UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

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

SOFTWARE TESTING DESIGN

DOCUMENTAZIONE TASK R1 – GESTIONE DEL PROFILO STUDENTE

PROF. FASOLINO - A.A 2024-2025

STUDENTI:

D'AVINO MATTEO	**M63001724**
FIENGA LUIGI	**M63001733**
ROSCIGNO ANDREA	**M63001778**

Sommario

Abstract

Il seguente documento serve da riferimento per mostrare gli sviluppi del progetto ENACTEST, assegnato al gruppo 8 durante il corso di Software Architecture Design, relativi al task R1. Verranno mostrati tutti i passi del lavoro, dall'analisi e specifica dei requisiti alla compilazione del codice, passando dalla fase di progettazione fino ad arrivare all'implementazione e il testing del sistema.

1 Introduzione

1.1 ENACTEST

Il progetto **European iNnovative AllianCe for TESTing (ENACTEST)** ha come obiettivo far riconoscere l'importanza cruciale del testing nello sviluppo e nell'implementazione di applicazioni web o più in generale nel ciclo di sviluppo di un qualsiasi Software. La disciplina del testing continua a essere frequentemente ignorata o sottovalutata; per rendere più travolgente l'apprendimento della medesima, il progetto sfrutta la strategia della **gamification** che, come suggerisce il termine, consiste nell'utilizzare elementi provenienti dai giochi, utilizzandoli in contesti non ludici. L'obiettivo è coprire in maniera esaustiva tutti i possibili scenari, offrendo ai partecipanti non solo un'esperienza pratica per affinare le proprie competenze, ma anche un approccio più coinvolgente e stimolante all'apprendimento della disciplina. Il seguente progetto è descritto nel seguente paper scientifico: *Anna Rita Fasolino, Caterina Maria Accetto, Porfirio Tramontana*: "Testing Robot Challenge: A Serious Game for Testing Learning". Gamify@ISSTA 2024: 26-29, https://dl.acm.org/doi/10.1145/3678869.3685686.

1.2 Architettura

Il software è strutturato su un'architettura a microservizi. Esso è un approccio di progettazione software in cui un'applicazione è suddivisa in una serie di piccoli servizi indipendenti, ciascuno dedicato a una funzionalità specifica. Questi servizi comunicano tra loro attraverso API, usando protocolli come ad esempio HTTP. Ogni microservizio è autonomo e può essere sviluppato, distribuito e scalato indipendentemente dagli altri. Ogni servizio si occupa di un dominio o funzionalità specifica. Questo approccio migliora la scalabilità, consentendo di gestire e distribuire i carichi di lavoro in modo mirato, e facilita lo sviluppo, poiché permette di lavorare su servizi diversi senza interferenze. Inoltre, offre maggiore resilienza: un errore in un microservizio non compromette l'intero sistema. Infine, promuove l'agilità, rendendo più semplice l'adozione di tecnologie diverse e il rilascio incrementale delle funzionalità.

1.3 Pattern architetturale

Il pattern architetturale utilizzato per ogni microservizio è **MVC** (Model-View-Controller), che consente di dividere la logica di business da quella di presentazione. I componenti principali sono:

1. **Model**:
    - Gestisce la logica dei dati e le regole di business dell'applicazione.
    - Comunica con il database o altre fonti di dati.
    - Aggiorna la View quando i dati cambiano.
2. **View**:
    - È la rappresentazione visuale dei dati (interfaccia utente).
    - Mostra le informazioni provenienti dal Model.
    - Non contiene logica di business, ma si limita a ricevere dati e visualizzarli.
3. **Controller**:
    - Gestisce l'interazione dell'utente e funge da intermediario tra Model e View.
    - Riceve input dall'utente, li elabora e richiama metodi del Model o aggiorna la View di conseguenza.

Considerando tale pattern, l'intera comunicazione tra i vari microservizi è affidata ai controller di ognuno di essi.

1.4 Task assegnato

Migliorare il profilo Giocatore, prevedendo una migliore strutturazione della pagina del profilo in sezioni separate per contenere prioritariamente:

a) le info del profilo (potendo farne Editing);
b) Le statistiche del giocatore;
c) Achievement ottenuti, con Badge per segnalare il raggiungimento di una soglia di una statistica);
d) eventualmente Premi (Token) guadagnati in base a Missioni svolte;
e) elenco giocatori seguiti

1.5 Stato iniziale del progetto

Il punto di partenza del progetto è stato la versione A13 del 12/11/2024. Da questo momento in poi, ci concentreremo esclusivamente sugli aspetti relativi al corretto funzionamento della gestione del Profilo. Pertanto, quando analizzeremo lo stato attuale dei microservizi, alcune funzionalità potrebbero non essere menzionate, poiché la nostra valutazione sarà limitata a quelle strettamente rilevanti per il funzionamento del Profilo. In questa versione, l'interfaccia del profilo si presenta in questo modo:

Figura 1.1 – Interfaccia profilo in A13 del 12/11/2024

Come si può notare, l'interfaccia consente solamente la visualizzazione dei dati associati all'utente, quali informazioni personali, statistiche e achievement (trofei), in quanto i due tasti in alto a destra non sono implementati.

Il diagramma dei casi d'uso per la gestione del profilo in questa versione è il seguente:

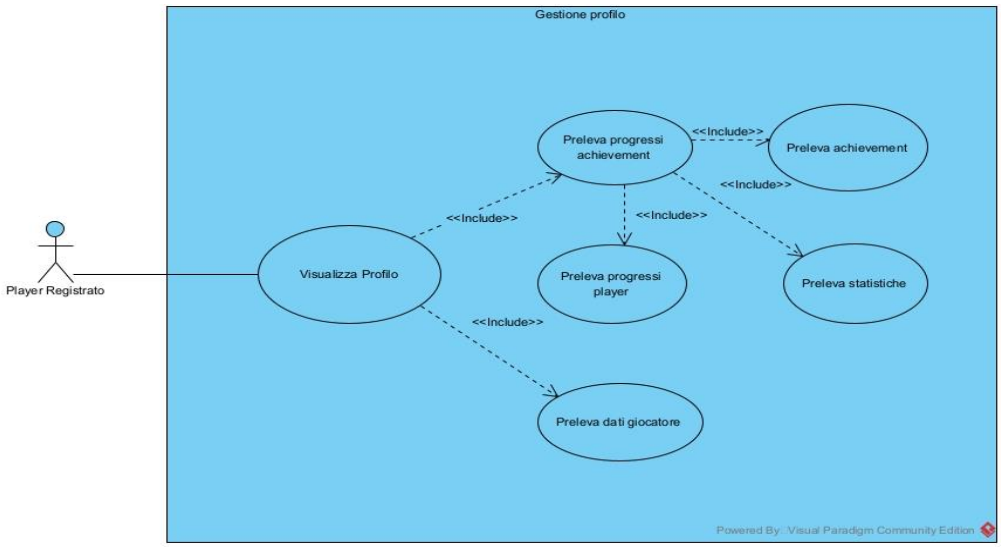


Figura 1.2 – Diagramma dei casi d'uso, su nostra valutazione, per la gestione profilo in A13 del 12/11/2024

Nel retrieve di queste informazioni, è il microservizio T5 a farsi carico della comunicazione con gli altri microservizi. Il seguente diagramma dei package illustra in maniera opportuna i componenti in gioco nel popolamento del profilo giocatore.

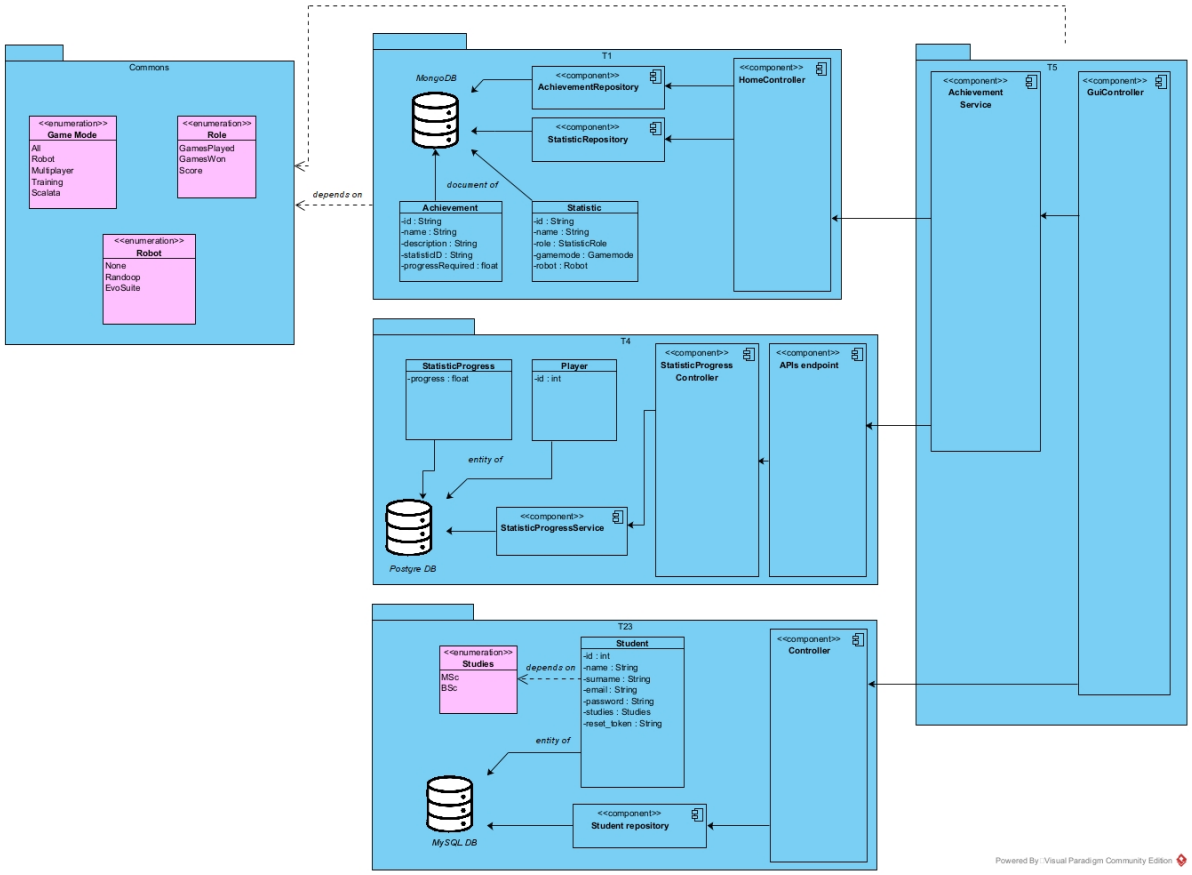


Figura 1.3 – Package diagram per il prelievo delle informazioni del player in A13 del 12/11/2024

Si riporta di seguito una breve descrizione dello stato dei vari microservizi coinvolti nella gestione del profilo nella versione di A13 dalla quale siamo partiti.

1.5.1 Stato del microservizio T1

Il microservizio T1 è interamente dedicato all'admin. In particolare, il servizio permette agli amministratori di registrarsi ed effettuare la procedura di login.

Inoltre, il medesimo permette agli amministratori, interagendo con un opportuno cruscotto, di:

- caricare, modificare, ordinare, filtrare, ricercare, scaricare ed eliminare classi di programmazione da testare all'interno del catalogo;
- visualizzare una classifica rudimentale dei players;
- creare e visualizzare le statistiche;
- creare e visualizzare gli achievement, ai quali è necessario associare una statistica creata in precedenza.

Infine, il servizio permette ai players correttamente autenticati ed in procinto di giocare, di visualizzare tutte le classi precedentemente caricate dagli amministratori.

È riportato di seguito un diagramma dei casi d'uso del microservizio T1, in cui si possono notare i casi d'uso di maggiore interesse nel completamento del task assegnato, ovvero il visualizza achievement (con le operazioni allegate) e il visualizza statistiche (con le operazioni allegate).

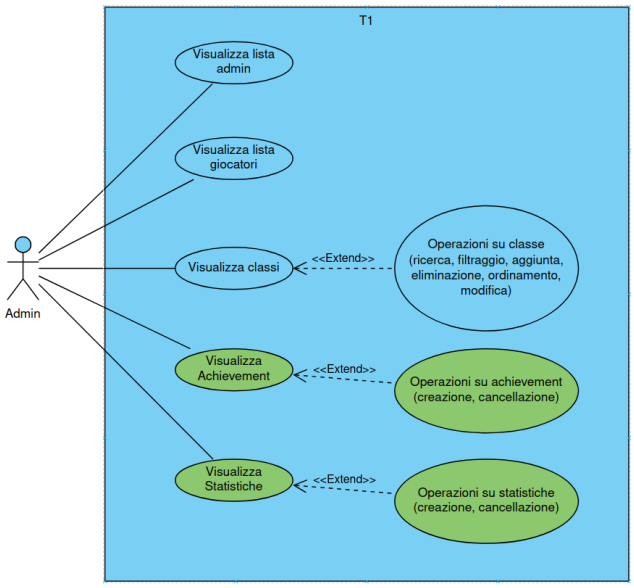


Figura 1.4 – Diagramma dei casi d'uso sintetio di Carmine Palmese, per T1 in A13 del 12/11/2024

1.5.2 Stato del microservizio T23

Il microservizio T23 è responsabile della gestione dei player registrati o no. In particolare, il servizio permette ai players di:

- registrarsi, inserendo: nome, cognome, e-mail, password e specificando il corso di studi (’BSc’, ’MSc’, ’ALTRO’);
- autenticarsi inserendo le proprie credenziali.

Il servizio permette ai players correttamente registrati di impostare una nuova password nel caso in cui l’avessero dimenticata effettuando un’operazione di reset.

Il servizio permette ai players correttamente registrati ed autenticatisi di effettuare il logout.

Essendo, quindi, responsabile della registrazione degli studenti, esso manterrà anche tutte le informazioni relative ad essi. Di seguito è riportato il modello ER del T23, che mette in risalto anche la relazione tra studente e utente autenticato, che sarà ovviamente un’associazione 1-1:

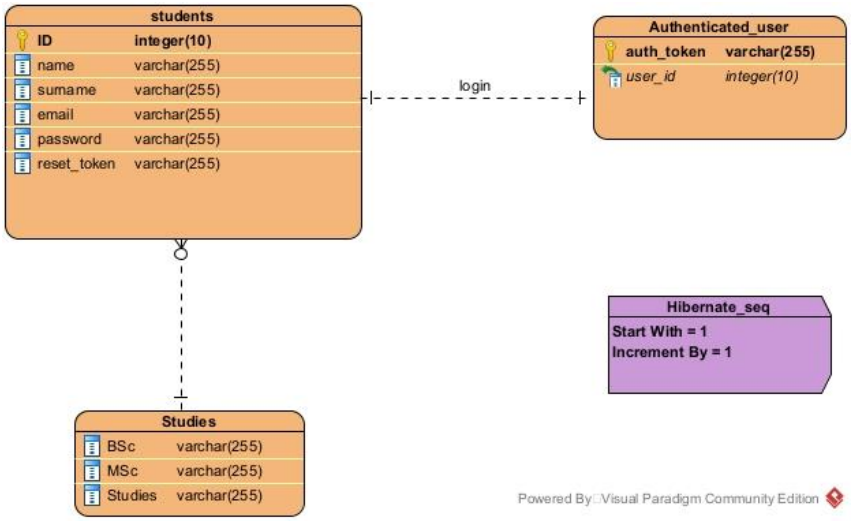


Figura 1.5 – Diagramma ER del T23 in A13 del 12/11/2024

1.5.3 Stato del microservizio T4

Il microservizio T4 è responsabile della gestione delle partite. In particolare, il servizio permette al game engine di:

- creare una partita, memorizzando una serie di informazioni;
- aggiornare la partita per recuperare la data e l’ora di inizio e fine;

- eliminare una partita;
- generare, contestualmente alla partita appena creata, i round;
- aggiornare i round;
- creare nell'ambito dei round, diversi turni associati a ciascun partecipante della partita;
- aggiornare i turni;
- recuperare, durante ciascun round, i punteggi prodotti dai robot relativi alla classe di test in gioco.

Il microservizio T4 è coinvolto nella gestione del profilo perché all'interno sono trattati i dati in merito ai progressi delle statistiche del player, con i quali si permette la corretta visione di queste informazioni all'utente e si determina se un utente ha completato o meno un achievement. Di seguito è riportato un diagramma dei casi d'uso sintetico per il microservizio T4:

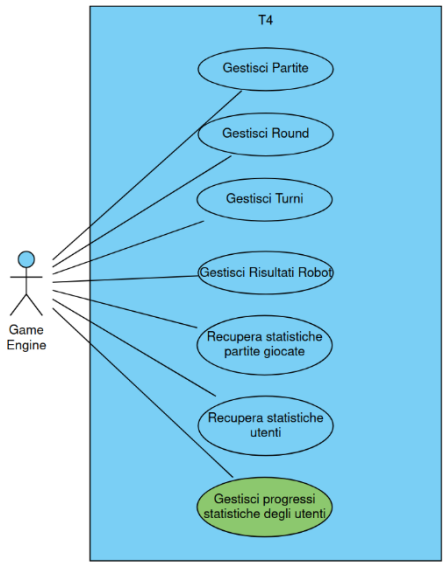


Figura 1.5 – Diagramma Casi d'uso sintetico di Carmine Palmese del T4 in A13 del 28/11/2024

Nota: Come riportato nella documentazione di Palmese, *"il Game Engine non è propriamente un attore, ma si usa questo abuso di concetto per segnalare che i casi d'uso sono attivabili da un componente esterno che è proprio il Game Engine"*

1.5.4 Stato del microservizio T5

Il microservizio T5 rappresenta il nucleo per quanto riguarda la gestione del profilo. È il microservizio con il quale si interfaccia l'utente registrato. Contiene l'interfaccia e la logica

dell'avvio delle partite, della selezione dei parametri delle partite, nonché l'interfaccia del profilo utente, su cui andrà a lavorare il nostro gruppo per completare il task assegnato.

Di seguito è riportato un breve diagramma dei casi d'uso del microservizio T5, nel quale viene posta enfasi su ciò che maggiormente riguarda il nostro task:

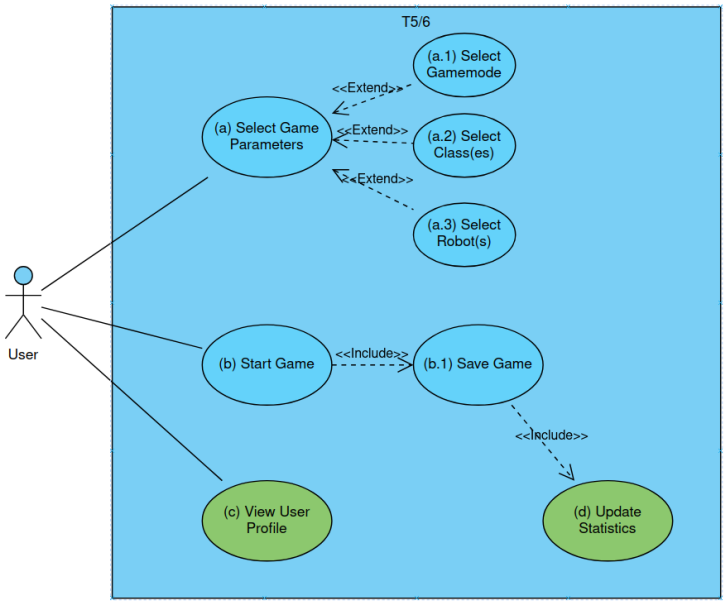


Figura 1.5 – Diagramma Casi d'uso sintetico di Carmine Palmese del T5 in A13 del 28/11/2024

Come si evince dal diagramma dei casi d'uso il T5 si occupa anche di gestire le partite attive in tempo reale.

Il microservizio T5 sarà quello maggiormente impattato dalle modifiche per il completamento del task. Si riporta quindi un diagramma delle classi che concerne solamente i componenti legati alla gestione del profilo:

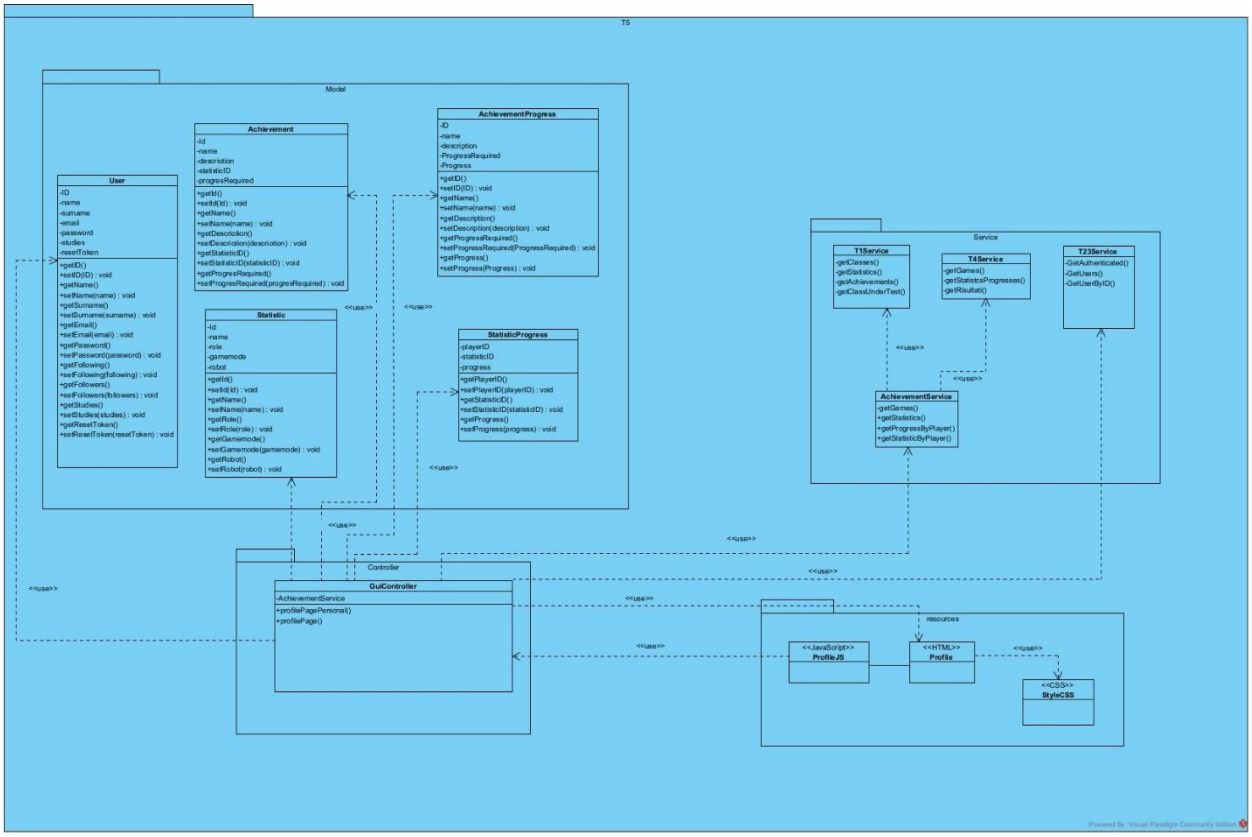


Figura 1.6 – Diagramma delle classi inerenti ai componenti della gestione del profilo della versione A13 del 12/11/2024

1.6 Problematiche riscontrate

È importante notare che, in questa versione del software, sono state riscontrate alcune problematiche riguardanti i microservizi T5, T4 e T23.

