ottenere tutti i dati dal database e restituirli in formato JSON. Le operazioni di modifica, come l'aggiunta o la rimozione di uno studente da un team, sono gestite tramite richieste HTTP di tipo PUT o DELETE, e le azioni vengono eseguite nel database tramite i metodi insertStudent() o deleteStudent() del repository.

In questo modo, ogni metodo del controllore corrisponde a un endpoint API che può essere chiamato da client esterni, consentendo di interagire con i dati relativi ai team di utenti in modo quanto più possibile sicuro e strutturato.

3.1.3.1. Gestione dei Team

La funzione `getAllTeams()` si occupa di recuperare la lista di tutti i team. Una volta ottenuti, i dati vengono trasformati in un formato più conciso, rappresentato da oggetti di tipo `ReducedDTO`, che contengono solo le informazioni essenziali, riducendo la quantità di dati restituiti nelle risposte.

Il metodo `createTeam()` consente di creare un nuovo team. Quando il controller riceve un oggetto `Team` nel corpo della richiesta, lo salva nel database tramite una funzione del `TeamRepository` e restituisce un `DTO` del team appena creato, confermando così al client l'avvenuta creazione.

Il metodo `getTeam(ID)` restituisce il team il cui ID corrisponde al valore fornito come parametro in ingresso. Se non esiste un team con l'ID fornito, il metodo non restituisce nulla (o potrebbe sollevare un'eccezione, a seconda dell'implementazione).

La funzione `deleteTeam(ID)` è responsabile dell'eliminazione di un team. Prima di cancellare il team dal database, il controller invia notifiche via e-mail a tutti gli studenti che ne fanno parte, così da informarli della sua eliminazione. Questo processo richiede che siano recuperate le e-mail degli utenti a partire dai loro identificatori. Un vantaggio di gestire i team all'interno del servizio T23 è che l'accesso alle informazioni degli studenti risulta semplificato, rendendo sufficiente l'uso di un oggetto `UserRepository`, senza la necessità di interagire con altri servizi. Successivamente, il controller rimuove il team dal sistema, eliminando anche qualsiasi esercizio associato. Quest'ultima operazione, tuttavia, richiede l'interazione con il servizio T5, responsabile degli esercizi. T5 mette a disposizione delle API specifiche che consentono al `TeamRestController` di interagire opportunamente con il sistema di gestione degli esercizi. Queste funzioni dovrebbero essere richiamate esclusivamente da T23 e non da altre parti del sistema, al fine di garantire la coerenza delle informazioni sugli esercizi salvate all'interno del servizio T23.

Con il metodo `modifyTeam(ID, DTO)`, il controller consente di aggiornare il nome e la descrizione di un determinato team. Se il team esiste, i nuovi valori, forniti dal `DTO` in input,

vengono utilizzati per aggiornare i dati nel database. Se il team con lo specifico ID non viene trovato, viene sollevata un'eccezione per segnalare l'errore.

3.1.3.2. Gestione degli Studenti nei Team

La funzione `listStudents(ID)` restituisce un insieme di oggetti `User`, rappresentanti gli studenti che appartengono a quello specifico team.

Il metodo `addStudent(User, team_id)` consente di aggiungere un nuovo studente a un team esistente. Quando un oggetto `User` viene ricevuto nel corpo della richiesta, il controller verifica che il team esista e, successivamente, inserisce lo studente nella tabella `student_team_association`. Dopo aver aggiornato il database, viene inviata una e-mail di notifica allo studente per informarlo dell'aggiunta al team. Infine, gli esercizi assegnati al team vengono aggiornati, richiamando un metodo ausiliario descritto più avanti.

Al contrario, `deleteStudent(User, team_id)` permette di rimuovere uno studente da un team. Dopo aver eliminato l'associazione nel database, viene sempre inviata una e-mail di notifica allo studente, informandolo della sua rimozione dal team. Anche in questo caso, vengono aggiornate le missioni del team. Se lo studente non è associato al team, viene sollevata un'eccezione.

