UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Documentazione Task R5: Features Admin

Anno Accademico 2024/2025

Team R5: Castaldo Giuseppe e Bellotti Carmine

Matr. [M63001741 – M63001742]
Email: [**giuseppe.castaldo22@studenti.unina.it – ca.bellotti@studenti.unina.it**]
Url Repository Github: **[https://github.com/Giuleppe09/A13]**

Sommario

Documentazione Task R5: Features Admin 1
Introduzione: Punto di partenza e Obiettivi del Progetto 5
Descrizione della Versione di Partenza 6
Use Case 6
Visione ad Alto livello del Sistema 7
Architettura di Riferimento 7
Nuovi requisiti Assegnati 8
Processo di Sviluppo Adottato 8
Strumenti Software di Supporto 9
Tecnologie Software 9
Analisi dei Requisiti Assegnati 10
Specifiche di Progetto**:** Utenti del Sistema Interessati 10
Requisiti Funzionali 10
User Stories 10
Tabella dei Requisiti funzionali 11
Criteri di Accettazione 13
Requisiti Non Funzionali 15
Vista ad Alto Livello del Sistema – Context Diagram 16
Diagrammi per la fase di Analisi 17
Use Case Diagram 17
Activity Diagram 18
Activity Diagram : Crea Team 18
Activity Diagram: Elimina Team 19
Activity Diagram: Modifica Nome Team 19
Activity Diagram: Aggiungi Studenti ad un Team 20
Activity Diagram: Rimuovi Studente da un Team 20
Activity Diagram: Visualizza Informazioni Team 21
Activity Diagram: Visualizza Informazioni di uno Studente di un Team 21
Activity Diagram : Creazione Assignment 22
Activity Diagram: Eliminazione Assignment 22
Modello dei dati 23
Analisi dell'Impatto dei Requisiti Assegnati sul Progetto Esistente 24
Progettazione della soluzione 25
Pattern Architetturale MVC 25
Motivazione delle Scelte di Progetto 28
Logica di Creazione dei Team 28

Logica di Aggiunta degli Studenti al Team 28
Logica di Visualizzazione degli Studenti nei Team 29
Component Diagram 30
Descrizione delle nuove RestAPI 31
Package Diagram 34
Model 34
View 34
Controller 35
Service 35
Application Class Diagram 36
Diagrammi per la dinamica dell'architettura 37
Sequence Diagram 37
Sequence Diagram: creaTeam 38
Sequence Diagram: deleteTeam 39
Sequence Diagram: modificaNomeTeam 40
Sequence Diagram: aggiungiStudenti 41
Sequence Diagram: rimuoviStudenteTeam 42
Sequence Diagram: visualizzaTeams 43
Sequence Diagram: cercaTeam 44
Sequence Diagram: ottieniStudentiTeam 45
Sequence Diagram: creaAssignment 46
Sequence Diagram: visualizzaTeamAssignments 47
Sequence Diagram: visualizzaAssignments 48
Sequence Diagram: deleteAssignment 49
Sequence Diagram: elencoClassiUT 50
Sequence Diagram: searchStudents 51
Problematiche riscontrate 52
Motivazioni del problema 52
Conseguenze operative 52
Approccio proposto per la risoluzione 53
Altre problematiche 53
Deployment Diagram 54
Testing 55
Test Suite per le API REST 55
Piano di Test: creaTeam 56
Piano di Test: deleteTeam 56
Piano di Test: modificaNomeTeam 57

Piano di Test: aggiungiStudentiTeam 57
Piano di Test: rimuoviStudenteTeam 58
Piano di Test: creaAssignment 58
Piano di Test: visualizzaTeamAssignments 59
Piano di Test: deleteAssignment 59
Piano di Test: searchStudents 60
Testing con Postman 61
Testing creaTeam 62
Testing deleteTeam 64
Testing modificaNomeTeam 65
Testing aggiungiStudentiTeam 67
Testing: rimuoviStudenteTeam 68
Testing: creaAssignment 69
Testing: visualizzaTeamAssignments 70
Testing: deleteAssignment 71
Testing: searchStudents 72

Introduzione: Punto di partenza e Obiettivi del Progetto

Questo documento presenta il lavoro svolto nell'ambito del progetto di Software Architecture Design, partendo dall'analisi e dall'implementazione di una versione migliorata del repository A13.

Il progetto originale, disponibile al seguente indirizzo A13 - versione ufficiale, è stato modificato per rispettare pienamente il pattern architetturale Model-View-Controller (MVC), che nella documentazione ufficiale non è stato implementato in modo coerente.

Il nostro focus si è concentrato sul task **R5 – Features Amministratore**, il quale ha richiesto la progettazione e lo sviluppo di funzionalità avanzate per la gestione dei **Team** e il monitoraggio delle performance degli studenti.

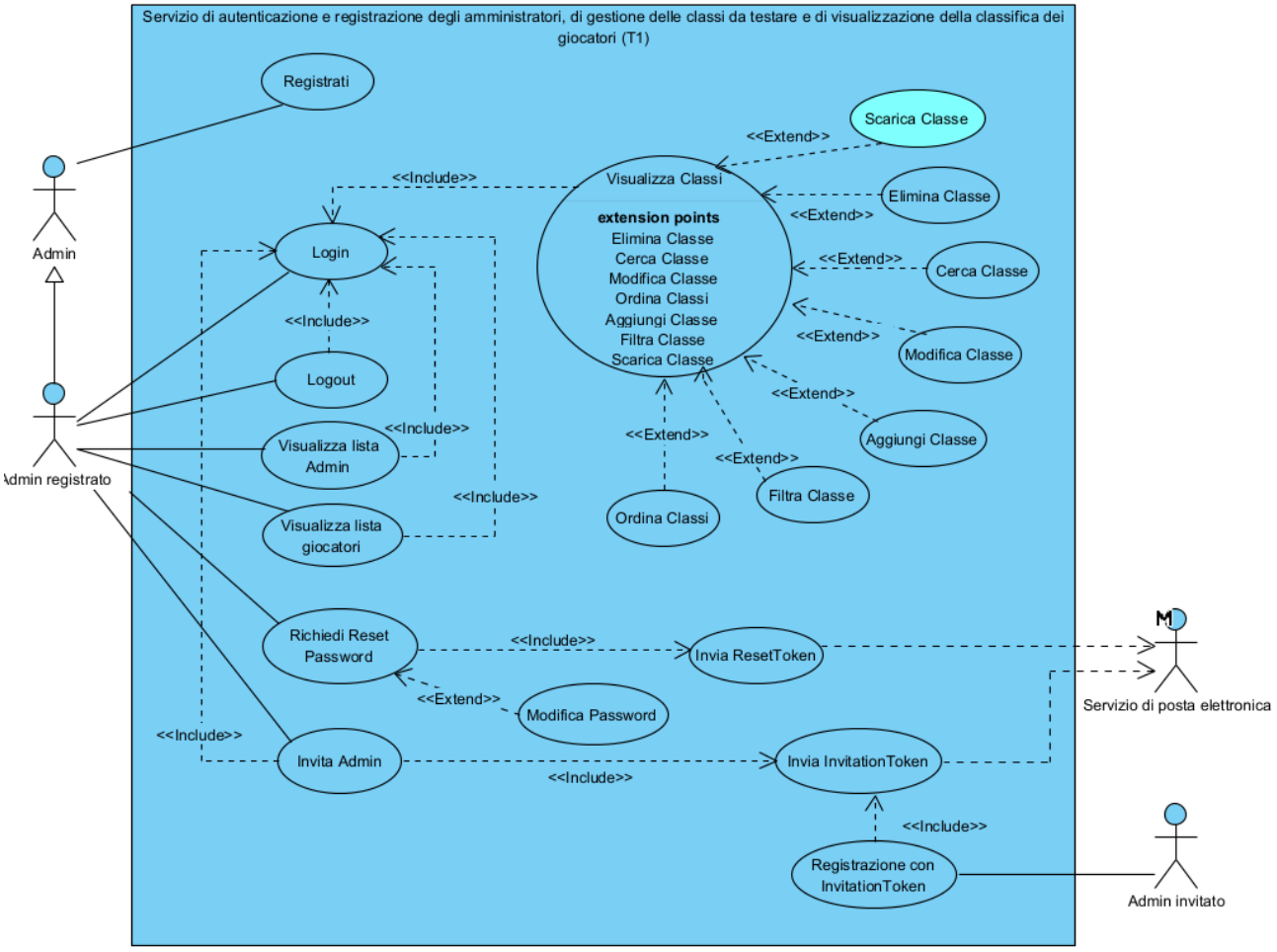
Questa documentazione approfondisce il processo decisionale, le soluzioni adottate e gli strumenti utilizzati per progettare un'architettura software robusta, scalabile e conforme ai requisiti richiesti, con particolare attenzione al pattern MVC e alle sue implicazioni nel contesto del progetto.

Descrizione della Versione di Partenza

Come accennato in precedenza, il progetto è stato avviato partendo da una repository che includeva un refactoring completo del componente principale oggetto del nostro lavoro, identificato come **T1 – ManVsClass**.

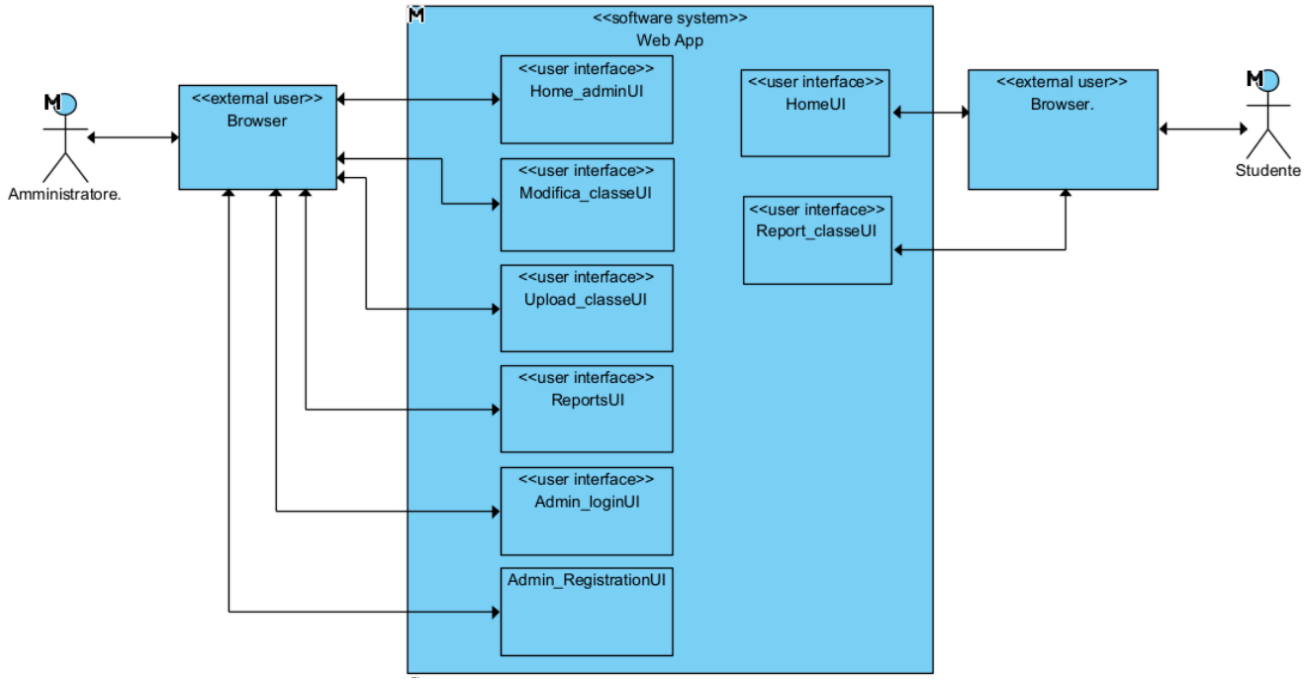
Use Case

I casi d'uso che erano già stati implementati dal **Team precedente** sono descritti dal seguente *Use Case Diagram*:



Visione ad Alto livello del Sistema

Il contesto iniziale del sistema è illustrato dal seguente *Context Diagram*:



Architettura di Riferimento

La repository di partenza contiene modifiche sostanziali rispetto alla documentazione ufficiale, in quanto il team precedente ha eseguito un refactoring completo del componente **T1**, adottando il pattern architetturale **Model-View-Controller (MVC)**.

Di conseguenza, si è deciso di non presentare in questa sezione una descrizione dettagliata dell'architettura iniziale, ma sarà cura del nostro team fornire successivamente diagrammi esplicativi che illustrino le modifiche apportate e la nuova configurazione del sistema.

Nuovi requisiti Assegnati

In seguito all'analisi della versione iniziale del sistema, sono stati assegnati nuovi requisiti per ampliare le funzionalità esistenti e migliorare l'interazione con gli utenti.

In particolare, le principali funzionalità richieste includono:

- **Creazione e gestione delle classi di studenti:** l'amministratore può definire e configurare nuove classi di studenti.
- **Assegnazione di sfide o compiti a tempo:** l'amministratore è in grado di distribuire attività agli studenti, con limiti temporali definiti.
- **Analisi dei risultati attraverso una dashboard interattiva:** fornisce metriche utili, come:
    - Tempo medio necessario per completare una sfida.
    - Percentuale media di copertura raggiunta dagli studenti.
    - Altre statistiche rilevanti per l'analisi delle prestazioni.

Processo di Sviluppo Adottato

Il progetto è stato sviluppato seguendo un processo iterativo, con una sequenza di iterazioni prefissate, come previsto dal corso. Ogni **settimana** sono stati presentati progressi e obiettivi tramite presentazioni PowerPoint esplicative, assicurando un monitoraggio costante delle attività svolte.

Per garantire un'organizzazione chiara e una progressione strutturata, il lavoro è stato suddiviso in tre iterazioni principali, ciascuna con obiettivi specifici:

Iterazione	**Attività Principali**
Prima Iterazione	- Analisi dettagliata dei requisiti. - Studio e test del codice sorgente fornito. - Approfondimento del framework Spring MVC. - Creazione di un *Use Case Diagram* per rappresentare i flussi funzionali del sistema. - Elaborazione di un *Class Diagram* per descrivere le entità presenti nel container T1.
Seconda Iterazione	- Sviluppo del back-end per la gestione dei team. - Implementazione del front-end per la gestione dei team. - Sviluppo del back-end per la gestione dei singoli team. - Refactoring del front-end della Home Admin. - Aggiornamento dei diagrammi UML realizzati nella prima iterazione. - Creazione di nuovi diagrammi necessari per il progetto.
Terza Iterazione	- Aggiornamento della documentazione del progetto. - Sviluppo del front-end per la gestione del singolo team. - Implementazione della gestione degli *assignment*. - Sviluppo del front-end per la gestione degli *assignment*. - Testing e refactoring del back-end.

Strumenti Software di Supporto

Durante lo sviluppo del progetto, sono stati utilizzati diversi strumenti software per supportare le diverse fasi del lavoro, dalla progettazione alla gestione del codice, fino ai test e alla documentazione.
Questi strumenti hanno permesso di ottimizzare il flusso di lavoro, garantire la qualità del codice e facilitare la collaborazione tra i membri del team. Di seguito sono elencati gli strumenti principali utilizzati nel progetto.

- **GitHub**: Utilizzato per il controllo delle versioni e la collaborazione tra il team, facilitando la gestione del codice sorgente.
- **VisualParadigm**, **PowerPoint** e **PlantUML**: Impiegati per la progettazione e la documentazione visiva dei diagrammi, come quelli dei casi d'uso e dei diagrammi UML.
- **Visual Studio Code**: Ambiente di sviluppo principale, scelto per la sua versatilità e le potenti estensioni per diversi linguaggi di programmazione.
- **Docker**: Utilizzato per creare ambienti di sviluppo isolati, facilitando la gestione delle dipendenze e la configurazione del sistema.
- **Postman**: Strumento impiegato per testare e documentare le API, garantendo che tutte le interazioni con i servizi web fossero correttamente implementate.

Tecnologie Software

Il sistema è stato sviluppato/ampliato adottando:

- **Back-End in Java con Spring Boot**: Il lato server dell'applicazione è stato sviluppato interamente in Java, sfruttando il framework Spring Boot per facilitare la gestione delle richieste, l'integrazione con il database e la creazione di un'architettura scalabile e manutenibile.

- **Database MongoDB**: Per la gestione dei dati, era già stato scelto MongoDB, un database NoSQL che permette di gestire facilmente grandi volumi di dati non strutturati e offre una buona scalabilità orizzontale.

- **Front-End in HTML, CSS e JavaScript**: Il Front-End è stato realizzato utilizzando HTML per la struttura della pagina, CSS per lo stile e JavaScript per l'interattività, creando così un'interfaccia utente moderna e responsiva.

Analisi dei Requisiti Assegnati

Specifiche di Progetto: Utenti del Sistema Interessati

Le modifiche apportate al progetto mirano a soddisfare le esigenze di uno specifico gruppo di utenti: **gli amministratori del sistema**. Questo ruolo, cruciale per il funzionamento del sistema, è stato ampliato per offrire strumenti e funzionalità avanzate che permettano agli amministratori di assumere un ruolo più simile a quello di un professore.

Grazie alle nuove funzionalità, gli amministratori potranno:

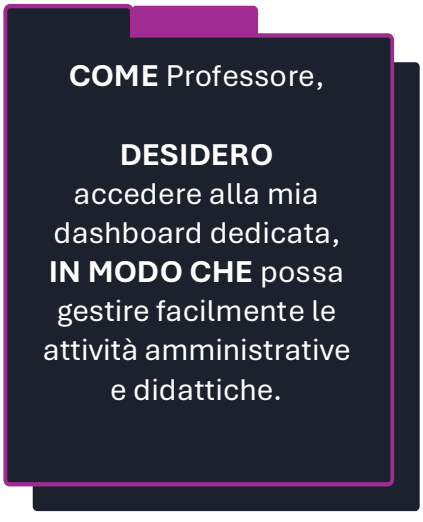
- **Gestire team di studenti:** creare, modificare e organizzare le classi di studenti, adattandole alle esigenze didattiche.
- **Assegnare compiti e sfide:** distribuire attività e sfide agli studenti con limiti temporali e monitorarne il completamento.
- **Analizzare le performance degli studenti:** accedere a una dashboard dettagliata che fornisce informazioni utili come il tempo medio di completamento delle sfide e la percentuale media di copertura.

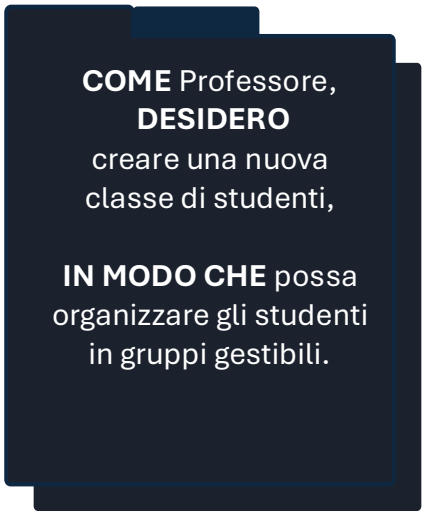
Requisiti Funzionali

I requisiti funzionali definiscono le capacità e i comportamenti del sistema per soddisfare le esigenze degli utenti finali.

User Stories

La raccolta e l'analisi dei requisiti si sono basate sulle ***user stories*** identificate, da cui sono stati derivati requisiti chiari e criteri di accettazione.





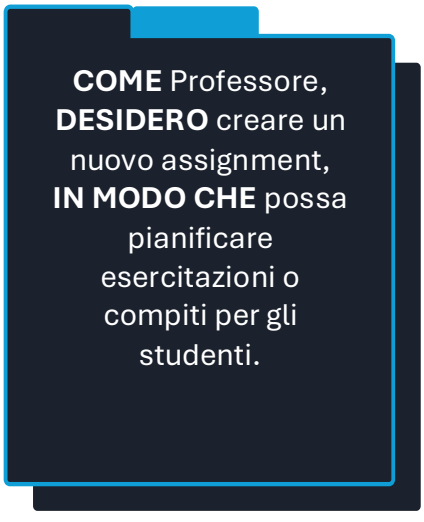


Tabella dei Requisiti funzionali

I requisiti funzionali derivati dalle *user stories* sono:

ID Requisito	Descrizione
R1	L'Amministratore-Professore deve poter accedere alla propria dashboard dedicata.
R2	L'Amministratore-Professore deve poter creare un Team di studenti.
R3	L'Amministratore-Professore deve poter modificare il nome di un Team.
R4	L'Amministratore-Professore deve poter aggiungere studenti a un Team.
R5	L'Amministratore-Professore deve poter rimuovere studenti da un Team.
R6	L'Amministratore-Professore deve poter eliminare un Team.
R7	L'Amministratore-Professore deve poter creare un nuovo assignment e assegnarlo direttamente ad un Team.
R8	L'Amministratore-Professore deve poter eliminare un assignment.
R9	L'Amministratore-Professore deve poter visualizzare gli assignment precedentemente assegnati.
R10	L'Amministratore-Professore deve poter visualizzare gli assignment relativi ad un Team.
R11 *	L'Amministratore-Professore deve poter accedere ai risultati di un Team relativi a un assignment.
R12 *	L'Amministratore-Professore deve poter accedere ai risultati dettagliati di un singolo studente per un assignment assegnato.

NB. I requisiti R11* e R12* non sono stati affrontati durante lo sviluppo per problematiche riscontrate durante la fase di sviluppo, come concordato con la Professoressa.
Guarda paragrafo delle Problematiche riscontrate.

Glossario

Il glossario fornisce una spiegazione concisa dei termini principali utilizzati nel sistema, per garantire una comprensione chiara e uniforme dei concetti chiave.

Studente	Utente registrato che fa parte di un Team e riceve assignment dall'amministratore.
Professore/Amministratore	Utente con privilegi avanzati, responsabile della gestione di Team, studenti e assignment.
Challenge/Sfida	Un'attività specifica proposta agli studenti come parte di un assignment, da completare in un tempo definito.
Dashboard Amministrativa	Interfaccia principale che consente al professore di accedere e gestire tutte le funzionalità del sistema.
Report dei Risultati	Panoramica o dettaglio delle performance degli studenti su un assignment, visualizzabile tramite la dashboard.
Assignment	Messaggio che verrà inviato agli **studenti** di un Team che conterrà il nome della classe di test da giocare.
Team	Classe di studenti, ovvero utenti registrati che sono afferenti ad un Admin che crea il Team.

Criteri di Accettazione

I criteri di accettazione definiscono le condizioni che ogni funzionalità del sistema deve soddisfare per essere considerata completa e conforme ai requisiti. Questi criteri stabiliscono scenari specifici basati sul comportamento atteso, descrivendo il contesto iniziale (**GIVEN**), l'evento scatenante (**WHEN**) e il risultato previsto (**THEN**). Sono progettati per garantire chiarezza, verificabilità e allineamento con le esigenze degli utenti finali, fungendo da guida per lo sviluppo e i test.

R1: Accesso alla dashboard dedicata.

- GIVEN un Amministratore-Professore autenticato,
- WHEN l'Amministratore-Professore accede al sistema,
- THEN il sistema deve mostrare la dashboard con tutte le funzionalità disponibili, inclusi i Team, gli Assignment e i risultati.

R2: Creare un Team di studenti.

- GIVEN un Amministratore-Professore autenticato nella sua dashboard,
- WHEN l'Amministratore-Professore inserisce i dettagli di un nuovo team (nome del team, studenti, ecc.),
- THEN il sistema deve creare un nuovo team e associarlo all'Amministratore-Professore.

R3: Modificare il nome di un Team.

- GIVEN un Amministratore-Professore autenticato e un team esistente,
- WHEN l'Amministratore-Professore modifica il nome di un team,
- THEN il sistema deve aggiornare correttamente il nome del team nel sistema.

R4: Aggiungere studenti a un Team.

- GIVEN un Amministratore-Professore autenticato e un team esistente,
- WHEN l'Amministratore-Professore aggiunge uno o più studenti a un team,
- THEN il sistema deve associare correttamente gli studenti al team.

R5: Rimuovere studenti da un Team.

- GIVEN un Amministratore-Professore autenticato e un team esistente con studenti,
- WHEN l'Amministratore-Professore rimuove uno o più studenti da un team,
- THEN il sistema deve rimuovere correttamente gli studenti dal team.

R6: Eliminare un Team.

- GIVEN un Amministratore-Professore autenticato e un team esistente,
- WHEN l'Amministratore-Professore elimina un team,
- THEN il sistema deve eliminare correttamente il team e tutte le sue associazioni (ad esempio, studenti, assignment).

R7: Creare un nuovo assignment.

- GIVEN un Amministratore-Professore autenticato,
- WHEN l'Amministratore-Professore crea un nuovo assignment e lo assegna a un team,
- THEN il sistema deve associare correttamente l'assignemnt al team specificato.

R8: Eliminare un assignment.

- GIVEN un Amministratore-Professore autenticato e un assignment esistente,
- WHEN l'Amministratore-Professore elimina un assignment,
- THEN il sistema deve eliminare correttamente l'assignment dal sistema.

R9: Visualizzare gli assignment precedentemente assegnati.

- GIVEN un Amministratore-Professore autenticato,
- WHEN l'Amministratore-Professore accede alla sezione degli assignment,
- THEN il sistema deve mostrare un elenco di tutti gli assignment precedentemente assegnati.

R10: Visualizzare gli assignment relativi a un Team.

- GIVEN un Amministratore-Professore autenticato e un team esistente,
- WHEN l'Amministratore-Professore seleziona un team dalla sua dashboard,
- THEN il sistema deve mostrare tutti gli assignment assegnati a quel team.

R11: Accedere ai risultati di un Team relativi a un assignment.

- GIVEN un Amministratore-Professore autenticato e un assignment assegnato a un team,
- WHEN l'Amministratore-Professore accede ai risultati del team per quell'assignemnt,
- THEN il sistema deve mostrare i risultati (ad esempio, punteggi, feedback) per il team relativo a quell'assignemnt.

R12: Accedere ai risultati dettagliati di un singolo studente per un assignment.

- GIVEN un Amministratore-Professore autenticato e un assignment assegnato a un team con studenti,
- WHEN l'Amministratore-Professore seleziona un singolo studente all'interno di un team per un assignment,
- THEN il sistema deve mostrare i risultati dettagliati (ad esempio, punteggio, commenti) per quel singolo studente relativamente a quell'assignemnt.

Requisiti Non Funzionali

I requisiti non funzionali definiscono le caratteristiche qualitative del sistema, assicurandone un funzionamento efficace, sicuro e scalabile. Nel contesto del progetto, questi requisiti includono:

- **Usabilità**: La dashboard deve offrire un'interfaccia intuitiva e facile da utilizzare, adatta anche a utenti con competenze tecniche limitate, garantendo una navigazione fluida tra le funzionalità.
- **Prestazioni**: Il sistema deve rispondere rapidamente alle richieste, mantenendo tempi di latenza minimi anche in presenza di un elevato numero di utenti e dati.
- **Affidabilità**: Deve essere garantita la protezione dei dati e la continuità operativa del sistema, minimizzando il rischio di perdita di informazioni in caso di errori o guasti.
- **Sicurezza**: Tutte le interazioni con il sistema devono essere protette da accessi non autorizzati, con particolare attenzione ai dati sensibili di studenti e professori.
- **Scalabilità**: L'architettura del sistema deve essere in grado di gestire un numero crescente di Team, studenti e assignment senza degradare le prestazioni.
- **Manutenibilità**: Il codice sorgente deve essere progettato in modo modulare e leggibile, per facilitare futuri aggiornamenti e l'aggiunta di nuove funzionalità.

Vista ad Alto Livello del Sistema – Context Diagram

Il sistema è stato arricchito con nuove interfacce utente per garantire una gestione più completa e funzionale delle operazioni amministrative.