Le problematiche individuate legate al T5 sono le seguenti:

- **Aggiornamento delle statistiche di un player:** tale operazione non viene effettuata in modo corretto in quanto non è implementato correttamente il salvataggio delle partite e, di conseguenza, non è possibile per l'applicazione aggiornare correttamente i dati del player, come si evince dal seguente video. Tale problema porta, quindi, ad un'errata visualizzazione sia delle statistiche che degli achievement ottenuti dal player.
- **Comunicazione tra il T5 e il T23:** il prelievo delle informazioni personali del player dal T23 non è gestito in maniera ottimale, essendo il Web Browser ad occuparsi di effettuare richieste dirette al microservizio T23.

Le due problematiche si differenziano per la loro natura e modalità di manifestazione. La prima è direttamente osservabile dall'interfaccia del profilo, poiché le statistiche e gli

achievement del player vengono visualizzati in modo errato a causa del mancato salvataggio corretto delle partite.

La seconda problematica, pur non essendo percepibile dal player, rappresenta una criticità tecnica nella comunicazione tra i microservizi, poiché il prelievo delle informazioni personali dal T23 è gestito in maniera inefficiente tramite richieste dirette effettuate dal Web Browser. Questa inefficienza deriva dal fatto che la logica di integrazione è distribuita nel frontend anziché centralizzata nel backend. Così facendo, il Web Browser, anziché delegare la gestione delle comunicazioni al **GuiController** (ovvero il controller preposto alla gestione delle GUI) del T5, esegue richieste dirette verso il microservizio T23, generando i seguenti problemi:

- La comunicazione diventa meno scalabile e difficile da manutenere, poiché eventuali cambiamenti nel T23 richiederebbero modifiche anche al codice lato frontend.
- Le API dei microservizi risultano esposte al client, aumentando il rischio di vulnerabilità e possibili abusi da parte di utenti malintenzionati.

Inoltre, per quanto riguarda il microservizio T4, si è notato che non esiste un'associazione nel DB Postgre tra robot e partita. Questo è un problema, in quanto non permette il corretto aggiornamento dei progressi delle statistiche dei Player. Ciò è dato dal fatto che non sapendo quale robot è stato sfidato in una partita, non si può aggiornare la statistica corretta (ad esempio "Partite Giocate contro Evosuite"), ma si aggiorneranno arbitrariamente tutte le statistiche che hanno come ruolo "Partite Giocate" senza fare differenza tra i Robot.

Infine, per quanto riguarda il microservizio T23, è stata riscontrata una leggera criticità riguardante l'inserimento del nome e cognome durante la fase di registrazione del player. In particolare, si è notato che non è possibile inserire liberamente né il carattere **apostrofo** né le **vocali accentate** nei suddetti campi, il che è limitante per le registrazioni di persone con un nome e/o cognome contenenti questi caratteri.

2. Metodologie di sviluppo

2.1 Scrum

Per lo sviluppo del progetto è stato adottato un approccio basato sulle metodologie agili, con particolare riferimento al framework Scrum. Questo ha permesso di suddividere il lavoro in iterazioni brevi e incrementali, garantendo un continuo allineamento tra il nostro team e il product owner. Scrum è un framework Agile utilizzato per gestire progetti complessi, in particolare nello sviluppo software, ma applicabile anche ad altri ambiti. È basato su iterazioni incrementali chiamate **Sprint**, che durano tipicamente da 1 a 4 settimane. Durante ogni Sprint, il team lavora su un insieme di obiettivi definiti per consegnare un incremento di prodotto funzionante e potenzialmente utilizzabile. Scrum si basa su tre pilastri principali:

1. **Trasparenza**: Tutti i membri del team devono avere visibilità completa sul lavoro e sui progressi.
2. **Ispezione**: I processi e i risultati vengono regolarmente valutati per garantire il miglioramento continuo.
3. **Adattamento**: Il piano di lavoro può essere modificato in base ai feedback e alle necessità emergenti.

Scrum prevede una serie di eventi strutturati:

1. **Sprint Planning**: Pianificazione degli obiettivi e delle attività dello Sprint.
2. **Daily Scrum**: Riunione giornaliera per sincronizzare il team e identificare ostacoli.
3. **Sprint Review**: Revisione del lavoro svolto con il coinvolgimento degli stakeholder.
4. **Sprint Retrospective**: Riflessione sullo Sprint per identificare miglioramenti.

Gli artefatti di Scrum includono:

- **Product Backlog**: Lista ordinata delle funzionalità e dei requisiti.
- **Sprint Backlog**: Selezione delle attività da completare durante uno Sprint.
- **Incremento**: La somma di tutti i deliverable completati, conforme ai criteri di qualità.

Per tutta la durata del lavoro sono stati effettuati dei daily scrum, nel corso delle quali venivano presentati, da ognuno di noi, i progressi ottenuti rispetto al backlog, insieme alle

perplessità riscontrate. Ciò è stato indispensabile per la risoluzione di ogni problematica affrontata e per una corretta integrazione del lavoro di ogni membro del team.

Sono state inoltre impostate delle scadenze per il completamento dei vari obbiettivi, basandosi sulle date delle review prefissate dai product owner. Le deadline sono state impostate ogni due settimane, al termine delle quali si è valutato se effettivamente si fossero raggiunti gli obbiettivi prefissati, contenuti negli **sprint backlog** (Appendice A) creati all'inizio di ogni sessione di lavoro bi-settimanale.

2.2 Suddivisione del lavoro

Il lavoro è stato inizialmente suddiviso nei seguenti task con stima associata di persone necessarie per portare ciascuno a completamento:

1. Prima iterazione:
    - Analisi dell'architettura di partenza, partendo dalla documentazione generale del progetto. 3 persone
    - Studio del codice. 3 persone
    - Provare il prototipo sulla macchina. 3 persone
2. Seconda iterazione:
    - Analisi e specifica dei requisiti. 1 persona
    - Analisi di impatto. 2 persone
3. Terza iterazione
    - Arricchire specifica dei requisiti. 1 persona
    - Fare diagrammi UML. 1 persona
    - Implementare casi d'uso. 2 persone
4. Quarta iterazione
    - Terminare Implementazione. 2 persone
    - Bug fixing e Testing. 1 persona
    - Fare diagrammi di progetto. 1 persona

2.3 Strumenti utilizzati

Microsoft 365 e Microsoft Teams

Per la realizzazione delle documentazioni, presentazioni e scambi di file abbiamo usato i Software e le architetture Cloud di Microsoft, messe in dotazione con le utenze universitarie. In particolare, sono stati usati OneDrive, Word, PowerPoint e Microsoft Teams; quest'ultimo ovviamente in merito alle videochiamate del team e scambio di messaggi.

GitHub & GitHub Desktop

GitHub è una piattaforma per ospitare e condividere codice, basata su **Git**. Permette a sviluppatori di collaborare su progetti, tracciare modifiche, gestire versioni e automatizzare processi come test e deploy. Offre strumenti per issue tracking, pull request e code review, ed è ampiamente usata per progetti open source e aziendali. È stato inoltre utilizzato **GitHub desktop**, che ha permesso una migliore efficienza e semplicità nelle operazioni di push e pull sul repository del progetto.

Visual Paradigm

Visual Paradigm è stato il tool di modellazione utilizzato, che ha consentito la creazione di tutti i **diagrammi UML** presenti all'interno della documentazione.

Visual Studio Code

L'IDE utilizzato per scrivere il codice è stato VSC (Visual Studio Code), per la sua semplicità ed intuitività.

MongoDB Compass, MySql Workbench, Pgadmin v4

Per controllare i risultati di tutte le operazioni che prevedevano un accesso ai vari database, sono stati utilizzati i seguenti programmi dell'interfacciamento grafico con il DB, ovvero MongoDB Compass, MySql Workbench e Pgadmin v4.

Miro

È una piattaforma collaborativa online basata su una lavagna digitale (board), progettata per facilitare il brainstorming, la pianificazione e la collaborazione tra team. È particolarmente utile per lavorare in modo visuale su progetti complessi, sia in tempo reale che in modalità asincrona**.**

2.4 Tecnologie

Java Spring

È un framework Java potente e flessibile progettato per lo sviluppo di applicazioni enterprise e web. Fornisce un ecosistema ricco di funzionalità per semplificare la creazione, la configurazione e la gestione di applicazioni, seguendo i principi della programmazione orientata agli oggetti. Le caratteristiche principali sono:

- **Modularità**: Composto da diversi moduli che possono essere usati in modo indipendente o combinato, come Spring Core, Spring MVC, Spring Data, Spring Security, e Spring Boot.
- **Inversion of Control (IoC)**: Gestisce automaticamente il ciclo di vita e le dipendenze dei componenti, facilitando la scrittura di codice pulito e mantenibile.
- **Aspect-Oriented Programming (AOP)**: Permette di separare le logiche trasversali (come logging, sicurezza) dal resto dell'applicazione.
- **Spring Boot**: Una sottostruttura che semplifica la configurazione e il deployment di applicazioni grazie a setup predefiniti e server integrati come Tomcat.
- **Integrazione e scalabilità**: Supporta facilmente l'integrazione con database, sistemi di messaggistica e servizi cloud, ed è altamente scalabile.
- **Supporto per microservizi**: Ideale per creare applicazioni basate su microservizi grazie a strumenti come Spring Cloud.

Postman

È stato lo strumento utilizzato per testare, documentare e automatizzare le API (Application Programming Interface) da noi sviluppate. È particolarmente utile per lo sviluppo e il debugging di applicazioni che comunicano attraverso API REST, SOAP, GraphQL o altre tecnologie di integrazione.

Maven

È uno strumento di build e gestione dei progetti, utilizzato principalmente per applicazioni Java. Semplifica il processo di gestione delle dipendenze, la compilazione, il packaging e il deployment del software. Maven segue un modello dichiarativo basato su file XML chiamato pom.xml (Project Object Model), dove si definiscono le dipendenze, le configurazioni di build e i plugin da utilizzare. Scarica automaticamente le librerie richieste dal progetto e le loro dipendenze transitivamente da repository centrali o locali, risparmiando tempo e riducendo la complessità nella gestione manuale. Utilizza una

struttura di progetto standardizzata (ad esempio, src/main/java per il codice sorgente e src/test/java per i test), semplificando la collaborazione tra sviluppatori. In sintesi, Maven è uno strumento potente per organizzare e semplificare la gestione di progetti Java e garantire che il ciclo di vita del progetto sia ben strutturato e ripetibile.

Hibernate

È un framework open-source per Java che semplifica l'interazione con i database relazionali attraverso il object-relational mapping (ORM). Permette di lavorare con dati e relazioni come oggetti Java, eliminando la necessità di scrivere molte query SQL manuali. È indipendente dal database, offre una gestione automatizzata delle operazioni CRUD e delle transazioni, e include meccanismi di caching per migliorare le prestazioni. Inoltre, supporta relazioni complesse tra entità e integra query native tramite JDBC se necessario. In breve, Hibernate rende lo sviluppo con database più semplice, efficiente e portabile.

JQuery

È una libreria JavaScript, progettata per semplificare la manipolazione del DOM, la gestione degli eventi, l'esecuzione di animazioni e le richieste AJAX. Introdotta per ridurre la complessità del codice JavaScript tradizionale, jQuery consente di interagire con gli elementi di una pagina web attraverso una sintassi chiara. È stato utilizzato per semplificare alcune operazioni legate alla manipolazione del DOM e alle richieste AJAX. Viene utilizzato per effettuare chiamate HTTP verso l'API, semplificando la configurazione e gestione delle richieste.

Bootstrap

È un framework front-end open-source progettato per facilitare lo sviluppo di siti web e applicazioni web responsive e moderne. Fornisce un insieme di strumenti predefiniti, tra cui griglie flessibili, componenti UI come pulsanti, modal e navbar, e stili CSS personalizzabili. Bootstrap utilizza HTML, CSS e JavaScript per creare layout che si adattano automaticamente a dispositivi di diverse dimensioni, semplificando il design responsive.

3. Task R1

3.1 Storie utente

La definizione delle storie utente ha rappresentato il primo passo verso una corretta specifica dei requisiti, in quanto queste forniscono una descrizione chiara e comprensibile dei requisiti del sistema dal punto di vista degli utenti finali.

Di seguito sono riportate le storie utente identificate per il task assegnato:

Figure 3.1 – User stories

3.2 Requisiti funzionali e non funzionali

I requisiti funzionali e non funzionali rappresentano la base per una buona progettazione, in quanto definiscono le funzionalità che il sistema deve offrire e quali vincoli prestazionali devono essere rispettati per fare ciò.

I requisiti funzionali identificati sono i seguenti:

RF1) Il sistema deve offrire ai Player registrati una funzionalità per **visualizzare correttamente** le proprie **informazioni personali**.

RF2) Il sistema deve offrire ai Player registrati una funzionalità per **modificare correttamente** le proprie **informazioni personali**.

RF3) Il sistema deve offrire ai Player registrati una funzionalità per **visualizzare correttamente** i **propri progressi** rispetto alle **statistiche** di gioco.

RF4) Il sistema deve offrire ai Player registrati una funzionalità per **filtrare i propri progressi delle statistiche** di gioco per **modalità di gioco** e **robot.**

RF5) Il sistema deve offrire ai Player registrati una funzionalità per **visualizzare correttamente** gli **Achievement ottenuti**.

RF6) Il sistema deve offrire ai Player registrati una funzionalità per **visualizzare correttamente** tutti gli **Achievement ottenibili**.

RF7) Il sistema deve offrire ai Player registrati una funzionalità per **visualizzare correttamente** i **token** guadagnati attraverso il completamento delle missioni.

RF8) Il sistema deve offrire ai Player registrati una funzionalità per **cercare altri player**.

RF9) Il sistema deve offrire ai Player registrati una funzionalità per **seguire altri player**.

RF10) Il sistema deve offrire ai Player registrati una funzionalità per **visualizzare la lista dei suoi seguiti**.

RF11) Il sistema deve offrire ai Player registrati una funzionalità per **visualizzare i profili dei suoi seguiti**.

RF12) Il sistema deve offrire ai Player registrati una funzionalità per **smettere di seguire** i player presenti nella lista seguiti.

I requisiti non funzionali individuati sono i seguenti:

RNF1) Il sistema deve garantire **la consistenza** dei dati dopo una modifica delle informazioni personali.

RFN2) Il sistema deve garantire una responsività ottimale durante la visualizzazione dell'interfaccia del profilo giocatore.

3.3 Glossario

La creazione del glossario è stata ritenuta necessaria per evitare qualsiasi tipo di ambiguità nell'interpretazione dei vari artefatti creati.

Il glossario è il seguente:

NOME	**DESCRIZIONE**
Player Registrato	Utente registrato come studente. Sinonimo: giocatore, user
Informazioni personali	Nome, cognome, e-mail, password, titolo di studio, biografia
Modalità di gioco	Sfida, allenamento, scalata e multiplayer
Robot	I robot che si possono sfidare nel gioco, ovvero Randoop ed Evosuite.
Statistica	Parametri utili per tenere traccia dei progressi in gioco di un player. Ogni Statistica è legata ad una "modalità di gioco", un "Robot" ed un "Ruolo", che può essere Partite Giocate, Partite Vinte o Score.
Ratio	Ove sono state dichiarate dall'Admin due Statistiche con medesima "Modalità di Gioco" e "Robot" ma una con il ruolo "Partite Giocate" e l'altra "Partite Vinte". Il Ratio è il rapporto tra P.V./P.G. Sarà quindi un valore compreso tra 0 e 1.
Achievement	Ricompensa ottenibile al raggiungimento di un determinato traguardo. Prevede l'assegnazione di un badge.
Badge	Riconoscimento tramite un supporto grafico del raggiungimento di un achievement.
Lista seguiti	Lista di altri player registrati seguiti.

Tabella 3.1 - Glossario

3.4 Analisi dei casi d'uso

Di seguito è riportata l'analisi dei casi d'uso effettuata in seguito alla specifica dei requisiti, compresa di attori coinvolti e diagramma UML.

ATTORI PRIMARI:

- Player registrato

ATTORI SECONDARI:

- Servizio di posta elettronica

CASI D'USO:

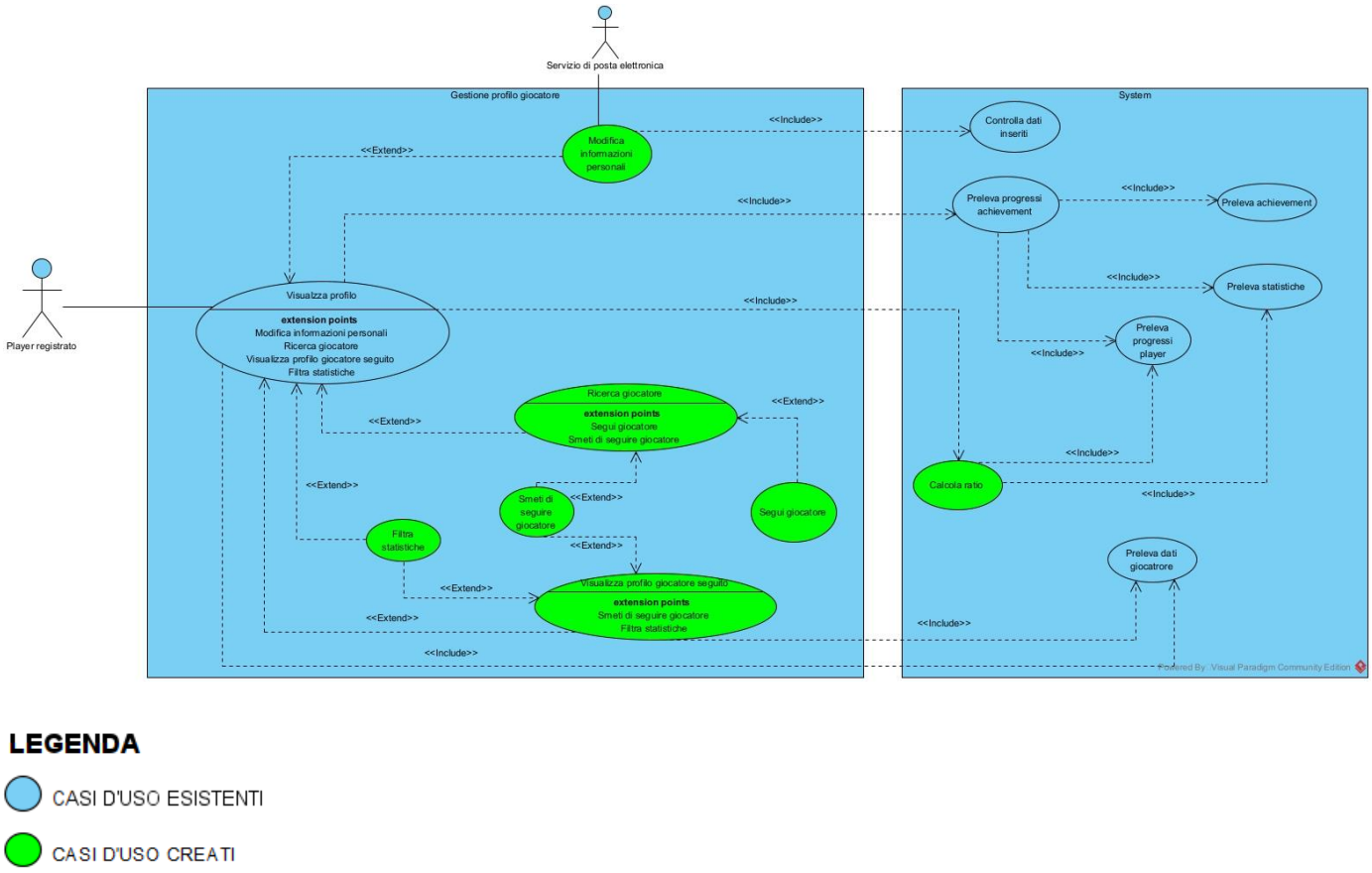
- Visualizza profilo

CASI D'USO DI INCLUSIONE:

- Preleva dati giocatore
- Controlla dati inseriti
- Preleva progressi achievement
- Preleva achievement
- Preleva statistiche
- Preleva progressi player
- Calcola ratio

CASI D'USO DI ESTENSIONE:

- Modifica informazioni personali
- Filtra statistiche
- Ricerca giocatore
- Segui giocatore
- Smetti di seguire giocatore
- Visualizza profilo giocatore seguito



LEGENDA

CASI D'USO ESISTENTI

CASI D'USO CREATI

Figure 3.2 – Diagramma UML casi d'uso

DESCRIZIONE CASI D'USO

- **Visualizza profilo:** è l'unico caso d'uso con cui si interfaccia il player registrato; esso consente all'utente di accedere alle informazioni dettagliate del proprio profilo, composte dalle informazioni personali, achievement ottenuti e ottenibili, statistiche di gioco e lista seguiti. In particolare, oltre alla semplice visualizzazione dei dati, permette la modifica delle informazioni personali, il filtraggio delle statistiche, la ricerca di un giocatore e la visualizzazione del profilo di un giocatore seguito.
- **Modifica informazioni personali:** è un caso d'uso di estensione di 'Visualizza Profilo' e consente al player registrato di modificare alcune o tutte le proprie informazioni personali. Affinché tali modifiche vengano apportate, è necessario inserire la password attuale, qualsiasi siano le informazioni che si vogliono aggiornare. Nel caso in cui venissero modificate e-mail e/o password, prevede l'invio di una e-mail per notificare l'aggiornamento avvenuto.
- **Controlla dati inseriti:** è un caso d'uso di inclusione di 'Modifica informazioni personali' e verifica sia che la password attuale inserita dal player è corretta, sia

che le informazioni personali aggiornate inserite dal player registrato rispettino tutti i criteri imposti. Essi sono:

- Nome: lunghezza minima di 2 caratteri e lunghezza massima di 30 caratteri, consentiti gli spazi, lettere accentate, apostrofi e non deve contenere caratteri speciali.
- Cognome: lunghezza minima di 2 caratteri e lunghezza massima di 30 caratteri, consentiti gli spazi, lettere accentate, apostrofi e non deve contenere caratteri speciali.
- E-mail: deve contenere il carattere '@' e il carattere '.'
- Biografia: lunghezza minima di 2 caratteri e lunghezza massima di 120 caratteri, consentiti: numeri, gli spazi, lettere accentate, apostrofi, ed esclusivamente i caratteri di punteggiatura: punto esclamativo, punto interrogativo, trattino e virgola. Qualunque altro carattere speciale o di punteggiatura sono esclusi.
- Password: lunghezza minima di 8 caratteri e lunghezza massima di 16 caratteri

Nota: i caratteri speciali sono: numeri, punteggiatura, simboli matematici e altri caratteri non alfanumerici, caratteri Unicode non latini.