3.1.3.3. Gestione degli Esercizi assegnati ai Team

La funzione `addTeamExercise(json, ID)` permette di associare un esercizio a un team. Una volta ricevuto un JSON contenente informazioni sull'esercizio da aggiungere, il controller aggiunge ad esso l'ID del team e l'elenco degli ID degli studenti del team e invia una richiesta HTTP al servizio esterno per creare l'esercizio. Se la risposta è positiva, l'ID dell'esercizio restituito viene aggiunto alla lista degli esercizi del team e il team viene aggiornato nel database.

La funzione `deleteTeamExercise(mission_id, team_id)` gestisce la rimozione di un esercizio da un team. Il controller verifica se il team e il suo esercizio esistono e, in caso affermativo, invia una richiesta di eliminazione al T5. Se l'esercizio viene rimosso con successo, il controller aggiorna la lista degli esercizi del team nel database del T23.

Il metodo `getTeamExercise(mission_id, team_id)` consente di ottenere informazioni su un determinato esercizio assegnato a un team da un amministratore. Il controller invia una richiesta

GET al servizio esterno con l'ID della missione per ottenerne i dettagli e restituisce la risposta al cliente.

Infine, `updateTeamExercise(mission_id, team_id, json)` aggiorna uno specifico esercizio. La funzione invia una richiesta POST al T5 con le informazioni aggiornate sull'esercizio. Anche in questo caso, gli identificatori degli studenti e del team vengono inclusi nel JSON prima di inviare la richiesta.

3.1.3.4. Funzioni Ausiliarie

I metodi descritti si avvalgono di alcune funzioni ausiliarie private. In particolare, `addStudentsToExerciseJson(json, team, mapper)` si occupa di arricchire il JSON dell'esercizio con le informazioni necessarie al T5 per poter gestire correttamente l'aggiunta di una missione. Il metodo `relayRequest(mission_id, method, body)` funge da intermediario nella comunicazione con il servizio esterno, inviando richieste HTTP per l'aggiunta, l'aggiornamento o la rimozione di un esercizio. Infine, `updateAllExercises(Team)` e `updateAllExercises(int)` si occupano di sincronizzare tutti gli esercizi di un team, aggiornando ciascuno di essi con le informazioni più recenti sugli studenti.

Le descrizioni fornite offrono una panoramica generale delle classi, senza entrare nel dettaglio di tutti i meccanismi e dei vari controlli. Per un'analisi più approfondita, che includa il flusso di esecuzione e la gestione delle eccezioni, è consigliabile consultare direttamente il codice sorgente.

Di seguito è riportata una tabella riassuntiva di tutte le API aggiunte nel servizio T23 dell'architettura.

Nome	Metodo HTTP	Endpoint	Descrizione	Path Variables
List Teams	GET	api/team	Ottiene una lista dei team	Nessuna
Create Team	PUT	api/team	Crea un team	Nessuna
Add Student	PUT	api/team/:ID/students	Aggiunge uno studente a un team	ID del team
Remove Student	DELETE	api/team/:ID/students	Rimuove uno studente da un team	ID del team

Delete Team	DELETE	api/team/:ID	Elimina un team	ID del team
Patch Team	PATCH	api/team/:ID	Modifica un team	ID del team
Get Team	GET	api/team/:ID	Ottiene un team	ID del team
Add Exercise	PUT	api/team/:ID/exercise	Assegna un esercizio a un team	ID del team
Delete Team	DELETE	api/team/:tid/exercise/:mid	Elimina un esercizio assegnato a un team	ID del team (tid), ID dell'esercizio (mid)
Get Exercise	GET	api/team/:tid/exercise/:mid	Ottiene un esercizio assegnato a un team	ID del team (tid), ID dell'esercizio (mid)
Patch Exercise	POST	api/team/:tid/exercise/:mid	Modifica un esercizio assegnato a un team	ID del team (tid), ID dell'esercizio (mid)