Accanto alle interfacce già esistenti, quali:

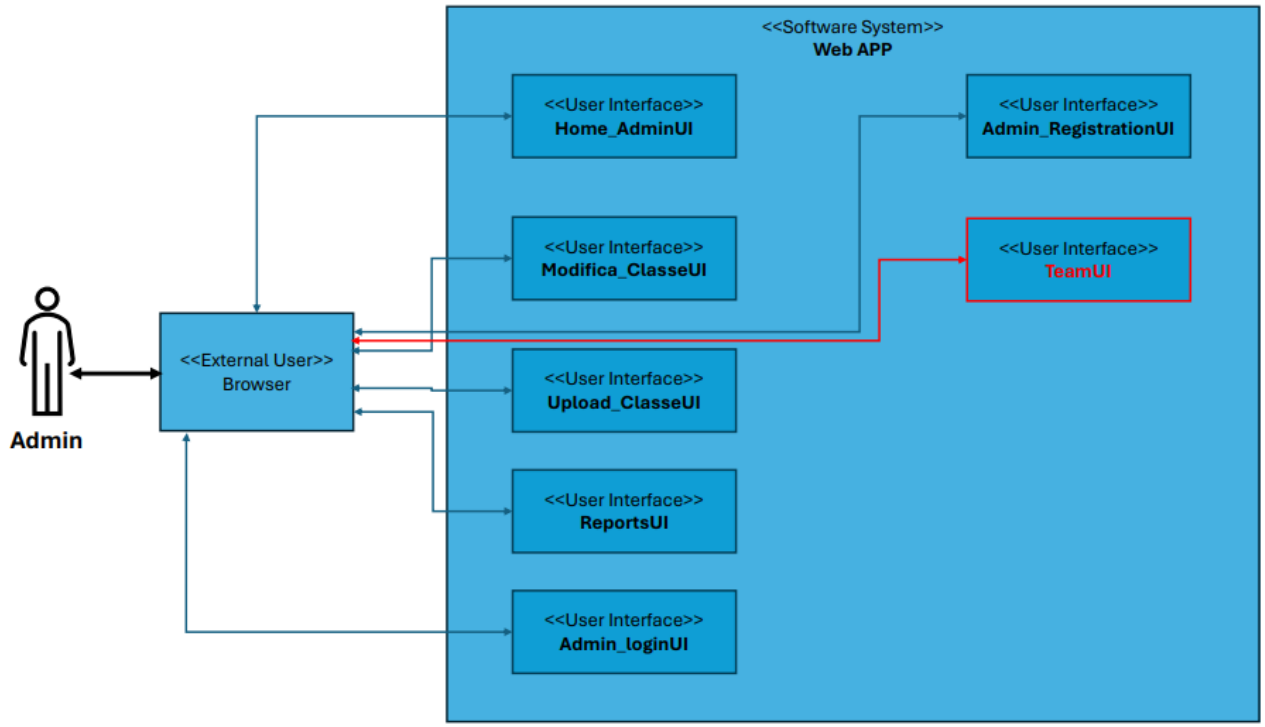
- **Registration_adminUI**
- **Admin_loginUI**
- **Home_AdminUI**
- **Upload_ClasseUI**
- **Modifica_ClasseUI**
- **Reports UI**,

Sono state aggiunte sezioni dedicate alla gestione avanzata dei Team, alla visualizzazione dettagliata di un singolo Team e alla gestione degli Assignment.

Questi aggiornamenti mirano a migliorare l'esperienza dell'amministratore, offrendo strumenti più intuitivi e accessibili, capaci di semplificare operazioni complesse.

È stato inoltre realizzato un restyling completo del **Front-End**, con un design più moderno e gradevole, che rende la navigazione tra le funzionalità del sistema più fluida e intuitiva. Questa nuova interfaccia favorisce l'accessibilità anche per utenti con competenze tecniche limitate, migliorando l'usabilità complessiva.

Abbiamo realizzato un **Context Diagram** che fornisce una rappresentazione visiva ad alto livello delle interazioni del sistema con l'ambiente esterno.



Diagrammi per la fase di Analisi

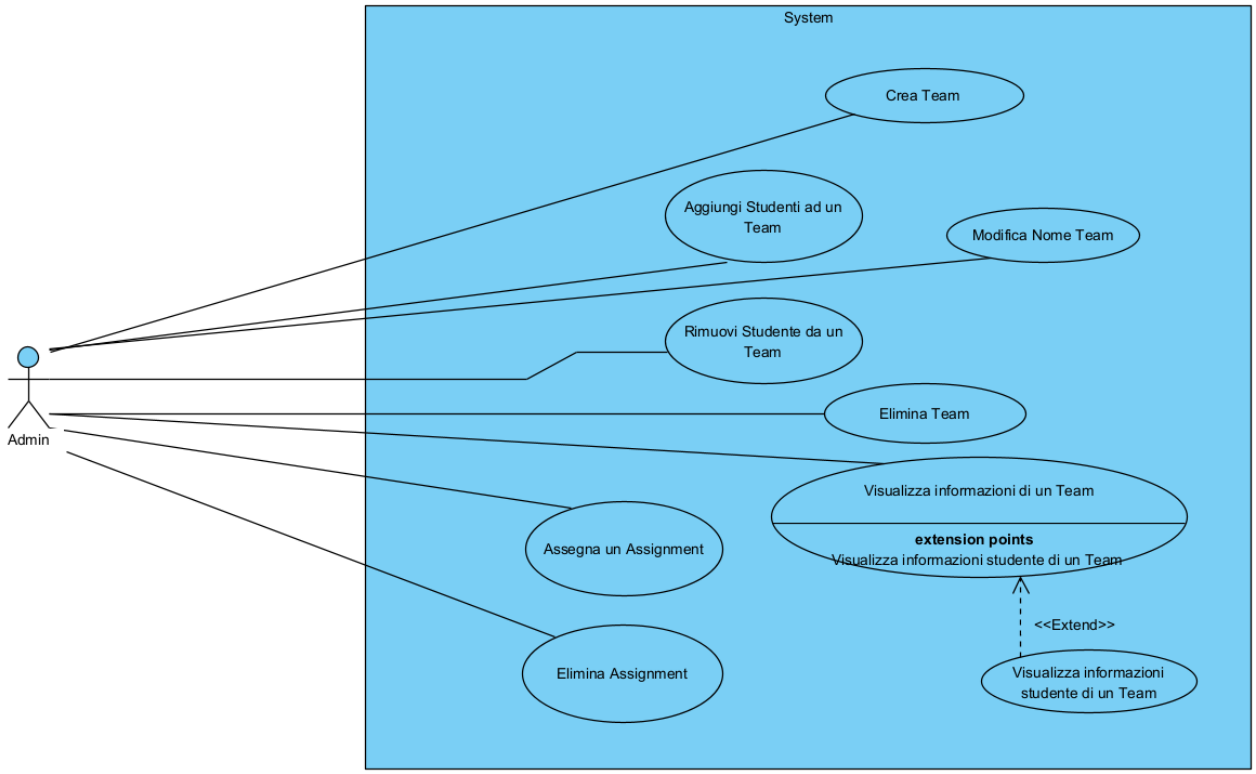
Questo capitolo documenta le attività di analisi condotte dal team per definire le funzionalità e i servizi del sistema.

Verranno presentati i diagrammi creati utilizzando la **notazione UML** (Unified Modeling Language).

Use Case Diagram

Il diagramma dei casi d'uso descrive le interazioni tra gli attori esterni e il sistema, mostrando le azioni che gli utenti possono compiere.

In particolare, sono elencati solo i casi d'uso derivanti dal **Task** assegnato.



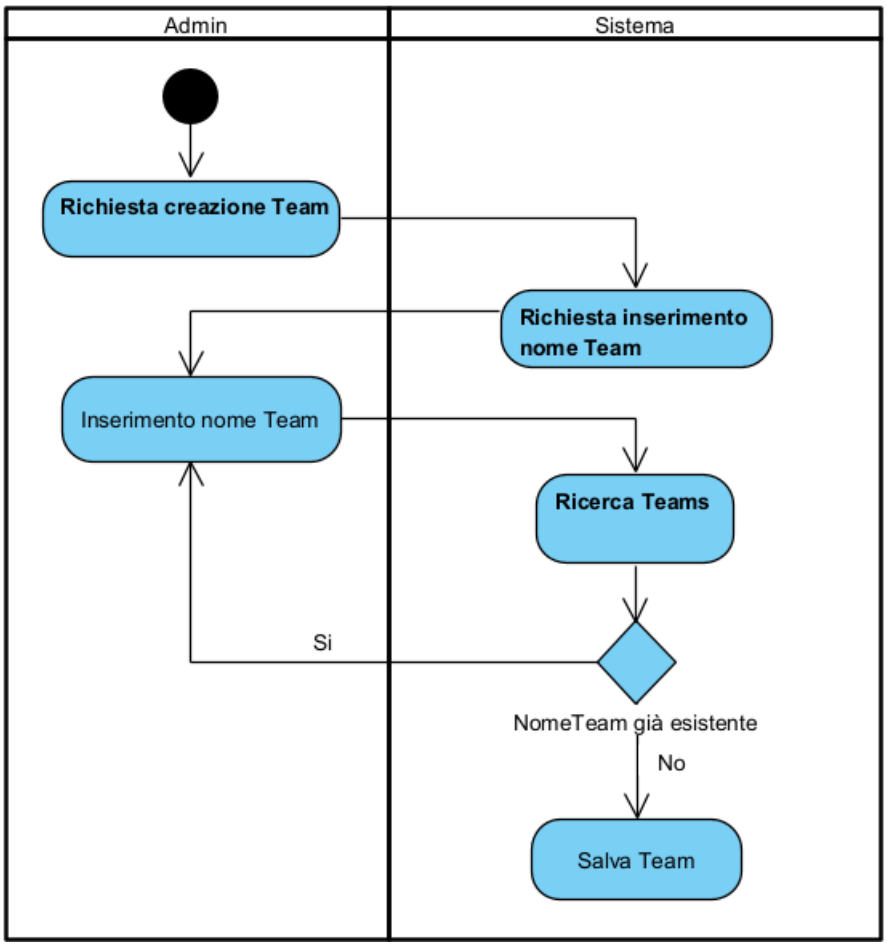
Activity Diagram

Gli Activity Diagram consentono la rappresentazione dei flussi di lavoro o delle attività che compongono un sistema software.

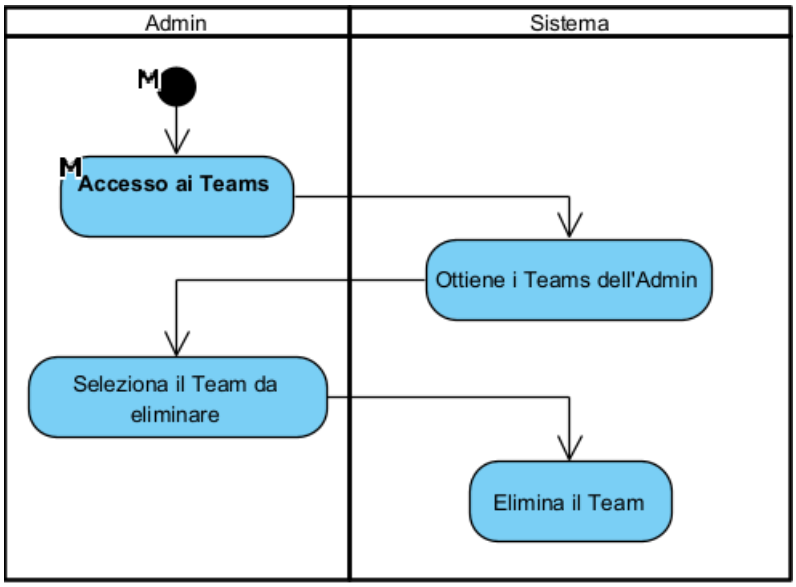
Questi diagrammi mettono in evidenza le attività, le decisioni e le transizioni, offrendo una panoramica chiara e ordinata della sequenza logica delle operazioni.

Nel presente lavoro, si parte dagli Activity Diagram per ottenere una visione d'insieme del processo complessivo. Questa prospettiva iniziale permette di identificare i principali flussi di attività e le decisioni critiche. Successivamente, nel capitolo dedicato alla progettazione, tali flussi saranno approfonditi attraverso i ***Sequence Diagram***, che si concentrano sui dettagli delle interazioni e degli scambi di messaggi tra gli attori e i componenti del sistema.

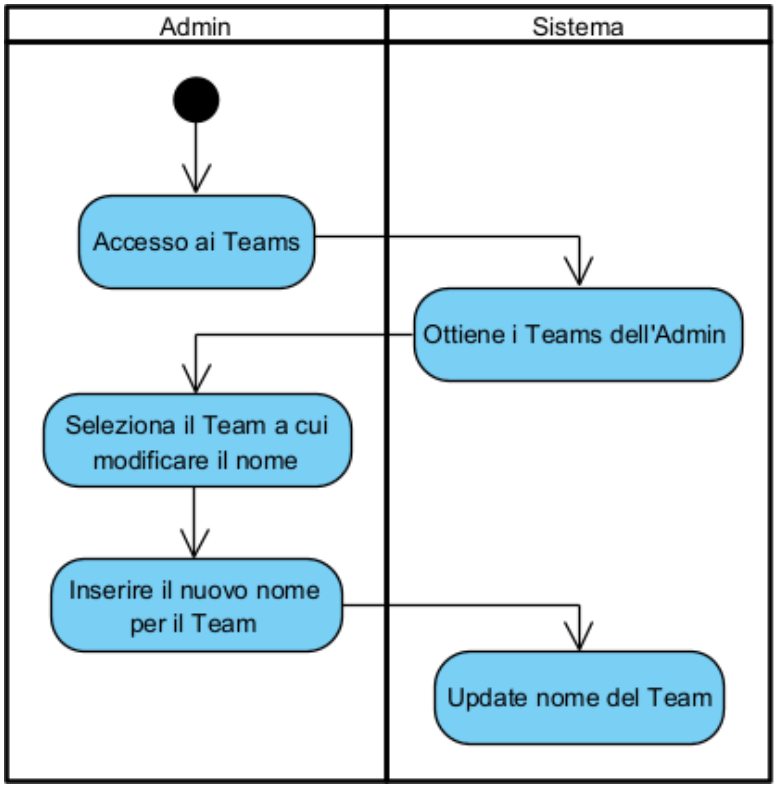
Activity Diagram : Crea Team



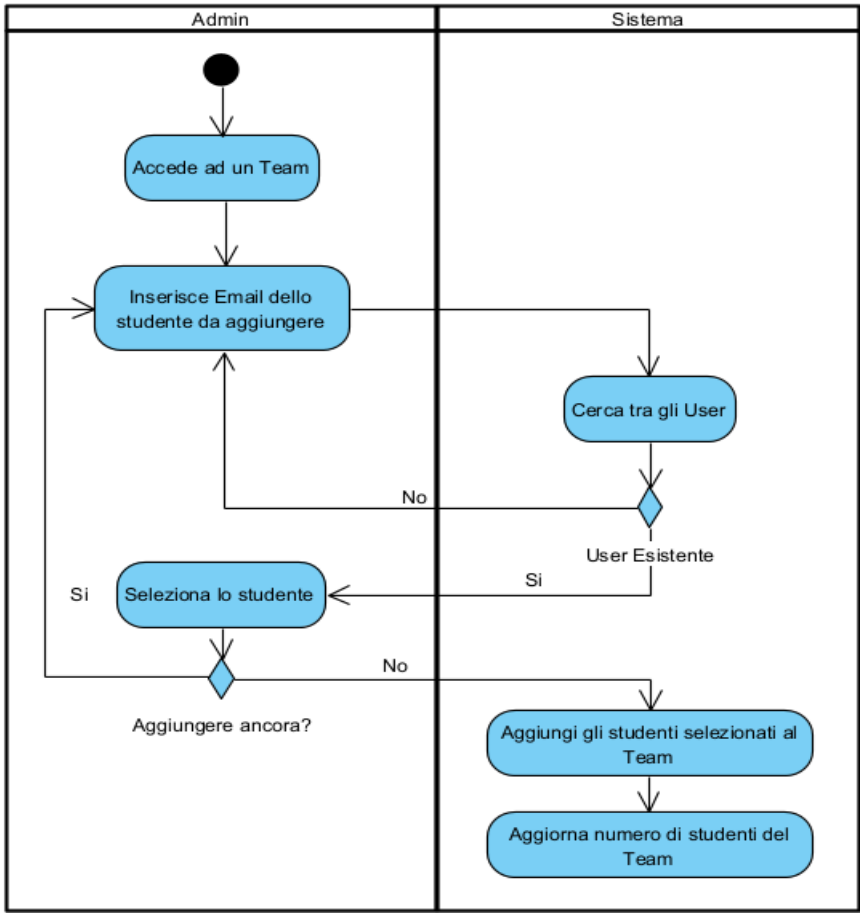
Activity Diagram: Elimina Team



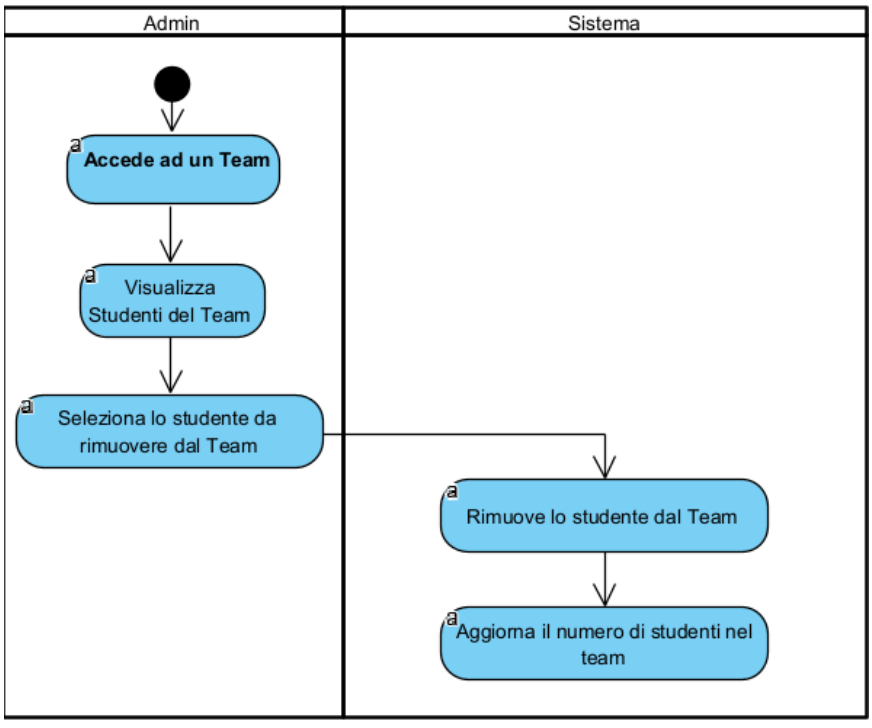
Activity Diagram: Modifica Nome Team



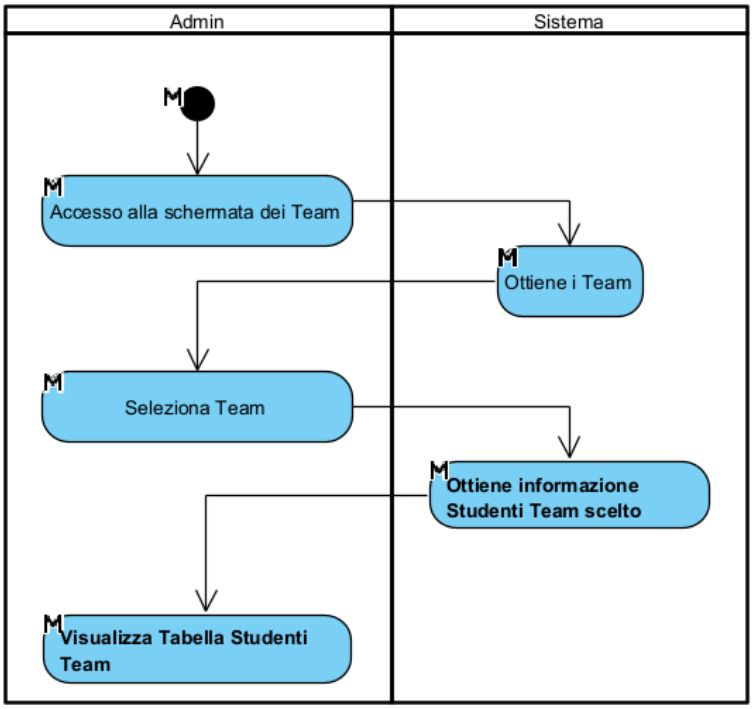
Activity Diagram: Aggiungi Studenti ad un Team



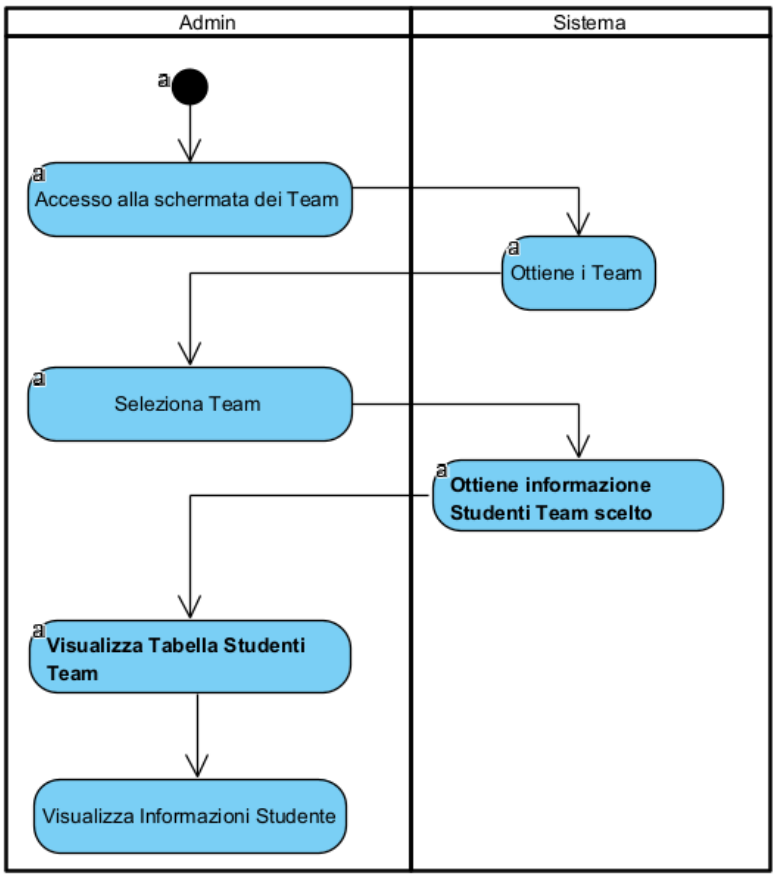
Activity Diagram: Rimuovi Studente da un Team



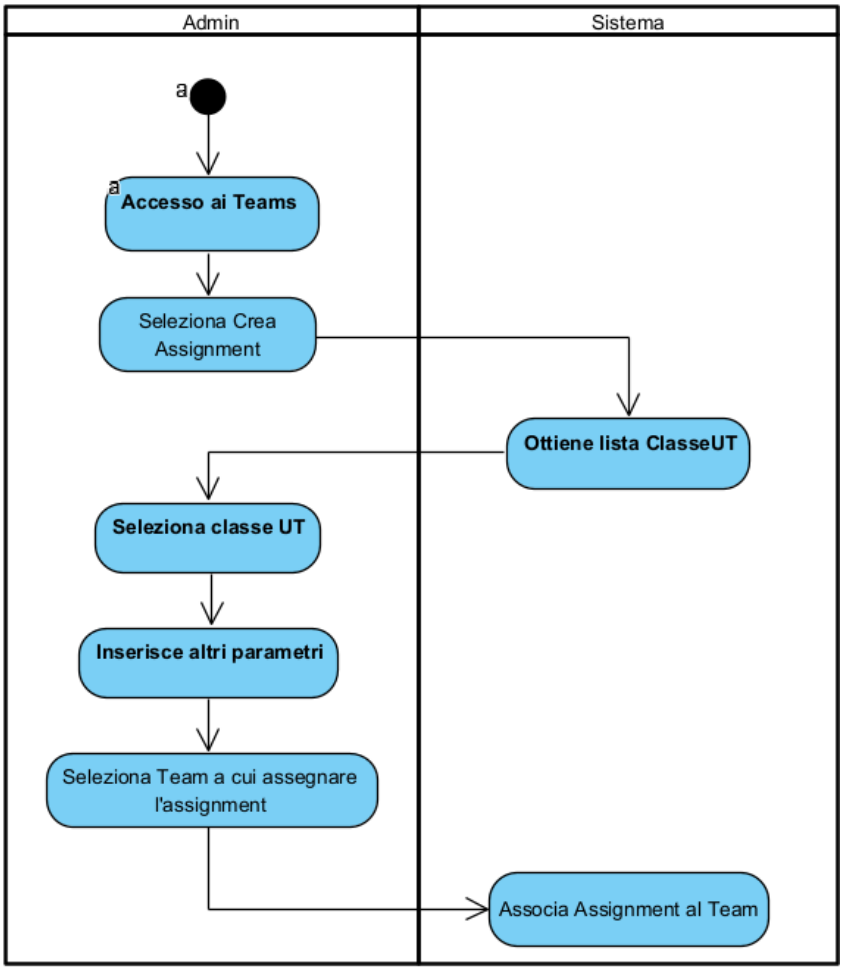
Activity Diagram: Visualizza Informazioni Team



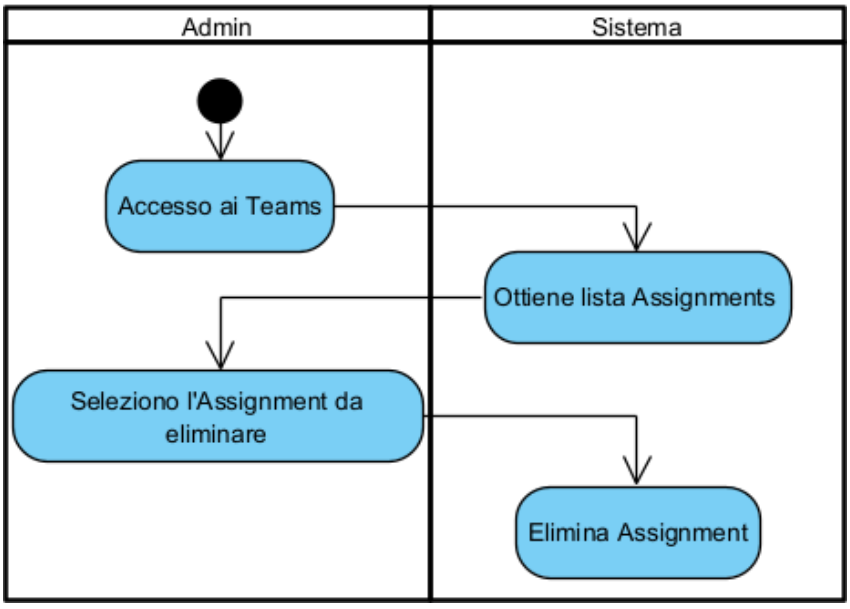
Activity Diagram: Visualizza Informazioni di uno Studente di un Team



Activity Diagram : Creazione Assignment



Activity Diagram: Eliminazione Assignment



Modello dei dati

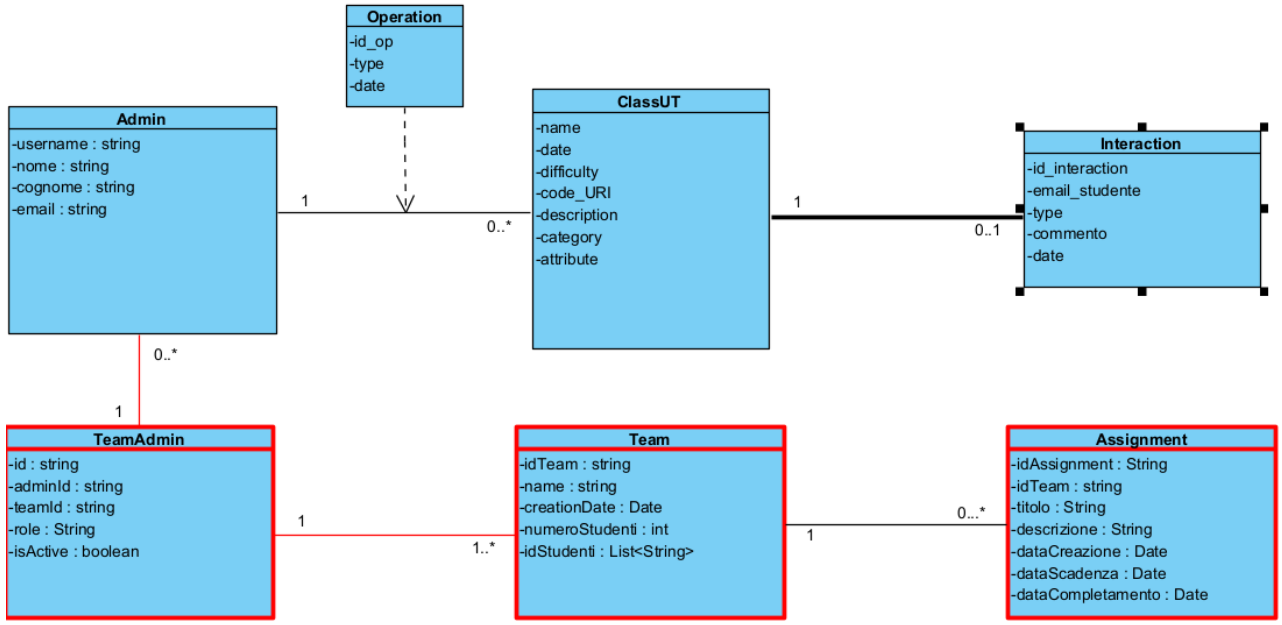
Nel modello dei dati è stata ampliata la struttura originaria con l'introduzione di nuove entità per introdurre la gestione dei **team**, nonché l'assegnazione di **compiti**.

Le nuove entità introdotte sono:

1. **Team**: funge da collezione centrale per raggruppare e organizzare gli utenti (studenti) in unità logiche.

