



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

Progetto di gruppo

Progetto SAD - A13

Anno Accademico 2024/25

Relatore

Prof. Anna Rita Fasolino

Candidati

Vincenzo Luigi Bruno matr. M63001670

Salvatore Cangiano matr. M63001647

Cristina Carleo

Abstract

Questo documento descrive lo sviluppo e l'implementazione di un sistema di gestione del profilo utente e di funzionalità social in un'applicazione di gioco. Il sistema offre agli utenti la possibilità di personalizzare il proprio profilo, selezionare avatar, modificare la descrizione e visualizzare le proprie statistiche. È stato implementato un servizio di notifiche che informa gli utenti su eventi significativi, nel nostro caso il raggiungimento di un achievement e l'aggiunta di nuovi follower. Sono state quindi introdotte funzionalità social che consentono agli utenti di seguire altri giocatori e visualizzare i loro progressi, creando una nuova pagina utente visibile agli amici. Il sistema, sviluppato utilizzando Spring MVC come framework implementativo e MySQL come database per il microservizio T23, offre un'interfaccia user-friendly e scalabile. Postman è stato utilizzato per testare le API, mentre Notion ha supportato la gestione degli sprint e la pianificazione delle fasi di progetto. Il sistema mira a migliorare l'interazione tra gli utenti, offrendo loro nuove funzionalità per la gestione del proprio profilo e delle relazioni sociali nel contesto del gioco.

Puoi trovare la nostra versione qui documentata al link: <https://github.com/iris-cmd22/A13>
[1] [4]

Contents

Abstract	i
1 Obiettivi del progetto	1
1.1 Punto di partenza	1
1.1.1 Diagramma dei Casi d'uso	1
1.1.2 Diagramma Architetturale dei componenti	2
1.1.3 Diagramma di Deploy	4
1.2 I nostri obiettivi	4
1.2.1 Nuovi requisiti assegnati: Task R1	4
1.2.2 Visione del sistema	5
1.2.3 Processo di Sviluppo	5
1.2.4 Studio di fattibilità	6
1.2.5 Diagramma di Contesto	6
2 Analisi dei requisiti	7
2.1 Analisi dei requisiti	7
2.1.1 Analisi preliminare dei requisiti	7
2.1.2 Requisiti Funzionali	8
2.1.3 Requisiti non funzionali	9
2.1.4 Requisiti sui dati	9
2.1.5 Utility Tree	10
2.2 Casi d'uso	11
2.2.1 Diagramma dei casi d'uso	11
2.2.2 Storie Utente	11
2.2.3 Criteri di Accettazione	13
2.3 Diagrammi di Interazione	15
2.3.1 Diagrammi di sequenza (alto livello)	15
3 Analisi dell'Impatto	17
3.1 Impatto stimato Selezione Foto Profilo	17
3.2 Impatto stimato Modifica Descrizione Profilo	17
3.3 Impatto stimato Visualizzazione Achievement	18
3.4 Impatto stimato Visualizzazione Statistiche	18
3.5 Impatto stimato Notifica Nuovo Achievement	18
3.6 Impatto stimato Gestione Notifiche	18
3.7 Impatto stimato Seguire un Altro Giocatore	19
3.8 Impatto stimato Visualizzazione Follower e Seguiti	19
3.9 Impatto stimato Visualizzazione Profili Giocatori	19
3.10 Impatto stimato Notifica Nuovo Follower	19
3.11 Impatto stimato Visualizzazione Token Guadagnati	19
4 Progetto dettagliato della soluzione	20
4.1 Composite Diagram dell'Architettura	20
4.2 Spring MVC	20
4.2.1 Che cos'è Spring Boot	21
4.3 Seconda iterazione	21
4.4 Feature 1: Edit Profile	21
4.4.1 GuiController.java	21
4.4.2 Package Model in T23	23
4.4.3 Modello dei dati	23
4.4.4 Jakarta Persistence o Java Persistence API (JPA)	24
4.4.5 UserProfile	24
4.4.6 UserProfileRepository	25
4.4.7 User Service in T23	25
4.4.8 Controller in T23	28
4.4.9 Endpoint editProfile	28

4.4.10	Integrazione con T5	28
4.4.11	ModelUserProfile in T5	29
4.4.12	T23Service in T5	30
4.4.13	UserProfileService	31
4.4.14	Model html: profile.html e Edit_Profile.html	33
4.4.15	Integrazione con PageBuilder	33
4.4.16	Diagrammi di interazione (dettaglio)	35
4.5	Feature 2: Notification Service	37
4.5.1	Modello dei dati	37
4.5.2	Package Model in T23	38
4.5.3	NotificationRepository	38
4.5.4	NotificationService in T23	39
4.5.5	REST API Endpoints in T23	41
4.5.6	GuiController in T5	46
4.5.7	T23Service in T5	47
4.5.8	Package Model in T5	49
4.5.9	Diagrammi di sequenza (dettaglio)	49
4.6	Terza iterazione	55
4.7	Feature 3: Social, follower e following	55
4.8	Issue: Partite giocate non salvate nel database	56
5	Struttura del progetto modificato	63
5.1	Nuovi Endpoint	63
5.2	Package diagram complessivo	66
5.3	Composite diagram complessivo	67
6	Strategia di test	70
6.1	GET <i>/get-followers</i>	70
6.1.1	Test Case 1.1: Utente con follower	70
6.1.2	Test Case 1.2: Utente non esistente	70
6.1.3	Test Case 1.3: Utente esistente senza follower	71
6.2	GET <i>/notifications</i>	71
6.2.1	Test Case 2.1: Utente senza notifiche	72
6.2.2	Test Case 2.2: Utente con notifiche	72
6.3	POST <i>/update-profile</i>	72
6.3.1	Test Case 3.1: Aggiornamento profilo successo	73
6.3.2	Test Case 3.2: Utente non esistente	73
6.4	POST <i>/new-notification</i>	74
6.4.1	Test Case 4.1: Creazione notifica successo	74
6.4.2	Test Case 4.2: Parametro mancante	74
6.4.3	Test Case 4.3: Utente inesistente	75
6.5	POST <i>/update-notification</i>	75
6.5.1	Test Case 5.1: Lettura corretta	76
6.5.2	Test Case 5.2: Notifica non esistente	76
6.6	DELETE <i>/delete-notification</i>	77
6.6.1	Test Case 6.1: Cancellazione corretta	77
6.6.2	Test Case 6.2: Email non esistente	77
6.6.3	Test Case 6.3: ID notifica non valido	78
6.6.4	Test Case 6.4: ID notifica non esistente	78
6.7	POST <i>/follow/{playerId}</i>	79
6.7.1	Test Case 7.1: Follow con successo	79
7	Potenziali sviluppi futuri	80
7.1	Ecosistema Teams	80
7.2	Sistema di Assignment e Riconoscimenti	80
7.3	Sistema Token	80
7.4	NotificationService	80
7.4.1	"How To": Integrare Notification Service	81
7.4.2	API Disponibili	81
7.5	Leaderboard	82
7.6	Sistema Statistiche	82

Chapter 1

Obiettivi del progetto

Il progetto si inserisce nel contesto del progetto ERASMUS ENACTEST, un'iniziativa triennale che mira a valorizzare l'importanza del testing nel percorso formativo universitario. L'approccio innovativo scelto è quello della **gamification**, concretizzato nello sviluppo del gioco "Man vs Automated Testing Tools challenges", dove gli studenti possono competere contro strumenti di testing automatizzato nella creazione di test JUnit.

1.1 Punto di partenza

Il team è partito dalla versione di A13 del progetto, che è stato strutturato secondo un'architettura a **microservizi**, una scelta progettuale che garantisce maggiore agilità nello sviluppo e una migliore scalabilità del sistema. I servizi principali includono la parte admin con relativa gestione delle classi da testare e degli achievement (**T1**), l'autenticazione degli utenti (**T23**), un repository per i dati di gioco (**T4**), interfacce front-end per l'avvio e lo svolgimento delle partite (**T5 e T6**), un servizio di compilazione ed esecuzione dei test (**T7**), e i servizi dedicati ai robot EvoSuite (**T8**) e Randoop (**T9**).

Successivamente, è stato effettuato un importante refactoring nella versione A13 che ha portato all'unificazione dei servizi T5 e T6, riorganizzando il codice in una struttura modulare più efficiente. Questa riorganizzazione ha portato alla definizione di pacchetti distinti con responsabilità ben definite:

- **Components**: dedicato agli strumenti per la creazione e gestione delle pagine web
- **Interfaces**: ottimizzato per la gestione delle chiamate ai servizi REST
- **Game**: focalizzato sulla logica di gioco e la gestione delle partite
- **Model**: responsabile della gestione dei dati e dei modelli
- **T5**: dedicato a servizi relativi all'interfaccia

L'integrazione tra questi servizi è stata realizzata attraverso il **Gateway Pattern**, che prevede due componenti principali: un **UI Gateway** che funge da punto d'accesso unico per i client, e un **API Gateway** che gestisce l'instradamento delle richieste e l'autenticazione degli utenti. Questa architettura consente una gestione efficiente delle richieste, garantendo al contempo la sicurezza attraverso un sistema di autenticazione basato su token.

La struttura modulare del sistema permette ai team di sviluppo di lavorare in autonomia sui singoli servizi, facilitando la manutenzione e l'integrazione di nuove funzionalità. Inoltre, la possibilità di scalare individualmente i servizi consente di ottimizzare le risorse in base alle effettive necessità del sistema.

1.1.1 Diagramma dei Casi d'uso

Il diagramma dei casi d'uso relativo al task T2-3 rappresenta il sistema di gestione utenti con le funzionalità base necessarie per l'applicazione. Mostra due tipi di utenti: il **Player non ancora registrato**, che può creare un account (sia tramite registrazione standard che via Facebook), e il **Player Registrato** che può effettuare login, logout e gestire eventuali problemi con la password. Il sistema comunica anche con un servizio di posta elettronica esterno per inviare mail di conferma registrazione e token per il reset della password.

Questa è stata la struttura iniziale, che è stata utilizzata come base per sviluppare le nostre funzionalità.

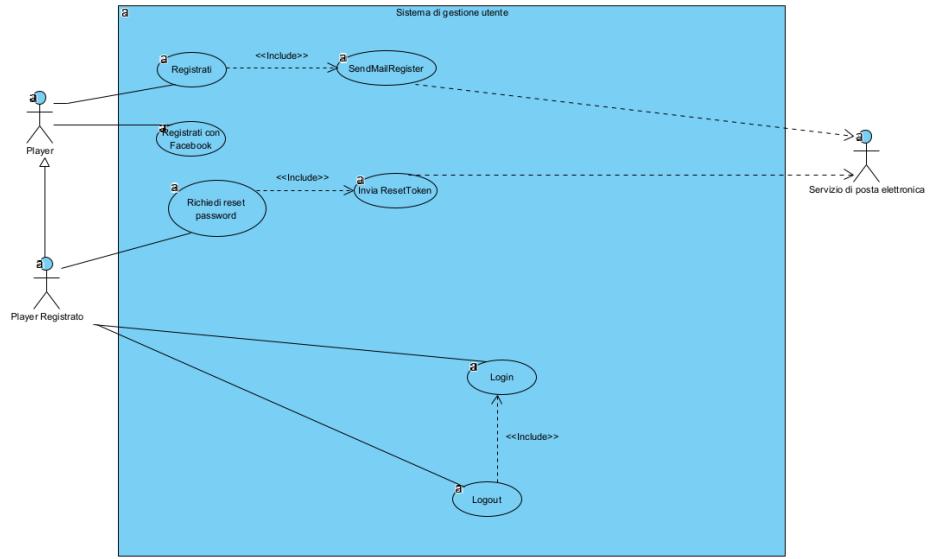


Figure 1.1: Diagramma dei casi d'uso Iniziale

1.1.2 Diagramma Architetturale dei componenti

Il Composite Diagram della versione A13 fornisce una panoramica chiara delle responsabilità e delle relazioni tra i diversi task di sviluppo. Questo diagramma evidenzia la **struttura modulare** del sistema e *come i vari componenti interagiscono tra loro* per fornire le funzionalità complete dell'applicazione.

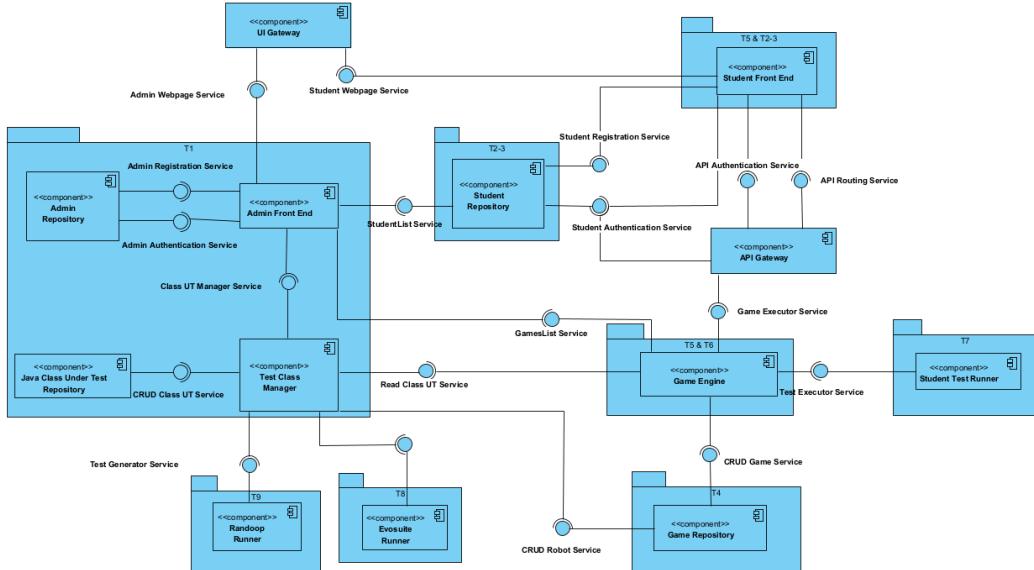


Figure 1.2: Composite Diagram versione A13

Il refactoring della versione A13 ha introdotto una nuova organizzazione del backend in T5 e T6, suddivisa in cinque componenti principali, ognuno con responsabilità ben definite:

1. **Interface:** implementa un Service Manager centralizzato per la gestione delle chiamate REST, facilitando l'aggiunta di nuovi task e l'integrazione dei servizi esistenti, standardizzando la comunicazione tra i diversi

componenti del sistema.

2. **Component**: trasforma la gestione delle pagine web da un controller monolitico a un sistema modulare, separando la logica di business dalla visualizzazione e introducendo un *controller specifico per routing* e gestione delle sessioni.
3. **Game**: sostituisce il precedente GameEngine frammentato con una struttura più robusta basata su *GameLogic*, semplificando l'implementazione di nuove modalità di gioco e la gestione delle partite attive.
4. **Model**: gestisce i dati e i modelli dell'applicazione, definendo le strutture dati fondamentali e implementando la logica di persistenza.
5. **T5**: fornisce servizi specializzati per l'interfaccia utente, gestendo l'interazione tra frontend e backend.

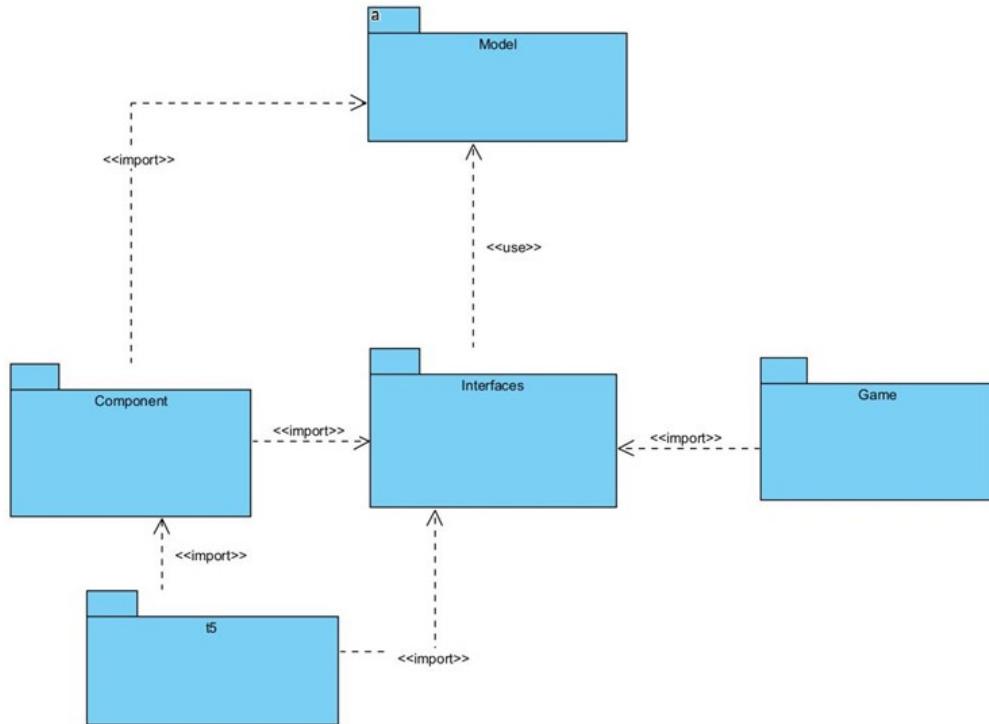


Figure 1.3: Package Diagram versione A13_refactoring

Per maggiori informazioni si prega di far riferimento alla documentazione relativa la versione dell'applicazione A13 e A13_refactoring.

1.1.3 Diagramma di Deploy

La configurazione di deployment rimane invariata rispetto alla documentazione A10-2024, garantendo stabilità e compatibilità con l'infrastruttura esistente.

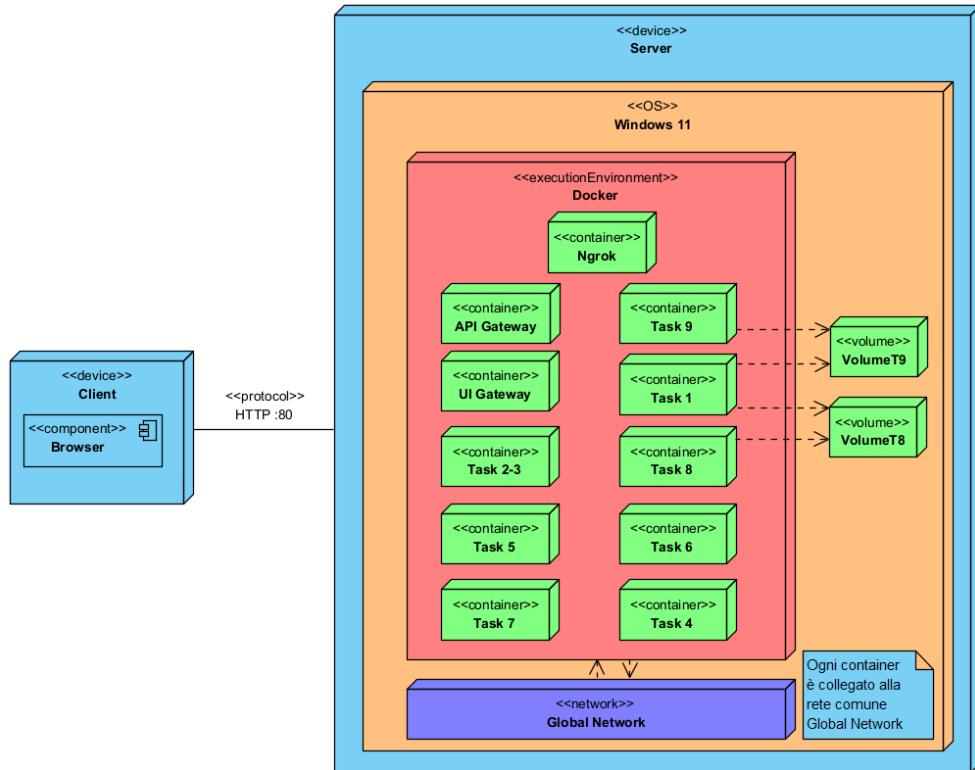


Figure 1.4: Deployment Diagram versione A10-2024

L'architettura di deployment utilizza Ngrok come *servizio di tunneling*. Questa configurazione è stata testata e validata.

Per ulteriori dettagli implementativi e specifiche tecniche, si rimanda alla documentazione completa delle versioni A13, A13_refactoring e A10-2024.

1.2 I nostri obiettivi

1.2.1 Nuovi requisiti assegnati: Task R1

Migliorare il profilo Giocatore, prevedendo:

- Strutturare la Pagina del profilo in sezioni separate per contenere prioritariamente:
 - **Informazioni del profilo**, con possibilità di *editing*.
 - **Statistiche del giocatore**, ossia:
 - * Partite giocate per tipologia;
 - * Partite vinte.
 - **Achievement ottenuti**, ossia *Badge* per segnalare il raggiungimento di una soglia di una statistica.
 - **Premi (Token)** guadagnati, eventualmente, in base a:
 - * Missioni giornaliere svolte;
 - * Altre regole definite.
 - **Elenco di altri giocatori/follower**.

Impatto Stimato

T5-T23 : Rivedere in particolare i *Model del Player* in T23.

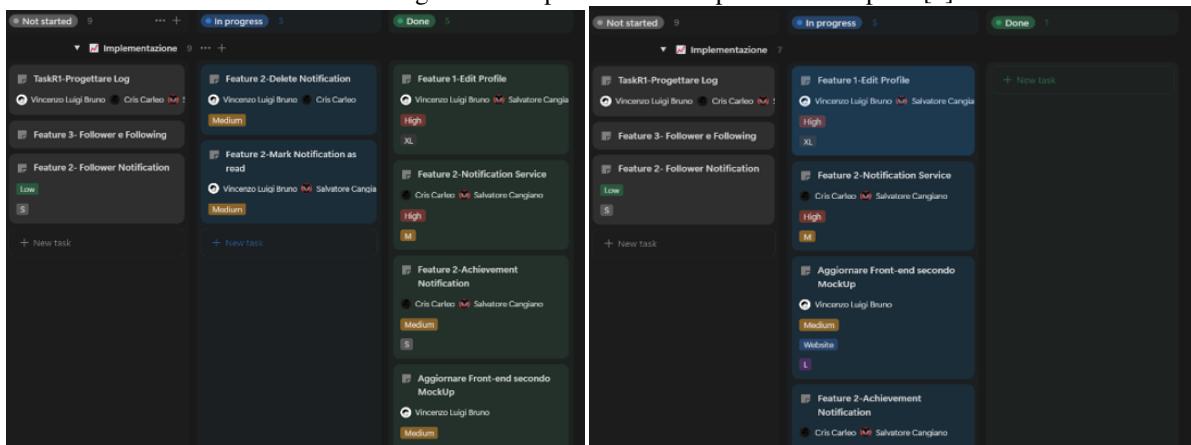
1.2.2 Visione del sistema

Mission

Elevare la qualità del software trasformando ogni sfida in opportunità, dove ogni bug diventa una vittoria da condividere e celebrare insieme al team.

1.2.3 Processo di Sviluppo

Per lo sviluppo del nostro software abbiamo adottato un *approccio agile* basato sulle iterazioni. La nostra metodologia di lavoro si è basata su *sprint settimanali o bisettimanali*, durante i quali i componenti del gruppo si sono concentrati su un insieme di funzionalità specifiche. Ogni sprint è stato preceduto da una pianificazione, in cui sono stati definiti gli obiettivi dell'iterazione e le attività necessarie per raggiungerli. La pianificazione è stata gestita attraverso un **template Notion** dedicato, aggiornato ad ogni iterazione. Questo template ha permesso al team di tenere traccia delle attività da svolgere e delle priorità definite per ciascuno sprint.[2]



Le modifiche e i progressi sono stati documentati giorno per giorno. Per il diario di bordo, è stato utilizzato un quaderno fisico, successivamente organizzato e suddiviso in quattro documenti: tre dedicati alle funzionalità implementate nei rispettivi sprint e uno specifico per documentare l'iterazione dello sprint stesso. Tale organizzazione ha garantito un livello elevato di trasparenza e chiarezza nell'archiviazione delle informazioni.

Nel nostro planning, abbiamo deciso di strutturare lo sviluppo in due iterazioni principali:

Prima iterazione: implementazione di una pagina dove editare il profilo e di un servizio di notifica consultabile direttamente dal profilo del giocatore. **Seconda iterazione:** implementazione delle funzionalità social, ossia la possibilità di seguire un altro giocatore e ricevere una notifica quando qualcuno ti segue. Durante lo sviluppo, è stato fatto ampio uso della pratica del **pair programming**, che ha previsto il lavoro a due su un singolo codice. Questa pratica è stata utilizzata nell'implementazione di tutte le feature, consentendo di migliorare la qualità del codice e di ridurre il numero di errori, poiché ogni riga di codice è stata esaminata da almeno due persone. Inoltre, il **pair programming** ha favorito la diffusione della conoscenza e delle competenze tra i membri del team, soprattutto grazie alla realizzazione della maggior parte delle implementazioni attraverso incontri di persona, che hanno facilitato la comunicazione del team.

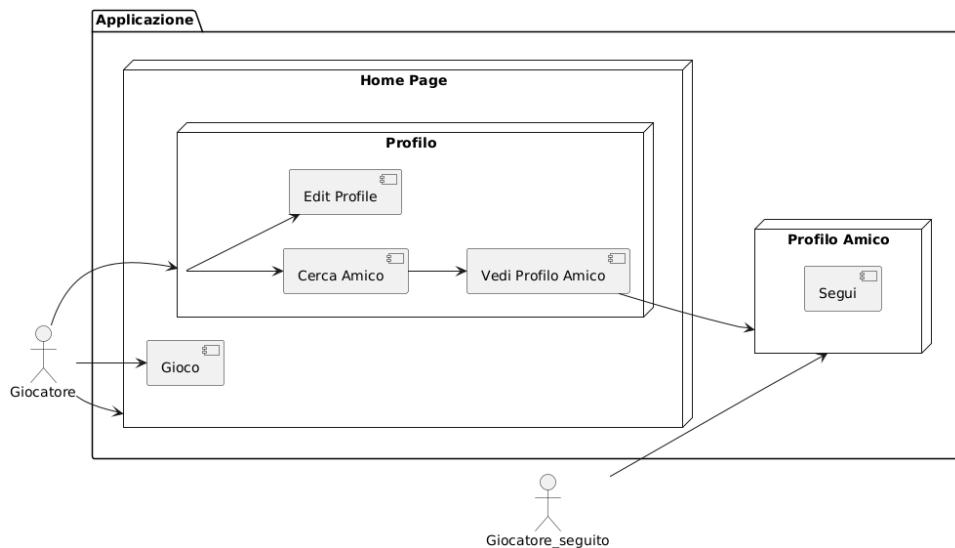
1.2.4 Studio di fattibilità

Analisi SWOT

Punti di Forza	Debolezze
<ul style="list-style-type: none"> • Percorso formativo strutturato • Feedback immediato sull'apprendimento • Sistema di mentorship integrato • Gamification dell'educazione 	<ul style="list-style-type: none"> • Necessità di aggiornamento continuo dei contenuti • Complessità nella validazione delle competenze • Richiesta di risorse per mentorship • Tempo necessario per la progressione • Mantenimento della qualità formativa
Opportunità	Minacce
<ul style="list-style-type: none"> • Community di apprendimento • Knowledge sharing strutturato • Identificazione talenti 	<ul style="list-style-type: none"> • Superficialità nell'apprendimento • Focus su badge invece che competenze • Obsolescenza dei contenuti

Table 1.1: Analisi SWOT

1.2.5 Diagramma di Contesto



Chapter 2

Analisi dei requisiti

2.1 Analisi dei requisiti

Dopo aver introdotto lo stato passato del sistema, possiamo passare all'analisi dei requisiti assegnati. Questa analisi mira principalmente a raffinare e chiarire i requisiti funzionali e non funzionali che è stato possibile estrarre dal task assegnatoci. Anticipiamo che il nostro task ha modificato principalmente T23 e T5, ma l'impatto effettivo verrà analizzato successivamente in questo documento.

2.1.1 Analisi preliminare dei requisiti

Di seguito è riportata una classificazione preliminare dei requisiti estratti:

id	requisiti
1	Il Giocatore può selezionare una foto profilo tra quelle disponibili di default.
2	Il Giocatore può modificare la descrizione del proprio profilo.
3	Il Giocatore può scegliere tra tre modalità di visualizzazione degli achievement: tutti gli achievement, achievement sbloccati, o achievement da sbloccare.
4	Il Giocatore può consultare le proprie statistiche.
5	Il Sistema invia una notifica al Giocatore ogni volta che viene sbloccato un nuovo achievement.
6	Il Giocatore può visualizzare e cancellare le notifiche ricevute.
7	Il Giocatore può seguire un altro giocatore.
8	Il Giocatore può consultare l'elenco dei follower e degli utenti seguiti (following).
9	Il Giocatore può visualizzare il profilo degli utenti seguiti (following) e degli follower.
10	Il Sistema notifica il Giocatore ogni volta che viene aggiunto un nuovo follower.
11	Il Giocatore può visualizzare i token guadagnati.

Table 2.1: Requisiti preliminari del Task

Abbiamo anche riportato un breve glossario per stabilire in maniera non equivoca il significato di certi vocaboli che verranno utilizzati nel documento:

Vocabolo	Spiegazione
Avatar	Immagine profilo del giocatore, composta da foto disponibili di default.
Bio	Breve descrizione che il giocatore possa personalizzare.
Achievement	Ricompensa sbloccata come conseguenza del raggiungimento di un dato obiettivo.
Token	Ricompensa ottenuta come ricompensa del superamento di una o più missioni giornaliere.