- **Preleva progressi achievement:** è un caso d'uso di inclusione di 'Visualizza Profilo' ed è responsabile del prelievo dei progressi del player registrato rispetto alle statistiche associate agli achievement esistenti. Esso si serve di altri casi d'uso per completare questa operazione.
- **Preleva achievement:** è un caso d'uso di inclusione di 'Preleva progressi achievement' ed è responsabile del prelievo di tutti gli achievement precedentemente inseriti dall'admin, ad ognuno dei quali è associata una determinata statistica.
- **Preleva statistiche:** è un caso d'uso di inclusione di 'Preleva progressi achievement' ed è responsabile del prelievo di tutte le statistiche precedentemente inserite dall'admin.
- **Preleva progressi player:** è un caso d'uso di inclusione di 'Preleva progressi achievement' ed è responsabile del prelievo dei progressi del player rispetto alle statistiche esistenti.
- **Calcola ratio** è un caso d'uso di inclusione di 'Visualizza Profilo' e si occupa del calcolo delle ratio per tutte le statistiche rispetto alle quali esiste un progresso del player diverso da 0.

- **Segui giocatore:** è un caso d'uso di estensione di 'Ricerca Giocatore' e consente di seguire il giocatore associato alla chiave di ricerca inserita.
- **Smetti di seguire giocatore:** è un caso d'uso di estensione di 'Ricerca giocatore' e 'Visualizza profilo giocatore seguito' e consente al player registrato di smettere di seguire un giocatore precedentemente seguito.
- **Ricerca Giocatore:** è un caso d'uso di estensione di 'Visualizza Profilo' e consente di ricercare un giocatore tramite e-mail o ID di gioco. Inoltre, permette al player di scegliere di seguire il giocatore (se non lo fa già) o di smettere di seguirlo (se lo segue già).
- **Filtra statistiche:** è un caso d'uso di estensione di 'Visualizza Profilo' e 'Visualizza profilo giocatore seguito' e consente al player di filtrare le statistiche di gioco per modalità e robot.
- **Visualizza profilo giocatore seguito:** è un caso d'uso di estensione di 'Visualizza Profilo' e consente al player registrato di visualizzare il profilo di un giocatore presente nella lista seguiti, al cui interno non è possibile alcuna interazione se non la possibilità di smettere di seguire il giocatore stesso oppure filtrare le sue statistiche di gioco.

Nota: all'interno del diagramma dei casi d'uso non sono presenti casi d'uso del tipo 'Visualizza Achievement' oppure 'Visualizza Statistiche' dato che tale visualizzazione è delegata a 'Visualizza Profilo' il quale, in seguito al retrieve di tutti i dati necessari, ha il compito di mostrarle a schermo.

3.5 Scenari

Di seguito sono riportati i scenari relativi al caso d'uso Visualizza Profilo e ai casi d'uso di estensione di quest'ultimo.

Per evitare ridondanza, sono stati inseriti solamente i nomi dei casi d'uso di inclusione e di estensione, essendo essi stati descritti precedentemente, nel paragrafo 3.4.

CASO D'USO	Visualizza Profilo
ATTORE PRIMARIO	Player registrato
ATTORE SECONDARIO	Nessuno
DESCRIZIONE	Permette ad un player registrato di visualizzare il proprio profilo contenente le informazioni personali, le statistiche di

	gioco, gli achievement ottenuti e ottenibili, e la lista di giocatori seguiti.
PRE-CONDIZIONI	Il player deve essere correttamente registrato e autenticato nel sistema.
SCENARIO PRINCIPALE DI SUCCESSO	1) Il player accede all'area 'Profilo' 2) **<<Include>> Preleva dati giocatore** 3) **<<Include>> Preleva progressi achievement** 4) **<<Include>> Calcola ratio** 5) Il sistema mostra a schermo tutte i dati dell'utente, organizzandoli in diverse sezioni all'interno della pagina 'Profilo'.
POST-CONDIZIONI	Nessuna modifica persistente è stata effettuata sui dati del sistema.
FLUSSI ALTERNATIVI	

Tabella 3.2 – Scenario caso d'uso 'Visualizza Profilo'

Nota: i flussi alternativi dovrebbero comprendere tutti i casi d'uso di estensione, i quali però sono descritti in maniera dettagliata negli scenari riportati di seguito.

CASO D'USO	Modifica informazioni personali
ATTORE PRIMARIO	Player registrato
ATTORE SECONDARIO	Servizio di posta elettronica
DESCRIZIONE	Permette ad un player di modificare le proprie informazioni personali (nome, cognome, e-mail, password, studies e biografia)
PRE-CONDIZIONI	• Il player deve essere correttamente registrato e autenticato nel sistema. • Il player deve trovarsi nella propria pagina 'Profilo'
SCENARIO PRINCIPALE DI SUCCESSO	1) Il player **seleziona 'Modifica'** dalla sezione delle informazioni personali 2) Il player sostituisce alcune o tutte le informazioni vecchie con quelle aggiornate 3) Il player inserisce anche la password attuale, qualsiasi sia la modifica che vuole effettuare 4) Il player conferma le modifiche 5) **<<*Include*>> Controlla dati inseriti** 6) Il sistema restituisce un messaggio di avvenuta modifica
POST-CONDIZIONI	Le nuove informazioni vengono salvate e mostrate a schermo al player
FLUSSI ALTERNATIVI	4.a) Il player non conferma le modifiche, quindi non si ha nessun aggiornamento delle informazioni personali.

	5.a) Il player inserisce una password sbagliata nel campo 'Password attuale' e il sistema mostra un messaggio di **ERRORE**, annullando le modifiche. 5.b) Le nuove informazioni personali inserite dal player non rispettano i criteri richiesti e il sistema mostra un messaggio di **ERRORE**, annullando le modifiche. 5.c) Se il campo modificato è una e-mail oppure una password, il sistema invia una e-mail di notifica.

Tabella 3.3 – Scenario caso d'uso 'Modifica Informazioni Personali'

CASO D'USO	Ricerca giocatore
ATTORE PRIMARIO	Player registrato
ATTORE SECONDARIO	Nessuno
DESCRIZIONE	Permette ad un player di ricercare un giocatore nel sistema utilizzando come chiave di ricerca un ID univoco o un'email.
PRE-CONDIZIONI	• Il player deve essere correttamente registrato e autenticato nel sistema. • Il player deve trovarsi nella propria pagina 'Profilo'
SCENARIO PRINCIPALE DI SUCCESSO	1) Il player seleziona 'Ricerca giocatore' dalla sezione 'Seguiti' del profilo. 2) Il player inserisce un criterio di ricerca 3) Il sistema mostra a schermo il giocatore che rispetta il criterio inserito.
POST-CONDIZIONI	Nessuna modifica viene apportata ai dati del sistema
FLUSSI ALTERNATIVI	2.a) Il player cerca sé stesso 2.a.1) Il sistema restituisce un messaggio informativo che segnala che il player non può cercare sé stesso. 3.a) Nessun giocatore corrisponde al criterio di ricerca 3.a.1) Il sistema restituisce un messaggio informativo che segnala che nessun giocatore corrisponde ai criteri inseriti. 3.a.2) Il player può modificare i criteri di ricerca o annullare l'operazione. 3.b) Il giocatore ricercato non è presente nella lista di giocatori seguiti e il player sceglie di seguirlo 3.b.1) **<<Extend>> Segui giocatore** 3.b.2) Il sistema aggiorna la lista dei giocatori seguiti aggiungendo il giocatore selezionato 3.c) Il giocatore ricercato è già presente nella lista di giocatori seguiti e il player sceglie di smettere di seguirlo

	3.c.1) **<<Extend>> Smetti di seguire giocatore** 3.c.2) Il sistema aggiorna la lista dei giocatori seguiti rimuovendo il giocatore selezionato.

Tabella 3.4 – Scenario caso d'uso 'Ricerca Giocatore'

CASO D'USO	Filtra statistiche
ATTORE PRIMARIO	Player registrato
ATTORE SECONDARIO	Nessuno
DESCRIZIONE	Permette ad un player di filtrare i progressi di gioco rispetto ad una modalità di gioco e ad un robot.
PRE-CONDIZIONI	• Il player deve essere correttamente registrato e autenticato nel sistema. • Il player deve trovarsi nella propria pagina 'Profilo' • Le statistiche devono essere state aggiunte in precedenza dall'admin
SCENARIO PRINCIPALE DI SUCCESSO	1) Il player seleziona il robot e la modalità di gioco rispetto ai quali vuole filtrare le statistiche. 2) Il sistema mostra a schermo le statistiche filtrate, aggiungendo anche la ratio delle stesse.
POST-CONDIZIONI	Nessuna modifica viene apportata ai dati del sistema
FLUSSI ALTERNATIVI	nessuno

Tabella 3.5 – Scenario caso d'uso 'Filtra statistiche'

CASO D'USO	Visualizza profilo giocatore seguito
ATTORE PRIMARIO	Player registrato
ATTORE SECONDARIO	Nessuno
DESCRIZIONE	Permette ad un player di visualizzare il profilo di un giocatore presente nella lista seguiti.
PRE-CONDIZIONI	• Il player deve essere correttamente registrato e autenticato nel sistema. • Il player deve trovarsi nella propria pagina 'Profilo' • Il player deve aver aggiunto alla lista seguiti il giocatore del quale vuole visualizzare il profilo
SCENARIO PRINCIPALE DI SUCCESSO	1) Il player clicca sul nome del giocatore di cui vuole visualizzare il profilo. 2)**<<Include>> Preleva dati giocatore** 3) Il sistema mostra a schermo la pagina 'Profilo' del giocatore selezionato dal player.
POST-CONDIZIONI	Nessuna modifica viene apportata ai dati del sistema
FLUSSI ALTERNATIVI	3.a) Il player decide di smettere di seguire il giocatore 3.a.1) **<<Extend>> Smetti di seguire giocatore**

	3.a.2) Il sistema richiede conferma. 3.a.2.a) Il player conferma la sua scelta 3.a.2.a.1) Il sistema aggiorna la lista seguiti del player e lo reindirizza alla pagina 'Profilo'. 3.a.2.b) Il player non conferma la sua scelta. 3.a.2.b.1) Il sistema annulla l'operazione e rimane sulla pagina attuale 3.b) **<<Extend>> Filtra statistiche.** 3.b.1) Il sistema mostra a schermo le statistiche filtrate

Tabella 3.6 – Scenario caso d'uso 'Visualizza Profilo Giocatore Seguito'

3.6 Diagrammi di sequenza di analisi

I diagrammi di sequenza sono estremamente utili per riportare il flusso di esecuzione all'interno di un determinato caso d'uso.

All'interno di tali diagrammi si troveranno come **lifeline** tre componenti:

- **Web browser:** è l'elemento che consente l'interfacciamento tra il player registrato e la Web application.
- **Back-end:** è il responsabile delle elaborazioni dei dati e delle interazioni con i database.
- **Database:** rappresenta l'unità di archiviazione dei dati. È importante ricordare che all'interno della Web Application non esiste un solo database, ma in questa fase di analisi si è scelto di generalizzare, considerando il livello di astrazione al quale ci si trova in questi diagrammi.

Vengono di seguito riportati i diagrammi di sequenza ad alto livello per ogni caso d'uso descritto nel paragrafo precedente escluso 'Filtra statistiche', essendo quest'ultimo molto banale e non prevedendo alcuna interazione con il back-end.

Diagramma di sequenza caso d'uso 'Visualizza Profilo'

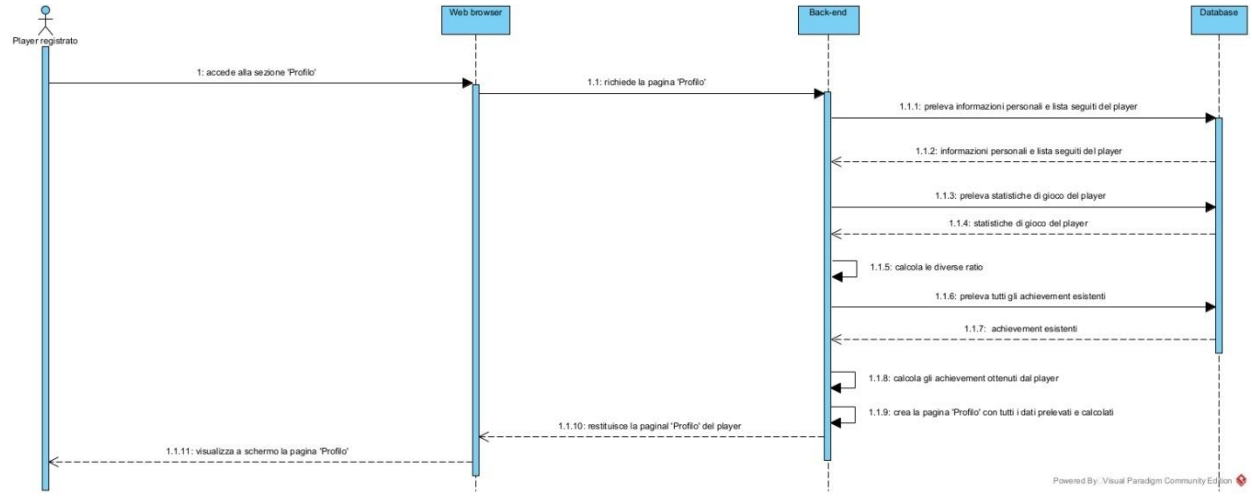


Figura 3.3 – Sequence diagram relative al caso d'uso 'Visualizza Profilo'

Diagramma di sequenza caso d'uso 'Modifica Informazioni Personali'

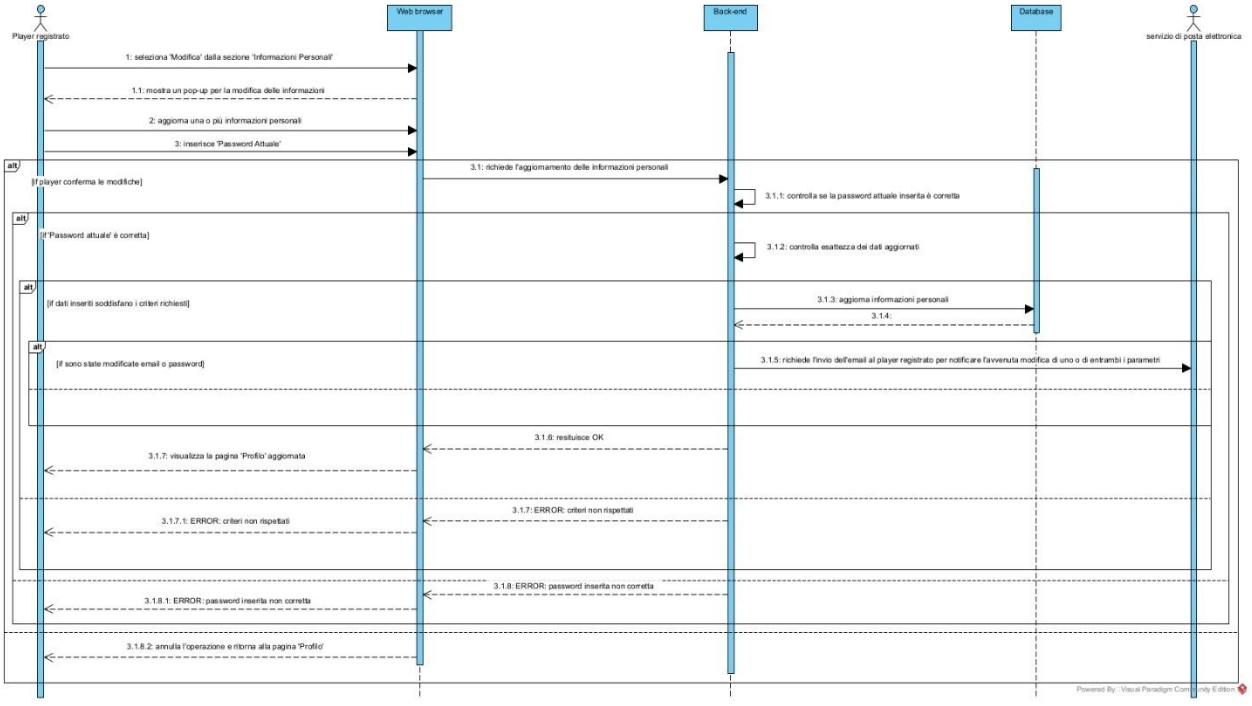


Figura 3.4 – Diagramma di sequenza di analisi del caso d'uso 'modifica informazioni personali'

Diagramma caso d'uso 'Ricerca Player'

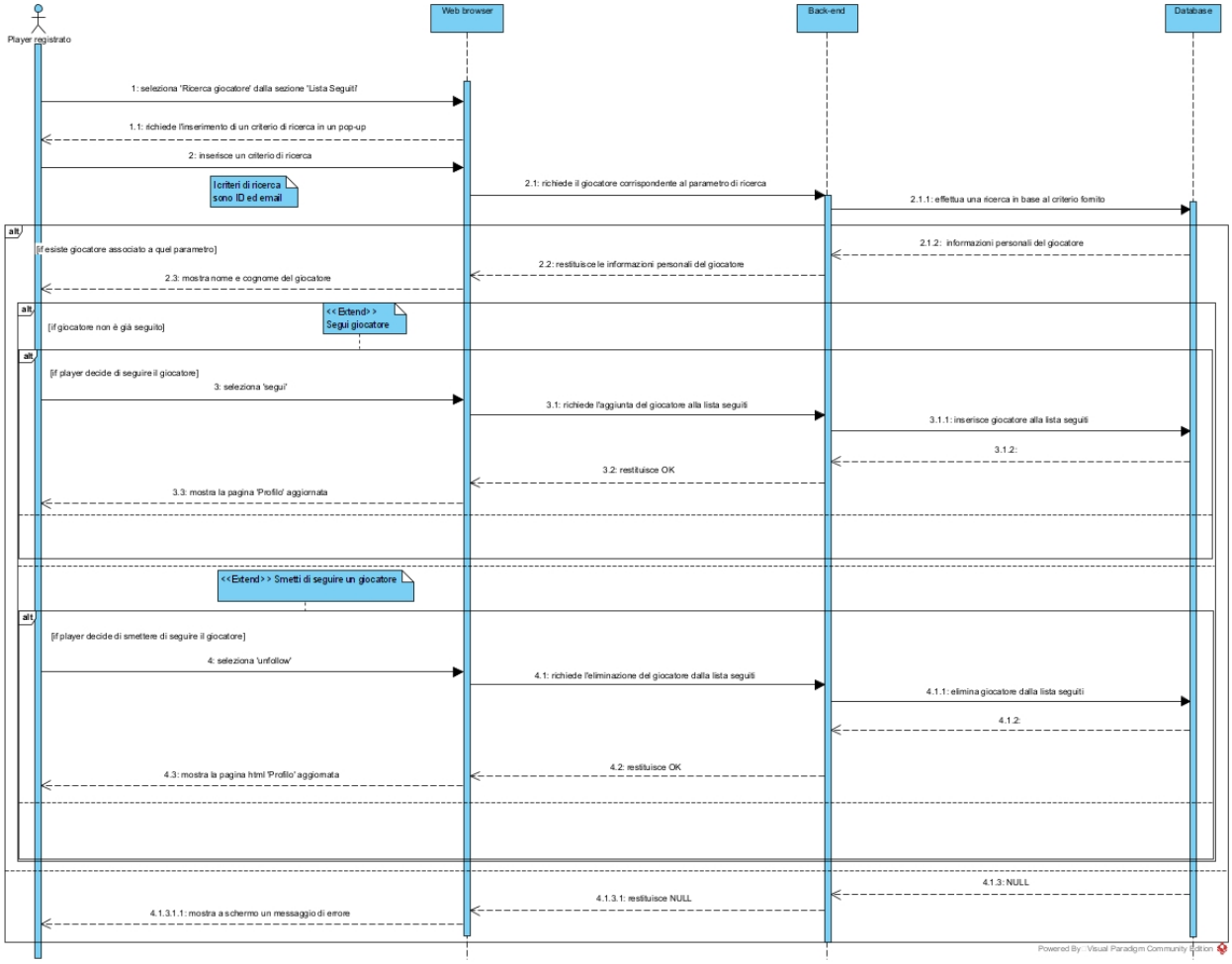


Figura 3.5 – Sequence diagram relative al caso d'uso 'Ricerca giocatore'

All'interno di questo diagramma sono presenti diversi blocchi 'Alt'. Alcuni di essi sono stati utilizzati per mettere in evidenza i casi d'uso di estensione 'Segui giocatore' e 'Smetti di seguire giocatore', essendo questi non sempre attivati nel caso d'uso principale.

Diagramma caso d’uso ‘Visualizza Profilo Giocatore Seguito’

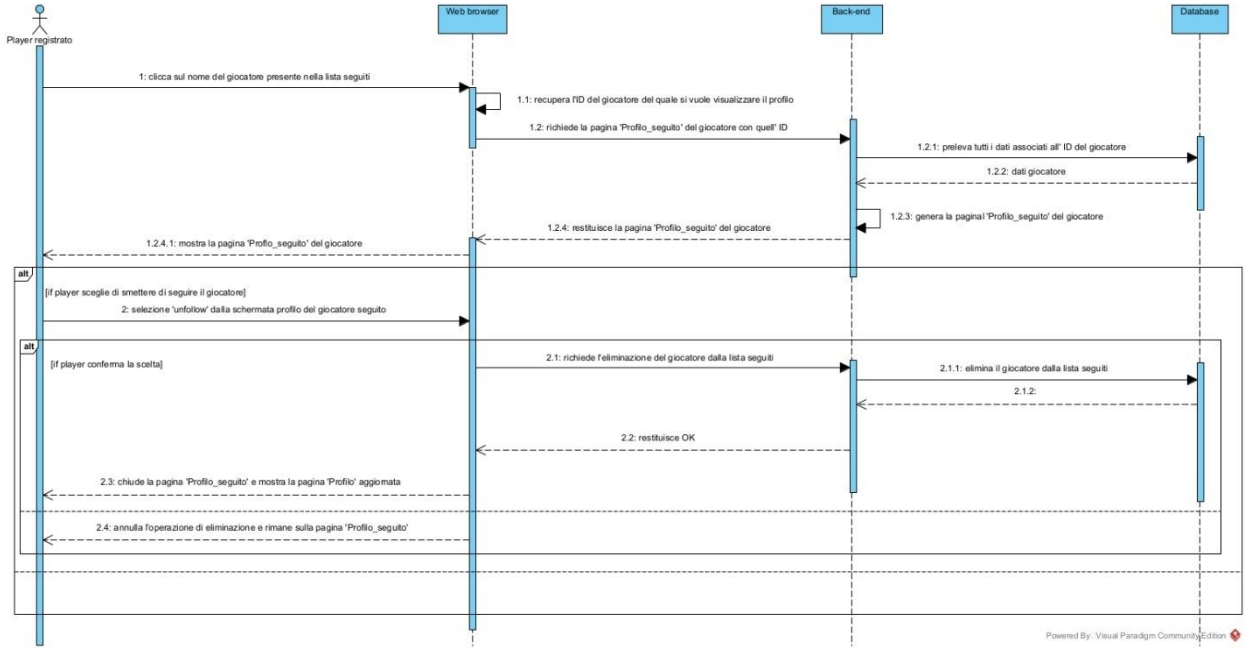


Figura 3.6 – Sequence diagram relativo al caso d’uso ‘Visualizza profilo giocatore seguito’

3.7 Progettazione

Precedentemente alla fase di progettazione, è stato effettuato uno studio molto approfondito sull'architettura del software utilizzata, soffermandosi principalmente sulle interazioni tra le varie parti della stessa.