2. **TeamAdmin**: Questa entità rappresenta l'associazione tra un team e il relativo amministratore.

3. **Assignment**:
   Questa entità è stata aggiunta per rappresentare i compiti o le attività assegnate ai membri di un team.



Relazioni

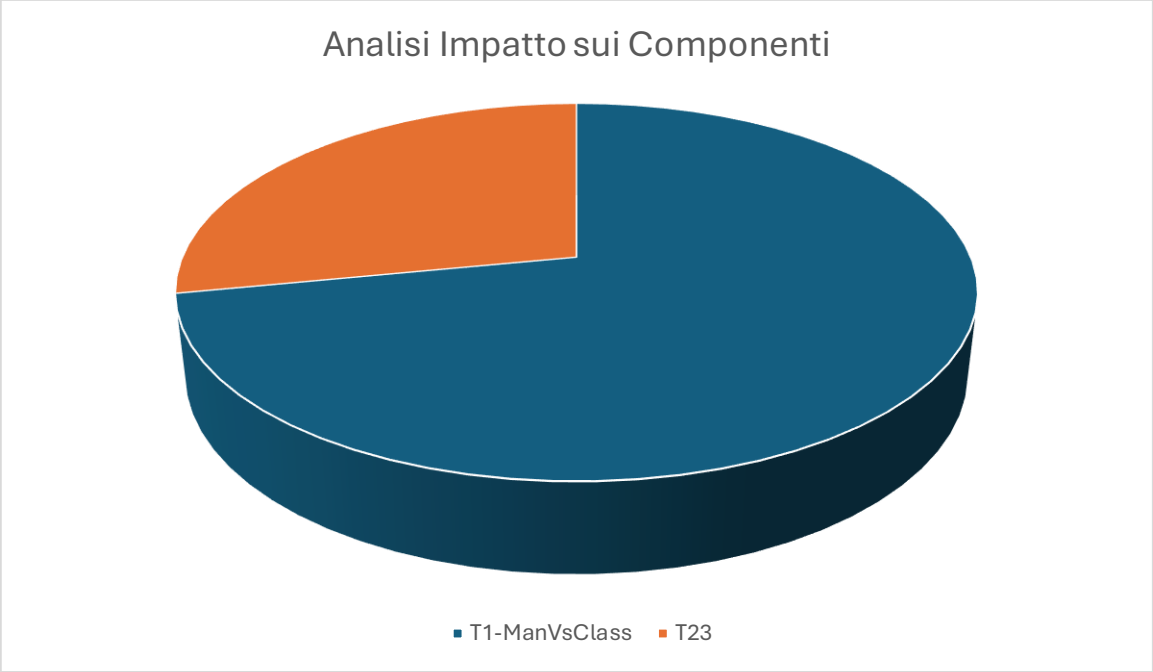
- La classe **Team** è collegata agli **studenti** mantenendo una **lista di id**.
- **TeamAdmin** stabilisce un'associazione specifica tra team e amministratori, garantendo che ogni Admin abbia un ruolo definito per la gestione.
- La classe **Assignment** è responsabile dell'associazione con il Team, dato che possiede l'ID del team a cui è stato assegnato.

Analisi dell’Impatto dei Requisiti Assegnati sul Progetto Esistente

L'implementazione dei nuovi requisiti assegnati avrà un impatto significativo principalmente sul componente **T1**.

In particolare, le modifiche riguarderanno l'architettura del T1, poiché dovranno essere introdotte funzionalità aggiuntive e che richiedono aggiornamenti nei modelli di dati, nei flussi di interazione e nella logica di business per gestire adeguatamente i nuovi processi.

Un impatto minore si avrà invece sul componente **T23**: sebbene le modifiche al T23 siano limitate, esse saranno cruciali per garantire che la dashboard del professore sia completa delle informazioni richieste.



Progettazione della soluzione

La fase di progettazione del sistema si è concentrata sul miglioramento delle soluzioni architetturali già esistenti e sull'integrazione di nuove funzionalità per rispondere in modo più completo ai requisiti.

Le scelte relative al pattern architetturale **MVC** e all'uso del framework **Spring MVC** erano già state effettuate nei cicli di sviluppo precedenti dai gruppi che avevano avviato il progetto. Il nostro contributo si è incentrato sull'ottimizzazione dell'implementazione esistente e sull'introduzione di nuove features per arricchire il sistema e migliorarne l'efficienza e la flessibilità.

Durante questa fase, sono stati analizzati e rivisti i diagrammi di progettazione, e sono state introdotte modifiche per perfezionare l'integrazione dei componenti e migliorare il flusso di dati. La documentazione aggiornata riflette queste modifiche, offrendo una visione completa delle scelte progettuali e delle loro motivazioni.

Pattern Architetturale MVC

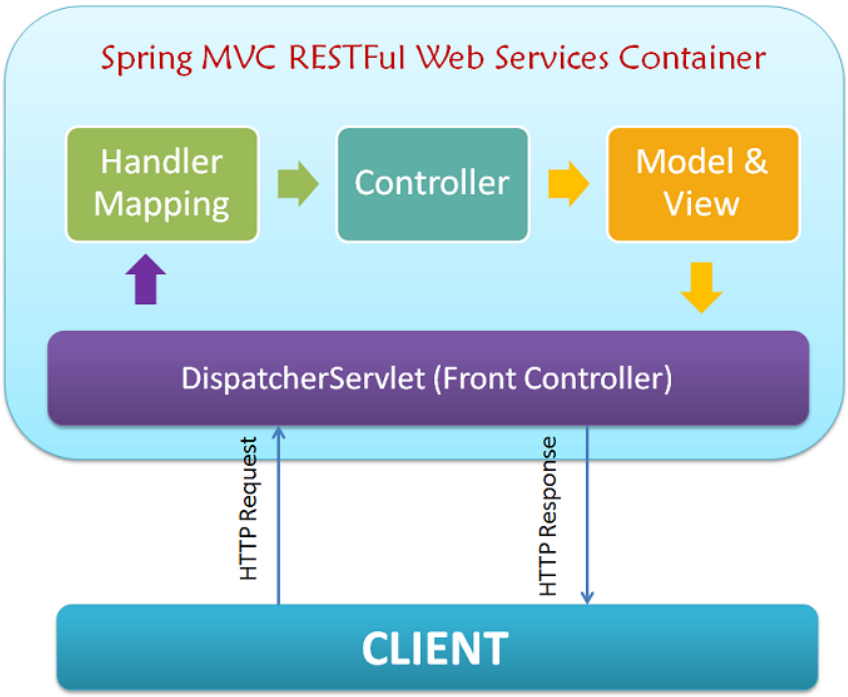
Il sistema adotta il pattern **MVC (Model-View-Controller)**, che suddivide logicamente il software in tre componenti principali:

- **Modello (Model):** si occupa della gestione dei dati e della logica applicativa.
- **Vista (View):** cura la presentazione dei dati e l'interfaccia utente.
- **Controllore (Controller):** funge da intermediario, coordinando l'interazione tra il modello e la vista in base alle richieste dell'utente.

L'adozione di questo pattern ha facilitato la separazione delle responsabilità, rendendo il codice più modulare e manutenibile. Le attività si sono concentrate sull'ottimizzazione del collegamento tra i tre livelli e sull'estensione delle funzionalità del sistema per garantire una maggiore flessibilità e scalabilità.

Spring MVC

Il framework **Spring MVC** è stato scelto nei cicli di sviluppo precedenti per implementare il pattern MVC e gestire l'architettura del sistema. Grazie a questo framework, è stato possibile realizzare un'architettura solida e scalabile, mantenendo una chiara separazione delle responsabilità tra le componenti.



Il nostro contributo su Spring MVC

Le attività principali svolte in questa fase hanno riguardato:

- **Miglioramento dell'implementazione:** Sono state ottimizzate le configurazioni esistenti e migliorate le performance di alcune funzionalità chiave, riducendo la complessità del codice e migliorandone la leggibilità.
- **Aggiunta di nuove funzionalità:** Sono state introdotte nuove feature, come la gestione avanzata delle richieste.

Vantaggi principali di Spring MVC

1. **Separazione delle responsabilità:**
   La chiara distinzione tra Modello, Vista e Controllore consente una gestione modulare e facilmente estendibile del codice, rendendo semplice l'aggiunta e la manutenzione di nuove funzionalità.
2. **Configurazione flessibile:**
   Grazie alle annotazioni e alla configurazione dichiarativa, il framework permette di adattare facilmente il sistema alle nuove esigenze, riducendo il rischio di errori e semplificando lo sviluppo.
3. **Scalabilità e performance:**
   Spring MVC fornisce strumenti avanzati per gestire le richieste concorrenti, ottimizzare le risorse e garantire performance elevate, rendendolo ideale per progetti con carichi di lavoro crescenti.
4. **Gestione delle richieste:**
   Il framework offre un'implementazione robusta per la mappatura delle richieste HTTP, che si è rivelata fondamentale per l'integrazione delle nuove feature introdotte in questa fase.

L'adozione di **Spring MVC** da parte dei gruppi precedenti ha fornito una solida base per lo sviluppo del sistema.

Motivazione delle Scelte di Progetto

Logica di Creazione dei Team

Durante la creazione di un Team, è necessario ottenere l'ID dell'Admin dalla sessione attiva, per consentire l'associazione del Team all'Admin stesso all'interno della collection *TeamAdmin*. Per realizzare ciò, abbiamo utilizzato il *JWT Token*, che include il subject rappresentato dall'Username dell'Admin.

A tal fine, è stato implementato un metodo dedicato nel *JWT Service*, che permette di estrarre l'Username dell'Admin direttamente dal Token. Questo Username viene quindi utilizzato nella rotta di creazione del Team, garantendo un collegamento sicuro e univoco tra l'Admin e il Team associato.

Logica di Aggiunta degli Studenti al Team

Dopo la creazione di un Team, è stato implementato un meccanismo che consente all'Admin di aggiungere più studenti contemporaneamente al Team, ottimizzando così le operazioni di scrittura sul database attraverso una singola query.

Per agevolare l'individuazione degli studenti da aggiungere, è stata integrata una rotta nel componente **T23** che permette la ricerca degli studenti tramite diversi criteri: email, nome, cognome, o una combinazione di essi. Questo approccio rende la funzionalità più user-friendly, consentendo all'Admin di effettuare una ricerca anche senza conoscere necessariamente l'email dello studente, ma basandosi su informazioni più generiche come il nome e il cognome.

Gli ID degli studenti selezionati durante la ricerca vengono raccolti in un array e inviati come payload unico al backend del container **T1**, dove viene processata la richiesta di aggiunta al Team. Questa strategia minimizza le interazioni con il database, garantendo un'operazione efficiente e altamente scalabile.

Inoltre, è stato implementato un sistema di notifiche automatiche via email. Ogni studente aggiunto a un Team riceve una comunicazione che lo informa del nuovo ruolo, migliorando così l'esperienza utente e la trasparenza del processo.

Logica di Visualizzazione degli Studenti nei Team

Per la visualizzazione degli studenti associati ai Team, abbiamo deciso di mantenere l'intera logica all'interno del container T1, in quanto l'accesso a tali informazioni avviene principalmente da parte degli Admin (professori) che gestiscono i Team.

La struttura del Team include una lista di ID degli studenti associati, evitando così la duplicazione dei dati degli *User* nel container T1. Quando è necessario recuperare le informazioni complete degli studenti, viene inviata una richiesta al container T23, che processa la lista di ID e restituisce tutti i dettagli richiesti. Questa soluzione garantisce una gestione centralizzata ed evita la ridondanza dei dati, mantenendo al contempo un sistema modulare e ben organizzato.

Logica Relativa agli Assignment

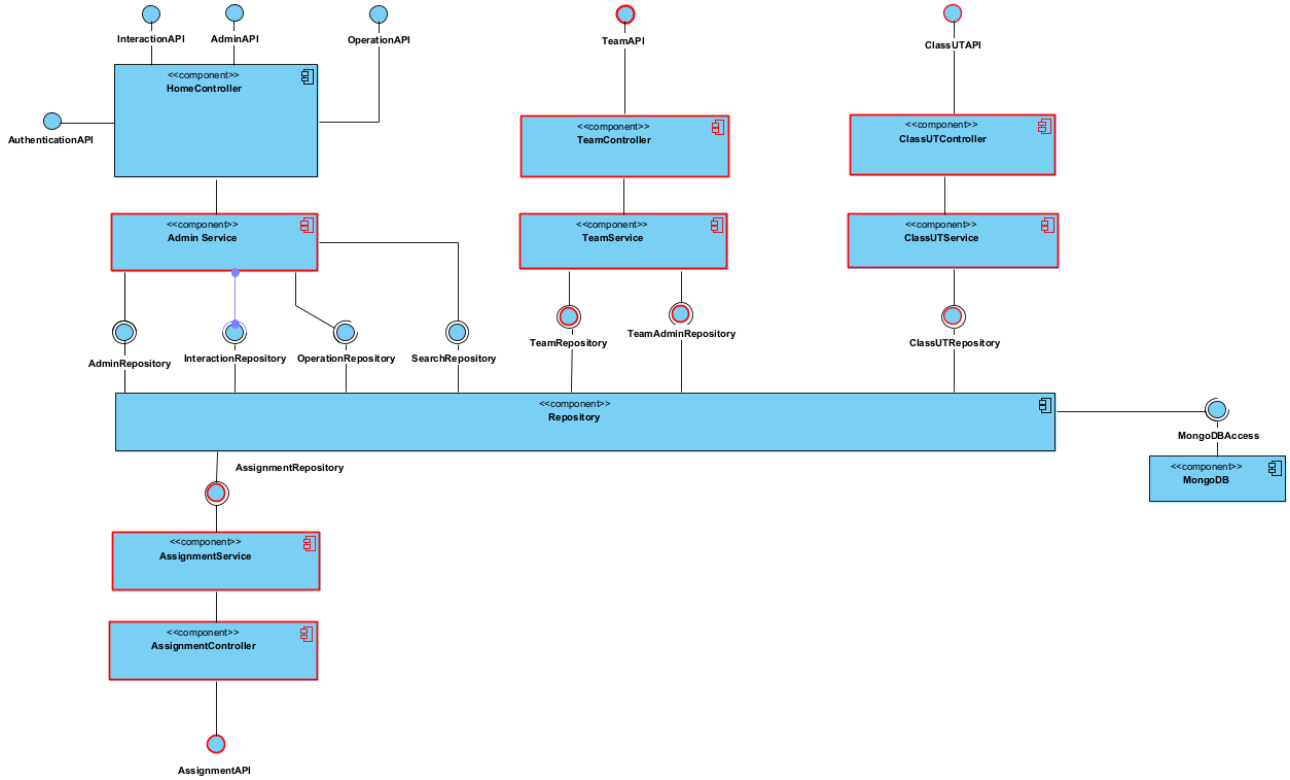
Inizialmente, la scelta del luogo in cui memorizzare gli *Assignment* è stata oggetto di valutazione approfondita. Considerando che un Assignment può essere interpretato come un "messaggio" destinato agli studenti, abbiamo deciso di memorizzare nel container T1 gli Assignment associati a ciascun Team, conservati in una collection dedicata all'interno del container T1.

Questa soluzione garantisce un accesso rapido e diretto ai dati, migliorando l'efficienza delle operazioni e la coerenza del sistema.

N.B. : al momento non è ancora possibile inviare una notifica agli studenti del Team ma nei prossimi sviluppi è prevista l'integrazione con il NotificationService.
Per sopperire a quest'ultimo al momento è implementato un email service che si occupa di inviare agli studenti di un team a cui è inviato il nuovo Assignment.
Questa funzionalità però è molto impattante sul sistema, rallentandolo e portando ad un decadimento dell' usabilità di quest'ultimo. Per questo motivo è al momento **commentata**.

Component Diagram

Il **diagramma dei componenti** illustra la struttura modulare del sistema, mettendo in evidenza i principali componenti software e le loro interazioni. Questo schema rappresenta un riferimento architetturale essenziale per comprendere come le varie parti del sistema collaborino per soddisfare i requisiti funzionali, garantendo modularità, manutenibilità e scalabilità.



Nel progetto attuale, abbiamo effettuato un refactoring per ottimizzare la gestione delle API e migliorare l'organizzazione delle responsabilità tra i diversi moduli. Le principali modifiche includono:

- **Gestione del Team:** Le funzionalità legate alla gestione dei team sono state trasferite dal HomeController al nuovo TeamController. A supporto, è stato introdotto il componente TeamService, che si connette alle interfacce esistenti nei repository (TeamRepository, TeamAdminRepository). Questo cambio evita il sovraccarico del HomeController.
- **Gestione degli Assignment:** Analogamente, la gestione degli assignment è stata riorganizzata introducendo il AssignmentController e il AssignmentService, che operano tramite il AssignmentRepository.
- **Funzionalità legate alle ClassUT:** Per affrontare il sovraccarico di API nel HomeController, è stato introdotto il ClassUTController e il relativo ClassUTService. Questi componenti si occupano di tutte le operazioni relative alle funzionalità delle classi, mantenendo il codice più modulare e separando le responsabilità.
- In più è stato introdotto l'**AdminService** responsabile della logica di business relativa alle API offerte dall'**Home Admin**.

Nella versione precedente del sistema (T1), erano presenti ulteriori interfacce e funzionalità. Tuttavia, in questa iterazione del progetto, alcune di queste funzionalità sono soggette a modifiche strutturali. Per evitare ambiguità, esse non sono state rappresentate nel diagramma attuale.

Il risultato complessivo è un'architettura più chiara e coerente, dove ogni controller è focalizzato su un insieme ben definito di responsabilità, con servizi dedicati per la logica applicativa e repository per la gestione dei dati.

Descrizione delle nuove RestAPI

Nel corso dello sviluppo del progetto, sono state progettate e implementate diverse nuove API per supportare le funzionalità avanzate del sistema. Queste API rappresentano i punti di interazione principali tra il client e il server, consentendo agli amministratori di gestire i Team, gli Assignment e gli studenti in modo efficiente.

L'elenco seguente fornisce una panoramica dettagliata delle nuove API intodotte in **T1** e **T23**, includendo il metodo HTTP, l'endpoint e una breve descrizione della loro funzione. Queste API sono progettate per essere intuitive, sicure e facilmente integrabili, garantendo un'esperienza d'uso ottimale per gli amministratori.

Endpoint nuovi in T1

Endpoint	Metodo HTTP	Descrizione	Input	Controller
GET /teams	GET	Mostra la pagina di gestione dei team.	Nessuno	HomeController
GET /visualizzaTeam/{idTeam}	GET	Visualizza i dettagli di un team specifico.	idTeam (PathVariable)	TeamController
GET /usernameAdmin	GET	Ottiene il nome utente dell'amministratore.	Token JWT (CookieValue)	HomeController
POST /creaTeam	POST	Crea un nuovo team.	team (RequestBody), Token JWT (CookieValue)	TeamController
DELETE /deleteTeam	DELETE	Elimina un team specificato.	idTeam (RequestBody), Token JWT (CookieValue)	TeamController
PUT /modificaNomeTeam	PUT	Modifica il nome di un team.	TeamModificationRequest (RequestBody), Token JWT (CookieValue)	TeamController
GET /visualizzaTeams	GET	Visualizza la lista di tutti i team.	Token JWT (CookieValue)	TeamController

Endpoint	Metodo HTTP	Descrizione	Input	Controller
GET /cercaTeam/{idTeam}	GET	Cerca un team specificato per ID.	idTeam (PathVariable), Token JWT (CookieValue)	TeamController
PUT /aggiungiStudenti/{idTeam}	PUT	Aggiunge studenti a un team specificato.	idTeam (PathVariable), Lista di idStudenti (RequestBody), Token JWT (CookieValue)	TeamController
GET /ottieniStudentiTeam/{idTeam}	GET	Ottiene la lista degli studenti di un team.	idTeam (PathVariable), Token JWT (CookieValue)	TeamController
PUT /rimuoviStudenteTeam/{idTeam}	PUT	Rimuove uno studente da un team specificato.	idTeam (PathVariable), idStudente (RequestBody), Token JWT (CookieValue)	TeamController
GET /elencoNomiClassiUT	GET	Ottiene l'elenco dei nomi delle classi UT.	Token JWT (CookieValue)	ClassUTController
POST /creaAssignment/{idTeam}	POST	Crea un nuovo assignment per un team specificato.	idTeam (PathVariable), assignment (RequestBody), Token JWT (CookieValue)	AssignmentController
GET /visualizzaTeamAssignments/{idTeam}	GET	Visualizza gli assignment di un team specificato.	idTeam (PathVariable), Token JWT (CookieValue)	AssignmentController
GET /visualizzaAssignments	GET	Visualizza tutti gli assignment.	Token JWT (CookieValue)	AssignmentController
DELETE /deleteAssignment/{idAssignment}	DELETE	Elimina un assignment specificato.	idAssignment (PathVariable), Token JWT (CookieValue)	AssignmentController

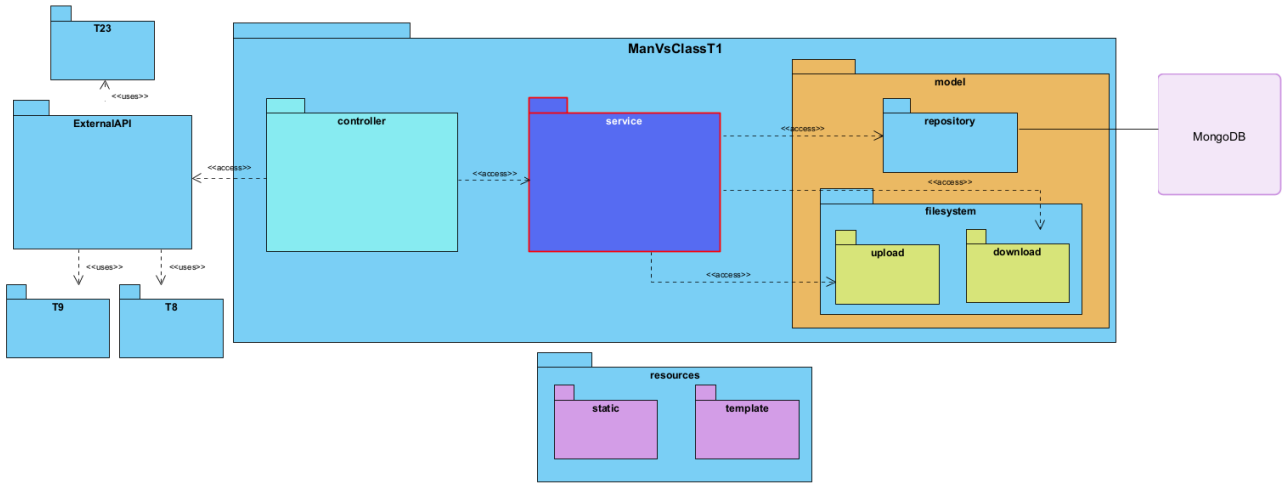
Endpoint nuovi in T23

Endpoint	Metodo HTTP	Descrizione	Input	Controller
POST /studentsByIds	POST	Ottiene la lista di utenti corrispondenti a una lista di ID.	Lista di idsStudenti (RequestBody)	Controller
GET /studentByEmail/{emailStudente}	GET	Ottiene l'utente corrispondente a un'email specifica.	emailStudente (PathVariable)	Controller
GET /studentsByNameSurname	GET	Ottiene una lista di utenti che corrispondono alle generalità fornite (nome e cognome).	request contenente name e surname (RequestBody)	Controller
POST /searchStudents	POST	Cerca studenti per generalità o email, combinando le funzionalità precedenti.	request contenente name, surname ed eventualmente email (RequestBody)	Controller

Package Diagram

Il *Package Diagram* è uno strumento utilizzato per rappresentare l'organizzazione e le **dipendenze** tra i package in un'applicazione software.

Fornisce una **panoramica strutturale**, evidenziando il raggruppamento dei package e le loro interazioni. Questa rappresentazione evidenzia il raggruppamento logico delle funzionalità e le interazioni tra i package, facilitando la comprensione dell'architettura del sistema.



Model

Il package *Model* contiene le classi principali del dominio e i metodi necessari per accedere e manipolare i dati dell'applicazione. Queste classi svolgono un ruolo chiave come ponte tra il *Controller* e la *View*, consentendo l'interazione con i dati e la loro presentazione.

- Il package include le classi principali del dominio.
- Contiene anche i sottopackage dedicati alla gestione dei dati:
  - **Repository**: si occupa dell'interazione con il database MongoDB, sfruttando le funzionalità offerte da MongoRepository di Spring.
  - **FileSystem**: fornisce supporto per il caricamento (*Upload*) e il download (*Download*) di file nella repository condivisa, includendo funzionalità per l'accesso e la gestione del file system.

View

In accordo con il pattern architetturale MVC, il package *View* è responsabile della presentazione dei dati e dell'interfaccia utente. È implementato all'interno del package *Resources*, che include file .html per la rappresentazione web dell'applicazione.

Controller

Il package *Controller* è responsabile della gestione delle richieste HTTP ricevute dai client. Coordina le operazioni tra il *Model* e la *View*, assicurandosi di elaborare le richieste e generare le risposte appropriate. Questo package rappresenta il cuore della logica di controllo dell'applicazione.
Nella nuova versione, per migliorare la modularità e ridurre il sovraccarico del precedente HomeController, sono stati introdotti tre nuovi controller:

- **TeamController**: gestisce tutte le operazioni relative ai team.
- **ClassUTController**: si occupa delle funzionalità legate alla gestione delle classi UT.
- **AssignmentController**: è dedicato alla gestione degli assignment.

Service

Una delle principali innovazioni introdotte nella nuova versione del sistema è il package *Service*. Questo package funge da intermediario tra il *Controller* e il *Repository*, centralizzando la logica applicativa. Isolando le operazioni di business logic, il package *Service* migliora significativamente la modularità, la manutenibilità e l'estensibilità del sistema.

Tra i nuovi servizi introdotti per ottimizzare la gestione delle funzionalità, troviamo:

- **TeamService**: si occupa della gestione delle operazioni relative ai team.
- **AssignmentService**: gestisce le funzionalità legate agli assignment.
- **ClassUTService**: dedicato alla gestione delle classi UT e delle relative operazioni.

Questa riorganizzazione garantisce una chiara separazione delle responsabilità e facilita l'aggiunta di nuove funzionalità, mantenendo una struttura del sistema più ordinata ed efficiente.

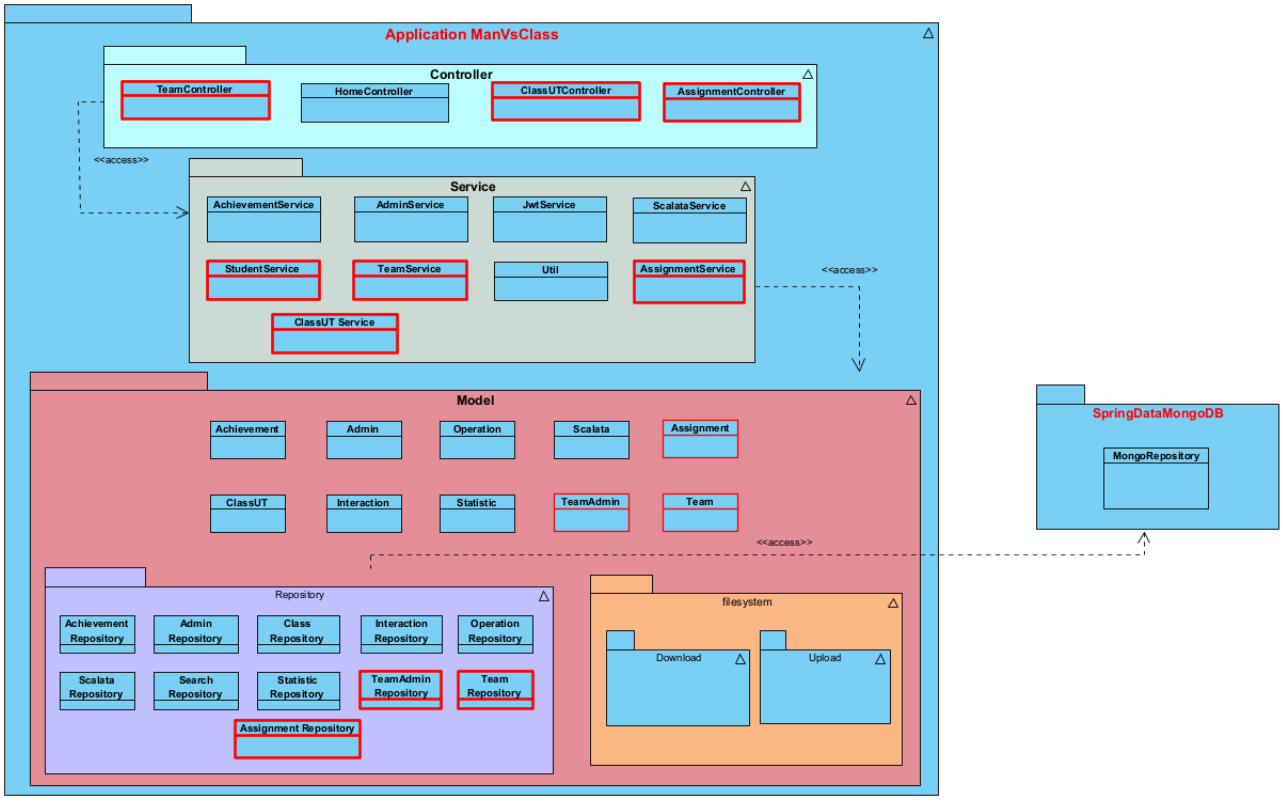
Application Class Diagram

L'**Application Class Diagram** è un diagramma che rappresenta la struttura delle classi di un'applicazione software, evidenziando le loro interazioni, le relazioni tra oggetti e come ciascuna classe contribuisce alla logica complessiva del sistema.

