



UNIVERSITÀ<sup>DEGLI STUDI DI</sup>  
NAPOLI FEDERICO II

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

Software Architecture Design

## *Documentazione Progetto SAD*

Anno Accademico 2023/2024

Professoressa: Prof.ssa Anna Rita Fasolino

Membri del Team:

Davide Fabio: M63001454

Giampetraglia Federica: M63001358

Riccardi Francesco: M63001372

De Nigris Fabio: M63001397

# Contents

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Descrizione del requisito assegnato . . . . .	3
1.2	Riassunto delle modifiche effettuate . . . . .	5
1.2.1	Task T2-3 . . . . .	6
1.2.2	Task T4 . . . . .	6
1.2.3	Task T5 . . . . .	7
1.2.4	Task T6 . . . . .	8
1.3	Panoramica delle API e degli Endpoints . . . . .	9
1.3.1	API Task T23 . . . . .	9
1.3.2	API Task T4 . . . . .	10
1.3.3	API Task T5 . . . . .	11
1.3.4	API Task T6 . . . . .	11
<b>2</b>	<b>ENACTE-ST</b>	<b>12</b>
2.1	Descrizione del progetto . . . . .	13
<b>3</b>	<b>Metodologie di sviluppo</b>	<b>14</b>
3.1	Adozione della Metodologia Agile . . . . .	14
3.1.1	Daily Scrum Meetings . . . . .	14
3.1.2	Definizione dello Sprint Backlog . . . . .	15
3.2	Strumenti di Comunicazione e Collaborazione . . . . .	15
3.2.1	Microsoft Teams . . . . .	15
3.2.2	Miro . . . . .	15
3.2.3	GitHub . . . . .	15
3.3	Tecnologie e Ambiente di Sviluppo . . . . .	16

3.3.1	Visual Studio Code . . . . .	16
3.3.2	Spring Boot . . . . .	16
3.3.3	Bootstrap . . . . .	16
3.3.4	Estensioni di Visual Studio Code per Spring Boot . . . . .	17
3.3.5	Apache Maven . . . . .	17
3.3.6	Go e PostgreSQL . . . . .	17
3.4	Strumenti di Modellazione e Documentazione . . . . .	18
3.4.1	Visual Paradigm . . . . .	18
3.4.2	SwaggerHub . . . . .	18
3.4.3	Draw.io . . . . .	18
<b>4</b>	<b>Organizzazione del Lavoro con Scrum</b>	<b>19</b>
4.1	Introduzione a Scrum . . . . .	19
4.1.1	Adozione di Scrum nel Progetto . . . . .	19
4.2	Primo sprint . . . . .	20
4.3	Secondo sprint . . . . .	21
4.4	Terzo sprint . . . . .	21
4.5	Quarto sprint . . . . .	22
4.6	Quinto sprint . . . . .	23
4.7	Sesto sprint . . . . .	24
4.8	Settimo sprint . . . . .	25
<b>5</b>	<b>Analisi del dominio</b>	<b>26</b>
5.1	Definizione dei Requisiti e User Stories . . . . .	26
5.1.1	Requisiti Funzionali . . . . .	28
5.1.2	Requisiti Non Funzionali . . . . .	29
5.2	Diagramma dei casi d'uso . . . . .	30
5.3	Scenari di funzionamento e diagrammi di attività . . . . .	32
5.3.1	Scenario 1 : Seleziona opzione . . . . .	32
5.3.2	Scenario 2 : Visualizza storico . . . . .	32
5.3.3	Scenario 3 : Visualizza classifica . . . . .	33
5.3.4	Scenario 4 : Seleziona opzione di gioco . . . . .	33
<b>6</b>	<b>Introduzione al contesto</b>	<b>34</b>

6.1	Vista dei componenti . . . . .	35
6.2	Considerazioni sull'implementazione: pattern MVC . . . . .	38
6.3	Refactoring del Codice . . . . .	39
6.4	Diagrammi di sequenza . . . . .	40
6.4.1	Visualizza Storico . . . . .	40
6.4.2	Visualizza Classifica . . . . .	42
6.4.3	Nuova Partita . . . . .	43
6.4.4	Logout . . . . .	45
<b>7</b>	<b>Modifiche al Task 2-3</b>	<b>46</b>
7.1	Diagramma delle classi . . . . .	46
7.2	Controller . . . . .	47
7.3	Codice: Controller . . . . .	48
<b>8</b>	<b>Modifiche al Task 4</b>	<b>51</b>
8.1	Diagramma ER di partenza . . . . .	51
8.1.1	Tabella turns . . . . .	53
8.1.2	Tabella players . . . . .	53
8.1.3	Tabella games . . . . .	54
8.2	Modifiche effettuate . . . . .	55
8.2.1	Diagramma ER . . . . .	55
8.2.2	Player . . . . .	56
8.2.3	Player: Codice . . . . .	57
8.2.4	Recupero Lista Giocatori . . . . .	58
8.2.5	Aggiornamento Dettagli Giocatore . . . . .	59
8.2.6	Turn . . . . .	61
8.2.7	Turn: Codice . . . . .	62
8.2.8	Recupero Storico Turni Giocatore . . . . .	64
8.2.9	Model . . . . .	66
8.2.10	Model: Codice . . . . .	67
<b>9</b>	<b>Modifiche al Task5</b>	<b>70</b>
9.1	Diagramma di contesto . . . . .	70
9.2	Diagrammma dei package . . . . .	72

9.3	Diagramma delle classi . . . . .	73
9.3.1	GUIController . . . . .	74
9.3.2	Codice: GUIController . . . . .	75
9.3.3	Verifica dell'Autenticazione . . . . .	78
9.3.4	Recupero della Classifica dei Giocatori . . . . .	78
9.3.5	Recupero delle Informazioni sui Robot . . . . .	78
9.3.6	Recupero dell'ID dell'Utente Autenticato . . . . .	79
9.3.7	Recupero del Nome dell'Utente Autenticato . . . . .	79
9.4	Chiamate interne all'applicazione - lista API . . . . .	79
9.4.1	GameDataWriter . . . . .	80
9.4.2	Codice: GameDataWriter . . . . .	80
9.4.3	Game . . . . .	81
9.4.4	Codice: Game . . . . .	82
9.5	Pagine create . . . . .	83
<b>10</b>	<b>Modifiche al Task 6</b>	<b>86</b>
10.1	Diagramma dei package . . . . .	86
10.2	MyController . . . . .	88
10.3	Codice: MyController . . . . .	88
<b>11</b>	<b>Deployment</b>	<b>91</b>
<b>12</b>	<b>Testing</b>	<b>93</b>
12.1	Testing End to End . . . . .	93
12.1.1	Casi di Test . . . . .	94
12.1.2	Setup . . . . .	94
12.1.3	Classifica test . . . . .	95
12.1.4	Storico test . . . . .	95
12.1.5	Nuova Partita test . . . . .	96
12.1.6	Scelta classi e robot test . . . . .	97
12.1.7	Risultato del Test . . . . .	98
<b>13</b>	<b>Guida all'installazione</b>	<b>99</b>
13.1	Docker e WSL . . . . .	99

## CONTENTS

---

13.2 Installazione . . . . .	99
13.2.1 Passo 1 . . . . .	99
13.2.2 Passo 2 . . . . .	100
13.2.3 Passo 3 . . . . .	100
13.2.4 Utilizzo . . . . .	101
<b>14 Glossary</b>	<b>103</b>

## CONTENTS

Il presente documento è fondamentale per tracciare lo sviluppo del progetto assegnato al team A9, con il compito R9. Dall'analisi delle storie utente alla compilazione del codice, ogni fase, incluso l'avanzamento nella progettazione di casi d'uso e scenari, sarà accuratamente documentata.

Con l'implementazione di una metodologia AGILE, il processo di sviluppo si svolgerà in modo incrementale, facilitando una progettazione e implementazione progressiva. Nel contesto della progettazione architetturale del software, saranno valutati gli aspetti cruciali legati all'interfaccia grafica. Particolare attenzione sarà rivolta agli aggiornamenti e alle ottimizzazioni dell'architettura software, garantendo che il risultato finale rifletta una struttura robusta e un'interfaccia utente intuitiva. L'approccio architetturale sarà determinante per assicurare coerenza, flessibilità e scalabilità nel corso delle iterazioni del progetto.

# Chapter 1

## Introduzione

Sul sito GitHub <https://github.com/Testing-Game-SAD-2023/T11-G41> è disponibile il progetto da cui siamo partiti per sviluppare il nostro requisito. Sul sito GitHub <https://github.com/Testing-Game-SAD-2023/A9-R9> è disponibile il codice e la documentazione prodotti dal nostro lavoro.

Il nostro lavoro si è concentrato sulla modifica dei seguenti task di cui si consiglia di leggere la documentazione <https://github.com/Testing-Game-SAD-2023> per una comprensione migliore del lavoro.

**Task T2-3:** [Requisiti sulla Registrazione dei Giocatori e Requisiti sulla Autenticazione dei Giocatori ]

**Task T4:** [Requisiti sul mantenimento delle Partite giocate]

**Task T5:** [Requisiti sull'avvio del Primo Scenario di Gioco]

**Task T6:** [Requisiti Sull'Editor di Test Case]

## 1.1 Descrizione del requisito assegnato

Il requisito R9 si focalizza sulla necessità di modificare l’interfaccia utente per facilitare al giocatore la gestione delle partite giocate, la visualizzazione della classifica totale dei giocatori, la visualizzazione dello storico del giocatore loggato e la scelta della tipologia di partita da giocare.

### a) Storico delle Partite:

Il sistema deve consentire al giocatore di visualizzare lo storico delle partite precedentemente giocate. Questo può essere ottenuto attraverso un’apposita sezione dell’interfaccia utente dedicata allo storico delle partite. Ogni voce in questo elenco di partite deve includere le seguenti informazioni:

- **ID Partita:** Indica l’identificativo univoco del turno .
- **Inizio Partita:** La data e l’ora in cui la partita è iniziata .
- **Fine Partita:** La data e l’ora in cui la partita è finita .
- **Classe Testata:** La classe di test a cui la partita è associata.
- **Robot Sfidato:** Il robot avversario contro cui il giocatore ha giocato.
- **Risultato partita:** Indicare se il giocatore ha vinto o perso la partita.
- **Difficoltà:** indica la difficoltà del robot sfidato.
- **Scores :** indica il punteggio che effettivamente il giocatore è riuscito a raggiungere contro il robot.

### b) Nuova Partita:

Il sistema deve offrire al giocatore la possibilità di iniziare una nuova partita. Questo può essere fatto attraverso un’opzione dedicata nell’interfaccia utente. Il giocatore ha la scelta fra due tipi di partite:

1. **Partita Base (già disponibile):** Il sistema mostra l’elenco delle classi, dei Robot e dei livelli disponibili, come già implementato nell’interfaccia utente esistente. Il giocatore può

effettuare le sue scelte e iniziare la partita.

2. **Partita Contro tutti i Robot (v. requisito R10):** Il sistema richiede solo la classe da testare, e visualizza i relativi Robot disponibili.

### c) Classifica Totale dei Giocatori

Dopo il login del giocatore, il sistema deve consentire al giocatore di scegliere se visualizzare la classifica totale dei giocatori. Il sistema deve fornire una sezione dedicata all'interno dell'interfaccia utente che mostra la classifica totale dei giocatori. La classifica deve essere ordinata in base al numero totale di partite vinte da ciascun giocatore.

Di seguito verrà fornito un sintetico riassunto delle modifiche apportate in ogni task. Una discussione dettagliata su ciascuna di queste modifiche, inclusa l'illustrazione di tutte le aggiunte e le modifiche apportate, sarà presentata nei capitoli successivi, dove verranno esaminati singolarmente i task modificati.

## 1.2 Riassunto delle modifiche effettuate

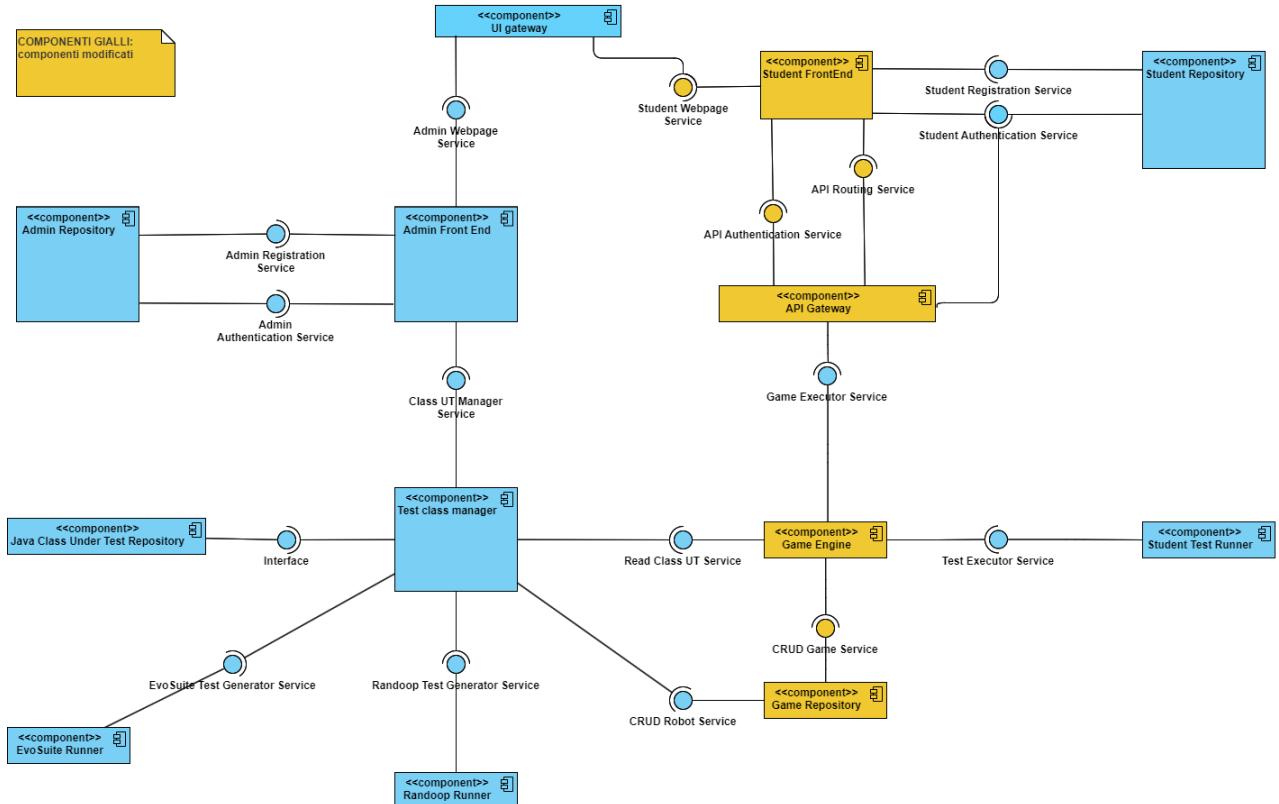


Figure 1.1: Vista dei componenti

I componenti gialli, che sono stati modificati, sono i seguenti:

- **API Gateway** funge da gestore delle richieste API, implementando routing, autenticazione e autorizzazione. Garantisce sicurezza tramite token JWT e sfrutta Netflix Zuul per il routing e il filtraggio dei servizi, operando con Spring Boot.
- **Student Front End** è il punto di interazione per gli studenti, dal login al gameplay. Costruito con Spring Boot per il back-end e CodeMirror per l'editor di testo front-end.
- **Game Engine** Gestisce la logica di gioco e interagisce con Student Test Runner e Game Repository, realizzato con Spring Boot.
- **Game Repository** Archivia lo storico delle partite e le classi giocabili, utilizzando Chi e PostgreSQL per il database.

### 1.2.1 Task T2-3

#### Miglioramenti al Controller

Nel *Controller*, abbiamo integrato nuove funzioni per facilitare la manipolazione dei dati utente:

- Implementazione delle funzioni `extractName` e `extractIDString` per estrarre rispettivamente il nome e l'ID dell'utente dal token JWT.
- Parallelamente a queste implementazioni, sono stati introdotti due nuovi endpoint nell'API. Il primo endpoint utilizza la funzione `extractName` per fornire l'accesso al nome dell'utente autenticato, mentre il secondo si avvale della funzione `extractIDString` per recuperare l'ID dell'utente.
- Aggiornamento della funzione `generateToken` per includere il nome del giocatore nel token JWT.

#### Impatto sul Progetto

Le modifiche apportate erano necessarie per poter generare la classifica e lo storico, per cui sono necessari il nome e l'ID del giocatore.

### 1.2.2 Task T4

#### Ristrutturazione del Database

L'analisi iniziale del diagramma Entità-Relazione (ER) ha evidenziato incongruenze strutturali e informative necessitanti interventi correttivi per supportare le nuove funzionalità di gioco, classifiche e storico partite.

#### Aggiornamento delle Tabelle

Modifiche sostanziali alla tabella *turns* per includere nuovi campi critici per la classifica e lo storico (*TestClass*, *Robot* e *Difficulty*). Aggiunta di nuove colonne alla tabella *players* per riflettere informazioni dettagliate sulle prestazioni dei giocatori.

### **Introduzione di Nuove API**

Implementazione di API aggiuntive per la tabella *players* per gestire l'interazione con i dati degli utenti e facilitare la creazione delle classifiche.

### **Miglioramenti alla Logica di Business**

Estensione della logica di business con nuove funzioni nel Service e aggiornamenti al DTO di Turn per migliorare l'accuratezza dello storico delle partite e dei punteggi.

### **Ottimizzazione del Modello**

Inserimento di nuove funzioni nel modello per popolare i campi aggiunti nella tabella *players* e garantire la correttezza delle statistiche di gioco aggiornate.

### **Endpoint API**

Aggiunta di endpoint API per recuperare e aggiornare informazioni sulle partite giocate e sui giocatori, cruciali per l'accesso e la presentazione dei dati storici e della classifica.

### **Codice e Struttura**

Rifinitura del codice sorgente per allinearsi alle modifiche strutturali e funzionali, con l'aggiunta di tutti gli endpoint necessari nel *main.go*.

### **Impatto sul Progetto**

Le modifiche apportate erano necessarie per poter generare la classifica e lo storico, per cui è necessario poter collegare un determinato giocatore (tramite nome e ID) alle partite giocate e riuscire a determinare il numero di partite vinte. Era inoltre necessario poter salvare nuove informazioni come difficoltà, robot sfidato e classe testata.

#### **1.2.3 Task T5**

##### **Ottimizzazione mediante Diagramma delle Classi**

L'uso del *diagramma delle classi* ha permesso di mappare in modo dettagliato le strutture dati e le loro interconnessioni, guidando l'ottimizzazione delle classi esistenti e l'integrazione di nuove

classi nel sistema.

### **Modifiche al GUIController**

Tra le novità più rilevanti, si segnala l'introduzione di variabili quali `IdAuth` e `NameAuth` nel `GUIController` per un'efficace gestione dell'autenticazione, e la rielaborazione del metodo `AllRobotsPage()` per migliorare la selezione dei robot nell'interfaccia utente.

### **Modifiche al GameDataWriter**

Il `GameDataWriter` è stato modificato per permettere di aggiungere il nome del giocatore all'interno della tabella `player`.

### **Sviluppo di Nuove Pagine Web**

Il sistema è stato arricchito con l'aggiunta di nuove pagine web, progettate per elevare l'esperienza utente e ampliare le possibilità di interazione con l'applicativo. Queste pagine comprendono la *Main*, la *Classifica*, lo *Storico delle partite*, la selezione della *Modalità di Gioco*, l'elenco di *Robot* e il *Report delle partite*.

#### **1.2.4 Task T6**

##### **Miglioramenti a MyController**

Nella classe `MyController`, abbiamo apportato modifiche significative:

- Introduzione di nuove variabili (`nomeCUT`, `robotScelto`, `difficoltà`) per gestire la selezione e il testing delle classi, il robot di testing scelto e il livello di difficoltà.
- Miglioramento della funzione `runner` per integrare queste nuove variabili e ottimizzare l'esecuzione dei test.

In questa sezione, abbiamo proceduto con aggiornamenti mirati alla tabella “Turn” del database (T4). L'obiettivo principale è stato quello di registrare nella tabella tutti quei parametri di gioco che, nonostante fossero disponibili, non venivano salvati nel database e, di conseguenza, non erano visualizzabili nell'applicazione.

## 1.3 Panoramica delle API e degli Endpoints

Il progetto ha visto l'introduzione e la revisione di numerose API e dei relativi endpoints, elementi fondamentali per ampliare le funzionalità del sistema e ottimizzare l'interazione degli utenti. In questa sezione, presentiamo un riassunto delle API chiave, mettendo in evidenza gli endpoints che sono stati aggiunti o modificati. Questa analisi offre una visione complessiva dell'architettura delle API, garantendo una comprensione chiara delle funzionalità disponibili e della loro integrazione nell'ambiente di gioco.

### 1.3.1 API Task T23

L'API del Task T23 offre una serie di endpoints focalizzati sulla gestione dell'autenticazione e delle informazioni relative agli utenti. La documentazione completa può essere consultata al seguente link: [https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD\\_T23/1.0.0](https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD_T23/1.0.0).