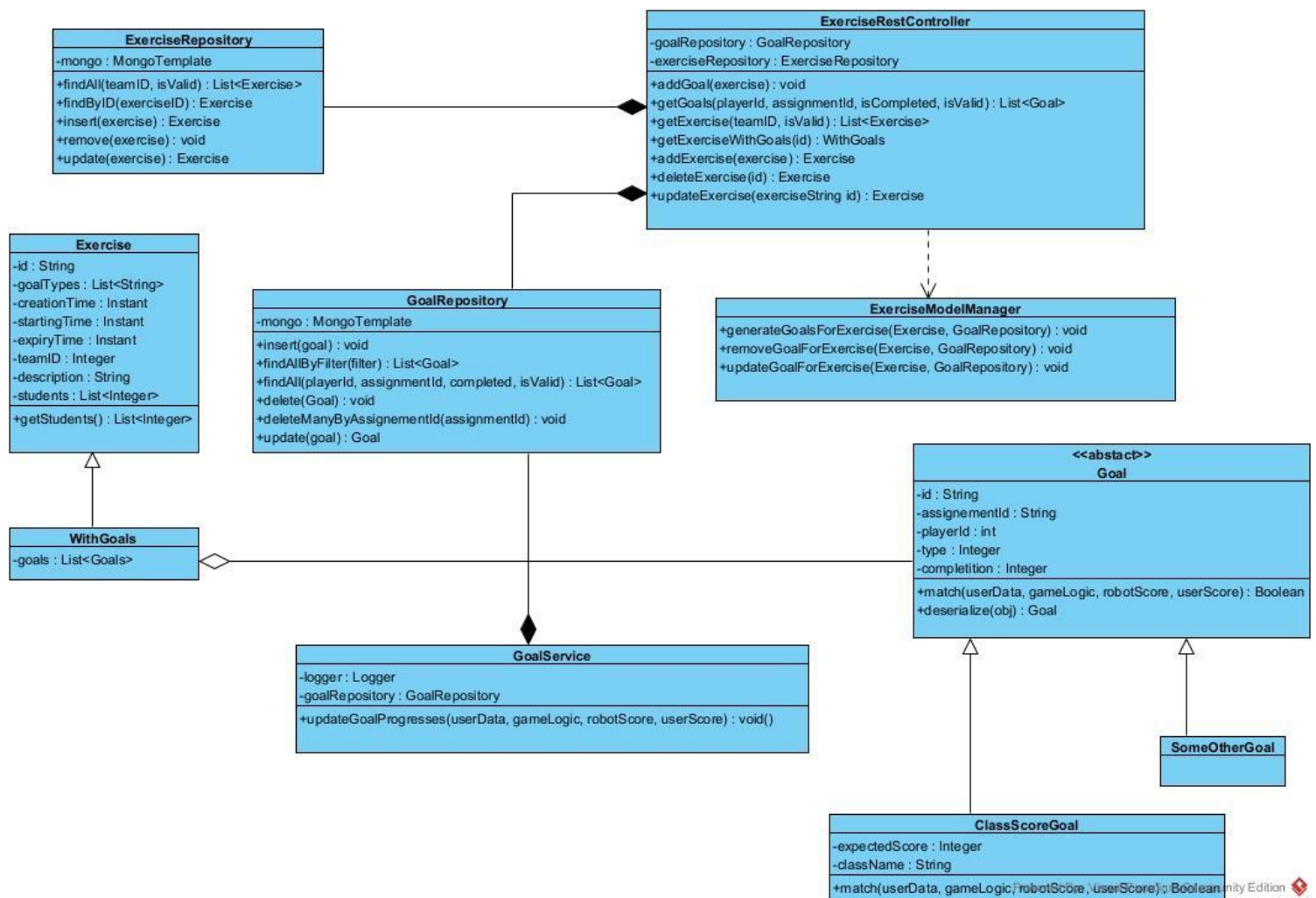
Per semplicità di lettura, si riporta nella tabella seguente il body della richiesta HTTP. In questa documentazione viene mostrato nel dettaglio solo il corpo della richiesta, ma il corpo della risposta è abbastanza intuitivo.

Nome	Request Body
List Teams	Nessuno
Create Team	{ "name": string, "description": string }
Add Student	{ "ID": "integer" }
Remove Student	{ "ID": "integer" }
Delete Team	Nessuno
Patch Team	{ "name": string, "description": string }
Get Team	Nessuno
Add Exercise	{ "goalTypes": ["string"], "expiryTime": string*, "creationTime": string*, "startingTime": string*, "description": string } *ISO timestamp

Delete Team	Nessuno
Get Exercise	Nessuno
Patch Exercise	{ "expiryTime": string*, "description": string } *ISO timestamp

3.2. Servizio T5

In questo paragrafo vengono descritte le modifiche implementate nel servizio T5, responsabile della gestione degli esercizi e dei relativi obiettivi. Nel diagramma delle classi seguente sono rappresentate le principali classi aggiunte.