A seguito dello studio sull'architettura e sulle interazioni tra i vari microservizi, sono stati individuati quali componenti e quali moduli sarebbero stati impattati dalle modifiche per la portata al termine del task.

Di seguito verranno presentate le scelte progettuali effettuate a valle di questo studio.

3.7.1 Component diagram

Il **Diagramma dei Componenti** (Component Diagram), rappresenta un utile strumento perchè:

a. fornisce una visione d'insieme dell'architettura, mostrando come i vari componenti sono organizzati ed interconnessi tra di loro in maniera tale da semplificare la comprensione del progetto agli occhi degli stakeholders
b. consente di evidenziare e comprendere le dipendenze tra questi componenti
c. consente di identificare i punti critici e potenziali problemi di progettazione.

Di seguito è riportato il Component Diagram riguardante solamente i componenti coinvolti nella gestione del profilo.

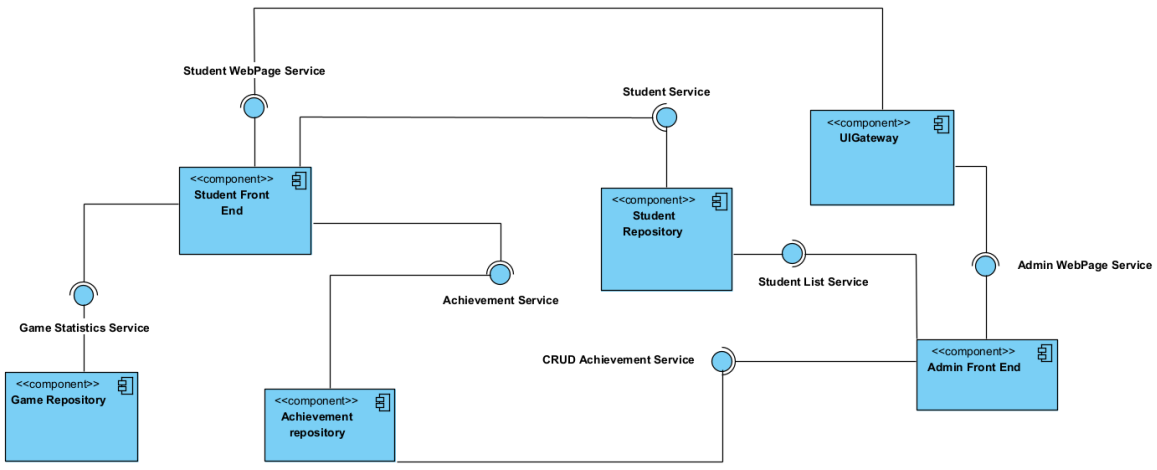


Figura 3.7 – Component diagram per la gestione profilo

Dei componenti e delle interfacce presenti in questo diagramma, sono state apportate modifiche prevalentemente a:

- **Student Repository,** per l'aggiunta di ulteriori informazioni relative allo studente**.**
- **Student Service**, il quale consente, oltre al retrieve delle informazioni, anche la modifica delle informazioni personali e l'interazione tra due student attraverso un meccanismo di following.
- **Student Front-end,** poiché sono state apportate modifiche alla pagina con cui lo studente interagisce e soprattutto il modo in cui tale pagina interagisce con il sistema.

Tali modifiche verranno descritte in maniera approfondita nel paragrafo 3.7.3.

3.7.2 Composite diagram

Per completezza si riporta anche il corrispettivo **Composite Diagram**, necessario per la visualizzazione dei task responsabili dello sviluppo e della gestione dei singoli componenti (anche in questo caso si fa solo riferimento ai componenti che concernono la gestione del profilo).

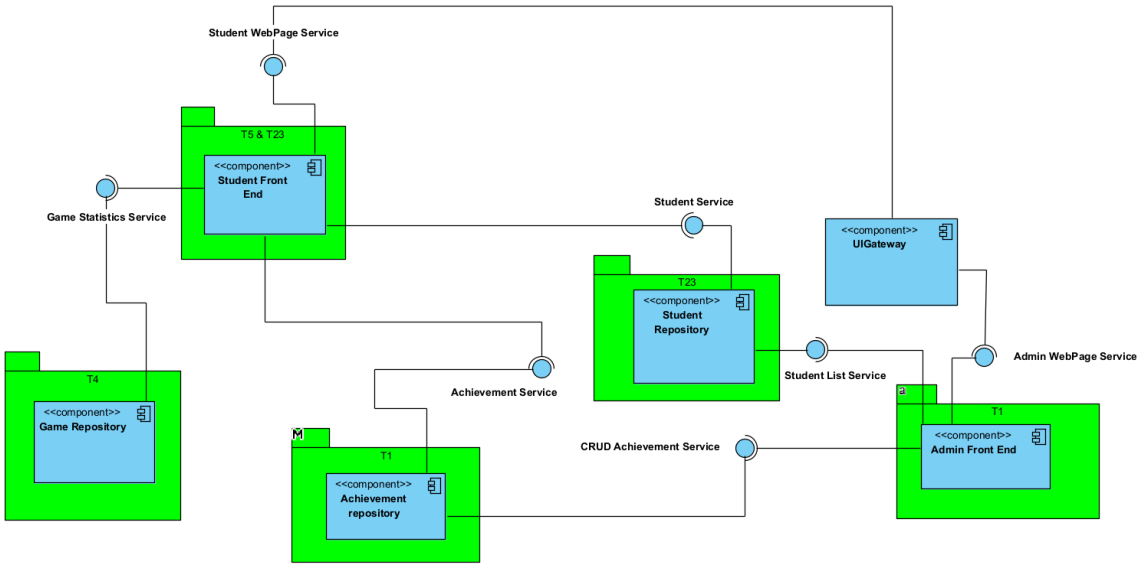


Figura 3.8 – Composite diagram per la gestione profilo

3.7.3 Interazione tra microservizi

L'interazione tra i microservizi che compongono la Web Application è un aspetto molto critico, a causa dell'elevata modularità dovuta allo stile architetturale utilizzato.

Per quanto riguarda la gestione del profilo, i microservizi coinvolti sono **T1, T23, T4, T5,** il cui stato di partenza presentava diverse criticità riguardanti la loro comunicazione.

Come già detto nel capitolo precedente, la logica di presentazione del profilo è contenuta nel microservizio T5, ed è quest'ultimo il punto da cui partono tutte le operazioni riguardanti la gestione del profilo. In particolare, è compito del GuiController predisporre il prelievo dei dati dai diversi microservizi, oltre che gestire le richieste del player effettuate tramite la view **Profile**.

Vediamo più nel dettaglio come comunicano i diversi microservizi coinvolti.

INTERAZIONE T5-T1

Il microservizio T1 è responsabile della gestione degli achievement e delle statistiche ad essi associate. La comunicazione tra T5 e T1 è necessaria per la corretta visualizzazione delle statistiche di gioco del player e per la visualizzazione degli achievement ottenuti ed ottenibili. All'interno del T5 è già presente un service **AchievementService** che gestisce completamente questa comunicazione, servendosi del **ServiceManager,** il quale effettua un routing delle richieste ai servizi competenti, e del **T1Service,** che realizza l'effettiva comunicazione con il controller del microservizio T1 chiamato **HomeController**. Sarà quest'ultimo ad effettuare le operazioni richieste interfacciandosi con il DB del T1, ovvero **MongoDB.**

Il flusso di comunicazione per il retrieve degli achievement è il seguente (è lo stesso anche per il prelievo delle statistiche, cambiano solamente metodi e route implicati):

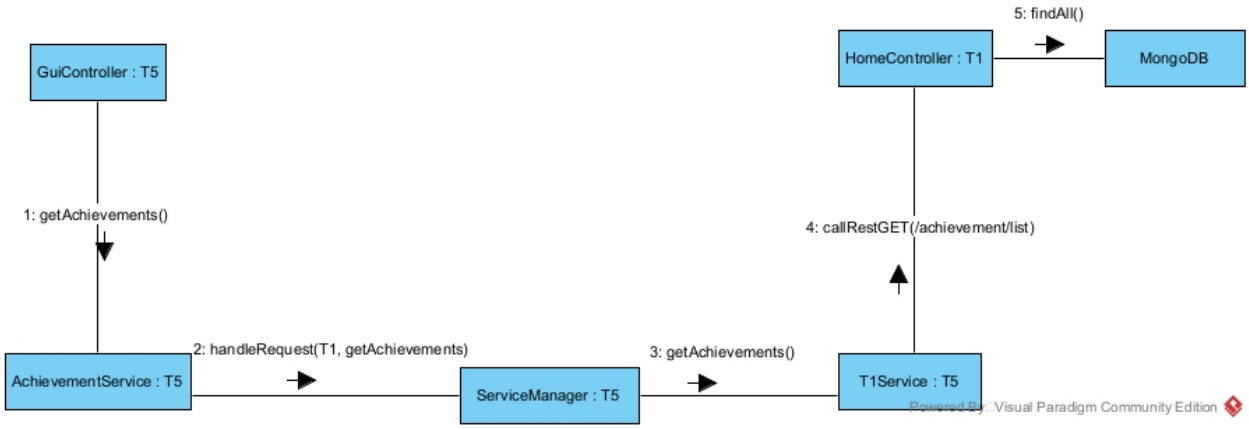


Figure 3.9 – Communication Diagram T1-T5 per il prelievo degli achievement

Non sono state apportate modifiche a questo flusso.

INTERAZIONE T5-T4

Il microservizio T4 è responsabile della gestione dei dati delle partite, in particolare anche per la memorizzazione dei progressi delle statistiche per Player. Per cui la comunicazione con questo è indispensabile per popolare la sezione 'Statistiche' del profilo con i valori corretti associati al giocatore. In questo caso il GuiController utilizza ancora una volta AchievementService, che effettua la richiesta al ServiceManager**,** il quale la reindirizza al **T4Service,** responsabile dell'effettiva comunicazione con il microservizio T4. In particolare, la richiesta viene inviata al controller, denominato **Controller**, il quale effettua le operazioni richieste interfacciandosi con il DB del T4, ovvero **Postgre DB.**

Il flusso di comunicazione è il seguente:

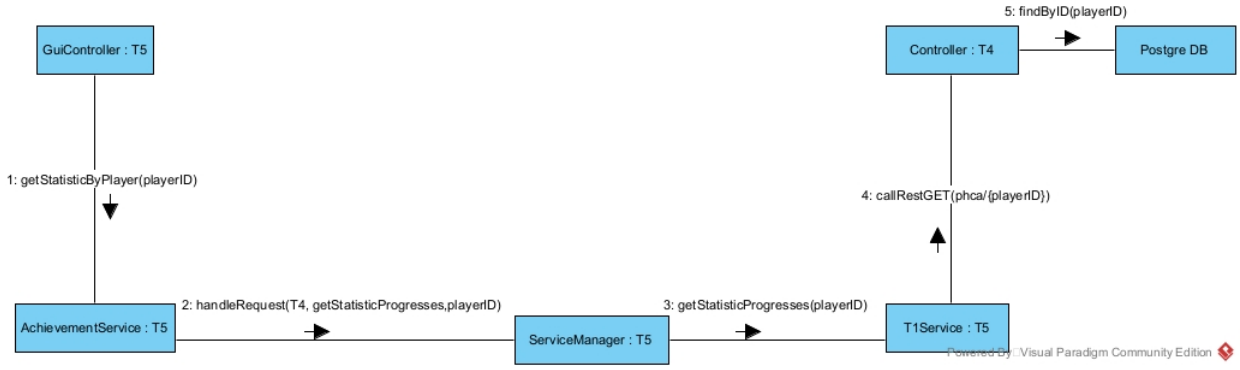


Figure 3.10 – Communication Diagram T4-T5 for il prelievo delle statistiche del player

Non sono state apportate modifiche a questo flusso.

INTERAZIONE T5-T23

Il microservizio T23 è responsabile della gestione dei player registrati. La comunicazione con tale microservizio è necessaria sia per ottenere le informazioni personali e la lista di giocatori seguiti di un player registrato, sia per le nuove funzionalità descritte nella fase di analisi.

Come riportato nel paragrafa 1.6, tale comunicazione prevedeva una criticità, ovvero una comunicazione diretta tra il Web browser e il microservizio T23. Affinché ciò non accadesse, è stato necessario reindirizzare tutte le richieste effettuate dallo stesso Web browser verso il GuiController, in modo tale da avere una logica centralizzata.

Inoltre, per una maggiore chiarezza e pulizia della comunicazione, è stato deciso di creare un servizio, denominato **UserService**, preposto unicamente alla gestione delle operazioni tra T5 e T23. Il ruolo dello UserService è molto simile a quello di AchievementService, con la differenza che UserService interagisce unicamente con il **T23Service** attraverso il ServiceManager. Sarà poi il T23Service a farsi carico della comunicazione con il controller, denominato **Controller**, del T23, il quale effettuerà le operazioni richieste interfacciandosi con il DB del T23, ovvero **MySql DB**.

Di seguito è riportato il flusso di comunicazione considerando il retrieve delle informazioni personali e lista seguiti (il flusso per le operazioni di modifica e ricerca è lo stesso, cambiano solamente i metodi coinvolti).

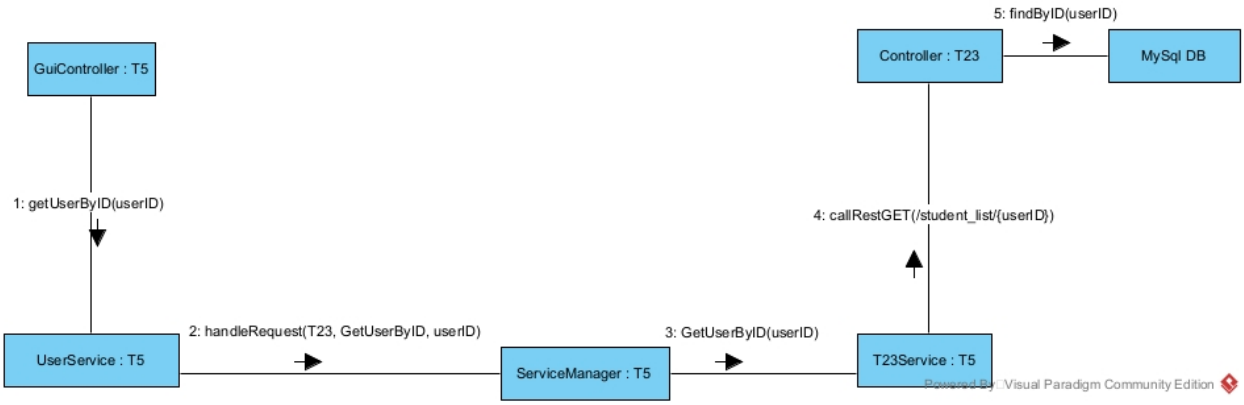


Figure 1.11 – Communication Diagram T23-T5 per il prelievo delle informazioni personali e lista seguiti di un player

Sono stati, quindi, modificati tutti i flussi di comunicazione per le operazioni tra T5 e T23.

3.7.4 Modifiche apportate

MICROSERVIZIO T23

- È stato modificato il diagramma ER del database per aggiungere la biografia, i token ricavati dalle missioni e la logica dei follow. Per realizzare quest'ultima è stata introdotta un'associazione ricorsiva tra gli studenti chiamata "follow", che presenta una cardinalità (**0, N**) sia per il **follower** che per il **followed**; ciò ovviamente definisce un'associazione molti-a-molti tra due studenti. Ovviamente il follower (identificato dal follower_id) è colui che segue il followed (identificato dal followed_id). La cardinalità è zero a molti in quanto uno studente può seguire nessuno o più studenti (lato follower) ed uno studente può essere seguito da nessuno o più studenti (lato followed).

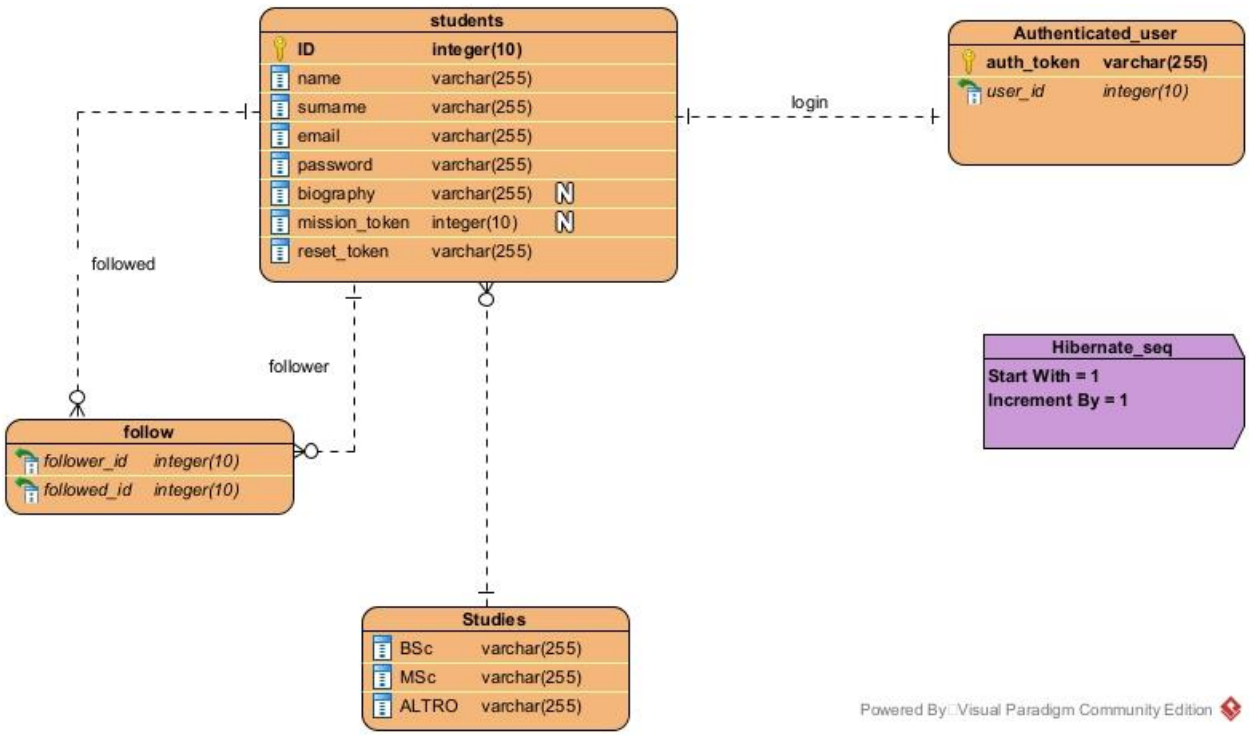


Figure 3.12 – Diagramma ER database T4

- Per coerenza con le modifiche fatte nel diagramma ER, è stato modificato anche il model User. In particolare, sono stati aggiunti quattro attributi:
    1. **String biography** per la biografia
    2. **Integer missionToken** per i token ottenuti dal giocatore
    3. **List<User> following** per memorizzare gli studenti seguiti da un player registrato.
    4. **List<User> followers** per memorizzare gli studenti che seguono un player registrato.

- Sono state introdotte all'interno del controller denominato "**Controller**", le seguenti endopoint:
    - **/modifyUser**
    - **/addFollow**
    - **/rmFollow**
    - **/searchPlayer,**

  le quali, come suggeriscono i nomi, permettono di implementare i casi d'uso descritti in precedenza. Queste verranno descritte in maniera dettagliata nel

paragrafo 3.8.

MICROSERVIZIO T5

All'interno del microservizio T5 è stata effettuata la modifica di alcuni componenti già esistenti, oltre che la creazione di nuovi componenti.

Sono state apportate modifiche ai seguenti componenti esistenti:

- **Profile.html** e **profile.js** per adattarli alle nuove funzionalità inserite e, per fare in modo che non vengano effettuate richieste dirette ad altri microservizi, bypassando il GuiController;
- **GuiController**, le quali sono indispensabili per rendere operative le funzionalità aggiunte. In particolare, sono state aggiunte le route API **profile/modifyUser**, **profile/addFollow**, **profile/rmFollow** e **profile/searchPlayer** (la cui spiegazione dettagliata viene riportata al paragrafo Implementazione);
- **Model User**, per allinearlo con quello predisposto dal T23 e, quindi, per mantenere consistenza tra i dati.
- In **AchivementService** è stato aggiunto il metodo **calculateRatiosForPlayer**, che serve a calcolare i Ratios tra partite vinte (**GamesWon**) e partite giocate (**GamesPlayed**) per un determinato giocatore, in base a specifici filtri come **gamemode** e **robot**.

I nuovi componenti aggiunti sono:

- **UserService,** utile nella comunicazione con il microservizio T23**;**
- **Profile_followed.html** (con lo script associato), poiché l'interfaccia della pagina profilo di un giocatore presente nella lista seguiti di un player è leggermente diversa da quella del player stesso, soprattutto nell'interazione.
- **Model Missioni**, le quali hanno un id, un nome, una descrizione e il numero di token restituiti in seguito al completamento.

A fronte di queste aggiunte e modifiche, si riporta in seguito il diagramma delle classi per il microservizio T5 contenente solamente ciò che è utile nella gestione del profilo.