#### User Management

<code>POST</code>	<code>/register</code> User Registration			
<code>GET</code>	<code>/login</code> User Login			
<code>GET</code>	<code>/password_reset</code> Password Reset Request			
<code>GET</code>	<code>/password_change</code> Change Password			
<code>POST</code>	<code>/logout</code> User Logout			
<code>GET</code>	<code>/get_ID</code> Get User ID			

#### Token Operations

<code>POST</code>	<code>/nameToken</code> Extract Name from Token			
<code>POST</code>	<code>/IdToken</code> Extract ID from Token			
<code>POST</code>	<code>/validateToken</code> Check Token Validity			

### 1.3.2 API Task T4

L'API del Task T4 si concentra sulla gestione dei dati di gioco, inclusi giocatori, partite e classifiche. La documentazione dettagliata è disponibile al seguente link: [https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD\\_T4/1.0.0](https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD_T4/1.0.0).

games	^
GET /games/{id} Retrieve a game by id	📋 ⚡ ▾
PUT /games/{id} Update a game	📋 ⚡ ▾
DELETE /games/{id} Delete a Game	📋 ⚡ ▾
POST /games Create a Game	📋 ⚡ ▾
GET /games Retrieve games by date interval	📋 ⚡ ▾
rounds	^
GET /rounds/{id} Retrieve a round	📋 ⚡ ▾
PUT /rounds/{id} Update a round	📋 ⚡ ▾
DELETE /rounds/{id} Delete a round	📋 ⚡ ▾
POST /rounds Creates a round	📋 ⚡ ▾
GET /rounds Retrieve rounds per game	📋 ⚡ ▾
turns	^
GET /turns/{id} Retrieve a turn	📋 ⚡ ▾
PUT /turns/{id} Update a turn	📋 ⚡ ▾
DELETE /turns/{id} Delete a turn	📋 ⚡ ▾
PUT /turns/{id}/files Upload turn files	📋 ⚡ ▾
GET /turns/{id}/files Download turn files	📋 ⚡ ▾
GET /turns Retrieve turns in a round	📋 ⚡ ▾
POST /turns Create turns for players	📋 ⚡ ▾
robots	^
GET /robots Retrieve test results with filters	📋 ⚡ ▾
DELETE /robots Delete test results by a specific class id	📋 ⚡ ▾
POST /robots Create a robot results in batch	📋 ⚡ ▾
players	^
GET /players Retrieve list of players	📋 ⚡ ▾
PUT /players/{accountId} Update player details	📋 ⚡ ▾

### 1.3.3 API Task T5

L'API del Task T5 amplia le funzionalità del sistema introducendo endpoints legati alla configurazione e alla gestione delle sessioni di gioco. La documentazione è accessibile qui: [https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD\\_T5/1.0.0](https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD_T5/1.0.0).

The screenshot shows the API documentation for Task T5. It includes two main sections:

- Page Navigation**: A list of GET endpoints for various pages:
  - /main Main Page
  - /classifica Leaderboard Page
  - /all\_robots All Robots Page
  - /game\_mode Game Mode Page
  - /storico History Page
  - /choose Choose Page
  - /report Report Page
  - /report1 Report1 Page
  - /editor Editor Page
- Game Management**: A list of POST endpoints for game management:
  - /save-data Save Game Data

### 1.3.4 API Task T6

L'API del Task T6 introduce miglioramenti e nuove funzionalità, in particolare per quanto riguarda l'interazione avanzata con i robot di gioco e la personalizzazione delle sfide. La documentazione è disponibile al link: [https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD\\_T6/1.0.0](https://app.swaggerhub.com/apis-docs/FRANCYRIC10/SAD_T6/1.0.0).

The screenshot shows the API documentation for Task T6 under the 'default' section. It lists several endpoints:

- POST /receiveClassUnderTest Receive Class Under Test
- POST /sendInfo Send Info
- GET /getResultXml Get Result XML
- GET /getResultRobot Get Result Robot
- GET /getJaCoCoReport Get JaCoCo Report
- POST /inviaDatiEFile Invia Dati e File

## Chapter 2

### ENACTE-ST

Il progetto European iNnovative AllianCe for TESTing, ENACTE-ST, si propone di enfatizzare l'importanza cruciale del testing nello sviluppo e nell'implementazione di applicazioni web. In diverse circostanze, si manifestano vulnerabilità ancora non identificate durante le fasi precedenti al lancio, evidenziando la necessità di affrontare tali questioni in modo efficace durante il processo di testing.

Nonostante ciò, il testing rimane spesso una disciplina trascurata e sottostimata all'interno del contesto dello sviluppo software. Al fine di superare questa sfida, l'Università di Napoli Federico II si impegna attivamente in collaborazione con piccole imprese per promuovere e consolidare l'importanza del testing.

L'approccio innovativo di ENACTE-ST si concretizza attraverso la creazione di un Educational Game. In tale contesto, i partecipanti sono chiamati a sfidare un software di generazione automatica di test, con l'obiettivo di coprire in modo esaustivo tutti i possibili scenari. Questo non solo offre un'opportunità pratica per lo sviluppo delle competenze nel testing, ma rende anche l'apprendimento di questa disciplina più coinvolgente e stimolante.

Attraverso ENACTE-ST, si auspica di incentivare una maggiore attenzione e valorizzazione per la disciplina del testing, contribuendo in modo significativo a garantire la solidità, la sicurezza e l'affidabilità delle applicazioni web.

## 2.1 Descrizione del progetto

"Man vs Automated Testing Tools Challenges" costituisce un gioco educativo nel quale gli studenti si confrontano con strumenti di testing automatizzati capaci di generare automaticamente casi di test JUnit, come ad esempio Randoop o EvoSuite. L'obiettivo primario consiste nel raggiungere un determinato livello di copertura, quale ad esempio la copertura delle linee di codice (LOC).

# **Chapter 3**

## **Metodologie di sviluppo**

L'avvio del progetto è stato caratterizzato dall'adozione di una metodologia Agile, che ha impostato le fondamenta per un ambiente di sviluppo dinamico e adattabile. Questa sezione delinea i principi operativi e gli strumenti di collaborazione e di sviluppo che hanno guidato il team nel corso dello sviluppo.

### **3.1 Adozione della Metodologia Agile**

La metodologia Agile è stata la pietra angolare del nostro processo di sviluppo, enfatizzando la flessibilità, la comunicazione continua e la risposta rapida ai cambiamenti. Attraverso l'implementazione delle pratiche Agile, il team ha potuto mantenere un ritmo di lavoro sostenibile, assicurando al contempo la qualità e l'efficacia delle soluzioni sviluppate.

#### **3.1.1 Daily Scrum Meetings**

Un elemento chiave della nostra adozione Agile è stata la pratica del Daily Scrum. Queste brevi riunioni quotidiane hanno offerto al team l'opportunità di condividere aggiornamenti sul progresso del lavoro, discutere sfide imminenti e pianificare attività per la giornata successiva. Il Daily Scrum ha facilitato una comunicazione trasparente e tempestiva tra i membri del team, consentendo una rapida identificazione e risoluzione delle problematiche.

### **3.1.2 Definizione dello Sprint Backlog**

La fase iniziale del progetto ha visto il team impegnato nella definizione dello Sprint Backlog. Attraverso un'analisi approfondita del progetto assegnato, sono state identificate le priorità, delineate le attività e distribuite le responsabilità. Questo processo ha consentito una pianificazione dettagliata e realistica delle milestones del progetto, ottimizzando l'allocazione delle risorse e garantendo il rispetto delle scadenze stabilite.

## **3.2 Strumenti di Comunicazione e Collaborazione**

### **3.2.1 Microsoft Teams**

Microsoft Teams si è rivelato essenziale per mantenere una comunicazione fluida e organizzata tra i membri del team. La piattaforma ha offerto un ambiente virtuale integrato per videoconferenze, condivisione di documenti e comunicazioni rapide, sostenendo efficacemente sia le fasi di lavoro collettivo che quelle individuali.

### **3.2.2 Miro**

Miro ha fornito un'interfaccia collaborativa visuale, permettendo al team di sfruttare lavagne digitali interattive per brainstorming, schematizzazioni e pianificazioni. L'uso di Miro ha migliorato significativamente l'efficacia delle sessioni di lavoro congiunto, rendendo possibile la visualizzazione e la manipolazione collettiva di concetti e strutture complesse.

### **3.2.3 GitHub**

L'utilizzo di GitHub ha centralizzato la gestione del codice sorgente, facilitando la collaborazione tra gli sviluppatori attraverso funzionalità di controllo versione e revisione del codice. La piattaforma ha supportato il lavoro distribuito, permettendo la sincronizzazione del lavoro e la gestione trasparente delle modifiche, contribuendo alla coerenza e all'integrità del codice prodotto.

In conclusione, l'impiego di questi strumenti ha svolto un ruolo fondamentale nel successo del progetto, consentendo una gestione efficace delle dinamiche di team e delle attività di sviluppo, in linea con i principi della metodologia Agile.

### 3.3 Tecnologie e Ambiente di Sviluppo

Il progetto ha integrato un insieme di tecnologie all'avanguardia e strumenti di sviluppo per facilitare la realizzazione di soluzioni software complesse. Questa sezione delinea le scelte tecnologiche fondamentali e descrive brevemente il loro ruolo e contributo nello sviluppo del progetto.

#### 3.3.1 Visual Studio Code

Visual Studio Code (VS Code) ha rappresentato l'ambiente di sviluppo primario per il team, grazie alla sua flessibilità e alla vasta gamma di estensioni disponibili. Caratterizzato da un'interfaccia utente intuitiva e da una leggerezza che non pregiudica le prestazioni, VS Code ha supportato efficacemente lo sviluppo di codice in diversi linguaggi di programmazione. Le sue capacità di integrazione con sistemi di controllo versione come Git e la presenza di estensioni dedicate a framework come Spring Boot hanno permesso una gestione efficiente del codice e un'ottimizzazione del flusso di lavoro di sviluppo.

#### 3.3.2 Spring Boot

Spring Boot si è distinto come un framework chiave per lo sviluppo di applicazioni Java, grazie alla sua capacità di ridurre la complessità di configurazione e promuovere una rapida prototipazione. La sua architettura basata su microservizi e il supporto per la creazione di applicazioni stand-alone con server web incorporati hanno semplificato significativamente lo sviluppo, il test e il deployment delle applicazioni. La facilità di integrazione con database, sistemi di messaggistica e altri servizi esterni ha reso Spring Boot uno strumento indispensabile per lo sviluppo di applicazioni backend scalabili e performanti.

#### 3.3.3 Bootstrap

Bootstrap è stato adottato come framework CSS per lo sviluppo dell'interfaccia utente, grazie alla sua vasta libreria di componenti predefiniti e al suo sistema di layout responsive. La sua capacità di accelerare lo sviluppo front-end, assicurando al contempo coerenza e compatibilità cross-browser, ha reso Bootstrap uno strumento prezioso per la progettazione di interfacce utente moderne e accessibili.

### 3.3.4 Estensioni di Visual Studio Code per Spring Boot

Le estensioni specifiche per Spring Boot in VS Code, come Spring Boot Tools e Spring Boot Dashboard, hanno arricchito l'ambiente di sviluppo con funzionalità dedicate alla gestione di applicazioni Spring. Queste estensioni hanno offerto una visualizzazione intuitiva delle applicazioni Spring Boot in esecuzione, semplificato il debugging e migliorato la produttività degli sviluppatori fornendo un rapido accesso alle configurazioni e ai log delle applicazioni.

### 3.3.5 Apache Maven

Apache Maven è stato utilizzato come strumento di automazione della build, sfruttando la sua potente gestione delle dipendenze e il suo modello di progetto basato su convenzioni. La sua integrazione con VS Code ha permesso di mantenere una struttura di progetto ordinata e di automatizzare compiti di build e deployment, incrementando l'efficienza dello sviluppo.

### 3.3.6 Go e PostgreSQL

L'impiego del linguaggio Go, in combinazione con il sistema di gestione del database PostgreSQL, ha costituito la base tecnologica per la gestione dei dati e la logica di business del progetto. Go, con la sua sintassi concisa e il suo modello di concorrenza, ha facilitato lo sviluppo di servizi backend ad alte prestazioni. PostgreSQL, noto per la sua affidabilità e le sue avanzate funzionalità SQL, ha garantito un'efficiente gestione e persistenza dei dati, supportando complesse operazioni di query e transazioni.

In sintesi, la selezione di queste tecnologie e strumenti ha contribuito significativamente all'efficacia e alla qualità dello sviluppo del progetto, permettendo al team di affrontare con successo le sfide tecniche e di realizzare un prodotto software funzionale e scalabile.

## 3.4 Strumenti di Modellazione e Documentazione

### 3.4.1 Visual Paradigm

Visual Paradigm si è affermato come strumento indispensabile nel processo di progettazione e analisi dei requisiti del progetto. Questa piattaforma di modellazione avanzata ha permesso di elaborare con precisione diagrammi UML, contribuendo significativamente alla documentazione e alla comprensione architettonica del sistema. Attraverso l'utilizzo di Visual Paradigm, il team ha potuto rappresentare efficacemente le strutture, i comportamenti e le interazioni all'interno del software, facilitando la comunicazione tra gli stakeholder e migliorando la collaborazione tra i membri del team di sviluppo.

### 3.4.2 SwaggerHub

SwaggerHub si è rivelato uno strumento fondamentale per la definizione, la documentazione e la visualizzazione delle interfacce API RESTful del progetto. La sua capacità di integrare specifiche OpenAPI ha permesso una progettazione API interattiva e collaborativa, migliorando la qualità e la coerenza dell'interfaccia di programmazione. SwaggerHub ha facilitato la creazione di una documentazione API chiara e navigabile, consentendo a sviluppatori e utenti di comprendere rapidamente le funzionalità e i punti di integrazione offerti dal sistema. L'adozione di questo strumento ha ottimizzato il flusso di lavoro di sviluppo, garantendo una maggiore allineamento tra il design delle API e le esigenze del progetto.

### 3.4.3 Draw.io

Draw.io si è distinto come uno strumento versatile per la creazione di diagrammi e flussi di lavoro all'interno del progetto. La sua interfaccia intuitiva e la vasta gamma di modelli pre-definiti hanno reso Draw.io una scelta ideale per la rappresentazione visiva di concetti complessi, processi e architetture di sistema. L'integrazione semplice con piattaforme di condivisione documenti come Google Drive e Microsoft OneDrive ha ulteriormente amplificato la sua utilità, permettendo ai membri del team di collaborare e condividere materiali visivi in modo efficiente.

# **Chapter 4**

## **Organizzazione del Lavoro con Scrum**

### **4.1 Introduzione a Scrum**

Scrum è un framework Agile focalizzato sulla gestione dei progetti e lo sviluppo del software. Basato su principi e valori definiti nel "Manifesto Agile," Scrum fornisce una struttura per aiutare i team a lavorare in modo collaborativo.

#### **4.1.1 Adozione di Scrum nel Progetto**

Nel contesto del progetto, abbiamo adottato il framework Scrum come metodologia di gestione e sviluppo per l'integrazione di nuove funzionalità. Il progetto si è sviluppato in sei iterazioni, con un numero variabile di sprint, per una durata complessiva di circa 4 mesi di lavoro. Ogni iterazione è iniziata con un Iteration Planning, durante il quale abbiamo identificato quali modifiche implementare nell'iterazione corrente. Ogni Sprint ha avuto un proprio Sprint Planning, durante il quale abbiamo suddiviso le User Stories in task assegnati ai membri del team. Abbiamo utilizzato piattaforme come Trello e Miro per tracciare i task.

## 4.2 Primo sprint

1° Sprint: 01/11/2023 - 08/11/2023		
DA COMPLETARE   7	COMPLETATI   6	PROBLEMATICA   2
Impostazione lavoro con Scrum e Sprint	Impostazione lavoro con Scrum e Sprint	Cambio documentazione durante lo sprint, da T10 G37 a T10 G40
Acquisizione materiale T10 G37	Acquisizione materiale T10 G37	Esecuzione T10 allo stato attuale
Acquisizione materiale T10 G40	Acquisizione materiale T10 G40	
Lettura documentazione e comprensione generale del sistema	Lettura documentazione e comprensione generale del sistema	
Installazione del software	Installazione del software	
Breve panoramica sul compito degli altri task	Breve panoramica sul compito degli altri task	
Esecuzione T10 allo stato attuale		
+		

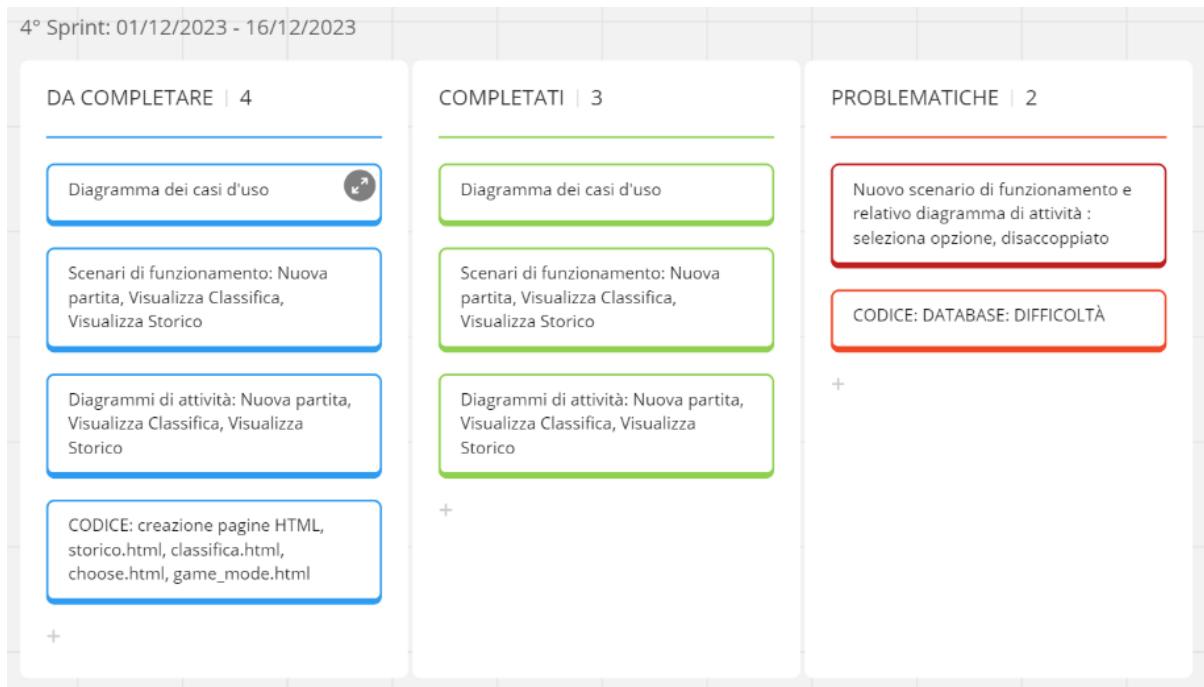
### 4.3 Secondo sprint

2° Sprint: 09/11/2023 - 16/11/2023		
DA COMPLETARE   7	COMPLETATI   7	PROBLEMATICHE   1
Acquisizione materiale T11-G41	Acquisizione materiale T11-G41	Cambio documentazione da T10 G40 a T11G41
Studio documentazione	Studio documentazione	
Installazione del software	Installazione del software	
Esecuzione T11 allo stato attuale	Esecuzione T11 allo stato attuale	
Breve verifica di funzionalità	Breve verifica di funzionalità	
Studio dei nuovi requisiti assegnati	Studio dei nuovi requisiti assegnati ↗	
Individuazione delle user stories	Individuazione delle user stories	

### 4.4 Terzo sprint

3° Sprint: 17/11/2023 - 01/12/2023		
DA COMPLETARE   3	COMPLETATI   2	PROBLEMATICHE   0
Specificazione dei requisiti funzionali e non funzionali partendo dalle user stories	Specificazione dei requisiti funzionali e non funzionali partendo dalle user stories	
Analisi preliminare, vista dei componenti	Analisi preliminare, vista dei componenti	
Bozza dell'interfaccia		

## 4.5 Quarto sprint



## 4.6 Quinto sprint

5° Sprint: 17/12/2023 - 30/12/2023		
DA COMPLETARE   7	COMPLETATI   7	PROBLEMATICA   0
DOCUMENTAZIONE: Strutturare la documentazione	DOCUMENTAZIONE: Strutturare la documentazione	+
DOCUMENTAZIONE: Definizione delle metodologie di sviluppo e degli strumenti utilizzati	DOCUMENTAZIONE: Definizione delle metodologie di sviluppo e degli strumenti utilizzati	
DOCUMENTAZIONE: Analisi del dominio Specifiche dei requisiti e storie utente, Diagramma dei casi d'uso, Scenari di funzionamento e diagrammi di attività, Diagramma di sequenza di logout	DOCUMENTAZIONE: Analisi del dominio Specifiche dei requisiti e storie utente, Diagramma dei casi d'uso, Scenari di funzionamento e diagrammi di attività, Diagramma di sequenza di logout	
CODICE: T4, Introduzione di player nel database e relative API	CODICE: T4, Introduzione di player nel database e relative API	
CODICE: T4, modifiche a Turn e model per renderli compatibili con player e per mostrare classifica e storico	CODICE: T4, modifiche a Turn e model per renderli compatibili con player e per mostrare classifica e storico	
CODICE: T2-3 rendere disponibile al database(T4) e all'interfaccia grafica (T5) il nome e l'id del giocatore registrato	CODICE: T2-3 rendere disponibile al database(T4) e all'interfaccia grafica (T5) il nome e l'id del giocatore registrato	
CODICE: T2-3 rendere disponibile al database(T4) e all'interfaccia grafica (T5) il nome e l'id del giocatore registrato	CODICE: T2-3 rendere disponibile al database(T4) e all'interfaccia grafica (T5) il nome e l'id del giocatore registrato	

## 4.7 Sesto sprint

6° Sprint: 02/01/2024-18/01/2024		
DA COMPLETARE   5	COMPLETATI   5	PROBLEMATICA   0
DOCUMENTAZIONE: introduzione al contesto, elenco dei task modificati, pattern MVC	DOCUMENTAZIONE: introduzione al contesto, elenco dei task modificati, pattern MVC	+
DOCUMENTAZIONE: Task4. Diagramma ER di partenza e Diagramma delle Classi	DOCUMENTAZIONE: Task4. Diagramma ER di partenza e Diagramma delle Classi	
DOCUMENTAZIONE: Task2-3. Diagramma delle classi	DOCUMENTAZIONE: Task2-3. Diagramma delle classi	
DOCUMENTAZIONE: Task6. Diagramma del package	DOCUMENTAZIONE: Task6. Diagramma del package	
CODICE: T-5 aggiungere funzioni per recuperare gli attributi resi disponibili nel T2-3	CODICE: T-5 aggiungere funzioni per recuperare gli attributi resi disponibili nel T2-3	

## 4.8 Settimo sprint

7° Sprint: 19/01/2024 -*****		
DA COMPLETARE   6	COMPLETATI   6	PROBLEMATICA   0
DOCUMENTAZIONE: guida all'installazione, modifiche	DOCUMENTAZIONE: guida all'installazione, modifiche	+
DOCUMENTAZIONE: Testing	DOCUMENTAZIONE: Testing	
CODICE: T-6 e T4 far si che nel database vengano salvati robot, classe e difficoltà nella tabella Turn.	CODICE: T-6 e T4 far si che nel database vengano salvati robot, classe e difficoltà nella tabella Turn.	
CODICE T5: elaborazione delle pagine html per rendere visibili robot, classe e difficoltà nello storico	CODICE T5: elaborazione delle pagine html per rendere visibili robot, classe e difficoltà nello storico	
CODICE T5: pagina allrobot.html e tutto ciò che ne consegue (report1 )	CODICE T5: pagina allrobot.html e tutto ciò che ne consegue (report1 )	
CODICE: T2-3 modifiche per recuperare nome giocatore e modifiche T5 per rendere disponibile	CODICE: T2-3 modifiche per recuperare nome giocatore e modifiche T5 per rendere disponibile	
+	+	

# Chapter 5

## Analisi del dominio

### 5.1 Definizione dei Requisiti e User Stories

Per delineare con precisione i requisiti del progetto, abbiamo adottato un approccio incentrato sulle *User Stories*, utilizzando la struttura "as an (attore) I want to (azione) so that (risultato/beneficio)". Questo metodo ci ha permesso di catturare le esigenze e le aspettative degli utenti finali in termini di funzionalità del sistema, offrendo una visione orientata agli obiettivi degli attori coinvolti.