Table 2.2: Requisiti preliminari del Task

2.1.2 Requisiti Funzionali

ID	Requisito	Origine
RF01	Il sistema deve offrire allo studente una funzionalità per visualizzare gli avatar disponibili nel suo profilo.	1
RF02	Il sistema deve offrire allo studente una funzionalità per selezionare gli avatar disponibili nel suo profilo.	1
RF03	Il sistema deve offrire allo studente una funzionalità per modificare la descrizione del suo profilo.	2
RF04	Il sistema deve offrire allo studente una funzionalità per selezionare una vista sugli achievement fra tutti gli achievement, achievement sbloccati, o achievement da sbloccare.	3
RF05	Il sistema deve offrire allo studente una funzionalità per visualizzare le statistiche disponibili nel suo profilo.	4
RF06	Il sistema deve inviare una notifica allo studente ogni volta che ottiene un achievement.	5
RF07	Il sistema deve offrire allo studente una funzionalità per visualizzare le notifiche ricevute.	6
RF08	Il sistema deve offrire allo studente una funzionalità per cancellare le notifiche ricevute.	6
RF09	Il sistema deve offrire allo studente una funzionalità per marcare come lette le notifiche ricevute.	6
RF10	Il sistema deve offrire allo studente una funzionalità per visualizzare un altro giocatore.	7
RF11	Il sistema deve offrire allo studente una funzionalità per seguire un altro giocatore.	7
RF12	Il sistema deve offrire allo studente una funzionalità per visualizzare un elenco dei propri follower.	8
RF13	Il sistema deve offrire allo studente una funzionalità per visualizzare un elenco dei propri following.	8
RF14	Il sistema deve offrire allo studente una funzionalità per visualizzare un elenco dei follower di un utente.	9
RF15	Il sistema deve offrire allo studente una funzionalità per visualizzare un elenco dei followings di un utente.	9
RF16	Il sistema deve inviare una notifica allo studente ogni volta che ottiene un follower.	10
RF17	Il sistema deve offrire allo studente una funzionalità per visualizzare i token guadagnati.	11

Table 2.3: Requisiti funzionali

2.1.3 Requisiti non funzionali

ID	Vincolo/Requisito	Origine
RNF01	L'interfaccia utente per la gestione degli achievement deve essere accessibile e facilmente navigabile per l'Admin	1,2
RNF02	L'applicazione viene rilasciata su Docker	
RNF03	Solo l'utente deve poter modificare la sua pagina	

Table 2.4: Tabella dei vincoli e delle loro origini

2.1.4 Requisiti sui dati

ID	Requisito	Origine
RD01	Un avatar è costituito da un'immagine e un titolo	1
RD02	Una descrizione o bio è costituito da una stringa di testo	2
RD03	Un achievement è costituito di un nome e un requisito da soddisfare	3
RD04	Una notifica di acquisizione di un achievement deve includere un messaggio e il nome dell'achievement acquisito	5
RD05	Una notifica è costituita da una stringa di testo	6
RD06	Un token è costituito da un id e un numero che ne specifica il valore numerico	11
RD07	Una partita è costituita da un file con delle metriche	

Table 2.5: Tabella dei requisiti e delle loro origini

2.1.5 Utility Tree

Quality Attribute	Attribute Refinement	ASR Scenario
Usability	Personalizzazione profilo	Il giocatore può selezionare una foto profilo tra quelle disponibili di default con un'interfaccia intuitiva. (H, M)
Usability	Personalizzazione profilo	Il giocatore può modificare la descrizione del proprio profilo con un editor semplice e user-friendly. (H, M)
Usability	Visualizzazione achievements	Il giocatore può scegliere tra tre modalità di visualizzazione: tutti gli achievement , achievement sbloccati , e achievement da sbloccare attraverso un'interfaccia chiara. (M, M)
Usability	Accesso alle statistiche	Il giocatore può consultare le proprie statistiche organizzate in modo leggibile e facilmente navigabile. (H, M)
Usability	Notifica automatica	Il sistema invia una notifica ogni volta che viene sbloccato un nuovo achievement , visibile nella dashboard principale. (H, L)
Modificabilità	Gestione notifiche	Il giocatore può visualizzare e cancellare le notifiche ricevute con un sistema di aggiornamento flessibile. (H, M)
Usability	Interazione con giocatori	Il giocatore può seguire un altro giocatore con un'interfaccia chiara che conferma immediatamente l'azione. (H, M)
Usability	Visualizzazione follower	Il giocatore può consultare l'elenco dei follower e degli utenti seguiti (following) in modo organizzato. (M, M)
Usability	Profilo utente	Il giocatore può visualizzare i profili dei follower e degli utenti seguiti (following) con una navigazione intuitiva. (M, M)
Modificabilità	Notifica nuovi follower	Il sistema notifica il giocatore ogni volta che viene aggiunto un nuovo follower in modo non invasivo e facilmente configurabile. (H, L)
Usability	Visualizzazione token	Il giocatore può visualizzare i token guadagnati con un'interfaccia grafica chiara e comprensibile. (H, M)

Table 2.6: Raffinamento dei requisiti

2.2 Casi d'uso

Ora che abbiamo definito nel dettaglio tutti i requisiti funzionali e non funzionali, possiamo passare alla stesura dei casi d'uso. Per l'analisi dei casi d'uso sono stati utilizzati diversi diagrammi, molti dei quali di alto livello.

2.2.1 Diagramma dei casi d'uso

E' stato deciso di costruire un diagramma dei casi d'uso dei task assegnati. Avendo trattato in precedenza come il sistema gestisce l'entità dei giocatori, riprenderemo il diagramma dei casi d'uso e lo estenderemo con le nuove funzionalità estratte dai requisiti funzionali. Nel diagramma vengono riportati in rosso i casi d'uso introdotti in questa iterazione di progetto e in giallo quelli implementati parzialmente, il diagramma dei casi d'uso è il seguente:

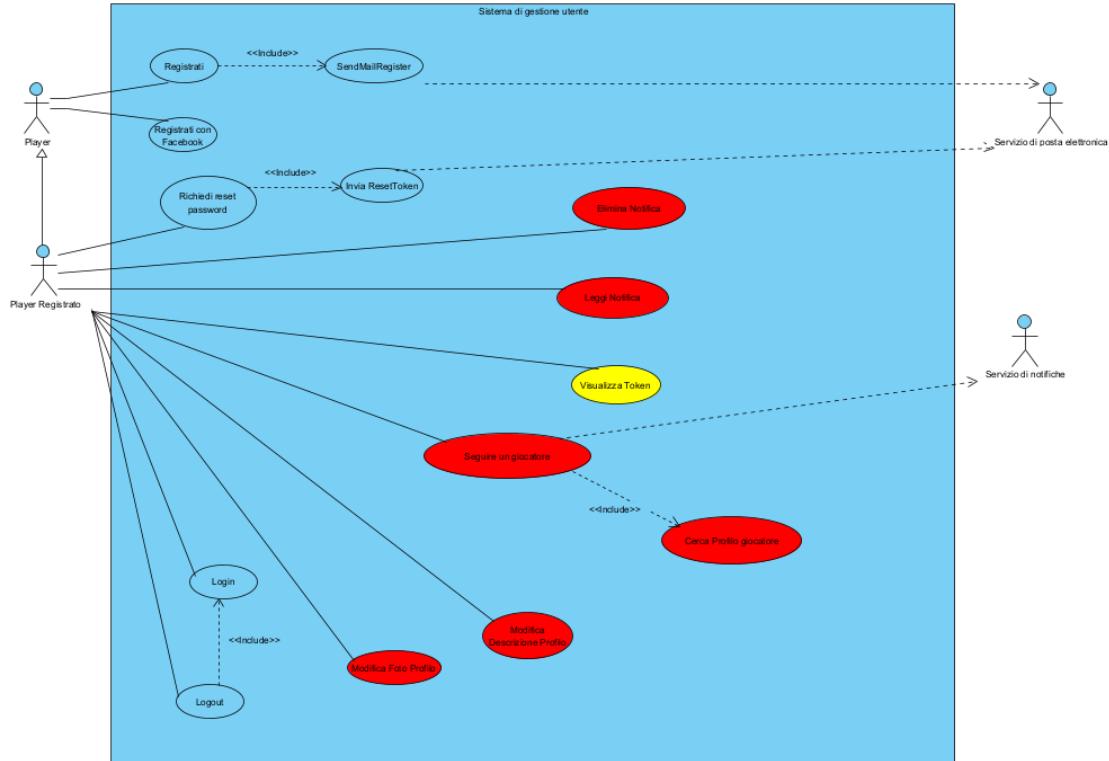


Figure 2.1: Diagramma dei casi d'uso dopo analisi dei requisiti

2.2.2 Storie Utente

Un'altra metodologia che abbiamo voluto adottare è quella delle storie utente, brevi descrizioni di funzionalità o requisiti del software, espresse dal punto di vista dell'utente finale. Nel nostro caso è stato ritenuto utile specificarle in questa fase di progetto poiché hanno aiutato il team a comprendere chi sono gli utenti finali e quali sono i loro obiettivi. Inoltre mantengono ancora alto il livello di astrazione, evitando di concentrarsi sui dettagli tecnici, caratteristica che permette ad un nuovo lettore di comprendere al meglio le nuove funzionalità del sistema. Le storie utente create sono le seguenti:

1. Foto profilo

- **COME:** un giocatore,
- **DESIDERO:** selezionare una foto profilo tra quelle disponibili di default,
- **IN MODO CHE:** possa personalizzare visivamente il mio profilo.

2. Modifica descrizione profilo

- **COME:** un giocatore,
- **DESIDERO:** modificare la descrizione del mio profilo,
- **IN MODO CHE:** possa esprimere chi sono o condividere informazioni con altri utenti.

3. Modalità visualizzazione achievement

1. Tutti gli achievement

- **COME:** un giocatore,
- **DESIDERO:** visualizzare tutti gli achievement,
- **IN MODO CHE:** possa monitorare i miei progressi e concentrarmi sugli obiettivi ancora da raggiungere.

2. Achievement sbloccati

- **COME:** un giocatore,
- **DESIDERO:** visualizzare gli achievement sbloccati,
- **IN MODO CHE:** possa monitorare i miei progressi e concentrarmi sugli obiettivi ancora da raggiungere.

3. Achievement da sbloccare

- **COME:** un giocatore,
- **DESIDERO:** visualizzare gli achievement non sbloccati,
- **IN MODO CHE:** possa monitorare i miei progressi e concentrarmi sugli obiettivi ancora da raggiungere.

4. Consultazione statistiche

- **COME:** un giocatore,
- **DESIDERO:** consultare le mie statistiche,
- **IN MODO CHE:** possa valutare le mie prestazioni e migliorare il mio gioco.

5. Notifiche nuovi achievement

- **COME:** un giocatore,
- **DESIDERO:** ricevere una notifica quando sblocco un nuovo achievement,
- **IN MODO CHE:** possa essere subito informato dei miei successi.

6. Gestione notifiche ricevute

- **COME:** un giocatore,
- **DESIDERO:** visualizzare e cancellare le notifiche ricevute,
- **IN MODO CHE:** possa mantenere organizzato il mio spazio notifiche.

7. Seguire altri giocatori

- **COME:** un giocatore,
- **DESIDERO:** seguire altri giocatori,
- **IN MODO CHE:** possa rimanere aggiornato sui loro progressi e attività.

8. Consultazione follower e following

- **COME:** un giocatore,
- **DESIDERO:** consultare l'elenco dei miei follower e degli utenti che seguono,
- **IN MODO CHE:** possa vedere chi è interessato al mio profilo e monitorare le mie connessioni.

9. Visualizzazione profili

1. Follower

- **COME:** un giocatore,
- **DESIDERO:** visualizzare il profilo degli utenti che mi seguono,
- **IN MODO CHE:** possa conoscere meglio le persone con cui interagisco.

2. Following

- **COME:** un giocatore,
- **DESIDERO:** visualizzare il profilo degli utenti che seguo,
- **IN MODO CHE:** possa conoscere meglio le persone con cui interagisco.

10. Notifica nuovi follower

- **COME:** un giocatore,
- **DESIDERO:** ricevere una notifica ogni volta che un nuovo follower mi aggiunge,
- **IN MODO CHE:** possa essere consapevole della crescita della mia rete.

11. Visualizzazione token guadagnati

- **COME:** un giocatore,
- **DESIDERO:** visualizzare i token guadagnati,
- **IN MODO CHE:** possa monitorare le mie ricompense e utilizzarle strategicamente.

Abbiamo incluso anche una stima preliminare dell'importanza attribuita a ciascuna storia utente, poiché esse rappresentano funzionalità chiave che si intendono implementare nell'ambito di questa evoluzione del progetto.

Storia Utente	Stima	Priorità
1: Foto Profilo	5	Alta
2: Modifica descrizione profilo	3	Alta
3: Modalità visualizzazione achievement	2	Alta
4: Consultazione statistiche	2	Alta
5: Notifiche nuovi achievement	8	Alta
6: Gestione notifiche ricevute	5	Alta
7: Seguire altri giocatori	8	Media
8: Consultazione follower e following	3	Media
9: Visualizzazione profili	5	Media
10: Notifica nuovi follower	5	Media
11: Visualizzazione token guadagnati	1	Bassa

2.2.3 Criteri di Accettazione

Durante la fase di analisi dei requisiti, è buona pratica stabilire criteri di accettazione che definiscano chiaramente le aspettative relative a ciascuna funzionalità o modifica del sistema. Per questa ragione, abbiamo deciso di redigere un elenco di criteri di accettazione, che ci permetterà di utilizzarlo in una fase avanzata del progetto per la verifica e la validazione delle nuove funzionalità.

GIVEN	WHEN	THEN
1. Un Giocatore autenticato con un profilo con una foto predefinita	Il Giocatore sceglie una delle foto profilo disponibili di default	La foto profilo del Giocatore viene aggiornata con l'immagine selezionata.
2. Un Giocatore autenticato con una descrizione profilo esistente o vuota	Il Giocatore modifica la descrizione nel proprio profilo	La nuova descrizione viene salvata e mostrata nel profilo del Giocatore.
3. Un Giocatore autenticato con almeno un achievement associato	Il Giocatore seleziona una delle tre viste disponibili: tutti gli achievement	Il sistema mostra al Giocatore la lista di achievement corrispondente alla vista scelta.
4. Un Giocatore autenticato con almeno un achievement associato	Il Giocatore seleziona una delle tre viste disponibili: achievement sbloccati	Il sistema mostra al Giocatore la lista di achievement corrispondente alla vista scelta.
5. Un Giocatore autenticato con almeno un achievement associato	Il Giocatore seleziona una delle tre viste disponibili: achievement da sbloccare	Il sistema mostra al Giocatore la lista di achievement corrispondente alla vista scelta.
6. Un Giocatore autenticato con statistiche registrate	Il Giocatore accede alla sezione delle statistiche	Il sistema mostra al Giocatore le sue statistiche aggiornate.
7. Un Giocatore autenticato che ha appena soddisfatto i criteri per sbloccare un achievement	L'achievement viene sbloccato	Il sistema invia una notifica al Giocatore per informarlo del nuovo achievement.
8. Un Giocatore autenticato che ha ricevuto una o più notifiche	Il Giocatore visualizza o elimina una notifica	Il sistema aggiorna lo stato della notifica (le visualizzate non appaiono più come "nuove" e le eliminate non sono più visibili).
9. Un Giocatore autenticato che vuole seguire un altro Giocatore	Il Giocatore ricerca un altro utente per email e lo segue	Il sistema registra il nuovo collegamento e aggiorna l'elenco dei seguiti (followed) del Giocatore.
10. Un Giocatore autenticato con uno o più follower e/o seguiti	Il Giocatore accede alla sezione relativa ai follower e ai seguiti	Il sistema mostra l'elenco completo dei follower e degli utenti seguiti dal Giocatore.
11. Un Giocatore autenticato con uno o più seguiti e/o uno più follower	Il Giocatore accede al profilo di un utente seguito	Il sistema mostra il profilo dell'utente selezionato.
12. Un Giocatore autenticato con uno o più seguiti e/o uno più follower	Il Giocatore accede al profilo di un utente follower	Il sistema mostra il profilo dell'utente selezionato.
13. Un Giocatore autenticato che riceve un nuovo follower	Un altro utente inizia a seguire il Giocatore	Il sistema invia una notifica per informare il Giocatore del nuovo follower.
14. Un Giocatore autenticato che ha guadagnato uno o più token	Il Giocatore accede alla sezione dei token	Il sistema mostra il totale dei token guadagnati dal Giocatore.

2.3 Diagrammi di Interazione

2.3.1 Diagrammi di sequenza (alto livello)

Per illustrare il funzionamento delle nuove funzionalità implementate nel sistema, abbiamo utilizzato dei sequence diagram di alto livello. Questi diagrammi sono stati progettati per rappresentare in modo visivo e conciso l'interazione tra gli attori (come gli utenti o il sistema) e i componenti del sistema stesso.

Essendo in una fase ancora iniziale dell'evoluzione del progetto questi diagrammi forniscono una panoramica delle principali interazioni senza entrare in troppi dettagli tecnici, i quali verranno rifiniti in una fase avanzata e di dettaglio.

Ci siamo soffermati su alcune delle funzionalità più importanti che verranno sviluppate, ossia La ricezione di una **notifica** a seguito del conseguimento di un **achievement** e Seguire un nuovo giocatore.

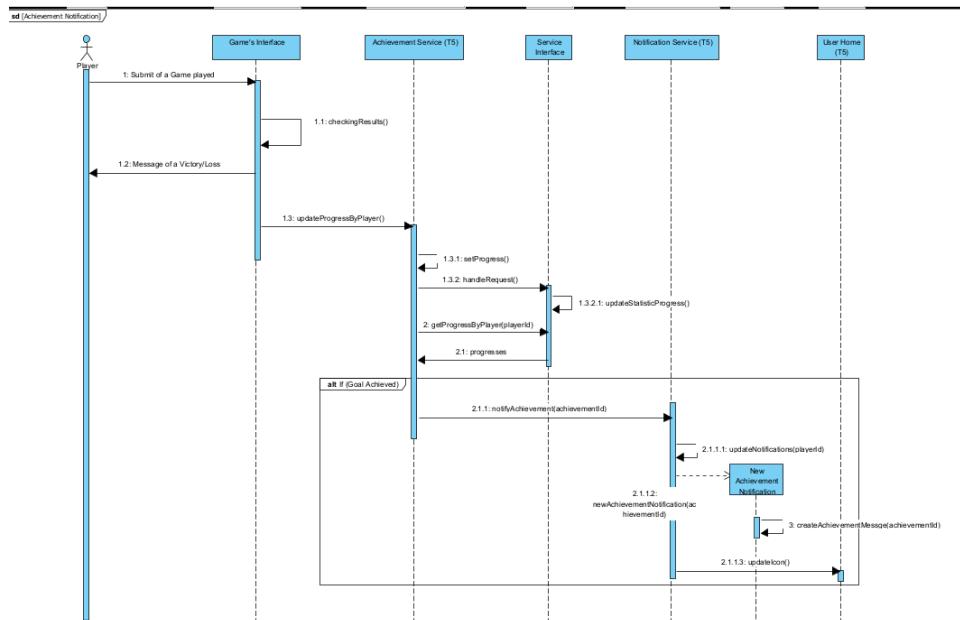


Figure 2.2: Diagramma di sequenza alto livello Notifica Achievement

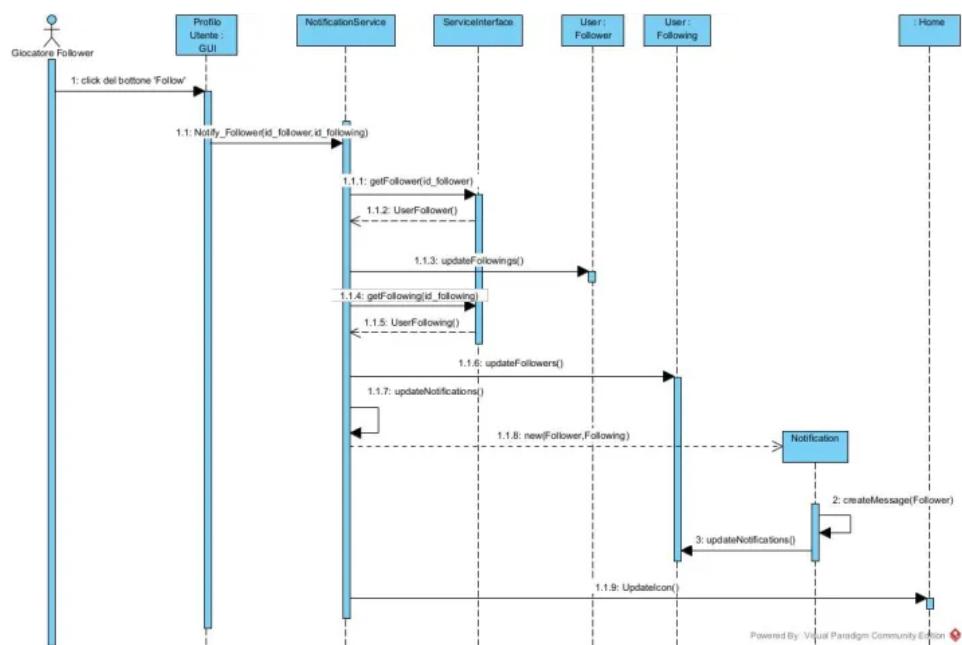


Figure 2.3: Diagramma di sequenza alto livello "Follow" giocatore

Chapter 3

Analisi dell’Impatto

Eccoci arrivati ad una parte molto delicata del progetto, ovvero l’analisi dell’impatto. Purtroppo molto spesso questa parte viene bistrattata ma è di fondamentale importanza poichè permette di individuare potenziali problemi o rischi legati ai cambiamenti richiesti o alle nuove funzionalità. Inoltre ci ha aiutato a stimare la quantità di lavoro necessario per ogni nuova modifica nel progetto. In questo capitolo presenteremo solo un’analisi dell’impatto stimato in quanto, le conseguenze dell’impatto effettivo è stato presentato successivamente nel documento, insieme alle relative scelte progettuali.

3.1 Impatto stimato Selezione Foto Profilo

1. Modifiche strutturali:

- Creazione del model `Profile` o modifica model `User` ([*T23*](#)).

2. Interfaccia utente:

- Aggiungere bottone “Modifica Profilo” nella pagina profilo: ([*T5*](#)).
 - Modifiche a `profile.html` e `GUIController` (componenti di logica).
- Creare una nuova pagina “Personalizza Profilo”: ([*T5*](#)).
 - Aggiungere un nuovo endpoint `@GetMapping`.
 - Creare la relativa pagina HTML.

3. Logica backend:

- Modificare il servizio `T23 Service`: ([*T5*](#)).
 - Aggiungere il supporto per aggiornare il path dell’immagine del profilo.

4. Immagini predefinite:

- Caricare immagini di default per il profilo: ([*T5*](#)).
 - Aggiungere i file nella directory `images`.

3.2 Impatto stimato Modifica Descrizione Profilo

1. Modifiche strutturali e interfaccia utente:

- Stessi passaggi della selezione foto profilo.

2. Logica backend:

- Aggiornare il servizio `T23 Service` per modificare la bio utente.

3.3 Impatto stimato Visualizzazione Achievement

1. Modifiche UI:
 - Aggiornare la sezione di visualizzazione degli achievement nella pagina profilo: (***T5***).
 - Modifiche a `profile_html` e `GUIController`.
2. Icone badge (*Priorità LOW*):
 - Aggiornamento delle icone (gestito principalmente dall'admin in T1).

3.4 Impatto stimato Visualizzazione Statistiche

1. Modifiche UI:
 - Aggiornare la sezione di visualizzazione delle statistiche nella pagina profilo: (***T5***).
 - Modifiche a `profile_html`.

3.5 Impatto stimato Notifica Nuovo Achievement

1. Modifiche strutturali:
 - Creare il model `Notifica`: (***T23***).
 - Collegare le notifiche a uno specifico `User`.
 - Implementare il repository `NotificationRepository`.
2. Logica backend:
 - Creare il servizio `NotificationService` (***T5***).
 - Modificare `T23 Service` per integrare la gestione delle notifiche.
 - Aggiornare il metodo `saveGame` in `GUIController` per generare una notifica al raggiungimento di un achievement.
3. Visualizzazione UI:
 - Mostrare le notifiche:
 - Nella home del giocatore (***T5: main_html***).
 - Nella pagina profilo (***T5: profile_html***).

3.6 Impatto stimato Gestione Notifiche

1. Logica backend:
 - Implementare operazioni CRUD in `T23 Service` e `NotificationRepository`.
2. Interfaccia utente:
 - Aggiungere buttoni per gestire notifiche nella pagina profilo (***T5: profile_html***).

3.7 Impatto stimato Seguire un Altro Giocatore

1. Modifiche strutturali:

- Aggiornare il model User: (*T23*).
 - Aggiungere liste followed e following.

2. Logica backend:

- Modificare il servizio T23 Service per gestire l'aggiornamento delle liste.

3. Interfaccia utente:

- Aggiornare la pagina profilo (*T5: profile_html*).

3.8 Impatto stimato Visualizzazione Follower e Seguiti

1. Interfaccia utente:

- Aggiungere una sezione nella pagina profilo per mostrare follower e seguiti (*T5: profile_html*).

3.9 Impatto stimato Visualizzazione Profili Giocatori

1. Interfaccia utente:

- Creare una nuova pagina: (*T5: GUIController/UI Gateway*).
 - Pagina simile alla pagina profilo personale ma con meno interazioni.
- Implementare nuovo endpoint @GetMapping con parametri per giocatore da visualizzare.

3.10 Impatto stimato Notifica Nuovo Follower

1. Logica backend:

- Generare una notifica quando un utente clicca il bottone "Segui" nella pagina profilo (*T5: profile_html*).

3.11 Impatto stimato Visualizzazione Token Guadagnati

1. Interfaccia utente:

- Aggiungere una sezione nella pagina profilo per visualizzare i token accumulati (*T5: profile_html*).

Chapter 4

Progetto dettagliato della soluzione

Dopo la prima iterazione che abbiamo dedicato alla definizione dei primi requisiti funzionali tramite storie utente, casi d'uso e quindi di criteri di accettazione e al raffinamento dei requisiti, abbiamo stabilito gli obiettivi delle successive iterazioni definendo quando implementare i nostri casi d'uso che abbiamo raggruppato in 3 features: Edit Profile, Notification Service, Social. In questa sezione riporteremo le conoscenze, il flusso logico e le conclusioni che abbiamo preso per poter implementare le funzionalità del nostro task.

4.1 Composite Diagram dell'Architettura

La nostra architettura dovrà avere nuovi endpoint, quindi interfacce differenti rispetto al progetto iniziale.

4.2 Spring MVC

Per comprendere appieno le modifiche implementate nei task T23 e T5, è fondamentale avere una panoramica del ruolo di Spring e del pattern MVC (Model-View-Controller) che esso implementa.

Spring Framework è un framework open-source per la creazione di applicazioni Java. Tra i suoi numerosi vantaggi, Spring eccelle nella gestione delle dipendenze e nell'implementazione del pattern MVC.

Il pattern **MVC** un modello architettonico che suddivide l'applicazione in tre componenti principali:

- **Model:** Rappresenta i dati dell'applicazione e la logica di business.
- **View:** Si occupa di presentare i dati all'utente, tipicamente tramite un'interfaccia grafica.
- **Controller:** Gestisce le richieste dell'utente, interagisce con il Model per recuperare i dati e li passa alla View per la visualizzazione.

Spring MVC, un modulo all'interno di Spring Framework, offre un'implementazione robusta e flessibile del pattern MVC. Un componente chiave di Spring MVC è il `DispatcherServlet`, che funge da Front Controller, ricevendo tutte le richieste in entrata e indirizzandole al controller appropriato.

Spring si occupa anche della **gestione delle istanze (Dependency Injection)**, un principio fondamentale per la creazione di applicazioni modulari e facilmente testabili. Con Dependency Injection, gli oggetti non creano le proprie dipendenze, ma le ricevono dall'esterno, tipicamente tramite un container di **Inversion of Control (IoC)** come Spring.

Nel nostro caso, T5 è stato costruito utilizzando Spring MVC, beneficiando quindi di tutte le funzionalità offerte da questo framework. T23, invece, non segue un framework specifico, ma le modifiche apportate seguono comunque il pattern MVC. Questa introduzione a Spring e al pattern MVC dovrebbe fornire al lettore il contesto necessario per comprendere le modifiche implementate nei task T23 e T5 e per inserirsi nell'ecosistema Spring quando si apporteranno modifiche al progetto.

4.2.1 Che cos'è Spring Boot

Spring Boot è un framework open-source che fornisce automaticamente configurazioni predefinite, in questo modo consente agli sviluppatori di concentrarsi sulla logica applicativa, riducendo il tempo speso per la configurazione e l'infrastruttura. Spring Boot analizza le dipendenze del progetto e configura automaticamente l'applicazione di conseguenza, eliminando la necessità di file di configurazione XML complessi. Per esempio, se Spring Boot rileva una dipendenza per Spring JDBC nel progetto, configurerà automaticamente un DataSource e un JdbcTemplate. Un altro vantaggio significativo di Spring Boot è l'inclusione di un server web incorporato, come Tomcat, Jetty o Undertow. Ciò consente agli sviluppatori di eseguire le loro applicazioni senza la necessità di distribuire l'applicazione su un server web esterno. Inoltre Spring Boot offre una serie di altre funzionalità che semplificano lo sviluppo:

- **Starter Dependency:** Spring Boot fornisce "starter" per diversi tipi di applicazioni, come applicazioni web, applicazioni batch e applicazioni di accesso ai dati. Questi "starter" includono tutte le dipendenze necessarie per iniziare a sviluppare un particolare tipo di applicazione.
- **Actuator:** Spring Actuator fornisce endpoint di gestione e monitoraggio per l'applicazione, consentendo agli sviluppatori di monitorare le metriche dell'applicazione, visualizzare lo stato di salute e persino modificare la configurazione in fase di runtime.
- **Spring Data JPA:** framework per l'accesso ai dati che semplifica l'interazione con i database. Spring Data JPA consente agli sviluppatori di definire interfacce di repository che estendono JpaRepository, fornendo metodi CRUD (Crea, Leggi, Aggiorna, Elimina) predefiniti e la possibilità di definire query personalizzate utilizzando metodi con nomi significativi. Lo vedremo poi in dettaglio.

4.3 Seconda iterazione

L'obiettivo di questa iterazione prevede l'implementazione di Edit Profile e Notification Service.

4.4 Feature 1: Edit Profile

Per questa funzionalità si è scelto di utilizzare un approccio top down, iniziando a lavorare sulla rotta del profilo, cioè sul come viene chiamata la pagina di profilo iniziale. Ci troviamo nel microservizio T5, ossia il gestore della home di gioco, quindi che fornisce l'accesso alle varie modalità di gioco e anche al profilo utente.

4.4.1 GuiController.java

Quando chiamo la pagina di profilo vengo indirizzato dal GuiController alla rotta /profile , in questa rotta avviene l'autenticazione.

Una volta autenticato vengo indirizzato a /profile/{profile_id}. Questo metodo è un handler per una richiesta **HTTP GET** che restituisce la pagina del profilo di un giocatore identificato da un playerID.

Il metodo costruisce dinamicamente i dati necessari per visualizzare la pagina profilo di un giocatore, recuperando informazioni personali, statistiche, progressi degli achievement, e fornendo il contesto autenticato grazie al cookie jwt.