La scelta di MongoDB è stata fatta per garantire le caratteristiche di polimorfismo e modularità prescritte nel precedente capitolo. Da un punto di vista implementativo, infatti, è possibile memorizzare oggetti che differiscono per qualità e quantità di attributi nella stessa collezione, e

per questo si possono aggiungere nuovi tipi di obiettivo senza modificare le logiche fondamentali già implementate oppure il database.

3.2.1. Gli esercizi

La classe `Exercise` rappresenta un esercizio all'interno del sistema ed è configurata come una raccolta in MongoDB. Questa classe include campi essenziali, come l'identificatore, il timestamp di creazione, la data di inizio e la scadenza dell'esercizio. Questi elementi permettono di definire e gestire un esercizio in modo completo, garantendo tutte le informazioni necessarie. La classe possiede anche una lista di tipi di goal, ognuno dei quali è una descrizione sintetica JSON di un obiettivo.

La classe `Exercise` viene estesa da `WithGoal`, un DTO arricchito con una lista di oggetti `Goal`, che rappresentano i progressi dei giocatori. Poiché ciò richiede una query aggiuntiva al database, si è preferito recuperare manualmente tali dati solo quando necessario.

Infine, il campo `students` di `Exercise` e il suo metodo accessorio `getStudents()` sono stati necessari per implementare la logica di creazione degli obiettivi. Tuttavia, il dato proviene sempre dal `RestController` nel momento in cui è necessario generare o aggiornare le associazioni dei progressi ed è, quindi, un dato esterno che non deve essere salvato come attributo dell'esercizio. Per questo motivo, si è utilizzata l'annotazione `@JsonIgnore` sul metodo `getStudents()`, escludendolo dalla serializzazione JSON e impedendo che venga salvato nel database o restituito come campo vuoto negli oggetti JSON.

La classe `ExerciseModelManager` si occupa della gestione delle operazioni di rimozione, aggiornamento e generazione dei goal associati a un esercizio. Fornisce tre metodi statici principali:

- **generateGoalsForExercise**: crea un nuovo goal per ogni combinazione di tipo di goal e studente, inserendoli nell'opportuno repository; si presuppone che l'esercizio in input abbia una lista di studenti.
- **removeGoalsForExercise**: elimina tutti i goal associati a un esercizio dal repository.
- **updateGoalsForExercise**: aggiorna la lista dei goal di un esercizio, verificando le combinazioni studente-tipo e gestendo di conseguenza l'inserimento o la rimozione dei goal dal database. In input, l'esercizio ha la lista completa degli studenti: bisogna

verificare che tutti i goal associati all'esercizio nel database siano associati anche ad uno studente della lista, in seguito bisogna verificare che tutti gli utenti nella lista abbiano dei progressi seppur nulli.

La classe `ExerciseRepository` mette a disposizione le operazioni CRUD sugli oggetti `Exercise` in MongoDB, implementando i metodi `findAll`, `findById`, `insert`, `remove` e `update`.

Infine, la classe `ExerciseRestController` funge da controller REST per la gestione degli esercizi e dei loro goal. Le dipendenze dai repository `exerciseRepository` e `goalRepository` vengono automaticamente iniettate per semplificare l'accesso ai dati. Il controller espone diversi endpoint per l'aggiunta, il recupero, l'aggiornamento e l'eliminazione di esercizi e goal. Più avanti è riportata una tabella con le API aggiunte al servizio T5.

3.2.2. Gli obiettivi degli esercizi

I goal sono definiti dalla classe astratta `Goal`, che ne stabilisce la struttura e le proprietà fondamentali. Questa classe contiene campi come:

- **id**: identificatore del goal,
- **assignmentId**: identificatore dell'assegnazione,
- **playerId**: identificatore del giocatore,
- **type**: tipo di goal,
- **completion**: percentuale di completamento del goal.