Le User Stories identificate sono state:

1. **Selezione Opzione:** "Come studente, voglio selezionare l'opzione desiderata per accedere alla funzionalità specifica."
2. **Avvio Nuova Partita:** "Come studente, voglio scegliere una modalità di gioco scegliendo tra (1v1) o (1vTutti)."
3. **Avvio Nuova Partita 1v1:** "Come studente, voglio avviare una nuova partita contro un robot scegliendo robot e classe per giocare."
4. **Avvio Nuova Partita 1vTutti:** "Come studente, voglio avviare una nuova partita contro tutti i robot scegliendo solo la classe per giocare."
5. **Visualizza Classifica:** "Come studente, voglio visualizzare la classifica dei giocatori, ordinata per partite giocate e vinte, per comprendere la mia posizione."

6. **Visualizza Storico:** "Come studente, voglio ottenere lo storico delle mie partite, inclusi dettagli come data, durata, classe testata e robot sfidato, per analizzare le mie prestazioni."

Queste User Stories sono state analizzate per estrarne requisiti funzionali e non funzionali, consentendo di definire in modo esplicito le funzionalità attese dal sistema e i criteri di qualità correlati. In questa fase, è stata data particolare attenzione alla conformità al paradigma INVEST per assicurare che ciascuna User Story fosse:

- **Indipendente:** Per garantire stime affidabili e flessibilità di implementazione.
- **Negoziabile:** Mantenendo il focus sulla necessità fondamentale, permettendo spazio per la negoziazione dei dettagli.
- **Di Valore:** Assicurando che ogni storia apportasse un beneficio chiaro e tangibile per l'utente.
- **Stimabile:** Permettendo una pianificazione accurata e realistica del lavoro.
- **Di Dimensioni Adeguate:** Evitando storie troppo ampie o troppo granulari, per una gestione ottimale.
- **Testabile:** Includendo criteri di accettazione chiari per validare il soddisfacimento dei requisiti.

La scomposizione delle User Stories in requisiti ha fornito una base dettagliata per la progettazione dei casi d'uso, illustrando in modo preciso le interazioni tra gli utenti e il sistema. Questo approccio ha assicurato una comprensione approfondita delle funzionalità necessarie e delle aspettative degli utenti, stabilendo una solida fondazione per le successive fasi di sviluppo e progettazione del sistema.

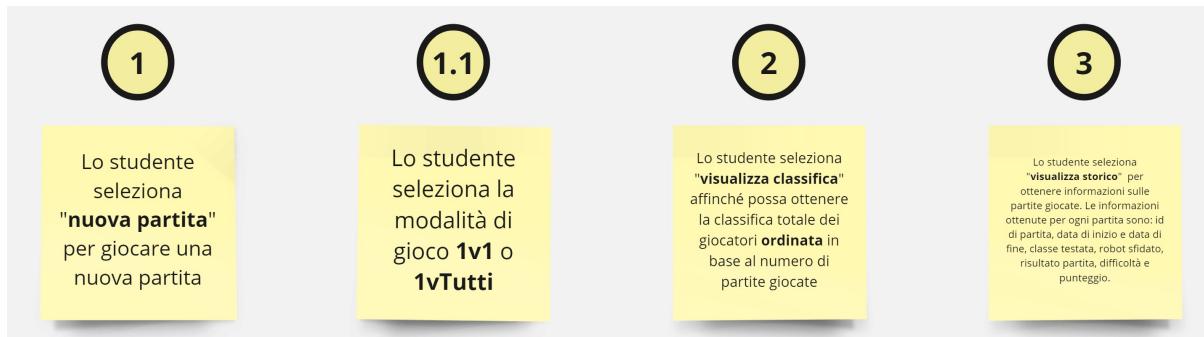


Figure 5.1: User Stories

### 5.1.1 Requisiti Funzionali

I requisiti funzionali rappresentano le funzionalità specifiche del sistema che sono necessarie per soddisfare le esigenze degli utenti. Di seguito sono elencati i requisiti funzionali derivati dalle storie utente e dalla traccia:

1. L'accesso al sistema di gioco è disponibile attraverso la procedura di login.
2. Dopo l'autenticazione, lo studente può visualizzare 3 differenti opzioni: Nuova Partita, Visualizza Storico, Visualizza Classifica.
3. Dopo l'autenticazione, lo studente può consultare lo storico delle partite da lui stesso giocate.
4. Dopo l'autenticazione, lo studente ha la possibilità di creare una nuova partita.
5. Dopo l'autenticazione, lo studente ha la possibilità di visualizzare la classifica totale di gioco.
6. Dopo aver scelto di effettuare una nuova partita, lo studente può selezionare una delle due modalità di gioco: **1v1** e **1vTutti**.
7. Dopo aver scelto di effettuare una nuova partita e dopo aver selezionato l'opzione **1vs1**, lo studente può selezionare la classe da testare ed il robot da sfidare.
8. Dopo aver scelto di effettuare una nuova partita e dopo aver selezionato l'opzione **1vsTutti**, lo studente può selezionare la classe da testare e successivamente avviare una nuova partita.

### 5.1.2 Requisiti Non Funzionali

I requisiti non funzionali definiscono le caratteristiche del sistema che non si manifestano direttamente in un caso d'uso, ma influenzano le sue proprietà generali. Di seguito sono elencati i requisiti individuati:

1. **Consistenza e Unicità dei Dati:** Il sistema deve garantire la coerenza dei dati e l'unicità degli identificativi.
2. **Usabilità:** Il sistema deve offrire un elevato livello di usabilità al fine di renderlo facilmente accessibile e fruibile da parte dell'utente.

## 5.2 Diagramma dei casi d'uso

Un **diagramma dei casi d'uso** è uno strumento UML (Unified Modeling Language) utilizzato per descrivere graficamente le interazioni tra gli utenti (o "attori") e i sistemi software per raggiungere obiettivi specifici. Ogni caso d'uso nel diagramma rappresenta una sequenza di eventi che il sistema eseguirà in risposta a un'azione dell'utente. Questi diagrammi sono essenziali per comprendere le funzionalità del sistema dal punto di vista degli utenti finali, e forniscono una guida visiva che aiuta a identificare e definire i requisiti del software in termini di comportamenti e interazioni esterne.

Il diagramma dei casi d'uso, come mostrato in figura, è stato sviluppato per delineare le interazioni tra gli studenti (precedentemente autenticati) e il sistema. Questo strumento UML è fondamentale per visualizzare le funzionalità del sistema dal punto di vista dell'utente e definire come queste interagiscono con i componenti del sistema.

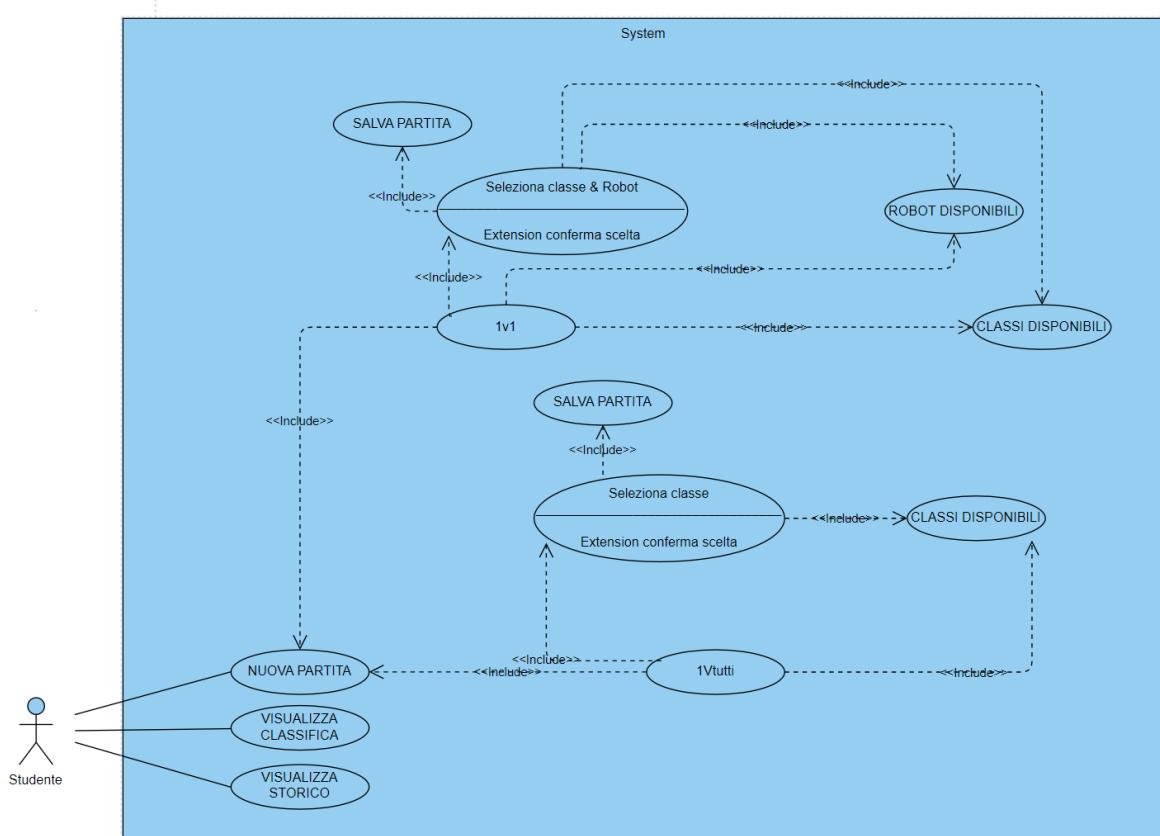


Figure 5.2: Diagramma dei Casi d'Uso

Il diagramma identifica le seguenti azioni principali disponibili per lo studente:

- **Nuova Partita:** Lo studente può iniziare una nuova partita, che è il caso d'uso centrale e include la selezione di una classe e di un robot come sotto-funzioni necessarie.
- **Visualizza Classifica:** Questa funzione permette allo studente di visualizzare la classifica dei giocatori, fornendo un feedback competitivo e un incentivo al miglioramento.
- **Visualizza Storico:** Consente allo studente di visualizzare lo storico delle partite giocate, permettendo un'analisi retrospettiva delle prestazioni.

Ogni azione principale è collegata a sotto-azioni o estensioni che rappresentano passaggi aggiuntivi o alternative. Ad esempio, il caso d'uso "Nuova Partita" include:

- **1v1:** Una decisione che permette allo studente di giocare una partita scegliendo una classe e un robot.
- **1vTutti:** Una decisione che permette allo studente di giocare una partita scegliendo una classe.
- **Salva Partita:** Un'azione che viene eseguita al termine di una partita per registrare i risultati.
- **Selezione Classe Robot:** Rappresenta la scelta dello studente di una particolare classe per il robot da utilizzare nella partita.
- **Extension Conferma Scelta:** Una decisione opzionale che permette allo studente di confermare o modificare la selezione fatta.

Le relazioni di inclusione («include») indicano che un'azione è una parte necessaria di un'altra azione, mentre le relazioni di estensione («extend») indicano un'azione che può avvenire in aggiunta al flusso principale.

## 5.3 Scenari di funzionamento e diagrammi di attività

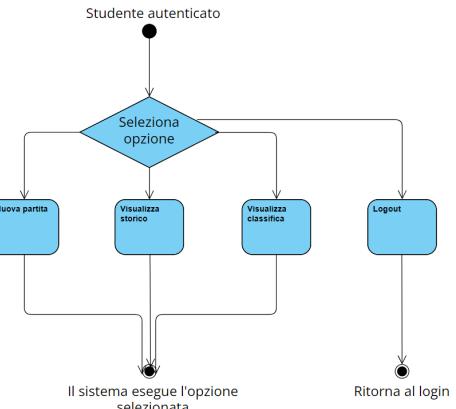
Dalle storie utente sono starti sviluppati i seguenti scenari di funzionamento, ognuno con portata, livello, attore principale, descrizione, precondizioni, scenario principale e flusso alternativo.

### 5.3.1 Scenario 1 : Seleziona opzione

Lo studente una volta autenticato può scegliere se giocare una nuova partita, visualizzare lo storico o visualizzare la classifica.

SCENARIO	<input checked="" type="checkbox"/> SELEZIONA OPZIONE
PORTATA	Applicazione gioco
LIVELLO	Obiettivo utente
ATTORE PRINCIPALE	Studente
DESCRIZIONE	una volta autenticato può scegliere se giocare una nuova partita, visualizzare lo storico o visualizzare la classifica.
PRECONDIZIONI	Lo studente deve essere autenticato correttamente
SCENARIO PRINCIPALE (flusso base)	1. Lo studente, dopo essersi identificato, visualizza le opzioni di gioco disponibili: Nuova partita, Visualizza Classifica, Visualizza Storico.
FLUSSO ALTERNATIVO	1.e. Lo studente non vuole effettuare nessuna scelta 1. Lo studente preme il pulsante di Logout per ritornare alla pagina di Login

(a) Scenario



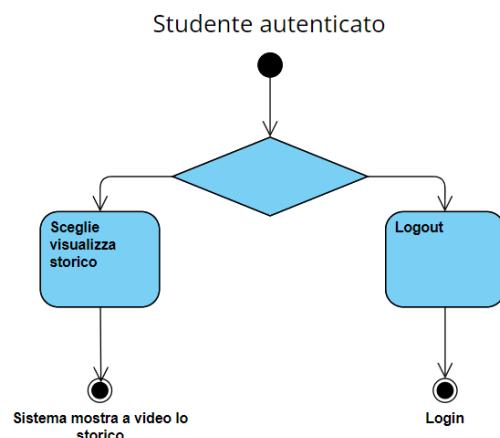
(b) Diagramma d'attività

### 5.3.2 Scenario 2 : Visualizza storico

Lo studente, previamente autenticato, vuole visualizzare lo storico

SCENARIO	<input checked="" type="checkbox"/> VISUALIZZA STORICO
PORTATA	Applicazione gioco
LIVELLO	Obiettivo utente
ATTORE PRINCIPALE	Studente
DESCRIZIONE	Lo studente vuole visualizzare lo storico
PRECONDIZIONI	Lo studente è stato autenticato
SCENARIO PRINCIPALE (flusso base)	1. L'utente sceglie l'opzione visualizza storico. 2. Il sistema fornisce allo studente lo storico.
FLUSSO ALTERNATIVO	1.e. Lo studente non vuole effettuare nessuna scelta 1. Lo studente preme il pulsante di Logout per ritornare alla pagina di Login

(c) Scenario



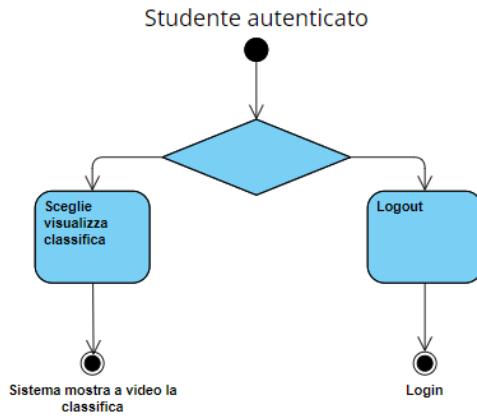
(d) Diagramma d'attività

### 5.3.3 Scenario 3 : Visualizza classifica

Lo studente, previamente autenticato, vuole visualizzare la classifica.

SCENARIO	VISUALIZZA CLASSIFICA
PORTATA	Applicazione gioco
LIVELLO	Obiettivo utente
ATTORE PRINCIPALE	Studente
DESCRIZIONE	Lo studente vuole visualizzare la classifica
PRECONDIZIONI	Lo studente è stato autenticato
SCENARIO PRINCIPALE (flusso base)	1. L'utente sceglie l'opzione visualizza classifica. 2. Il sistema fornisce allo studente la classifica.
FLUSSO ALTERNATIVO	1.e. Lo studente non vuole effettuare nessuna scelta 1. Lo studente preme il pulsante di Logout per ritornare alla pagina di Login

(e) Scenario



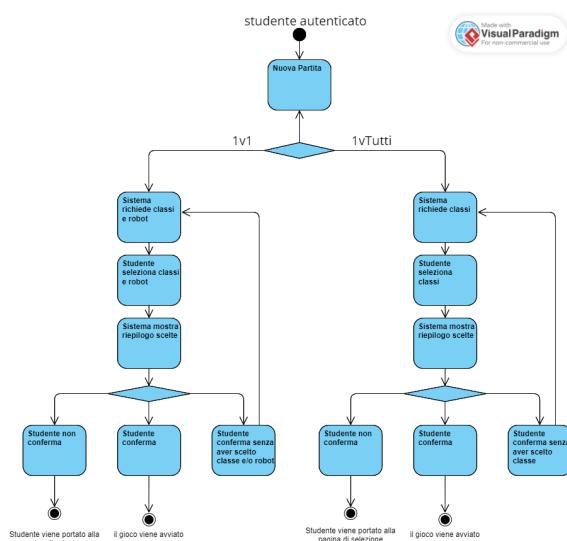
(f) Diagramma d'attività

### 5.3.4 Scenario 4 : Seleziona opzione di gioco

Lo studente, previamente autenticato dopo aver selezionato l'opzione nuova partita , seleziona le classi e i roboot per giocare.

SCENARIO	SELEZIONA NUOVA PARTITA
PORTATA	Applicazione gioco
LIVELLO	Obiettivo utente
ATTORE PRINCIPALE	Studente
DESCRIZIONE	Lo studente, previamente autenticato dopo aver selezionato l'opzione nuova partita , seleziona la modalità di gioco
PRECONDIZIONI	Lo studente deve essere autenticato correttamente e deve aver scelto l'opzione Nuova Partita
SCENARIO PRINCIPALE (flusso base)	1. Lo studente, dopo essersi identificato e aver scelto di iniziare una nuova partita, visualizza la scelta tra una partita 1vs1 e 1vsTutti 2. Il sistema richiede l'elenco delle classi (e dei robot se si è scelto 1v1) e li mostra allo studente. 3. Lo studente seleziona la classe (e il robot se si è scelto 1v1) che gli interessa sfidare. 4. Il sistema mostra un riepilogo delle scelte. 5. Lo studente conferma la sua scelta.
FLUSSO ALTERNATIVO	1.e. Lo studente non conferma. 1. Il sistema riporta lo studente alla scelta delle 3 opzioni di gioco: Nuova Partita, Visualizza Classifica, Visualizza Storico 2.e. Lo studente conferma la propria scelta senza aver selezionato correttamente nessuna classe e/o nessun robot da testare. 1. Il sistema richiede allo studente di effettuare una selezione valida. 2. Lo studente procede come da flusso base.

(g) Scenario



(h) Diagramma d'attività

# Chapter 6

## Introduzione al contesto

Nel capitolo precedente, è stata condotta un'analisi approfondita del dominio del problema, focalizzandosi sui requisiti, sia funzionali che non funzionali, e sui potenziali scenari operativi del software. In questa sezione, tuttavia, si orienta l'attenzione verso l'elaborazione di diagrammi che dettagliano l'architettura del sistema sotto molteplici prospettive. Questi diagrammi si sono rivelati essenziali per guidare lo sviluppo del codice sia del front-end che del back-end.

Al fine di soddisfare pienamente il requisito delineato nel capitolo introduttivo, si è proceduto con un'estensiva serie di modifiche che hanno interessato molteplici aspetti del progetto. Le attività coinvolte in questo comprensivo sforzo di sviluppo comprendono modifiche ai seguenti task:

**Task T2-3:** [Login]

**Task T4:** [Database]

**Task T5:** [Creazione e inizializzazione partite]

**Task T6:** [interfaccia grafica Logica di gioco ed editor]

Ogni task è stato oggetto di una revisione e un aggiornamento dettagliato, riflettendo un impegno significativo sul fronte del miglioramento e dell'adeguamento funzionale del software. Questo ampio intervento ha comportato un lavoro intensivo su diverse sezioni del codice e dell'architettura, assicurando che ogni componente fosse allineato con gli obiettivi di progetto e rispondesse efficacemente ai requisiti identificati.

## 6.1 Vista dei componenti

La *vista dei componenti* nell’architettura del software dettaglia l’organizzazione e le interazioni dei componenti modulari, essenziali per la modularità, riusabilità, encapsulamento, sostituibilità, scalabilità e la comunicazione del sistema. Questa vista semplifica la gestione della complessità e incrementa la qualità architetturale, migliorando lo sviluppo, il testing e la manutenzione, oltre a facilitare la riusabilità e l’aggiornamento dei componenti.