Istanzio quindi il PageBuilder (model, jwt, ...) per il Profile e creo gli ObjectComponents per le cose che voglio vedere istanziate.

```

//Definisco la mia pagina utente
@GetMapping("/profile/{playerID}")
public String profilePage(Model model,
    @PathVariable(values="playerID") String playerID,
    @CookieValue(name = "jwt", required = false)
String jwt) {
    //Mi sto istanziando il profilo come PageBuilder
    PageBuilder profile = new PageBuilder(serviceManager, "profile",
model);
    //Do l'autenticazione
    profile.SetAuth(jwt);

    //Mi prendo l'id del giocatore, cosi da filtrare per il suo id i suoi
    progressi degli achievement e le sue statistiche
    int userId = Integer.parseInt(playerID);

    // PROVARE A RENDERE REALE IL PASSAGGIO DEI DATI ALLA PAGINA PROFILO
    //UserProfile profileDTO=new UserProfile(userId,"Inserisci qui la tua
    bio...","sample_propic.jpg",null,null);
    // Mi prendo prima tutti gli utenti e poi l'utente che mi interessa
    con l'id con un filtraggio
    List<User> users = (List<User>) serviceManager.handleRequest("T23",
    "GetUsers");
    User user = users.stream().filter(u -> u.getId() ==
    userId).findFirst().orElseThrow(() -> new RuntimeException("User not found"));

    // Mi prendo i suoi dati da passare alla pagina
    String email = user.getEmail();
    String studies = user.getStudies();
    String username = user.getName();
    String surname = user.getSurname();

    //mi prendo immagine e bio
    String image = userProfileService.getProfilePicture(userId);
    String bio = userProfileService.getProfileBio(userId);

    // Mi prendo i progressi degli achievement e le statistiche
    List<AchievementProgress> achievementProgresses =
    achievementService.getProgressesByPlayer(userId);
    List<StatisticProgress> statisticProgresses =
    achievementService.getStatisticsByPlayer(userId);
    List<Statistic> allStatistics = achievementService.getStatistics();
    Map<String, Statistic> IdToStatistic = new HashMap<>();

    for (Statistic stat : allStatistics)
        IdToStatistic.put(stat.getID(), stat);

    // Creo i componenti per passare i dati alla pagina
    GenericObjectComponent objEmail = new GenericObjectComponent("email",
    email);
    GenericObjectComponent objStudies = new
    GenericObjectComponent("studies", studies);
    GenericObjectComponent objUsername = new
    GenericObjectComponent("username", username);
    GenericObjectComponent objSurname = new
    GenericObjectComponent("surname", surname);
    GenericObjectComponent objImage = new GenericObjectComponent("propic",
    image);
    GenericObjectComponent objBio = new GenericObjectComponent("bio",
    bio);

    GenericObjectComponent objAchievementProgresses = new
    GenericObjectComponent("achievementProgresses", achievementProgresses);
    GenericObjectComponent objStatisticProgresses = new
    GenericObjectComponent("statisticProgresses", statisticProgresses);
    GenericObjectComponent objIdToStatistic = new
    GenericObjectComponent("IdToStatistic", IdToStatistic);
    GenericObjectComponent objUserID = new
    GenericObjectComponent("userID", userId);

    // Aggiungo i componenti alla pagina
    profile.setObjectComponents(objEmail);
    profile.setObjectComponents(objStudies);
    profile.setObjectComponents(objUsername);
    profile.setObjectComponents(objSurname);
    profile.setObjectComponents(objImage);
    profile.setObjectComponents(objBio);
    profile.setObjectComponents(objAchievementProgresses);
    profile.setObjectComponents(objStatisticProgresses);
    profile.setObjectComponents(objIdToStatistic);
    profile.setObjectComponents(objUserID);

    return profile.handlePageRequest();
}

```

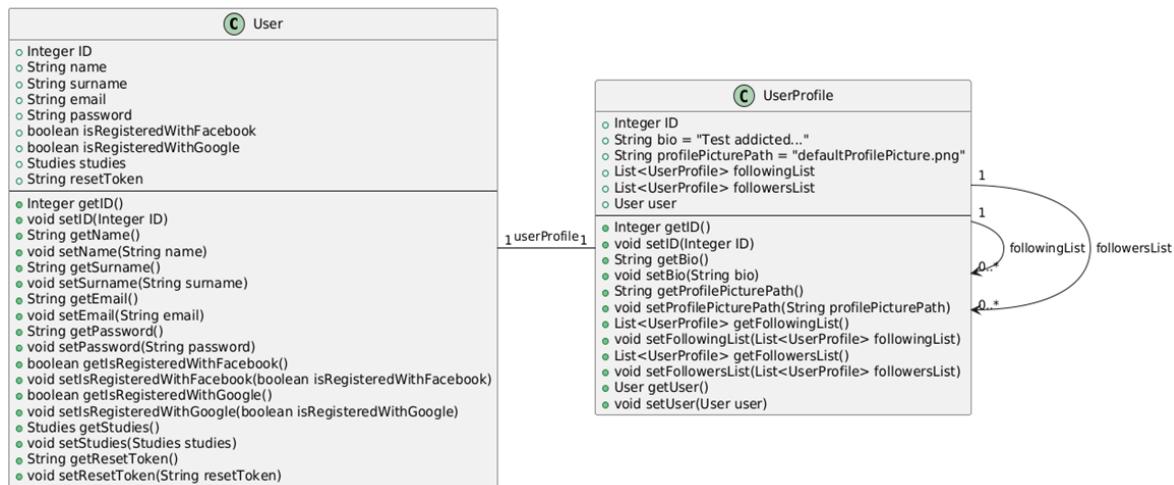
Figure 4.1: GuiController.java

- 1. Definizione della route e parametri** La route `/profile/\{playerID\}` accetta un `playerID` come parametro di path per identificare il giocatore. Il metodo usa anche un cookie JWT per gestire l'autenticazione dell'utente.
- 2. Inizializzazione del PageBuilder** `PageBuilder` è usato per costruire una pagina dinamica. È configurato con il contesto fornito dal `Model` e inizializzato con il parametro `jwt` per l'autenticazione.
- 3. Recupero dell'utente** Dato il `playerID`, il metodo recupera la lista degli utenti tramite il servizio `serviceManager`, quindi filtra la lista per trovare l'utente corrispondente al `playerID`. Se non trovato, viene sollevata un'eccezione.
- 4. Recupero dei dati del profilo** Email, studi, nome, e cognome sono recuperati direttamente dall'oggetto `User`. Immagine del profilo e bio sono ottenuti da un servizio dedicato: `userProfileService`.
- 5. Recupero dei progressi e statistiche** Gli achievement e statistiche del giocatore sono recuperati tramite il servizio `achievementService`.
- 6. Creazione di componenti** I dati del profilo e le statistiche vengono incapsulati in oggetti `GenericObjectComponent` per essere passati al `PageBuilder`.
- 7. Aggiunta dei componenti alla pagina** Tutti i componenti vengono aggiunti alla configurazione del `PageBuilder`.
- 8. Render della pagina** Infine, il metodo restituisce l'output della chiamata `handlePageRequest()` del `PageBuilder`, che genera la pagina del profilo configurata con i dati preparati.

4.4.2 Package Model in T23

Per creare la pagina di profilo ho bisogno di un'immagine di profilo e di una descrizione. Abbiamo scelto di mettere queste informazioni in un'entità distinta da `User`, in modo da dividere le informazioni "social" dell'utente dalle sue informazioni personali. Per portare avanti il lavoro abbiamo inserito nell'entità del profilo anche la lista dei follower e dei following che useremo nella prossima iterazione. Queste entità avranno una relazione uno a uno con gli `User` e verranno create al momento della sua iscrizione. Secondo questa decisione, procediamo a istanziare un oggetto `UserProfile` in T23, da cui prendere gli oggetti che voglio renderizzare nella pagina di profilo del singolo utente. Descriviamo quindi il file `UserProfile.java` e relativa Repository che abbiamo istanziato. Questi file appartengono al package `Model` del progetto e definiscono rispettivamente l'entità `UserProfile` e il repository JPA associato, utilizzato per interagire con il database.

4.4.3 Modello dei dati



4.4.4 Jakarta Persistence o Java Persistence API (JPA)

Si tratta di un layer di disaccoppiamento dai tool utilizzati per il mapping con il database. In questo modo le entity vengono gestite tramite Java e tutto quello che dobbiamo fare è un mapping fra il mio oggetto e quello che trovo nel database. Questo mapping viene realizzato chiamando la classe con lo stesso nome della tabella nel database. Il mapping effettivo verrà realizzato da un tool di mapping come Hibernate o EclipseLink, JPA non altro è che uno standard che questi tool sono in grado di implementare.

4.4.5 UserProfile

La classe `UserProfile` rappresenta una tabella del database denominata `Profiles` (nello **schema** `studentsrepo`). Questa entità modella il profilo di un utente, includendo attributi come la biografia, il percorso immagine del profilo e le relazioni sociali come follower e following.

Annotazioni principali

- `@Table`: Specifica il nome della tabella (`Profiles`) e lo schema associato (`studentsrepo`).
- `@Entity`: Contrassegna la classe come entità JPA.
- `@Id` e `@GeneratedValue`: Indicano che l'attributo `ID` è la chiave primaria e sarà generato automaticamente.
- `@OneToOne`, `@OneToMany` e `@JoinTable`: Gestiscono le relazioni tra questa entità e altre, come utenti e profili correlati.

Campi e funzionalità

1. **ID (Chiave primaria)** Un intero generato automaticamente per identificare univocamente il profilo.

Relazione con User Ogni `UserProfile` è associato a un utente tramite una relazione **One-to-One**. Questa relazione è mappata alla colonna `user_id` e viene ignorata dalla serializzazione JSON grazie a `@JsonIgnore`. Questa scelta ha portato a delle conseguenze su `User.java`, che è stato modificato aggiungendo il seguente codice:

```

1  @OneToOne(mappedBy = "user", cascade = CascadeType.ALL, fetch =
2      FetchType.LAZY)
  private UserProfile userProfile;

```

Listing 4.1: Codice aggiunto a `User.java`

2. **Campi di profilo**

- **Bio**: Una descrizione personale fino a 500 caratteri. Default: "Test addicted...".
- **ProfilePicturePath**: Nome del file immagine che rappresenta il profilo.

Abbiamo deciso di rappresentare la foto del profilo utilizzando un percorso (*path*), poiché l'uso di una stringa semplifica la gestione della comunicazione tramite *JSON*. Inoltre, questa scelta ci consente di definire percorsi relativi anziché assoluti, in linea con il principio di qualità e flessibilità che guida questo progetto.

3. **Relazioni con altri utenti**

- **FollowingList**: Elenco dei profili seguiti da questo utente, mappato alla tabella di join `user_following`.
- **FollowersList**: Elenco dei profili che seguono questo utente, mappato alla tabella di join `user_followers`.

4.4.6 UserProfileRepository

Questa classe fornisce un'API per l'interazione con il database tramite operazioni CRUD su entità UserProfile. Utilizza *Spring Data JPA*.

```

1 import org.springframework.data.jpa.repository.JpaRepository;
2 import org.springframework.stereotype.Repository;
3 @Repository
4 public interface UserProfileRepository extends
5     JpaRepository<UserProfile, Integer>{
6     UserProfile findById(Integer ID);
7 }
```

Listing 4.2: Definizione UserProfileRepository

1. Metodi ereditati da JpaRepository

- **CRUD:** Include i metodi standard per operazioni di creazione, lettura, aggiornamento e cancellazione.
- **Ricerca avanzata:** Supporta il querying dinamico grazie alla sintassi findBy.

2. Metodo `findById(Integer ID)`: Restituisce un oggetto UserProfile basato sull'ID.

4.4.7 User Service in T23

Nei *Service* viene definita la logica di business del nuovo servizio implementato, esaminiamone i nuovi metodi:

Metodo `findProfileByEmail` Il metodo `findProfileByEmail` fornisce una funzionalità per recuperare il profilo di un utente, rappresentato dall'oggetto `UserProfile` partendo dall'indirizzo email fornito come input. Questo metodo si articola in tre passaggi principali:

1. **Ricerca dell'utente** L'email fornita viene utilizzata per interrogare il repository `userRepository` attraverso il metodo `findByEmail`, al fine di recuperare un oggetto 'User' corrispondente.
2. **Controllo di esistenza** Una volta ottenuto il risultato, il metodo verifica che l'utente esista. In caso contrario, viene sollevata un'eccezione di tipo `IllegalArgumentException` con un messaggio esplicativo.
3. **Recupero del profilo** Se l'utente è presente, il metodo restituisce il profilo associato attraverso il metodo `getUserProfile()`.

```

1 public UserProfile findProfileByEmail(String email) {
2     // Recupera l'utente con l'email specificata
3     User user = userRepository.findByEmail(email);
4     // Controlla se l'utente esiste
5     if (user == null) {
6         throw new IllegalArgumentException("User with email "+email+" not found");
7     }
8     // Restituisce il profilo dell'utente
9     return user.getUserProfile();
10 }
```

Listing 4.3: Metodo `findProfileByEmail`

Metodo `saveProfile` Il metodo `saveProfile` è progettato per salvare un oggetto `UserProfile` nel database. La logica di funzionamento include:

1. **Validazione dell'input** Il metodo verifica che l'oggetto `userProfile` non sia nullo. Se questa condizione non è soddisfatta, viene sollevata un'eccezione `IllegalArgumentException`.
2. **Persistenza dei dati** Utilizzando il repository `userRepository`, il metodo salva o aggiorna l'oggetto `UserProfile`. Se l'entità è nuova, viene creato un nuovo record; altrimenti, l'entità esistente viene aggiornata.

```
1  public void saveProfile(UserProfile userProfile) {
2      if (userProfile == null) {
3          throw new IllegalArgumentException("Profile not found");
4      }
5  }
```

Listing 4.4: UserService.java

```
package com.example.db_setup;

import java.util.List;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.Data;

@Table (name = "Profiles", schema = "studentsrepo")
@Data
@Entity
public class UserProfile {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    public Integer ID;

    @OneToOne
    @JoinColumn(name = "user_id")
    @JsonIgnore
    private User user;

    @Column(length = 500)
    public String bio = "Test addicted...";

    // Nome dell'immagine, usata come parte finale del path nel servizio dove sono salvate le immagini
    public String profilePicturePath = "defaultProfilePicture.png";

    @OneToMany
    @JoinTable(
        name = "user_following",
        joinColumns = @JoinColumn(name = "profile_id"),
        inverseJoinColumns = @JoinColumn(name = "following_id")
    )
    public List<UserProfile> followingList;

    @OneToMany
    @JoinTable(
        name = "user_followers",
        joinColumns = @JoinColumn(name = "profile_id"),
        inverseJoinColumns = @JoinColumn(name = "follower_id")
    )
    public List<UserProfile> followersList;

    public void setUser(User user){
        this.user = user;
    }

    //List<Statistic> allStatistics;
}
```

Figure 4.2: Struttura della classe UserProfile

4.4.8 Controller in T23

Il controller definisce le rotte per l'accesso ai servizi di gestione del profilo. Di seguito sono illustrati due endpoint principali.

Endpoint `getProfileByEmail`

Questo endpoint **GET** consente di recuperare il profilo di un utente in base all'indirizzo email fornito come parametro di query. Le sue caratteristiche principali includono:

1. **Validazione del parametro** L'email viene validata per assicurarsi che non sia vuota o nulla.
2. **Chiamata al servizio** Il metodo utilizza il servizio 'userService' per recuperare il profilo.
3. **Risposta** Se il profilo è disponibile, viene restituito come JSON con un codice HTTP 200; in caso contrario, un codice HTTP 400 viene restituito con un messaggio d'errore.

```

1 public ResponseEntity<UserProfile> getProfileByEmail(@RequestParam("email") String
2   email) {
3     // Validazione del parametro email
4     if (email == null || email.isEmpty()) {
5       return ResponseEntity.badRequest().body(null); // HTTP 400
6     }
7     // Recupero del profilo tramite il servizio
8     UserProfile userProfile = userService.findProfileByEmail(email);
9     return ResponseEntity.ok(userProfile); // HTTP 200
10 }
```

Listing 4.5: `UserService.java`

4.4.9 Endpoint `editProfile`

Questo endpoint **POST** consente di modificare il profilo di un utente, aggiornandone la biografia e l'immagine profilo. La procedura include:

1. **Parametri della richiesta** Sono richiesti tre parametri: 'email', 'bio' e 'profilePicturePath'.
2. **Recupero e validazione del profilo** Il profilo viene recuperato tramite il servizio `userService`. Se non è presente, il metodo restituisce un errore HTTP 400.
3. **Aggiornamento dei dati** I nuovi valori vengono applicati agli attributi del profilo, che viene successivamente salvato.
4. **Risposta al client** Se l'operazione è completata con successo, viene restituito un messaggio di conferma "Profile edited successfully" con codice HTTP 200.

```

1 @PostMapping("/edit_profile")
2   public ResponseEntity<String> editProfile(@RequestParam("email") String email,
3                                             @RequestParam("bio") String bio,
4                                             @RequestParam("profilePicturePath") String profilePicturePath) {
5     UserProfile profile = userService.findProfileByEmail(email);
6     if (profile == null) {
7       return ResponseEntity.status(HttpStatus.BAD_REQUEST)
8         .body("Profile not found");
9     }
10    profile.setBio(bio);
11    profile.setProfilePicturePath(profilePicturePath);
12    return ResponseEntity.ok("Profile edited successfully");
13 }
```

Listing 4.6: `UserService.java`

4.4.10 Integrazione con T5

Per migliorare la separazione delle responsabilità, la gestione delle immagini di profilo è stata spostata da T23 a T5. Questo consente al front-end di recuperare le immagini (in termini del loro path relativo) in modo più efficiente. Vediamo ora come utilizzare lato T5 le funzionalità implementate in T23.

4.4.11 ModelUserProfile in T5

La classe UserProfile rappresenta l'entità per la gestione delle informazioni personali degli utenti. Gli attributi principali includono:

- **ID**: Identificativo univoco.
- **Bio**: Descrizione personale.
- **Percorso immagine profilo**: Path dell'immagine.
- **Liste di following e follower**: Liste per la gestione delle relazioni fra utenti.



```

public class UserProfile {

    @JsonProperty("id")
    private Integer ID;
    @JsonProperty("bio")
    private String bio;
    @JsonProperty("profilePicturePath")
    private String profilePicturePath;
    @JsonProperty("followingList")
    private List<UserProfile> followingList;
    @JsonProperty("followersList")
    private List<UserProfile> followersList;

    public UserProfile(Integer ID, String bio, String profilePicturePath, List<UserProfile>
        followingList, List<UserProfile> followersList) {
        this.ID = ID;
        this.bio = bio;
        this.profilePicturePath = profilePicturePath;
        this.followingList = followingList;
        this.followersList = followersList;
    }

    //Costruttore vuoto necessario per thymeleaf
    public UserProfile(){}
}

```

La classe utilizza le annotazioni '@JsonProperty' di Jackson per la mappatura JSON. Questo permette la conversione automatica tra oggetti Java e JSON; la personalizzazione dei nomi dei campi nella serializzazione; la gestione consistente dei dati nelle chiamate API. La classe fornisce un set completo di getter e setter per tutti gli attributi, garantendo l'incapsulamento dei dati e permettendo la manipolazione controllata delle proprietà, mentre il metodo 'toString()' fornisce una rappresentazione testuale dell'oggetto, utile per debugging e logging.

La gestione di JSON con Jackson

Jackson è una libreria Java utilizzata per gestire JSON in modo efficiente. Essa consente di **serializzare** oggetti Java in stringhe o file JSON e di **deserializzare** stringhe o file JSON in oggetti Java.

Jackson fornisce una serie di annotazioni per personalizzare il processo di serializzazione/deserializzazione:

- **@JsonProperty**: Specifica il nome della proprietà JSON.
- **@JsonIgnore**: Esclude un campo durante la serializzazione/deserializzazione.
- **@JsonInclude**: Include un campo solo in determinate condizioni (es., non nullo).
- **@JsonFormat**: Formatta i dati, ad esempio date o numeri.

Può serializzare e deserializzare oggetti con relazioni complesse, incluse strutture annidate e array. Il suo ObjectMapper può essere configurato per ignorare proprietà sconosciute o per indentare il JSON generato per migliorarne la leggibilità.

Integrazione con Thymeleaf

Thymeleaf è un motore di template Java moderno che si integra perfettamente con la classe UserProfile per la generazione di viste HTML dinamiche.

- **Costruttore Vuoto:** Il costruttore vuoto `public UserProfile()` è essenziale per Thymeleaf: permette l'istanziazione dinamica degli oggetti durante il rendering delle view e facilita il binding dei form HTML con gli oggetti UserProfile

Abbiamo dovuto modificare anche User.java:

```

1  @JsonProperty("userProfile")
2  private UserProfile userProfile;
3
4  public User(Long id, String name, String surname, String email, String password,
5      boolean isRegisteredWithFacebook, String studies, UserProfile
6          userProfile, String resetToken) {
7      this.id = id;
8      this.name = name;
9      this.surname = surname;
10     this.email = email;
11     this.password = password;
12     this.isRegisteredWithFacebook = isRegisteredWithFacebook;
13     this.studies = studies;
14     this.userProfile = userProfile; //MODIFICA
15     this.resetToken = resetToken;
16 }
17 public UserProfile getUserProfile() {
18     return userProfile;
19 }
20 public void setUserProfile(UserProfile userProfile) {
21     this.userProfile = userProfile;
22 }
```

Listing 4.7: UserService.java

Questa relazione rappresenta un'associazione uno-a-uno tra User e UserProfile, dove:

- Ogni utente può avere un solo profilo e questo contiene informazioni social e dettagli estesi
- La separazione permette una gestione modulare dei dati

4.4.12 T23Service in T5

Posso quindi gestire la chiamata nel Service lato T5.

Il metodo EditProfile gestisce la modifica del profilo utente attraverso una chiamata REST POST. È progettato per aggiornare la biografia e l'immagine del profilo di un utente specifico.

```

1 // Metodo per modificare il profilo di un utente
2 private Boolean EditProfile(String userEmail, String bio, String imagePath) {
3     final String endpoint = "/edit_profile";
4
5     MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
6     map.add("email", userEmail);
7     map.add("bio", bio);
8     map.add("profilePicturePath", imagePath);
9
10    return callRestPost(endpoint, map, null, Boolean.class);
11 }
```

Listing 4.8: T23Service.java

1. **Parametri:** sono richiesti tre parametri: userEmail, bio, imagePath
2. **Costruzione Request:** Utilizza MultiValueMap per i parametri della richiesta e organizza i dati in coppie chiave-valore
3. **Endpoint:** Path relativo: '/edit_profile', Metodo HTTP: POST
4. **Response:** Ritorna true se modifica avvenuta con successo, false se la modifica è fallita.

4.4.13 UserProfileService

è un servizio in Spring che gestisce le operazioni relative ai profili utente, inclusa la gestione delle biografie e delle immagini del profilo.

getProfileBio

```

1 public String getProfileBio(int playerID) {
2     int userId = playerID;
3     List<User> users = (List<User>) serviceManager.handleRequest("T23", "GetUsers");
4     User user = users.stream()
5         .filter(u -> u.getId() == userId)
6         .findFirst()
7         .orElseThrow(() -> new RuntimeException("User not found"));
8     return user.getUserProfile().getBio();
9 }
```

Listing 4.9: UserProfileService.java

1. Assegna l'ID del giocatore ricevuto come parametro
2. Richiede la lista completa degli utenti al ServiceManager
3. **Ricerca Utente Specifico**
 - Utilizza Stream API per filtrare gli utenti
 - Cerca l'utente con l'ID corrispondente
 - Lancia un'eccezione se non trova l'utente
4. Accede al profilo dell'utente e ne restituisce la biografia

getAllProfilePictures

```

1 public List<String> getAllProfilePictures() {
2     List<String> list_images = new ArrayList<>();
3     list_images.add("defaultProfilePicture.png");
4     list_images.add("SimpleFemale.png");
5     list_images.add("SimpleMale.png");
6     return list_images;
7 }
```

Listing 4.10: UserProfileService.java

1. **Inizializzazione Lista:** Crea una nuova ArrayList vuota per memorizzare i nomi dei file delle immagini
2. **Popolamento Lista:** Aggiunge tre immagini predefinite alla lista
3. **Restituzione Lista:** Restituisce la lista contenente i percorsi delle immagini profilo disponibili

getProfilePicture

```

1 public String getProfilePicture(int playerID) {
2     int userId = playerID;
3     List<User> users = (List<User>) serviceManager.handleRequest("T23", "GetUsers");
4     User user = users.stream()
5         .filter(u -> u.getId() == userId)
6         .findFirst()
7         .orElseThrow(() -> new RuntimeException("User not found"));
8 }
```

```

9  List<String> list_images = this.getAllProfilePictures();
10
11 // Validazione del path dell'immagine
12 Boolean propicvalid = false;
13 for (int i = 0; i < list_images.size(); i++) {
14     if (user.getUserProfile().getProfilePicturePath()
15         .equals(list_images.get(i))) {
16         propicvalid = true;
17         break;
18     }
19 }
20
21 return propicvalid ? user.getUserProfile().getProfilePicturePath() : null;
22 }
```

Listing 4.11: UserProfileService.java

- 1. Acquisizione ID Utente:** Memorizza l'ID del giocatore passato come parametro
- 2. Recupero Lista Utenti:** Richiede la lista completa degli utenti tramite ServiceManager
- 3. Ricerca Utente Specifico**
 - Utilizza Stream API per trovare l'utente con l'ID specificato
 - Lancia eccezione se l'utente non esiste
- 4. Ottenimento Lista Immagini Disponibili:** Richiama getAllProfilePictures() per ottenere la lista delle immagini valide
- 5. Validazione Path Immagine**
 - Verifica se il path dell'immagine profilo dell'utente è presente nella lista delle immagini valide
 - Interrompe il ciclo appena trova una corrispondenza
- 6. Restituzione Risultato**
 - Se l'immagine è valida (propicvalid = true): restituisce il path
 - Se l'immagine non è valida: restituisce null

4.4.14 Model html: `profile.html` e `Edit_Profile.html`

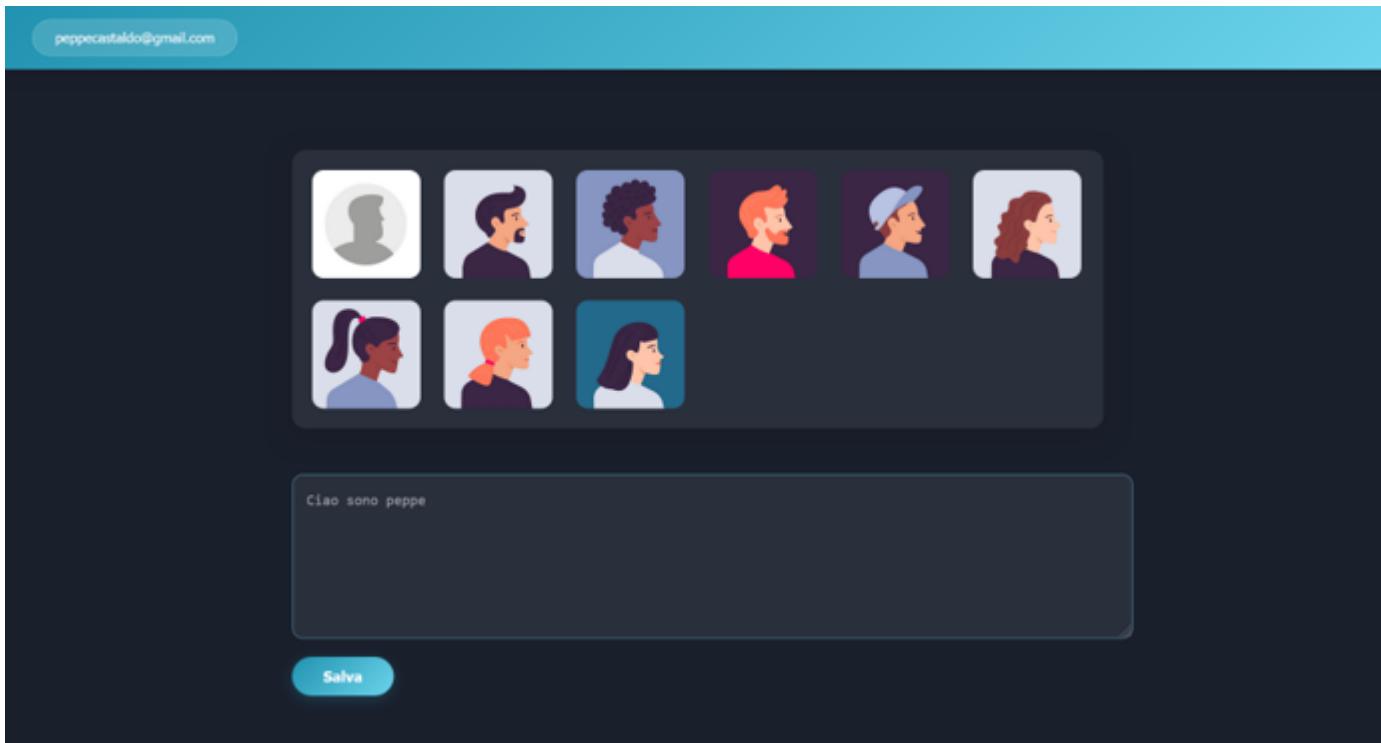
Il sistema di gestione del profilo utente è composto da due template principali:

Template Visualizzazione Profilo

Questo template mostra le informazioni del profilo e include:

- Header con bottoni di navigazione - Card profilo utente con: - Immagine profilo - Bottone **Modifica Profilo** ('onclick="location.href='/edit_profile'") reindirizza l'utente alla pagina di modifica.
- Abbiamo scelto di creare una nuova pagina di modifica per rendere la visualizzazione lato utente più pulita.
- Informazioni utente (nome, cognome, email, bio) - Sezione trofei - Sezione statistiche

Template Modifica Profilo



Questo template permette la modifica delle informazioni del profilo e si compone di :

- Header con bottone per tornare al profilo
- Galleria immagini profilo selezionabili
- Campo textarea per la modifica della bio
- Bottone **Salva** per confermare le modifiche

Funzionalità

1. L'utente accede alla pagina di modifica dal profilo
2. Può selezionare una nuova immagine profilo
3. Può modificare la bio nel campo textarea
4. Le modifiche vengono salvate al click del bottone **Salva**

4.4.15 Integrazione con PageBuilder

Il **template html** è stato creato per integrarsi con il component `PageBuilder` che gestisce:

- Caricamento dati utente nei template

- Gestione autorizzazioni
- Gestione errori e reindirizzamenti
- Aggiornamento del modello con i dati modificati

Default.conf in Uigateway

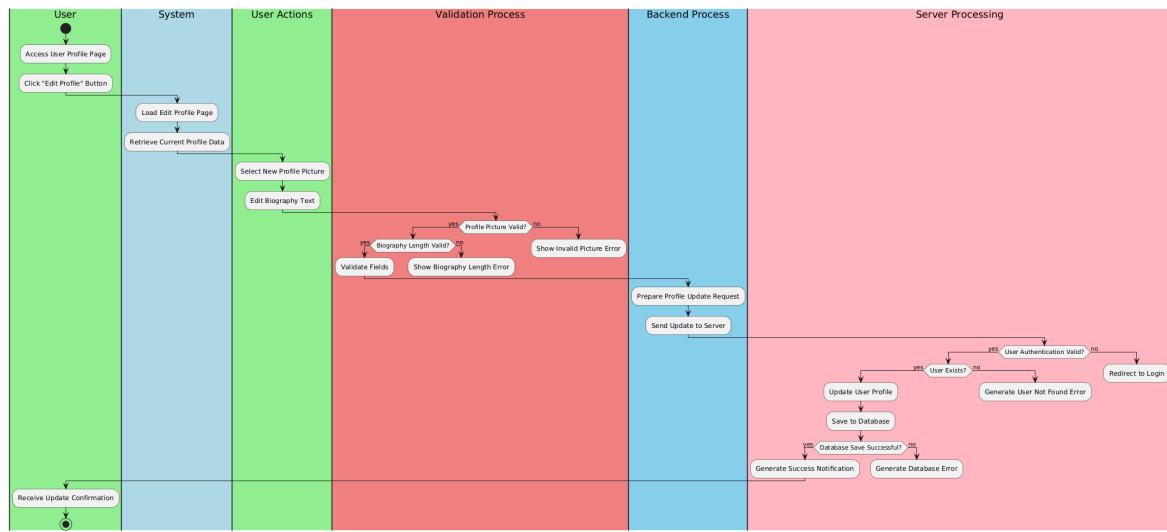
```
jsx
    location ~ ^/(leaderboard|gamemode_scalata|profile|edit_profile|update-
profile|main|gamemode|editor|report|t5|run|StartGame|changeLanguage) {
        include /etc/nginx/includes/proxy.conf;
        proxy_pass http://t5-app-1:8080;
        proxy_next_upstream error timeout http_502 http_503 http_504;
        proxy_next_upstream_tries 3;
    }

    location ~
^/(menu|login|logout|register|mail_register|password_change|password_reset|t23|students_list|oauth2/aut-
horization/google|validateToken|checkService|checkSession) {
        include /etc/nginx/includes/proxy.conf;
        proxy_pass http://t23-g1-app-1:8080;
        proxy_next_upstream error timeout http_502 http_503 http_504;
        proxy_next_upstream_tries 3;
}
```

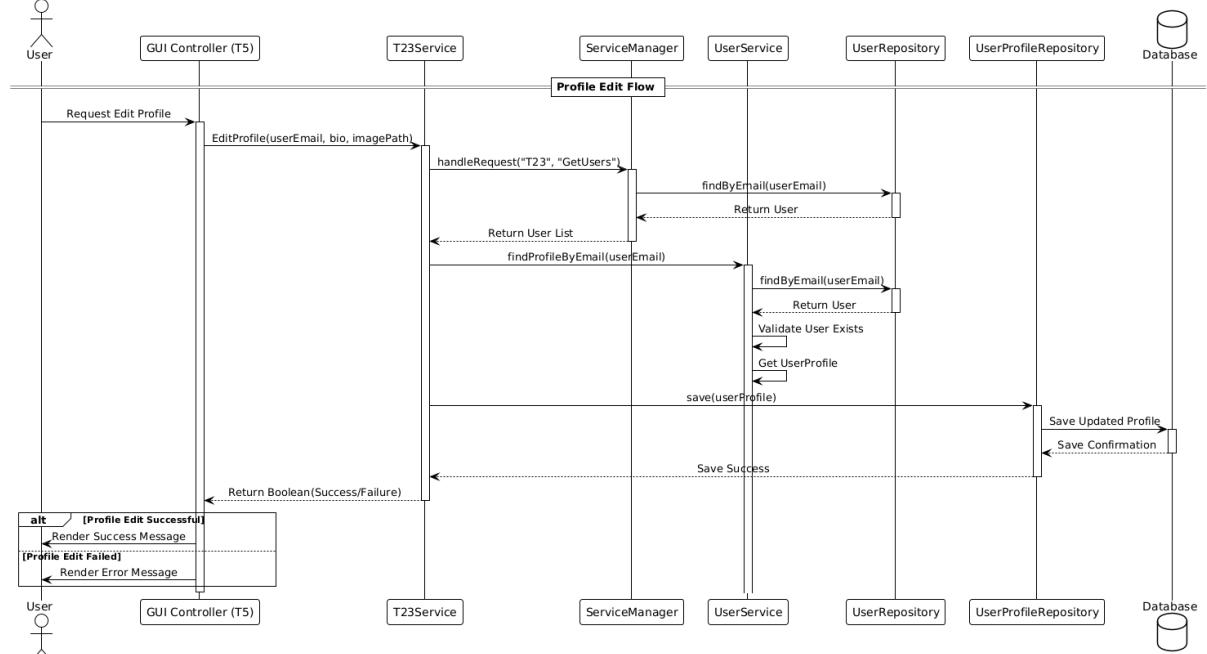
Sono state aggiunte le nuove rotte nel file di configurazione dell'uigateway, così facendo il sistema è in grado di gestire correttamente le richieste http a quelle rotte e inoltrarle ai rispettivi server.