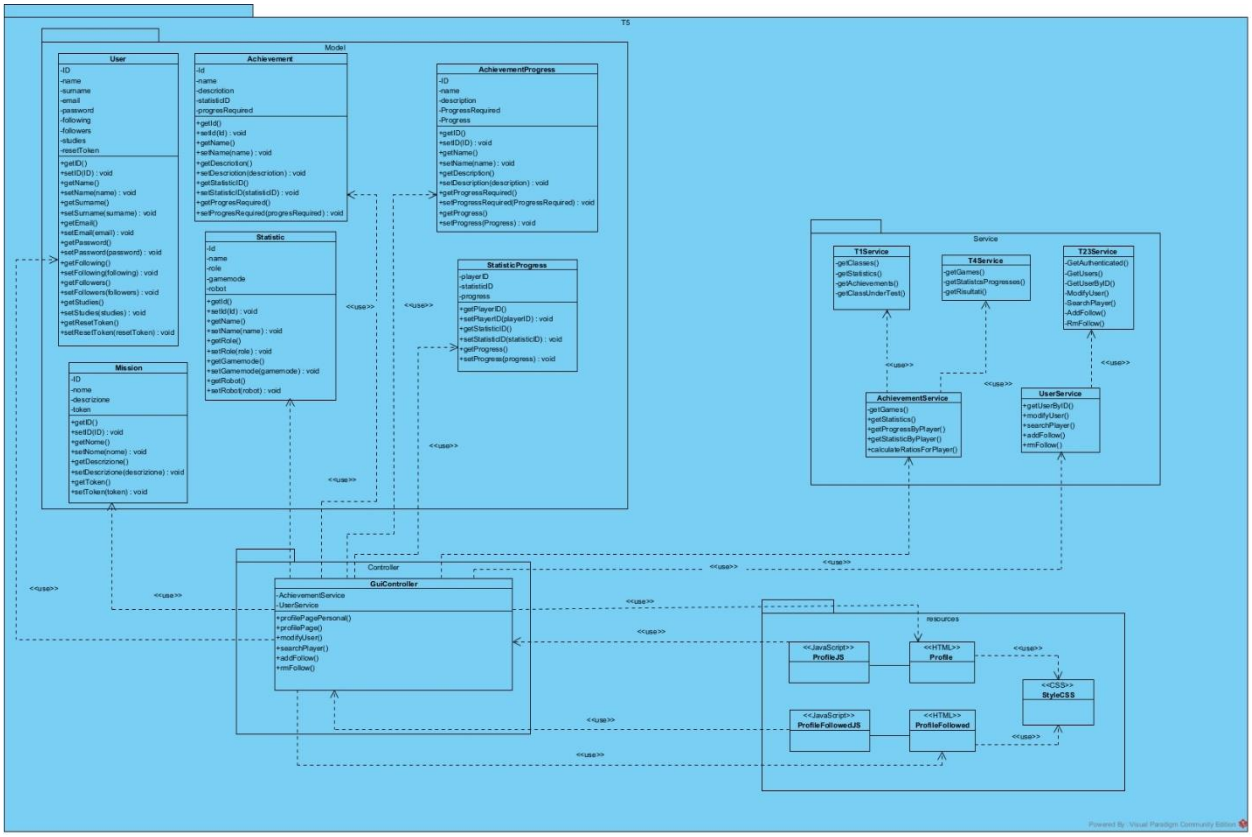


Figure 3.13 – Class Diagram T5

3.7.5 Architettura finale complessiva

È riportato a fini di chiarezza un **Package Diagram** con l'architettura complessiva della gestione del profilo giocatore.

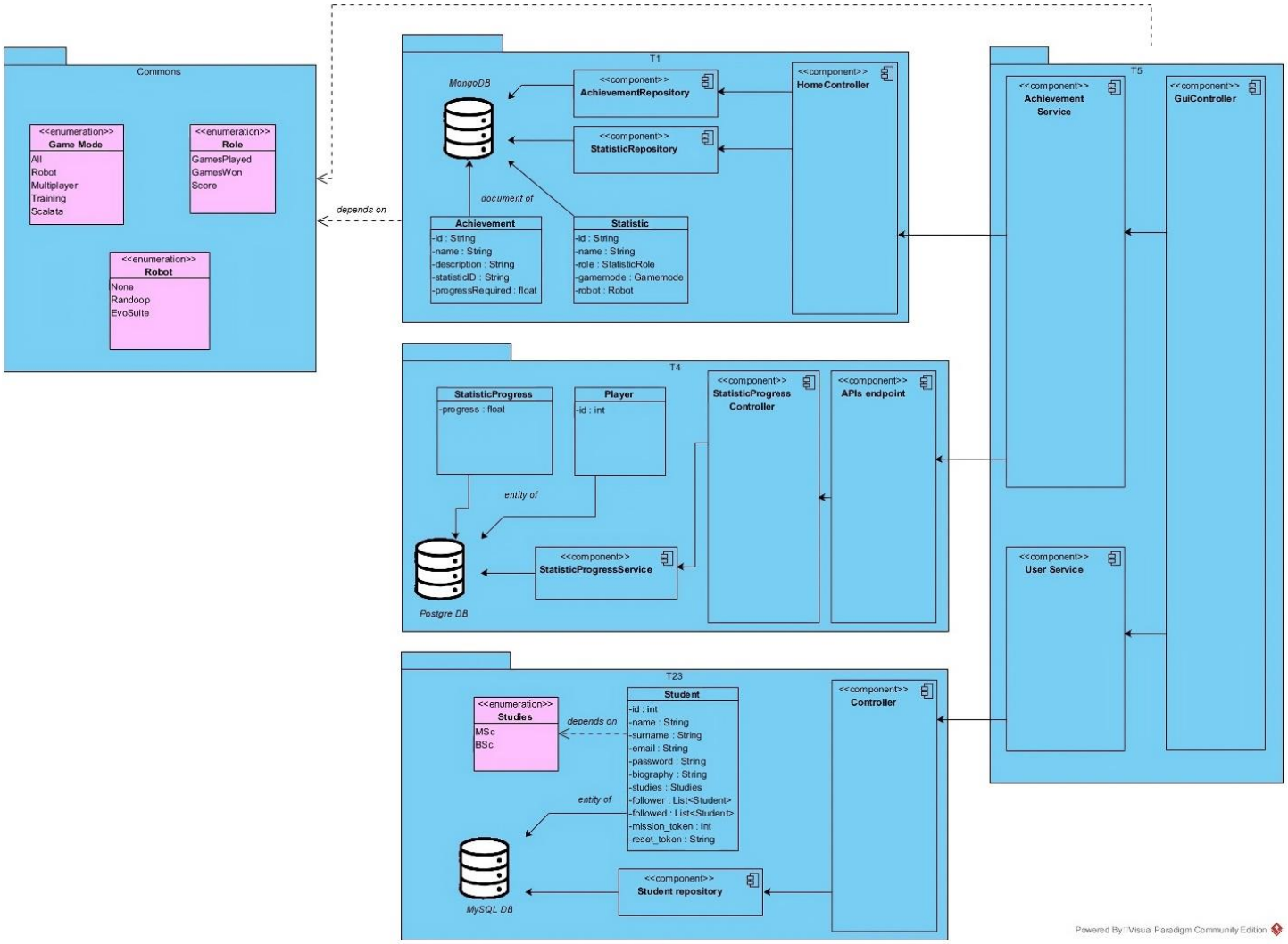


Figure 3.14 – Package Diagram

NB. nel diagramma dei package riportato non sono stati inseriti tutti i moduli Service che vengono coinvolti nella comunicazione per questioni di pulizia del diagramma.

3.7.6 Diagrammi di sequenza di Progetto

Nei seguenti diagrammi è riportato il flusso completo della comunicazione per ogni caso d'uso precedentemente descritto. Si nota che, per motivi di chiarezza e pulizia del diagramma, non è stata inserito il ServiceManager, il cui ruolo è stato descritto nel paragrafo 3.7.3.

Inoltre, si è anche supposto che tutte le precondizioni siano correttamente verificate, per cui non vengono riportati i controlli effettuati per verificare che il player sia correttamente autenticato (jwt) quando effettua le operazioni.

Diagramma di sequenza caso d'uso 'Visualizza profilo'

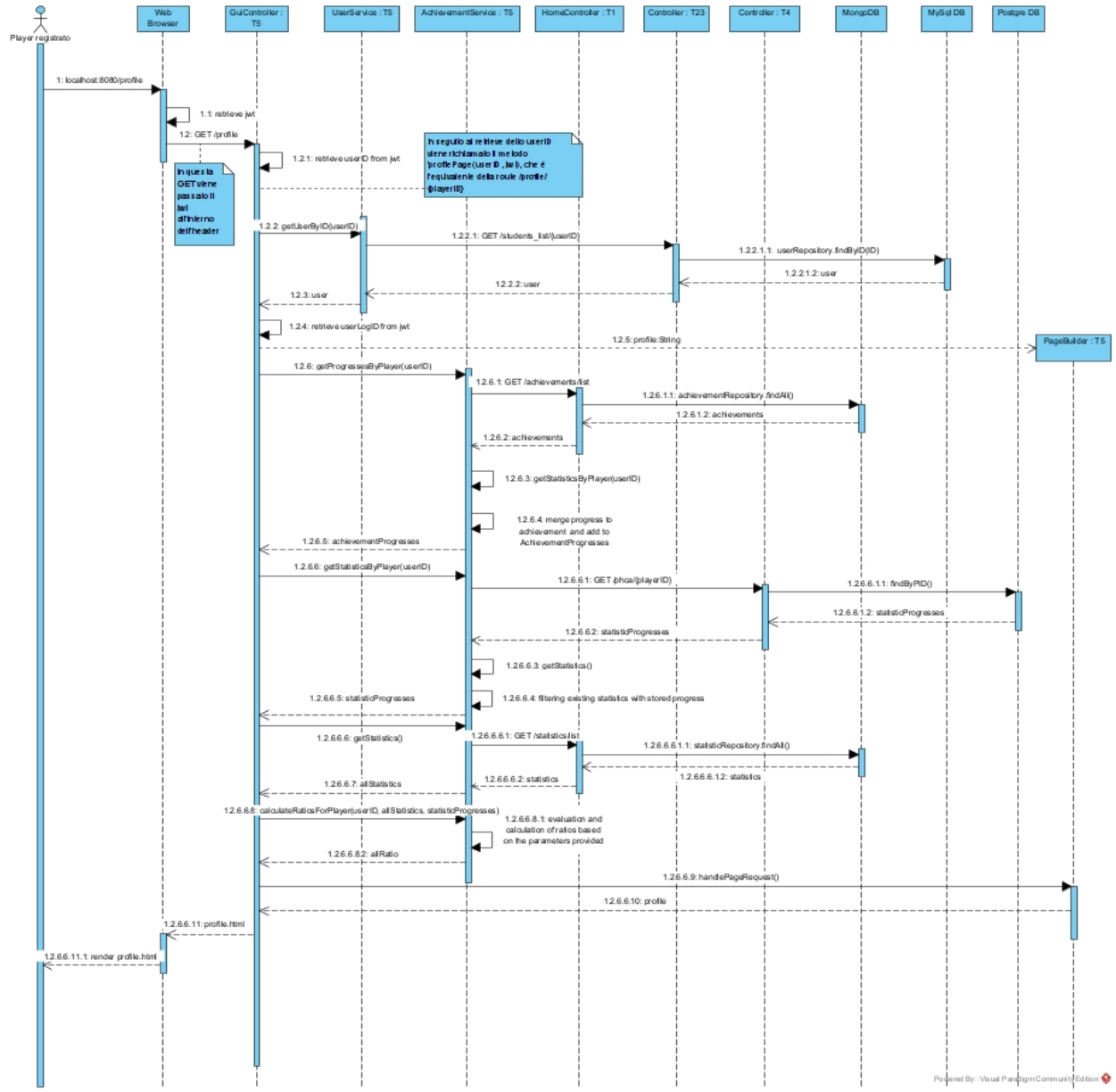


Figura 3.15 – Sequence diagram di progetto di 'Visualizza Profilo'

Essendo tale diagramma ricco di informazioni, si è deciso di non inserire i vari service T1Service, T23Service e T4Service, il cui ruolo è lo stesso che è stato descritto nel paragrafo 3.7.3.

In riferimento all'istanziazione del pageBuilder (1.2.5 del diagramma) per la creazione della pagina Profilo, viene effettuato un opportuno controllo a valle dell'operazione. Questo controllo consiste nel confrontare lo userID con userLogID (prelevato dal jwt). Se essi sono identici, si istanzia, quindi, un pageBuilder preposto alla creazione della pagina Profile.

Diagramma di sequenza caso d'uso 'Modifica informazioni personali'

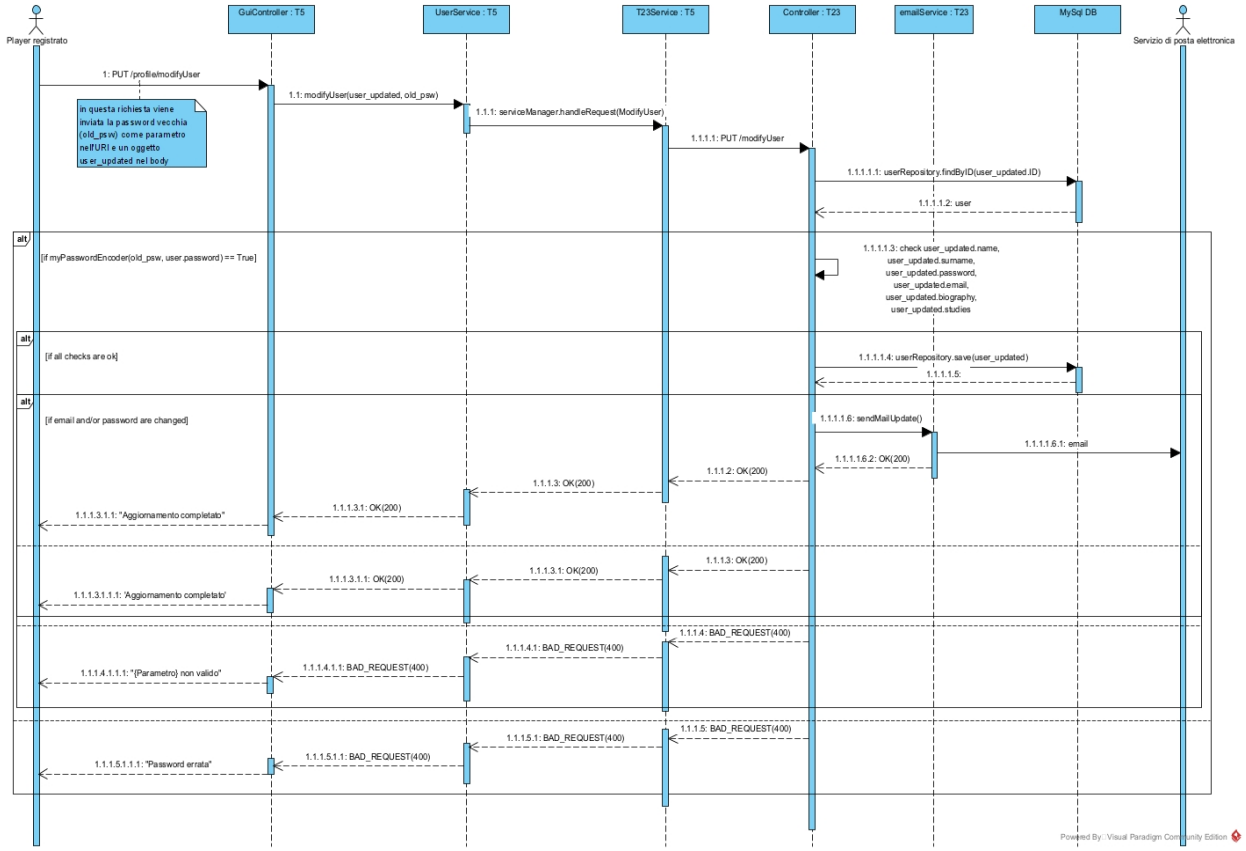


Figura 3.16 – Sequence diagram di progetto di 'Modifica informazioni profilo'

In questo diagramma si può notare come la lifeline Web Browser non sia stata inserita. È stata effettuata la scelta di inglobare la logica del suddetto in quanto non è responsabile di effettuare nessuna operazione comprendente controlli necessari al fine del completamento del caso d'uso.

Inoltre, non sono state riportati tutti i possibili scenari di errore nel controllo della correttezza dei dati forniti, ma semplicemente una bad request nel caso in cui un parametro generico sia sbagliato, ovviamente questi sono stati correttamente implementati nella loro completezza.

Diagramma di sequenza caso d'uso 'Ricerca player'

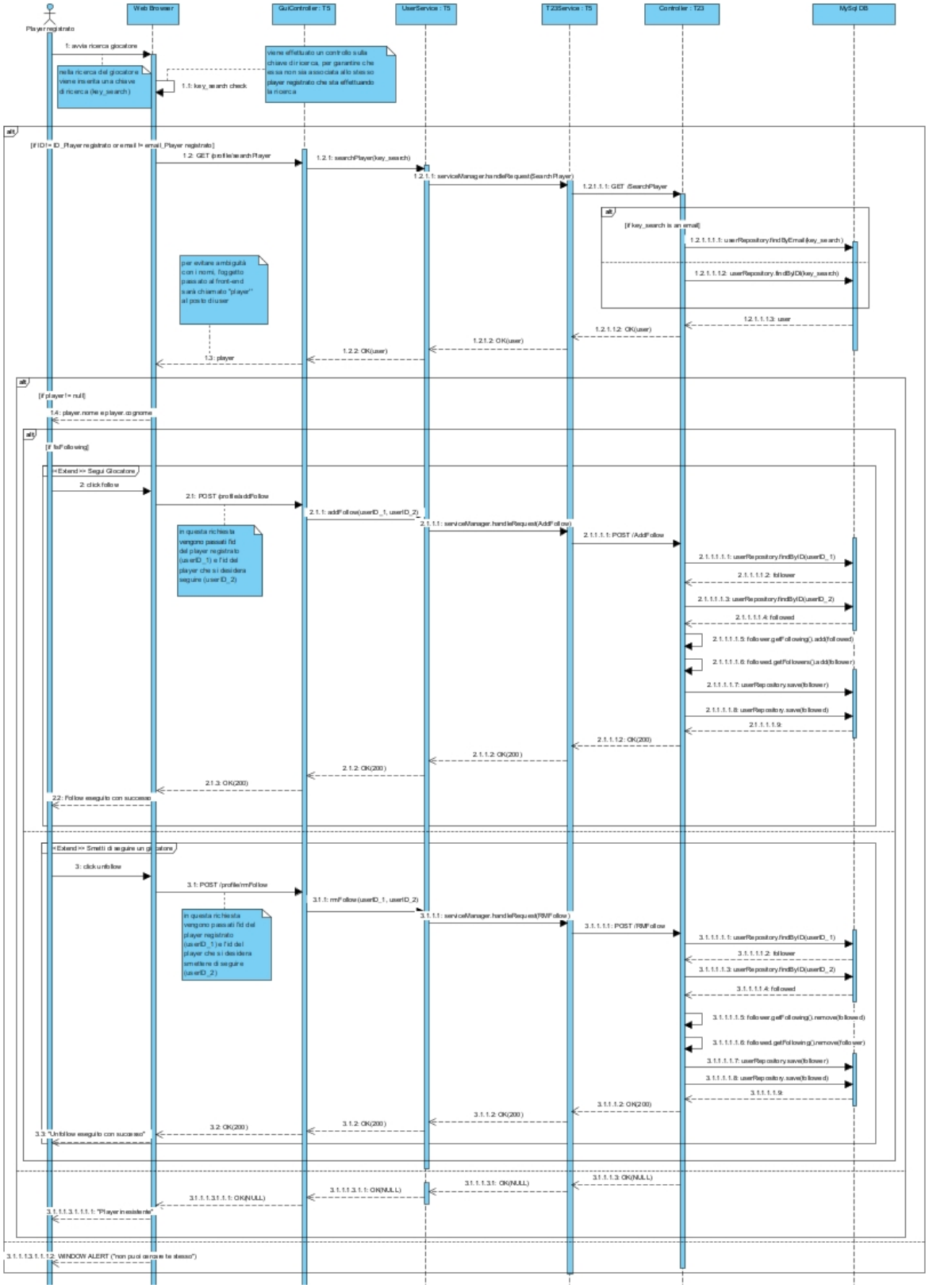


Figura 3.17 – Sequence diagram di progetto di 'Ricerca giocatore'

In merito al precedente sequence diagram, si specifica che nei casi di estensione di “Segui Giocatore” e “Smetti di seguire un Giocatore” non sono stati riportati tutti gli scenari alternativi di controllo per mantenere consistenza e persistenza dei dati nel DB, che sono stati invece correttamente implementati. Ciò perché in questo diagramma si sta dando per vero che ci siano tutte le condizioni per effettuare correttamente queste operazioni. Si riportano i diagrammi completi di “Segui Giocatore” e “Smetti di seguire un Giocatore” di seguito.