Questo tipo di diagramma è particolarmente utile per comprendere l'architettura e la progettazione a livello di codice di un'applicazione, fornendo una visione dettagliata delle classi, dei loro attributi, metodi e delle relazioni tra di esse.

Questo diagramma aiuta a:

1. **Visualizzare la struttura interna** dell'applicazione, fornendo una rappresentazione grafica della suddivisione del sistema in classi e package.
2. **Identificare le dipendenze** tra le varie classi.
3. **Analizzare il design** dell'applicazione, per verificare se le classi sono correttamente separate in termini di responsabilità.



Diagrammi per la dinamica dell’architettura

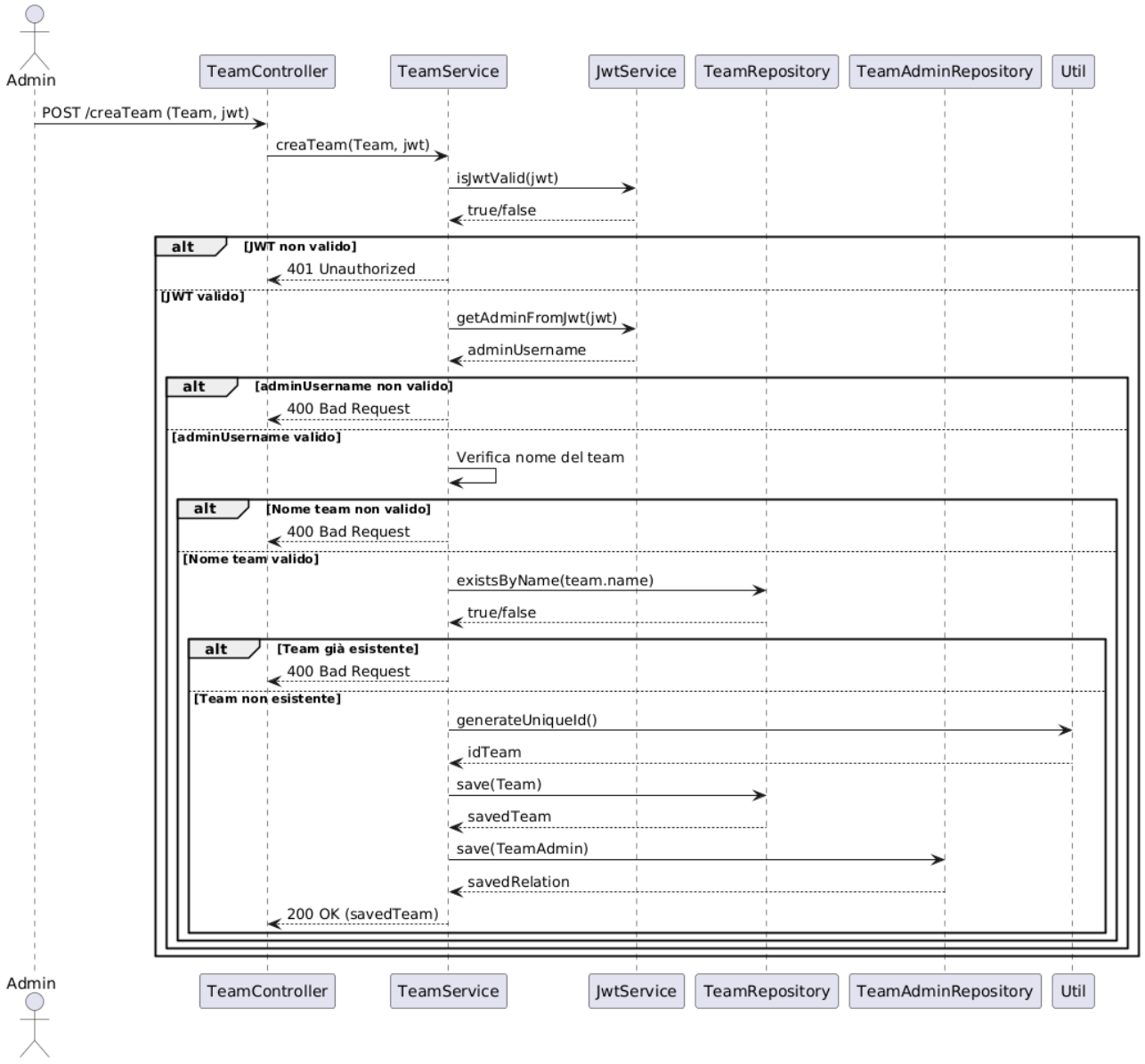
I diagrammi per la dinamica dell'architettura rappresentano il comportamento e le interazioni dei componenti di un sistema nel tempo, evidenziando il flusso di dati, i cambiamenti di stato e le dipendenze operative.
Questi diagrammi sono essenziali per comprendere non solo la struttura statica di un sistema, ma anche il modo in cui i suoi elementi collaborano per realizzare le funzionalità richieste.
Partendo dagli **Activity Diagram** fatti in fase di analisi, siamo arrivati a dei **Sequence diagram** dettagliati e tarati sull’architettura aggiornata.

Sequence Diagram

I **Sequence Diagram** forniscono una rappresentazione dettagliata dei flussi di interazione tra gli attori e i componenti del sistema.
Questi diagrammi sono fondamentali per comprendere come le funzionalità principali del sistema vengono implementate e orchestrate, evidenziando le sequenze di messaggi scambiati tra i diversi moduli.

Sequence Diagram: creaTeam

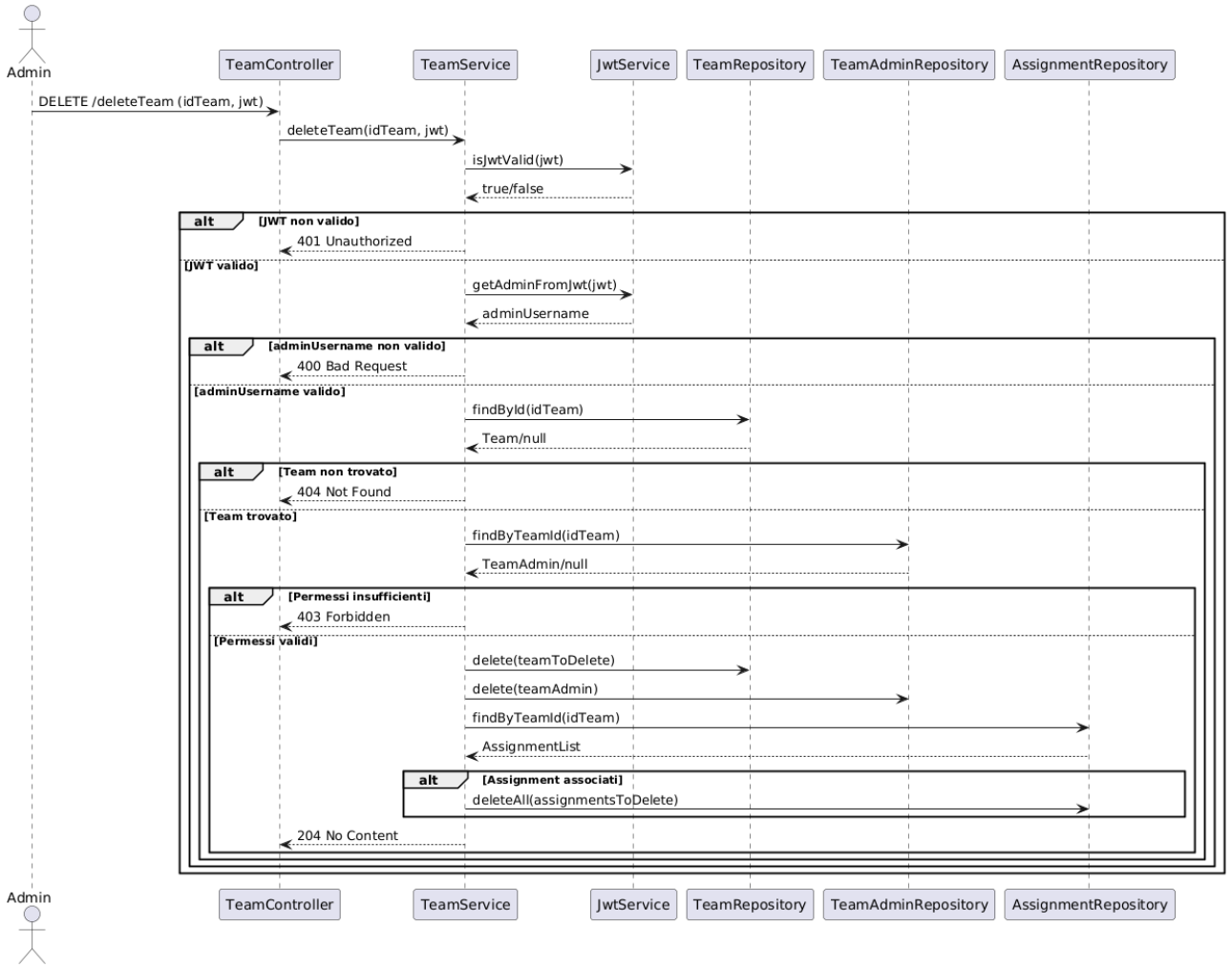
L'utente (admin) invia una richiesta `POST` per creare un nuovo team. Il **TeamController** richiama il metodo `creaTeam()` del **TeamService**, che verifica il token JWT, estrae l'admin dal token, valida il nome del team e controlla che non esista già un team con lo stesso nome.
Successivamente, salva il team nel database, crea una relazione tra l'admin e il team e salva anche questa nel database.
Infine, il TeamController restituisce il team creato al client.



Sequence Diagram: deleteTeam

L'utente (admin) invia una richiesta `DELETE` per eliminare un team specificato dall'ID. Il TeamController richiama il metodo `deleteTeam()` del TeamService, che verifica il token JWT e valida che l'admin sia associato al team come "*Owner*".

Dopo aver confermato l'esistenza del team e dei permessi dell'admin, il servizio elimina il team dal database, rimuove l'associazione con l'admin e cancella eventuali assignment associati. Infine, il TeamController restituisce una risposta che conferma l'avvenuta eliminazione.

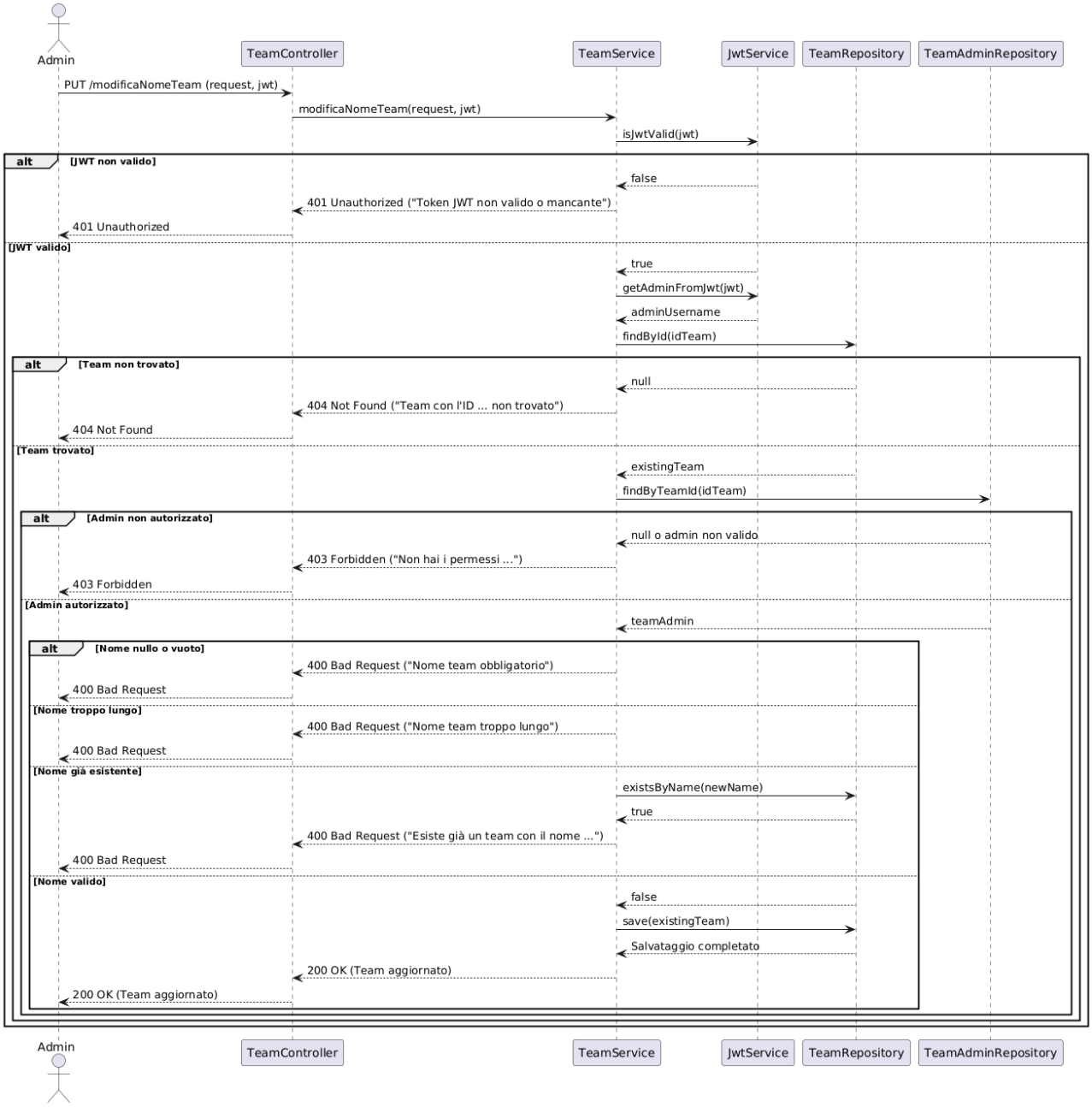


Sequence Diagram: modificaNomeTeam

L'utente (admin) invia una richiesta `PUT` per modificare il nome di un team specificato dall'ID e dal nuovo nome desiderato.

Il **TeamController** richiama il metodo `modificaNomeTeam()` del **TeamService**, che verifica il token JWT e controlla che l'admin sia associato al team come "*Owner*".
Dopo aver confermato l'esistenza del team e i permessi dell'admin, il servizio aggiorna il nome del team e salva le modifiche nel database.
Ovviamente, prima di effettuare il salvataggio, sono stati considerati tutte i possibili controlli sul newName.

Infine, il **TeamController** restituisce il team aggiornato al client.

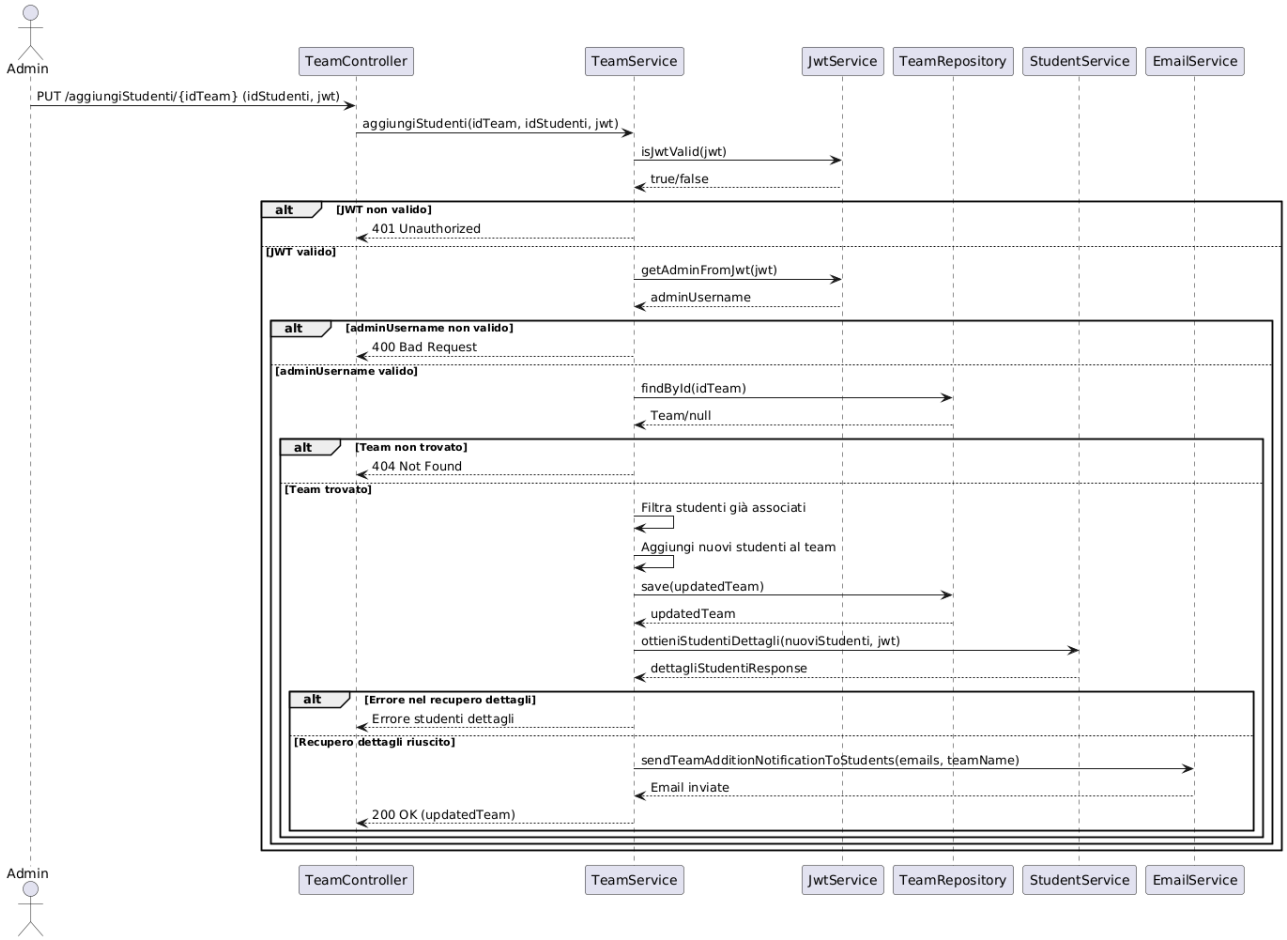


Sequence Diagram: aggiungiStudenti

L'utente (admin) invia una richiesta `PUT` per aggiungere una lista di studenti a un team specificato dall'ID.

Il **TeamController** chiama il metodo `aggiungiStudenti()` del TeamService, che verifica la validità del token JWT e controlla che l'admin sia associato al team come "*Owner*". Dopo aver confermato l'esistenza del team e i permessi dell'admin, il servizio filtra gli studenti già associati al team, aggiunge solo quelli nuovi e aggiorna il database.

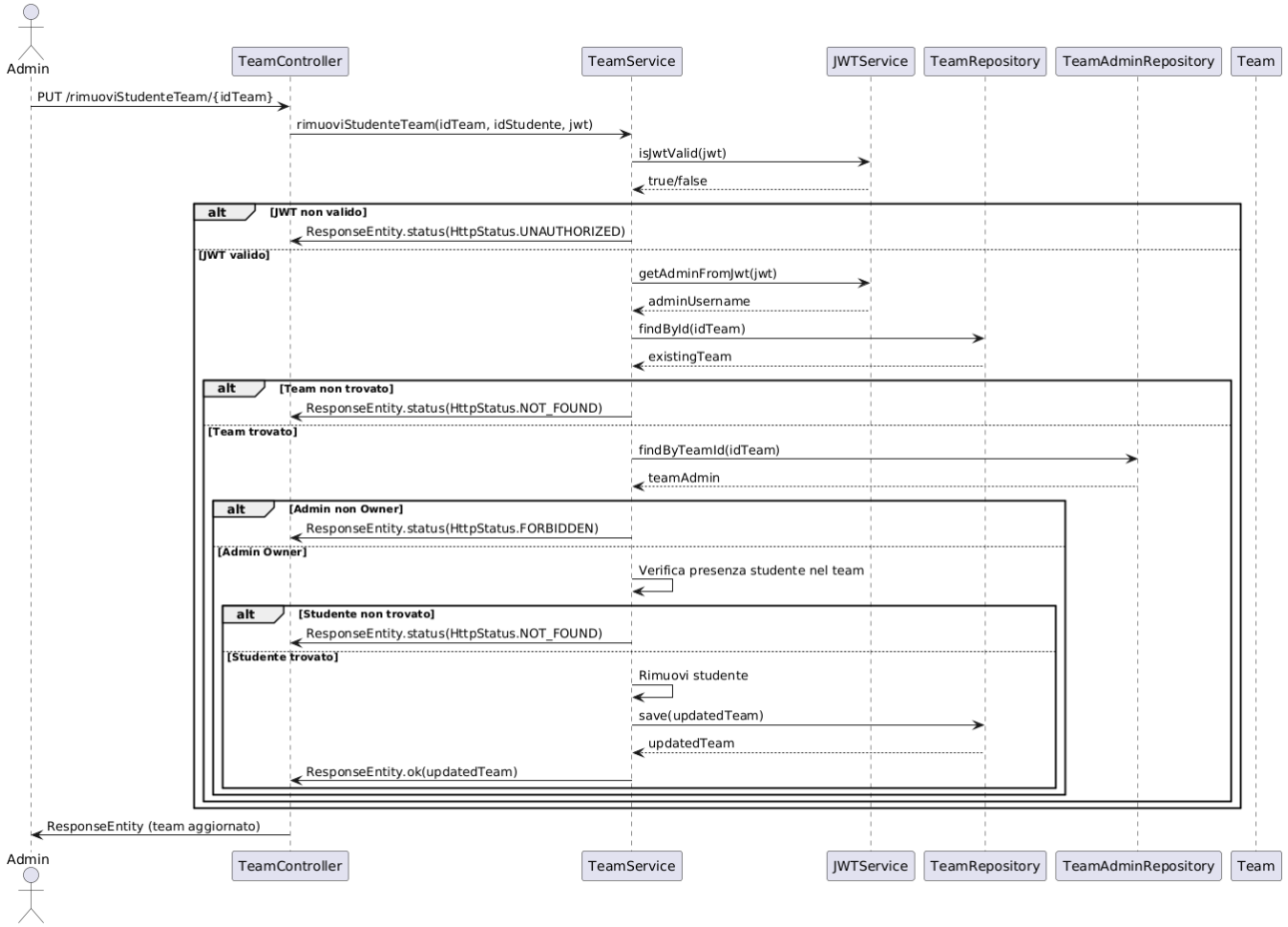
Successivamente, il servizio recupera i dettagli degli studenti appena aggiunti, invia notifiche email e restituisce il team aggiornato al client.



Sequence Diagram: rimuoviStudenteTeam

L'utente (admin) invia una richiesta `PUT` per rimuovere uno studente da un team specificato dall'ID del team e dell'ID dello studente.

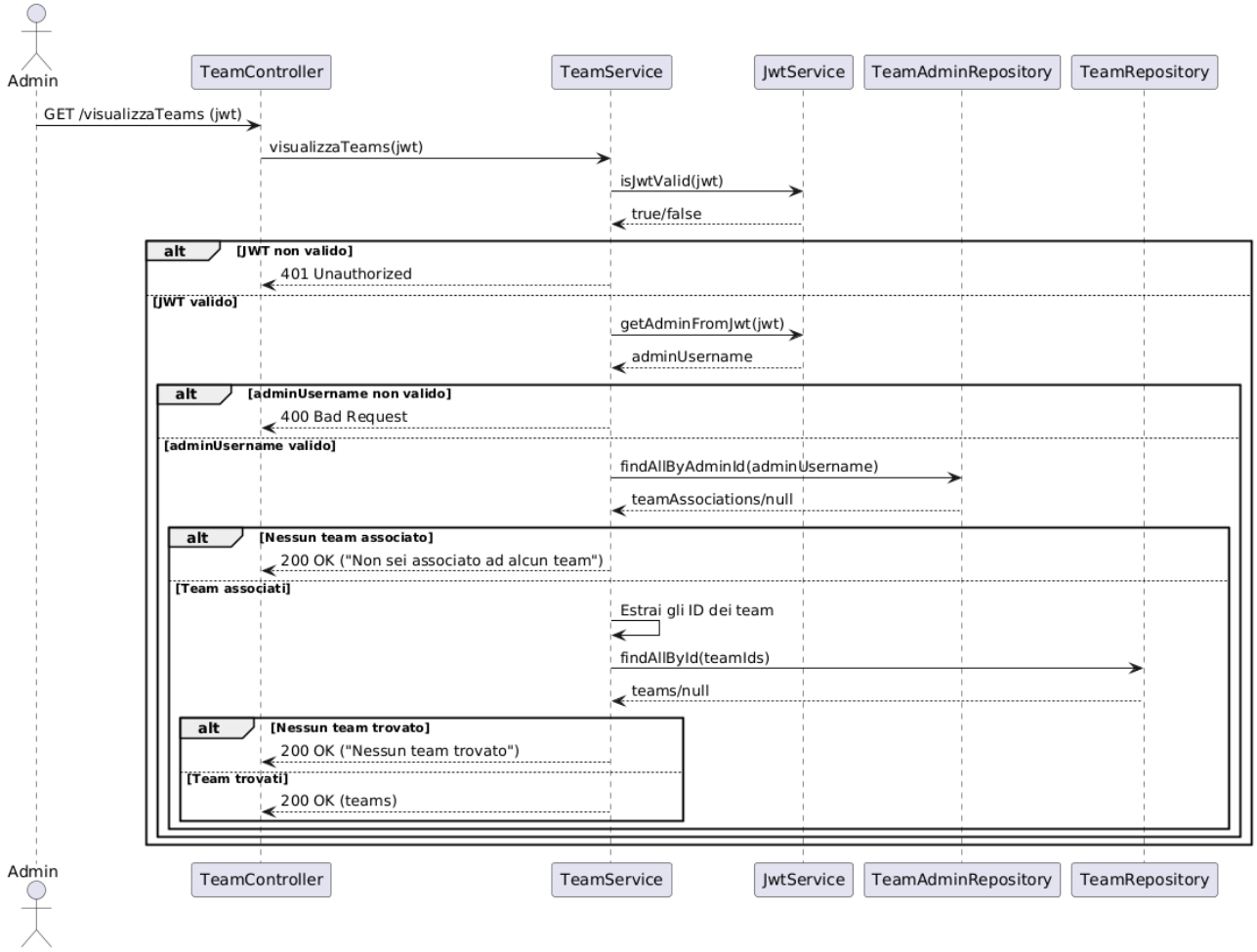
Il **TeamController** richiama il metodo `rimuoviStudenteTeam()` del **TeamService**, che verifica la validità del token JWT e conferma che l'admin sia associato al team come "*Owner*".
Dopo aver verificato che il team esista e che lo studente sia effettivamente membro, il servizio rimuove lo studente dal team, aggiorna il database con il nuovo numero di studenti e restituisce il team aggiornato al client.



Sequence Diagram: visualizzaTeams

L'utente (**admin**) invia una richiesta `GET` per visualizzare i team associati al proprio account.

Il **TeamController** richiama il metodo `visualizzaTeams()` del **TeamService**, che verifica la validità del token JWT e identifica l'admin dal token.
Successivamente, il servizio recupera tutte le associazioni tra l'admin e i team dal database e ottiene i dettagli dei team corrispondenti agli ID trovati.
Infine, il **TeamController** restituisce la lista dei team trovati al client o un messaggio indicante che non ci sono team associati, in caso contrario.