4.4.16 Diagrammi di interazione (dettaglio)

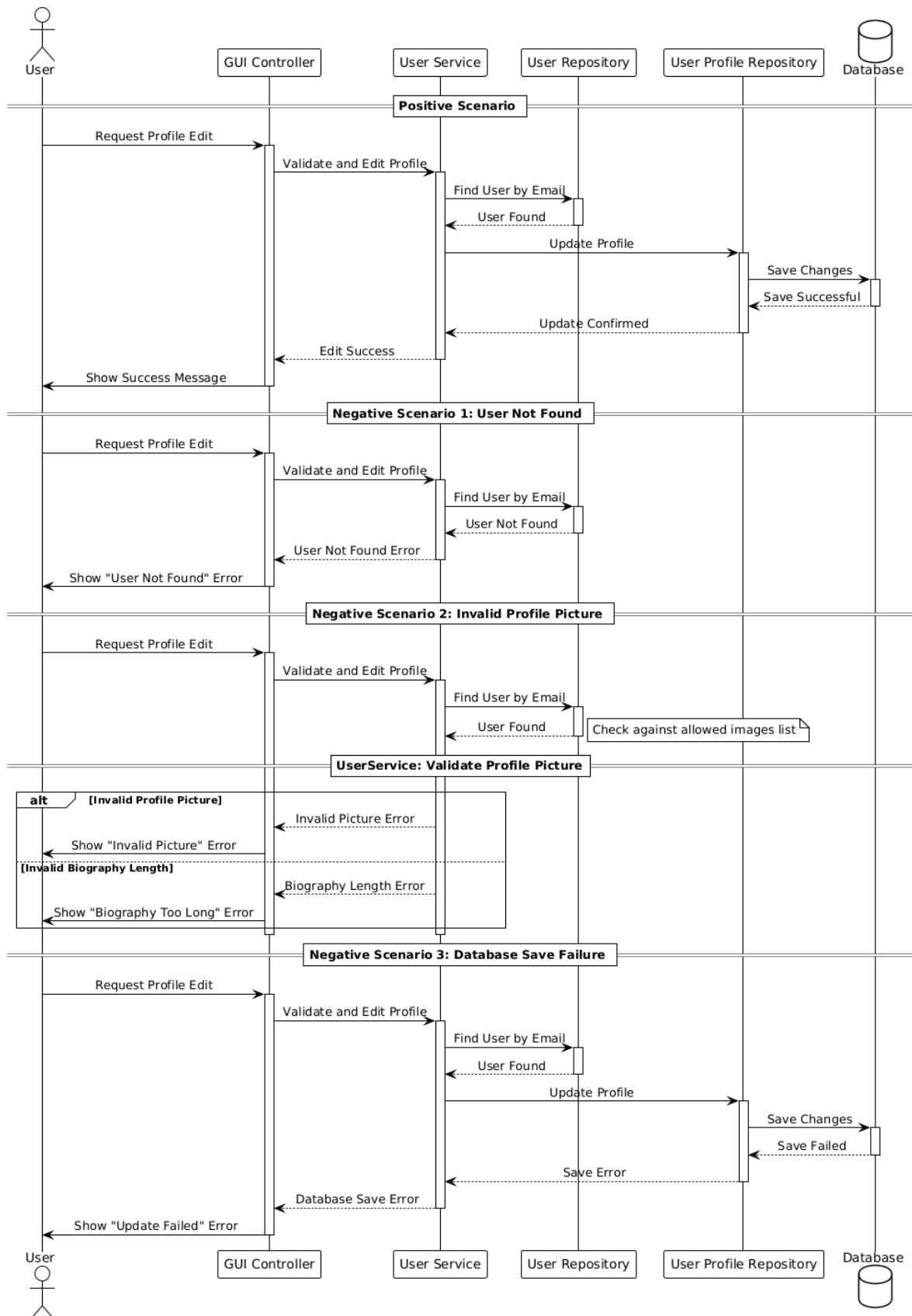
Activity Diagram



Sequence Diagram di dettaglio



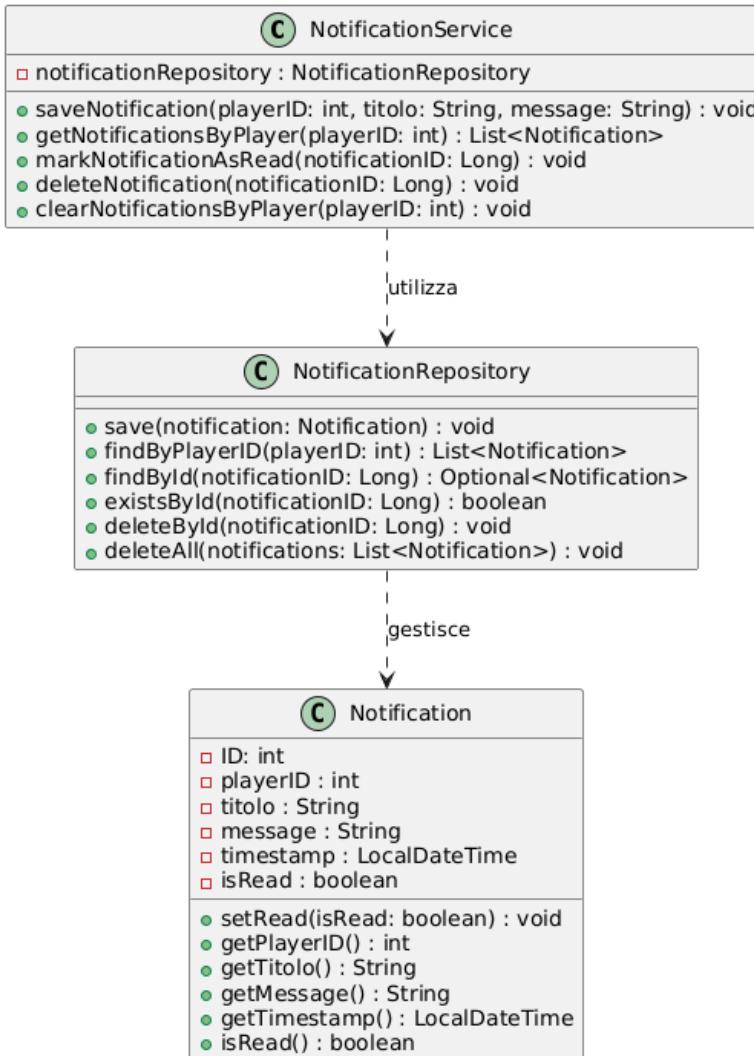
Sequence Diagram di dettaglio: scenari negativi



4.5 Feature 2: Notification Service

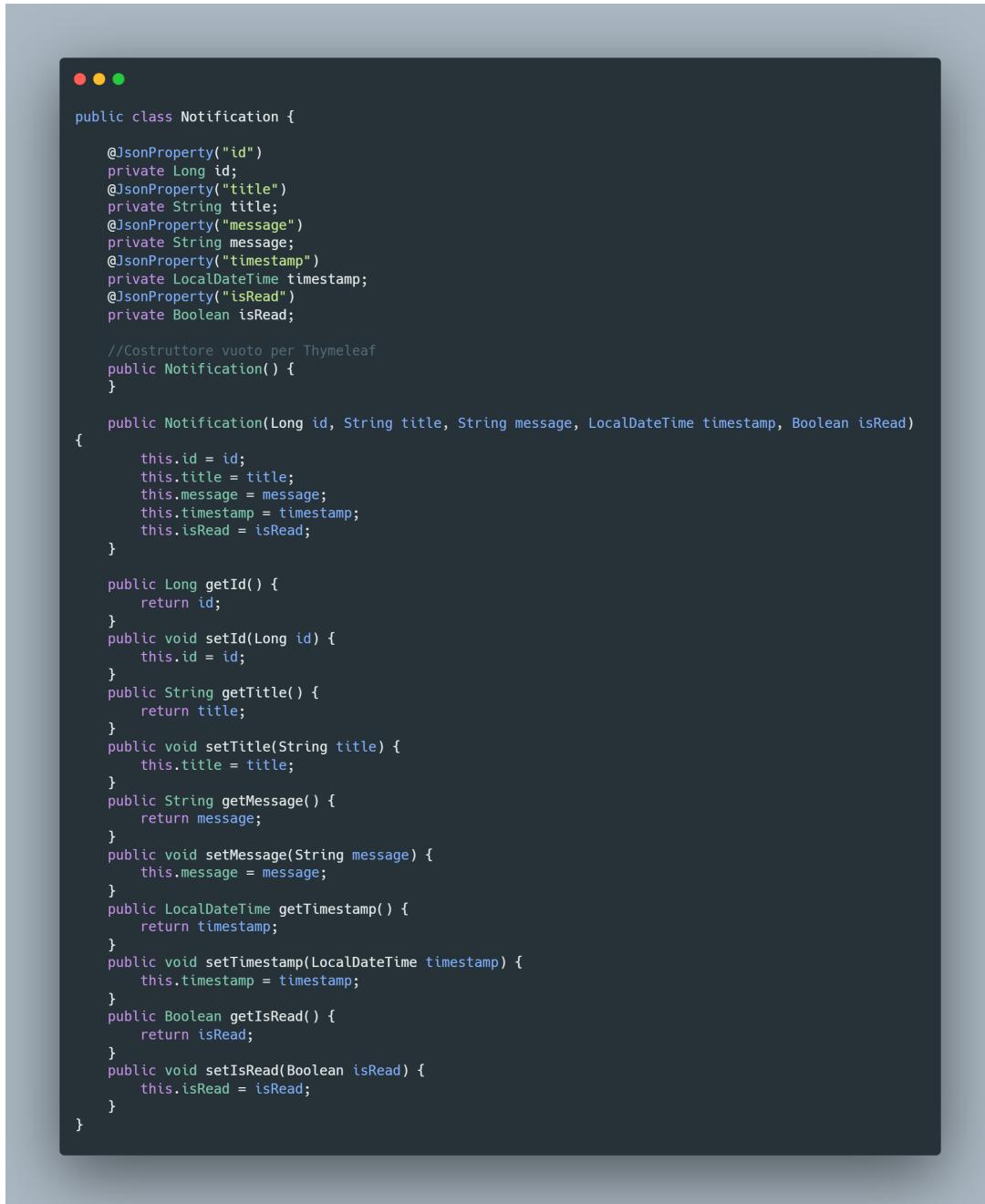
Il sistema di gestione delle notifiche permette di creare, salvare e visualizzare messaggi personalizzati per ogni utente. Altri servizi possono usare l'interfaccia di T23 per generare notifiche ad hoc, adattandole al contesto. Le notifiche vengono salvate insieme ai dati del profilo utente nel database di T23, così da garantire coerenza e facilità di accesso. In questo modo, possono essere recuperate e gestite in T5 insieme al resto delle informazioni del profilo. Abbiamo riflettuto su quale fosse il luogo più appropriato per salvare le notifiche nel database. Abbiamo deciso di conservarle in T23, in quanto strettamente legate al singolo utente. Questa scelta ci è sembrata coerente con l'approccio adottato per altri oggetti del profilo, facilitando il loro successivo recupero da T5.

4.5.1 Modello dei dati



4.5.2 Package Model in T23

Siamo partiti quindi dal model di T23 e abbiamo definito una classe `Notification.java`, secondo il class diagram riportato di seguito:



```

public class Notification {

    @JsonProperty("id")
    private Long id;
    @JsonProperty("title")
    private String title;
    @JsonProperty("message")
    private String message;
    @JsonProperty("timestamp")
    private LocalDateTime timestamp;
    @JsonProperty("isRead")
    private Boolean isRead;

    //Costruttore vuoto per Thymeleaf
    public Notification() {
    }

    public Notification(Long id, String title, String message, LocalDateTime timestamp, Boolean isRead) {
        this.id = id;
        this.title = title;
        this.message = message;
        this.timestamp = timestamp;
        this.isRead = isRead;
    }

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public LocalDateTime getTimestamp() {
        return timestamp;
    }
    public void setTimestamp(LocalDateTime timestamp) {
        this.timestamp = timestamp;
    }
    public Boolean getIsRead() {
        return isRead;
    }
    public void setIsRead(Boolean isRead) {
        this.isRead = isRead;
    }
}

```

4.5.3 NotificationRepository

Quindi abbiamo implementato una classe `NotificationRepository.java` che permette la ricerca per id del giocatore delle notifiche.

```

1  @Repository
2  public interface NotificationRepository extends JpaRepository<Notification, Long> {
3      List<Notification> findByPlayerID(int playerID);
4  }

```

Listing 4.12: NotificationRepository

4.5.4 NotificationService in T23

Questo Service implementa il salvataggio, il recupero, la cancellazione delle notifiche e permette di marcarle come lette o cancellarle in blocco.

```

public void saveNotification(int playerID, String titolo, String message) {
    Notification notification = new Notification(playerID, titolo, message);
    notificationRepository.save(notification);
}

public List<Notification> getNotificationsByPlayer(int playerID) {
    return notificationRepository.findByPlayerID(playerID);
}

public void markNotificationAsRead(Long notificationID) {
    Optional<Notification> optionalNotification = notificationRepository.findById(notificationID);
    if (optionalNotification.isPresent()) {
        Notification notification = optionalNotification.get();
        notification.setRead(true);
        notificationRepository.save(notification);
    } else {
        throw new RuntimeException("Notification with ID not found");
    }
}

```

saveNotification

Crea e salva una nuova notifica per un dato giocatore.

```

1 public void saveNotification(int playerID, String titolo, String message) {
2     Notification notification = new Notification(playerID, titolo, message,
3         LocalDateTime.now(), false);
4     notificationRepository.save(notification); }

```

Listing 4.13: NotificationService.java

- Istanzia un oggetto `Notification` con i parametri forniti.
- Imposta `timestamp` a `LocalDateTime.now()` (tempo corrente).
- Imposta `isRead` a `false` (non letto).
- Utilizza il repository per salvare la notifica nel database.

getNotificationsByPlayer

Recupera tutte le notifiche associate a un determinato `playerID`.

```

1 public List<Notification> getNotificationsByPlayer(int playerID) {
2     return notificationRepository.findByPlayerID(playerID); }

```

Listing 4.14: NotificationService.java

- Usa il metodo `findById` del repository per recuperare le notifiche.
- Restituisce una lista di notifiche per il giocatore specificato.

markNotificationAsRead

Segna una notifica come "letta" (imposta `isRead` a `true`).

```

1  public void markNotificationAsRead(Long notificationID) {
2      Optional<Notification> optionalNotification =
3          notificationRepository.findById(notificationID);
4      if (optionalNotification.isPresent()) {
5          Notification notification = optionalNotification.get();
6          notification.setRead(true);
7          notificationRepository.save(notification);
8      } else {
9          throw new RuntimeException("Notifica con ID " + notificationID + " non
10             trovata.");
11     }
12 }
```

Listing 4.15: NotificationService.java

- Recupera la notifica con l'ID fornito usando `findById`.
- Se la notifica esiste:
 - Aggiorna il campo `isRead` a `true`.
 - Salva l'oggetto aggiornato.
- Se la notifica non esiste, lancia un'eccezione.

deleteNotification

Elimina una notifica identificata dal suo ID.

```

1  public void deleteNotification(Long notificationID) {
2      if (notificationRepository.existsById(notificationID)) {
3          notificationRepository.deleteById(notificationID);
4      } else {
5          throw new RuntimeException("Notifica con ID " + notificationID + " non
6             trovata.");
7     }
7 }
```

Listing 4.16: NotificationService.java

- Controlla se la notifica esiste nel database con `existsById`. - Se esiste, la elimina utilizzando `deleteById`.
- Se non esiste, lancia un'eccezione.

4.5.5 REST API Endpoints in T23

Il controller T23 offre un'interfaccia REST per gestire le notifiche associate a utenti specifici. Queste API utilizzano i servizi di NotificationService e UserService per creare, recuperare, aggiornare e cancellare notifiche in base all'email dell'utente e ai dettagli delle notifiche.

POST /new_notification

Crea una nuova notifica per un utente specifico.

```

1  @PostMapping("/new_notification")
2  public ResponseEntity<String> updateNotifications(@RequestParam("email") String
3  email,
4  @RequestParam("title") String title,
5  @RequestParam("message") String message) {
6  UserProfile profile = userService.findProfileByEmail(email);
7  if (profile == null) {
8      return ResponseEntity.status(HttpStatus.BAD_REQUEST)
9          .body("Profile not found");
10 }
11 notificationService.saveNotification(profile.getUser()
12         .getID(), title, message);
13 return ResponseEntity.ok("Profile notifications updated successfully");
}

```

Listing 4.17: New Notification Endpoint

1. Cerca il profilo utente usando l'email tramite **UserService**.
2. Se il profilo non esiste, restituisce 400 BAD_REQUEST con un messaggio.
3. Se il profilo esiste, crea una nuova notifica con il metodo `saveNotification` di **NotificationService**.
4. Restituisce 200 OK al successo.

GET /notifications

Recupera tutte le notifiche associate all'utente.

```

1 @GetMapping("/notifications")
2     public List<Notification> getNotifications(@RequestParam("email") String
3         email) {
4         UserProfile profile = userService.findProfileByEmail(email);
5         if (profile == null) {
6             System.out.println(ResponseEntity
7                 .status(HttpStatus.BAD_REQUEST).body("[T23 Controller] UserProfile
8                     not found"));
9             return null;
10        }
11        return notificationService.getNotificationsByPlayer(profile
12                         .getUser().getID());
13    }

```

Listing 4.18: New Notification Endpoint

1. Cerca il profilo utente con l'email fornita.
2. Se il profilo non esiste, restituisce 400 BAD_REQUEST e logga un messaggio.
3. Recupera le notifiche tramite NotificationService.getNotificationsByPlayer() usando l'ID dell'utente.
4. Restituisce la lista di notifiche.

POST /update_notification

Marca una notifica specifica come letta.

```

● ● ●

@PostMapping("/update_notification")
public ResponseEntity<String> updateNotification(@RequestParam("email") String email,
                                                    @RequestParam("id notifica") String notificationID)
{
    // Cerca il profilo dell'utente utilizzando l'email fornita come parametro
    UserProfile profile = userService.findProfileByEmail(email);

    // Controlla se il profilo dell'utente è stato trovato
    if (profile == null) {
        // Se il profilo non esiste, restituisce un errore 400 (BAD_REQUEST) con un messaggio
        // appropriato
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("[T23 Controller] UserProfile not
found");
    }

    Long notifID = Long.parseLong(notificationID); // Converte l'ID della notifica da stringa a
    tipo Long
    boolean found = false; // Inizializza una variabile booleana per determinare se la notifica è
    stata trovata
    // Recupera tutte le notifiche associate all'utente utilizzando il suo ID
    List<Notification> notifications =
    notificationService.getNotificationsByPlayer(profile.getUser().getID());

    // Cerca la notifica specifica all'interno della lista di notifiche
    for (Notification notif : notifications) {
        if (notifID == notif.getId()) {
            found = true;
            break;
        }
    }

    // Se la notifica è stata trovata
    if (found) {
        // Marca la notifica come letta utilizzando il servizio notifiche
        notificationService.markNotificationAsRead(notifID);
        // Restituisce una risposta 200 (OK) per indicare il successo dell'operazione
        return ResponseEntity.ok("Profile notifications updated successfully");
    } else {
        // Se la notifica non è stata trovata, restituisce un errore 404 (NOT_FOUND) con un
        // messaggio appropriato
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("[T23 Controller] Notification not
found");
    }
}

```

1. Cerca il profilo utente usando l'email.
2. Se il profilo non esiste, restituisce 400 BAD_REQUEST.
3. Recupera l'ID della notifica e lo cerca tra quelle dell'utente.
4. Se la notifica esiste, chiama markNotificationAsRead per marcarla come letta e restituisce 200 OK.
5. Se non trovata, restituisce 400 NOT_FOUND.

DELETE /delete_notification

Elimina una notifica specifica di un utente.



```

● ● ●

@Override
public ResponseEntity<String> deleteNotification(@RequestParam("email") String email,
                                                    @RequestParam("id notifica") String notificationID) {
    // Cerca il profilo dell'utente utilizzando l'email fornita come parametro
    UserProfile profile = userService.findProfileByEmail(email);

    // Controlla se il profilo dell'utente è stato trovato
    if (profile == null) {
        // Se il profilo non esiste, restituisce un errore 400 (BAD_REQUEST) con un messaggio
        // appropriato
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("[T23 Controller] UserProfile not
found");
    }

    try {
        // Converte l'ID della notifica da stringa a tipo Long
        Long notifID = Long.parseLong(notificationID);

        // Verifica che la notifica esista per l'utente specificato
        List<Notification> notifications =
notificationService.getNotificationsByPlayer(profile.getUser().getID());
        boolean found = notifications.stream().anyMatch(notif -> notifID.equals(notif.getId()));

        if (!found) {
            // Se la notifica non esiste, restituisce un errore 404 (NOT_FOUND)
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("[T23 Controller] Notification
not found");
        }

        // Elimina la notifica utilizzando il servizio notifiche
        notificationService.deleteNotification(notifID);
        // Restituisce una risposta 200 (OK) per indicare il successo dell'operazione
        return ResponseEntity.ok("Notification deleted successfully");

    } catch (NumberFormatException e) {
        // Se l'ID della notifica non è valido, restituisce un errore 400 (BAD_REQUEST)
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("[T23 Controller] Invalid
Notification ID");
    }
}

```

1. Cerca il profilo utente con l'email fornita.
2. Se il profilo non esiste, restituisce texttt400 BAD_REQUEST.
3. Converte e valida l'ID della notifica.
4. Verifica che la notifica esista per l'utente specificato.
5. Se trovata, chiama texttdeleteNotification e restituisce texttt200 OK.
6. Se non trovata, restituisce texttt404 NOT_FOUND.
7. Se l'ID non è valido, restituisce texttt400 BAD_REQUEST.

DELETE `/clear_notifications`

Elimina tutte le notifiche di un utente.

```

1  @DeleteMapping("/clear_notifications")
2  public ResponseEntity<String> clearNotifications(@RequestParam("email") String
3      email) {
4      // Cerca il profilo dell'utente utilizzando l'email fornita come parametro
5      UserProfile profile = userService.findProfileByEmail(email);
6      // Controlla se il profilo dell'utente e' stato trovato
7      if (profile == null) {
8          // Se il profilo non esiste, restituisce un errore 400 (BAD_REQUEST)
9          // con un messaggio appropriato
10         return ResponseEntity.status(HttpStatus.BAD_REQUEST)
11             .body("[T23 Controller] UserProfile not found");
12     }
13
14     // Ottiene l'ID dell'utente dal profilo
15     int playerID = profile.getUser().getID();
16
17     // Elimina tutte le notifiche associate all'utente
18     notificationService.clearNotificationsByPlayer(playerID);
19
20     // Restituisce una risposta 200 (OK) per indicare il successo
21     // dell'operazione
22     return ResponseEntity.ok("All notifications cleared successfully");
23 }
```

Listing 4.19: New Notification Endpoint

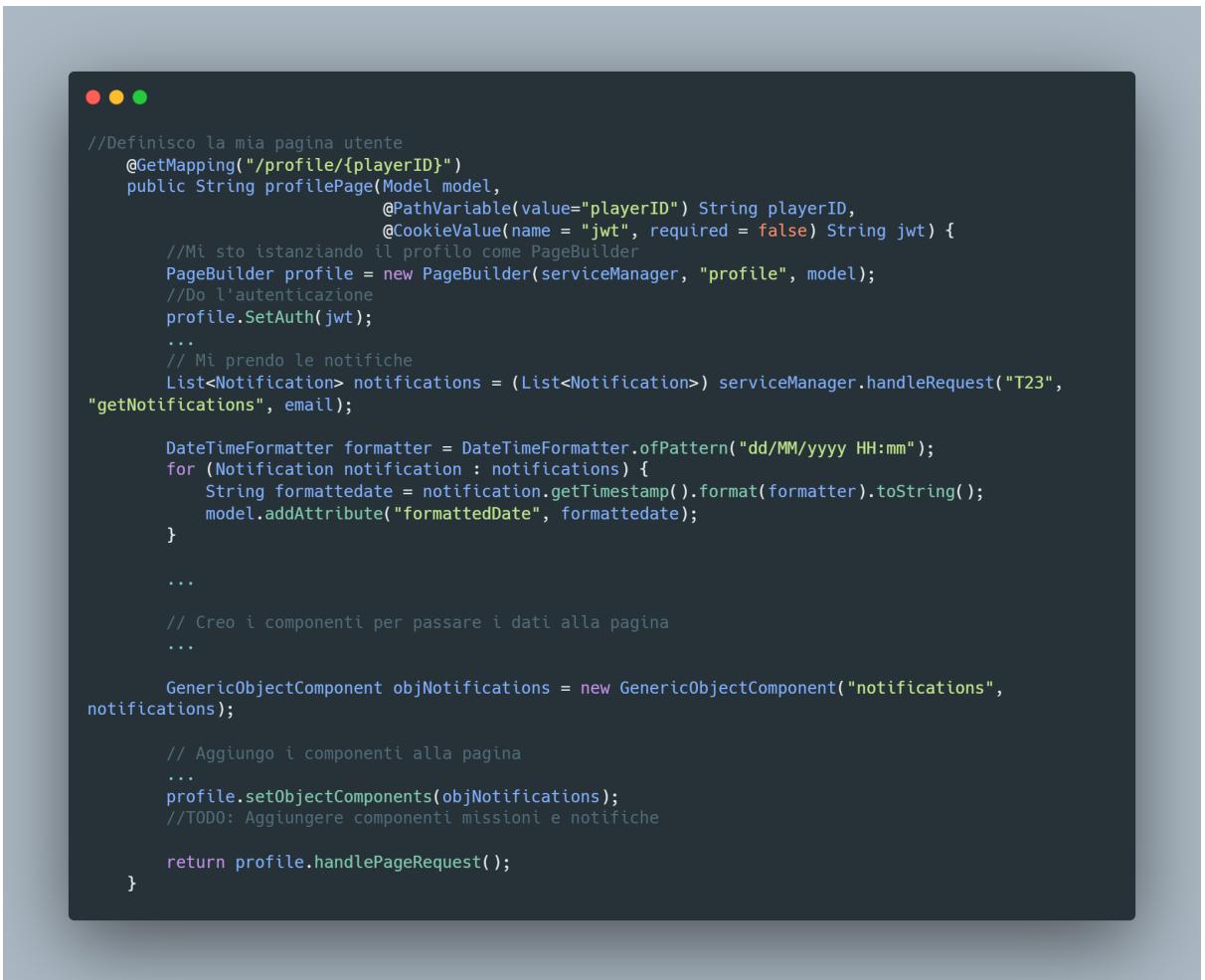
1. Cerca il profilo utente usando l'email.
2. Se il profilo non esiste, restituisce 400 BAD_REQUEST.
3. Recupera l'ID utente dal profilo e chiama `clearNotificationsByPlayer` per cancellare tutte le notifiche associate.
4. Restituisce 200 OK.