DIAGRAMMA DI SEQUENZA CASO D’USO ‘SEGUI UN GIOCATORE’

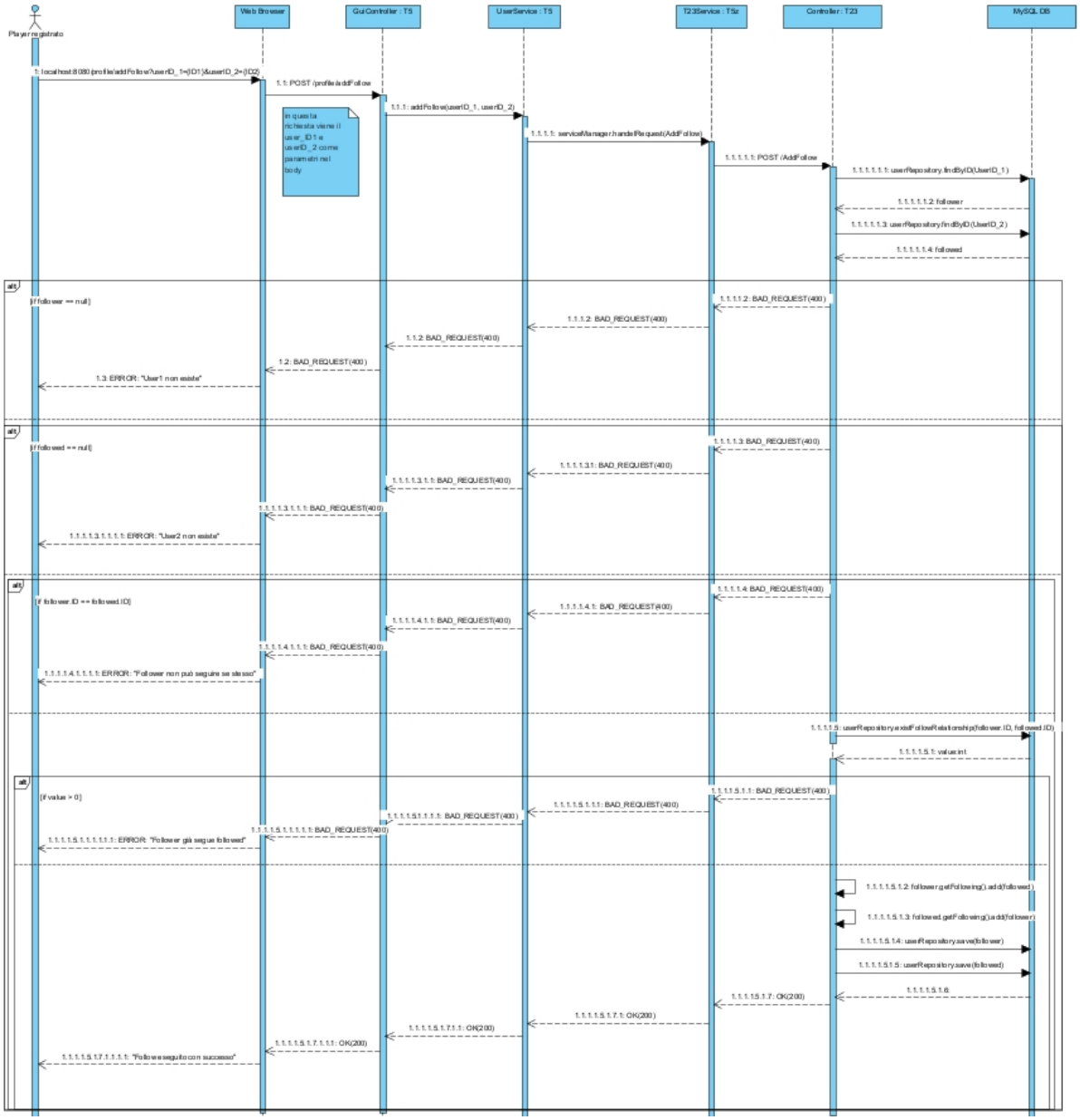


Figura 3.18 - Sequence Diagram completo del caso d’uso di estensione ‘smetti di seguire un giocatore’

DIAGRAMMA DI SEQUENZA CASO D'USO 'SMETTI DI SEGUIRE UN GIOCATORE'

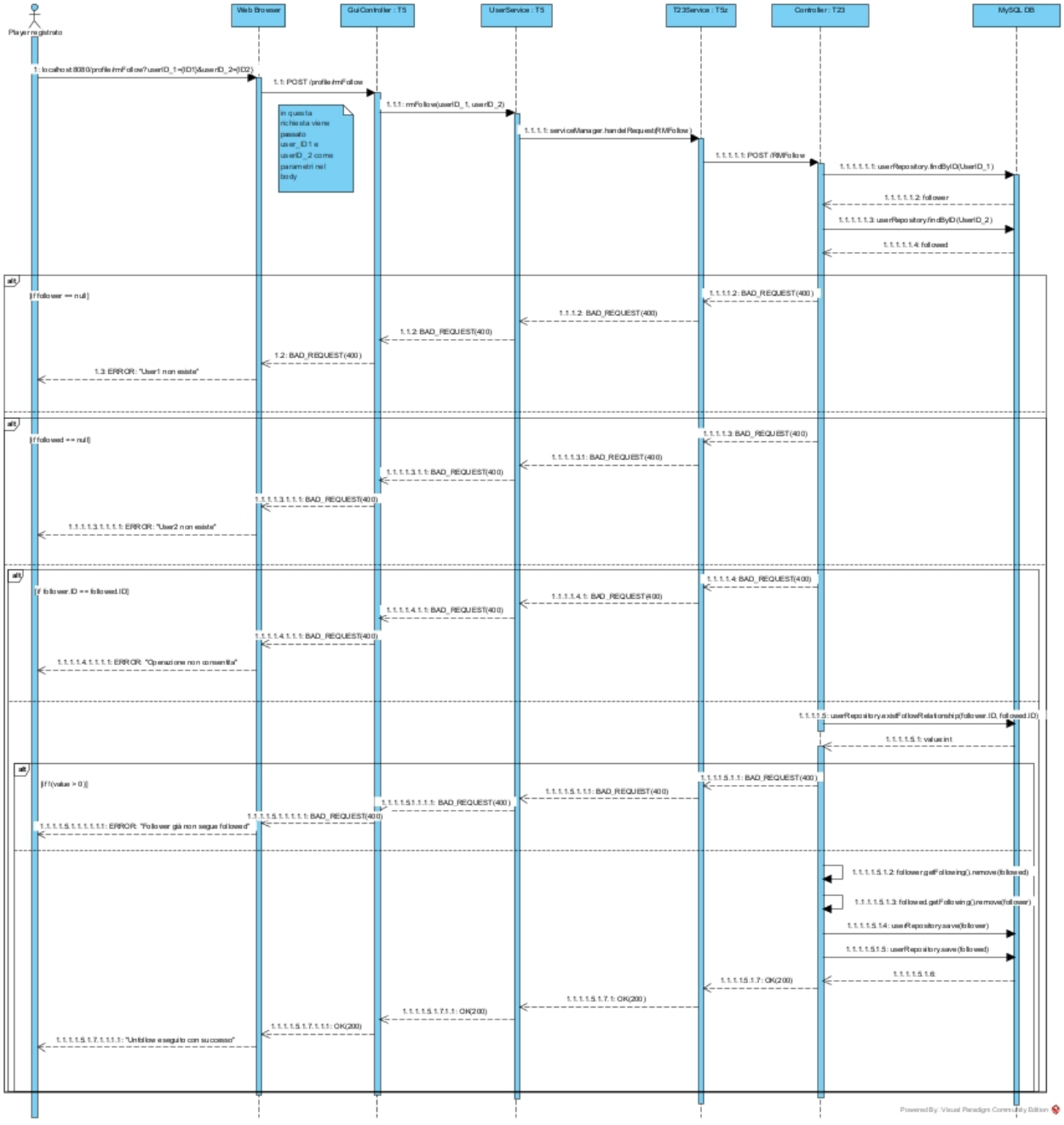


Figura 3.19 – Sequence Diagram completo del caso d'uso di estensione 'smetti di seguire un giocatore'

Diagramma di sequenza caso d'uso 'Visualizza profilo giocatore seguito'

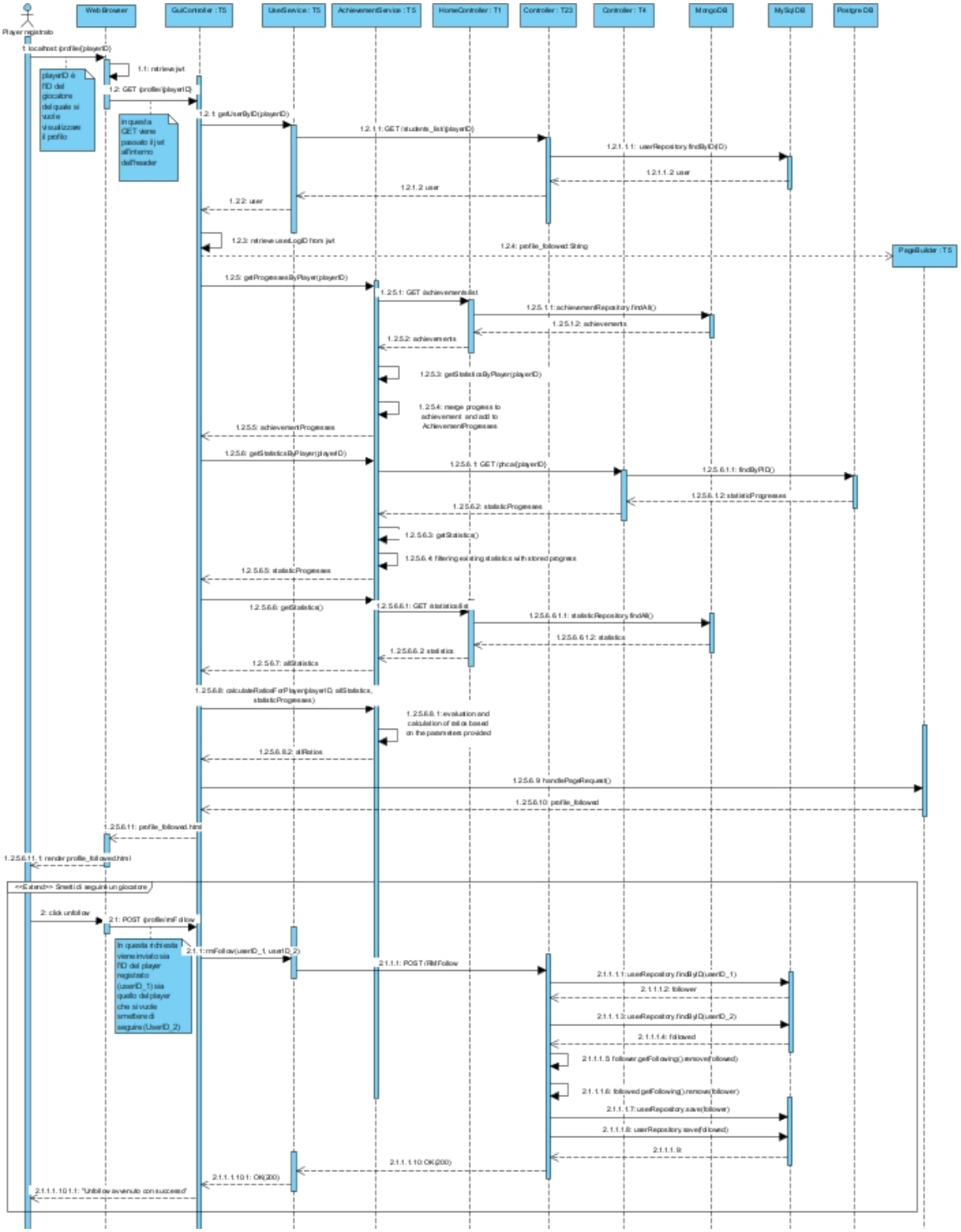


Figura 3.20 – Sequence diagram di progetto di 'Visualizza profilo giocatore seguito'

In riferimento all'istanziazione del pageBuilder (1.2.4 del diagramma) per la creazione della pagina Profilo_followed, viene effettuato un opportuno controllo a valle dell'operazione. Questo controllo consiste in primis nel prelevare l'oggetto user associato al playerID e vedere se all'interno della lista di Follower di user è contenuto lo userLogID (prelevato dal jwt). Supponendo vera questa condizione viene creato correttamente il pagebuilder di profile_followed.

3.8 Implementazione

Nel seguente paragrafo saranno illustrate nel dettaglio le modifiche o aggiunte implementative per i microservizi T23 e T5 che sono i due che abbiamo impattato.

3.8.1 Microservizio T23

Controller.java – Route API

/modifyUser					
HTTP method	Description	Operation ID	parameters	RequestBody	Response
PUT	Modifica dei campi di un utente, fornendogli un user_updated con dei campi non per forza tutti nuovi ma anche solo alcuni, e un old_psw per verificare che effettivamente sia il proprietario dell'account a modificare i dati.	modifyUser		required: true content: user_updated (User), old_psw (String)	200: "Agiornamento Completato" 400: "Utente non esiste" (ID user_updated non associato a nessun utente al DB) 400: "Password vecchia sbagliata" (old_psw non corrisponde con quella memorizzata al DB) 400: "Nome nuovo non valido" (Nome non rispetta le Validation Rules) 400: "Cognome nuovo non valido" (Cognome non rispetta le Validation Rules) 400: "Utente con questa email nuova già registrato" (Email nuova già associata ad un utente al DB) 400: "Email nuova non valida" (Email non rispetta la Validation Rules) 400: "Biografia nuova non valido" (Biografia non rispetta la Validation Rules) 400: "Password nuova non valida" (Password non rispetta la Validation Rules)

/searchPlayer					
HTTP method	Description	Operation ID	parameters	RequestBody	Response
GET	Ricerca un giocatore in base ad una chiave di ricerca (ID o e-mail)	searchPlayer		required: true content: key_search (String)	200: user (Oggetto di tipo User) 200: *null*
/addFollow					
HTTP method	Description	Operation ID	parameters	RequestBody	Response
POST	Permette di aggiungere alla lista dei following di user_1 lo user_2 e di aggiungere alla lista di followers di user_2 lo user_1.	addFollow		required: true content: userID_1 (String), userID_2 (String)	200: “Operazione completata con successo” 400: “User1 non esiste” 400: “User2 non esiste” 400: “follower non può seguire se stesso” 400: “follower già segue followed”
/rmFollow					
HTTP method	Description	Operation ID	parameters	RequestBody	Response
POST	Permette di rimuovere alla lista dei following di user_1 lo user_2 e di rimuovere alla lista di followers di user_2 lo user_1.	rmFollow		required: true content: userID_1 (String), userID_2 (String)	200: “Operazione completata con successo” 400: “User1 non esiste” 400: “User2 non esiste” 400: “follower già non segue followed”

Tabella 3.7 – Route API T23

- In merito alla Route di “/modifyUser”, è possibile cambiare la password dell’utente anche se era già esistenti le route /password_reset e /passpassword_change, ma a nostra interpretazione esse erano predisposte per il reset della password in uno scenario di dimenticanza.
- La route “/searchPlayer” ci restituirà un oggetto di tipo User, in base al criterio fornito. Nel caso in cui non fosse trovato nulla viene restituito *null*, il quale corrisponderà, in termini di HTTP, ad un JSON vuoto.

Model – User.java

Nel model User.java presente nel microservizio T23 è stato opportunatamente modificato per soddisfare le esigenze del task assegnatoci. In particolare, abbiamo aggiunto i campi: biography (String), missionToken (Integer), following (List<User>) e followers (List<User>). Spendiamo qualche parola per gli ultimi due:

```
// Relazione per la logica di "Follow"
@ManyToMany
@JoinTable(
    name = "Follows",
    schema = "studentsrepo",
    joinColumns = @JoinColumn(name = "follower_id"),
    inverseJoinColumns = @JoinColumn(name = "followed_id")
)
@JsonIgnoreProperties({"followers", "following"})
public List<User> following;

@ManyToMany(mappedBy = "following")
@JsonIgnoreProperties({"followers", "following"})
public List<User> followers;
```

Figura 3.21 – Aggiunte per la logica dei follower in model User.java

Evidenziamo le seguenti particolarità:

- **@ManyToMany** serve per definire una relazione molti a molti tra due entità di Utenti.
- "**following**" rappresenta la lista di utenti che un utente sta seguendo. "**followers**" rappresenta la lista di utenti che stanno seguendo un utente.
- **@JoinTable** serve ad indicare come mappare correttamente la relazione molti a molti, in particolare su una tabella chiamata "Follows" e su uno schema chiamato "studentsrepo"
- L'attributo **joinColumns** indica la colonna della tabella di giunzione che fa riferimento al lato corrente della relazione (follower_id rappresenta l'utente che segue un altro),
- **inverseJoinColumns** è la colonna della tabella di giunzione che fa riferimento al lato opposto della relazione (followed_id rappresenta l'utente seguito).
- Sul campo followers, è presente l'attributo **mappedBy** , uguale a "following" , che specifica che questa lista è il **lato inverso** della relazione, la relazione è già stata definita in following e Hibernate non creerà un'altra tabella per questo campo.
- **@JsonIgnoreProperties** serve ad ignorare i parametri di "following" e "followers" in modo tale che non vengano generati dei JSON ciclicamente infiniti.

UserRepository.java

All'interno di questa classe, è stato aggiunto il metodo ***existsFollowRelationship***, che implementa la query:

```
@Query(value = "SELECT COUNT(*) > 0 FROM studentsrepo.follows WHERE follower_id = :followerId AND followed_id = :followedId", nativeQuery = true)
int existsFollowRelationship(@Param("followerId") Integer followerId, @Param("followedId") Integer followedId);
```

Figura 3.22 – metodo existsFollowRelationship

per controllare se un "follower_Id "segue un "followed_id".
Il quale viene usato per controllare se esiste già una relazione di follow tra due utenti, in modo tale che un utente non risegua nuovamente uno stesso Utente. La medesima funzione è usata per la rimozione di un follow per controllare lo speculare, cioè, se una persona che già non segue un'utente prova a defollowarlo.

REGEX

Affinché si filtrino correttamente i dati forniti in ingresso rispetto a quanto detto nel paragrafo 3.4, sono state cambiate e aggiunte le regex per soddisfare tali requisiti.
In particolare, per il **nome e il cognome** si usa la seguente struttura:

^[A-Za-zÀ-ÿ][a-zà-ÿ]*(?:'?[A-Za-zÀ-ÿa-zà-ÿ]+)*(?:\\s[A-Za-zÀ-ÿa-zà-ÿ]+(?:'?[A-Za-zÀ-ÿa-zà-ÿ]+)*)*$

Ciò significa che il **primo carattere** della stringa deve essere una lettera latina o accentata (maiuscola o minuscola) {*[A-Za-zÀ-ÿ]*}, **dopo** il primo carattere, possono esserci altre lettere minuscole (accentate o no) {*[a-zà-ÿ]**}, può essere seguita da una o più sezioni che gestiscono nomi composti da almeno una lettera (latina, maiuscola, minuscola, o accentata), o parole che contengono apostrofi {*(?:'?[A-Za-zÀ-ÿa-zà-ÿ]+)**}, infine l'ultima sezione serve a gestire i nomi composti dagli spazi con la possibilità che si possano ripetere più di una volta {*(?:\\s[A-Za-zÀ-ÿa-zà-ÿ]+(?:'?[A-Za-zÀ-ÿa-zà-ÿ]+)*)**}

Per la biografia invece:

^[A-Za-zÀ-ÿ0-9\\s.,'!?-]+$

Questa regex verifica che la stringa **sia composta esclusivamente** dai seguenti caratteri: lettere maiuscole e minuscole (A-Z, a-z), comprese le lettere accentate (À-ÿ), cifre numeriche (0-9), spazi, il punto, la virgola, l'accento, il punto esclamativo, il punto interrogativo e il trattino. Inoltre, la stringa deve contenere almeno **un carattere**.

Le regex del nome e cognome sono state sostituite anche nella route in merito alla registrazione dell'utente.

EmailService.java

Sono stati aggiunti al già esistente servizio i metodi: "*sendMailUpdate*" e "*sendMailPassword*" per l'invio di una mail per la rispettiva conferma del cambio E-mail e Password.

3.8.2 Microservizio T5

Problema dell'update dei Progressi delle Statistiche – GameController.java

Si specifica fin da subito che per questa correzione si deve ringraziare il gruppo di: **Vincenzo Luigi Bruno, Salvatore Cangiano e Cristina Carleo** per la correzione di questa porzione di codice.
Il controller "GameController.java" ha il compito di: gestire le **partite attive** in tempo reale; offrire diverse route API per interagire con il sistema; integrare con servizi esterni il calcolo della coverage, punteggi robot, e altro tramite microservizi. In particolare, uno dei problemi individuati riguarda la route /StartGame dove vi era commentata la riga contenente:

gameLogic.CreateGame();

La quale permette di interagire con il microservizio T4 per istanziare una partita. Il secondo passo è stato integrare all'interno del metodo ***gestisciPartita*** l'effettivo l'update dei progressi delle statistiche che non veniva fatto in precedenza. La porzione di codice aggiunto è evidenziata in rosso:

```
// Controllo fine partita
if (isGameEnd || gameLogic.isGameEnd()) {
    activeGames.remove(playerId);
    logger.info(msg:"[GAMECONTROLLER] /run: risposta inviata con GameEnd true");

    List<User> users = (List<User>) serviceManager.handleRequest(serviceName:"T23", action:"GetUsers");
    Integer userId = Integer.parseInt(playerId);
    User user = users.stream().filter(u -> u.getId() == userId).findFirst().orElse(other:null);
    List<AchievementProgress> newAchievements = achievementService.updateProgressByPlayer(userId.intValue());

    return createResponseRun(userData, robotScore, userScore, gameOver:true, lineCoverage, branchCoverage, instructionCoverage);
} else {
    logger.info(msg:"[GAMECONTROLLER] /run: risposta inviata con GameEnd false");
    return createResponseRun(userData, robotScore, userScore, gameOver:false, lineCoverage, branchCoverage, instructionCoverage);
}
```

Figura 3.23 – Codice 'updateProgressi' aggiunto al GameController

Si specifica che anche con l'integrazione del seguente codice, non vengono correttamente aggiornati i progressi delle statistiche del player all'interno della tabella *phca*, in quanto nel T4 non è stato correttamente predisposto il filtraggio per i Robot, nello specifico non esiste un'associazione nel DB Postgre tra robot e partita. In conclusione, con un esempio concreto se si vincesse una partita in modalità Sfida, verrebbero incrementati i progressi delle statistiche che hanno la categoria Partite Vinte senza fare differenze tra Robot.
Per apprezzare maggiormente la visione dei progressi delle statistiche sull'interfaccia del profilo, abbiamo interagito materialmente mediante l'uso di PgAdmin per creare dei progressi fittizi.

GuiController.java – API Route

/profile/modifyUser					
HTTP method	Description	OperationID	parameters	RequestBody	Response
PUT	Modifica dei campi di un utente, fornendogli un user_updated con dei campi non obbligatoriamente tutti aggiornati e un old_psw per verificare che effettivamente sia il proprietario dell'account a modificare i dati.	modifyUser		required: true content: user_updated (User), old_psw (String), jwt (String)	200: *result* (Risultato della chiamata alla omonima route del T23) 400: "Utente non loggato" (jwt scaduto 400: *e.getCause().getMessage()* (Risultato di Bad Request della chiamata alla omonima route del T23) 500: "Internal Server Error"
/profile/searchPlayer					
HTTP method	Description	OperationID	parameters	RequestBody	Response
GET	Ricerca un giocatore in base ad una chiave di ricerca (ID o e-mail)	searchPlayer		required: true content: key_search (String), jwt (String)	200: *result* (Risultato di tipo User della chiamata alla omonima route del T23) 400: *null* (jwt scaduto) 400: *null* (Risultato di Bad Request della chiamata alla omonima route del T23) 500: null

/profile/addFollow					
HTTP method	Description	OperationID	parameters	RequestBody	Response
POST	Permette di aggiungere alla lista dei following di user_1 lo user_2 e di aggiungere alla lista di followers di user_2 lo user_1.	addFollow		required: true content: userID_1 (String), userID_2 (String), jwt (String)	200: *result* (Risultato di tipo String della chiamata alla omonima route del T23) 400: "Utente non loggato" (jwt scaduto) 400: *e.getCause().getMessage()* (Risultato di Bad Request della chiamata alla omonima route del T23) 500: "Internal Server Error"
/profile/rmFollow					
HTTP method	Description	OperationID	parameters	RequestBody	Response
POST	Permette di rimuovere alla lista dei following di user_1 lo user_2 e di rimuovere alla lista di followers di user_2 lo user_1.	rmFollow		required: true content: userID_1 (String), userID_2 (String), jwt (String)	200: *result* (Risultato di tipo String della chiamata alla omonima route del T23) 400: "Utente non loggato" (jwt scaduto) 400: *e.getCause().getMessage()* (Risultato di Bad Request della chiamata alla omonima route del T23) 500: "Internal Server Error"

Tabella 3.8 – Route API T5

Le seguenti endpoint sono utilizzate da profile.js affinché il Web Browser comunichi correttamente con il T5, il microservizio che appunto predispone e compone la pagina web, con i medesimi servizi implementati nel T23. Esse sono state implementate per seguire il corretto flusso operativo, come definito nei diagrammi di sequenza. Infine in ognuna di esse viene effettuate un controllo per verificare se il jwt è scaduto.