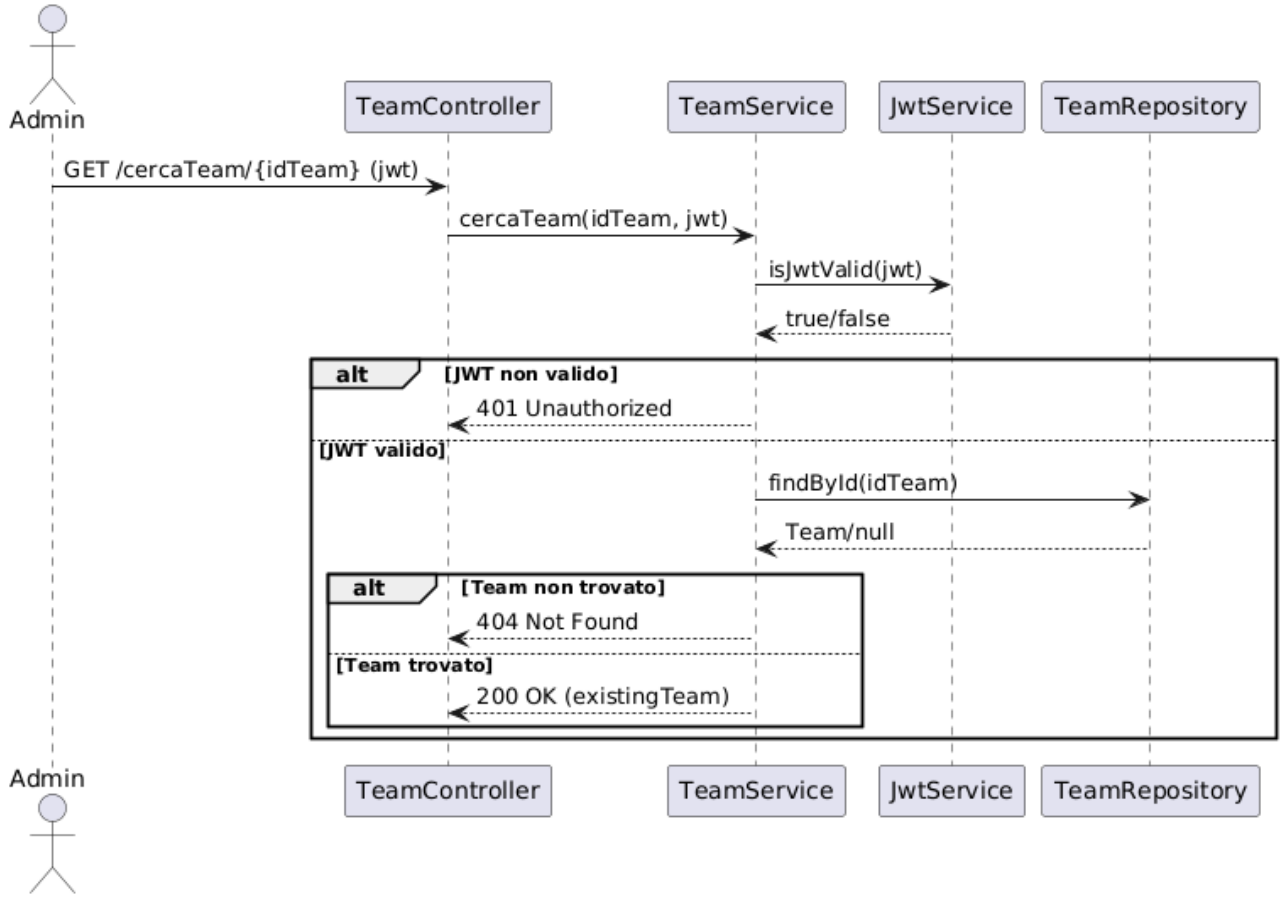


Sequence Diagram: cercaTeam

L'utente (admin) invia una richiesta `GET` per cercare un team specificato dall'ID.

Il **TeamController** chiama il metodo `cercaTeam()` del **TeamService**, che verifica la validità del token JWT per autenticare l'utente.

Successivamente, il servizio cerca il team nel database utilizzando l'ID fornito. Se il team esiste, i suoi dettagli vengono restituiti al client. In caso contrario, viene restituito un messaggio che informa che il team non è stato trovato.

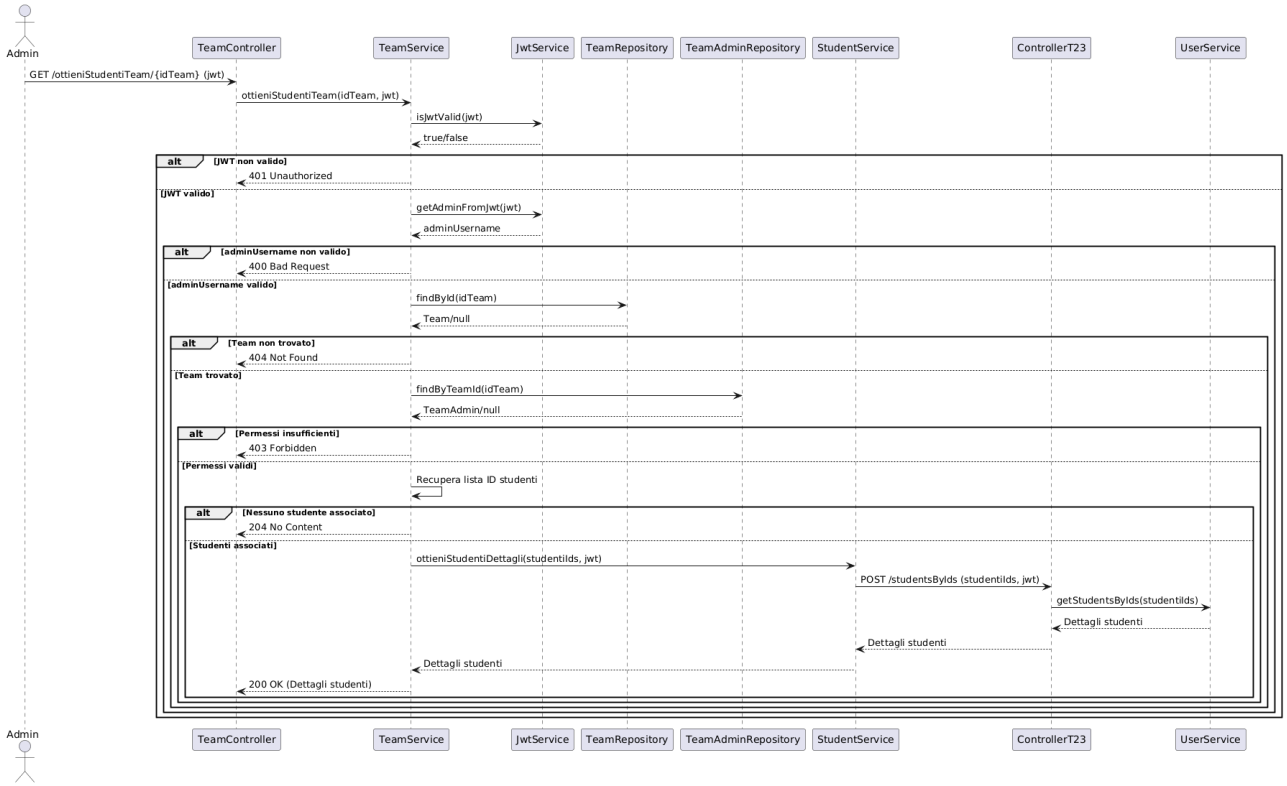


Sequence Diagram: ottieniStudentiTeam

L'utente (admin) invia una richiesta `GET` per ottenere i dettagli degli studenti associati a un team specificato dall'ID.

Il **TeamController** chiama il metodo `ottieniStudentiTeam()` del **TeamService**, che verifica la validità del token JWT per autenticare l'admin. Dopo aver confermato l'esistenza del team e i permessi dell'admin, il servizio ottiene la lista degli ID degli studenti associati al team.
Successivamente, invia una richiesta `HTTP POST` al **T23** per ottenere i dettagli completi degli studenti utilizzando il servizio `ottieniStudentiDettagli()`.

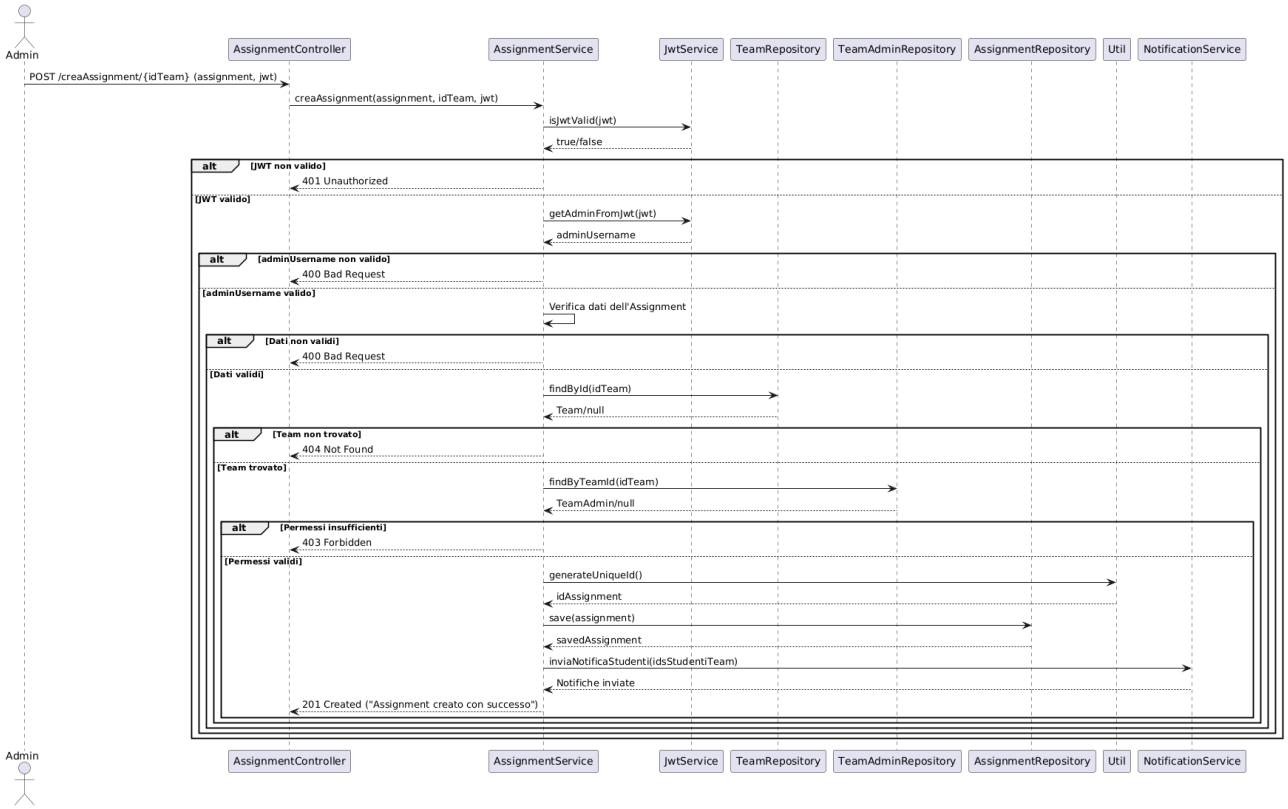
Il **T23** risponde con la lista degli studenti trovati, che viene infine restituita all'admin.



Sequence Diagram: creaAssignment

L'utente (admin) invia una richiesta `POST` per creare un nuovo assignment associato a un team specificato dall'ID.

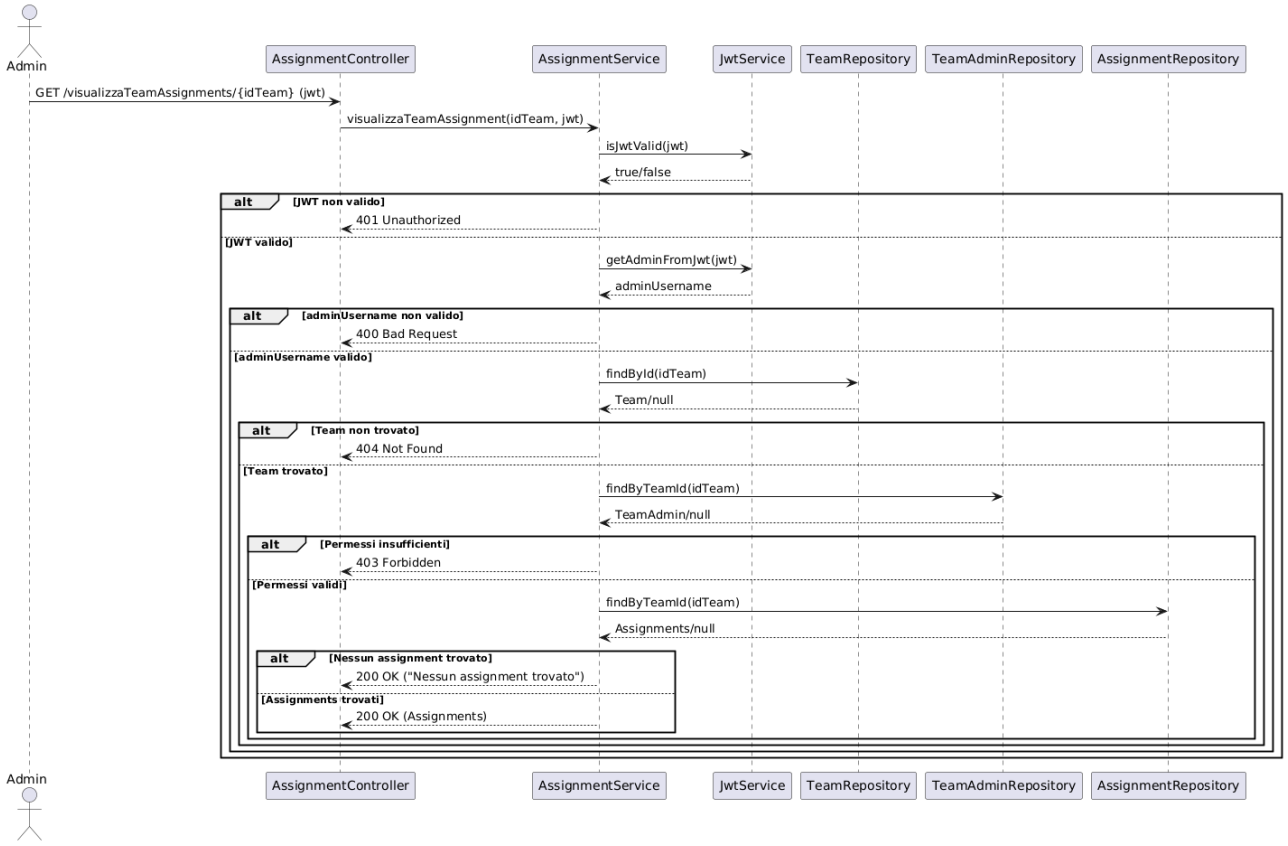
Il **AssignmentController** chiama il metodo `creaAssignment()` del **AssignmentService**, che verifica la validità del token JWT e autentica l'admin.
Successivamente, il servizio controlla che i dati dell'assignment siano validi, recupera il team dal database e verifica che l'admin abbia i permessi necessari per creare un assignment per il team. Dopo aver aggiornato i dettagli dell'assignment con le informazioni del team, il servizio salva l'assignment nel database, invia notifiche agli studenti del team, e restituisce una risposta che conferma la creazione dell'assignment.



Sequence Diagram: visualizzaTeamAssignments

L'utente (admin) invia una richiesta `GET` per ottenere gli assignment associati a un team specificato dall'ID.

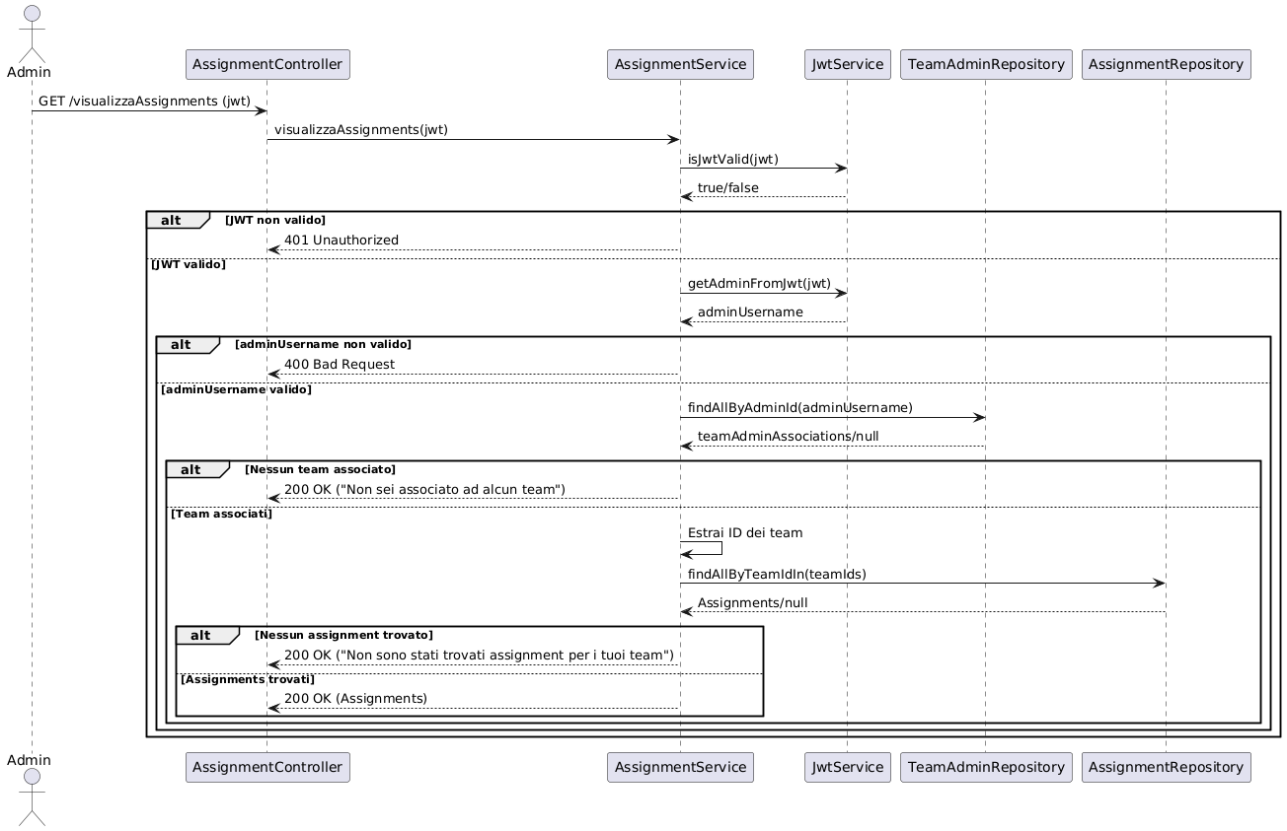
L' **AssignmentController** chiama il metodo del **AssignmentService**, che verifica la validità del token JWT e autentica l'admin. Successivamente, il servizio recupera il team dal database utilizzando l'ID e verifica che l'admin abbia i permessi necessari per visualizzare gli assignment. Se il team esiste e l'admin ha i permessi corretti, il servizio recupera gli assignment associati al team e restituisce la lista al client. In caso contrario, vengono restituiti messaggi di errore appropriati.



Sequence Diagram: visualizzaAssignments

L'utente (admin) invia una richiesta `GET` per ottenere tutti gli assignment associati ai team di cui fa parte.

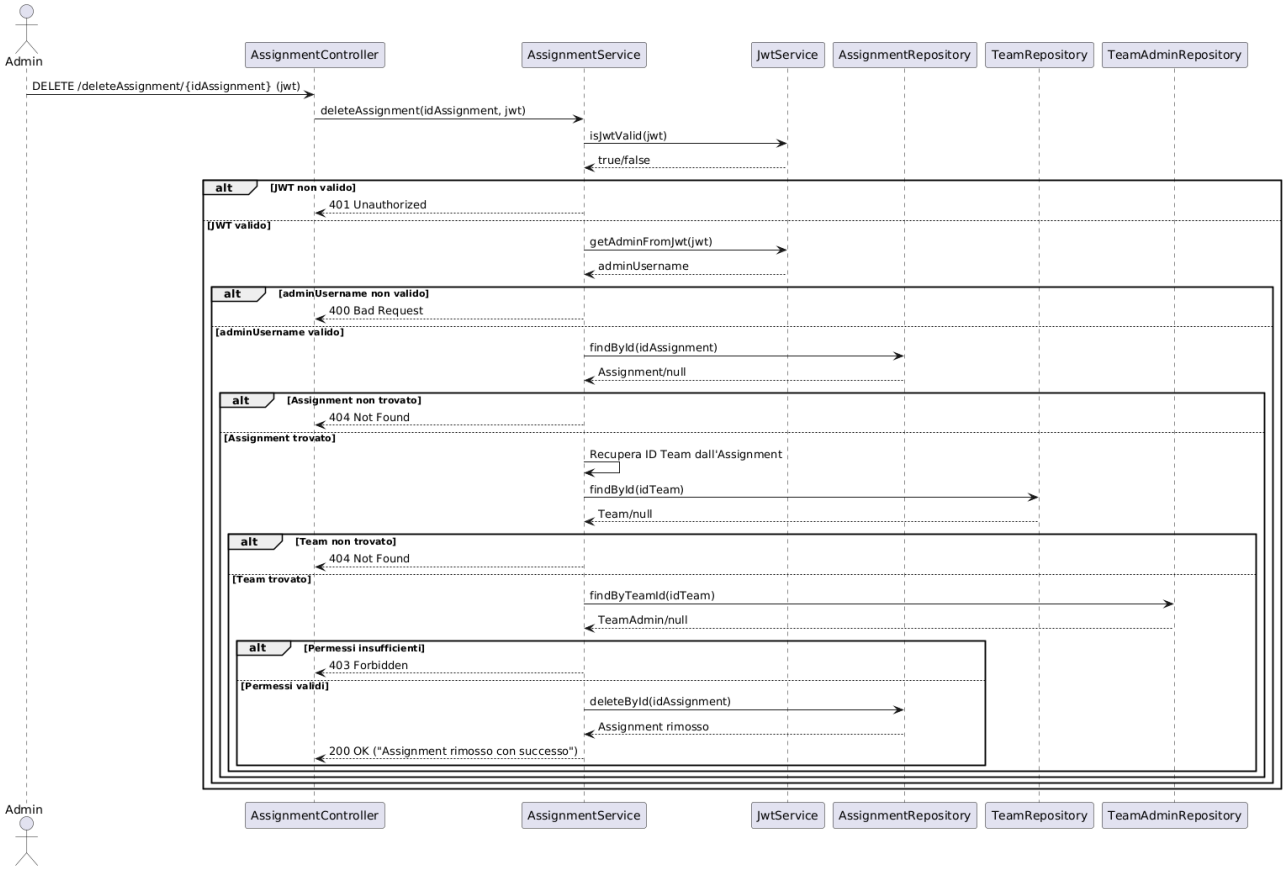
Il **AssignmentController** chiama il metodo `visualizzaAssignments()` del **AssignmentService**, che verifica la validità del token JWT per autenticare l'admin. Successivamente, il servizio recupera tutti i team associati all'admin e, a partire dagli ID dei team, cerca tutti gli assignment correlati nel database. Infine, il servizio restituisce al client la lista degli assignment trovati o un messaggio che indica l'assenza di assignment.



Sequence Diagram: deleteAssignment

L'utente (admin) invia una richiesta DELETE per rimuovere un assignment specificato dall'ID.

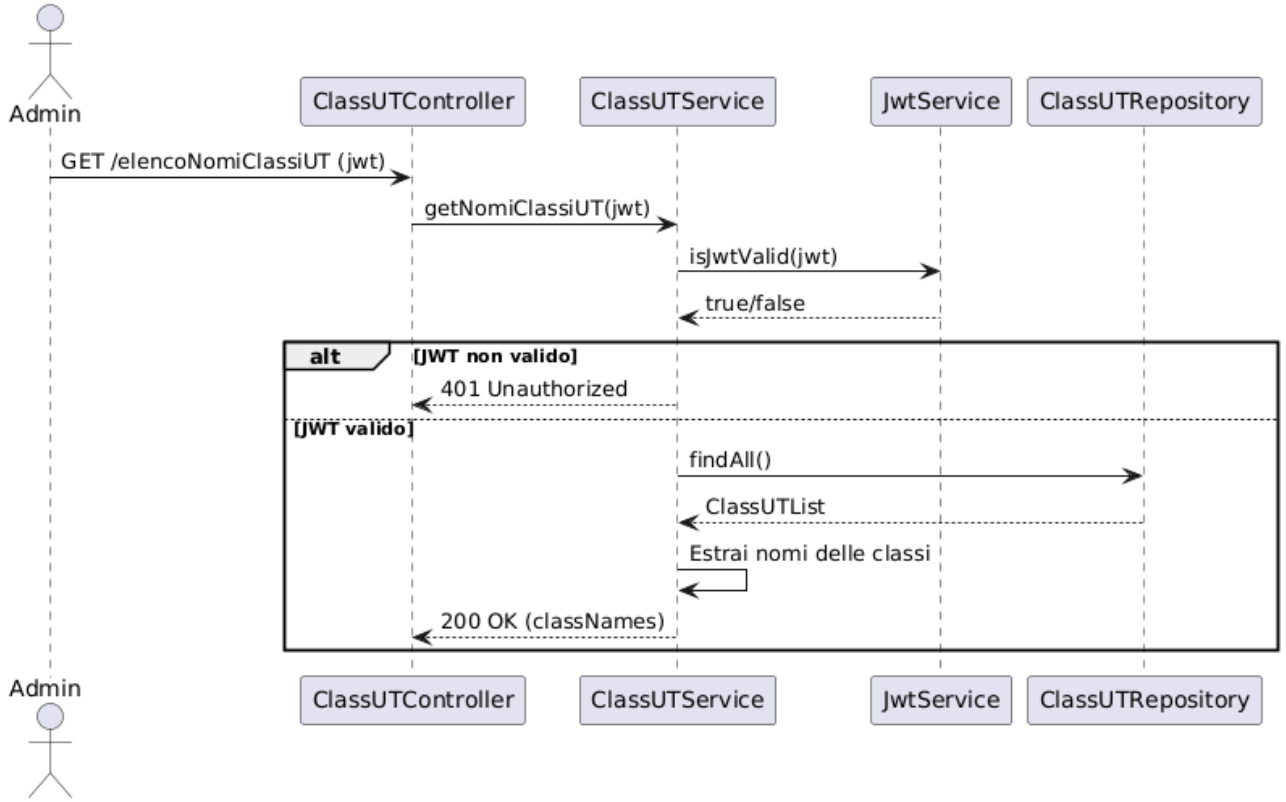
Il **AssignmentController** chiama il metodo `deleteAssignment()` del **AssignmentService**, che verifica la validità del token JWT e autentica l'admin. Successivamente, il servizio recupera l'assignment dal database utilizzando l'ID fornito, controlla che il team associato all'assignment esista e verifica che l'admin abbia i permessi necessari per rimuovere l'assignment. Se tutte le condizioni sono soddisfatte, l'assignment viene rimosso dal database e una risposta di successo viene restituita al client. In caso contrario, vengono restituiti messaggi di errore appropriati.



Sequence Diagram: elencoClassiUT

L'utente invia una richiesta `GET` per ottenere l'elenco dei nomi delle classi UT.

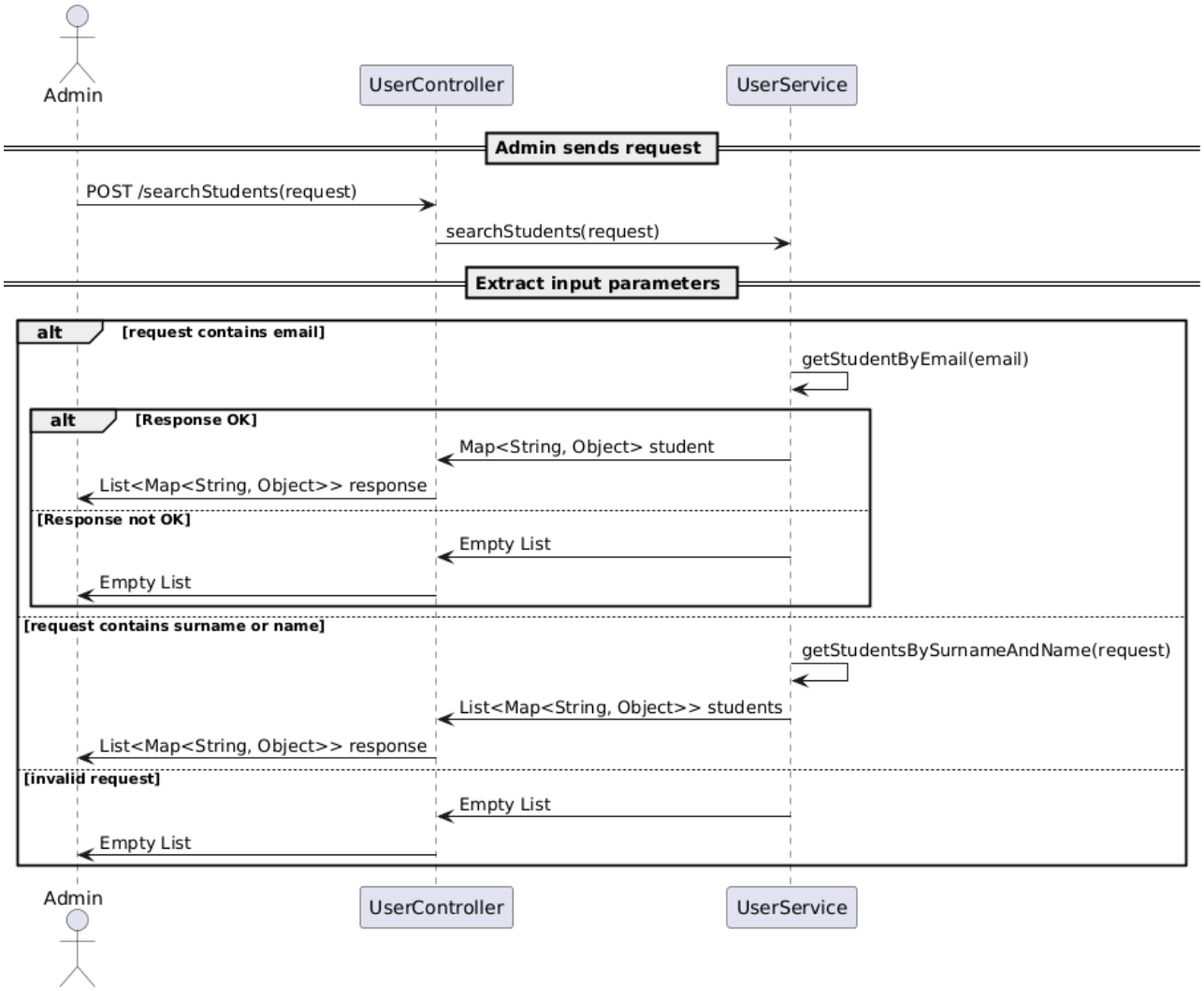
Il **ClassUTController** chiama il metodo del **ClassUTService**, che verifica la validità del token JWT per autenticare l'utente. Successivamente, il servizio recupera tutte le classi UT dal repository e estrae solo i nomi delle classi. Infine, il servizio restituisce la lista dei nomi delle classi al client con una risposta di successo (200 OK). Se il token JWT non è valido o mancante, viene restituito un messaggio di errore appropriato (401 Unauthorized).