Perché abbiamo esposto tutte queste rotte lato T23?

Le notifiche vengono salvate in T23, ma qualunque funzionalità deve essere in grado di personalizzare la notifica da inviare al giocatore e il giocatore deve essere in grado di gestirle come desidera in T5 tramite la sua vista. Seguendo il **principio di disaccoppiamento** del pattern MVC è giusto che il modello di queste sia separato dalla loro vista, inoltre questa scelta è guidata dall'esigenza di **modificabilità** della nostra applicazione.

4.5.6 GuiController in T5

Richiamiamo le notifiche nel profilo:



```

//Definisco la mia pagina utente
@GetMapping("/profile/{playerID}")
public String profilePage(Model model,
    @PathVariable(value="playerID") String playerID,
    @CookieValue(name = "jwt", required = false) String jwt) {
    //Mi sto istanziando il profilo come PageBuilder
    PageBuilder profile = new PageBuilder(serviceManager, "profile", model);
    //Do l'autenticazione
    profile.SetAuth(jwt);
    ...
    // Mi prendo le notifiche
    List<Notification> notifications = (List<Notification>) serviceManager.handleRequest("T23",
        "getNotifications", email);

    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
    for (Notification notification : notifications) {
        String formatteddate = notification.getTimestamp().format(formatter).toString();
        model.addAttribute("formattedDate", formatteddate);
    }
    ...

    // Creo i componenti per passare i dati alla pagina
    ...

    GenericObjectComponent objNotifications = new GenericObjectComponent("notifications",
        notifications);

    // Aggiungo i componenti alla pagina
    ...
    profile.setObjectComponents(objNotifications);
    //TODO: Aggiungere componenti missioni e notifiche

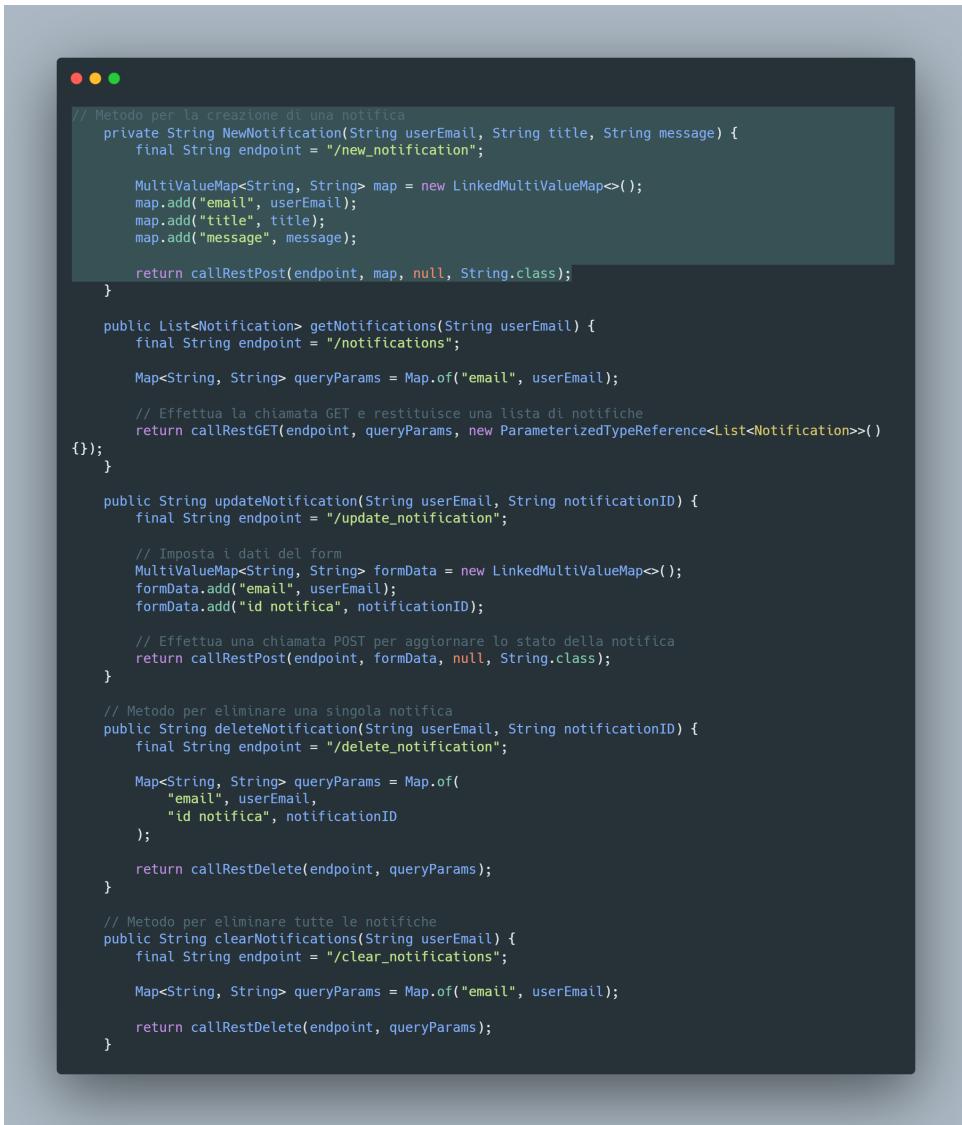
    return profile.handlePageRequest();
}

```

Adesso la pagina profilo è completa! Gestiamo quindi le notifiche lato T5.

4.5.7 T23Service in T5

Creiamo delle azioni che corrispondono alle rotte esposte dal controller di T23.



```

// Metodo per la creazione di una notifica
private String NewNotification(String userEmail, String title, String message) {
    final String endpoint = "/new_notification";

    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add("email", userEmail);
    map.add("title", title);
    map.add("message", message);

    return callRestPost(endpoint, map, null, String.class);
}

public List<Notification> getNotifications(String userEmail) {
    final String endpoint = "/notifications";

    Map<String, String> queryParams = Map.of("email", userEmail);

    // Effettua la chiamata GET e restituisce una lista di notifiche
    return callRestGET(endpoint, queryParams, new ParameterizedTypeReference<List<Notification>>()
    {});
}

public String updateNotification(String userEmail, String notificationID) {
    final String endpoint = "/update_notification";

    // Imposta i dati del form
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<>();
    formData.add("email", userEmail);
    formData.add("id notifica", notificationID);

    // Effettua una chiamata POST per aggiornare lo stato della notifica
    return callRestPost(endpoint, formData, null, String.class);
}

// Metodo per eliminare una singola notifica
public String deleteNotification(String userEmail, String notificationID) {
    final String endpoint = "/delete_notification";

    Map<String, String> queryParams = Map.of(
        "email", userEmail,
        "id notifica", notificationID
    );

    return callRestDelete(endpoint, queryParams);
}

// Metodo per eliminare tutte le notifiche
public String clearNotifications(String userEmail) {
    final String endpoint = "/clear_notifications";

    Map<String, String> queryParams = Map.of("email", userEmail);

    return callRestDelete(endpoint, queryParams);
}

```

I metodi descritti utilizzano **Map** e **MultiValueMap** per gestire i parametri delle chiamate REST, differenziando il loro utilizzo in base al tipo di richiesta (GET, POST, DELETE).

Map per i parametri delle richieste GET o DELETE

Le richieste GET e DELETE passano i parametri tramite la query string dell'URL (**BASEURL** + /endpoint).

- **Map<String, String>** è una struttura dati semplice che mappa chiavi (nomi dei parametri) a valori (valori dei parametri).
- I parametri vengono poi aggiunti alla query string dall'utilità `callRestGET` o `callRestDelete`.

MultiValueMap per i parametri delle richieste POST

Le richieste POST spesso inviano dati nel corpo della richiesta, come form data o JSON.

- **MultiValueMap<String, String>** consente di rappresentare più valori per una stessa chiave, utile nei form data che possono avere chiavi duplicate. È comunemente usata con ‘RestTemplate’ di Spring per inviare dati.
- Metodi come `NewNotification` e `updateNotification` utilizzano **MultiValueMap** perché devono inviare dati al server come form data.

Caratteristica	Map	MultiValueMap
Supporto per chiavi multiple	Ogni chiave ha un solo valore	Ogni chiave può avere più valori
Uso tipico	Parametri di query string	Form data o richieste POST
Librerie supportate	Standard Java Collections	Spring Framework (LinkedMultimap)

Table 4.1: Differenze chiave tra Map e MultiValueMap

Ricordiamoci di registrare le action!

```

1 registerAction("NewNotification", new ServiceActionDefinition(
2     params -> NewNotification((String) params[0], (String) params[1],
3         (String) params[2]),
4         String.class, String.class, String.class));
5 registerAction("GetNotifications", new ServiceActionDefinition(
6     params -> getNotifications((String) params[0]),
7     String.class));
8 registerAction("UpdateNotification", new ServiceActionDefinition(
9     params -> updateNotification((String) params[0], (String) params[1]),
10    String.class, String.class));
11 registerAction("DeleteNotification", new ServiceActionDefinition(
12    params -> deleteNotification((String) params[0], (String) params[1]),
13    String.class, String.class));
14 registerAction("ClearNotifications", new ServiceActionDefinition(
15    params -> clearNotifications((String) params[0]),
16    String.class));

```

Listing 4.20: New Actions

Serve un model per le notifiche, essenziale per serializzare e deserializzare i dati JSON durante le chiamate REST.

4.5.8 Package Model in T5

Il tipo Notification rappresenta un'entità che descrive una notifica, tipicamente con i seguenti campi:

- ‘id’: Identificativo univoco della notifica (Long).
- ‘title’: Titolo della notifica (String).
- ‘message’: Messaggio della notifica (String).
- ‘timestamp’: Data e ora di creazione (LocalDateTime).
- ‘isRead’: Indica se la notifica è stata letta (Boolean).

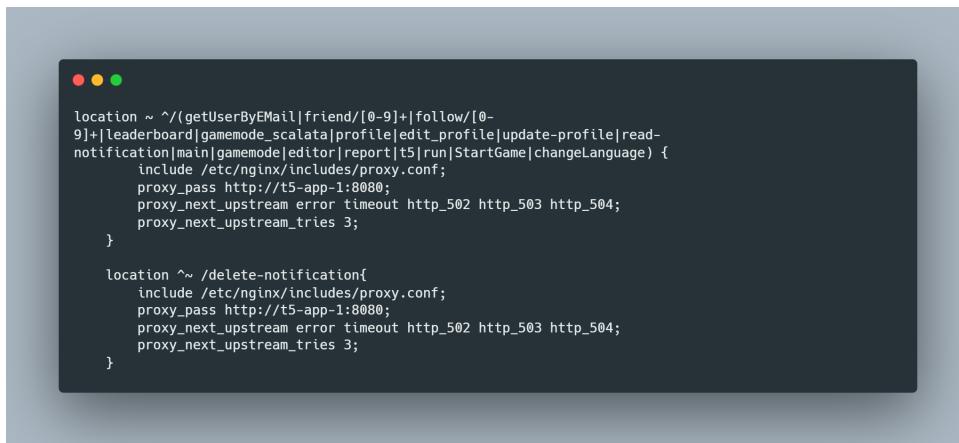
```

1  @Data
2  @Entity
3  @Table(name = "Notifications", schema = "studentsrepo")
4  public class Notification {
5
6      @Id
7      @GeneratedValue(strategy = GenerationType.AUTO)    // Genera un ID univoco
8          automaticamente
9      private Long id;
10
11     private int playerID;
12     private String titolo;
13     private String message;
14     private LocalDateTime timestamp;
15     private boolean isRead;
16         //Costruttore vuoto per Thymeleaf
17     public Notification() {}
18
19     public Notification(int playerID, String titolo, String message, LocalDateTime
20         timestamp, boolean isRead) {
21         this.playerID = playerID;
22         this.titolo = titolo;
23         this.message = message;
24         this.timestamp = timestamp;
25         this.isRead = isRead;
26     }

```

Listing 4.21: New Notification Model

Default.conf in Uigateway



```

location ~ ^/(getUserByEmail|friend/[0-9]+|follow/[0-
9]+|leaderboard|gameMode_scalata|profile|edit_profile|update-profile|read-
notification|main|gameMode|editor|report|t5|run|startGame|changeLanguage) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t5-app-1:8080;
    proxy_next_upstream error timeout http_502 http_503 http_504;
    proxy_next_upstream_tries 3;
}

location ~^/delete-notification{
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t5-app-1:8080;
    proxy_next_upstream error timeout http_502 http_503 http_504;
    proxy_next_upstream_tries 3;
}

```

Sono state aggiunte le nuove rotte nel file di configurazione dell’uigateway, così facendo il sistema è in grado di gestire correttamente le richieste http a quelle rotte e inoltrarle ai rispettivi server. **Attenzione!** Le delete vanno in configurazioni diverse!

4.5.9 Diagrammi di sequenza (dettaglio)

Activity Diagram

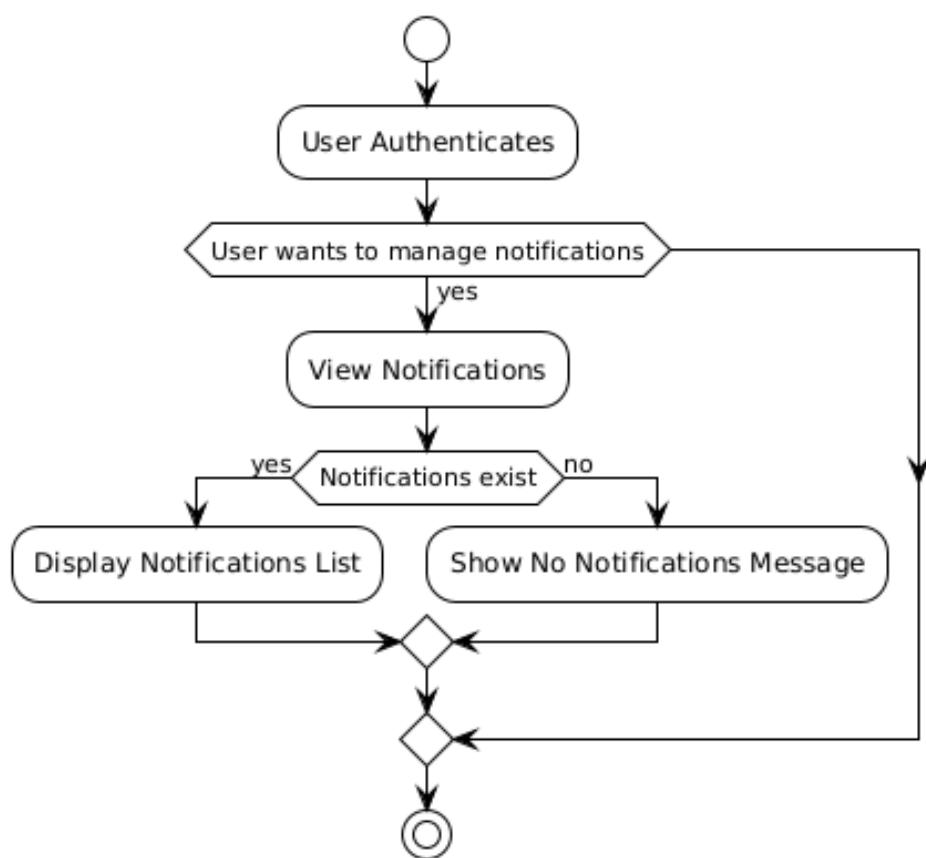


Figure 4.3: Create Notification

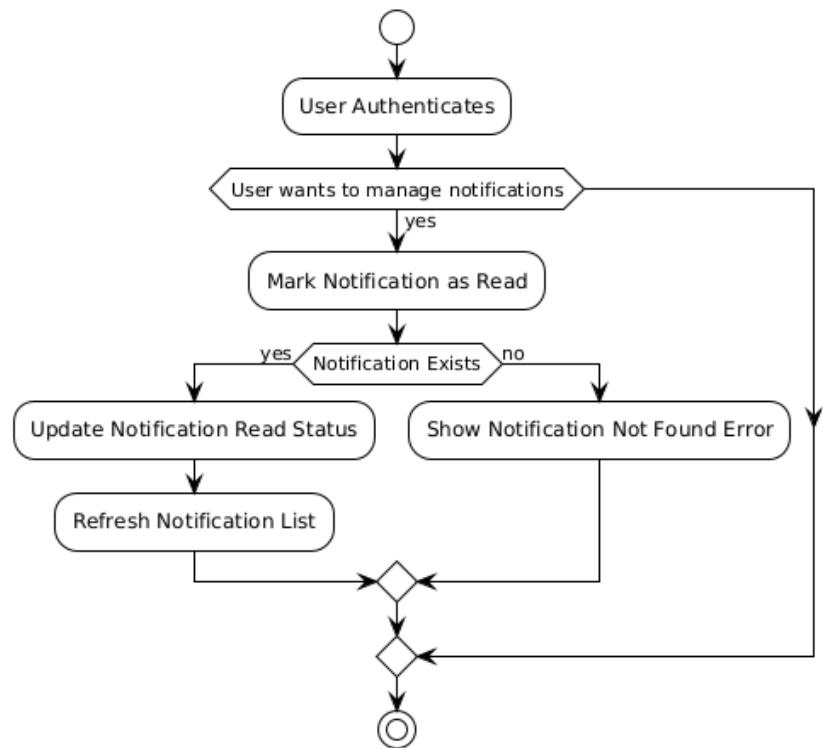


Figure 4.4: View Notification

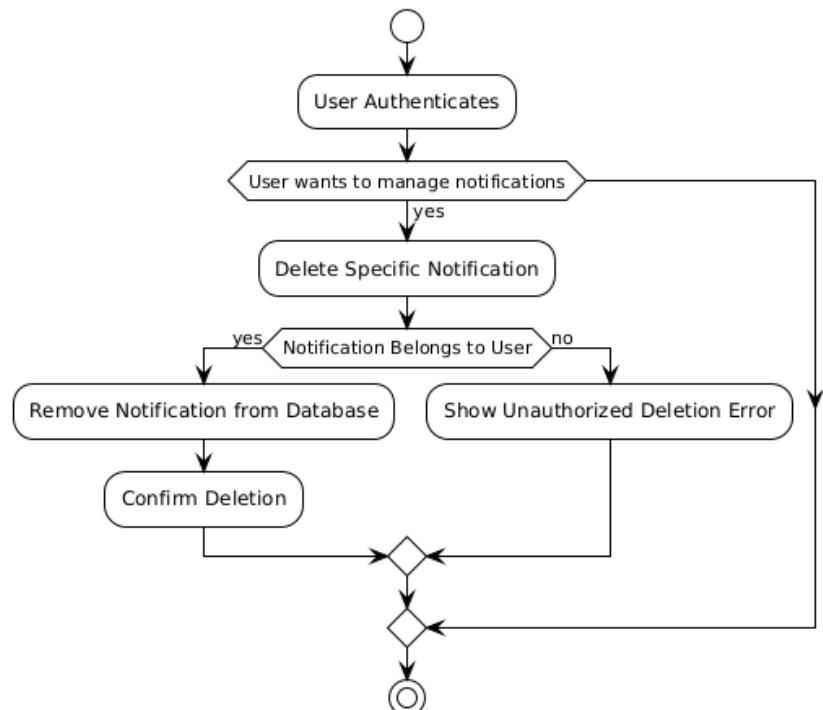


Figure 4.5: Mark As Read Notification

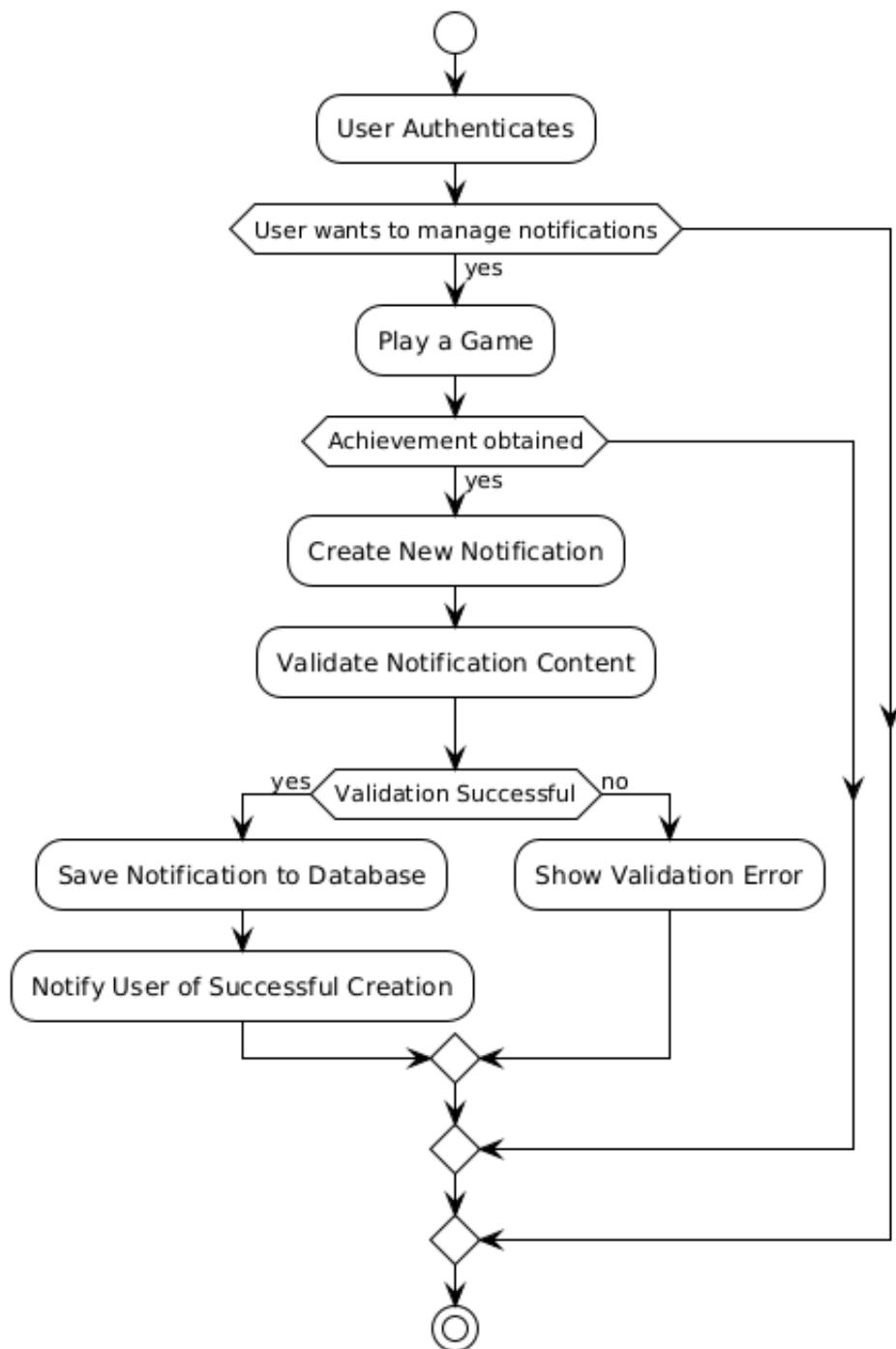
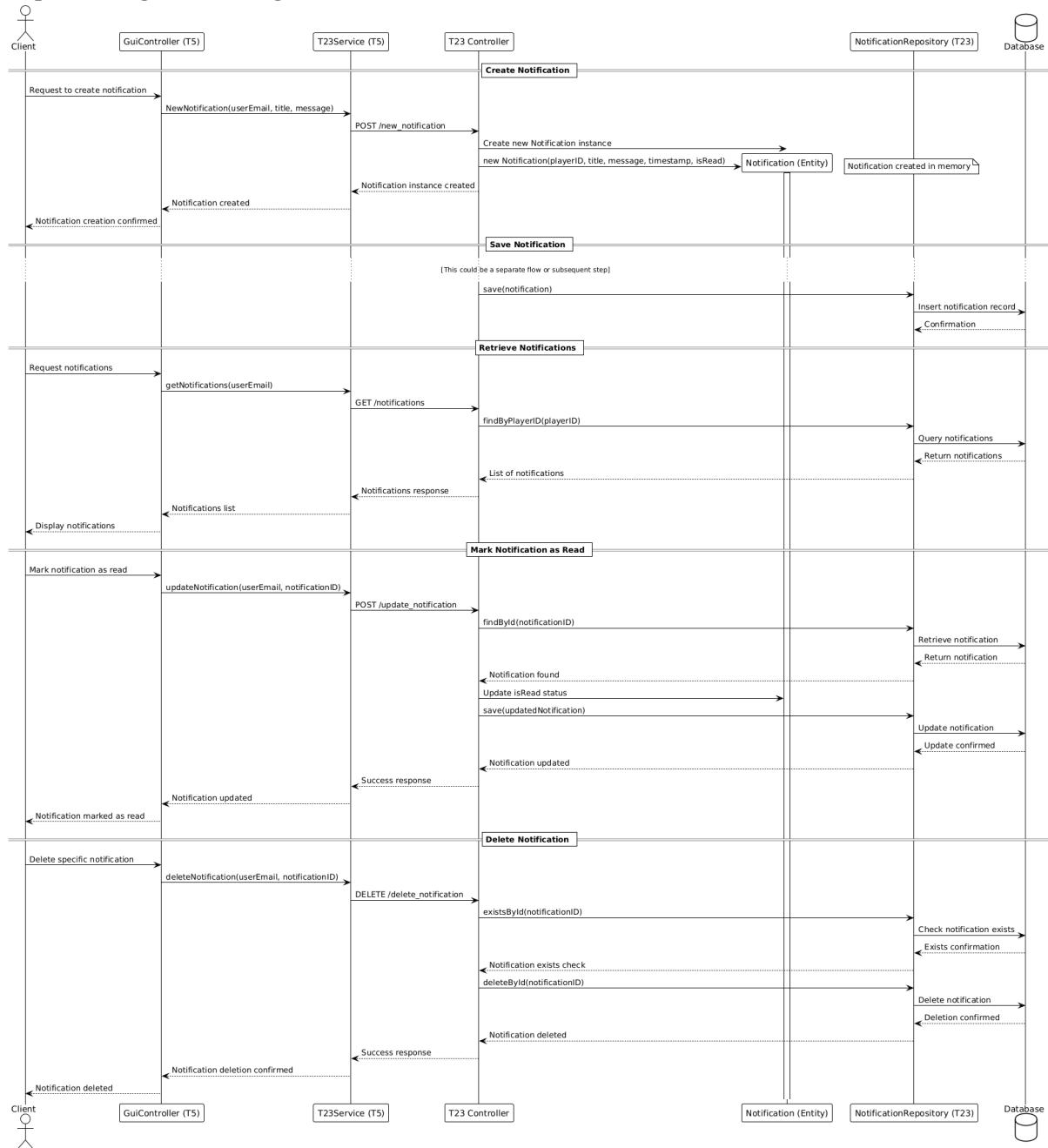
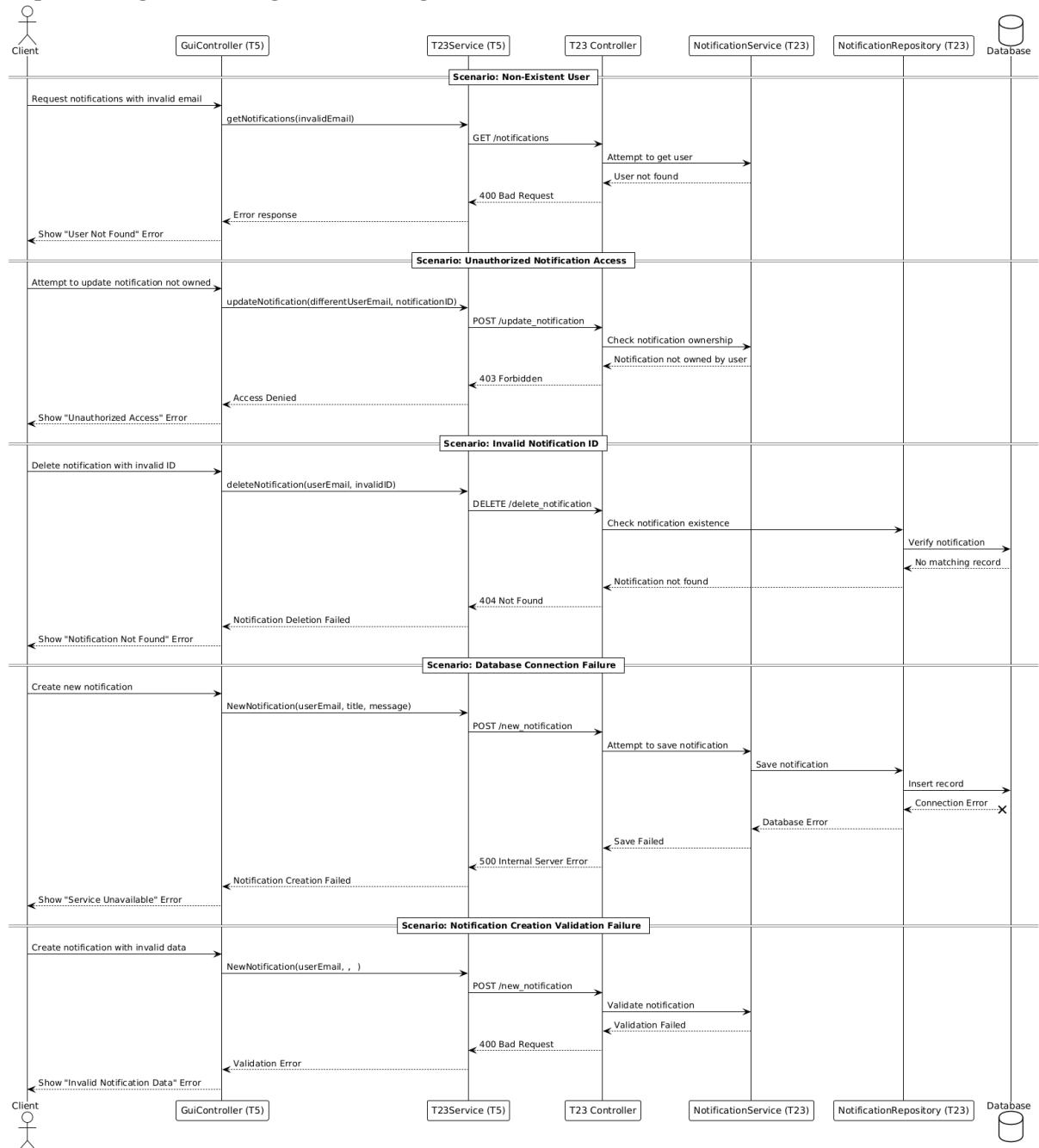


Figure 4.6: Delete Notification

Sequence Diagram di dettaglio



Sequence Diagram di dettaglio: scenari negativi



4.6 Terza iterazione

4.7 Feature 3: Social, follower e following

Nuove Funzionalità

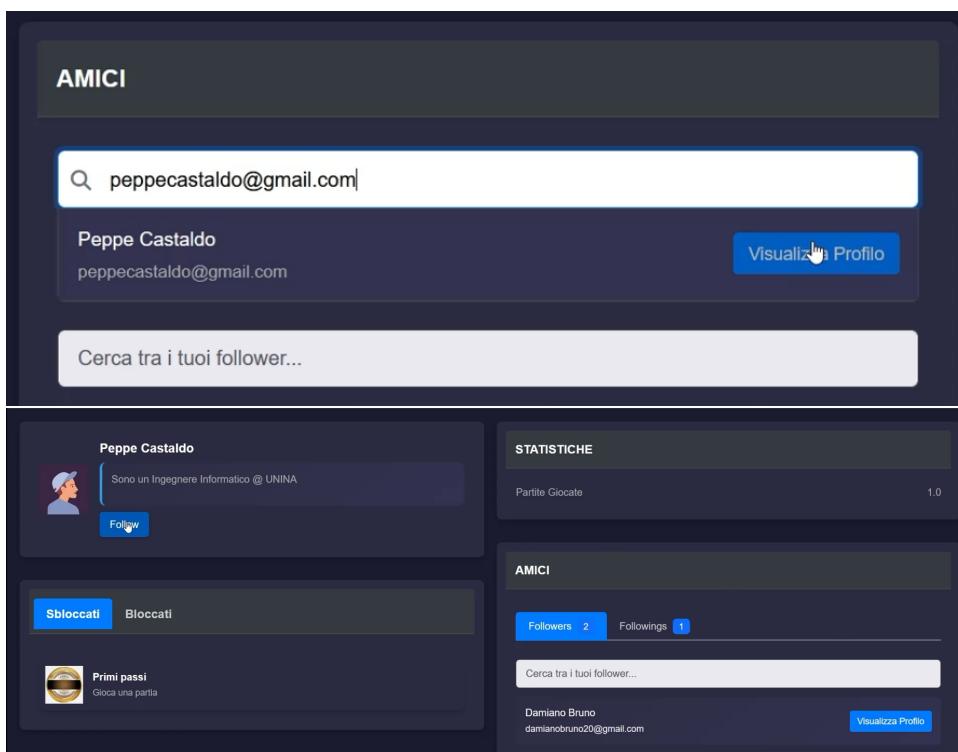
Miglioramenti Interazione Social

1. Ricerca Utenti

- Implementata ricerca dei giocatori tramite email.
- Aggiunta possibilità di seguire gli utenti dai risultati di ricerca.