Il metodo `match` è astratto e deve essere implementato dalle classi derivate per verificare se i parametri della partita soddisfano le condizioni del goal, aggiornando eventualmente lo stato dell'oggetto Java. Inoltre, la classe astratta implementa il metodo `factory deserialize`, che deserializza un oggetto JSON in un'istanza di una "sottoclasse" di `Goal`.

Due goal che differiscono per `id` e per `completion` sono considerati uguali, poiché, secondo la modellazione logica, non avrebbe senso assegnare due goal dello stesso tipo e con gli stessi parametri a un esercizio: entrambi verrebbero completati con un solo evento. Per implementare questo comportamento, i due attributi sono stati annotati con `@EqualsAndHashCode.Exclude`.

Le operazioni CRUD per la gestione dei goal memorizzati in MongoDB sono affidate alla classe `GoalRepository`.

L'aggiornamento dello stato dei goal avviene tramite la classe `GoalService`, che fornisce un servizio per monitorare i progressi degli obiettivi alla fine di una partita. Questo servizio utilizza `GoalRepository` per recuperare e aggiornare i goal nel database quando viene chiamato a valutare un evento utente.

Il metodo principale della classe è `updateGoalProgresses` e ha il compito di verificare e aggiornare lo stato dei goal in base ai risultati della partita. Questo metodo viene sempre invocato al termine di una partita per aggiornare, se necessario, gli esercizi del giocatore. I suoi parametri includono:

- **userData**: una mappa contenente diverse entità tra cui il codice della classe sotto test,
- **gameLogic**: un'istanza della logica di gioco,
- **playerId**: identificatore del giocatore,
- **robotScore**: punteggio del robot,
- **playerScore**: punteggio del giocatore.

Il metodo recupera tutti i goal non ancora completati e validi per il giocatore specificato, registra il numero di goal da aggiornare e, per ciascuno di essi, verifica se le condizioni sono soddisfatte chiamando il metodo `match`. Se avviene il matching dei requisiti, il goal viene aggiornato nel repository con il nuovo stato e, in futuro, potrebbero essere previste azioni aggiuntive, come la propagazione di eventi o l'invio di notifiche (ad esempio, via e-mail).

Attualmente sono state implementate due sottoclassi di `Goal`:

- **ClassScoreGoal** rappresenta l'obiettivo di ottenere un punteggio specifico su una determinata classe in fase di test. Il metodo di matching confronta i dati di gioco con i due attributi `className` e `expectedScore`, aggiornando lo stato di completamento al 100% se l'obiettivo è raggiunto.
- **SomeOtherGoal** è stata creata come mock per testare il requisito del polimorfismo.

Il polimorfismo, oltre a quello gestito internamente da `MongoDB`, è garantito dall'uso del metodo `factory`, che deserializza la stringa JSON dell'esercizio chiamando i costruttori di ciascuna sottoclasse di `Goal`. Inoltre, `MongoTemplate`, un driver Java, salva il nome del package e della classe Java in un attributo nascosto nel documento del database. Questo permette ai metodi di

ricerca di restituire sempre un oggetto della classe derivata, anche se al driver viene richiesto di restituire un oggetto Goal generico.

La tabella seguente riassume le API integrate nel servizio T5. Queste vengono chiamate in cascata dal T23, il quale gestisce i vincoli di identità referenziale; perciò, si sconsiglia di utilizzare le seguenti direttamente.

Nome	Metodo HTTP	Endpoint	Descrizione	Path Variables	Parameters
Add Exercise	PUT	api/exercise	Crea un esercizio	Nessuna	Nessuno
Get Exercises	GET	api/exercise	Ottiene gli esercizi	Nessuna	teamId, isValid
Get Exercise	GET	api/exercise/:id	Ottiene un esercizio	ID esercizio	Nessuno
Delete Exercise	DELETE	api/exercise/:id	Elimina un esercizio	ID esercizio	Nessuno
Update Exercise	PATCH	api/exercise/:id	Modifica un esercizio	ID esercizio	
Create Goal	PUT	api/goal	Crea un nuovo obiettivo	Nessuna	
Get Goals	GET	api/goal	Ottiene gli obiettivi	Nessuna	playerId, assignmentId, isCompleted, isValid

Anche in questo caso, per semplicità di lettura viene riportata un'ulteriore tabella con la request body.