Sequence Diagram: searchStudents

L'utente (client) invia una richiesta per cercare studenti in base a vari parametri come email, nome o cognome.

Il **T23Controller** riceve la richiesta e chiama il metodo `searchStudents()` del **UserService**. Il servizio esamina i parametri della richiesta per determinare come effettuare la ricerca. Se l'email è fornita, il sistema cerca un solo studente con quella email e restituisce i dettagli. Se il nome o il cognome sono forniti, il sistema cerca gli studenti con questi parametri. Se nessun parametro è valido, viene restituito un elenco vuoto. Infine, il sistema restituisce i risultati al client sotto forma di una lista di mappe con i dettagli degli studenti.



Problematiche riscontrate

Durante la progettazione dei requisiti **R11** e **R12**, relativi all'accesso ai risultati di un team o di un singolo studente per un determinato assignment, sono emerse difficoltà legate all'architettura del container T4.

Questo componente, progettato inizialmente senza considerare gli assignment come possibili requisiti, non consente di correlare direttamente i dati delle partite agli assignment stessi.

Motivazioni del problema

1. **Requisiti non previsti durante la progettazione del T4**: Il container T4 è stato sviluppato con l'obiettivo di gestire partite, round, turni e relativi risultati, ma all'atto della sua creazione gli assignment non erano stati considerati come entità rilevanti. Di conseguenza, non esistono collegamenti diretti tra i dati memorizzati nel T4 e quelli degli assignment.

2. **Limitazioni delle API attuali**: Le API esistenti del T4 sono ottimizzate per operazioni come creazione, modifica e recupero di dati relativi alle partite e ai turni, ma non supportano query trasversali che coinvolgano nuove entità, come gli assignment. Questo limita la possibilità di estrarre informazioni utili per soddisfare i requisiti R11 e R12.

3. **Concorrenza nello sviluppo**: Attualmente altri gruppi stanno lavorando sull'evoluzione del T4, e ogni modifica all'architettura o alle API deve essere attentamente coordinata. L'introduzione di nuovi requisiti potrebbe interferire con il lavoro in corso, aumentando la complessità gestionale e i tempi di sviluppo.

4. **Problemi preesistenti nel salvataggio delle partite**: Oltre alle limitazioni di design, sono già stati riscontrati problemi relativi alla coerenza e all'affidabilità del salvataggio dei dati delle partite. Questi problemi, sommati alla mancanza di integrazione con gli assignment, complicano ulteriormente il processo di recupero dei dati richiesti.

Conseguenze operative

La mancanza di un collegamento tra assignment e partite rende impossibile implementare i requisiti R11 e R12.

Approccio proposto per la risoluzione

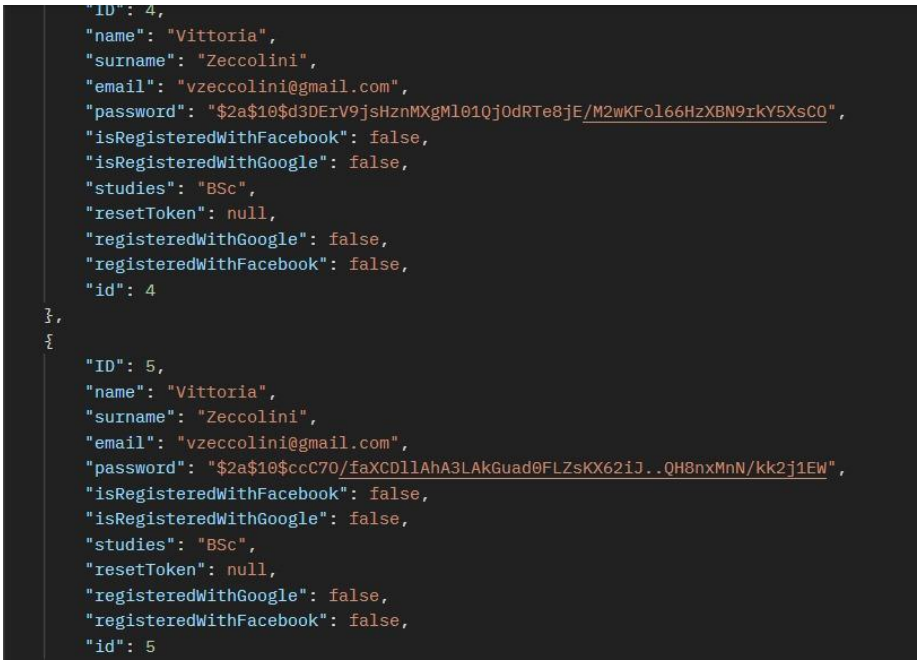
1. **Collaborazione con i gruppi responsabili del T4**: Coinvolgere i team che stanno lavorando sul T4 per valutare la fattibilità di modifiche all'architettura esistente, introducendo un collegamento tra assignment e partite.

2. **Estensione del modello di dominio**: Integrare una nuova entità, come un `AssignmentLink`, che funzioni da ponte tra assignment e partite, facilitando query trasversali e recupero dati. Tale entità potrebbe essere aggiunta come un'estensione non invasiva per non interferire con le funzionalità esistenti.

3. **Aggiornamento delle API**: Creare endpoint specifici per il recupero dei dati richiesti dai requisiti R11 e R12, sfruttando la nuova entità introdotta. Questo approccio consentirebbe di soddisfare i requisiti senza compromettere il lavoro in corso sugli altri task del T4.

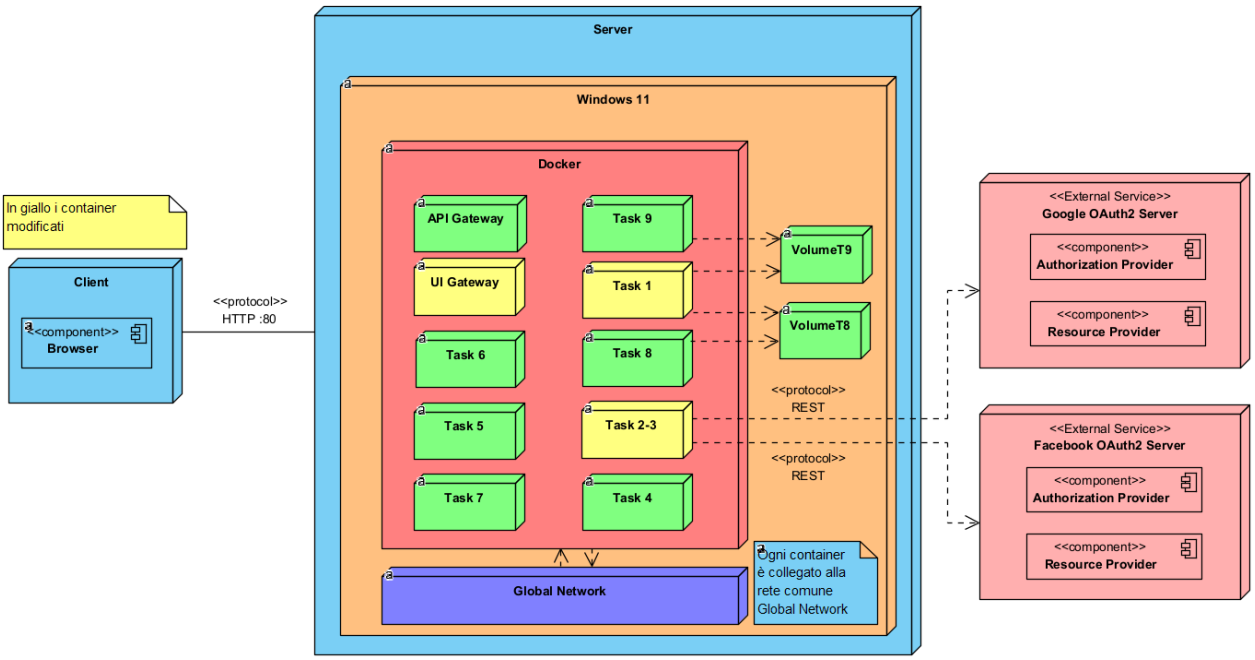
Altre problematiche

Abbiamo notato, durante la fase di testing che è possibile registrare due utenti (User) con la stessa Email senza aver alcuni tipo di controllo!



Deployment Diagram

Il **Deployment Diagram** fornisce una rappresentazione visiva delle modifiche architetturali apportate al sistema, concentrandosi sugli aggiornamenti che sono stati effettuati ai principali componenti. In particolare, sono stati modificati il **T1**, il **T23** e l'**UI Gateway**. Questi cambiamenti sono stati necessari per rendere disponibili e raggiungibili le nuove rotte implementate, migliorando l'interoperabilità e l'efficienza del sistema. Il diagramma evidenzia come i diversi moduli interagiscono tra loro, assicurando che le nuove funzionalità siano correttamente integrate e accessibili, ottimizzando così il flusso di dati e le operazioni tra i vari livelli del sistema. In sostanza, il diagramma fornisce una chiara visione delle modifiche infrastrutturali che supportano l'evoluzione delle funzionalità software, consentendo una gestione più efficace delle comunicazioni e degli accessi tra le componenti.



Testing

Test Suite per le API REST

La test suite delle API REST è un elemento chiave per garantire la qualità e l'affidabilità delle applicazioni.
Attraverso un insieme di test, la test suite verifica il corretto funzionamento delle API, consentendo di identificare bug, errori o comportamenti indesiderati. Questo strumento permette di effettuare test approfonditi su diverse funzionalità delle API, inclusi i metodi HTTP, i parametri di input, le risposte attese e gli scenari di errore.

Grazie alla test suite, i test possono essere eseguiti in modo coerente e ripetibile, garantendo la robustezza delle API e fornendo una solida base per il loro sviluppo e manutenzione.
Questi piani di test coprono i principali scenari per ciascuna delle funzionalità richieste. Ogni test case è progettato per verificare la corretta gestione degli input e per garantire che il sistema gestisca correttamente i casi di errore.

Piano di Test: creaTeam

Input: Nome del team (`name`).

ID Test	Descrizione	Input	Condizione da verificare	Risultato atteso
TC01	Creazione corretta di un team	Nome: "Team Alpha"	La richiesta è valida e i dati vengono salvati.	Team creato con successo.
TC02	Creazione con nome duplicato	Nome: "Team Alpha"	Nome team già esistente per lo stesso assignment.	Errore: "Nome team già esistente".
TC03	Nome team assente	Nome: ""	Validazione fallisce per assenza del nome.	Errore: "Nome team obbligatorio".
TC04	Nome del team troppo lungo	Nome: "stringa da 256 caratteri"	Validazione fallisce per lunghezza massima superata.	Errore: "Nome team troppo lungo".
TC05	Nome del team troppo breve	Nome: "T"	Validazione fallisce per nome troppo corto.	Errore: "Nome team troppo breve".
TC06	Nome del team con spazi bianchi	Nome: " "	Validazione fallisce per solo spazi bianchi nel nome.	Errore: "Nome team non valido".

Piano di Test: deleteTeam

Input: ID dei team sono generati casualmente in formato alfanumerico.

ID Test	Descrizione	Input	Condizione da verificare	Risultato atteso
TC01	Cancellazione corretta di un team	ID: “b2f37ae7”	Il team con l’ID specificato esiste nel sistema.	Team cancellato con successo.
TC02	Tentativo di cancellazione di un team non esistente	ID: “d9c6f3b2”	Il team con l’ID specificato non esiste nel sistema.	Errore: “Team non trovato”.
TC03	Cancellazione con ID nullo	ID: null	L’ID passato è nullo.	Errore: “ID team non valido”.
TC04	Cancellazione di un team già cancellato	ID: “b2f37ae7”	Il team con l’ID specificato è già stato cancellato.	Errore: “Team già eliminato”.

Piano di Test: modificaNomeTeam

Input: ID del team (`idTeam`) e nuovo nome del Team (`newName`).

ID Test	Descrizione	Input	Condizione da verificare	Risultato atteso
TC01	Modifica corretta del nome del team	ID: b2f37ae7, Nuovo Nome: “Team Beta”	Il team con l’ID specificato esiste nel sistema.	Nome del team modificato con successo.
TC02	Tentativo di modifica di un team non esistente	ID: d4e23f6a, Nuovo Nome: “Team Beta”	Il team con l’ID specificato non esiste nel sistema.	Errore: “Team non trovato”.
TC03	Modifica con nome vuoto	ID: b2f37ae7, Nuovo Nome: “”	Il nuovo nome è vuoto.	Errore: “Nome team obbligatorio”.
TC04	Modifica con nome troppo lungo	ID: b2f37ae7, Nuovo Nome: stringa 32 char	Il nuovo nome supera la lunghezza massima.	Errore: “Nome team troppo lungo”.
TC05	Modifica nome di un team con un nome di un altro Team	ID: b2f37ae7, Nuovo Nome: “Team Alpha”	Verifica che il nuovo nome non sia già utilizzato da un altro team	Errore: “Esiste già un team con il nome 'Team Alpha'”

Piano di Test: aggiungiStudentiTeam

Input: ID del team (`idTeam`) e lista di ID degli studenti (`listaIdStudente`).

ID Test	Descrizione	Input	Condizione da verificare	Risultato atteso
TC01	Aggiunta corretta di studenti a un team	IDTeam: d2c0a911, IDStudenti: [1, 2]	Gli studenti esistono e vengono associati correttamente al team.	Studenti aggiunti con successo.
TC02	Aggiunta con ID del team non valido	IDTeam: abcd1234, IDStudenti: [101]	Il team con l'ID specificato non esiste nel sistema.	Errore: "Team non valido".
TC03	Tentativo di aggiungere una lista vuota di studenti	IDTeam: d2c0a911, IDStudenti: []	La lista di studenti fornita è vuota.	Errore: "Nessun studente da aggiungere".

Non può capitare da front-end l’inserimento di un IDStudente non valido, proprio per come è fatta la logica di aggiunta di uno studente ad un Team.
Aggiungere un controllo impatterebbe enormemente sull’operazione in quanto occorrerebbe un ulteriore richiesta verso il T23 per verificare l’esistenza degli ID degli studenti.

Piano di Test: rimuoviStudenteTeam

Input: ID del team (`idTeam`) e ID dello studente (`idStudente`).

ID Test	Descrizione	Input	Condizione da verificare	Risultato atteso
TC01	Rimozione corretta di studenti da un team	IDTeam: d2c0a911, IDStudenti: [101, 102]	Gli studenti esistono e sono associati correttamente al team.	Studenti rimossi con successo.
TC02	Rimozione con ID del team non valido	IDTeam: abcd1234, IDStudenti: [101]	Il team con l'ID specificato non esiste nel sistema.	Errore: "Team non valido".

Analoghe considerazioni fatte per aggiungiTeamStudenti valgono per il seguente metodo.

Piano di Test: creaAssignment

Input: ID del team (`idTeam`) e l'assignment stesso (`titolo`, `descrizione`, `dataScadenza`).

ID Test	Descrizione	Input	Condizione da verificare	Risultato atteso
TC01	Creazione corretta di un assignment	IDTeam: 1, Titolo: "Assignment 1", Descrizione: "Test", DataScadenza: "2024-12-31"	Il team esiste e l'assignment viene creato correttamente con i dati forniti.	Assignment creato con successo.
TC02	Tentativo di creazione con team non esistente	IDTeam: 999, Titolo: "Assignment 1", Descrizione: "Test", DataScadenza: "2024-12-31"	Il team con l'ID 999 non esiste nel sistema.	Errore: "Team non trovato".
TC03	Creazione con dati dell'assignment incompleti	IDTeam: 1, Titolo: "", Descrizione: "Test", DataScadenza: "2024-12-31"	Il titolo dell'assignment è vuoto.	Errore: "Titolo dell'assignment obbligatorio".
TC04	Creazione con data scadenza passata	IDTeam: 1, Titolo: "Assignment 2", Descrizione: "Test", DataScadenza: "2023-12-31"	La data di scadenza è nel passato.	Errore: "Data scadenza non valida".

Piano di Test: visualizzaTeamAssignments

Input: ID del team (`idTeam`).

ID Test	Descrizione	Input	Passo da verificare	Risultato atteso
TC01	Visualizzazione degli assignments di un team esistente	IDTeam: 1	Il team esiste nel sistema e gli assignments vengono restituiti correttamente.	Assignments restituiti con successo.
TC02	Tentativo di visualizzare gli assignments di un team non esistente	IDTeam: 999	Il team con l'ID 999 non esiste nel sistema.	Errore: "Team non trovato".

Piano di Test: deleteAssignment

Input: ID dell'assignment (`idAssignment`).

ID Test	Descrizione	Input	Passo da verificare	Risultato atteso
TC01	Cancellazione corretta di un assignment	IDAssignment: bd719ad1	L'assignment esiste nel sistema e viene cancellato correttamente.	Assignment cancellato con successo.
TC02	Tentativo di cancellare un assignment non esistente	IDAssignment: bd719ad1k3k3k3k3kk	L'assignment con l'ID 999 non esiste nel sistema.	Errore: "Assignment non trovato".

Piano di Test: searchStudents

Input: email, surname, e name dello student da cercare.

ID Test	Descrizione del Test	Input	Risultato Atteso	Esito Atteso
TC01	Verifica ricerca per email	{ “email”: “student@example.com” }	La lista contiene i dettagli dello studente con quella email	Successo: La ricerca restituisce un solo studente con l’email fornita.
TC02	Verifica ricerca per nome	{ “name”: “Giuseppe” }	La lista contiene studenti con il nome “Giuseppe”	Successo: La ricerca restituisce tutti gli studenti con il nome “Giuseppe”.
TC03	Verifica ricerca per cognome	{ “surname”: “Rossi” }	La lista contiene studenti con il cognome “Rossi”	Successo: La ricerca restituisce tutti gli studenti con il cognome “Rossi”.
TC04	Verifica ricerca per nome e cognome	{ “name”: “Giuseppe”, “surname”: “Rossi” }	La lista contiene studenti con il nome “Giuseppe” e cognome “Rossi”	Successo: La ricerca restituisce gli studenti con il nome “Giuseppe” e il cognome “Rossi”.
TC05	Verifica che la ricerca ritorni un risultato vuoto quando i parametri sono vuoti	{ “name”: “”, “surname”: “”, “email”: “” }	La lista restituita è vuota	Successo: La ricerca non fornisce alcun risultato.
TC06	Verifica ricerca con solo cognome e email vuoti	{ “surname”: “Rossi” }	La lista contiene studenti con il cognome “Rossi”	Successo: La ricerca restituisce tutti gli studenti con il cognome “Rossi”.
TC07	Verifica ricerca con solo nome e email vuoti	{ “name”: “Giuseppe” }	La lista contiene studenti con il nome “Giuseppe”	Successo: La ricerca restituisce tutti gli studenti con il nome “Giuseppe”.
TC08	Verifica che la ricerca non restituisca alcun risultato per email non esistente	{ “email”: “nonexistent@example.com” }	La lista è vuota	Successo: Nessun studente trovato per l’email fornita.
TC09	Verifica gestione di una ricerca con nome e cognome vuoti	{ “name”: “”, “surname”: “” }	La lista è vuota	Successo: La ricerca non fornisce alcun risultato.
TC10	Verifica che la ricerca gestisca correttamente un parametro mancante	{ “surname”: “Rossi” } (senza email e name)	La lista contiene studenti con il cognome “Rossi”	Successo: La ricerca restituisce tutti gli studenti con il cognome “Rossi”.

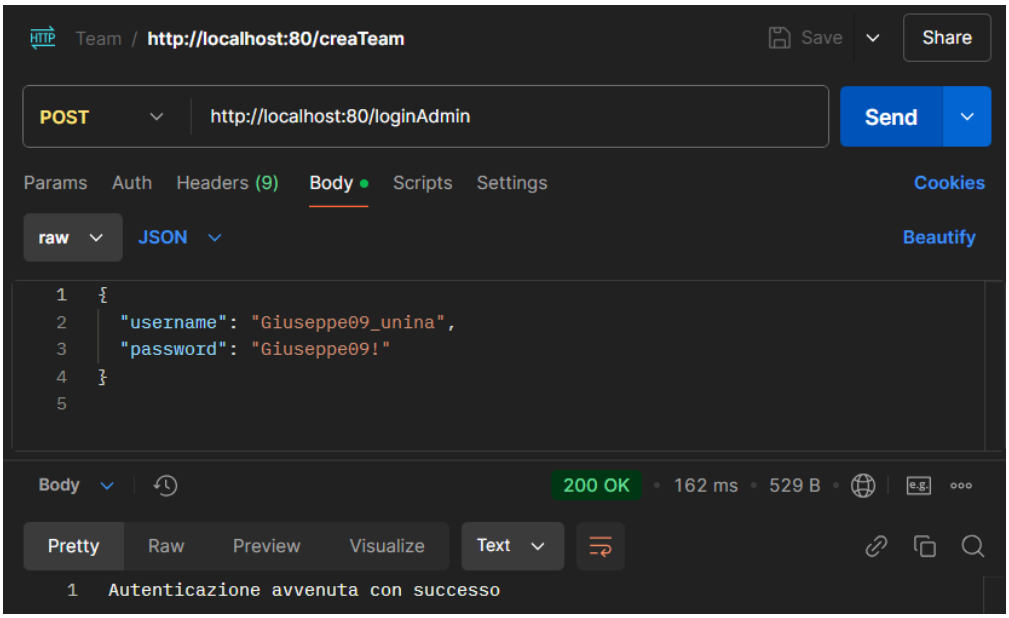
Testing con Postman

Il testing delle API è stato automatizzato utilizzando **Postman**, uno strumento che permette di inviare richieste http e visualizzare le risposte in modo interattivo.

Questo strumento facilita il controllo delle funzionalità dell'API, consentendo di verificare se le risposte ottenute siano conformi alle aspettative. Postman semplifica e accelera il processo di test, migliorando l'affidabilità e la qualità del software attraverso una gestione più efficiente dei casi di test.

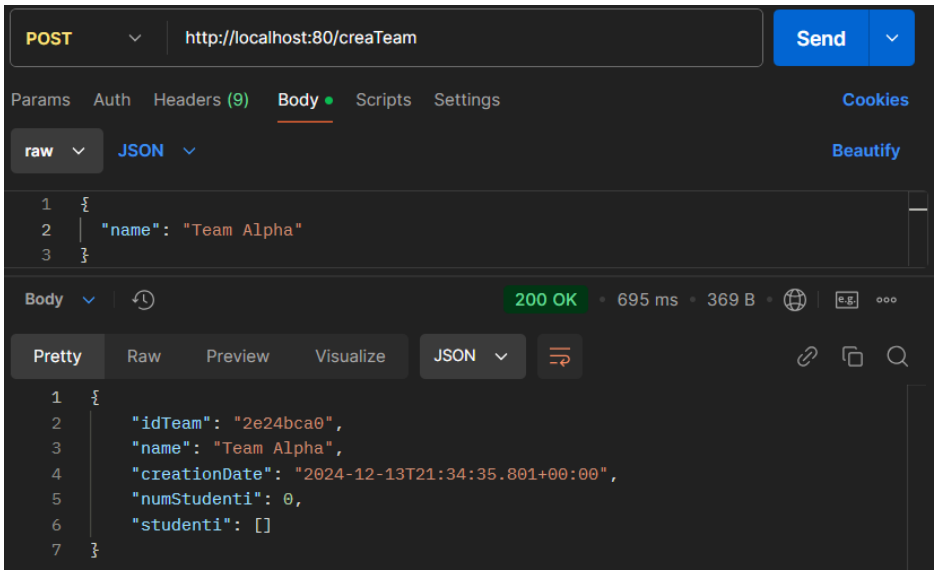
Soprattutto ha permesso una suddivisione ottimale delle responsabilità durante lo sviluppo delle modifiche.

Tutti i **test** effettuati partono dal **login** per ottenere il **token JWT**.