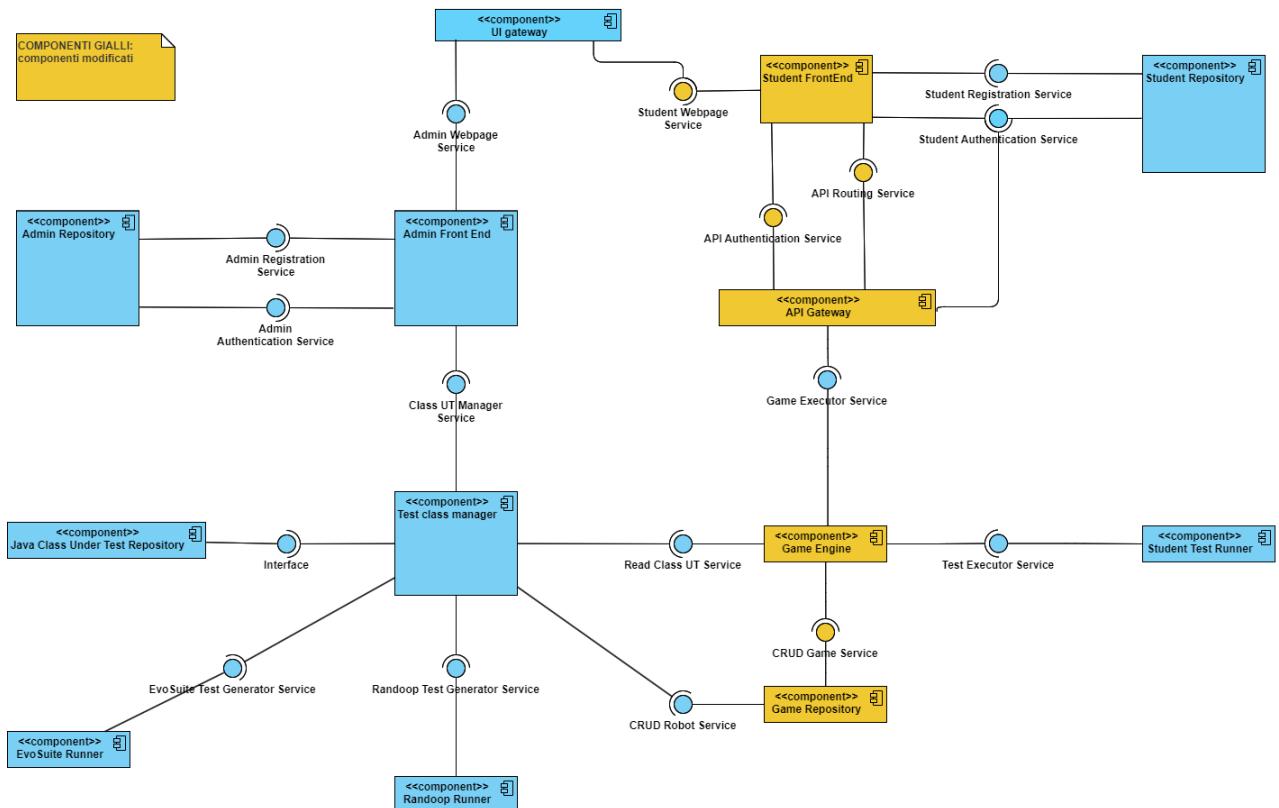


Figure 6.1: Vista dei componenti

Componente	Descrizione	Task
UI Gateway	Agisce come dispatcher per le richieste UI, utilizzando Nginx per il routing verso i servizi di interfaccia utente.	None
API Gateway	Gestisce le richieste API, implementando sicurezza, autenticazione e routing con Spring Boot e Netflix Zuul.	None
Student Front End	Interfaccia utente per gli studenti, sviluppata con Spring Boot e CodeMirror per l'editor di testo.	T2-3, T5
Student Repository	Gestisce l'autenticazione degli studenti e mantiene i dati degli utenti con Spring Boot e MySQL.	T2-3
Admin Front End	Fornisce interfacce utente per gli amministratori, realizzate con Spring Boot.	T1
Admin Repository	Si occupa dell'autenticazione degli amministratori e gestisce i dati con Spring Boot e MongoDB.	T1
Game Engine	Gestisce la logica di gioco, interfacciandosi con vari servizi e componenti, sviluppato con Spring Boot.	T5, T6
Student Test Runner	Compila il codice di test degli studenti e calcola la copertura con Maven, JaCoCo e Spring Boot.	T7
Test Class Manager	Gestisce le classi giocabili e interagisce con il Game Repository, sviluppato con Spring Boot.	T1
Java Class Under Test Repository	Persiste le classi inviate dagli amministratori utilizzando MongoDB e Spring Boot.	T1
Randoop Runner	Genera test con Randoop e calcola la copertura con Emma, salvando i test localmente.	T9
EvoSuite Runner	Genera test con EvoSuite, salvando test e report in CSV, sviluppato con EvoSuite.	T8
Game Repository	Archivia storico delle partite, classi giocabili e risultati dei robot, realizzato con Chi e PostgreSQL.	T4

Table 6.1: Descrizione dei Componenti

Dall'analisi della vista dei componenti attuale, creata utilizzando Visual Paradigm, è stata effettuata un'attenta valutazione delle modifiche necessarie per soddisfare i requisiti del task in oggetto. La vista fornisce una rappresentazione chiara dell'architettura software del sistema di gioco, identificando in modo esplicito le relazioni e le dipendenze tra i vari componenti.

È stato identificato che per implementare la funzionalità di visualizzazione della classifica dei giocatori e dello storico delle partite, sono richieste modifiche sostanziali a due componenti

principali: il front-end studente e il Game Repository. Il front-end studente, che rappresenta l’interfaccia utente attraverso cui gli studenti interagiscono con il gioco, necessita di essere esteso per includere nuove sezioni per la visualizzazione della classifica e dello storico delle partite. Questo coinvolgerà l’aggiunta di nuovi elementi di interfaccia utente e la possibile ristrutturazione di quelli esistenti per accogliere le nuove funzionalità in un’esperienza utente coerente e intuitiva.

Parallelamente, il Game Repository, il componente responsabile della gestione dei dati del gioco, richiederà l’integrazione di nuove operazioni CRUD (Create, Read, Update, Delete) che supporteranno la persistenza e il recupero delle informazioni dettagliate relative alla classifica dei giocatori e allo storico delle partite. Questo implicherà modifiche ai modelli di dati esistenti o l’introduzione di nuovi schemi per catturare e memorizzare i dati necessari.

Le modifiche a questi due componenti chiave saranno cruciali per realizzare le nuove funzionalità richieste, garantendo che l’interazione tra l’interfaccia utente e il back-end sia fluida e che i dati siano presentati e gestiti in modo efficace. La visione attuale dei componenti fornisce un’ottima base di partenza per pianificare e eseguire queste modifiche con precisione, garantendo che l’evoluzione del sistema sia allineata con le esigenze degli utenti e gli obiettivi del progetto.

Il componente ApiGateway è stato aggiornato per arricchire le informazioni contenute nel token JWT, includendo ora anche il nome dell’utente, oltre ai dati precedentemente memorizzati. A supporto di questa estensione, sono stati introdotti due nuovi endpoint: il primo consente di recuperare il nome del giocatore direttamente dal token, mentre il secondo permette di ottenere l’ID del giocatore, anch’esso salvato nel token. Queste integrazioni sono state fondamentali per soddisfare le esigenze del task assegnato, in particolare per rendere la classifica e lo storico delle partite il più dettagliati e personalizzati possibile.

## 6.2 Considerazioni sull'implementazione: pattern MVC

L'architettura del software che stiamo migliorando è stata strutturata secondo il pattern Model-View-Controller (MVC), una scelta che si è dimostrata vantaggiosa per diversi motivi. Originariamente, il software era già strutturato in una maniera che permetteva una distinzione chiara delle responsabilità e, proseguendo su questa linea con l'uso di Spring Boot, abbiamo potuto massimizzare l'efficacia del nostro lavoro di evoluzione e manutenzione del codice esistente.

Il pattern MVC segmenta le responsabilità dell'applicativo in tre componenti distinte:

- **Modello (Model):** Gestisce i dati e la logica di business dell'applicazione, fungendo da rappresentazione dei dati e delle regole di gestione.
- **Vista (View):** Offre la rappresentazione visuale dei dati, ovvero l'interfaccia utente che gli utenti interagiscono per visualizzare e inserire dati.
- **Controller:** Agisce come intermediario tra Modello e Vista, gestendo la logica di controllo e coordinando le risposte alle azioni dell'utente.

La scelta del pattern MVC per il software esistente è stata motivata dalla sua efficacia nel favorire la separazione dei compiti, rendendo così il codice più modulare e riutilizzabile. Questa separazione consente agli sviluppatori di lavorare su singoli aspetti dell'applicazione (come l'interfaccia utente o la logica di business) in maniera isolata, riducendo la complessità e migliorando la manutenibilità.

Inoltre, il pattern MVC è particolarmente adatto per applicazioni web come la nostra, dove la scalabilità è un requisito importante. La sua ampia adozione nei principali framework e tecnologie per lo sviluppo web garantisce accesso a un ecosistema consolidato di strumenti e risorse, che supporta l'efficienza nello sviluppo e l'integrazione con altre tecnologie e servizi.

A confronto con altri pattern, come il Boundary-Control-Entity (BCE), MVC offre una maggiore flessibilità, soprattutto nella gestione delle interfacce utente. Il pattern BCE, pur essendo valido, si è rivelato meno adattabile alle nostre esigenze di evoluzione del software, specialmente considerando la necessità di frequenti aggiornamenti e l'introduzione di nuove funzionalità. Il pattern MVC, con la sua chiara separazione e il suo forte supporto da parte della comunità di sviluppatori, rende più agevole la comprensione e l'espansione del nostro software.

### 6.3 Refactoring del Codice

Prima di procedere con le modifiche nei task T2-3, T4, T5 e T6, abbiamo intrapreso un'attenta fase di refactoring del codice. Il refactoring è una pratica di sviluppo software volta a ristrutturare il codice esistente senza modificarne il comportamento esterno. Questo processo è fondamentale per migliorare la leggibilità, la manutenibilità e l'efficienza del codice.

Il nostro refactoring ha incluso le seguenti attività:

- Modifica dei nomi di alcune variabili per renderli più esplicativi e descrittivi.
- Aggiunta di commenti dettagliati nel codice per chiarire la logica e la sequenza delle azioni

Queste azioni hanno migliorato notevolmente la qualità del codice, rendendolo più accessibile per le future fasi di sviluppo e manutenzione. Di seguito, elenchiamo i file principali che sono stati oggetto di refactoring:

- T2-3: \_ProgettoSAD\_T11-G41-master\_T23-G1\_src\_main\_java\_com\_example\_db\_setup\_Controller.java
- T4: \_ProgettoSAD\_T11-G41-master\_T4-G18\_api\_robot
- T4: \_ProgettoSAD\_T11-G41-master\_T4-G18\_api\_round
- T4: \_ProgettoSAD\_T11-G41-master\_T4-G18\_api\_turn
- T4: \_ProgettoSAD\_T11-G41-master\_T4-G18\_model\_model.go
- T5: \_ProgettoSAD\_T11-G41-master\_T5-G2\_t5\_src\_main\_java\_com\_g2\_t5\_GameDataWriter.java
- T5: \_ProgettoSAD\_T11-G41-master\_T5-G2\_t5\_src\_main\_java\_com\_g2\_t5\_GuiController.java
- T6: \_ProgettoSAD\_T11-G41-master\_T6-G12\_T6\_src\_main\_java\_com\_example\_T6\_MyController.java

Questo refactoring ha gettato le basi per le modifiche successive, assicurando che il codice su cui si costruisce sia solido e comprensibile.

## 6.4 Diagrammi di sequenza

Un diagramma di sequenza è un diagramma UML utilizzato per visualizzare l'ordine cronologico delle interazioni tra oggetti in un processo specifico. Mostra come gli oggetti scambiano messaggi e collaborano per svolgere una funzione. I diagrammi di sequenza sono importanti perché chiariscono la complessità delle interazioni software, aiutano a individuare problemi di progettazione e facilitano la comunicazione tra sviluppatori e stakeholder. Sono essenziali per analizzare, progettare e documentare i flussi operativi all'interno dei sistemi software.

### 6.4.1 Visualizza Storico

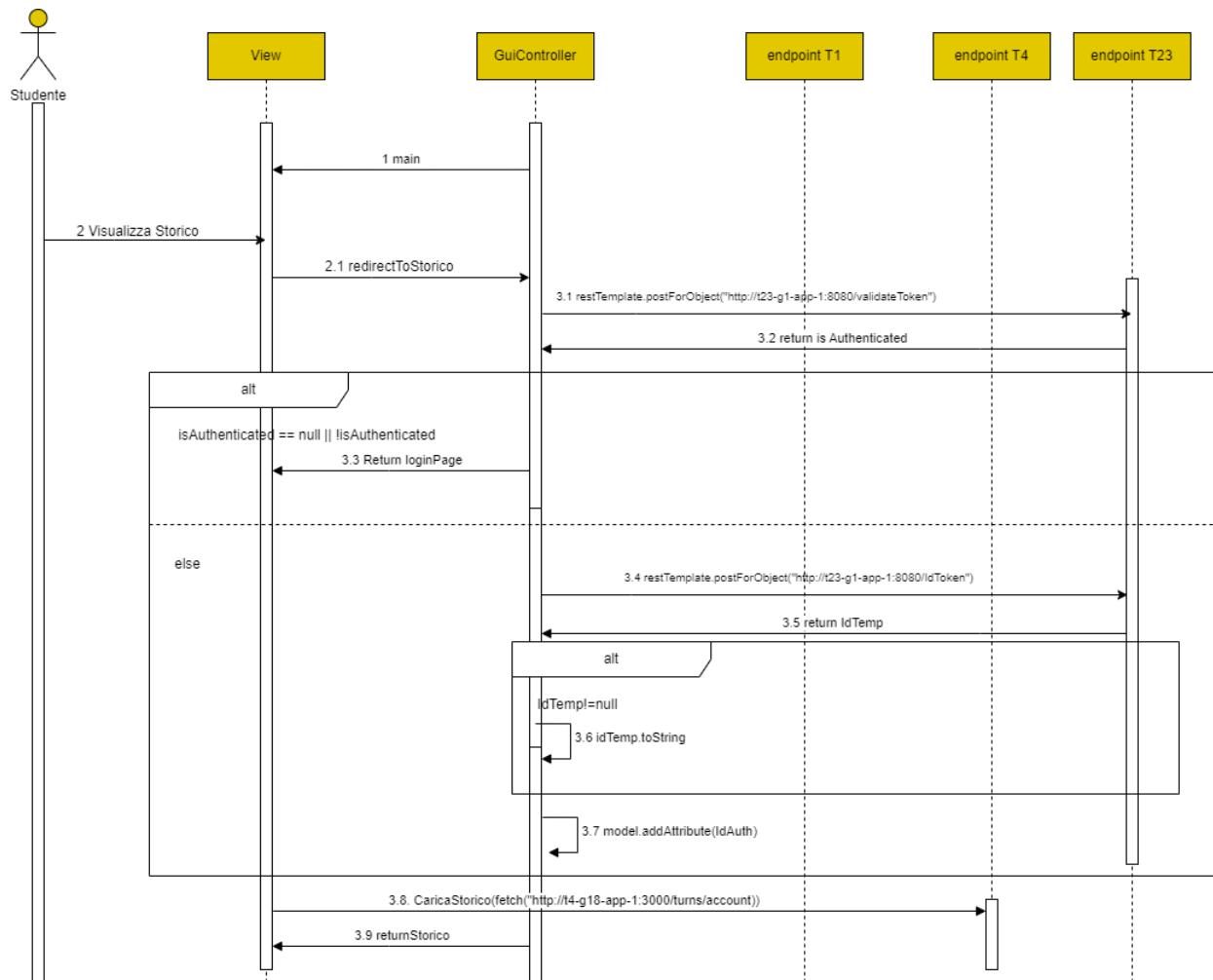


Figure 6.2: Diagramma di sequenza: Visualizza storico

Il diagramma di sequenza dettaglia le operazioni eseguite dall'applicazione quando un *Studente* richiede di accedere alla propria cronologia di gioco:

1. L'interazione inizia quando lo *Studente* seleziona l'opzione per visualizzare lo storico nel sistema di *View*.
2. Il *GuiController* riceve la richiesta e tenta un reindirizzamento alla pagina dello storico.
3. Se l'autenticazione fallisce, lo studente viene reindirizzato alla pagina di login.
4. Una volta autenticato, il *GuiController* invoca una serie di richieste POST verso gli endpoint T1, T4 e T23.
5. Queste richieste sono volte a validare il token JWT e a recuperare un ID.
6. Infine, lo storico delle partite viene recuperato dal servizio associato all'endpoint T23 utilizzando l'ID.

Questa sequenza di eventi assicura che solo gli utenti autenticati possano accedere alle informazioni dello storico, mantenendo la sicurezza e l'integrità dei dati.

### 6.4.2 Visualizza Classifica

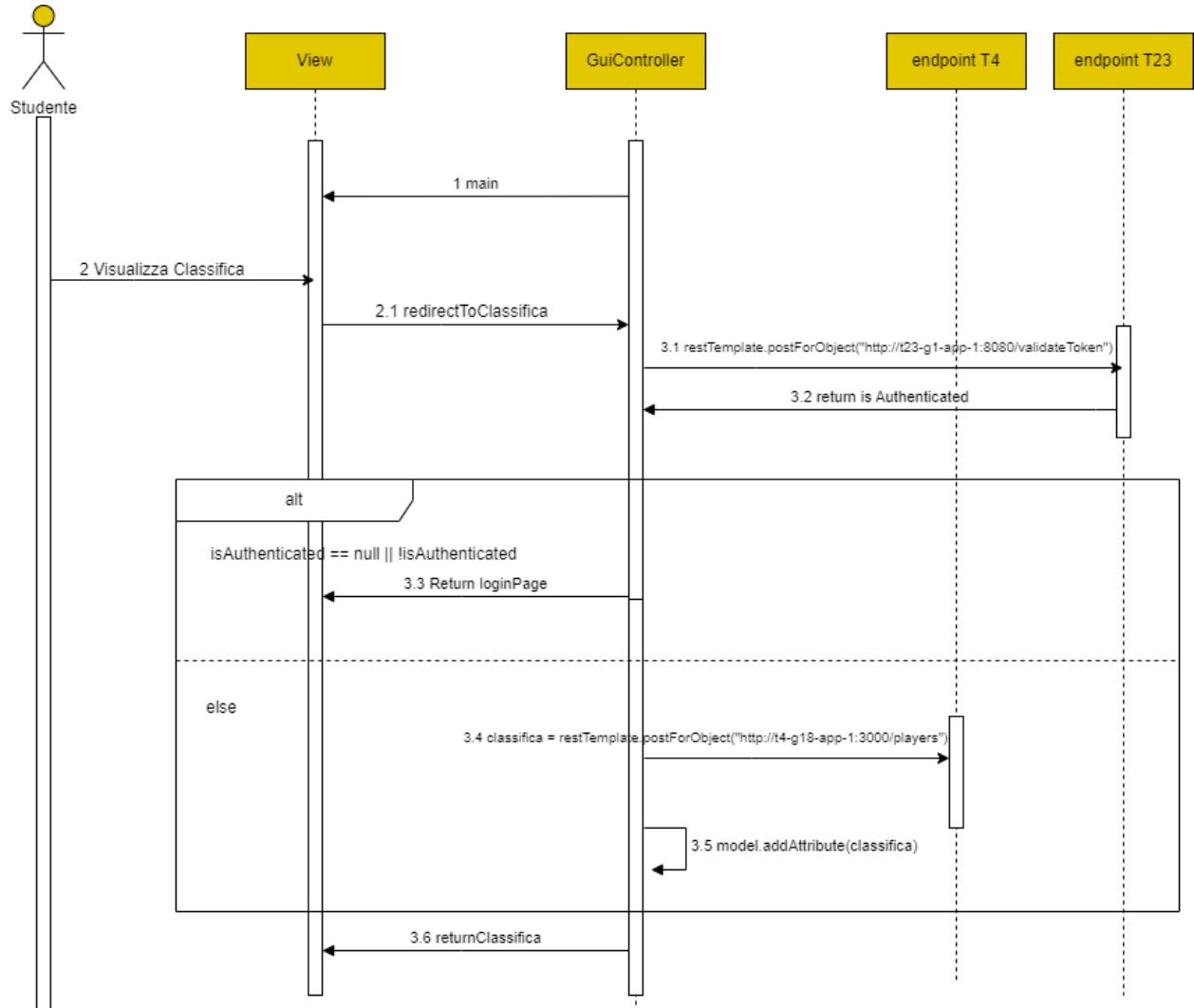


Figure 6.3: Diagramma di sequenza: Visualizza Classifica

Il diagramma di sequenza mostra le operazioni eseguite quando uno studente tenta di visualizzare la classifica. Il processo inizia con la View che invia una richiesta al GuiController. A seconda dello stato di autenticazione dell'utente, il controller può reindirizzare alla pagina di login o procedere alla richiesta dei dati della classifica.

- Il controller verifica l'autenticazione dell'utente.
- Se autenticato, interroga gli endpoint T4 e T23 per ottenere i dati della classifica.
- I dati recuperati vengono poi passati alla View per essere visualizzati all'utente.
- Se l'utente non è autenticato, viene reindirizzato alla pagina di login.