Nome	Request Body
Add Exercise	<pre>{ "students": [integer], "goalTypes": [jsonString], "expiryTime": string*, "creationTime": string*, "startingTime": string*, "description": string*, "teamId": 1 }</pre> *ISO timestamp
Get Exercises	Nessuno
Get Exercise	Nessuno

Delete Exercise	Nessuno
Update Exercise	{ "students": [string], "expiryTime": string*, "description": string }
Create Goal	{ "type": integer, ... }
Get Goals	Nessuno

3.3. Front-end (T1 e T5)

Le pagine web aggiuntive sono state implementate in due servizi distinti: T1 e T5. Nel servizio T1 sono state sviluppate le due pagine principali: "teams" e "team_dashboard". La pagina "teams" è accessibile tramite un pulsante aggiunto nell'homepage dell'amministratore. Permette una gestione di base dei team, con la possibilità di crearne di nuovi, modificare nome e descrizione, eliminare quelli esistenti e visualizzare l'elenco aggiornato di tutti i team. Da questa stessa pagina si può accedere al "team_dashboard" dedicato alla gestione dettagliata di uno specifico team. Qui è possibile visualizzare e aggiornare sia la lista degli studenti del team sia degli esercizi assegnati.

Il servizio T5 ospita la pagina "teams_and_missions", attualmente accessibile a un utente tramite un bottone inserito nel profilo temporaneo. Questa pagina mostra in modo chiaro i team a cui l'utente appartiene e i corrispondenti esercizi, offrendo un'esperienza personalizzata.

Ogni pagina è supportata da file HTML, JavaScript e CSS specifici, garantendo una struttura chiara e conforme agli standard di sviluppo web moderni. Di seguito sono riportate delle schermate dell'interfaccia, ma la documentazione include anche un video che mostra un utilizzo pratico del sistema, sia da parte di un amministratore che di un utente, mostrando come le nuove pagine si integrano con quelle esistenti.







3.3.1. Teams

Teams

Team Name:

Team Description:

Create Team

ID	Name	Description	Actions
4	Gruppo A	Descrizione del gruppo A	  
5	Gruppo B	Descrizione del gruppo B	  

Back

3.3.2. Team's Dashboard

Dashboard for Team 4

Name: Gruppo A

Description: Descrizione del gruppo A







Team Students

Id	Name	Surname	Email	Studies
2	Mario	Rossi	cavallosasso1.1@gmail.com	MSc
1	Federica	Del Vecchio	feffedelvecchio@gmail.com	BSc

Add Students

Delete Students

Team Exercises

Description	Goal Types	Starting Time	Expiry Time	Completion	Actions
Descrizione dell'esercizio A.1	Type: 1, Expected Score: 20, Class Name: Calcolatrice	02/02/2025, 16:02:00	09/02/2025, 16:02:00	50.00%	  
Descrizione dell'esercizio A.2	Type: 1, Expected Score: 5, Class Name: VCardBean Type: 1, Expected Score: 25, Class Name: Range	02/02/2025, 16:03:00	23/02/2025, 16:03:00	0.00%	  

AddExercise

Back

3.3.3. Student's teams and missions

Student's Teams

Gruppo A

Descrizione del gruppo A

Description: Descrizione dell'esercizio A.1

Starting Time: 02/02/2025, 16:02:00

Expiry Time: 09/02/2025, 16:02:00

Goal Types:

Type: 1, Expected Score: 20, Class: Calcolatrice

Description: Descrizione dell'esercizio A.2

Starting Time: 02/02/2025, 16:03:00

Expiry Time: 23/02/2025, 16:03:00

Goal Types:

Type: 1, Expected Score: 5, Class: VCardBean

Type: 1, Expected Score: 25, Class: Range

Gruppo B

Descrizione del gruppo B

Description: Descrizione dell'esercizio B.1

Starting Time: 02/02/2025, 16:05:00

Expiry Time: 02/03/2025, 16:05:00

Goal Types:

Type: 1, Expected Score: 80, Class: Calcolatrice

4. SVILUPPI FUTURI

Per migliorare l'applicazione, sono state individuate diverse aree su cui intervenire, con l'obiettivo di rendere l'esperienza dell'utente più fluida e soddisfacente.