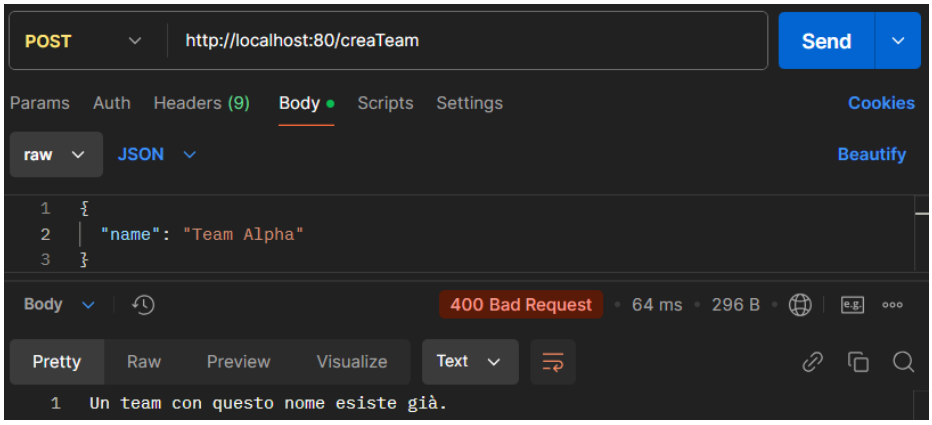


Testing creaTeam

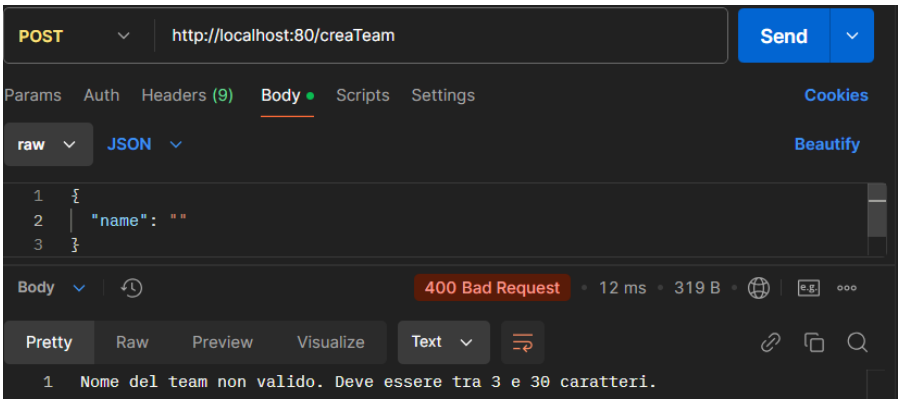
TC01 – Creazione corretta di un team:



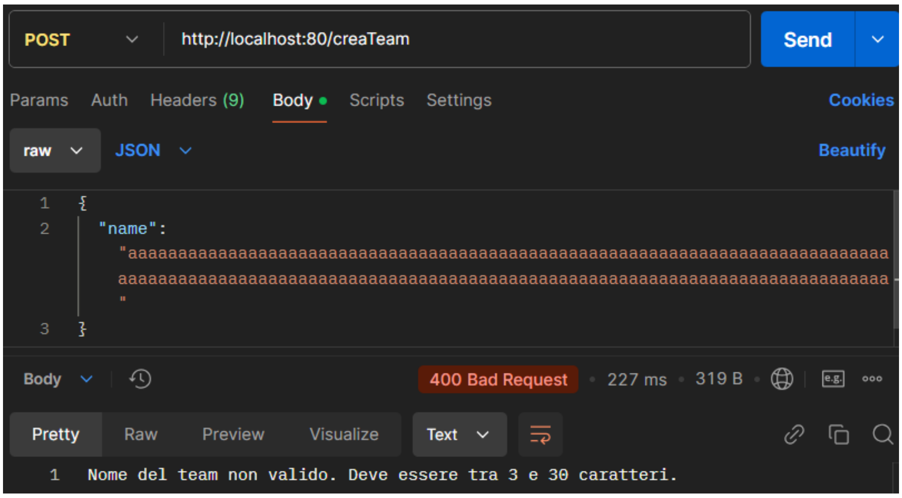
TC02 – Creazione con nome duplicato:



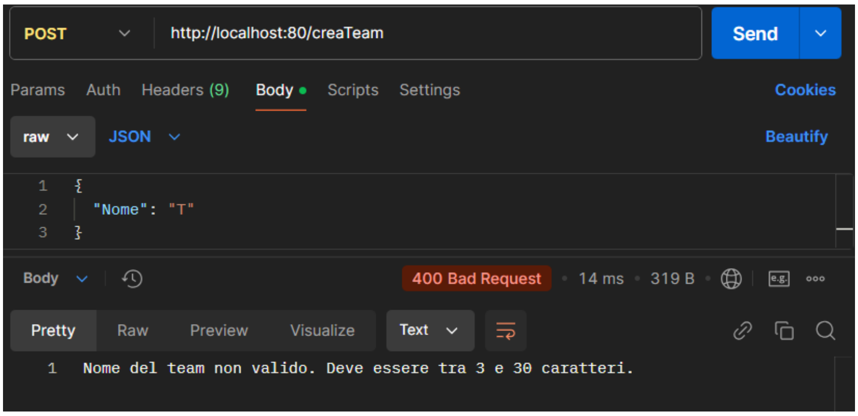
TC03 – Nome team assente:



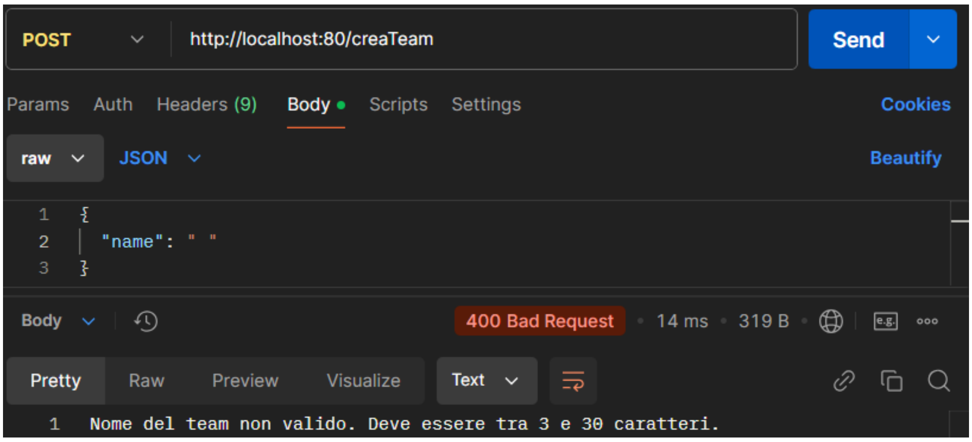
TC04 – Nome del team troppo lungo:



TC05 – Nome del team troppo breve:

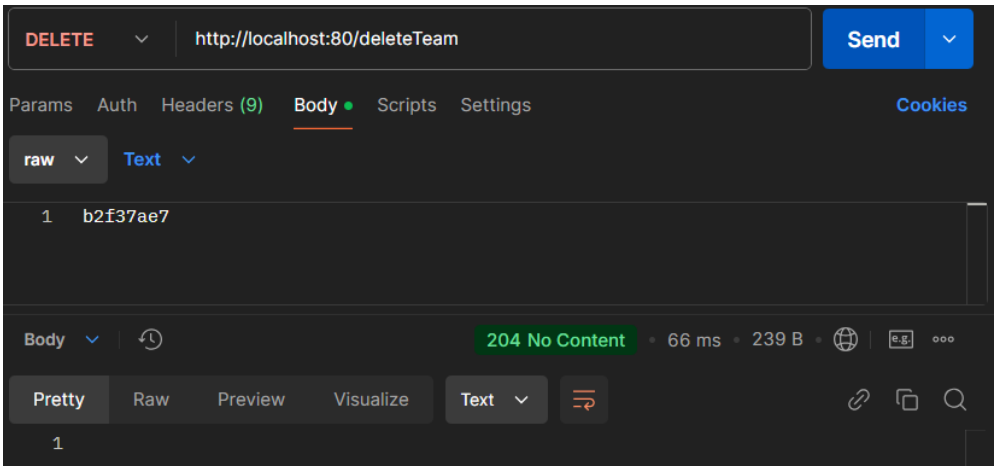


TC06 – Nome del team con spazi bianchi:

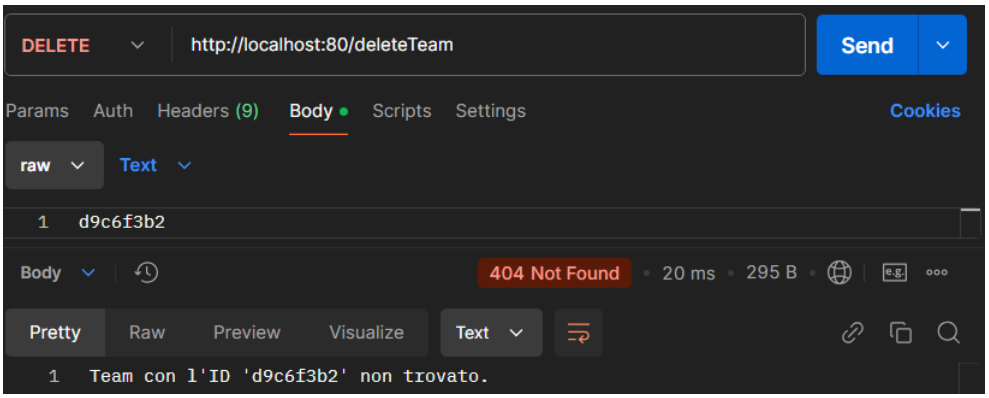


Testing deleteTeam

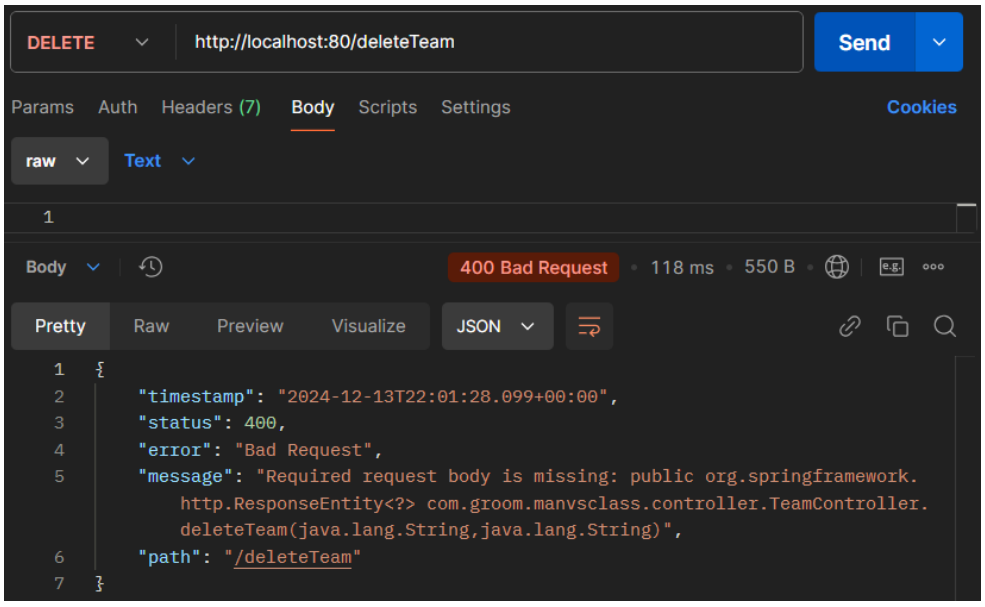
TC01 - Cancellazione corretta di un team



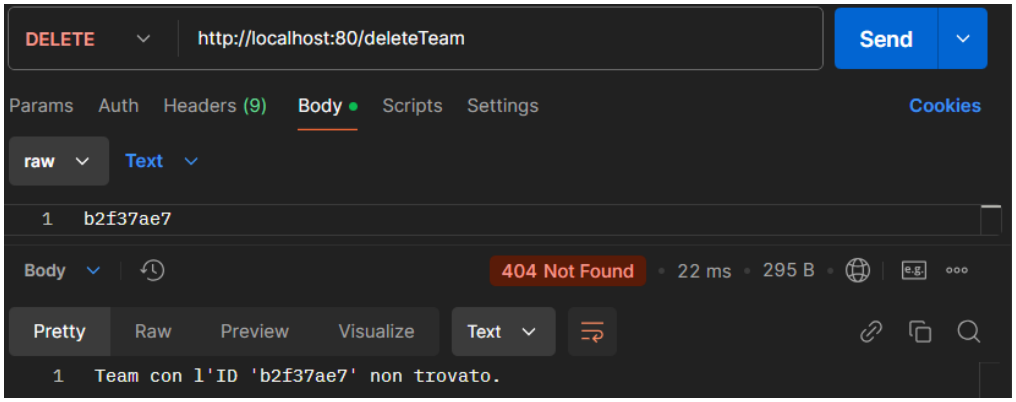
TC02 - Tentativo di cancellazione di un team non esistente



TC03 - Cancellazione con ID nullo

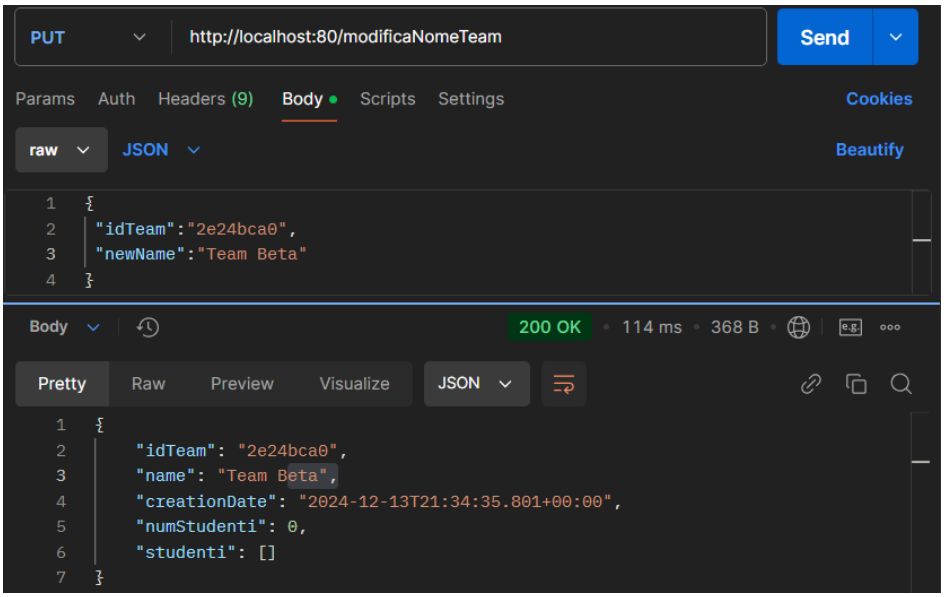


TC04 - Cancellazione di un team già cancellato

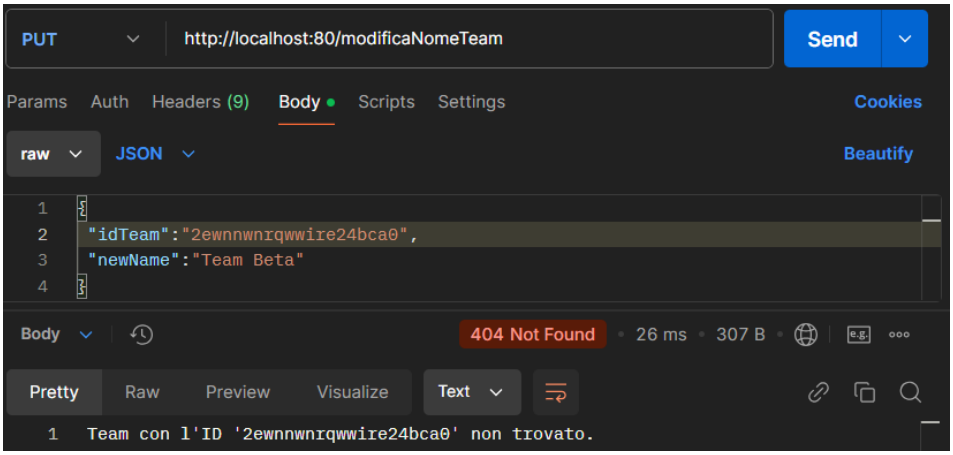


Testing modificaNomeTeam

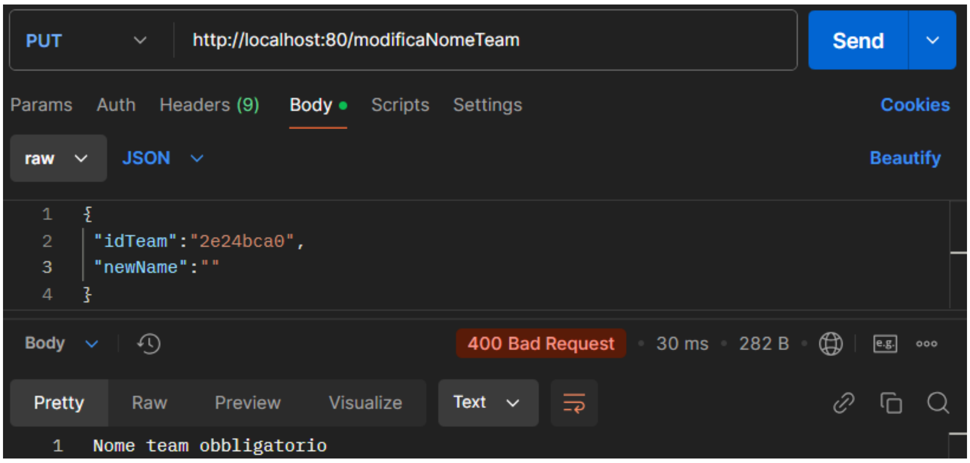
TC01 – Modifica corretta del nome del team



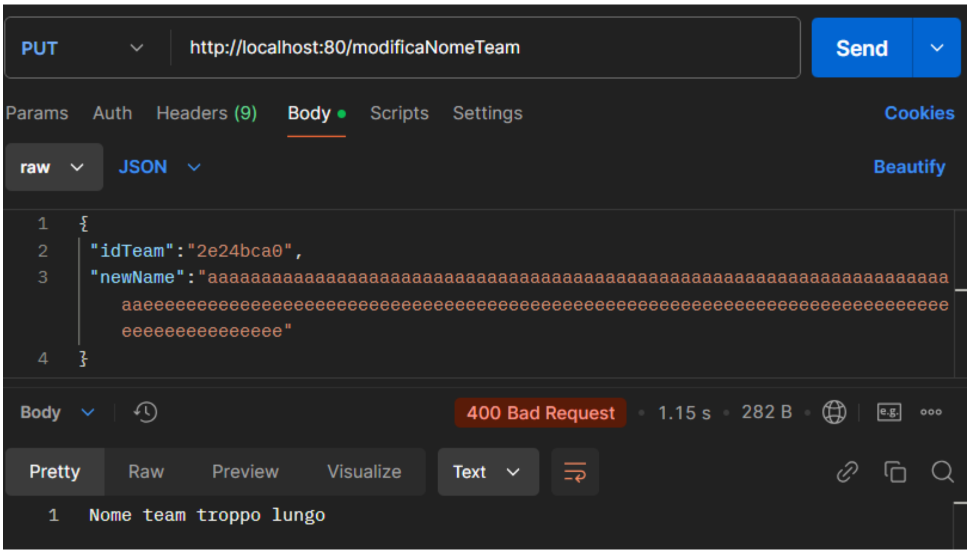
TC02 - Tentativo di modifica di un team non esistente



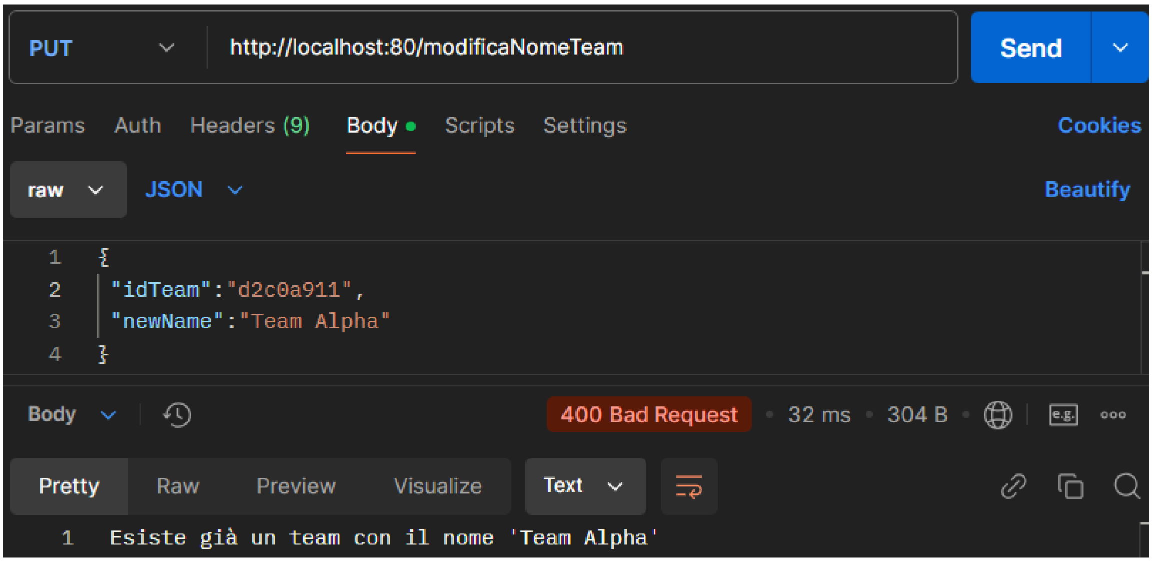
TC03 - Modifica con nome vuoto



TC04 - Modifica con nome troppo lungo

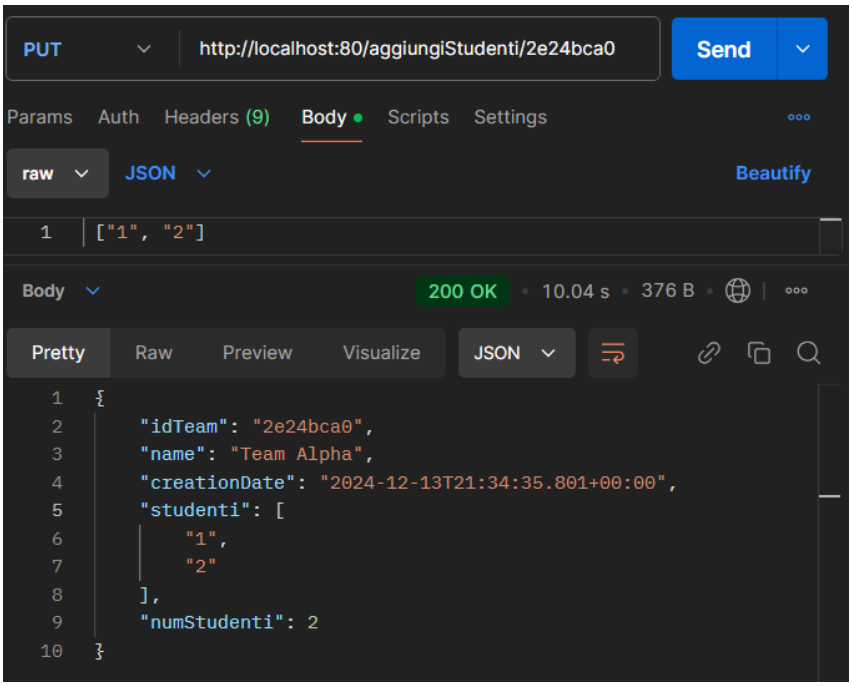


TC05 - Modifica nome di un team con un nome di un altro Team

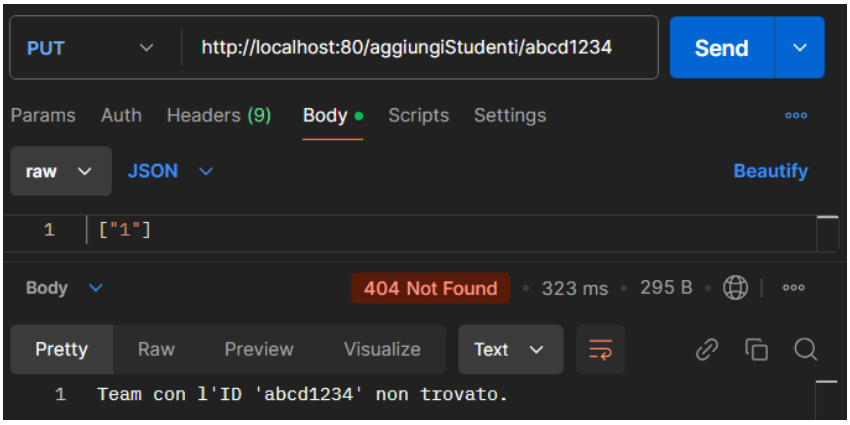


Testing aggiungiStudentiTeam

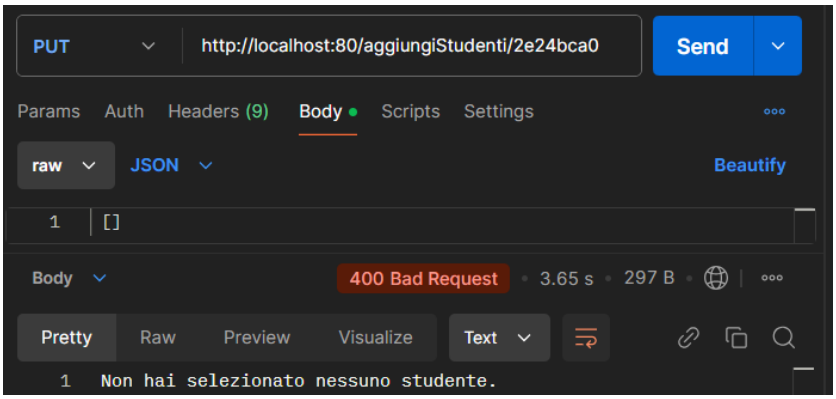
TC01 - Aggiunta corretta di studenti a un team



TC02 - Aggiunta con ID del team non valido

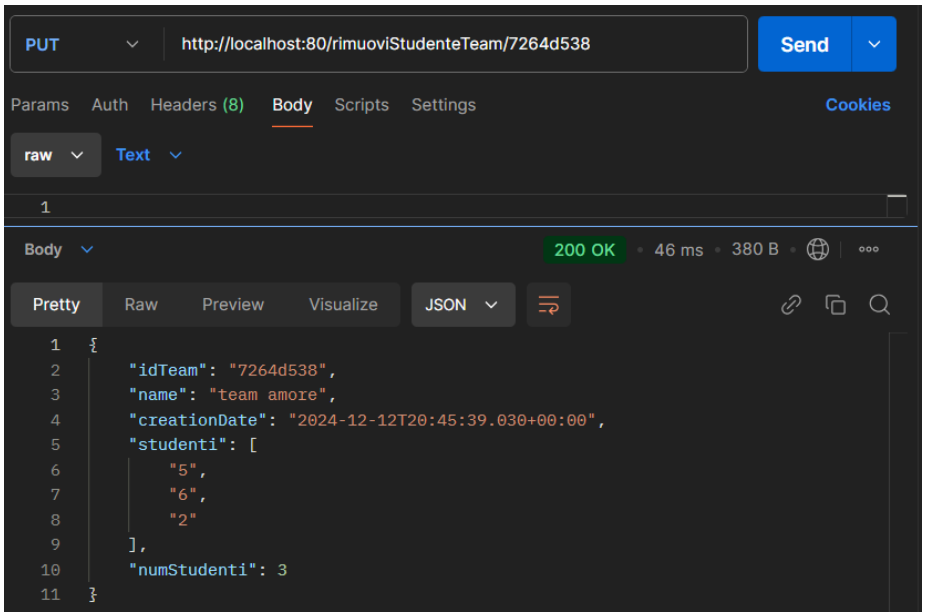


TC03 - Tentativo di aggiungere una lista vuota di studenti

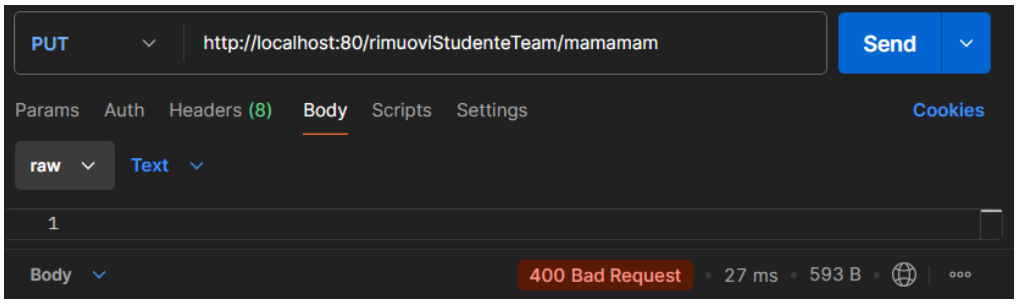


Testing: rimuoviStudenteTeam

TC01 - Rimozione corretta di studenti da un team

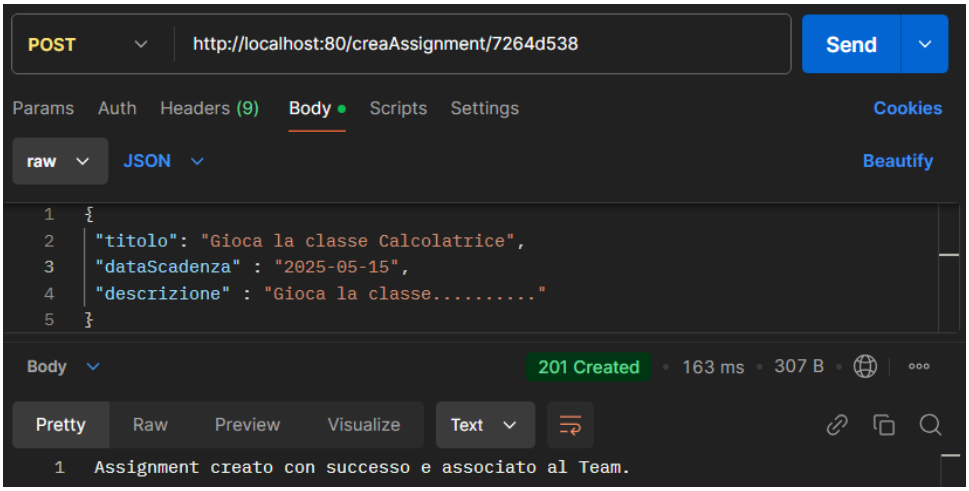


TC02 - Rimozione con ID del team non valido

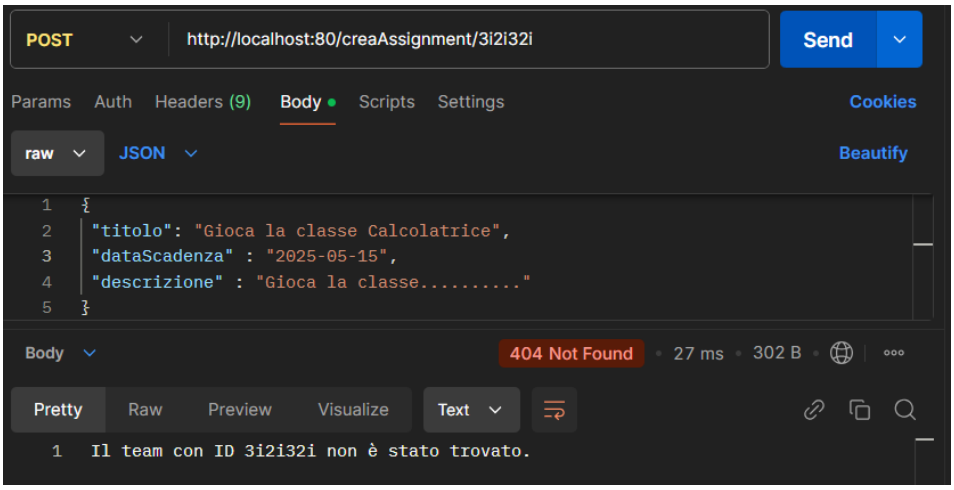


Testing: creaAssignment

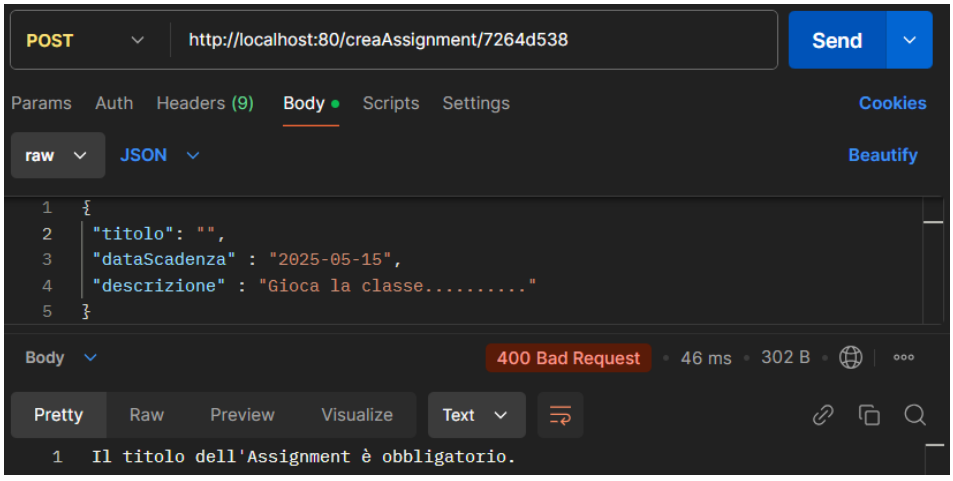
TC01 - Creazione corretta di un assignment.