/profile e /profile/{playerID}

Le route di **/profile e /profile/{playerID}** sono state opportunatamente modificate, aggiungendo il controllo per verificare se il jwt sia scaduto o meno. Riguardo /profile/{playerID}, è stato implementato un meccanismo per verificare chi sta "usando" questa rotta. In altre parole, se l'utente loggato sta cercando di accedere al proprio profilo (avente come playerID l'ID dell'utente stesso) allora mostrerà la pagina personale. Se si inserisce l'ID di un player che non è il proprio, la route controlla se lo si segue e mostrerà la

pagina della persona seguita, altrimenti riporterà l'utente al main. Questi controlli vengono fatti decriptando il jwt che contiene le informazioni riguardo allo userID attualmente loggato. La route di /profile/{playerID} già presente è stata usata anche per far visualizzare la pagina di un utente seguito.

```
PageBuilder profile = null;

int userId = Integer.parseInt(playerID);
User user = userService.getUserbyID(userId);

byte[] decodedUserObj = Base64.getDecoder().decode(jwt.split(regex:"\\.")[1]);
String decodedUserJson = new String(decodedUserObj, StandardCharsets.UTF_8);

try {
    ObjectMapper mapper = new ObjectMapper();
    @SuppressWarnings("unchecked")
    Map<String, Object> map = mapper.readValue(decodedUserJson, valueType:Map.class);
    String jwt_userId = map.get(key:"userId").toString();

    if(jwt_userId.equals(playerID)){
        profile = new PageBuilder(serviceManager, PageName:"profile", model);
    }else if(userService.isUserInFollower(user,Integer.parseInt(jwt_userId))){
        profile = new PageBuilder(serviceManager, PageName:"profile_followed", model);
    }else{
        return "redirect:/main";
    }
}
catch (Exception e) {
    System.out.println("(/profile) Error requesting profile: " + e.getMessage());
}

profile.SetAuth(jwt);
```

Figura 3.24 – codice selezione pageBuilder corretto

UserService.java e T23Service.java

La prima classe è stata creata e aggiunta al package di Service, affinché possa gestire correttamente tutte le comunicazioni con il T23, mettendo a disposizione del T5 i servizi del medesimo, usando la già esistente classe T23Service. In particolare, sono stati inseriti in UserService i metodi:

- getAuthenticated(String jwt)
- getUserbyID(int user_id)
- modifyUser(User user_updated, String old_psw)

- searchPlayer(String key_search)
- addFollow(String userID_1, String userID_2)
- rmFollow(String userID_1, String userID_2)
- isUserInFollower(User user, Integer targetUserId)

Tutti questi metodi vengono usati all'interno del "GuiController.java" per la corretta implementazione delle route descritte prima. Il metodo "***isUserInFollower***" non comunica con il T23 ma semplicemente controlla che un certo ID sia contenuto nella lista Follower di uno User.

"T23Service.java" era già presente ed è un'estensione di "BaseService" (una classe base che implementa l'interfaccia *ServiceInterface* per il dispatcher *ServiceManager* e fornisce svariati metodi che mappano POST, GET, PUT E DELETE, in vari formati e header), dove sono stati registrati i servizi: ***GetAuthenticated***, ***ModifyUser***, ***SearchPlayer***, ***AddFollow*** e ***RmFollow***.

AchivementeService.java

È stato aggiunto il metodo:

calculateRatiosForPlayer (Integer playerID, List<Statistic> statistics, List<StatisticProgress> progresses)

che permette di calcolare, quando sono presenti per una certa Gamemode ed un certo Robot, le statistiche aventi un certo Role, ossia GamesPlayed e GamesWon. Il Ratio associato, restituisce una List<Ratio> in base a tutte le possibili combinazioni.
Per questa classe è stato corretto anche il metodo **getStatisticsByPlayer**, che non caricava correttamente il playerID allo StatisticProgress generato.

Model

Il model nella versione di partenza A13 utilizzava Long come tipo di id.

```
public class User {

    private Long id;
    private String name;
    private String surname;
    private String email;
    private String password;
    private boolean isRegisteredWithFacebook;
    private String studies;
    private String resetToken;
```

Figura 3.25 – model User.java in documentazione A13

È stato modificato il model di User.java sulla base delle aggiunte del corrispettivo User.java sul T23, il risultato finale è il seguente:

```
public class User {

    private Integer ID;
    private String name;
    private String surname;
    private String email;
    private String password;
    private String biography;
    private List<User> following;
    private List<User> followers;
    private boolean isRegisteredWithFacebook;
    private boolean isRegisteredWithGoogle;
    private String studies;
    private String resetToken;
    private Integer missionToken;
```

Figura 3.26 – model User.java utilizzato nella nostra versione

Inoltre, sono stati predisposti i model:

- **Mission.java**: è l'implementazione del concetto di Missione, il quale viene predisposto anche sulla pagina del profilo con degli esempi statici per mostrarne il corretto funzionamento. I campi del seguente model sono: ***Integer ID; String name; String description e Integer numToken***. Ovvero, una chiave primaria che identifica quella missione, un nome, una descrizione e il numero di Token che possono essere vinti se completata.

```
public class Mission {

    private Integer ID;
    private String name;
    private String description;
    private Integer numToken;

    public Mission(Integer ID, String name, String description, Integer numToken) {
        this.ID = ID;
        this.name = name;
        this.description = description;
        this.numToken = numToken;
    }
```

Figura 3.27 – model Missione

- **Ratio.java:** è stata introdotta per memorizzare correttamente i Ratio calcolati dal metodo precedente per fornirli agilmente al PageBuilder di profile, in particolare ha come campi: ***Integer playerID; Gamemode gamemode; Robot robot e Float value***. Ovvero, contiene l'id del giocatore, una specifica modalità di gioco, modalità di robot ed il valore di ratio calcolato.

```
public class Ratio {

    private Integer playerID;
    private Gamemode gamemode;
    private Robot robot;
    private Float value;


    public Ratio(Integer playerID, Gamemode gamemode, Robot robot, float value) {
        this.playerID = playerID;
        this.gamemode = gamemode;
        this.robot = robot;
        this.value = value;
    }
```

Figura 3.28 – model Ratio

Front-end

Affinché il Web Browser comunichi correttamente con il T5, è stato sviluppato opportunamente un file javascript che usa i framework citati in precedenza. Esso è chiamato profile.js ed è contenuto in resources/static/t5/js. All'interno di esso è stato aggiunto il seguente elenco di metodi al di fuori dei primi due:

- **parseJwt**: Decodifica un JSON Web Token (JWT) e restituisce il payload come oggetto JavaScript. Divide il token in base al separatore, e recupera la seconda parte (payload). Decodifica la parte in Base64 utilizzando ***atob*** e la converte in oggetto JSON. Se la decodifica fallisce, restituisce null.

- **getCookie**: Recupera il valore di un cookie dato il suo nome. Aggiunge un prefisso ";" a tutti i cookie concatenati per facilitare la ricerca. Divide la stringa dei cookie per name per isolare il valore desiderato. Se trova il valore, lo restituisce (senza il punto e virgola finale), altrimenti restituisce null.

- **populateForm**: Popola un modulo HTML con i valori di un oggetto globale userUpdate. Recupera i campi del modulo tramite getElementById e assegna loro i valori corrispondenti di userUpdate (o una stringa vuota se assente).

- **updateUserObject**: Aggiorna l'oggetto userUpdate con i valori correnti del modulo. Itera su una mappa di nomi dei campi e ID degli input. Aggiorna userUpdate solo se l'input non è vuoto.

- **viewCompletedAchievements**: Mostra solo gli achievement completati. Seleziona tutti gli elementi con classe "*.achievement-container*". Controlla l'attributo data-completed e mostra o nasconde gli elementi in base al valore (true o false).

- **viewAllAchievements**: Mostra tutti gli achievement. Rimuove lo stile di display nascosto da ogni elemento "*.achievement-container*".

- **toggleAchievements**: Alterna la visualizzazione tra achievement completati e tutti gli achievement. Cambia il testo del bottone e chiama le funzioni viewAllAchievements() o viewCompletedAchievements().

- **saveChanges**: è progettato per inviare una richiesta al server per salvare le modifiche apportate a un profilo utente. Questo metodo è invocato per aggiornare un oggetto userUpdate con i dati più recenti inseriti dall'utente nel modulo. Popola lo userUpdate con i campi del modulo. L'utente deve fornire la password attuale (passwordoldInput) come verifica di autenticazione. Se non viene inserita, viene mostrato un messaggio di avviso e il processo si interrompe. Il metodo controlla che l'utente sia autenticato recuperando un token JWT dai cookie tramite la funzione getCookie. Se il token è assente, viene mostrato un avviso e il metodo termina.

- **search**: è progettato per cercare un giocatore utilizzando una chiave di ricerca fornita dall'utente. Il metodo verifica che l'utente sia autenticato controllando l'esistenza del token JWT (getCookie('jwt')). Se il token non è presente, viene mostrato un messaggio di avviso e il metodo si interrompe. L'utente deve inserire una chiave di ricerca (key_search) nel modulo. Se il campo è vuoto, viene mostrato un messaggio di avviso e il metodo si ferma. Il metodo verifica che la chiave di ricerca non corrisponda all'ID o all'email dell'utente autenticato. Questo evita che l'utente possa cercare sé stesso. La codifica della chiave di ricerca con encodeURIComponent previene possibili problemi legati a caratteri speciali nell'URL.

- **renderSearchResults**: è responsabile di visualizzare i risultati della ricerca di un giocatore nel DOM. Il metodo seleziona un elemento HTML con l'ID searchResults, che funge da contenitore per i risultati della ricerca. All'inizio, svuota i contenuti per evitare che i risultati di ricerche precedenti rimangano visibili. Se il parametro player è nullo o non definito, viene aggiunto un messaggio al contenitore per indicare che non sono stati trovati giocatori. Il metodo si interrompe qui. La funzione isUserFollowing(player) verifica se l'utente corrente segue già il giocatore trovato. In base al risultato, il testo del pulsante è impostato su Follow (se non lo segue) o Unfollow (se lo segue).

- **isUserFollowing**: Controlla se l'utente corrente segue un altro giocatore. Usa Array.some() per verificare se l'ID del giocatore è nella lista user.following

- **toggleFollow**: Aggiunge o rimuove un follow. Determina l'azione (addFollow o rmFollow) in base allo stato corrente. Effettua una chiamata AJAX POST al server. Aggiorna il modello user manualmente e aggiorna i risultati della ricerca.

- **selectRobotFilter** e **selectGameModeFilter**: Filtrano le statistiche per robot o modalità di gioco selezionati.
- **filterStatistics**: è progettato per filtrare e mostrare o nascondere righe di statistiche basandosi su criteri specifici, come il robot selezionato e la modalità di gioco. Tutte le righe delle statistiche, identificate dalla classe CSS statistic-row, vengono selezionate usando querySelectorAll. Questo genera una *NodeList* contenente gli elementi da filtrare. Per ogni riga, il valore dell'attributo data-robot viene recuperato e memorizzato nella variabile robot. Il valore dell'attributo data-gamemode viene recuperato e memorizzato nella variabile gamemode. Questi attributi sono definiti nel markup HTML e rappresentano i criteri di filtraggio. Se entrambe le condizioni (robotMatch e gameModeMatch) sono soddisfatte, la riga viene mostrata impostando row.style.display a una stringa vuota (''), che ripristina il valore predefinito. In caso contrario, la riga viene nascosta impostando row.style.display su 'none'.

- **updateRatioVisibility**: è progettato per gestire la visibilità di un contenitore HTML che mostra le ratio, basandosi sui filtri attualmente selezionati e sulla disponibilità di dati validi. Se i filtri globali currentRobot o currentGameMode non sono impostati (sono null), il contenitore della ratio (#ratio-container) viene nascosto e il metodo si interrompe. Si iterano tutte le righe di statistiche visibili (quelle non nascoste dal filtro precedente). Per ogni riga visibile viene controllato l'attributo data-role della riga. Se il ruolo è GamesPlayed, viene impostato hasGamesPlayed = true. Se il ruolo è GamesWon, viene impostato hasGamesWon = true. Lo scopo è determinare se sono presenti statistiche filtrate per partite giocate e partite vinte, prerequisiti per calcolare la ratio. Se entrambi i tipi di statistiche sono disponibili procede con l'elaborazione delle righe. In caso contrario, nasconde il contenitore della ratio.

- **clearFilter**: Resetta i filtri e mostra tutte le statistiche.

Inoltre, è stato modificato la pagina profile.html soddisfando il task richiesto, e sono stati creati i file main3.css (resources/static/t5/css) e profile_followed.html (resources/static/t5/templates). Il primo include lo stile usato per la pagina di profile.html, il secondo serve per predisporre correttamente al web browser la pagina dell'utente seguito (/profile/{playerID}). Di questo si specifica che nel medesimo codice HTML è stato incluso sia la parte del css che del js (mediante i tag di style e script), in quanto si è vinto che in fase di testing che qualche meccanismo per il prelievo dei medesimi non funziona correttamente. In sintesi, ispezionando con gli strumenti del motore di ricerca che nel

nostro caso è stato Chrome si nota che i file di .css e .js non sono caricati a differenza di quello che succede nella richiesta della pagina di /profile in cui vengono prelevati correttamente. Per le analisi fatte non siamo riusciti a trovare un vero discriminante per capire il problema, ma forse ci sarebbe da investigare sul funzionamento del PageBuilder.

3.9 Testing

Si riportano i seguenti screen che mostrano le chiamate HTTP usate, tramite **Postman**, per testare il corretto funzionamento delle route implementate nel microservizio **T23**:

- AddFollow
  200: OK L'utente 4 ha seguito l'utente 5

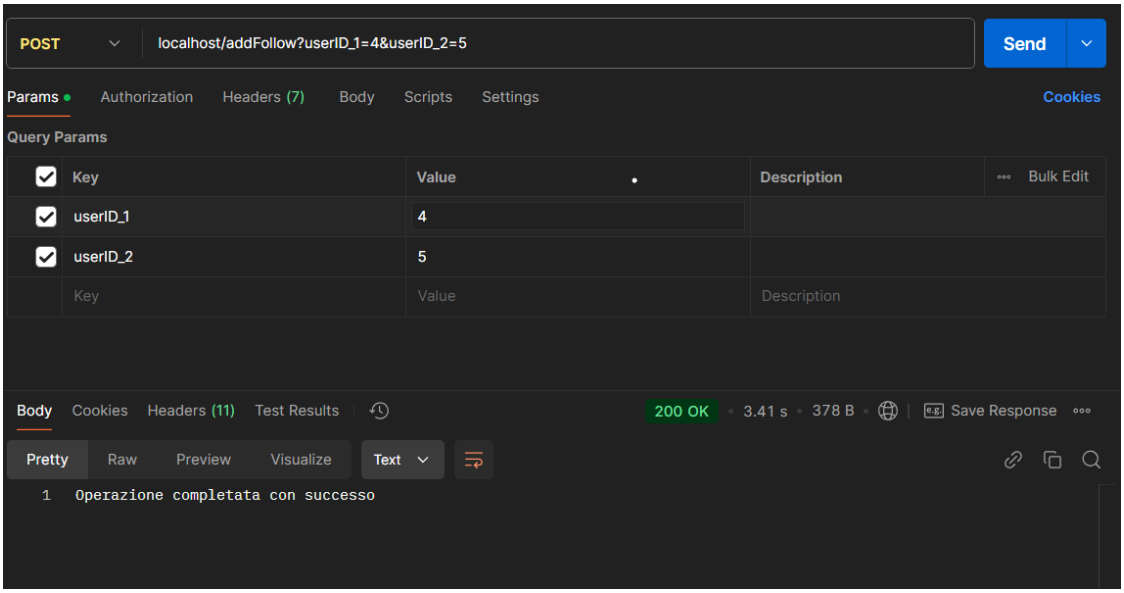


Figura 3.29 – Test addFollow con esito OK

400: BAD_REQUEST follower già segue followed

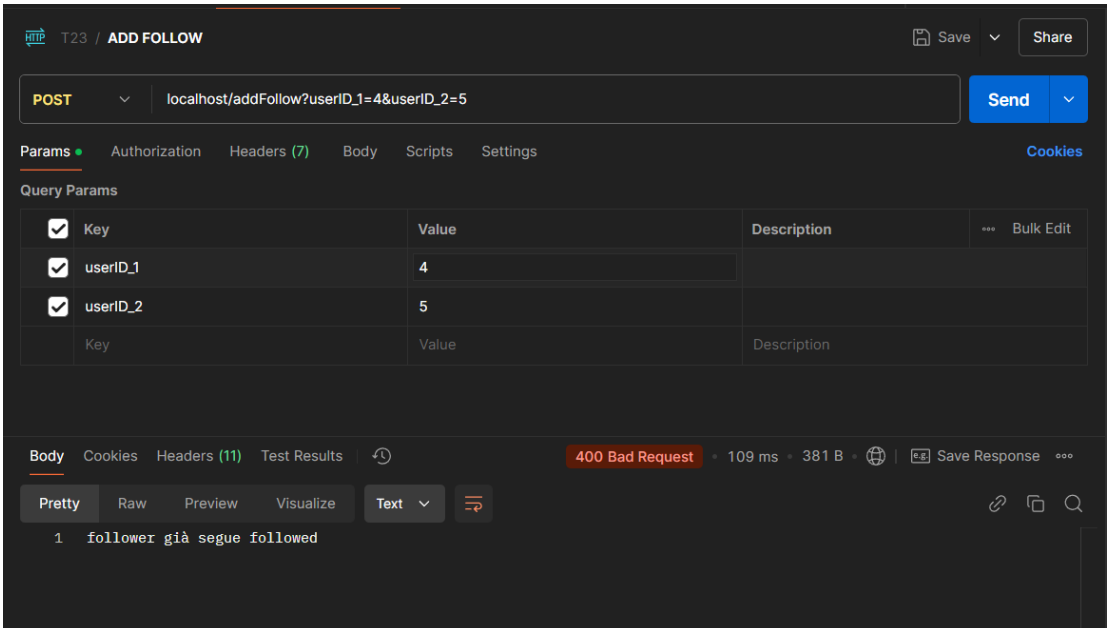


Figura 3.30 – Test addFollow con esito BAD_REQUEST

- rmFollow
  200: OK Utente 4 ha smesso di seguire utente 5

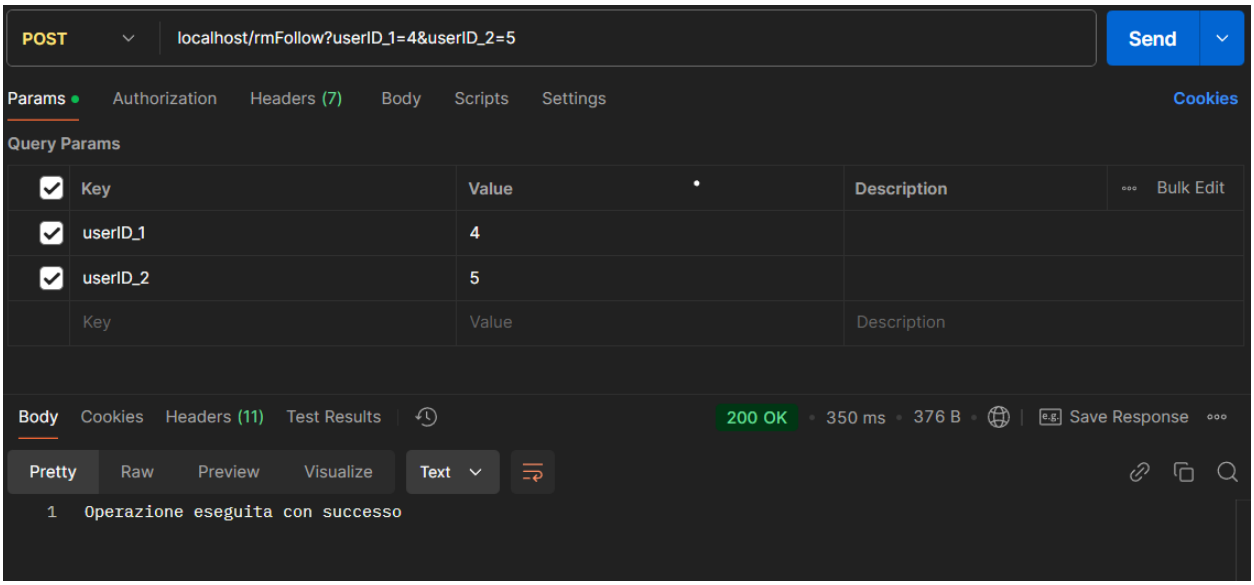


Figura 3.31 – Test rmFollow con esito OK

400: BAD_REQUEST L'utente 4 già non segue l'utente 5

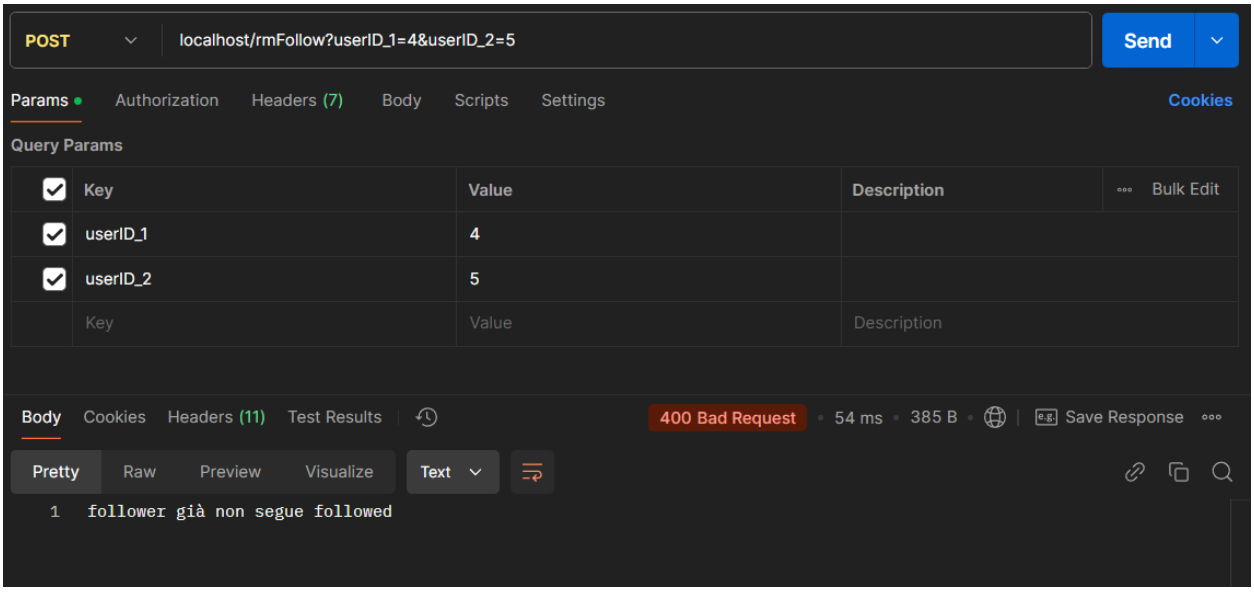


Figura 3.32 – Test rmFollow con esito BAD_REQUEST

- searchPlayer
  200: OK JSON dell'utente trovato
  Il JSON contenuto nel body è di questo tipo.