Prima di tutto, si potrebbe creare un sistema più efficiente dal punto di vista dei partecipanti ai team. Un miglioramento utile sarebbe l'invio di notifiche via e-mail per avvisare tempestivamente gli utenti sull'assegnazione delle missioni e per ricordare gli esercizi in scadenza, inviando un promemoria prima della scadenza stessa.

Inoltre, la pagina a cui accede lo studente del team potrebbe essere arricchita con funzionalità più avanzate, come la visualizzazione dei progressi per ciascun esercizio. Tuttavia, questi miglioramenti non sono ancora stati implementati perché è in corso lo sviluppo di una nuova versione del profilo utente. Per questo motivo, invece di aggiungere queste funzionalità su una pagina separata, si prevede che vengano integrate direttamente nel nuovo profilo, centralizzando così la gestione delle informazioni personali dello studente.

Le pagine web a cui accedono gli amministratori sono più complete, ma potrebbero essere ulteriormente migliorate separando gli esercizi tra attivi/non attivi e completati/non completati. Anche se queste informazioni sono già visibili, potrebbero essere meglio evidenziate per semplificarne la consultazione. Oppure si potrebbero prevedere dei meccanismi di filtraggio sia per quanto riguarda gli studenti che gli esercizi.

Inoltre, è necessario migliorare la validazione degli input. Ad esempio, è importante controllare che i tempi associati agli esercizi siano coerenti: il `creation_time` deve essere minore o uguale allo `starting_time`, e quest'ultimo deve essere inferiore all'`expiry_time`. Un altro controllo importante sarebbe verificare che la classe Java da testare esista effettivamente. Questo potrebbe essere semplificato permettendo agli amministratori di selezionare una classe tra quelle già presenti nel sistema, invece di dovere digitare manualmente il nome, proprio come accade già per gli studenti. Si prevede che questa modifica non sia complessa, poiché le classi da testare si trovano proprio nel servizio T1, dove si trovano anche le pagine web dell'admin.

Un cambiamento relativamente semplice ma molto potente da implementare sarebbe l'aggiunta di nuovi tipi di obiettivi. Questo consiste nell'estendere la classe `Goal` per aggiungere un nuovo

obiettivo che implementi un metodo di match appropriato. Inoltre, il metodo `Goal.deserialize` dovrebbe essere aggiornato per associare l'intero Type alla classe creata.

Per permettere, poi, uno sviluppo agile o del tutto programmatico delle interfacce grafiche per la definizione degli esercizi, si potrebbe beneficiare dall'avere una tabella nel database del T5 che contenga i tipi di obiettivi, descritti in formato JSON. Ogni tipo di obiettivo avrebbe un nome e una descrizione dei campi con tipo che devono essere compilati (come, ad esempio, il nome della classe e il punteggio).

Type	Name	Fields
1	ClassScoreGoal	{"ClassName": "String", "Score": "Integer"}

Ad esempio, se nel JSON viene definito un obiettivo di tipo "ClassScoreGoal" con i campi "Class Name" e "Score", la pagina per creare un nuovo esercizio potrebbe generare automaticamente due campi dove l'amministratore può inserire il nome della classe e il punteggio. Una volta che l'amministratore ha compilato i campi e inviato il modulo, il sistema genererebbe un file JSON con i dati inseriti (in questo esempio, il nome della classe e il punteggio), che poi verrebbe utilizzato dalle API esistenti per elaborare l'obiettivo.

Un altro aspetto su cui si potrebbe lavorare è la performance della pagina web `team_dashboard`, che risulta un po' lenta a causa del numero elevato di richieste necessarie. Sono stati adottati approcci per minimizzare questo problema, come ad esempio memorizzare i dati in variabili globali e riutilizzarli finché non è necessario aggiornarli. Tuttavia, la naturale distribuzione del task assegnato influisce ancora sulla velocità.