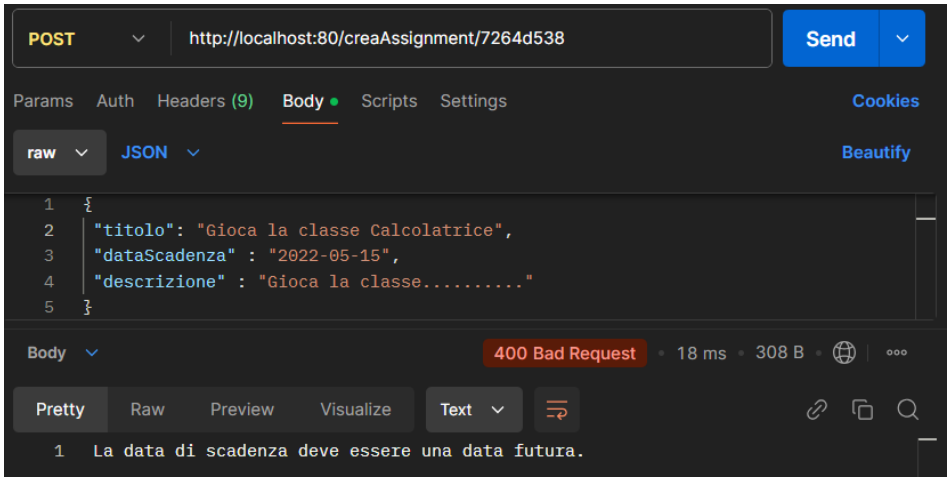
TC02: Tentativo di creazione con team non esistente



TC03 – Creazione con dati dell'assignment incompleti

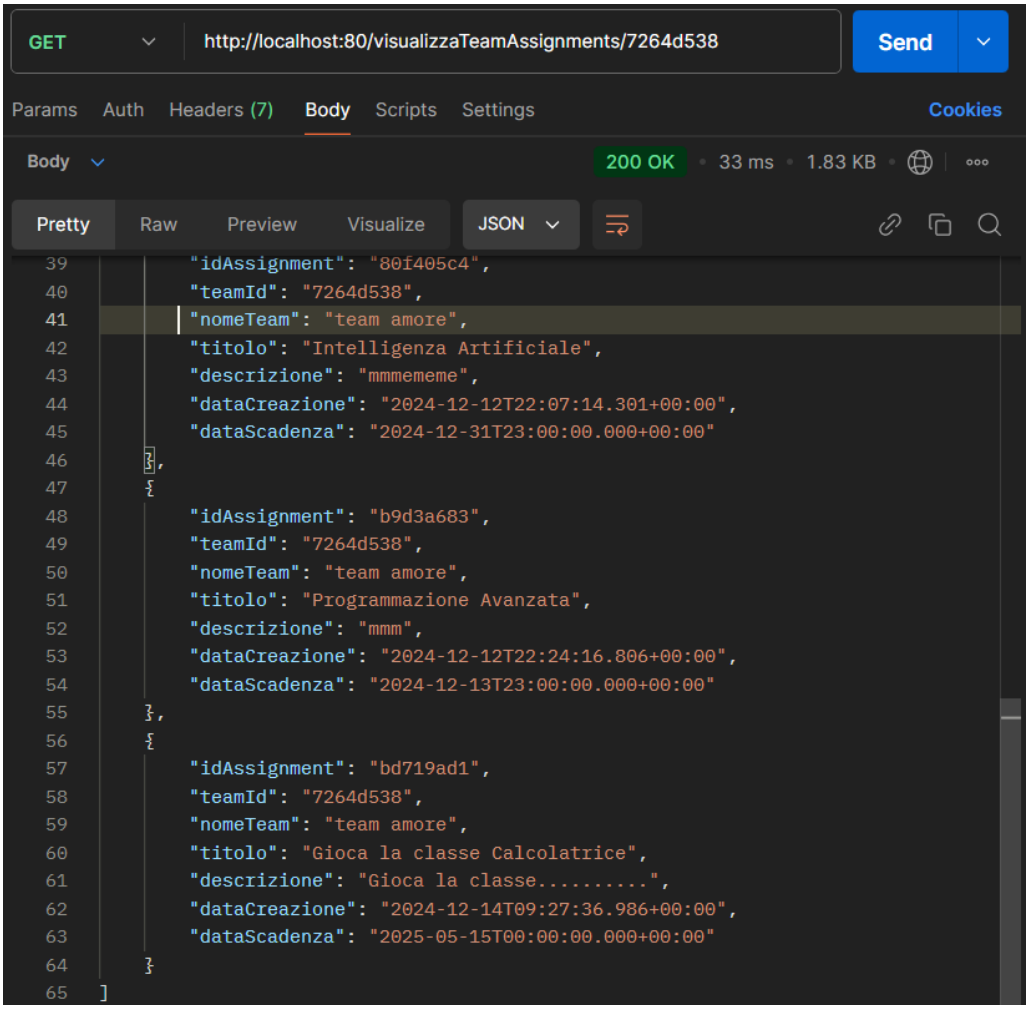


TC04 - Creazione con data scadenza passata

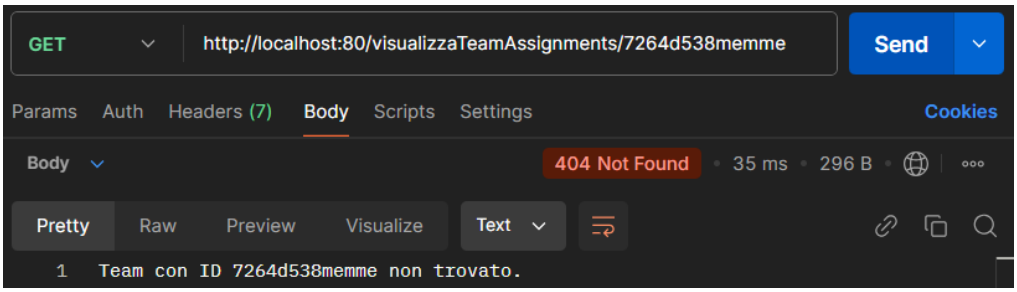


Testing: visualizzaTeamAssignments

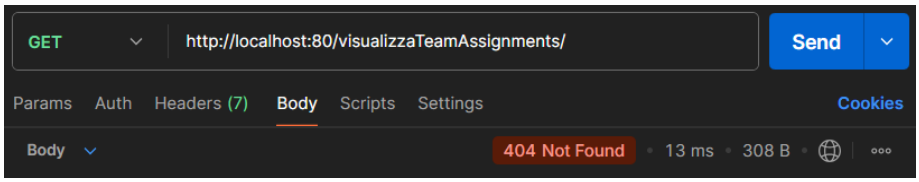
TC01 - Visualizzazione degli assignments di un team esistente.



TC02 - Tentativo di visualizzare gli assignments di un team non esistente.

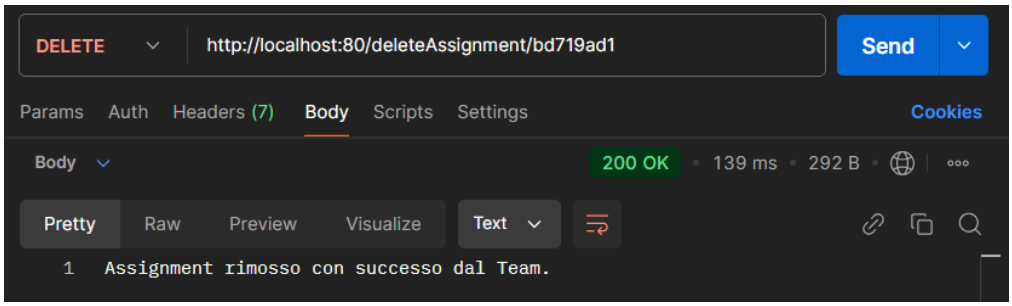


TC03 - Visualizzazione con ID team nullo.

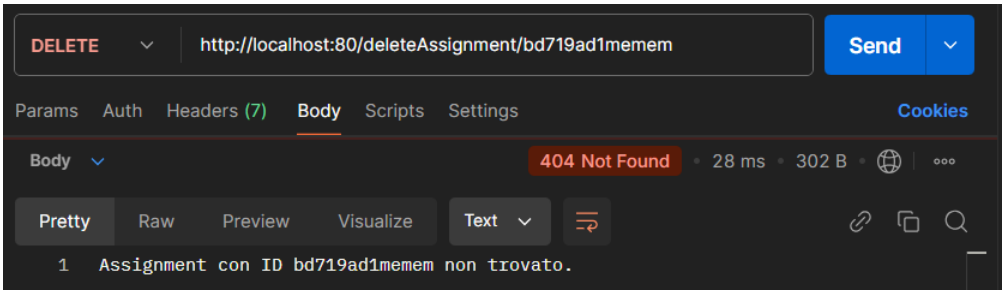


Testing: deleteAssignment

TC01 – Cancellazione corretta di un assignment

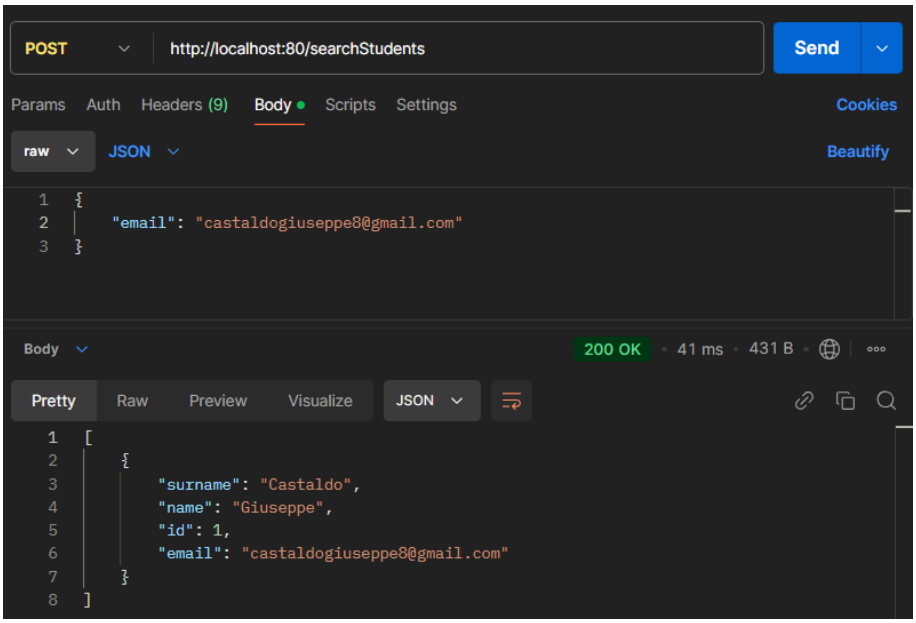


TC02 - Tentativo di cancellare un assignment non esistente

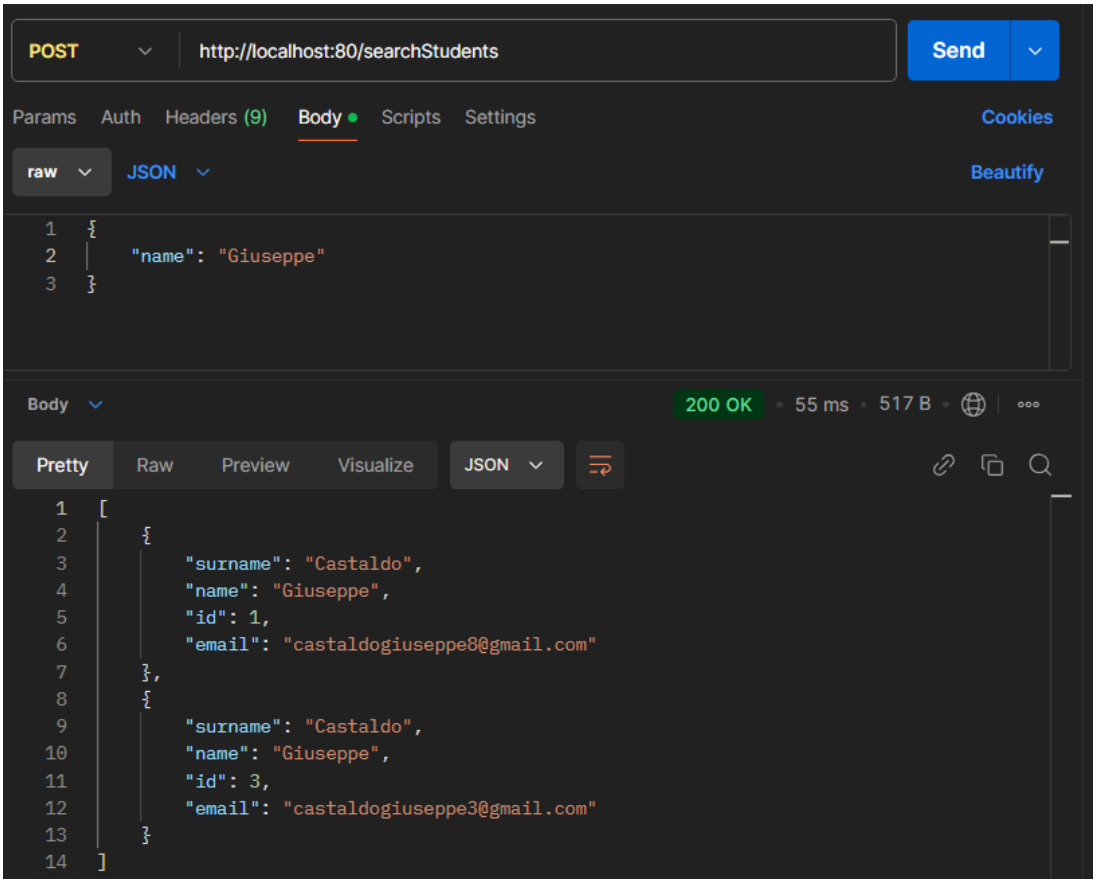


Testing: searchStudents

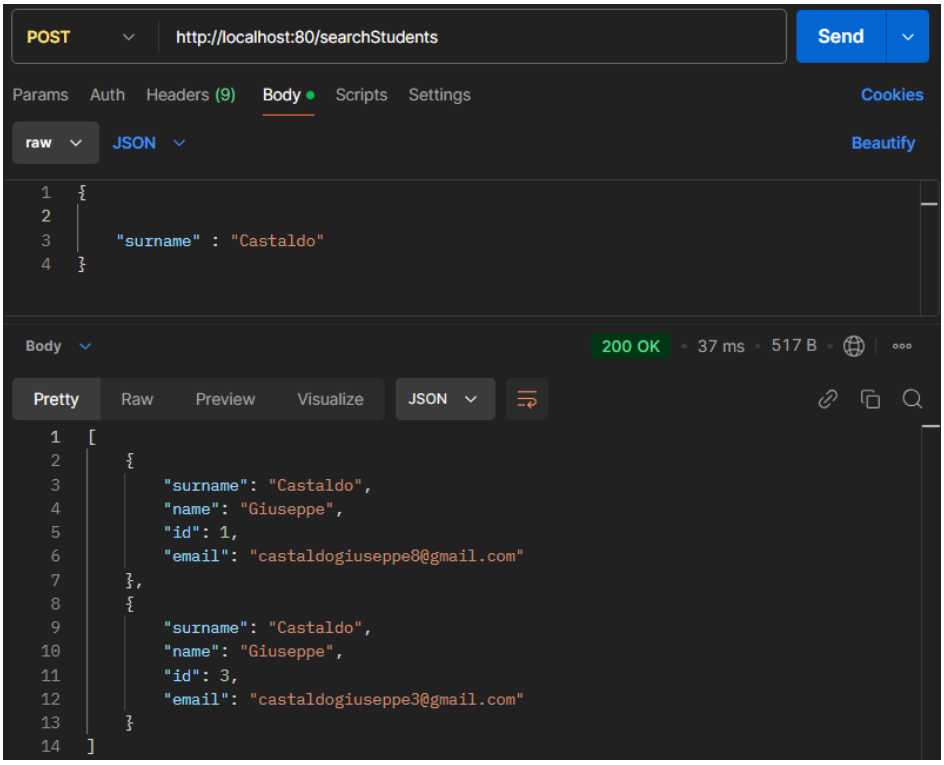
TC01 - Verifica ricerca per email



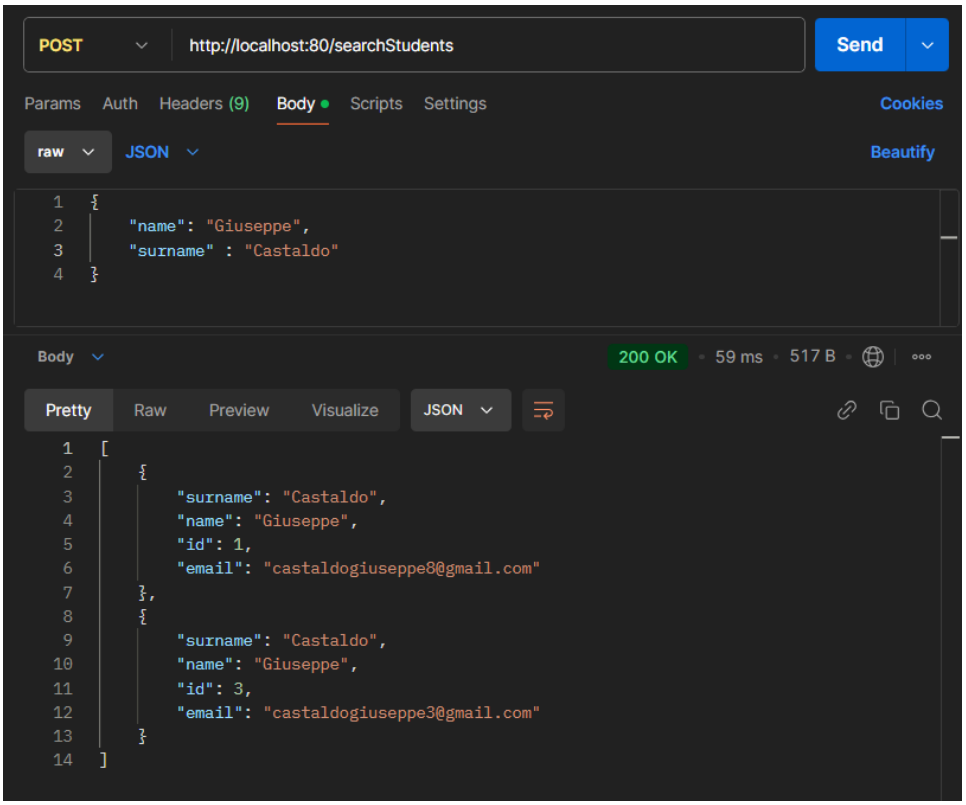
TC02 - Verifica ricerca per nome



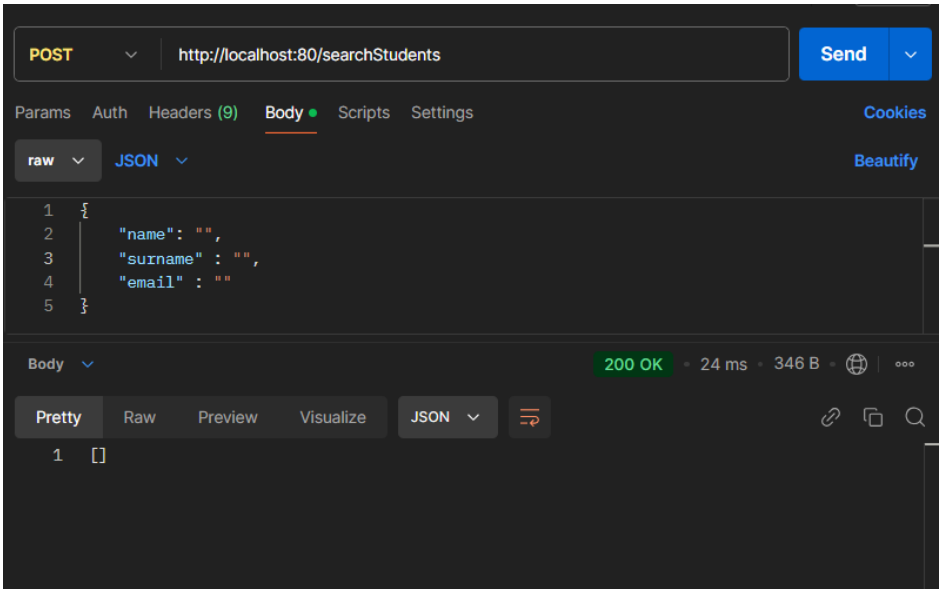
TC03 - Verifica ricerca per cognome



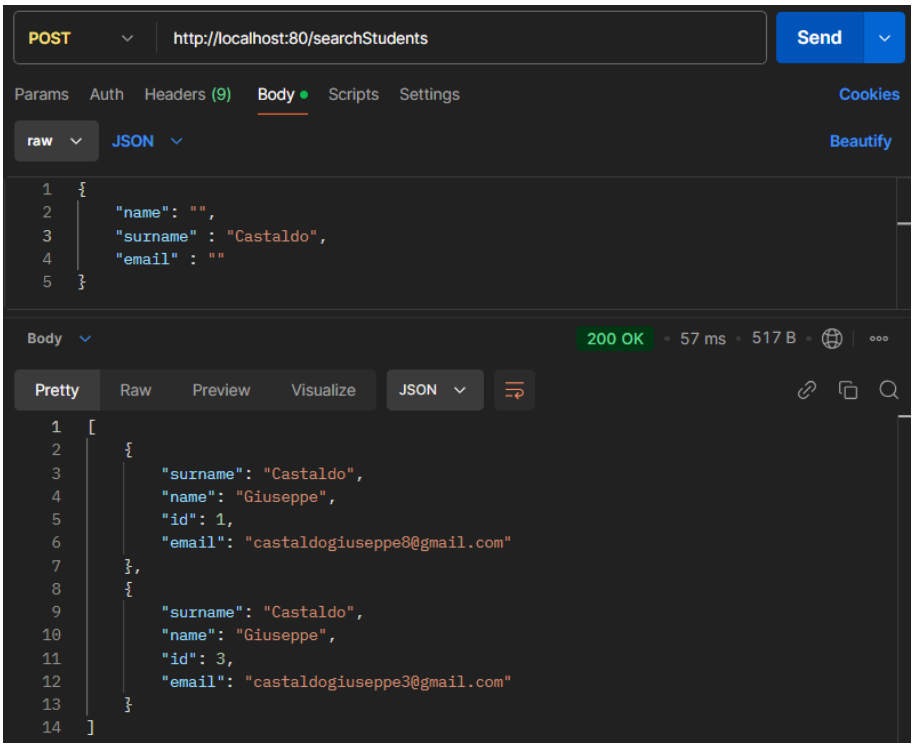
TC04 - Verifica ricerca per nome e cognome



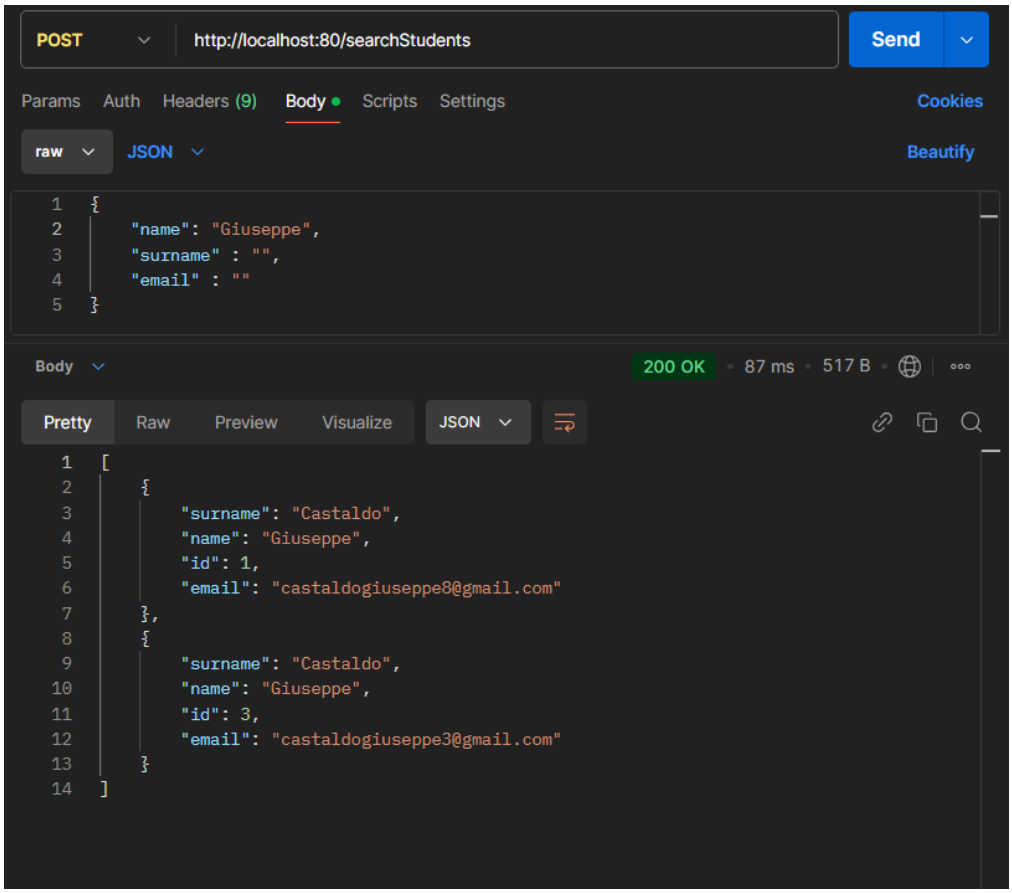
TC05 - Verifica che la ricerca ritorni un risultato vuoto quando i parametri sono vuoti



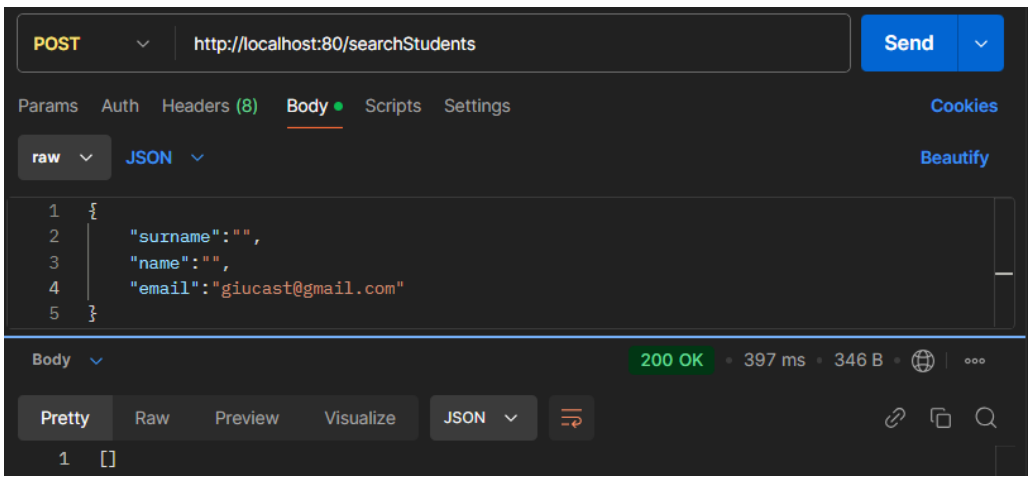
TC06 – Verifica ricerca con solo cognome e email vuoti



TC07 - Verifica ricerca con solo nome e email vuoti



TC08 - Verifica che la ricerca non restituisca alcun risultato per email non esistente.



TC09 – Verifica gestione di una ricerca con nome e cognome vuoti.