### 6.4.3 Nuova Partita

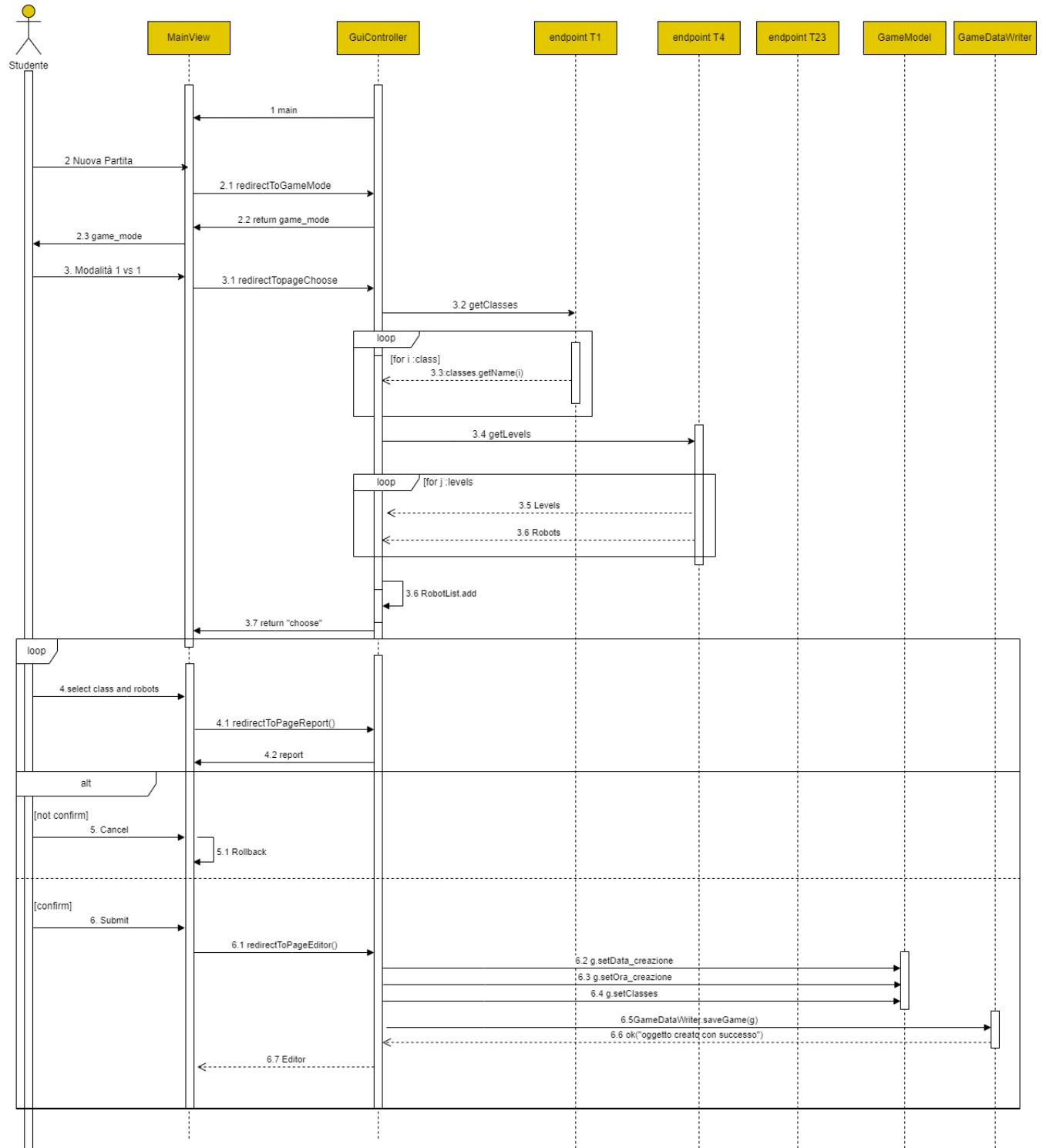


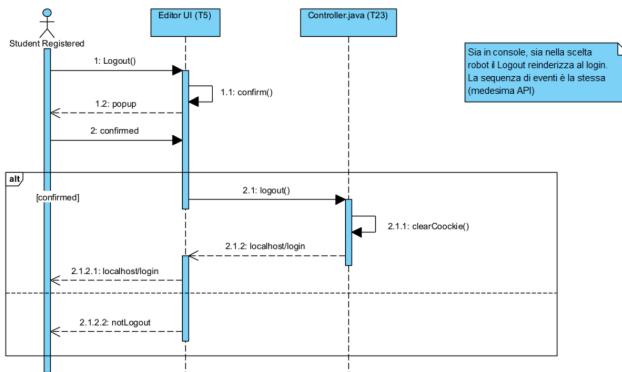
Figure 6.4: Diagramma di sequenza: Nuova Partita 1v1

Il diagramma di sequenza illustra il flusso di eventi innescati da un utente che intende avviare una nuova partita 1v1 all'interno dell'applicazione. Inizialmente, l'utente seleziona l'opzione per una nuova partita nella MainView. Questa azione porta l'UI Controller a richiedere la modalità di gioco, che in questo caso è la modalità 1v1. Successivamente, l'utente è guidato attraverso una serie di scelte: selezione delle classi e dei robot da una lista fornita dagli endpoint T1, T4 e T23, che l'UI Controller recupera iterando su liste di possibili classi e livelli di difficoltà dei robot.

Una volta fatta la selezione, l'utente ha la possibilità di confermare o annullare la scelta. Se la scelta è confermata, l'UI Controller invia i dettagli della selezione al GameModel, che a sua volta istruisce il GameDataWriter a salvare la nuova partita. Se tutto procede senza errori, l'utente viene reindirizzato all'editor di gioco per iniziare la partita.

È rilevante sottolineare che il diagramma non mostra i controlli di autenticazione per mantenere una rappresentazione semplificata; tuttavia, nella realtà applicativa, ogni interazione con l'UI Controller sarebbe preceduta da un controllo di autenticazione per garantire che l'utente sia autorizzato ad eseguire l'azione richiesta. Questa misura di sicurezza è cruciale per proteggere l'integrità del gioco e la privacy degli utenti.

#### 6.4.4 Logout



Il diagramma di sequenza incluso riflette le interazioni definite nella documentazione originale del task T5. Benché questo diagramma non abbia subito modifiche durante il nostro intervento, si riconosce l'importanza della sua inclusione nella documentazione attuale. La sua presenza è fondamentale per illustrare le dinamiche operative del software e per garantire una comprensione del comportamento del sistema.

#### Descrizione del Diagramma di Sequenza del Processo di Logout:

1. **Lo studente (utente)** innesca un'azione di logout nell'**Editor UI (T5)**.
2. L'**Editor UI** presenta un popup all'utente per confermare l'azione di logout.
3. Se l'utente conferma (come indicato dal frammento combinato "alt" con la condizione **[confirmed]**), si verificano i seguenti passaggi:
  - (a) La conferma (**2: confirmed**) viene inviata all'Editor UI.
  - (b) L'Editor UI chiama il metodo di logout sul **Controller** ('**1.1: confirm()**' seguito da '**2.1: logout()**').
  - (c) Il Controller esegue il metodo '**clearCookie()**' ('**2.1.1: clearCookie()**').
  - (d) L'utente viene reindirizzato alla pagina **localhost** ('**2.1.2: localhost/login**').
4. Se l'utente non conferma il logout, c'è un flusso隐式 che comporterebbe il mantenimento dell'accesso da parte dell'utente, come indicato dal percorso alternativo ('**2.1.2.2: notLogout**').

# Chapter 7

## Modifiche al Task 2-3

### 7.1 Diagramma delle classi

Il **diagramma delle classi** è essenziale nello sviluppo software, illustrando in modo chiaro le relazioni tra le classi del sistema. Funziona come un mezzo per delineare la struttura del software e per migliorare la comunicazione e comprensione del progetto tra sviluppatori e stakeholder. Il diagramma fornisce una rappresentazione dettagliata delle classi, mettendo in evidenza le loro proprietà e funzioni e risultando fondamentale per la riuscita del progetto.

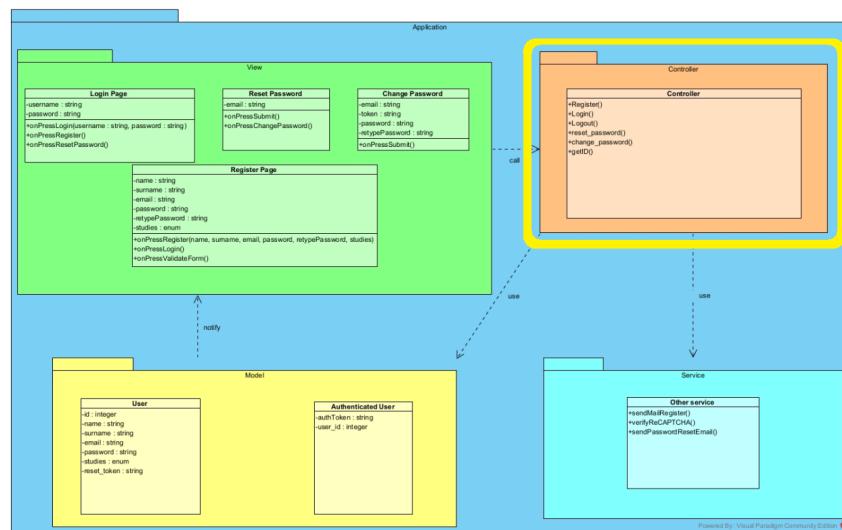


Figure 7.1: Diagramma delle Classi T2-3

## 7.2 Controller

Il punto di partenza per le modifiche al task T2-3 è stato il diagramma delle classi. Il componente su cui si è lavorato è Controller, il nuovo componente è

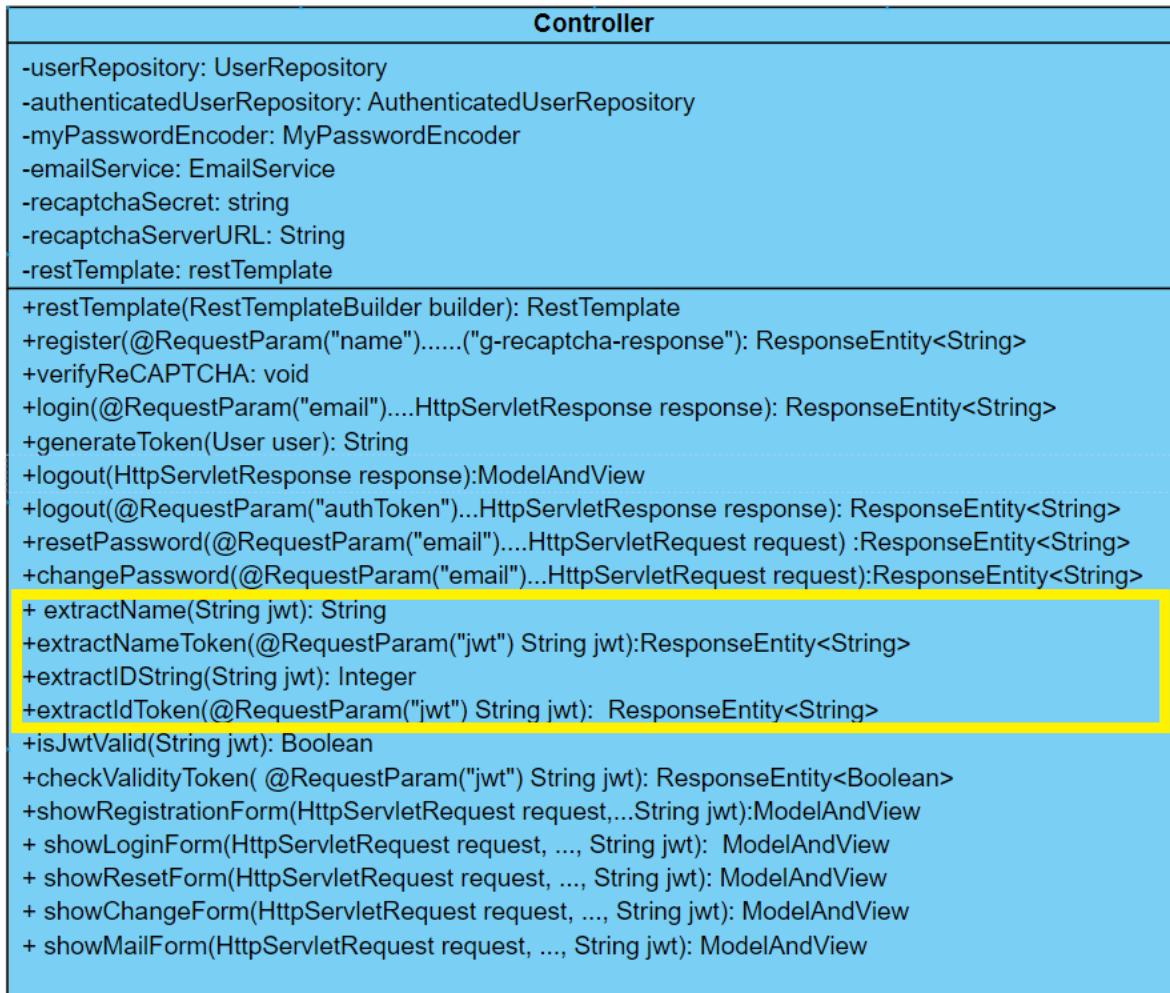


Figure 7.2: Diagramma delle Classi T2-3: Controller Aggiornato

Nel controller sono state integrate quattro nuove funzioni:

- `extractName`: estraie il nome dell'utente dal token JWT di autenticazione.
- `extractNameToken`: Questa è una funzione controller in un'applicazione Spring Boot che gestisce una richiesta POST al percorso "/nameToken".
- `extractIDString`: ottiene la stringa identificativa (ID) del giocatore dal token.
- `extractIdToken`: Questa è una funzione controller in un'applicazione Spring Boot che

gestisce una richiesta POST al percorso "/IdToken".

Queste funzioni sono fondamentali per il recupero delle informazioni dell'utente autenticato, come il nome e l'ID, dal token di autenticazione. Questi dati sono poi utilizzati per associare correttamente le azioni all'utente corrispondente nelle funzionalità di classifica e storico delle partite.

Un'altra modifica è stata fatta alla funzione generateToken per fare in modo che nel token JWT fosse presente anche il nome del giocatore.

### 7.3 Codice: Controller

Nel Task T23, abbiamo apportato modifiche significative alla sezione del codice responsabile della generazione dei token. Il nostro obiettivo principale è stato quello di aggiungere un parametro Name al token JWT e implementare le funzionalità necessarie per estrarre e renderlo disponibile in altre parti dell'applicazione.

Nello specifico, il primo metodo presentato nell'immagine seguente è progettato per recuperare il nome dal JWT. Questa funzione analizza il token, verifica la sua autenticità e, se valido, estrae il valore associato al campo Name.

Il secondo metodo mostrato riguarda un controller. Questo controller espone un endpoint HTTP POST, permettendo agli utenti di utilizzare la funzionalità di estrazione del nome. Quando un client invia una richiesta a questo endpoint con un JWT, il controller chiama il metodo di estrazione per recuperare il nome dal token e restituisce il risultato all'utente.

Attraverso queste modifiche, abbiamo migliorato la nostra applicazione rendendo più accessibile e gestibile l'informazione contenuta nel token JWT, in particolare il nome dell'utente. Queste funzionalità sono fondamentali per garantire che le parti interessate possano facilmente accedere e utilizzare i dati importanti codificati nel token.

```

● 207     public static String generateToken(User user) {
208         Instant now = Instant.now();
209         Instant expiration = now.plus(1, ChronoUnit.HOURS);
210
211         String token = Jwts.builder()
212             .setSubject(user.getEmail())
213             .claim(name:"name",user.getName()) //aggiunto per ottenere il nome
214             .setIssuedAt(Date.from(now))
215             .setExpiration(Date.from(expiration))
216             .claim(name:"userId", user.getID())
217             .claim(name:"role", value:"user")
218             .signWith(SignatureAlgorithm.HS256, base64EncodedSecretKey:"mySecretKey")
219             .compact();
220
221         return token;
222     }

```

Figure 7.3: Generate Token

```

319     public String extractName(String jwt){
320         try{
321             Claims c = Jwts.parser().setSigningKey(base64EncodedKeyBytes:"mySecretKey").parseClaimsJws(jwt).getBody();
322             return c.get(claimName:"name", requiredType:String.class);
323         } catch (Exception e){
324             e.printStackTrace();
325         }
326         return null;
327     }
328
329     @PostMapping("/nameToken")
330     public ResponseEntity<String> extractNameToken(@RequestParam("jwt") String jwt) {
331         String name = extractName(jwt);
332         if (name != null) {
333             return ResponseEntity.ok(name);
334         } else {
335             return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Impossibile estrarre il nome dal token.");
336         }
337     }
338

```

Figure 7.4: Metodi per estrarre il Name dal Token

Durante il processo di sviluppo, ci siamo resi conto che, nonostante il token JWT includesse l'ID dell'utente registrato, mancava un meccanismo dedicato all'estrazione di tale ID. Pertanto, si è resa necessaria l'implementazione di ulteriori metodi per colmare questa lacuna.

Abbiamo quindi introdotto due nuovi metodi per affrontare questa esigenza. Il primo è specificamente progettato per estrarre l'ID dell'utente dal token JWT. Questo metodo analizza il token, ne verifica l'integrità e, se giudicato valido, recupera l'ID dell'utente associato.

Il secondo metodo aggiunto è un controller che espone un endpoint HTTP, simile a quello precedentemente descritto per il nome dell'utente. Questo endpoint permette agli utenti di richiedere l'estrazione dell'ID dall'interno del loro token JWT. Quando riceve una tale richiesta, il controller esegue il metodo di estrazione dell'ID e fornisce il risultato come risposta al

client.

Questa integrazione migliora ulteriormente la flessibilità e l'efficienza della nostra applicazione, consentendo un facile accesso e utilizzo dei dati critici contenuti nei token JWT, come l'ID dell'utente, che è spesso fondamentale per le operazioni di autenticazione e autorizzazione.

```

339     //Funzione per l'estrazione dell'ID dal Token
340     public Integer extractIDString(String jwt){
341         try{
342             Claims c = Jwts.parser().setSigningKey(base64EncodedKeyBytes:"mySecretKey").parseClaimsJws(jwt).getBody();
343             return c.get(claimName:"userId", requiredType:Integer.class);
344         } catch (Exception e){
345             e.printStackTrace();
346         }
347         return null;
348     }
349
350     //endpoint per estrarre l'ID dal Token
351     @PostMapping("/IdToken")
352     public ResponseEntity<String> extractIdToken(@RequestParam("jwt") String jwt) {
353         Integer userId = extractIDString(jwt);
354         if (userId != null) {
355             return ResponseEntity.ok(userId.toString());
356         } else {
357             return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Impossibile estrarre l'Id dal token.");
358         }
359     }

```

Figure 7.5: Metodi per estrarre l'ID dal Token

# Chapter 8

## Modifiche al Task 4

### 8.1 Diagramma ER di partenza

Per soddisfare con precisione i requisiti del nostro Task, che prevedono l'introduzione di nuove modalità di gioco, la visualizzazione di classifiche e la conservazione di un archivio storico delle partite, è stata essenziale un'accurata interazione con i database per l'acquisizione delle informazioni rilevanti. In questa ottica, abbiamo iniziato analizzando il diagramma Entità-Relazione (ER) fornito nella documentazione del Task T4-G18.

Un diagramma Entità-Relazione (ER) è uno strumento di modellazione che rappresenta graficamente le entità di un sistema informativo e le relazioni tra di esse, utilizzato per strutturare e progettare database. Le entità, che sono oggetti o concetti dotati di dati, sono collegate da relazioni che ne indicano le interazioni e le dipendenze.

Durante l'esame approfondito del suddetto diagramma ER in congiunzione con la struttura attuale del database, abbiamo rilevato alcune incongruenze informative.

Ulteriormente, ci siamo imbattuti in una criticità riguardante la generazione della classifica, piuttosto che la gestione dello storico delle partite. Il database in questione, fornito con il Task T4, non registrava il nome del giocatore impegnato in una particolare partita. Di conseguenza, l'assenza di questi dati rendeva impraticabile il recupero del nome del giocatore per la sua inclusione nella classifica.

In sintesi, abbiamo dovuto affrontare e risolvere questioni strutturali e di integrazione dati per consentire al nostro software di operare efficacemente con le nuove funzionalità previste.

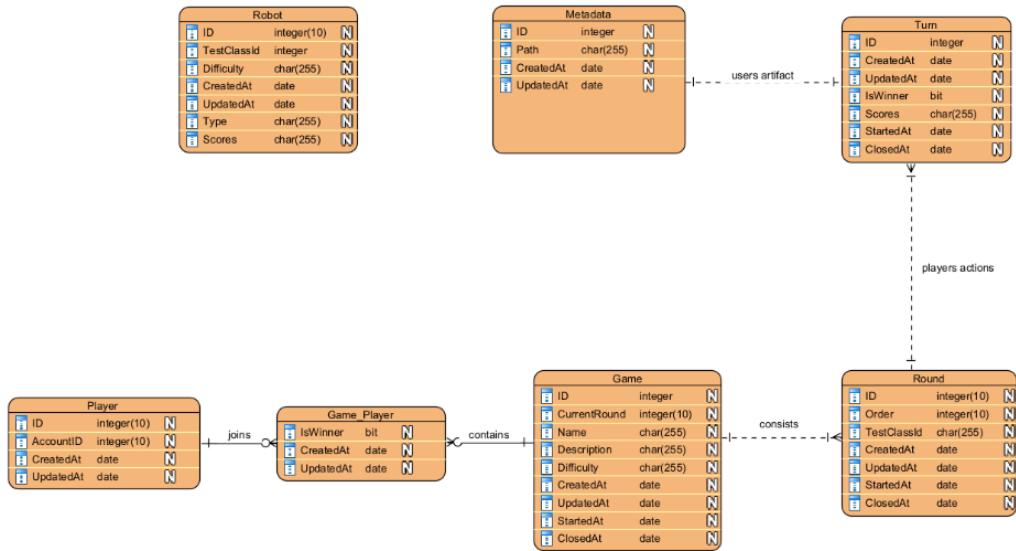


Figure 8.1: Diagramma ER iniziale

Da un'analisi approfondita, emerge chiaramente che l'attuale database non è adeguato per il corretto sviluppo dei nostri task. Tale inadeguatezza è dovuta principalmente alla sua incompletezza e imprecisione. Infatti, come dimostrato dalle immagini incluse nella sezione successiva, si evidenzia una discrepanza significativa tra le informazioni teoricamente previste nel diagramma Entità-Relazione (ER) iniziale e quelle effettivamente memorizzate nel database. Questa divergenza impedisce una fedele implementazione del modello ER nel database esistente.