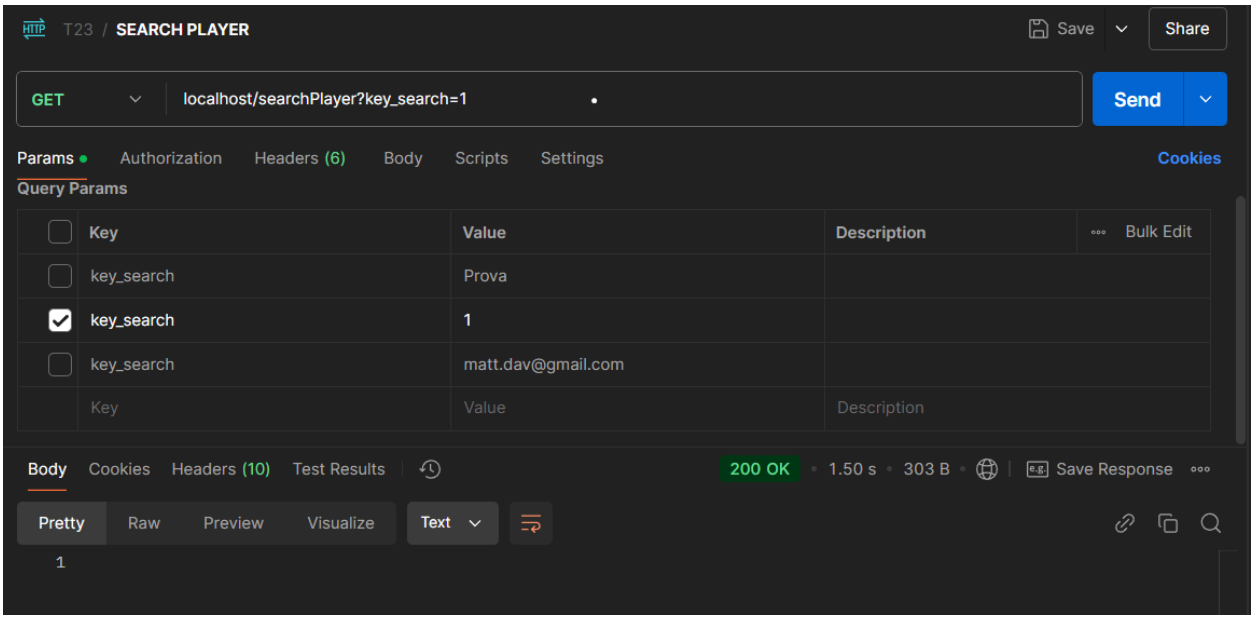


Figura 3.33 – Test searchPlayer con esito OK

200: *Body null* rappresentante che non è stato trovato quell'utente.

Figura 3.34 – Test searchPlayer con esito OK ma con il body null

- /modifyUser
  200: OK Aggiornamento Completato (struttura json)

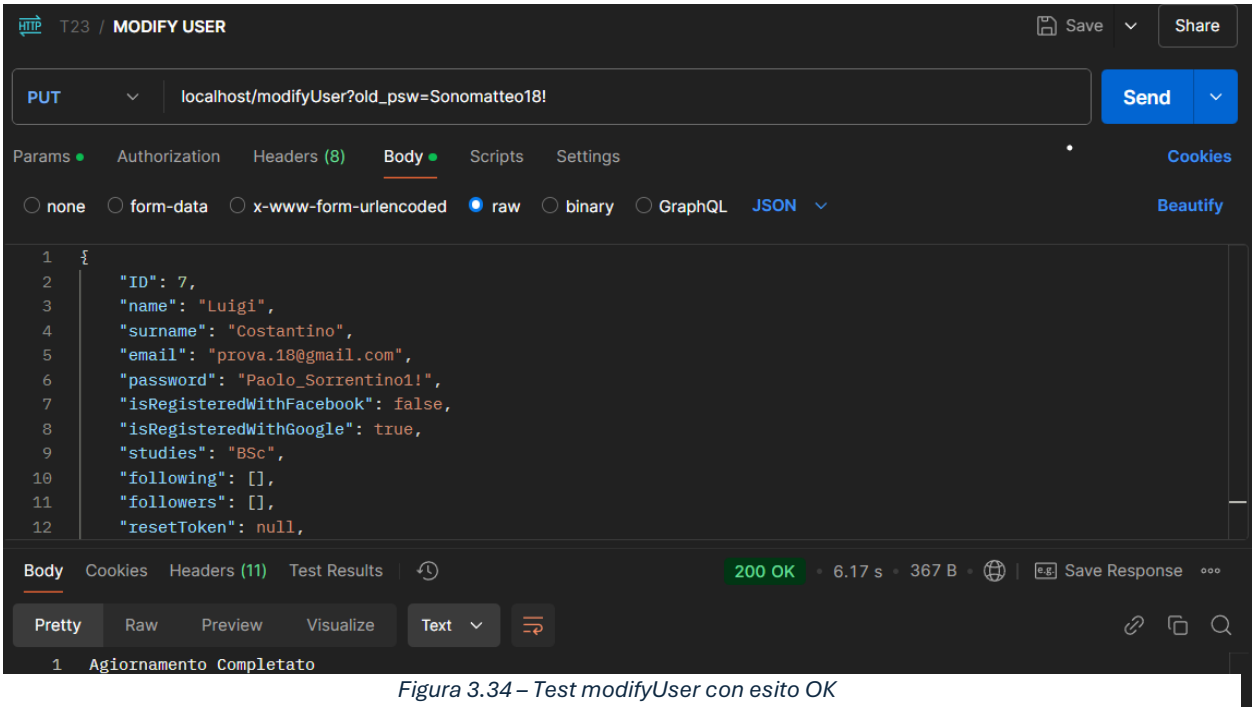


Figura 3.34 – Test modifyUser con esito OK

400: BAD REQUEST [Per la modifica dell'utente ci sono diversi errori 400 causati dai possibili errori nei campi inseriti, per semplificare useremo qui solo quella della vecchia password sbagliata]

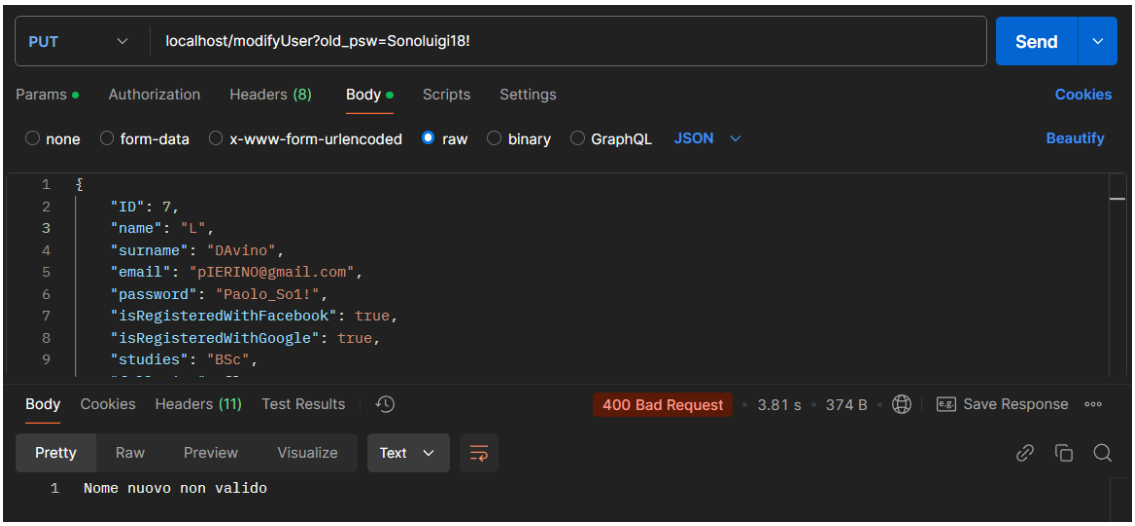


Figura 3.36 – Test modifyUser con esito BAD_REQUEST

4 Deploy

Basandoci sul contenuto delle documentazioni “Documentazione A10” [Vincenzo D’Angelo, Giorgio Di Costanzo e Aurelio Salvati] e “Documentazione_A13”, si riporta il Deployment Diagram per rappresentare la configurazione dei componenti hardware e software in un sistema distribuito. Non avendo apportato modifiche in merito si riporta semplicemente il diagramma realizzato nella versione A10.

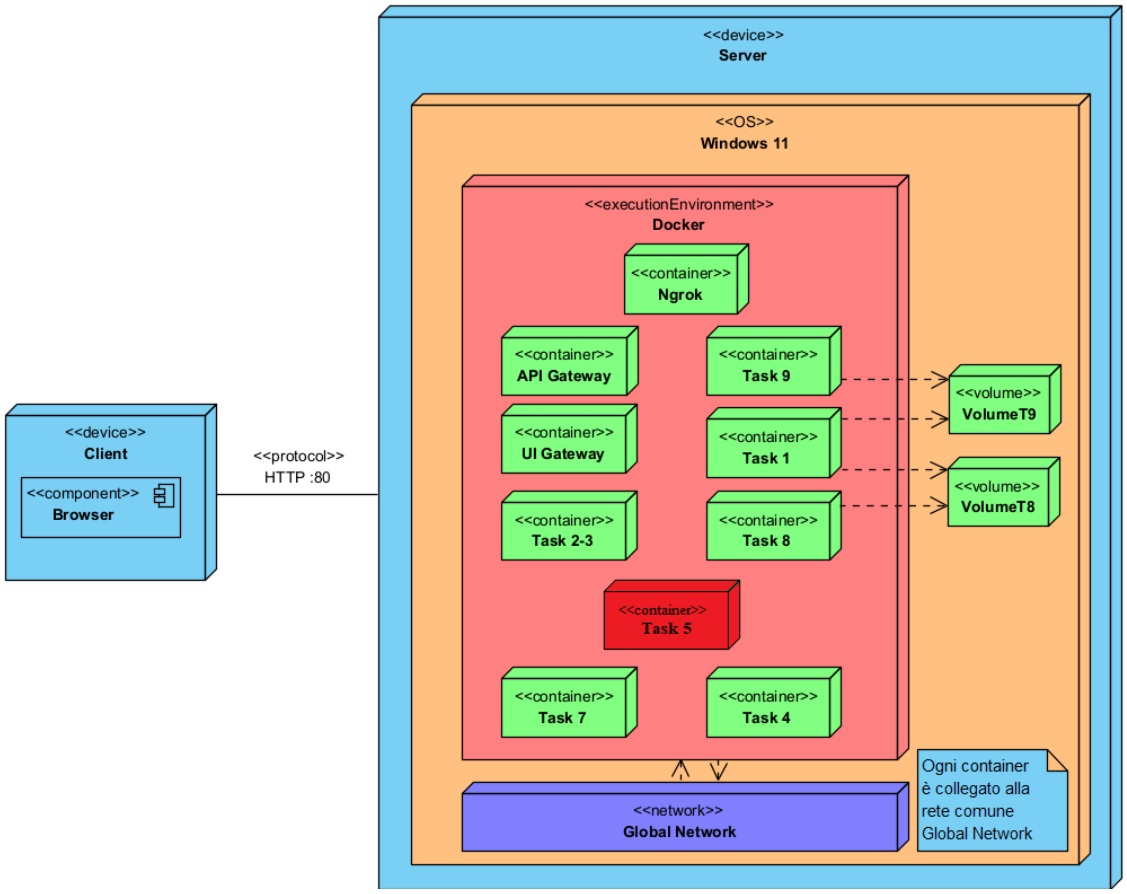


Figure 4.1 – Diagramma di deployment

Per implementare questa architettura si usa il servizio di “tunneling” promosso da **Ngrok**. Tale tecnica serve ad instradare il traffico di rete attraverso una connessione crittografata, creando un “tunnel” tra la rete locale e un indirizzo pubblico. Si preferisce questo approccio perché, come riportato nelle altre documentazioni, è stata la soluzione più testata e quindi più “stabile”; inoltre Ngrok si interfaccia direttamente con Docker. Quello che fa è creare un proprio container all’interno di Docker, fornendo un indirizzo pubblico e gestendo le richieste indirizzandole all’indirizzo locale.

Un’alternativa a Ngrok è **Pinggy**, che offre la possibilità di avere un indirizzo pubblico per un tempo limitato di 60 minuti. Rispetto a Ngrok non è necessario iscriversi, né installare nulla, ma con una sola riga di comando è possibile instradare localhost su un indirizzo pubblico temporaneo. Pinggy si basa sul protocollo SSH, e si interfaccia direttamente con l’indirizzo locale sul porto locale utilizzato dall'applicazione web. Quest’ultima soluzione ha il problema di generare un link contenente il proprio ip pubblico; quindi, è da usare con cautela su ambienti controllati.
In generale la documentazione A10 riporta che le soluzioni di deploy proposte sono da valutare solo in ottica per una sola fase di testing ma non sono definitive.

5 Migliorie per il Futuro

Appsettings.json

Potrebbe essere comodo all’interno del microservizio T23, aggiungere un file denominandolo “appsettings.json” al cui suo interno contenere tutte le Regex usate per i controlli nella fase di Registrazione o dell’Update dell’Utente e prelevarle all’occorrenza. In questo modo si ha un codice più modulare e “pulito”: modulare in quanto una modifica di una Regex non deve essere ripetuta per tutte le volte che viene ripetuta nel codice ma solo nell’appsettings (riducendo così errori o dimenticanze); pulito in quanto riteniamo essere più snello e leggibile aggiungere delle costanti interne alla classe per ogni Regex e usarle all’interno dei metodi.

Package di Utility
Per rendere il codice più leggero e manutenibile, sarebbe opportuno introdurre nei microservizi dei package di utility contenenti porzioni di codice comuni a più metodi o che appesantiscono le route API. Questa soluzione presenta un duplice vantaggio: da un lato, migliora la leggibilità del codice, facilitandone la comprensione da parte del lettore; dall’altro, elimina la ripetizione di codice, racchiudendo logiche condivise in metodi riutilizzabili. In questo modo, si garantisce una maggiore coerenza tra la fase di analisi dei casi d’uso, in cui uno stesso caso d’uso di inclusione o estensione può essere condiviso tra più casi d’uso, e la fase di implementazione, dove lo stesso codice non viene riscritto più volte ma riutilizzato efficacemente.

Integrazione della nostra soluzione con altre modifiche

La soluzione da noi proposta, a nostro avviso, è consistente e ben strutturata dal lato “back-end”. Come già esposto ampiamente, abbiamo ben valutato i flussi comunicativi tra i vari servizi e pensiamo che questa soluzione possa essere facilmente integrabile con

altre proposte per nuove logiche di business o migliorie lato front-end con un'estetica più accattivante alla visione dell'utente.

Framework front-end

Ci siamo limitati ad usare, per questo lato di sviluppo, gli strumenti già adottati dai vecchi teams. Potrebbe essere un'idea integrare delle tecnologie diverse come Angular o React.js, quest'ultimo ha compatibili delle librerie molto interessanti che permettono di abbellire in maniera consistente una Web Application come Material UI.

Ricostruire T4

Il modulo T4 è fondamentale, che è però implementato nel linguaggio GO, e risulta essere molto difficile da "studiare" o fare debbuging per scovare errori se non lo si conosce. Inoltre, dovrebbe permettere di memorizzare correttamente i progressi delle statistiche all'interno del table phca, implementando l'associazione tra robot e partita, per poter filtrare correttamente l'inserimento anche per i robot. In conclusione, dovrebbe essere rifatto da zero usando delle tecnologie simili a quelle usate sugli altri servizi, e permettere di predisporre correttamente la memorizzazione dei progressi delle statistiche per utente.

6 Tutorial installazione

L'installazione di questo software rappresenta una novità per chi, come noi, non ha mai utilizzato prima Docker. Riportiamo una breve introduzione agli strumenti utilizzati e qualche piccolo passo di installazione che non ci è stato subito chiaro.

Per prima cosa potremmo chiederci, cos'è **docker**? Esso è una piattaforma che permette di creare, distribuire ed eseguire applicazioni in contenitori leggeri, isolati e portabili. I container sono simili alle macchine virtuali, ma molto più efficienti in termini di risorse, poiché condividono lo stesso kernel del sistema operativo host. Docker è particolarmente utile per lo sviluppo, il testing e il deployment, poiché garantisce che l'applicazione funzioni allo stesso modo in ambienti diversi. Docker è progettato per funzionare nativamente su sistemi Linux. Con **WSL 2**, gli utenti Windows possono sfruttare il kernel Linux per eseguire i container Docker in modo efficiente e nativo, senza dover installare una macchina virtuale completa come in passato. WSL 2, utilizza un kernel Linux reale eseguito in una macchina virtuale leggera, garantendo una maggiore compatibilità e prestazioni migliori rispetto alla versione precedente. WSL 2 offre prestazioni elevate grazie al kernel Linux ottimizzato, rendendo l'esecuzione dei container molto più veloce rispetto alle configurazioni tradizionali basate su VM. Con Docker e WSL, gli sviluppatori possono lavorare in un ambiente Linux direttamente su Windows, accedendo agli strumenti e alle librerie disponibili su Linux senza cambiare sistema operativo.

I passi per installare l'applicazione sono:

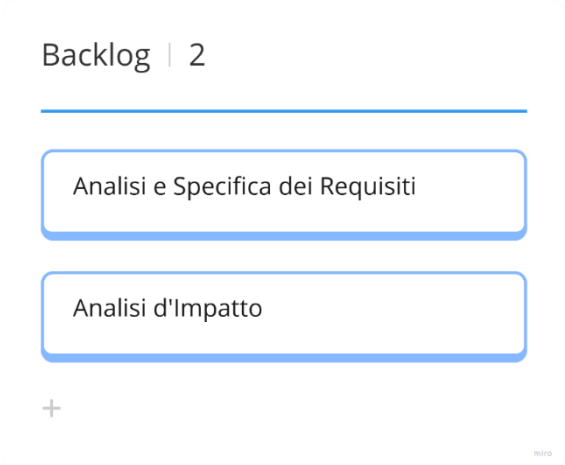
- scaricare e installare Docker Desktop. Se si sta operando in ambiente Windows, installare preliminarmente WSL; per farlo basta aprire una nuova finestra del terminale e lanciare il comando "wsl –install";
- assicurarsi di avere correttamente scaricato Java11 e Maven;
- effettuare il **git clone** del repository al link https://github.com/Testing-Game-SAD-2023/A13;
- avviare lo script "installer.bat" da terminale.

Tale installazione porterà alla creazione della rete "global-network" comune a tutti i container, del volume "VolumeT9" comune ai Task T1 e T9, del volume "VolumeT8" comune ai Task T1 e T8, e dei singoli container in Docker Desktop. Il container relativo al Task T9 (Progetto-SAD-G19-master) si sospenderà autonomamente dopo l'avvio, dato che viene utilizzato solo per popolare il volume VolumeT9 condiviso con il Task T1. Dopo l'installazione, controllare dunque che tutti gli altri container eccetto quello del T9 siano funzionanti, in caso contrario riavviarli manualmente lanciando per ultimo "ui gateway", il quale dovrà essere avviato solo dopo l'esecuzione di tutti gli altri. Per maggiori informazioni di natura tecnica, si può consultare nella seguente sezione del READ.ME fino ad arrivare al paragrafo di "Attenzione".

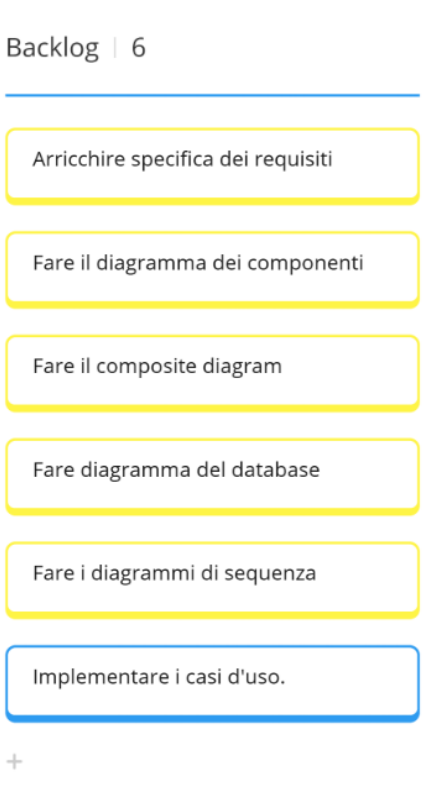
Viene riportato di seguito il link al Read.me contenente i passi per il corretto deploy.

Appendice

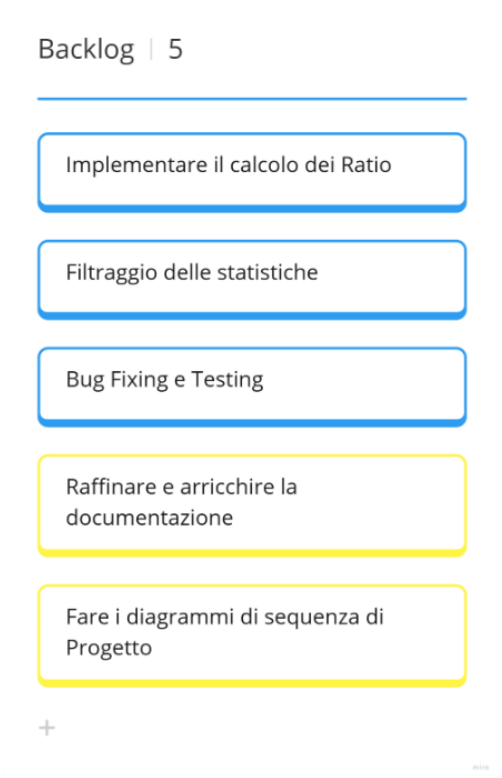
Appendice A : Tabelle dei backlog



Backlog prima iterazione 7/11/2024-21/11/2024



Backlog seconda iterazione 22/11/2024-5/11/2024



Backlog terza iterazione 6/12/2024 – 12/12/2024