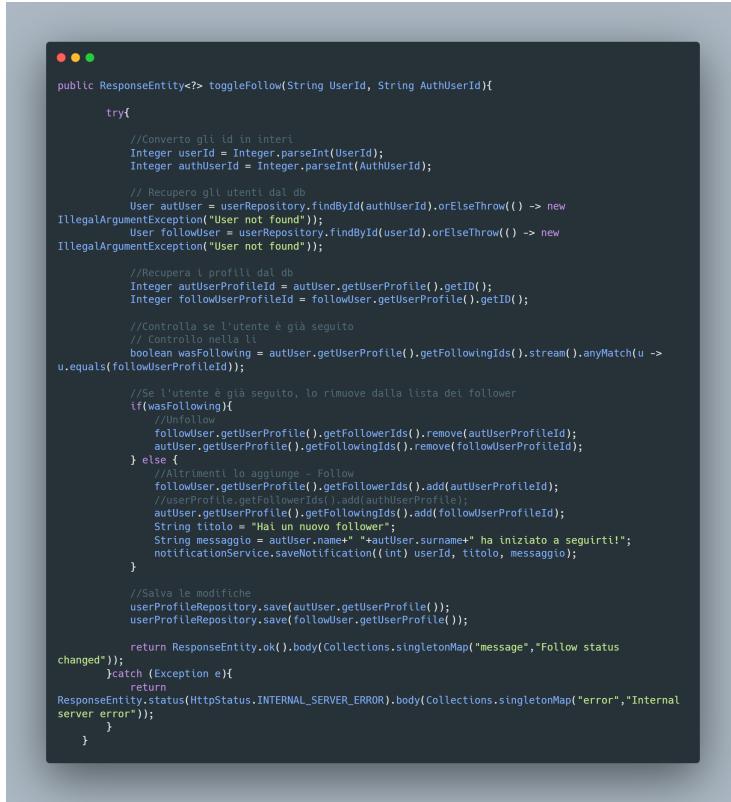
2. Visualizzazione Network Sociale

- Aggiunta vista lista follower.
- Aggiunta vista lista following.
- Migliorata la visualizzazione del profilo utente.
- Implementata funzionalità follow/unfollow dalla vista profilo.



Gestione Follow: UserService in T23

- /add-follow



```
public ResponseEntity<?> toggleFollow(String userId, String authUserId){  
    try{  
        //Converti gli id in interi  
        Integer userId = Integer.parseInt(userId);  
        Integer authUserId = Integer.parseInt(authUserId);  
  
        // Recupero gli utenti dal db  
        User autUser = userRepository.findById(authUserId).orElseThrow(() -> new  
IllegalArgumentException("User not found"));  
        User followUser = userRepository.findById(userId).orElseThrow(() -> new  
IllegalArgumentException("User not found"));  
  
        //Recupera i profili dal db  
        Integer autUserProfileId = autUser.getUserProfile().getId();  
        Integer followUserProfileId = followUser.getUserProfile().getId();  
  
        //Controlla se l'utente è già seguito  
        // Controlla nella lista dei follower  
        boolean wasFollowing = autUser.getUserProfile().getFollowingIds().stream().anyMatch(u ->  
u.equals(followUserProfileId));  
  
        //Se l'utente è già seguito, lo rimuove dalla lista dei follower  
        if(wasFollowing){  
            //Unfollow  
            followUser.getUserProfile().getFollowerIds().remove(autUserProfileId);  
            autUser.getUserProfile().getFollowingIds().remove(followUserProfileId);  
        } else {  
            //Altrimenti lo aggiunge - Follow  
            followUser.getUserProfile().getFollowerIds().add(autUserProfileId);  
            //userProfile.getFollowerIds().add(authUserProfile);  
            autUser.getUserProfile().getFollowingIds().add(followUserProfileId);  
            String titolo = "Ha un nuovo follower";  
            String messaggio = autUser.getName() + " " + autUser.getSurname() + " ha iniziato a seguirti!";  
            notificationService.saveNotification((int) userId, titolo, messaggio);  
        }  
  
        //Salva le modifiche  
        userProfileRepository.save(autUser.getUserProfile());  
        userProfileRepository.save(followUser.getUserProfile());  
  
        return ResponseEntity.ok().body(Collections.singletonMap("message", "Follow status  
changed"));  
    } catch (Exception e){  
        return  
        ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(Collections.singletonMap("error", "Internal  
server error"));  
    }  
}
```

- Endpoint unificato per seguire/smettere di seguire gli utenti.
- Il comportamento cambia in base allo stato precedente del follow.
- Semplifica la funzionalità di follow/unfollow in un unico endpoint.

- /get-followers

```

public List<User> getFollowers(String UserId) {
    Integer userIdInt = Integer.parseInt(UserId);

    // Trova l'utente
    User user = userRepository.findById(userIdInt)
        .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "User not found"));

    // Ottieni gli ID dei following
    List<Integer> followerIds = user.getUserProfile().getFollowerIds();
    System.out.println("Following IDs: " + followerIds);

    if (followerIds == null || followerIds.isEmpty()) {
        return new ArrayList<>();
    }

    // Prova a trovare ogni utente singolarmente per debug
    List<User> followers = new ArrayList<>();
    for (Integer followerId : followerIds) {
        UserProfile userProfile = userProfileRepository.findById(followerId)
            .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "UserProfile not found"));
        User followerUser = userRepository.findByUserProfile(userProfile);
        //Optional<User> followerUser =
        userRepository.findById(userRepository.findByUserProfile(userProfile).getId());
        if (followerUser != null) {
            followers.add(followerUser);
            System.out.println("Found user with id " + followerId + ": " + followerUser.getName());
        } else {
            System.out.println("User with id " + followerId + " not found");
        }
    }

    System.out.println("Found following: " + followers);
    return followers;
}

```

- Recupera la lista completa degli utenti che seguono l'utente in input.
- Supporta la funzionalità di visualizzazione dei follower.

- **/get-following**

```

public List<User> getFollowing(String UserId) {
    Integer userIdInt = Integer.parseInt(UserId);

    // Trova l'utente
    User user = userRepository.findById(userIdInt)
        .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "User not found"));

    // Ottieni gli ID dei following
    List<Integer> followingIds = user.getUserProfile().getFollowingIds();
    System.out.println("Following IDs: " + followingIds);

    if (followingIds == null || followingIds.isEmpty()) {
        return new ArrayList<>();
    }

    // Prova a trovare ogni utente singolarmente per debug
    List<User> following = new ArrayList<>();
    for (Integer followingId : followingIds) {
        UserProfile userProfile = userProfileRepository.findById(followingId)
            .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "UserProfile not found"));
        User followingUser = userRepository.findByUserProfile(userProfile);
        //Optional<User> followingUser =
        userRepository.findById(userRepository.findByUserProfile(userProfile).getId());
        if (followingUser != null) {
            following.add(followingUser);
            System.out.println("Found user with id " + followingId + ": " + followingUser.getName());
        } else {
            System.out.println("User with id " + followingId + " not found");
        }
    }

    System.out.println("Found following: " + following);
    return following;
}

```

- Recupera la lista completa degli utenti seguiti dall'utente in input.
- Permette la visualizzazione della lista dei following.

Controller in T23



```
//Rotta per Seguire o smettere di seguire un utente
@PostMapping("/add-follow")
public ResponseEntity<User> toggleFollow(@RequestParam("targetUserId") String targetUserId,
                                         @RequestParam("authUserId") String authUserId) {
    System.out.println(targetUserId);
    System.out.println(authUserId);
    return userService.toggleFollow(targetUserId, authUserId);
}

//Rotta per ottenere i followers di un utente
@GetMapping("/get-followers")
public List<User> getFollowers(@RequestParam("userId") String userId) {
    return userService.getFollowers(userId);
}

//Rotta per ottenere gli utenti seguiti da un utente
@GetMapping("/get-following")
public List<User> getFollowing(@RequestParam("userId") String userId) {
    return userService.getFollowing(userId);
}

//Modifica 04/12/2024 Giuseppe: Aggiunta roitta
@PostMapping("/studentsByIds")
public ResponseEntity<List<String>> getStudentTeam(@RequestBody List<String> idsStudenti){
    return userService.getStudentTeam(idsStudenti);
}
```

4.7.1 Gestione Profilo (T5)

T23Service in T5



```
// Metodo per follow/unfollow di un utente
public String followUser(Integer targetUserId, Integer authUserId) {
    final String endpoint = "/add-follow";
    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add("targetUserId", String.valueOf(targetUserId));
    map.add("authUserId", String.valueOf(authUserId));
    return callRestPost(endpoint, map, null, String.class);
}

public List<User> getFollowers(String userId) {
    final String endpoint = "/get-followers";
    Map<String, String> queryParams = Map.of("userId", userId);
    return callRestGET(endpoint, queryParams, new ParameterizedTypeReference<List<User>>() {});
}

public List<User> getFollowing(String userId) {
    final String endpoint = "/get-following";
    Map<String, String> queryParams = Map.of("userId", userId);
    return callRestGET(endpoint, queryParams, new ParameterizedTypeReference<List<User>>() {});
}
```

UserProfileService in T5



```
public List<Integer> getFollowingList(int playerId){
    // Mi prendo l'id del giocatore
    int userId = playerId;

    // Mi prendo prima tutti gli utenti
    @SuppressWarnings("unchecked")
    List<User> users = (List<com.g2.Model.User>)serviceManager.handleRequest("T23", "GetUsers");

    // Mi prendo l'utente che mi interessa con l'id
    User user = users.stream().filter(u -> u.getId() == userId).findFirst().orElseThrow(() -> new
    RuntimeException("User not found"));

    return user.getUserProfile().getFollowingList();
}

public List<Integer> getFollowersList(int playerId){
    // Mi prendo l'id del giocatore
    int userId = playerId;

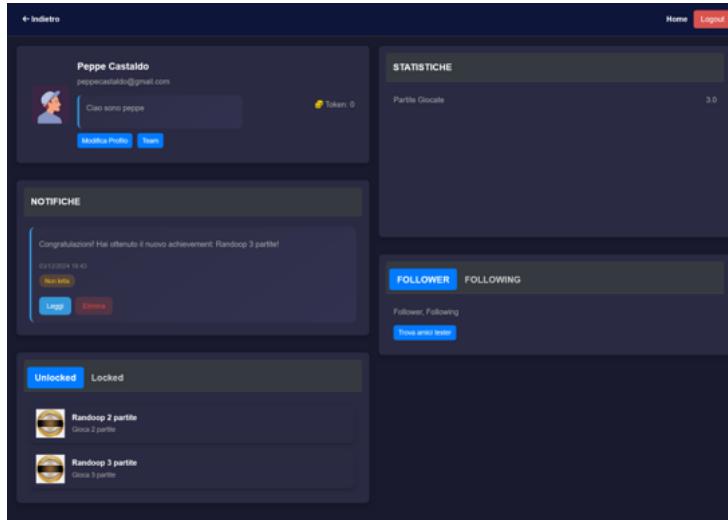
    // Mi prendo prima tutti gli utenti
    @SuppressWarnings("unchecked")
    List<User> users = (List<com.g2.Model.User>)serviceManager.handleRequest("T23", "GetUsers");

    // Mi prendo l'utente che mi interessa con l'id
    User user = users.stream().filter(u -> u.getId() == userId).findFirst().orElseThrow(() -> new
    RuntimeException("User not found"));

    return user.getUserProfile().getFollowersList();
}
```

GuiController in T5

- Modificata /profile/{playerID}



- Aggiunta sezione Amici.

```

// Visualizzazione profilo di un amico
@GetMapping("/friend/{playerID}")
public String friendProfilePage(Model model, @PathVariable(value="playerID") String playerID,
@CookieValue(name = "jwt", required = false) String jwt) {
try {
    // Istanzo il profilo come PageBuilder
    PageBuilder profile = new PageBuilder(serviceManager, "friend_profile", model);
    // Autenticazione
    profile.setAuth(jwt);
    // Converti l'ID del giocatore
    int userId = Integer.parseInt(playerID);
    // Recupero l'utente
    List<User> users = ((List<User>) serviceManager.handleRequest("T23", "GetUsers"));
    User user = users.stream()
        .filter(u -> u.getId() == userId)
        .findFirst()
        .orElseThrow(() -> new RuntimeException("Utente non trovato"));
    // Recupero i dati pubblici da visualizzare
    String username = user.getUsername();
    String surname = user.getSurname();
    // Recupero immagine e bio pubbliche
    String image = userProfileService.getProfilePicture(userId);
    String bio = userProfileService.getProfileBio(userId);
    // Recupero i progressi degli achievement pubblici
    List<AchievementProgress> achievementProgresses =
achievementService.getProgressesByPlayer(userId);
    // Divido i progressi degli achievement in "locked" e "unlocked"
    List<AchievementProgress> unlockedAchievements = achievementProgresses.stream()
        .filter(a -> a.getProgress() >= a.getProgressRequired())
        .toList();
    List<AchievementProgress> lockedAchievements = achievementProgresses.stream()
        .filter(a -> a.getProgress() < a.getProgressRequired())
        .toList();
    // Recupero le statistiche pubbliche del giocatore
    List<StatisticProgress> statisticProgresses =
achievementService.getStatisticsByPlayer(userId);
    List<Statistic> allStatistics = achievementService.getStatistics();
    Map<String, Statistic> idToStatistic = new HashMap<>();
    for (Statistic stat : allStatistics) {
        idToStatistic.put(stat.getId(), stat);
    }
    // Mi prendo la lista dei follower e dei following
    List<User> followersList = new ArrayList<>();
    List<User> followingList = new ArrayList<>();
    followersList = ((List<User>) serviceManager.handleRequest("T23", "getFollowers",
String.valueOf(userId)));
    followingList = (List<User>) serviceManager.handleRequest("T23", "getFollowing",
String.valueOf(userId));
    Integer followersListSize = followersList.size();
    Integer followingListSize = followingList.size();
    // Decodifico l'utente che ha fatto questa richiesta
    byte[] decodedUserObj = Base64.getDecoder().decode(jwt.split("\\.")[1]);
    String decodedUserJson = new String(decodedUserObj), StandardCharsets.UTF_8;
    ObjectMapper mapper = new ObjectMapper();
    @SuppressWarnings("unchecked")
    Map<String, Object> map = mapper.readValue(decodedUserJson, Map.class);
    String authUserId = map.get("userId").toString();
    // Ottengo l'ID dello UserProfile dell'utente autenticato
    Integer authUserProfileID = userProfileService.getProfileID(Integer.parseInt(authUserId));
    // Ottengo l'ID del profilo dell'utente visualizzato
    Integer userProfileID = userProfileService.getProfileID(userId);
    // Verificiamo se l'utente autenticato è tra i follower dell'utente visualizzato
    boolean isFollowing = userProfileService.getFollowersList(userId) != null &&
        userProfileService.getFollowersList(userId).stream()
            .anyMatch(f -> f.equals(authUserProfileID));
    System.out.println("Sto cercando le informazioni dell'utente: "+userId);
    System.out.println("Id profilo: "+userProfileID);
    System.out.println("Lista dei follower: "+userProfileService.getFollowersList(userId));
    System.out.println("Lista dei following: "+userProfileService.getFollowingList(userId));
    System.out.println("isFollowing: "+isFollowing);
    // Creo i componenti per passare i dati pubblici alla pagina
    GenericObjectComponent objUsername = new GenericObjectComponent("username", username);
    GenericObjectComponent objSurname = new GenericObjectComponent("surname", surname);
    GenericObjectComponent objImage = new GenericObjectComponent("profile", image);
    GenericObjectComponent objBio = new GenericObjectComponent("bio", bio);
    GenericObjectComponent objFollowingList = new GenericObjectComponent("followingList",
followingList);
    GenericObjectComponent objFollowersList = new GenericObjectComponent("followersList",
followersList);
    GenericObjectComponent objFollowingListSize = new
GenericObjectComponent("followingListSize", followingListSize);
    GenericObjectComponent objFollowersListSize = new
GenericObjectComponent("followersListSize", followersListSize);
    GenericObjectComponent objUnlockedAchievements = new
GenericObjectComponent("unlockedAchievements", unlockedAchievements);
    GenericObjectComponent objLockedAchievements = new
GenericObjectComponent("lockedAchievements", lockedAchievements);
    GenericObjectComponent objStatistics = new
GenericObjectComponent("statistics", allStatistics);
    GenericObjectComponent objIdToStatistic = new GenericObjectComponent("idToStatistic",
idToStatistic);
    GenericObjectComponent objIsFollowing = new GenericObjectComponent("isFollowing",
isFollowing);
    GenericObjectComponent objUserId = new GenericObjectComponent("userId", userId); //friendID
    // Aggiungo i componenti alla pagina
    profile.setObjectComponents(objUsername);
    profile.setObjectComponents(objSurname);
    profile.setObjectComponents(objImage);
    profile.setObjectComponents(objBio);
    profile.setObjectComponents(objFollowingList);
    profile.setObjectComponents(objFollowersList);
    profile.setObjectComponents(objFollowingListSize);
    profile.setObjectComponents(objFollowersListSize);
    profile.setObjectComponents(objUnlockedAchievements);
    profile.setObjectComponents(objLockedAchievements);
    profile.setObjectComponents(objStatistics);
    profile.setObjectComponents(objIdToStatistic);
    profile.setObjectComponents(objIsFollowing);
    profile.setObjectComponents(objUserId);
    return profile.handlePageRequest();
} catch (NumberFormatException e) {
    System.out.println("(friend) ID utente non valido: " + e.getMessage());
    return "error";
} catch (RuntimeException e) {
    System.out.println("(friend) Utente non trovato: " + e.getMessage());
    return "error";
} catch (Exception e) {
    System.out.println("(friend) Errore generico: " + e.getMessage());
    return "error";
}
}

```

- Nuovo endpoint per visualizzare i profili di altri utenti.
- Permette l’interazione sociale tra utenti.

- **/follow/{playerID}**



```

// Seguire o smettere di seguire un utente
@PostMapping("/follow/{playerID}")
@ResponseBody
public ResponseEntity<?> toggleFollow(@PathVariable(value="playerID") String playerId,
                                         @CookieValue(name = "jwt", required = false) String jwt) {
    try{
        // Converto l'ID del giocatore di cui voglio fare il follow/unfollow
        Integer userId = Integer.parseInt(playerId);

        // Decodifica JWT per ottenere l'ID dell'utente autenticato
        byte[] decodedUserObj = Base64.getDecoder().decode(jwt.split("\\.")[1]);
        String decodedUserJson = new String(decodedUserObj, StandardCharsets.UTF_8);
        ObjectMapper mapper = new ObjectMapper();
        @SuppressWarnings("unchecked")
        Map<String, Object> map = mapper.readValue(decodedUserJson, Map.class);
        String authUserIdString = map.get("userId").toString();
        Integer authUserId = Integer.parseInt(authUserIdString);

        // Chiamo il servizio per seguire o smettere di seguire l'utente
        String result = (String) serviceManager.handleRequest("T23", "followUser", userId,
                                                               authUserId);

        return ResponseEntity.ok(result);
    }catch(Exception e){
        System.out.println("(follow) Errore generico: " + e.getMessage());
        return ResponseEntity.badRequest().body("Errore generico");
    }
}

```

- Endpoint dedicato per le azioni di follow/unfollow.
- Gestisce le relazioni tra utenti.

4.8 Issue: Partite giocate non salvate nel database

Notifica Achievement

Una volta implementato il servizio di notifica, abbiamo provato ad usarlo per notificare l'utente dell'ottenimento di un Achievement. Vogliamo che la notifica arrivi nel momento in cui la partita finisce e vengono quindi aggiornati i progressi degli achievement. Se presenti, vogliamo che il sistema mandi la relativa notifica.

Achievement Notification

Nella versione iniziale del software, i progressi degli Achievement vengono calcolati nella rotta \save_data del GuiController.java. Abbiamo quindi provato a:

1. Aggiornare AchievementService implementando una nuova funzione, così da inviare una notifica ad hoc per gli achievement:

```

1   public void updateNotificationsForAchievements(String userEmail,
2         List<AchievementProgress> newAchievements) {
3     for (AchievementProgress achievement : newAchievements) {
4       String titolo = "Nuovo Achievement";
5       String message = "Congratulazioni! Hai ottenuto il nuovo
6           achievement: " + achievement.Name + "!";
7       serviceManager.handleRequest("T23", "NewNotification", userEmail,
8             titolo, message);
9     }
10 }
```

Listing 4.22: AchievementService.java

2. Richiamare questo metodo nella rotta \save_data del GuiController:

```

1  @PostMapping("/save-data")
2  public ResponseEntity<String> saveGame(@RequestParam("playerId") int
3      playerId,
4          @RequestParam("robot") String robot,
5          @RequestParam("classe") String classe,
6          @RequestParam("difficulty") String difficulty,
7          @RequestParam("gamemode") String gamemode,
8          @RequestParam("username") String username,
9          @RequestParam("selectedScalata") Optional<Integer>
10         selectedScalata,
11         HttpServletRequest request) {
12     if (!request.getHeader("X-UserID")
13         .equals(String.valueOf(playerId))) {
14         return ResponseEntity.badRequest()
15             .body("Unauthorized");
16     }
17     DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");
18     LocalTime oraCorrente = LocalTime.now();
19     String oraFormattata = oraCorrente.format(formatter);
20     GameDataWriter gameDataWriter = new GameDataWriter();
21     Game g = new Game(playerId, gamemode, "nome", difficulty, username);
22     g.setData_creazione(LocalDate.now());
23     g.setOra_creazione(oraFormattata);
24     g.setClasse(classe);
25     g.setUsername(username);
26     JSONObject ids = gameDataWriter.saveGame(g, username, selectedScalata);
27     if (ids == null) {
28         return ResponseEntity.badRequest().body("Bad Request");
29     }
30 }
```

```
28 System.out.println("Checking achievements...");
29 List<User> users = (List<com.g2.Model.User>)serviceManager
30     .handleRequest("T23", "GetUsers");
31 User user = users.stream().filter(u -> u.getId() ==
32     playerId).findFirst().orElse(null);
33 String email = user.getEmail();
34 List<AchievementProgress> newAchievements = achievementService
35     .updateProgressByPlayer(playerId);
36 achievementService
37     .updateNotificationsForAchievements(email, newAchievements);
38 return ResponseEntity.ok(ids.toString());
}
```

Listing 4.23: GuiController.java

Ma a seguito di un merge, il sistema non funziona più. Il sistema in realtà non salva neanche le partite nel database!

Come verificare che in T4 vengano salvate le partite

In T4 il database utilizzato è PostgreSQL e il T4 che lo gestisce è implementato in Go. Per aprire questo database basta aprire il cmd, quindi scrivere:

- Comando iniziale: `docker exec -it t4-g18-db-1 bash` - Esegue una shell bash interattiva all'interno del container Docker chiamato "t4-g18-db-1".
 - Seguito da: `psql -U postgres -d postgres` - Connette al database PostgreSQL usando l'utente "postgres" sul database "postgres".
 - Il comando `\dt` viene eseguito all'interno di psql e mostra la lista delle tabelle nel database.

Procediamo quindi con la nostra query dopo aver giocato una partita.

```
C:\Users\ladyC> docker exec -it t4-g18-db-1 bash
afe03e9268a7:/# psql -U postgres -d postgres
psql (14.10)
Type "help" for help.

postgres=# \dt
              List of relations
 Schema |           Name            | Type | Owner
-----+---------------------+-----+-----
 public |      games          | table | postgres
 public |   metadata         | table | postgres
 public | player_games       | table | postgres
 public | player_has_category_achievement | table | postgres
 public |   players          | table | postgres
 public |    robots           | table | postgres
 public |     rounds          | table | postgres
 public | scalata_games      | table | postgres
 public |     turns           | table | postgres
(9 rows)
```

Perché non salva le partite?

La linea che dovrebbe salvare le partite è `JSONObject ids = gameDataWriter.saveGame(g, username, selectedScalata);`, andiamo a vedere cosa fa nella classe `GameDataWriter` e vediamo questo:

```

1  /*
2  * Questo e' codice legacy non piu' utilizzato !
3  */
4  public JSONObject saveGame(Game game, String username, Optional<Integer>
5      selectedScalata) {
6      try {
7          String time = ZonedDateTime.now(ZoneOffset.UTC)
8              .format(DateTimeFormatter.ISO_INSTANT);
9          JSONObject obj = new JSONObject();
10         ...
11     }

```

Listing 4.24: GameDataWriter.java

Quindi `\save_data` non viene effettivamente mai utilizzata e la logica del salvataggio delle partite è stata spostata in `GameController`.

Abbiamo quindi spostato la logica di aggiornamento degli Achievement e delle notifiche in `GameController`, ma ci resta da capire cosa succede alle partite giocate e perché il nuovo controller non le salva nel database.

GameController

Il controller espone varie rotte, quella che dovrebbe creare le partite è:

```

1 @PostMapping("/StartGame")
2 public ResponseEntity<String> StartGame(@RequestParam String playerId,
3     @RequestParam String type_robot,
4     @RequestParam String difficulty,
5     @RequestParam String mode,
6     @RequestParam String underTestClassName) {
7
8     try {
9         GameFactoryFunction gameConstructor = gameRegistry.get(mode);
10        if (gameConstructor == null) {
11            logger.error("[GAMECONTROLLER] /StartGame errore modalita' non
12                esiste/non registrata");
13            return createErrorResponse("[/StartGame] errore modalita' non
14                esiste/non registrata", "0");
15        }
16        GameLogic gameLogic = activeGames.get(playerId);
17        if (gameLogic == null) {
18            gameLogic = gameConstructor.create(this.serviceManager, playerId,
19                underTestClassName, type_robot, difficulty, mode);
20            //MODIFICA DI CUI SI PARLA
21            gameLogic.CreateGame(); // era commentata!!
22            activeGames.put(playerId, gameLogic);
23            logger.info("[GAMECONTROLLER] [StartGame] Partita creata con
24                successo.");
25        }
26        String currentMode = gameLogic.getClass().getSimpleName();
27        logger.info("[GAMECONTROLLER] [StartGame] Partita gia' esistente modalita':
28            " + currentMode);
29
30        boolean isGameExisting = currentMode.equals(mode) &&
31            gameLogic.CheckGame(type_robot, difficulty, underTestClassName);
32    }

```

```

28     String errorMessage = null;
29     String errorCode = null;
30     if (isGameExisting) {
31         errorMessage = "errore esiste già la partita";
32         errorCode = "2";
33     } else {
34         errorMessage = "errore l'utente ha cambiato le impostazioni della
35             partita";
36         errorCode = "1";
37         activeGames.remove(playerId);
38         gameLogic = gameConstructor.create(this.serviceManager, playerId,
39             underTestClassName, type_robot, difficulty, mode);
40         activeGames.put(playerId, gameLogic);
41     }
42     logger.error("[GAMECONTROLLER] [StartGame] " + errorMessage);
43     return createErrorResponse(errorMessage, errorCode);
44 } catch (Exception e) {
45     logger.error("[GAMECONTROLLER] [StartGame] errore: ", e);
46     return createErrorResponse("[StartGame]" + e.getMessage(), "3");
}
}

```

Listing 4.25: GameController.java

Ci accorgiamo che `gameLogic.CreateGame()`; era commentata. Il log segnalava un successo che non corrispondeva al vero. Infatti analizzando il codice di `CreateGame` in `GameLogic`:

```

1 protected void CreateGame() {
2     String Time = ZonedDateTime.now(ZoneOffset.UTC)
3         .format(DateTimeFormatter.ISO_INSTANT);
4     this.GameID = (int) serviceManager.handleRequest("T4", "CreateGame", Time,
5         "difficulty", "name", this.gamemode, this.PlayerID);
6     this.RoundID = (int) serviceManager.handleRequest("T4", "CreateRound",
7         this.GameID, this.ClasseUT, Time);
}

```

Listing 4.26: GameLogic.java

Ci rendiamo conto che era questa la funzione che creava la partita nel database.

Problema della persistenza delle partite

Nella nuova versione del software, viene conservata la partita in corso in una lista di `activeGames`. Questo significa che non abbiamo più un `endGame` come accadeva prima per salvare la partita nel DB, calcolare il progresso degli achievement e quindi inviare la notifica.