### 8.1.1 Tabella turns

L'analisi del database ha rivelato l'assenza di una colonna cruciale: quella dedicata al nome o all'ID del giocatore. Se questa colonna fosse stata inclusa, avrebbe permesso di raccogliere e visualizzare facilmente i nomi dei giocatori nelle classifiche. La sua mancanza costituisce un ostacolo significativo nel rappresentare in modo efficace e completo i dati all'interno della classifica.

	id	big	created_at	updated_at	started_at	closed_at	scores_text	is_winner	player_id	round_id	round_start_time
1	1		2023-12-07 19:37:16	2023-12-07 19:37:16	2023-12-07 19:37:16	(NULL)	(NULL)	false	1	1	
2	2		2023-12-13 18:35:	2023-12-13 18:35:20	2023-12-13 18:35:15	(NULL)	(NULL)	false	1	2	
3	3		2023-12-13 21:37:	2023-12-13 21:37:31	2023-12-13 21:37:31	(NULL)	(NULL)	false	1	3	
4	4		2023-12-13 21:49:	2023-12-13 21:49:24	2023-12-13 21:49:24	(NULL)	(NULL)	false	1	4	
5	5		2023-12-13 23:13:	2023-12-13 23:13:46	2023-12-13 23:13:46	(NULL)	(NULL)	false	1	5	
6	6		2023-12-14 09:36:	2023-12-14 09:36:17	2023-12-14 09:36:17	(NULL)	(NULL)	false	1	6	
7	7		2023-12-14 11:03:	2023-12-14 11:03:19	2023-12-14 11:03:19	(NULL)	(NULL)	false	1	7	
8	8		2023-12-14 11:19:	2023-12-14 11:19:43	2023-12-14 11:19:43	(NULL)	(NULL)	false	1	8	
9	9		2023-12-14 12:52:	2023-12-14 12:52:30	2023-12-14 12:52:24	2023-12-14 12:57	100	true	1	9	
10	10		2023-12-14 13:04:	2023-12-14 13:04:53	2023-12-14 13:04:53	2023-12-14 13:07	100	true	1	10	
11	11		2023-12-14 13:10:	2023-12-14 13:10:36	2023-12-14 13:10:36	2023-12-14 13:15	100	true	1	11	
12	12		2023-12-14 13:26:	2023-12-14 13:26:40	2023-12-14 13:26:41	2023-12-14 13:38	100	true	1	12	
13	13		2023-12-14 13:41:	2023-12-14 13:41:55	2023-12-14 13:41:54	2023-12-14 13:49	9	false	1	13	
14	14		2023-12-14 13:50:	2023-12-14 13:50:55	2023-12-14 13:50:55	2023-12-14 13:52	9	false	1	14	
15	15		2023-12-14 13:52:	2023-12-14 13:52:56	2023-12-14 13:52:56	2023-12-14 13:56	9	false	1	15	
16	16		2023-12-14 13:58:	2023-12-14 13:58:40	2023-12-14 13:58:41	(NULL)	(NULL)	false	1	16	
17	17		2023-12-14 14:12:	2023-12-14 14:12:15	2023-12-14 14:12:15	2023-12-14 14:15	32	true	1	17	
18	18		2023-12-14 14:32:	2023-12-14 14:32:43	2023-12-14 14:32:43	2023-12-14 14:37	100	true	1	18	
19	19		2023-12-15 14:13:	2023-12-15 14:13:50	2023-12-15 14:13:50	(NULL)	(NULL)	false	1	19	
20	20		2024-01-07 14:46:	2024-01-07 14:46:55	2024-01-07 14:46:55	2024-01-07 14:50	100	true	1	20	
21	21		2024-01-07 16:23:	2024-01-07 16:23:58	2024-01-07 16:23:58	2024-01-07 16:25	100	true	1	21	
22	22		2024-01-10 15:51:	2024-01-10 15:51:51	2024-01-10 15:51:51	2024-01-10 15:54	9	false	1	22	

Figure 8.2: Tabella turns

### 8.1.2 Tabella players

	id	big	account_id	created_at	updated_at
1	1		1	2023-12-07 19:37:16.572279+00	2024-01-10 23:02:45.396628+00

Figure 8.3: Tabella players

### 8.1.3 Tabella games

La problematica maggiore che si ha in questo caso invece, è sicuramente legata alla colonna `test_class_id` presente all'interno della tabella. In questo caso come si evince, il `test_class_id` non viene mai memorizzato all'interno del database, pertanto non potrà in nessun modo essere recuperato successivamente.

	order	bigint	started_at	timestamp with time zone	closed_at	timestamp with time zone	updated_at	timestamp with time zone	created_at	timestamp with time zone	test_class_id	text	game_id	big
1	1		2023-12-07 19:37:16.285835+00	(NULL)		2023-12-07 19:37:16.679509+00		2023-12-07 19:37:16.679509+00			1			
2	1		2023-12-13 18:35:19.413798+00	(NULL)		2023-12-13 18:35:20.233911+00		2023-12-13 18:35:20.233911+00			2			
3	1		2023-12-13 21:37:30.324638+00	(NULL)		2023-12-13 21:37:31.600156+00		2023-12-13 21:37:31.600156+00			3			
4	1		2023-12-13 21:49:24.465551+00	(NULL)		2023-12-13 21:49:24.665846+00		2023-12-13 21:49:24.665846+00			4			
5	1		2023-12-13 23:13:45.566634+00	(NULL)		2023-12-13 23:13:46.81484+00		2023-12-13 23:13:46.81484+00			5			
6	1		2023-12-14 09:36:17.160435+00	(NULL)		2023-12-14 09:36:17.379342+00		2023-12-14 09:36:17.379342+00			6			
7	1		2023-12-14 11:03:19.703207+00	(NULL)		2023-12-14 11:03:19.799673+00		2023-12-14 11:03:19.799673+00			7			
8	1		2023-12-14 11:19:43.672543+00	(NULL)		2023-12-14 11:19:43.818467+00		2023-12-14 11:19:43.818467+00			8			
9	1		2023-12-14 12:52:29.985123+00	2023-12-14 12:57:24.125321+00		2023-12-14 12:52:30.060554+00		2023-12-14 12:52:30.060554+00			9			
10	1		2023-12-14 13:04:53.444157+00	2023-12-14 13:07:46.027715+00		2023-12-14 13:04:53.505838+00		2023-12-14 13:04:53.505838+00			10			
11	1		2023-12-14 13:05:36.264877+00	2023-12-14 13:15:06.281344+00		2023-12-14 13:10:36.292853+00		2023-12-14 13:10:36.292853+00			11			
12	1		2023-12-14 13:26:40.62752+00	2023-12-14 13:38:34.750197+00		2023-12-14 13:26:40.654604+00		2023-12-14 13:26:40.654604+00			12			
13	1		2023-12-14 13:41:54.861018+00	2023-12-14 13:49:39.347757+00		2023-12-14 13:41:55.023998+00		2023-12-14 13:41:55.023998+00			13			
14	1		2023-12-14 13:50:55.529704+00	2023-12-14 13:52:02.400876+00		2023-12-14 13:50:55.64283+00		2023-12-14 13:50:55.64283+00		VCardBean	14			
15	1		2023-12-14 13:52:55.975569+00	2023-12-14 13:56:07.721665+00		2023-12-14 13:52:56.082502+00		2023-12-14 13:52:56.082502+00		VCardBean	15			
16	1		2023-12-14 13:58:40.480063+00	(NULL)		2023-12-14 13:58:40.536207+00		2023-12-14 13:58:40.536207+00		VCardBean	16			
17	1		2023-12-14 14:12:15.385019+00	2023-12-14 14:15:38.622679+00		2023-12-14 14:12:15.430161+00		2023-12-14 14:12:15.430161+00			17			
18	1		2023-12-14 14:32:43.505215+00	2023-12-14 14:37:49.032965+00		2023-12-14 14:32:43.555667+00		2023-12-14 14:32:43.555667+00			18			
19	1		2023-12-15 14:13:50.149617+00	(NULL)		2023-12-15 14:13:50.731074+00		2023-12-15 14:13:50.731074+00			19			
20	1		2024-01-07 14:46:54.487244+00	2024-01-07 14:50:23.582943+00		2024-01-07 14:46:55.3664187+00		2024-01-07 14:46:55.3664187+00			20			
21	1		2024-01-07 16:23:57.97229+00	2024-01-07 16:25:04.634335+00		2024-01-07 16:23:58.049119+00		2024-01-07 16:23:58.049119+00			21			
22	1		2024-01-10 15:51:50.624838+00	2024-01-10 15:54:51.957852+00		2024-01-10 15:51:51.561552+00		2024-01-10 15:51:51.561552+00			22			

Figure 8.4: Tabella games

40	1	40	nome	(NULL)		2024-01-15 09:2024-01-15 09:54:36.404819+00		2024-01-15 09:54:36.399854+00		(NULL)			
41	1	41	nome	(NULL)		2024-01-15 09:2024-01-15 09:54:55.98973+00		2024-01-15 09:54:55.982622+00		(NULL)			
42	1	42	nome	(NULL)		2024-01-15 09:2024-01-15 09:55:53.440417+00		2024-01-15 09:55:53.435945+00		(NULL)			
43	1	43	nome	(NULL)		2024-01-15 10:2024-01-15 10:16:39.573928+00		2024-01-15 10:16:39.567607+00		(NULL)			
44	1	44	nome	(NULL)		2024-01-15 10:2024-01-15 10:16:46.599058+00		2024-01-15 10:16:46.591751+00		(NULL)			
45	1	45	nome	(NULL)		2024-01-15 10:2024-01-15 10:16:53.330206+00		2024-01-15 10:16:53.324325+00		(NULL)			
46	1	46	nome	(NULL)		2024-01-15 10:2024-01-15 10:19:17.985615+00		2024-01-15 10:19:17.963619+00		(NULL)			
47	1	47	nome	(NULL)		2024-01-15 10:2024-01-15 10:19:32.159619+00		2024-01-15 10:19:32.145017+00		(NULL)			
48	1	48	nome	(NULL)		2024-01-15 10:2024-01-15 10:26:44.331208+00		2024-01-15 10:26:44.322249+00		(NULL)			
49	1	49	nome	(NULL)		2024-01-15 10:2024-01-15 10:37:13.375412+00		2024-01-15 10:37:13.368964+00		(NULL)			
50	1	50	nome	(NULL)		2024-01-15 12:2024-01-15 12:10:09.722238+00		2024-01-15 12:10:09.69288+00		(NULL)			
51	1	51	nome	(NULL)		2024-01-15 23:2024-01-15 23:25:11.277991+00		2024-01-15 23:25:10.619823+00		(NULL)			
52	1	52	nome	(NULL)		2024-01-15 23:2024-01-15 23:25:27.334782+00		2024-01-15 23:25:27.31347+00		(NULL)			
53	1	53	nome	(NULL)		2024-01-15 23:2024-01-15 23:27:34.764818+00		2024-01-15 23:27:34.758506+00		(NULL)			
54	1	86	nome	(NULL)		2024-01-17 10:2024-01-17 10:04:42.342388+00		2024-01-17 10:04:42.060089+00		(NULL)			
55	1	87	nome	(NULL)		2024-01-17 10:2024-01-17 10:05:26.842096+00		2024-01-17 10:05:26.832306+00		(NULL)			
56	1	88	nome	(NULL)		2024-01-17 10:2024-01-17 10:05:34.254814+00		2024-01-17 10:05:34.238053+00		(NULL)			
57	1	89	nome	(NULL)		2024-01-17 10:2024-01-17 10:24:03.799481+00		2024-01-17 10:24:03.653125+00		(NULL)			
58	1	90	nome	(NULL)		2024-01-17 10:2024-01-17 10:35:40.46395+00		2024-01-17 10:35:40.196619+00		(NULL)			

Figure 8.5: Parte finale tabella games

## 8.2 Modifiche effettuate

### 8.2.1 Diagramma ER

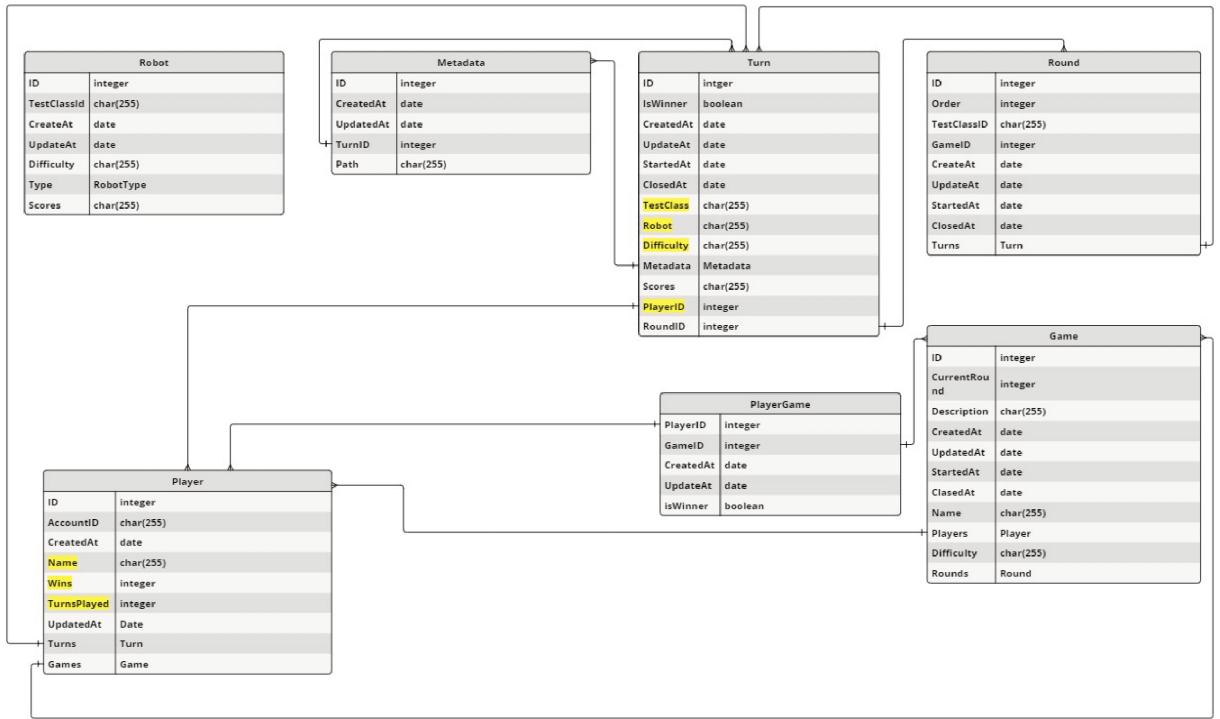


Figure 8.6: Diagramma ER aggiornato

Nel diagramma ER illustrato, abbiamo apportato modifiche sia in base alle nostre specifiche che alle modifiche proposte dal T11. Tuttavia, è importante notare che, anche se il T11 aveva spiegato le modifiche, non aveva creato un nuovo diagramma ER per rappresentarle.

### 8.2.2 Player

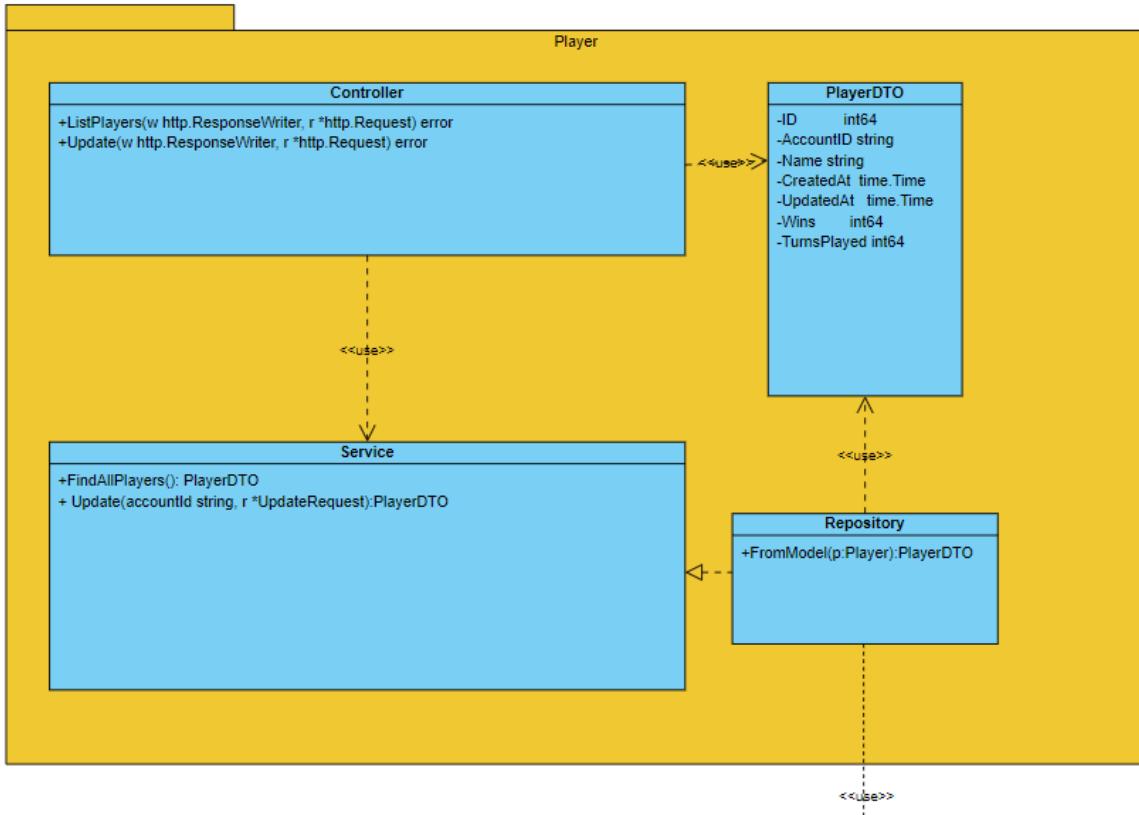


Figure 8.7: System Domain Model: Focus Player

Come precedentemente definito, ciò che era stato realizzato non risultava sufficientemente completo per consentire la visualizzazione adeguata della classifica e dello storico, considerando tutti i campi richiesti. Di conseguenza, si è optato per l'aggiunta di un *folder player*, poiché inizialmente erano state definite solamente le API per le tabelle *game*, *robot*, *round* e *turn*, ma non per *player*. Abbiamo quindi deciso di strutturare il *folder player*, che include i seguenti campi:

- ID
- AccountID
- Name
- CreateAt
- UpdateAt
- Wins

- Turnedplayed

All'interno di questo folder, sono state successivamente definite delle funzioni che permettono la conversione dei tipi di dati, ad esempio da intero a stringa. Queste funzioni sono state create per essere poi impiegate correttamente nelle chiamate API. Per quanto riguarda la tabella *player*, sono stati implementati due diversi servizi all'interno del Controller:

1. FindAllPlayers
2. Update

La funzione *ListPlayers* ci ha permesso di ottenere un elenco completo di tutti i giocatori presenti nella tabella *player*. Al contrario, la funzione *Update* è utilizzata per aggiornare il nome di un giocatore in base al suo *accountId*. Specificamente, quando si richiama questo endpoint con un *accountId*, se questo corrisponde a uno presente nella tabella *player*, il campo nome viene aggiornato.

All'interno del *service*, sempre nel folder *player*, abbiamo implementato due funzioni: una per stampare l'elenco completo dei giocatori e l'altra per aggiornare il nome del giocatore in base all'*accountId* dopo ogni partita. Abbiamo scelto di utilizzare l'*accountId* piuttosto che l'ID perché, nel database in uso, l'ID non segue una sequenza incrementale ma viene generato in modo casuale. L'*accountId*, invece, corrisponde all'indice utilizzato nella tabella MySQL del database fornito da T2.

### 8.2.3 Player: Codice

Come precedentemente menzionato, di seguito le modifiche apportate alla tabella "Player" per adattarla alle nostre esigenze: è stato aggiunto il campo "Wins", il quale conta il numero di turni in cui il campo "isWinner" della tabella "Turn" è impostato su "true" per ogni giocatore, è stato aggiunto il campo "TurnsPlayed", per tenere traccia del numero totale di turni giocati e completati con successo da ciascun giocatore. Inoltre, è stato aggiunto il campo "Name" per memorizzare i nomi degli utenti registrati.

```

39 < type Player struct {
40     ID      int64    `gorm:"primaryKey;autoIncrement"`
41     AccountID string   `gorm:"unique"`
42     Name      string
43     CreatedAt time.Time `gorm:"autoCreateTime"`
44     UpdatedAt time.Time `gorm:"autoUpdateTime"`
45     Turns     []Turn   `gorm:"foreignKey:PlayerID;constraint:OnDelete:SET NULL;"` 
46     Games     []Game   `gorm:"many2many:player_games;foreignKey:AccountID;joinForeignKey:PlayerID;"` 
47     Wins      int64    `gorm:"default:0"` //aggiunto
48     TurnsPlayed int64   `gorm:"default:0"` //aggiunto
49 }

```

Figure 8.8: Tabella Player

Come indicato nel paragrafo precedente, per rispondere efficacemente alle esigenze del task assegnato, è stato necessario implementare delle nuove API specifiche per la tabella Player. Queste API non erano presenti nel progetto originale, ma la loro introduzione si è resa indispensabile per la realizzazione di una funzionalità chiave: la costruzione della classifica generale dei giocatori. Grazie a queste nuove API, possiamo ora interagire in modo più diretto e funzionale con i dati memorizzati nella tabella Player del database, facilitando la creazione e la gestione della classifica dei giocatori registrati.

#### 8.2.4 Recupero Lista Giocatori

- **Endpoint:** /players
- **Metodo HTTP:** GET
- **URL:** `http://t4-g18-app-1:3000/players`
- **Descrizione:** Questo endpoint recupera la lista dei giocatori con i dettagli delle loro informazioni di gioco. È utilizzato per popolare la tabella classifica nel front-end.
- **Parametri di richiesta:** Nessuno.
- **Risposta di esempio (successo):**

---

```

1 {
2     "codice HTTP": 200,
3     "corpo": [
4         {
5             "accountId": "12345",

```

```
6         "name": "Giocatore1",
7         "wins": 10,
8         "turnsPlayed": 20
9     },
10    {
11        "accountId": "67890",
12        "name": "Giocatore2",
13        "wins": 5,
14        "turnsPlayed": 15
15    }
16 ]
17 }
```

---

- **Errori Comuni:**

- **Codice HTTP:** 500 Internal Server Error
  - **Descrizione:** Errore interno del server.

#### 8.2.5 Aggiornamento Dettagli Giocatore

- **Endpoint:** /players/{accountId}
- **Metodo HTTP:** PUT
- **URL:** http://t4-g18-app-1:3000/players/{accountId}
- **Descrizione:** Questo endpoint aggiorna i dettagli di un giocatore specifico identificato da accountId. Viene utilizzato per aggiornare il nome del giocatore durante il salvataggio del gioco.

- **Parametri di richiesta:**

- **Path Variable:** accountId (ID del giocatore)
  - **Corpo della richiesta:**

```
1  {  
2      "name": "NuovoNomeGiocatore"  
3  }
```

---

- **Risposta di esempio (successo): Codice HTTP:** 200 OK

- **Errori Comuni:**

- **Codice HTTP:** 400 Bad Request
- **Descrizione:** Richiesta non valida (ad esempio, formato JSON errato).
- **Codice HTTP:** 404 Not Found
- **Descrizione:** Giocatore non trovato.
- **Codice HTTP:** 500 Internal Server Error
- **Descrizione:** Errore interno del server.

### 8.2.6 Turn

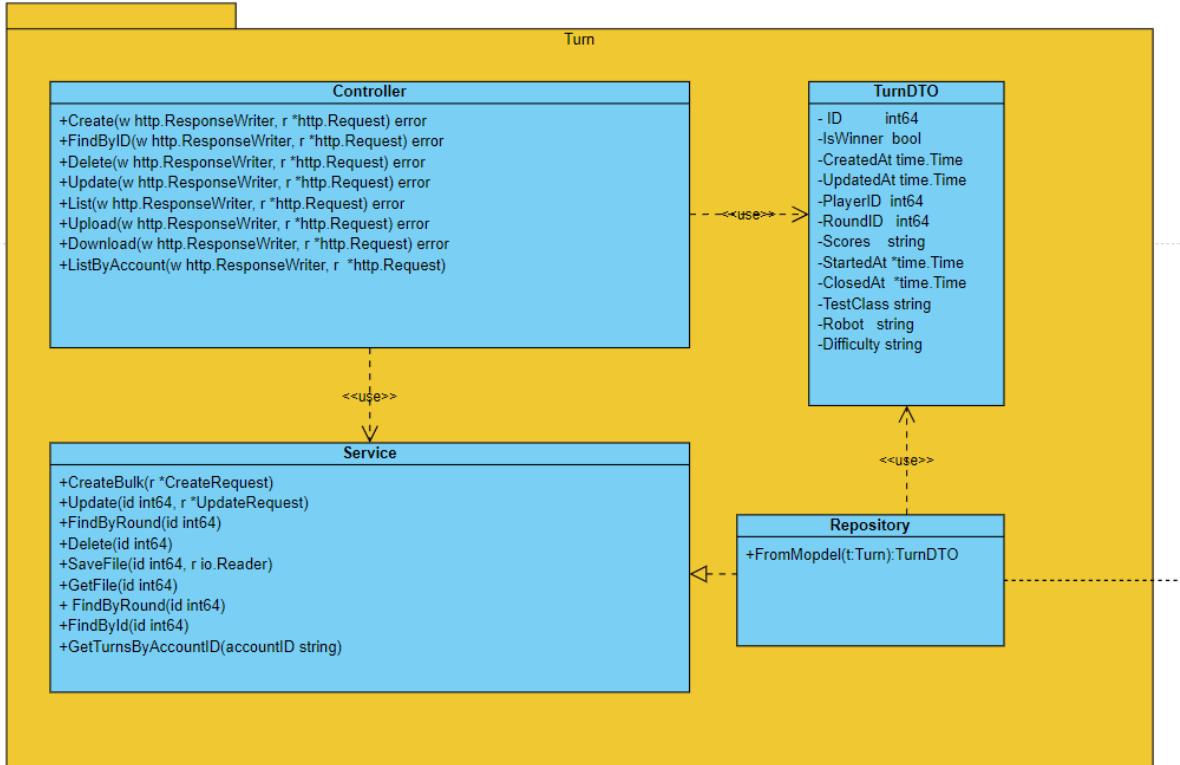


Figure 8.9: System Domain Model: Focus Turn

La tabella *turn* ha il compito di salvare tutti i turni giocati e si collega alla tabella *player* tramite l'ID del giocatore. Tuttavia, questo ID non corrisponde all'*AccountId*. Di conseguenza, considerando che l'utente loggato viene identificato dall'*AccountId*, abbiamo stabilito un collegamento con la tabella *player*, utilizzando l'*AccountId* per trovare il giocatore corrispondente nella tabella *turn*.

Nella tabella *turn*, sono state introdotte modifiche sostanziali per arricchire le funzionalità del sistema e migliorare la rappresentazione dello storico delle partite. Le modifiche apportate comprendono:

- Nel Controller:** È stata aggiunta la funzione `ListByAccount`, che permette di elencare i turni di gioco associati a un account specifico.
- Nel Service:** La logica di business è stata estesa con `GetTurnsByAccountId`, facilitando il recupero dei turni di gioco in base all'identificativo dell'account.

- Nel DTO di Turn:** Sono stati aggiunti campi quali `TestClass`, `Robot` e `Difficulty`, che consentono di dettagliare ulteriormente ciascun turno di gioco, inclusi la difficoltà, i tempi di inizio e i punteggi ottenuti.

Queste modifiche mirano a fornire una visione dettagliata dello storico delle partite (`difficulty`), il momento di inizio (`startedAt`) e i punteggi (`scores`). Inoltre, stabiliscono un collegamento diretto tra gli account degli utenti e i turni di gioco effettuati (`ListByAccount`), garantendo così un sistema di tracciamento coerente e funzionale.

### 8.2.7 Turn: Codice

Nella tabella "Turn", sono stati introdotti tre nuovi campi: "TestClass", "Robot" e "Difficulty". Il campo "TestClass" è utilizzato per registrare la classe testata in ogni turno, "Robot" serve a memorizzare il robot sfidato durante il turno, e "Difficulty" è destinato a registrare il livello di difficoltà del robot testato.

```

71 v type Turn struct {
72     ID      int64      `gorm:"primaryKey;autoIncrement"`
73     CreatedAt time.Time `gorm:"autoCreateTime"`
74     UpdatedAt time.Time `gorm:"autoUpdateTime"`
75     StartedAt *time.Time `gorm:"default:null"`
76     ClosedAt *time.Time `gorm:"default:null"`
77     TestClass string    `gorm:"default:null"`
78     Robot     string    `gorm:"default:null"`
79     Difficulty string   `gorm:"default:null"`
80     Metadata  Metadata  `gorm:"foreignKey:TurnID;constraint:OnDelete:SET NULL;"`  

81     Scores    string    `gorm:"default:null"`
82     IsWinner  bool      `gorm:"default:false"`
83     PlayerID int64     `gorm:"index:idx_playerturn,unique;not null"`
84     RoundID   int64     `gorm:"index:idx_playerturn,unique;not null"`
85 }
```

Figure 8.10: Tabella Turn

Nell'ambito delle API relative ai Turn, è stata introdotta una modifica significativa alla funzione `Update` del servizio. Questa modifica consiste nell'aggiunta di una chiamata alla funzione `UpdatePlayerWins`, la quale necessita dell'`PlayerID` come parametro di ingresso. La modifica mira a estendere la funzionalità dell'API in modo che, ogni volta che un turno viene aggiornato tramite la funzione `Update` – una funzione invocata al termine di ogni turno –, l'operazione non si limiti più a semplicemente aggiornare i record specificati nell'`UpdateRequest`. Ora,

essa comprende anche l'aggiornamento dei dati relativi al numero di vittorie e al conteggio dei turni giocati nella tabella Player.

```

102 func (tr *Repository) Update(id int64, r *UpdateRequest) (Turn, error) {
103     var turn model.Turn
104     err := tr.db.First(&turn, id).Error
105     if err != nil {
106         return Turn{}, api.MakeServiceError(err)
107     }
108
109     playerID := turn.PlayerID
110     log.Printf("Updating wins for playerID: %d", playerID)
111
112     err = tr.db.Model(&turn).Updates(r).Error
113     if err != nil {
114         return Turn{}, api.MakeServiceError(err)
115     }
116
117     if err := model.UpdatePlayerWins(tr.db, playerID); err != nil {
118         log.Printf("Error updating player wins: %v", err)
119         return Turn{}, api.MakeServiceError(err)
120     }
121     return fromModel(&turn), nil
122 }
123 }
```

Figure 8.11: Funzione Update()

Nel paragrafo dedicato al Model, viene fornita una spiegazione dettagliata della funzione `UpdatePlayerWins`. Tale funzione è progettata per aggiornare, nel database, i campi che tengono traccia del numero di vittorie e del numero di turni giocati associati a un determinato giocatore.

Con questa modifica, l'API di Turn non solo aggiorna le informazioni specifiche di un turno, ma contribuisce anche attivamente alla gestione delle statistiche complessive dei giocatori, assicurando che il conteggio delle vittorie e dei turni giocati rimanga accurato e aggiornato dopo ogni turno. Questo meccanismo garantisce una sincronizzazione efficace tra le prestazioni individuali nei turni e le statistiche cumulative dei giocatori, riflettendo così l'impatto di ogni turno sulle prestazioni complessive dei giocatori nel gioco o nella competizione.

Inoltre, per soddisfare le richieste specificate nel task, è stata ritenuta necessaria l'aggiunta di un endpoint alle API. Questo endpoint è progettato per recuperare l'elenco dei turni associati a un determinato utente, facilitando così la creazione di uno storico delle partite. Tale endpoint richiede in input l'AccountID dell'utente loggato. Utilizzando la tabella "Player", l'AccountID viene collegato all'ID dell'utente. Una volta ottenuto l'ID dell'utente, l'endpoint è in grado di

mostrare i dettagli dei turni generati dall'utente stesso.

### 8.2.8 Recupero Storico Turni Giocatore

- **Endpoint:** /turns/account/{accountID}
- **Metodo HTTP:** GET
- **URL:** http://t4-g18-app-1:3000/turns/account/{accountID}
- **Descrizione:** Questo endpoint recupera l'elenco dei turni giocati da un utente specificato dall'accountID. Utilizzato per mostrare lo storico delle partite dell'utente.
- **Parametri di richiesta:**
  - **Path Variable:** accountID (ID dell'account utente)
- **Risposta di esempio (successo):**

---

```
1 [  
2   {  
3     "id": 1,  
4     "isWinner": true,  
5     "createdAt": "2022-07-21T12:00:00Z",  
6     "updatedAt": "2022-07-21T12:30:00Z",  
7     "playerId": 123,  
8     "roundId": 456,  
9     "scores": "100",  
10    "startedAt": "2022-07-21T12:00:00Z",  
11    "closedAt": "2022-07-21T12:30:00Z",  
12    "testClass": "ClassA",  
13    "robot": "RobotX",  
14    "difficulty": "1"  
15  },  
16  ...  
17 ]
```

---

- **Errori Comuni:**

- **Codice HTTP:** 400 Bad Request
- **Descrizione:** Richiesta non valida (ad esempio, formato ID errato).
- **Codice HTTP:** 404 Not Found
- **Descrizione:** Account non trovato.
- **Codice HTTP:** 500 Internal Server Error
- **Descrizione:** Errore interno del server.

### 8.2.9 Model

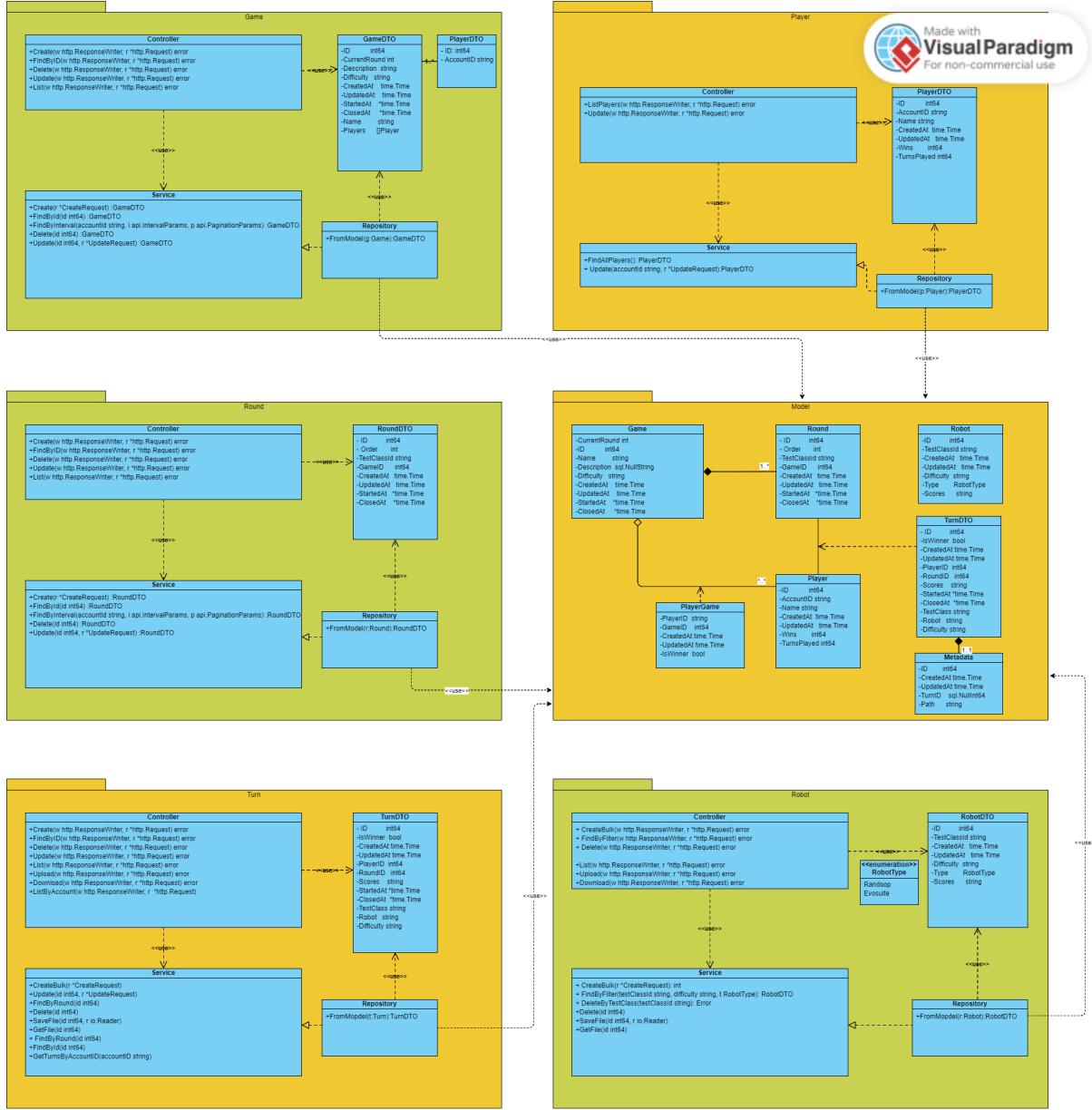


Figure 8.12: System Domain Model T4

In aggiunta, all'interno del file *model.go* nel model, la tabella *player* è stata modificata aggiungendo due campi:

- *Wins*, che conta il numero di vittorie.
- *TurnsPlayed*, ovvero il numero di turni giocati.

Il campo *Wins* viene aggiornato tramite la funzione `UpdatePlayerWins`, che conta quante volte un *player\_id* specifico ha il campo *is\_winner* impostato su `true`.

Infine, in *main.go*, sono stati aggiunti tutti gli endpoint necessari per il corretto funzionamento del gioco e per l'adeguato popolamento del database.

### 8.2.10 Model: Codice

Nel modello sono state apportate modifiche alle tabelle precedentemente descritte, in particolare alle tabelle "Player" e "Turn". Per evitare ridondanze, non verranno riportati i dettagli dei campi aggiunti né la struttura aggiornata delle tabelle. È importante sottolineare che sono state introdotte due nuove funzioni, le quali sono utili per popolare i campi aggiunti nella tabella "Player".

La funzione `UpdatePlayerWins` serve ad aggiornare i campi `wins` e `TurnsPlayed` nella tabella `Player` del database. La funzione calcola il numero di vittorie e il numero totale di turni giocati da un giocatore specifico, identificato tramite `PlayerID`, e aggiorna questi dati nel database. In caso di errori nelle query o negli aggiornamenti, la funzione restituisce l'errore corrispondente. Se tutto va a buon fine, restituisce `nil`, indicando il successo dell'operazione.

Importante sottolineare è che questa funzione viene richiamata all'interno della funzione `Update()` nel service di `Turn`. Tale integrazione assicura che, ogni volta che un turno viene aggiornato attraverso `Update()`, oltre agli aggiornamenti specificati nell'`UpdateRequest`, vengano anche ricalcolati e aggiornati i campi `wins` e `TurnsPlayed` per il giocatore corrispondente, mantenendo così aggiornate le statistiche complessive dei giocatori nella tabella `Player`.

```

91 // Funzione per aggiornare il campo Wins e il campo turnsPlayed per un giocatore specifico
92 func UpdatePlayerWins(db *gorm.DB, playerID string) error {
93     var winsCount, turnsPlayed int64
94
95     // Calcola il numero di vittorie per il giocatore specifico
96     result := db.Model(&Turn{}).Where("player_id = ? AND is_winner = ?", playerID, true).Count(&winsCount)
97     if result.Error != nil {
98         return result.Error
99     }
100
101    //Calcola il numero totale di partite giocate e terminate correttamente per il giocatore specificato
102    result = db.Model(&Turn{}).Where("player_id = ? AND closed_at IS NOT NULL", playerID).Count(&turnsPlayed)
103    if result.Error != nil {
104        return result.Error
105    }
106
107    // Aggiorna il campo Wins e TurnsPlayed nella tabella Player
108    result = db.Model(&Player{}).Where("id = ?", playerID).Updates(map[string]interface{}{
109        "wins":      winsCount,
110        "turns_played": turnsPlayed,
111    })
112    if result.Error != nil {
113        return result.Error
114    }
115    return nil
116 }
```

Figure 8.13: Funzione UpdatePlayerWins

L'hook `AfterSave` definito per la struttura `Turn` viene attivato automaticamente dopo il salvataggio di un record nella tabella `Turns` ed aggiorna i campi `TurnsPlayed` e `wins`, nella tabella `Player`, mantenendo così aggiornate le statistiche di gioco per ogni giocatore.

```

150 //Hook che si attiva ogni volta che si salva un record nella tabella Turns
151 func(pg *Turn) AfterSave(tx *gorm.DB) (err error){
152     playerID := pg.PlayerID
153     var turnsPlayed int64
154     result := tx.Model(&Turn{}).Where("player_id = ? AND closed_at IS NOT NULL", playerID).Count(&turnsPlayed)
155     if result.Error != nil{
156         return result.Error
157     }
158
159     //Aggiorna il campo TurnsPlayed nella tabella Player
160     result = tx.Model(&Player{}).Where("id = ?", playerID).Update("turns_played", turnsPlayed)
161     if result.Error != nil{
162         return result.Error
163     }
164
165     // Ottieni il valore IsWinner dalla tabella Turn per il giocatore specifico
166     var isWinner bool
167     result = tx.Model(&Turn{}).Where("player_id = ? AND is_winner = ?", playerID, true).Select("is_winner").Scan(&isWinner)
168     if result.Error != nil {
169         return result.Error
170     }
171
172     // Se isWinner è true, aggiorna il campo Wins nella tabella Player
173     if isWinner {
174         var winsCount int64
175
176         // Calcola il numero di vittorie per il giocatore specifico dalla tabella Turn
177         result := tx.Model(&Turn{}).Where("player_id = ? AND is_winner = ?", playerID, true).Count(&winsCount)
178         if result.Error != nil {
179             return result.Error
180         }
181
182         // Aggiorna il campo Wins nella tabella Player con il numero di vittorie ottenuto
183         result = tx.Model(&Player{}).Where("id = ?", playerID).Update("wins", winsCount)
184         if result.Error != nil {
185             return result.Error
186         }
187     }
188
189     return nil
}

```

Figure 8.14: Hook AfterSave

# Chapter 9

## Modifiche al Task5

### 9.1 Diagramma di contesto

Per rispondere ai requisiti stabiliti, abbiamo apportato delle modifiche anche al task T5. La prima azione intrapresa è stata la definizione di un nuovo *diagramma di contesto*.

Un *diagramma di contesto* è uno strumento utilizzato nella fase di analisi di un progetto software per fornire una visione generale e comprensiva dell'applicativo. Esso rappresenta graficamente gli attori coinvolti e tutti gli elementi chiave che interagiscono con il sistema, facilitando la comprensione del flusso di lavoro e delle interazioni tra le varie componenti.

Il nostro punto di partenza per l'analisi è stata proprio la creazione di tale diagramma, che mira a delineare una panoramica generale dell'esecuzione dell'applicativo. Questo include la rappresentazione degli attori coinvolti e di tutti gli elementi essenziali per il funzionamento del servizio implementato.

Nello specifico, abbiamo identificato due attori principali: *Utente* e *Studente*, con quest'ultimo che rappresenta una specializzazione del primo. Entrambi gli attori interagiscono con l'applicativo tramite un browser, attraverso il quale accedono all'interfaccia utente fornita dal sistema software per sfruttare le varie funzionalità.

L'attore *Utente* ha accesso solamente alla pagina di login, in quanto solo gli utenti registrati e autenticati possono partecipare al gioco. Una volta autenticato, l'*Utente* diventa uno *Studente*

e può utilizzare l'interfaccia utente per selezionare tra diverse opzioni: visualizzare la classifica generale dei giocatori, visualizza storico, iniziare una nuova partita in modalità 1vs1 o 1vsTutti. Dopo aver effettuato la scelta, lo studente verrà indirizzato alla visualizzazione della classifica (o dello storico) o all'interfaccia che permette di scegliere la modalità di gioco desiderata.