— Questa nota implica che a prescindere dalle nostre modifiche le partite non verranno più salvate come prima e non possibile recuperare la logica di come avveniva la chiusura delle partita prima della modifica di Marano: è cambiata la logica! Quello che accade osservando il database è che il gioco viene creato e ne vengono incrementati i turni, ma il gioco non viene mai effettivamente chiuso, quindi se cerchiamo i games nel database non troviamo quello che ci aspettavamo di trovare prima delle modifiche. I dati adesso dovranno essere prelevati dai turni e dalla partita aperta da cui quei turni provengono.

In corrispondenza della rotta `/run` troviamo:

```

1 @PostMapping(value = "/run", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
2 public ResponseEntity<String> Runner(@RequestParam(value = "testingClassName",
3     required = false, defaultValue = "") String testingClassName,
4     @RequestParam("playerId") String playerId,
5     @RequestParam("isGameEnd") Boolean isGameEnd,
6     @RequestParam(value = "eliminaGame", required = false, defaultValue =
7         "false") Boolean eliminaGame) {

```

```

6   try {
7     ...
8     return gestisciPartita(userData, gameLogic, isGameEnd, robotScore, playerId);
9   } catch (Exception e) {
10    logger.error("[GAMECONTROLLER] /run: errore", e);
11    return createErrorResponse("[/RUN] " + e.getMessage(), "2");
12  }
13}

```

Listing 4.27: GameController.java

La funzione `gestisciPartita` dovrebbe quindi aggiornare i dati della partita terminata e quindi possiamo qui inserire la logica degli Achievement che prima avevamo inserito nel `GuiController`.

```

1 private ResponseEntity<String> gestisciPartita(Map<String, String> userData,
2   GameLogic gameLogic, Boolean isGameEnd, int robotScore, String playerId) {
3   if (userData.get("coverage") != null && !userData.get("coverage").isEmpty()) {
4     // Calcolo copertura e punteggio utente
5     ...
6     gameLogic.playTurn(userScore, robotScore);
7     if (isGameEnd || gameLogic.isGameEnd()) {
8       activeGames.remove(playerId);
9       logger.info("[GAMECONTROLLER] /run: risposta inviata con GameEnd
10      true");
11
12      List<User> users = (List<User>) serviceManager.handleRequest("T23",
13        "GetUsers");
14      Long userId = Long.parseLong(playerId);
15      User user = users.stream().filter(u -> u.getId() ==
16        userId).findFirst().orElse(null);
17      String email = user.getEmail();
18      List<AchievementProgress> newAchievements = achievementService
19        .updateProgressByPlayer(userId.intValue());
20      achievementService
21        .updateNotificationsForAchievements(email
22          , newAchievements);
23      return createResponseRun(userData, robotScore, userScore, true,
24        lineCoverage, branchCoverage, instructionCoverage);
25    } else {
26      logger.info("[GAMECONTROLLER] /run: risposta inviata con GameEnd
27      false");
28      return createResponseRun(userData, robotScore, userScore, false,
29        lineCoverage, branchCoverage, instructionCoverage);
30    }
31  } else {
32    logger.info("[GAMECONTROLLER] /run: risposta inviata errori di
33      compilazione");
34    return createResponseRun(userData, 0, 0, false, null, null, null);
35  }
36}

```

Listing 4.28: GameController.java

Gli Achievement vengono aggiornati! Cerchiamo le notifiche nel database MySQL di T23.

Come verificare che in T23 vengano salvate le notifiche relative all'utente

```
PS C:\Users\ladyc> docker exec -it t23-g1-db-1 bash
bash-5.1# mysql -U studentsrepo -p
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 104
Server version: 9.1.0 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| studentsrepo |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql> use studentsrepo
Database changed
mysql> show tables;
+-----+
| Tables_in_studentsrepo |
+-----+
| authenticated_users |
| hibernate_sequence |
| notifications |
| profiles |
| students |
| user_followers |
| user_following |
+-----+
7 rows in set (0.00 sec)

mysql> SELECT * FROM notifications;
Empty set (0.00 sec)
```

Problema della POST anonima

Non vengono salvate le notifiche. Nel docker di T5 viene segnalata una anonymous POST nel database di T23 relativa all'utente. Questo avviene perché la notifica viene creata dal sistema e il sistema, non essendo autenticato, non supera i controlli di Spring Security. Possiamo risolvere in due modi:

- Dando un JWT al sistema in modo che esso sia autenticato
- Passando il JWT dell'utente alla richiesta

Abbiamo scelto la seconda strada, modificando quindi il file di security config per ammettere questo tipo di richieste. Le notifiche vengono create e recuperate da T5!

```
mysql> SELECT * FROM notifications
-> SELECT * FROM notifications;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near 'SELECT * FROM notifications' at line 2
mysql> SELECT * FROM notifications;
+-----+-----+-----+-----+
| id | is_read | message | titolo | playeri |
|-----+-----+-----+-----+
| 3 | 0x00 | Congratulazioni! Hai ottenuto il nuovo achievement: Novellino! | Nuovo Achievement | |
| 4 | 0x00 | Congratulazioni! Hai ottenuto il nuovo achievement: Hai vinto la tua prima partita!! | Nuovo Achievement | |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Correzioni Bug

Miglioramenti Gestione Profilo

- Sistemata funzionalità di lettura
 - Implementato correttamente il flag Boolean `isRead`.
 - Migliorato il monitoraggio dello stato di lettura dei messaggi.
- Migliorata flessibilità Edit-Profile
 - Implementato aggiornamento indipendente di bio e immagine profilo.
 - Gli utenti possono ora modificare uno dei due campi senza influenzare l'altro.

Requisiti Non Funzionali

Internazionalizzazione

- Implementato supporto completo per le lingue:
 - Italiano
 - Spagnolo
 - Inglese
- Applicato a tutte le pagine e componenti relativi al task.

Chapter 5

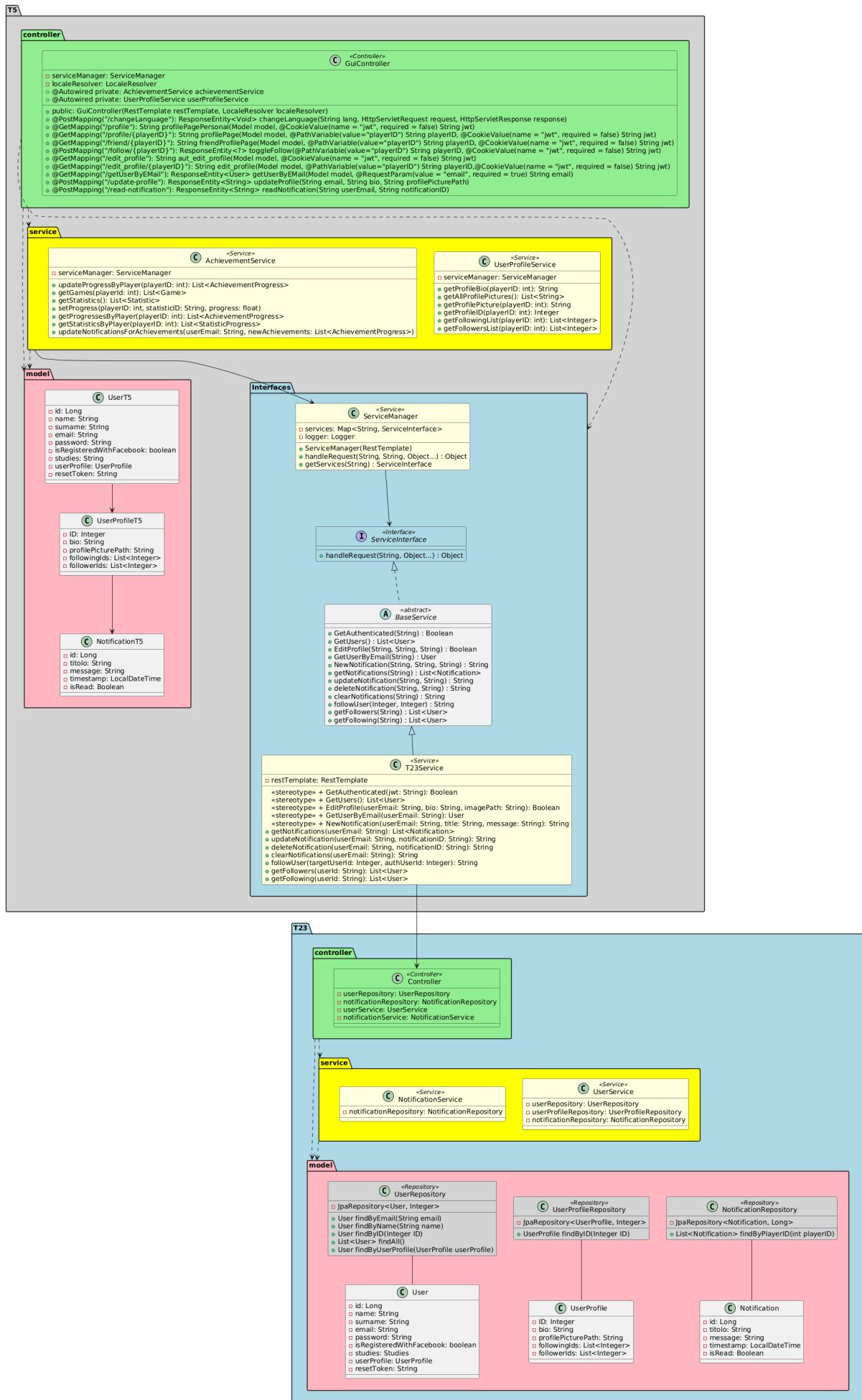
Struttura del progetto modificato

5.1 Nuovi Endpoint

/students_list					
c	description	operationID	parameters	RequestBody	response
GET	Restituisce la lista di tutti gli studenti	getAllStudents	-	-	"200": List<User>
/students_list/{ID}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce uno studente specifico	getStudent	ID (path variable, String)	-	"200": User
/user_by_email					
	description	operationID	parameters	RequestBody	response
GET	Restituisce un utente tramite email	getUserByEmail	email (query parameter, String)	-	"200": User
/profile_by_email					
	description	operationID	parameters	RequestBody	response
GET	Trova il profilo di un utente tramite email	findProfileByEmail	email (query parameter, String)	-	"200": UserProfile "400": "User with email [email] not found"
/edit_profile					
	description	operationID	parameters	RequestBody	response
POST	Modifica il profilo di un utente	editProfile	email, bio, profilePicturePath (query parameters, String)	-	"200": Boolean "400": "Profile not found"
/add-follow					
	description	operationID	parameters	RequestBody	response
POST	Aggiorna la lista di seguiti e di seguaci dei profili coinvolti	toggleFollow	-	id follower, id following (query parameters, String)	"200": "Follow status changed" "400": "User not found" "500": "Internal server error"
/get-followers					
	description	operationID	parameters	RequestBody	response

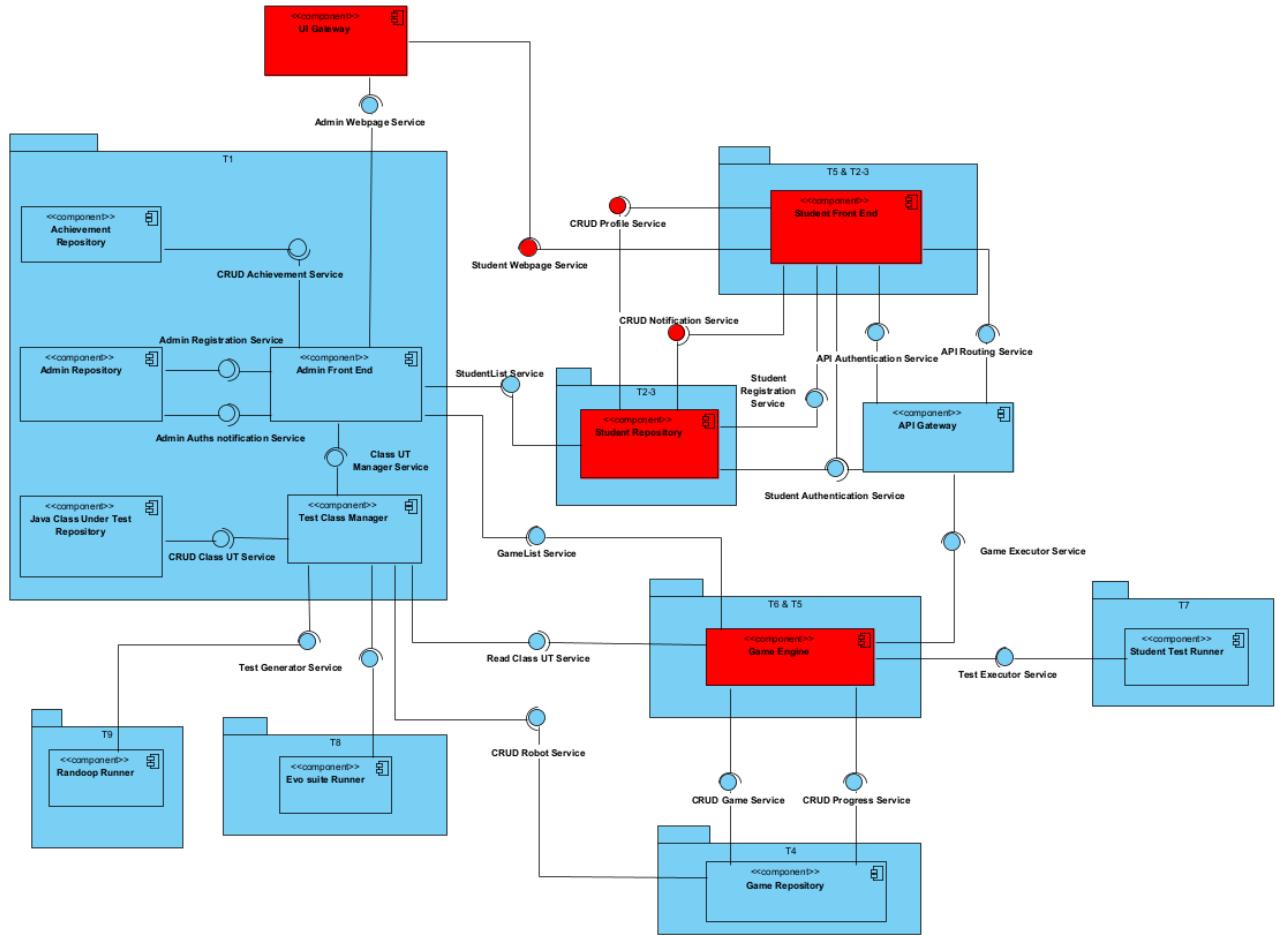
GET	Ottiene una lista di followers	getFollowers	-	id user (query parameters, String)	"200": List<User> "404": "User not found" "404": "UserProfile not found"
/get-following					
	description	operationID	parameters	RequestBody	response
GET	Ottiene una lista di following	getFollowing	-	id user (query parameters, String)	"200": List<User> "404": "User not found" "404": "UserProfile not found"
/new_notification					
	description	operationID	parameters	RequestBody	response
POST	Crea una nuova notifica per un utente	saveNotification	playerID (int), titolo (String), message (String)	-	"200": Notifica salvata
/notifications					
	description	operationID	parameters	RequestBody	response
GET	Ottiene tutte le notifiche di un utente	getNotifications ByPlayer	playerID (int)	-	"200": List<Notification>
/update_notification					
	description	operationID	parameters	RequestBody	response
POST	Segna una notifica come letta	markNotification AsRead	notificationID (Long)	-	"200": Notifica aggiornata "404": "Notifica con ID [notificationID] non trovata"
/delete_notification					
	description	operationID	parameters	RequestBody	response
DELETE	Elimina una notifica specifica	deleteNotification	notificationID (Long)	-	"200": Notifica eliminata "404": "Notifica con ID [notificationID] non trovata"
/studentsByIds					
	description	operationID	parameters	RequestBody	response
POST	Ottiene una lista di studenti dai loro ID	getStudentiTeam	-	idsStudenti (Lista di Stringhe)	"200": List<User> "400": "Lista degli ID vuota" "400": "Formato degli ID non valido" "404": "Nessun utente trovato" "500": "Errore interno del server"

5.2 Package diagram complessivo



5.3 Composite diagram complessivo

In rosso vengono riportati i componenti e le interfacce che abbiamo modificato/implementato:



Microservizio	Path	Nome File	Descrizione	Tipo
T23-G1	/src/main/java/com/example/db_setup	Controller.java	Implementazione di nuovi endpoint per la gestione delle notifiche e del profilo utente	Modifica
T23-G1	/src/main/java/com/example/db_setup	Notification.java	Definizione dell'entità per la persistenza delle notifiche nel database	Aggiunta
T23-G1	/src/main/java/com/example/db_setup	NotificationRepository.java	Repository per l'accesso e la gestione dei dati delle notifiche nel database	Aggiunta
T23-G1	/src/main/java/com/example/db_setup	SecurityConfig.java	Configurazione delle autorizzazioni per il sistema di notifiche	Modifica
T23-G1	/src/main/java/com/example/db_setup/Service	NotificationService.java	Implementazione della logica di business per la gestione delle notifiche	Aggiunta
T23-G1	/src/main/java/com/example/db_setup/Service	UserService.java	Aggiunta della logica di business per i nuovi endpoint del profilo utente	Modifica
T23-G1	/src/main/java/com/example/db_setup	User.java	Implementazione della relazione one-to-one tra User e UserProfile	Modifica
T23-G1	/src/main/java/com/example/db_setup	UserProfile.java	Definizione dell'entità per la gestione dei profili utente	Aggiunta
T23-G1	/src/main/java/com/example/db_setup	UserProfileRepository.java	Repository per l'accesso e la gestione dei dati dei profili utente	Aggiunta
T23-G1	/src/main/java/com/example/db_setup	UserRepository.java	Aggiunta delle query findByUserProfile per /get-follower e /get-following	Modifica

Table 5.2: Modifiche e aggiunte effettuate al microservizio T23-G1

Microservizio	Path	Nome File	Descrizione	Tipo
T5-G2	/t5/src/main/java/com/g2/Game	GameController.java	Correzione del sistema di salvataggio delle partite	Modifica
T5-G2	/t5/src/main/java/com/g2/Interfaces	T23Service.java	Aggiunte action relative agli end-point	Modifica
T5-G2	/t5/src/main/java/com/g2/Interfaces	T4Service.java	Issue Salvataggio partita risolto: update delle statistiche	Modifica
T5-G2	/t5/src/main/java/com/g2/Model	Notification.java, UserProfile.java	Aggiunta model Notifica e UserProfile per funzionamento Thymeleaf	Aggiunta
T5-G2	/t5/src/main/java/com/g2/Service	AchievementService.java	Metodo per creare la notifica una volta ottenuto l'achievement	Modifica
T5-G2	/t5/src/main/java/com/g2/Service	UserProfileService.java	Servizio che tramite l'interfaccia con T23 fa il retrieve delle informazioni UserProfile	Aggiunta
T5-G2	/t5/src/main/java/com/g2/t5	GuiController.java	Endpoint chiamati dalla GUI per PageBuilder o retrieve di informazioni Thymeleaf	Modifica
T5-G2	/t5/src/main/resources/lang	messages_*.properties	Internazionalizzazione testo GUI	Modifica
T5-G2	/t5/src/main/resources/templates	profile.html, Edit_profile.html, friend_profile.html	Modifica sostanziale e aggiunta di template html conformi alla lingua e Thymeleaf	Aggiunta
T5-G2	t5/src/main/resources/static/t5/css	profile.css, profile-edit.css	Fogli di stile relativi ai template html	Aggiunta
T5-G2	t5/src/main/resources/static/t5/js	profile.js, profile-edit.js, friend-profile.js	Dinamica dei template HTML e chiamata agli end-point del GUI Controller	Aggiunta
T5-G2	t5/src/main/resources/static/t5/images/profileImages	default.png, men-*.png, women-*.png	Immagini di profilo tra cui scegliere	Aggiunta
UI-Gateway	ui_gateway	default.conf	Aggiunte rotte nella configurazione NGINX	Modifica

Table 5.3: Modifiche e aggiunte effettuate al microservizio T5-G2 e UI-Gateway

Chapter 6

Strategia di test

Questa documentazione descrive i test eseguiti sulle API del sistema. I test coprono le funzionalità principali introdotte come la gestione dei follower, delle notifiche, l'aggiornamento dei profili utente e le interazioni sociali.[3]

Ogni test case è strutturato per verificare sia i casi di successo che gli scenari di errore, garantendo che il sistema risponda appropriatamente in tutte le situazioni. La documentazione include:

- Descrizione dettagliata di ogni test case
- Metodo HTTP utilizzato
- Parametri richiesti
- Risposta attesa
- Risultato effettivo
- Screenshots delle risposte
- Note aggiuntive rilevanti

6.1 GET */get-followers*

Test suite per verificare il recupero dei follower di un utente.

6.1.1 Test Case 1.1: Utente con follower

Descrizione: Verifica che il sistema restituisca correttamente la lista dei follower per un utente esistente con follower.

- **Metodo:** GET
- **Risposta attesa:** 200 OK
- **Risultato:** Success

Note: La risposta include informazioni dettagliate sul profilo utente inclusi bio, immagini e stato di login.

6.1.2 Test Case 1.2: Utente non esistente

Descrizione: Verifica la gestione di una richiesta per un utente non esistente.

- **Metodo:** GET
- **Risposta attesa:** 404 Not Found
- **Risultato:** Success

Note: Il sistema restituisce un messaggio di errore appropriato "user not found".

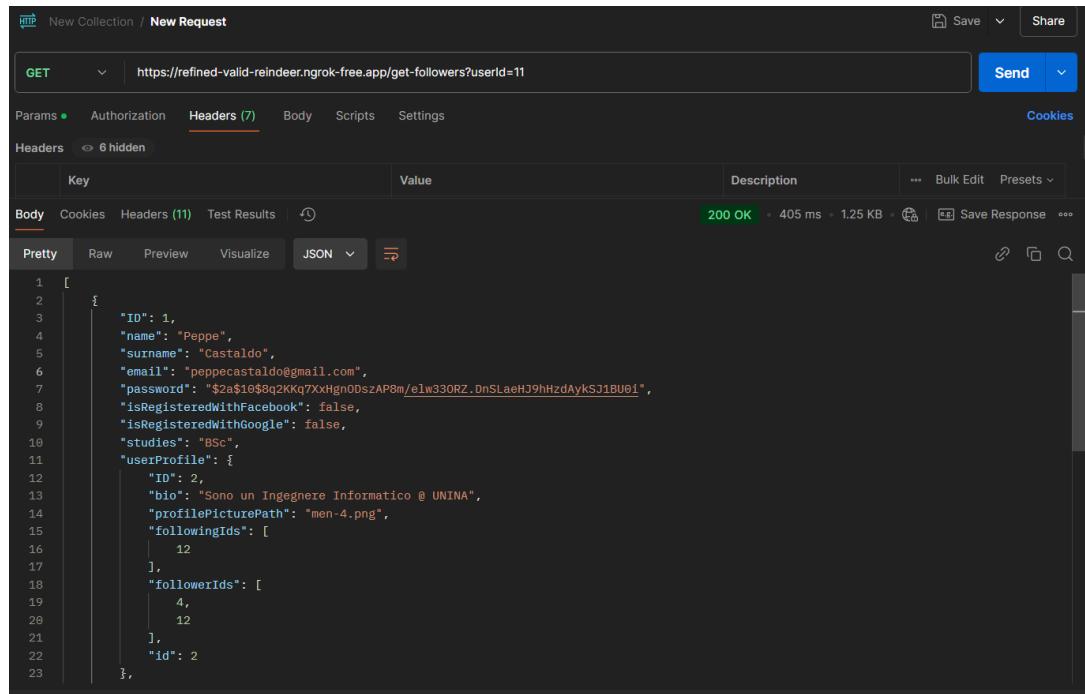


Figure 6.1: Screenshot Test Case 1.1 - Postman

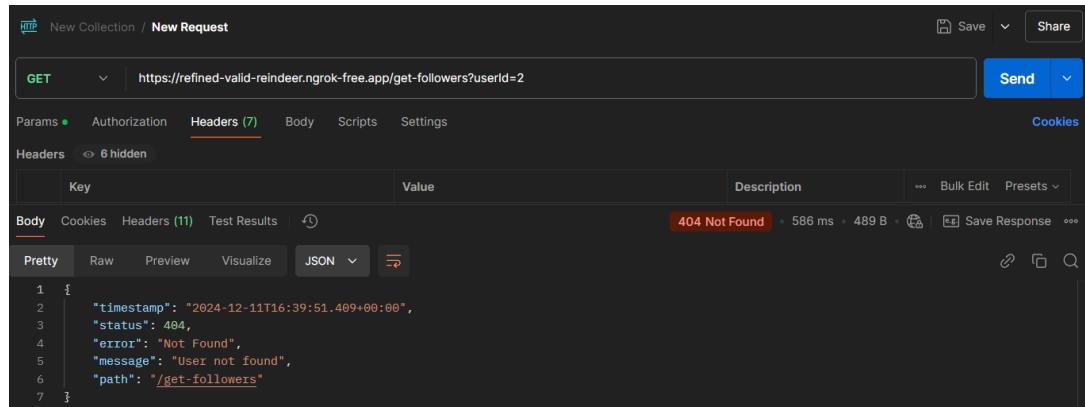


Figure 6.2: Screenshot Test Case 1.2 - Postman

6.1.3 Test Case 1.3: Utente esistente senza follower

Descrizione: Verifica il comportamento quando si richiede la lista follower di un utente senza seguaci.

- **Metodo:** GET
- **Risposta attesa:** 200 OK
- **Risultato:** Success

Note: Il sistema restituisce una lista vuota.

6.2 GET /*notifications*

Test suite per il recupero delle notifiche utente.

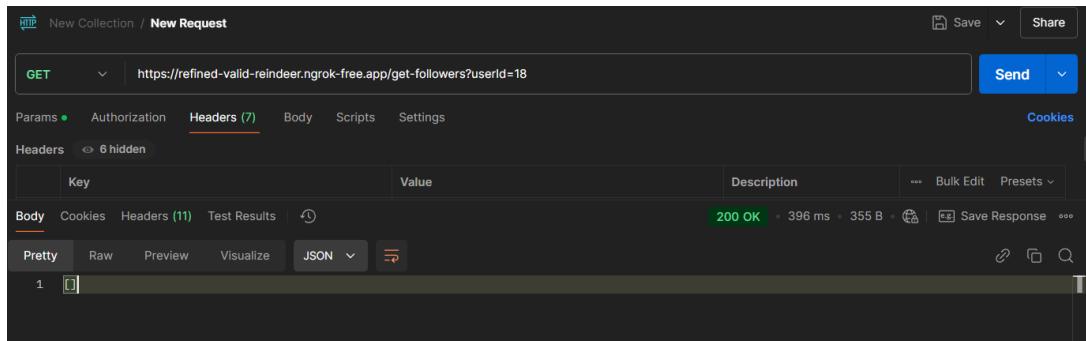


Figure 6.3: Screenshot Test Case 1.3 - Postman

6.2.1 Test Case 2.1: Utente senza notifiche

Descrizione: Verifica il comportamento quando un utente non ha notifiche.

- **Metodo:** GET
- **Risposta attesa:** 200 OK
- **Risultato:** Success

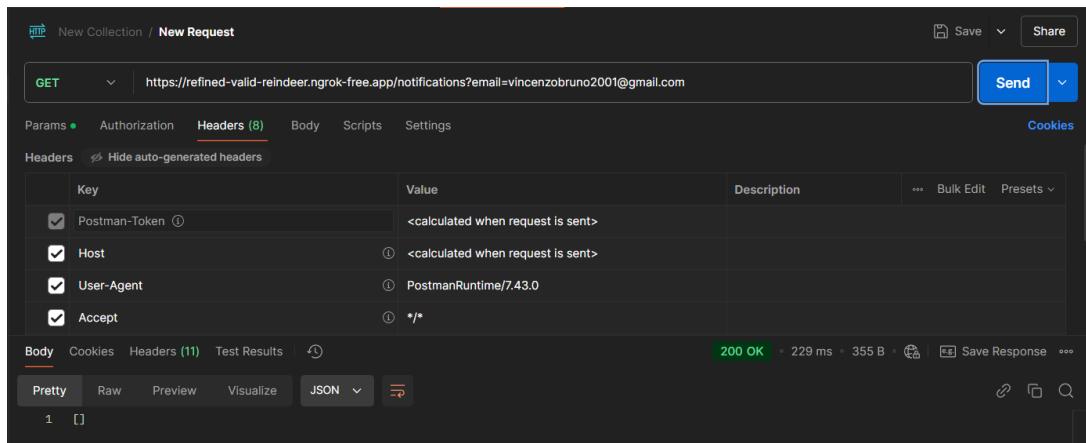


Figure 6.4: Screenshot Test Case 2.1 - Postman

Note: Il sistema restituisce una lista vuota di notifiche.

6.2.2 Test Case 2.2: Utente con notifiche

Descrizione: Verifica il recupero delle notifiche per un utente con notifiche esistenti.

- **Metodo:** GET
- **Risposta attesa:** 200 OK
- **Risultato:** Success

Note: Le notifiche includono informazioni su nuovi follower e timestamp.

6.3 POST /update-profile

Test suite per l'aggiornamento del profilo utente.

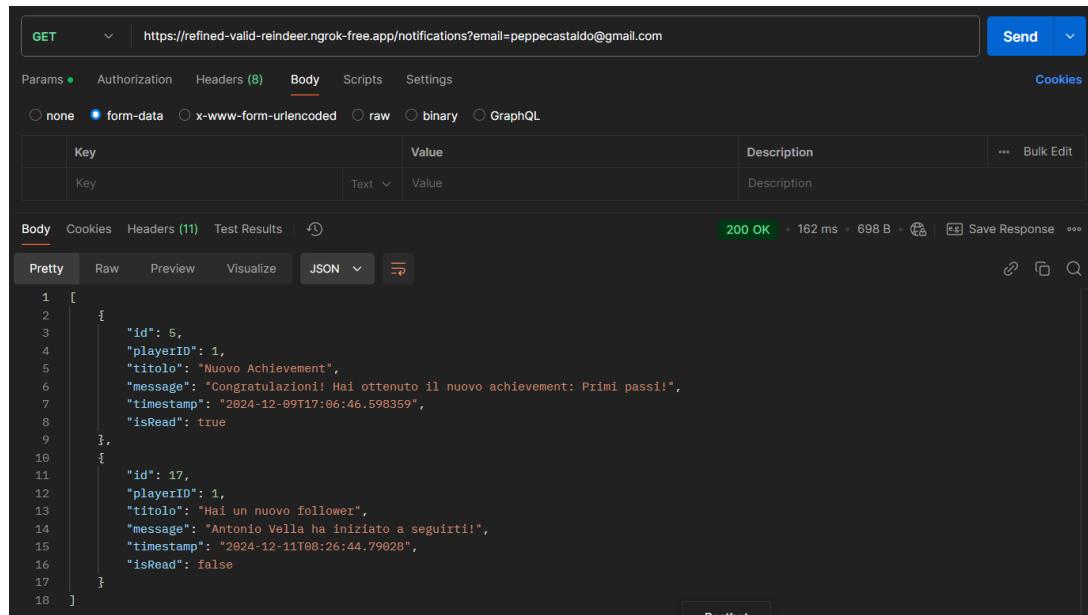


Figure 6.5: Screenshot Test Case 2.2 - Postman

6.3.1 Test Case 3.1: Aggiornamento profilo successo

Descrizione: Verifica l'aggiornamento corretto del profilo utente.

- **Metodo:** POST
- **Parametri:** email, bio, profilePicturePath
- **Risposta attesa:** 200 OK
- **Risultato:** Success

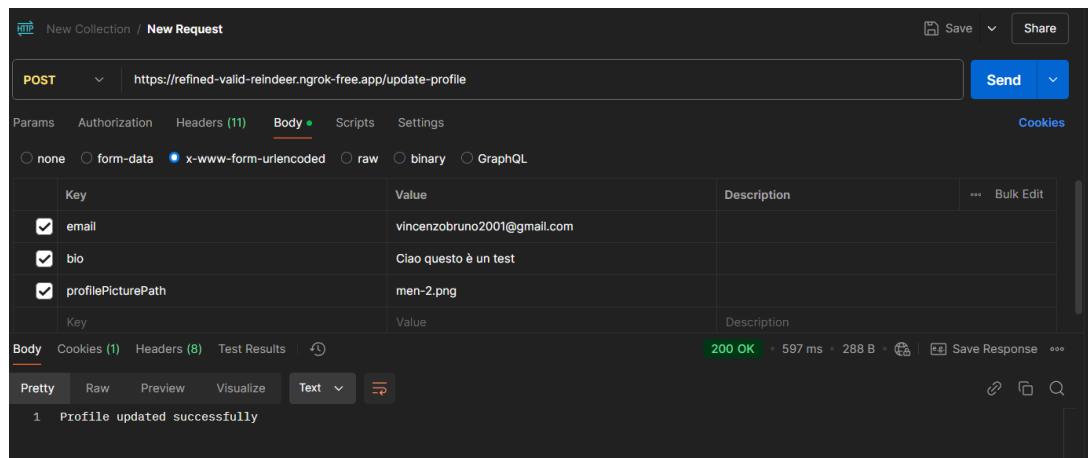


Figure 6.6: Screenshot Test Case 3.1 - Postman

Note: Il profilo viene aggiornato correttamente con conferma "Profile updated successfully".

6.3.2 Test Case 3.2: Utente non esistente

Descrizione: Verifica la gestione dell'aggiornamento per un utente non esistente.

- **Metodo:** POST
- **Parametri:** email, bio, profilePicturePath
- **Risposta attesa:** 500 Internal Server Error
- **Risultato:** Success

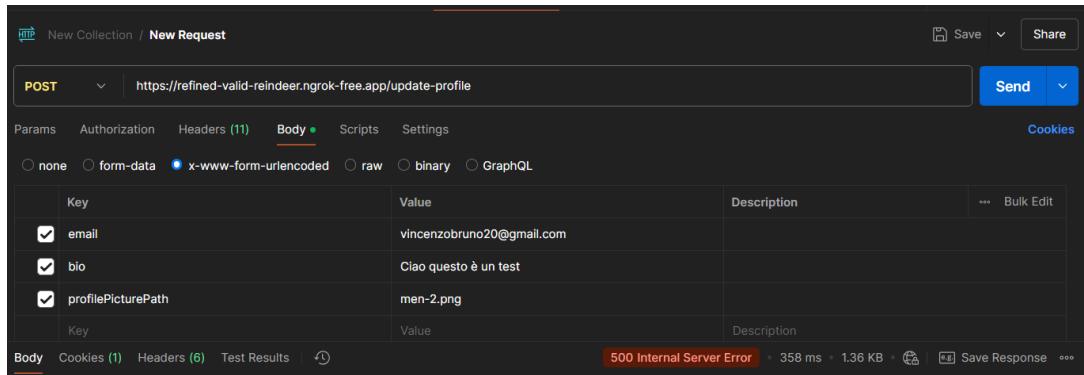


Figure 6.7: Screenshot Test Case 3.2 - Postman

Note: Il sistema gestisce appropriamente il caso di utente non esistente.

6.4 POST /new-notification

Test suite per la creazione di nuove notifiche.

6.4.1 Test Case 4.1: Creazione notifica successo

Descrizione: Verifica la creazione di una nuova notifica per un utente esistente.

- **Metodo:** POST
- **Parametri:** email, title, message
- **Risposta attesa:** 200 OK
- **Risultato:** Success

Note: La notifica viene creata correttamente con conferma.

6.4.2 Test Case 4.2: Parametro mancante

Descrizione: Verifica la gestione di una richiesta con parametri mancanti.

- **Metodo:** POST
- **Parametri:** email, title (message mancante)
- **Risposta attesa:** 400 Bad Request
- **Risultato:** Success

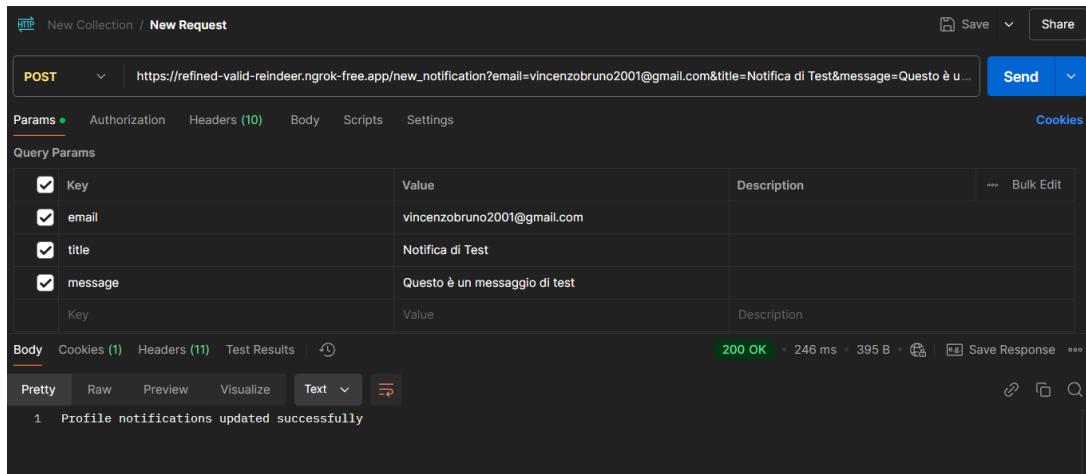


Figure 6.8: Screenshot Test Case 4.1 - Postman

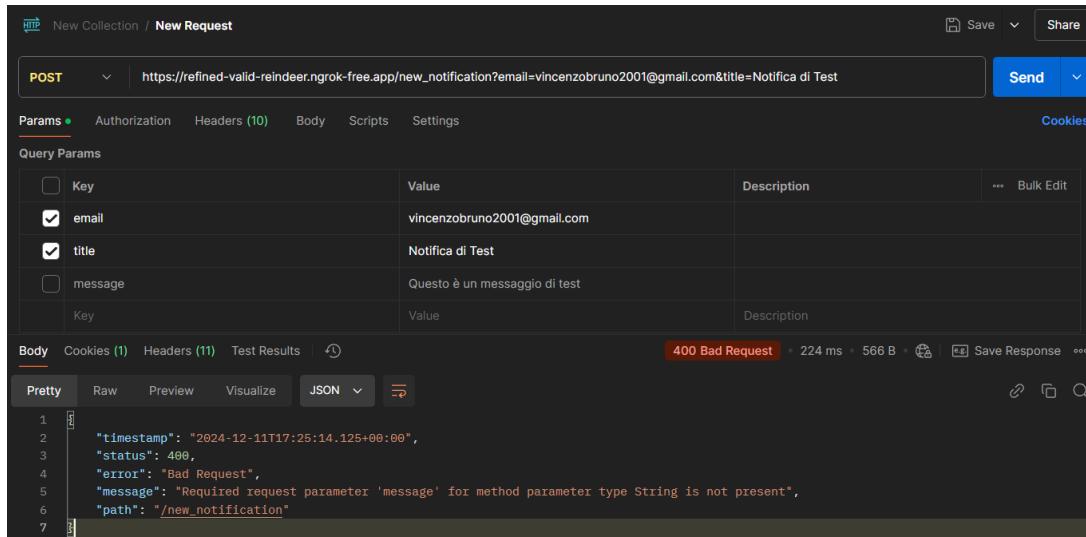


Figure 6.9: Screenshot Test Case 4.2 - Postman

Note: Il sistema identifica correttamente il parametro mancante.

6.4.3 Test Case 4.3: Utente inesistente

Descrizione: Verifica la creazione di una notifica per un utente non esistente.

- **Metodo:** POST
- **Parametri:** email non esistente, title, message
- **Risposta attesa:** 500 Internal Server Error
- **Risultato:** Success

6.5 POST /*update-notification*

Test suite per la lettura delle notifiche.

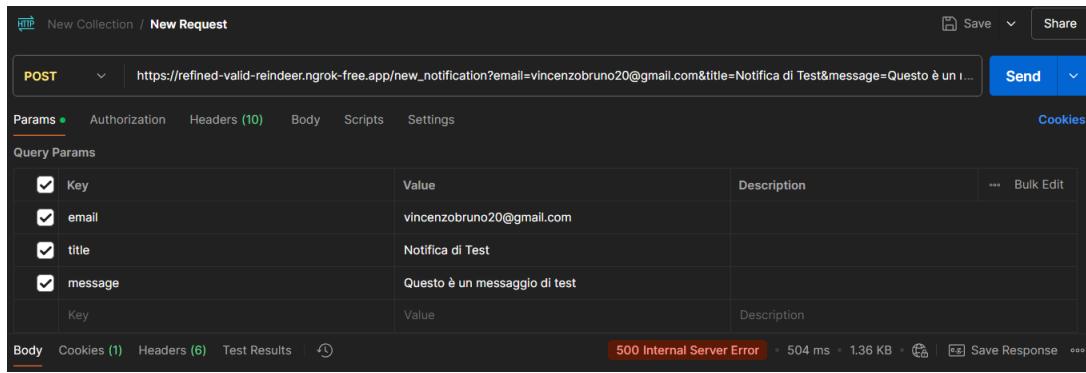


Figure 6.10: Screenshot Test Case 4.3 - Postman

6.5.1 Test Case 5.1: Lettura corretta

Descrizione: Verifica la lettura di una notifica esistente.

- **Metodo:** POST
- **Parametri:** email, id notifica
- **Risposta attesa:** 200 OK
- **Risultato:** Success

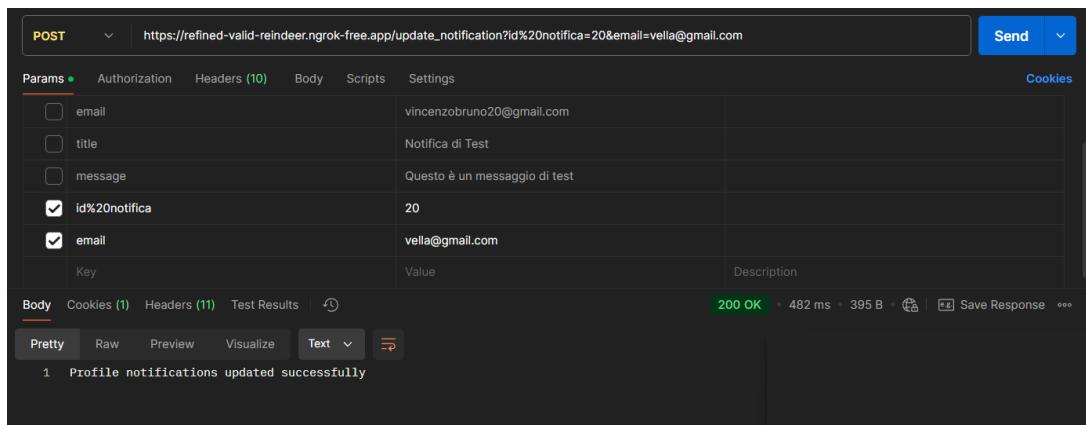


Figure 6.11: Screenshot Test Case 5.1 - Postman

Note: La notifica viene letta con successo.

6.5.2 Test Case 5.2: Notifica non esistente

Descrizione: Verifica la lettura di una notifica non esistente.

- **Metodo:** POST
- **Parametri:** email, id notifica non esistente
- **Risposta attesa:** 404 Not Found
- **Risultato:** Success

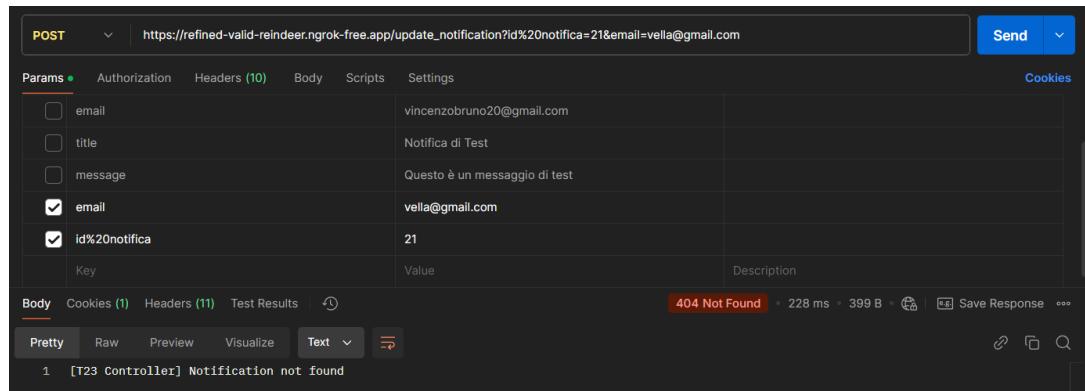


Figure 6.12: Screenshot Test Case 5.2 - Postman

Note: Il sistema gestisce appropriamente il caso di notifica non esistente.

6.6 DELETE /*delete-notification*

Test suite per la cancellazione delle notifiche.

6.6.1 Test Case 6.1: Cancellazione corretta

Descrizione: Verifica la cancellazione di una notifica esistente.

- **Metodo:** DELETE
- **Parametri:** email, id
- **Risposta attesa:** 200 OK
- **Risultato:** Success

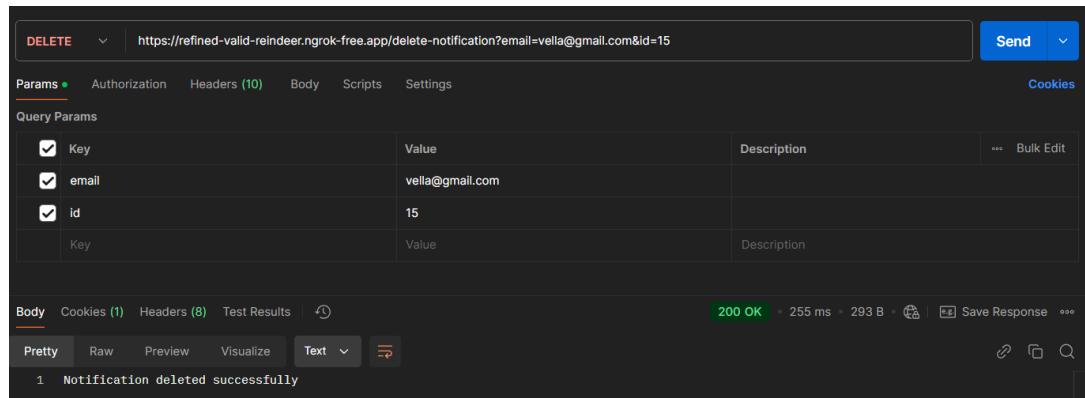


Figure 6.13: Screenshot Test Case 6.1 - Postman

Note: La notifica viene eliminata con successo.

6.6.2 Test Case 6.2: Email non esistente

Descrizione: Verifica la cancellazione per un'email non esistente.

- **Metodo:** DELETE

- **Parametri:** email non esistente, id
- **Risposta attesa:** 500 Internal Server Error
- **Risultato:** Success

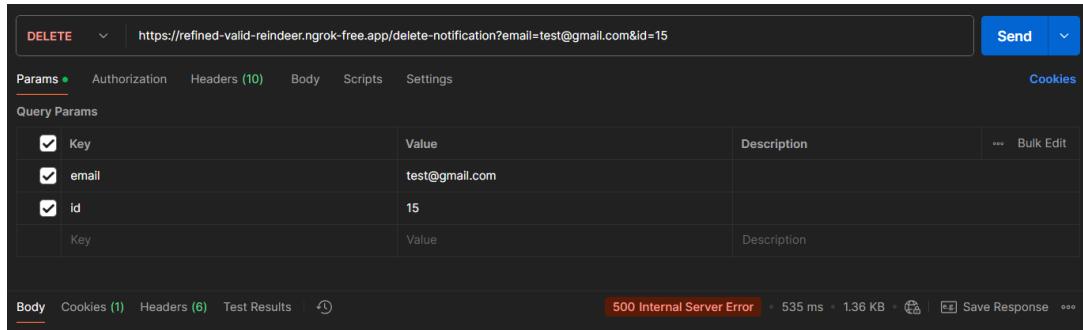


Figure 6.14: Screenshot Test Case 6.2 - Postman

6.6.3 Test Case 6.3: ID notifica non valido

Descrizione: Verifica la gestione di un ID notifica non valido.

- **Metodo:** DELETE
- **Parametri:** email, id non valido
- **Risposta attesa:** 500 Internal Server Error
- **Risultato:** Success

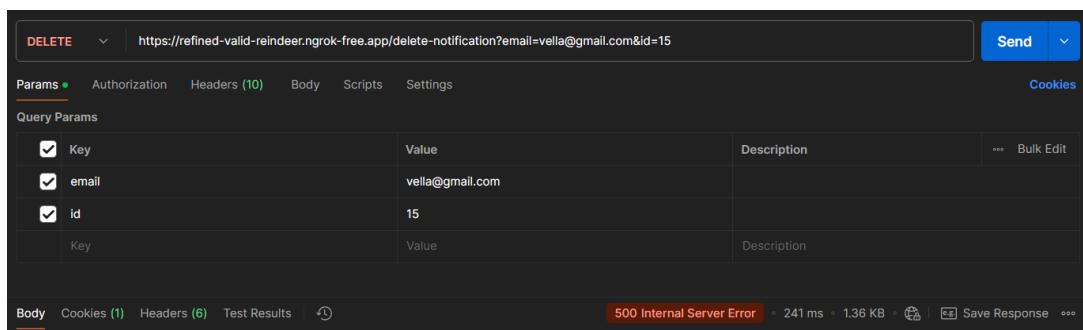


Figure 6.15: Screenshot Test Case 6.3 - Postman

6.6.4 Test Case 6.4: ID notifica non esistente

Descrizione: Verifica la cancellazione di una notifica non esistente.

- **Metodo:** DELETE
- **Parametri:** email, id non esistente
- **Risposta attesa:** 404 Not Found
- **Risultato:** Success

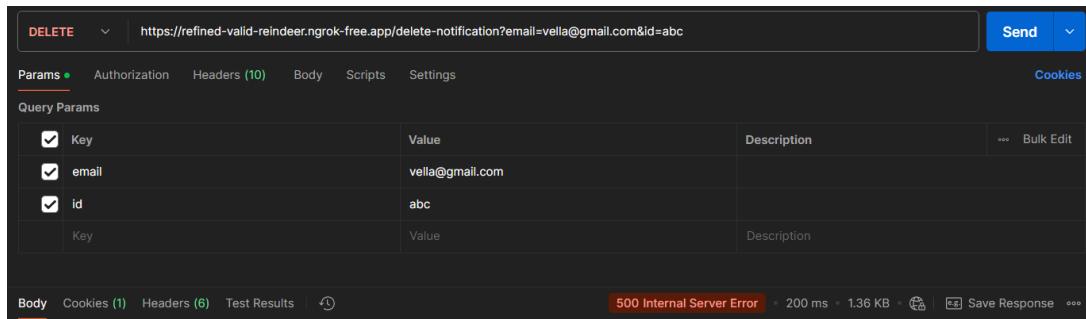


Figure 6.16: Screenshot Test Case 6.4 - Postman

6.7 POST /follow/{playerId}

Test suite per la funzionalità di following.

6.7.1 Test Case 7.1: Follow con successo

Descrizione: Verifica la funzionalità di follow per un utente esistente.

- **Metodo:** POST
- **Parametri:** playerId
- **Risposta attesa:** 200 OK
- **Risultato:** Success

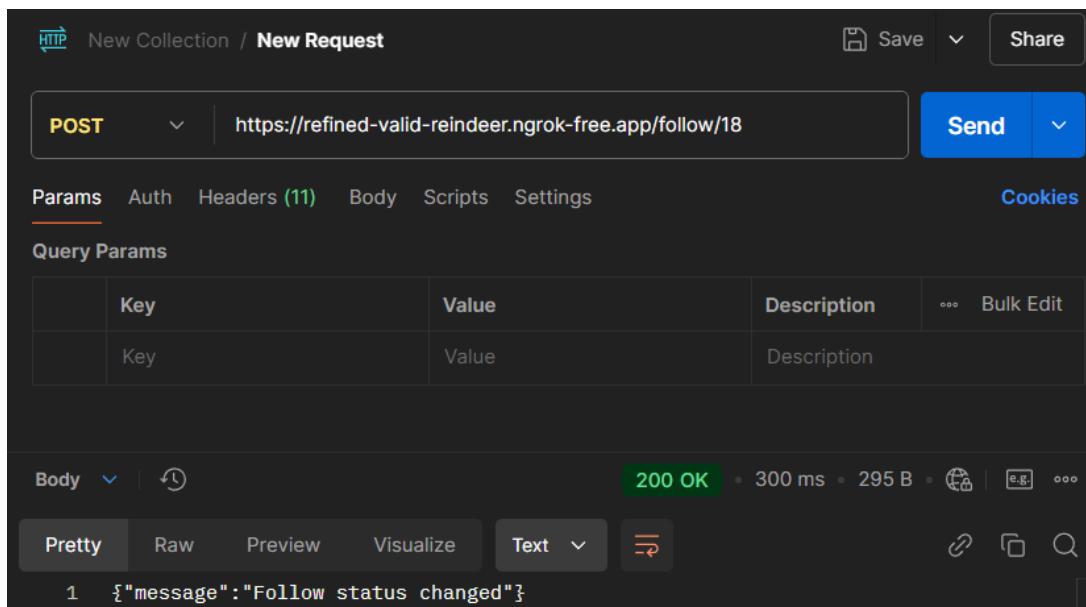


Figure 6.17: Screenshot Test Case 7.1 - Postman

Note: Il sistema conferma il cambio di stato del follow.

Chapter 7

Potenziali sviluppi futuri

7.1 Ecosistema Teams

Impatto: Integrazione con task R5 - Classi di Studenti

Base tecnica: Implementazione del bottone Teams nell'interfaccia utente

Prospettive: La predisposizione tecnica già implementata apre la strada a un ecosistema Teams più ricco e interattivo. Gli utenti potranno accedere a una vista dettagliata del proprio team, esplorando non solo la composizione del gruppo ma anche le dinamiche collaborative in atto. Questo permetterà di visualizzare in tempo reale le statistiche di squadra e monitorare i progressi collettivi, creando un senso di appartenenza e motivazione condivisa all'interno del team.

7.2 Sistema di Assignment e Riconoscimenti

Impatto: Integrazione con task R5 - Classi di Studenti

Base tecnica: Struttura modulare della pagina profilo con sezioni distinte e tab trofei

Prospettive: Una naturale evoluzione potrebbe essere l'introduzione di una nuova sezione o tab dedicata agli assignment, posizionata accanto all'attuale area dei trofei. Questo spazio potrebbe ospitare badge e premi specifici legati alle performance di team, creando un sistema di riconoscimenti più articolato.

7.3 Sistema Token

Impatto: Integrazione con task R3 - Missioni

Base tecnica: Sistema di visualizzazione token implementato

Prospettive: La visualizzazione dei Token che abbiamo predisposto potrebbe evolversi in un vero e proprio sistema di economia virtuale. Una delle possibili applicazioni potrebbe essere l'acquisto di elementi di personalizzazione, come foto profilo esclusive che potrebbero essere bloccate fino al raggiungimento di determinati obiettivi.

7.4 NotificationService

Impatto: Integrazione con task R5 - Classi di Studenti

Base tecnica: NotificationService operativo

Prospettive: Il NotificationService sviluppato rappresenta un'infrastruttura pronta per essere integrata con il microservizio T1. Questa integrazione potrebbe abilitare l'invio automatico di notifiche per nuovi assignment o aggiornamenti relativi ai Team, migliorando il coinvolgimento degli utenti.

7.4.1 "How To": Integrare Notification Service

Questo piccolo tutorial vuole rendere meno tediosa l'integrazione delle notifiche di sistema in un future nuove modifiche del progetto. Per capire al meglio la struttura delle notifiche, i dati che le compongono, dove vengono salvate e le scelte progettuali legate a quest'ultime, consigliamo al lettore di ritornare ai capitoli precedenti. Di seguito invece saranno riportati brevemente come poter utilizzare le API sviluppati in questa iterazione di progetto in maniera facile e veloce.

7.4.2 API Disponibili

Riportiamo le API che abbiamo predisposto nel Task T23, ossia il container dove vengono create e salvate permanentemente le notifiche di sistema. Ecco un elenco con relative descrizioni dei parametri da passare durante la richiesta HTTP:

- /new_notification: creazione di una nuova notifica
- /notifications: restituisce tutte le notifiche di un utente
- /update_notifications: marca la notifica come letta
- /delete_notification: elimina la notifica
- clear_notifications: elimina tutte le notifiche di un utente

Metodo HTTP	Rotta	Content-Type	Parametri Richiesti
POST	/new_notification	application/x-www-form-urlencoded	email, title, message
GET	/notifications	Nessuno	email
POST	/update_notification	application/x-www-form-urlencoded	email, id, notifica
DELETE	/delete_notification	application/x-www-form-urlencoded	email, id, notifica
DELETE	/clear_notifications	Nessuno	email

Aggiungere una nuova notifica

Di seguito verrà mostrato un esempio di creazione di notifica. Ci soffermeremo solo su questa API, poichè la riteniamo più utile per sviluppi futuri. L'esempio riprende la creazione di una notifica al conseguimento di un nuovo Achievement nel servizio **T5**.

Esempio di richiesta:

```
POST /new_notification
Content-Type: application/x-www-form-urlencoded
email=utente@example.com&title=TitoloAchievement&message
=Hai+sbloccato+un+nuovo+Achievement
```

Nel servizio T5 viene fatta una chiamata **REST** dopo aver eseguito la propria logica di backend. In particolare, T5 è stato implementato utilizzando il pattern Model View Controller, con l'utilizzo della tecnologia **SpringBoot**, dove è buona norma fare chiamate a servizi esterni tramite l'utilizzo di "**Services**". Riportiamo come è stata implementata la richiesta al servizio **T23** da parte di **AchievementService**

Come riportato nel codice, si può vedere che per la creazione di una notifica vengono creati e passati i parametri **IdUser, Titolo, Messaggio**

Successivamente viene richiamato il metodo del sotto servizio **T23Service**, sempre presente in T5. Ovviamente l'implementazione della chiamata cambia dall'ambiente in cui viene implementata. Adesso riportiamo direttamente il metodo che esegue la chiamata REST.

```

public void updateNotificationsForAchievements(String userEmail, List<AchievementProgress> newAchievements) {
    for (AchievementProgress achievement : newAchievements) {
        String titolo = "Nuovo Achievement";
        String message = "Congratulazioni! Hai ottenuto il nuovo achievement: " + achievement.Name + "!";
        serviceManager.handleRequest("T23", "NewNotification", userEmail, titolo, message);
    }
}

```

Figure 7.1: Metodo chiamato da AchievementService

```

// Metodo per la creazione di una notifica
private String NewNotification(String userEmail, String title, String message) {
    final String endpoint = "/new_notification";
    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add("email", userEmail);
    map.add("title", title);
    map.add("message", message);
    return callRestPost(endpoint, map, null, String.class);
}

```

Figure 7.2: Chiamata al servizio T23 per creazione di una notifica

```

// Metodo per chiamate POST con content type a application/x-www-form-urlencoded
protected <R> R callRestPost(String endpoint, MultiValueMap<String, String> formData,
    Map<String, String> queryParams, Map<String, String> customHeaders,
    Class<R> responseType) {
    if (formData == null) {
        throw new IllegalArgumentException("formData non può essere nullo");
    }
    return executeRestCall("callRestPost", () -> {
        String url = buildUri(endpoint, queryParams);
        // Imposta gli header, incluso il Content-type di default
        HttpHeaders headers = new HttpHeaders();
        // Aggiunge gli header personalizzati
        if (customHeaders != null) {
            customHeaders.forEach(headers::add);
        }
        // Imposta il content type a application/x-www-form-urlencoded se non
        // specificato
        if (!headers.containsKey(HttpHeaders.CONTENT_TYPE)) {
            headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        }
        HttpEntity<MultiValueMap<String, String>> requestEntity = new HttpEntity<>(formData, headers);
        ResponseEntity<R> response = restTemplate.postForEntity(url, requestEntity, responseType);
        return response.getBody();
    });
}

```

Figure 7.3: Chiamata Rest Effettiva

Così facendo se tutto va a buon fine è stata creata una nuova notifica, che verrà opportunamente visualizzata nella pagina profilo utente. I risultati più comuni sono i seguenti:

- 200 OK: La notifica è stata salvata con successo.
- 400 BAD REQUEST: L'email fornita non corrisponde a un profilo valido.

7.5 Leaderboard

Impatto: Integrazione con task R2 - Classifiche

Base tecnica: Pagina profilo amico implementata e sistema di following operativo

Prospettive: La futura integrazione della Leaderboard potrebbe includere funzionalità social più avanzate, permettendo agli utenti di esplorare i profili dei giocatori in classifica e seguirli. Questo potrebbe creare una dimensione più comunitaria all'interno della piattaforma.

7.6 Sistema Statistiche

Impatto: Integrazione con task R4 - Punteggi

Base tecnica: Sezione statistiche nel profilo utente

Prospettive: La sezione statistiche presenta numerose opportunità di miglioramento, come l'integrazione di nuove metriche e la creazione di visualizzazioni grafiche più sofisticate per i dati storici. Queste implementazioni potrebbero offrire una comprensione più intuitiva dell'andamento del giocatore nel tempo.

Bibliography

- [1] Leslie Lamport. *LATEX: a document preparation system: user's guide and reference manual.* Addison-wesley, 1994.
- [2] Notion Labs. Notion. <https://www.notion.so/>. Accesso: 2024-12-16.
- [3] Postman Inc. Postman. <https://www.postman.com/>. Accesso: 2024-12-16.
- [4] Testing-Game-SAD-2023. A13 repository. <https://github.com/Testing-Game-SAD-2023/A13>, 2023. Accesso: 2024-12-16.