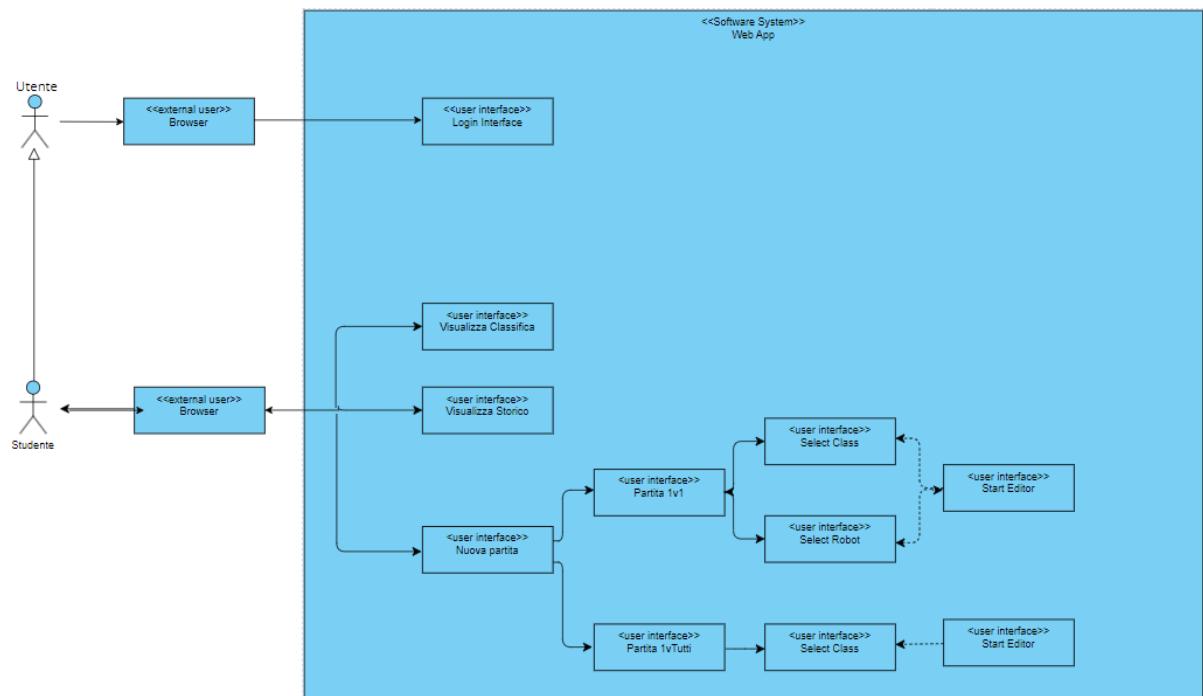


Figure 9.1: Diagramma Di Contesto

## 9.2 Diagramma dei package

Il **diagramma dei package** rappresenta è uno strumento fondamentale per comprendere la struttura organizzativa di un sistema software basato sul modello MVC. Esso offre una visualizzazione delle relazioni tra i componenti **Model**, **View** e **Controller**, fornendo una panoramica dell'architettura dell'applicazione e delle dipendenze tra i diversi componenti alla base di quest'ultima.

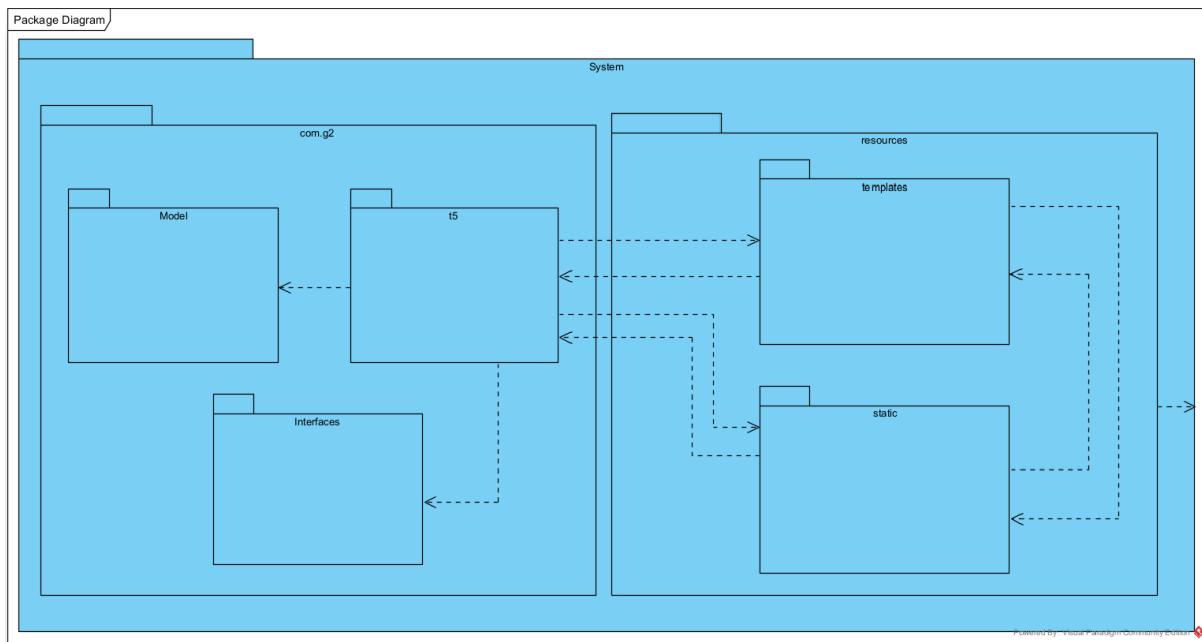


Figure 9.2: Diagramma dei package

### 9.3 Diagramma delle classi

Il **diagramma delle classi** rappresenta un elemento cruciale nel processo di sviluppo software, fornendo una mappa chiara e strutturata delle classi e delle loro interrelazioni all'interno del sistema. Questo strumento gioca un ruolo chiave nel delineare la struttura interna e le dinamiche di interazione del software, migliorando la comunicazione tra gli sviluppatori e facilitando la comprensione da parte degli stakeholder coinvolti. Il diagramma delle classi eccelle nell'offrire una visione approfondita e dettagliata delle classi, evidenziando le loro proprietà e metodi. La capacità di descrivere meticolosamente le classi si traduce in benefici tangibili per il progetto.

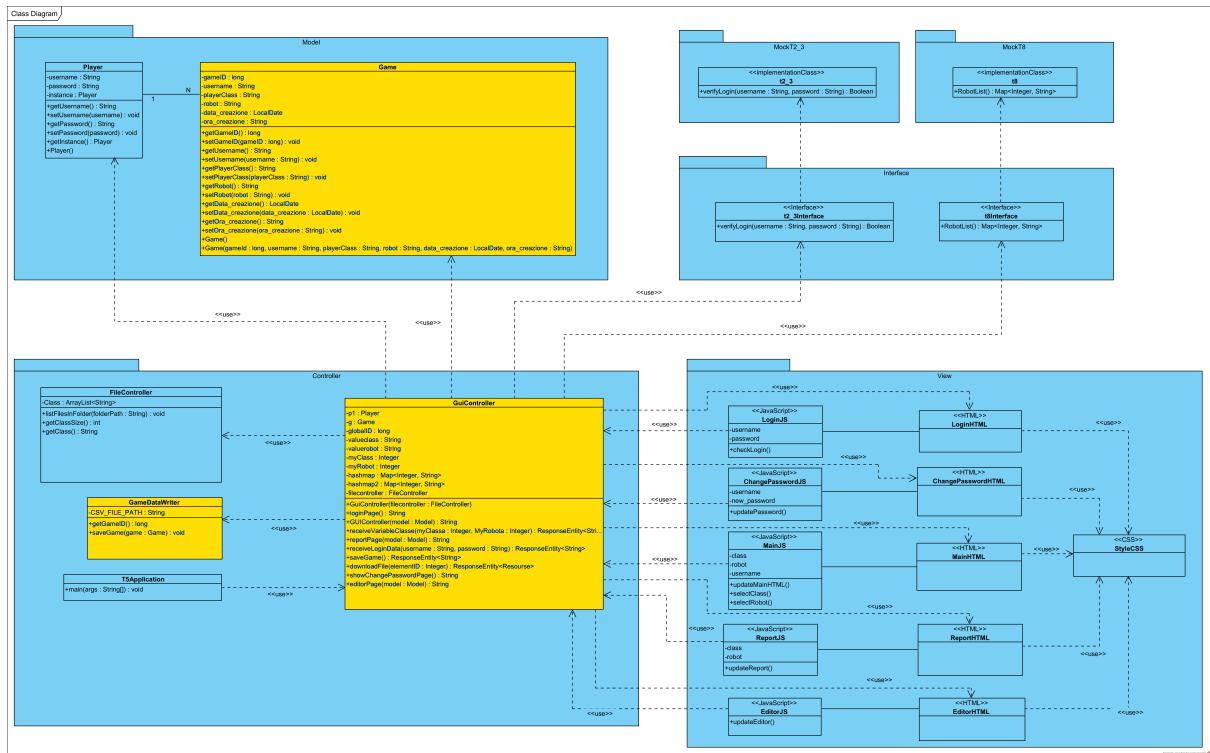


Figure 9.3: Diagramma delle Classi T5

### 9.3.1 GuiController

GuiController
<pre> - private RestTemplate restTemplate; - private String nameAuth - private String IdAuth  + GuiController(RestTemplate restTemplate) + getLevels(String className): List&lt;String&gt; + mainPage(Model model, @CookieValue(required = false) String jwt) : String + classificaPage():String + AllRobotsPage(Model model, @CookieValue(required = false) String jwt):String + String GameModePage():String + String storicoPage():String + getClasses(): ResponseEntity&lt;List&lt;ClassUT&gt;&gt; + GUIController(Model model, @CookieValue(required = false) String jwt): String + reportPage(Model model, @CookieValue(required = false) String jwt): String + report1Page(Model model, @CookieValue(required = false) String jwt):String + saveGame(@RequestParam int playerId, .... HttpServletRequest request): ResponseEntity&lt;String&gt; + editorPage(Model model, @CookieValue(required = false) String jwt):String </pre>

Figure 9.4: GuiController modifiche effettuate - Diagramma delle Classi

Nel GuiController sono state introdotte nuove variabili, IdAuth e NameAuth, per mantenere traccia dell'identificativo e del nome dell'utente autenticato.

All'interno del metodo AllRobotsPage(), abbiamo implementato la logica per recuperare dal database l'elenco delle classi disponibili e dei robot associati. Il codice è stato strutturato per assicurare facilità di modifica, favorendo future implementazioni, specialmente per la modalità di gioco che prevede la sfida contro tutti i robot.

La funzione StoricoPage è stata sviluppata con l'obiettivo di presentare agli utenti lo storico delle partite giocate, fornendo un resoconto dettagliato delle attività svolte.

Abbiamo aggiornato il GuiController per includere una nuova pagina, la quale consente agli utenti di selezionare la classe e il robot desiderati, una volta scelta la modalità di gioco 1v1. Questa modifica mira a migliorare l'interfaccia utente, rendendo la selezione più intuitiva e accessibile.

In fine abbiamo modificato la funzione SaveGame per permettere di salvare il nome del giocatore durante il salvataggio dei dati.

### 9.3.2 Codice: GUIController

Le modifiche apportate al codice del GUIcontroller sono strettamente legate alla logica del Model-View-Controller (MVC) implementata nelle varie pagine aggiunte all'applicazione web. Questi aggiustamenti assicurano che il flusso dei dati tra il modello, le viste e il controller sia ottimizzato per le nuove funzionalità introdotte, garantendo così coerenza e integrità nell'architettura dell'applicazione.

- **Main Page:** Questo metodo gestisce le richieste alla pagina principale ("main"). Verifica l'autenticazione dell'utente attraverso un token JWT contenuto in un cookie. Se l'utente non è autenticato, viene reindirizzato alla pagina di login. Se è autenticato, viene visualizzata la pagina principale.

```

93     @GetMapping("/main")
94     public String mainPage(Model model, @CookieValue(required = false) String jwt) {
95         MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
96         formData.add("key", "jwt");
97
98         Boolean isAuthenticated = restTemplate.postForObject(url:"http://t23-g1-app-1:8080/validateToken", formData, responseType:Boolean.class);
99         if(isAuthenticated == null || !isAuthenticated) return "redirect:/login";
100
101     }

```

Figure 9.5: GUIController: MainPage

La pagina 'Main' costituisce la pagina principale dell'applicazione, offrendo agli utenti un'interfaccia intuitiva per accedere alle principali funzionalità. In questa pagina, l'utente può optare per iniziare una nuova partita, consultare la classifica, o esplorare lo storico delle partite precedenti. Dettagli aggiuntivi su queste funzionalità, comprese le modifiche specifiche apportate all'interfaccia utente (front-end), sono trattati in modo approfondito in una sezione dedicata di questo capitolo.

- **Classifica:** il codice riportato in basso è responsabile di gestire la richiesta per la pagina della classifica, verificare l'autenticazione dell'utente, come nella pagina precedente, recuperare e ordinare i dati della classifica, mediante una richiesta GET all'indirizzo "http://t4-g18-app-1:3000/players" per ottenere la lista dei giocatori. Infine, Dopo l'ordinamento di tale lista in base al numero di vittorie, viene aggiunta al modello per passare i dati della classifica alla vista associata.

```

103    @GetMapping("/classifica")
104    public String classificaPage(Model model, @CookieValue(required = false) String jwt){
105        MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
106        formData.add(key:"jwt", jwt);
107
108        Boolean isAuthenticated = restTemplate.postForObject(url:"http://t23-g1-app-1:8080/validateToken", formData, responseType:Boolean.class);
109        if(isAuthenticated == null || !isAuthenticated) return "redirect:/login";
110
111        List<Map<String, Object>> classifica = restTemplate.getForObject(url:"http://t4-g18-app-1:3000/players", responseType>List.class);
112        classifica.sort(Comparator.comparingInt((Map<String, Object> giocatore) -> (Integer) giocatore.get("wins")).reversed());
113        model.addAttribute(attributeName:"classifica", classifica);
114
115        return "classifica";
116    }

```

Figure 9.6: GUIController: Classifica

- Storico:** Il codice illustrato gestisce la richiesta per la pagina dello Storico. Inizialmente, verifica l'autenticazione dell'utente mediante una richiesta POST al servizio 'http://t23-g1-app-1:8080/validateToken' per validare il token JWT. Se l'autenticazione è confermata, procede con una seconda richiesta POST a 'http://t23-g1-app-1:8080/IdToken' per ottenere l'ID dell'utente associato al JWT. Una volta ottenuto l'ID, questo viene aggiunto al modello per essere passato alla vista associata. Se l'utente non è autenticato, viene reindirizzato alla pagina di login.

```

166    @GetMapping("/storico")
167    public String storicoPage(Model model, @CookieValue(required = false) String jwt){
168        MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
169        formData.add(key:"jwt", jwt);
170
171        Boolean isAuthenticated = restTemplate.postForObject(url:"http://t23-g1-app-1:8080/validateToken", formData, responseType:Boolean.class);
172
173
174        if(isAuthenticated == null || !isAuthenticated) return "redirect:/login";
175
176        Integer IdTemp = restTemplate.postForObject(url:"http://t23-g1-app-1:8080/IdToken", formData, responseType=Integer.class);
177        if(IdTemp != null){
178            IdAuth = IdTemp.toString();
179            System.out.println("ID utente: " + IdAuth);
180        }else{
181            System.out.println("ID utente non ricevuto o errore nella richiesta");
182        }
183
184        model.addAttribute(attributeName:"IdAuth", IdAuth);
185
186
187        return "storico";
188    }

```

Figure 9.7: GUIController: Storico

- AllRobots:** Il codice presentato gestisce la richiesta per la pagina *AllRobot*. Inizialmente, verifica l'autenticazione dell'utente mediante una richiesta POST al servizio "http://t23-g1-app-1:8080/validateToken" per validare il token JWT. Se l'autenticazione è confermata, procede con le seguenti operazioni:

- Recupero delle Classi Under Test:** Utilizza il metodo `getClasses()` per ottenere una lista delle *Classi Under Test*.

**2. Inizializzazione delle Mappe:** Inizializza due mappe, hashMap e robotList.

La mappa hashMap associa un indice intero ad ogni nome di classe, mentre robotList associa ogni indice a una lista di oggetti MyData rappresentanti i dati dei robot.

**3. Organizzazione dei Livelli:** Per ogni classe, recupera i suoi livelli e li organizza in una lista chiamata *evo*. Questa lista è una rielaborazione della lista dei livelli, organizzata per rappresentare sia i livelli attuali che le evoluzioni future dei robot.

**4. Creazione degli Oggetti MyData:** Per ogni livello, crea un oggetto MyData, che contiene informazioni sui livelli e le evoluzioni, e lo aggiunge alla lista *struttura*, che viene poi associata alla classe corrispondente nella mappa robotList.

**5. Disponibilità dei Dati alla Vista:** Aggiunge le mappe hashMap e robotList al modello, rendendole disponibili alla vista associata.

In conclusione, il codice si occupa di verificare l'autenticazione dell'utente e, una volta confermata, prepara e organizza i dati riguardanti le diverse classi di robot, rendendoli disponibili per la visualizzazione nella pagina *AllRobot*.

```

117 @GetMapping("/all_robots")
118 public String AllRobotsPage(Model model, @CookieValue(required = false) String jwt){}
119     MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
120     formData.add("key", "jwt");
121     Boolean isAuthenticated = restTemplate.postForObject(url:"http://t23-g1-app-1:8080/validateToken", formData, responseType:Boolean.class);
122     if(isAuthenticated == null || !isAuthenticated) return "redirect:/login";
123
124     List<ClassUT> classes = getClasses();
125     Map<Integer, String> hashMap = new HashMap<>();
126     Map<Integer, List<MyData>> robotList = new HashMap<>();
127     for (int i = 0; i < classes.size(); i++) {
128         String valore = classes.get(i).getName();
129         List<String> levels = getLevels(valore);
130         System.out.println(levels);
131         List<String> evo = new ArrayList<>(); //aggiunto
132         for(int j = 0; j<levels.size(); j++){
133             if(j>levels.size()/2)
134                 evo.add(j,levels.get(j-(levels.size()/2)));
135             else{
136                 evo.add(j,levels.get(j+(levels.size()/2)));
137             }
138         }
139         System.out.println(evo);
140         List<MyData> struttura = new ArrayList<>();
141         for(int j = 0; j<levels.size(); j++){
142             MyData strutt = new MyData(levels.get(j),evo.get(j));
143             struttura.add(j,strutt);
144         }
145         for(int j = 0; j<struttura.size(); j++)
146             System.out.println(struttura.get(j).getList1());
147         hashMap.put(i, valore);
148         robotList.put(i, struttura);
149         //evosuiteLevel.put(i, evo);
150     }
151     model.addAttribute(attributeName:"hashMap", hashMap);
152     model.addAttribute(attributeName:"hashMap2", robotList);
153     return "all_robots";
154 }
```

Figure 9.8: GUIController: AllRobots

- GameMode:** Il codice illustrato gestisce la richiesta GET per la pagina game\_mode.

Inizialmente, verifica l'autenticazione dell'utente inviando una richiesta POST al servizio `http://t23-g1-app-1:8080/validateToken` per validare il token JWT. Se l'utente non è autenticato (ovvero se il token JWT non è valido o non è presente), l'utente viene reindirizzato alla pagina di login. Se l'autenticazione è confermata, il metodo restituisce il nome della vista `game_mode`, che corrisponde a una pagina HTML. Questa pagina rappresenta una schermata di selezione delle modalità di gioco per l'utente.

```
155 @GetMapping("/game_mode")
156 public String GameModePage(Model model, @CookieValue(required = false) String jwt){
157     MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
158     formData.add("jwt", jwt);
159
160     Boolean isAuthenticated = restTemplate.postForObject(url:"http://t23-g1-app-1:8080/validateToken", formData, responseType:Boolean.class);
161
162     if(isAuthenticated == null || !isAuthenticated) return "redirect:/login";
163     return "game_mode";
164 }
165 }
```

Figure 9.9: GUIController: GameMode

## Descrizione delle chiamate API

### 9.3.3 Verifica dell'Autenticazione

- **Metodo HTTP:** POST
- **URL:** `http://t23-g1-app-1:8080/validateToken`
- **Descrizione:** Questo servizio verifica la validità del token JWT per confermare l'identità dell'utente.

### 9.3.4 Recupero della Classifica dei Giocatori

- **Metodo HTTP:** GET
- **URL:** `http://t4-g18-app-1:3000/players`
- **Descrizione:** Fornisce una lista ordinata dei giocatori in base alle vittorie, in formato JSON.

### 9.3.5 Recupero delle Informazioni sui Robot

- **Metodo HTTP:** GET

- **URL:** `http://manvsclass-controller-1:8080/home`
- **Descrizione:** Restituisce dati sulle classi dei robot, inclusi nomi e livelli di difficoltà.

### 9.3.6 Recupero dell'ID dell'Utente Autenticato

- **Metodo HTTP:** GET e POST
- **URL:** `http://t23-g1-app-1:8080/IdToken`
- **Descrizione:** Restituisce l'ID numerico dell'utente associato al token JWT.

### 9.3.7 Recupero del Nome dell'Utente Autenticato

- **Metodo HTTP:** POST
- **URL:** `http://t23-g1-app-1:8080/nameToken`
- **Descrizione:** Estrae e restituisce il nome dell'utente dal token JWT.

## 9.4 Chiamate interne all'applicazione - lista API

	<b>Metodo</b>	<b>URL</b>	<b>Scopo</b>
1	GET	/main	Recupero della pagina principale
2	GET	/classifica	Recupero della pagina della classifica
3	GET	/all_robots	Recupero della pagina di tutti i robot
4	GET	/game_mode	Recupero della pagina della modalità di gioco
5	GET	/storico	Recupero della pagina dello storico
6	GET	/choose	Recupero della pagina di selezione
7	GET	/report	Recupero della pagina del report
8	GET	/report1	Recupero della pagina del report 1
9	POST	/save-data	Salvataggio dei dati del gioco
10	GET	/editor	Recupero della pagina dell'editor

Table 9.1: Chiamate interne all'applicazione

#### 9.4.1 GameDataWriter

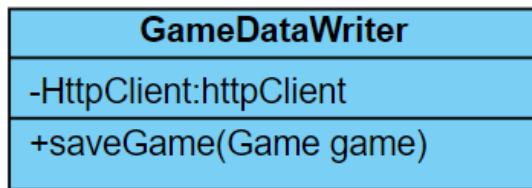


Figure 9.10: GameDataWriter modifiche effettuate - Diagramma delle Classi

Il GameDataWriter è stato modificato per permettere di aggiungere il nome del giocatore all'interno della tabella player

#### 9.4.2 Codice: GameDataWriter

La classe GameDataWriter è responsabile della creazione e del salvataggio delle informazioni relative a una sessione di gioco. Ciò include l'inizializzazione e l'aggiornamento delle informazioni nelle tabelle del database relative a *Game*, *Turn*, e *Round*, necessarie per memorizzare i dettagli di ogni partita.

Una parte significativa di questo processo è l'aggiornamento delle informazioni nella tabella *Player*. Questo viene realizzato attraverso una richiesta PUT all'endpoint `http://t4-g18-app-1:3000/players/{id}`, dove `{id}` è l'ID del giocatore ottenuto tramite il metodo `game.getPlayerId()`. Nella richiesta PUT, viene incluso il nome del giocatore, ottenuto dal metodo `game.get playerName()`, che viene aggiunto all'utente corrispondente all'ID specificato nel database.

In conclusione, quindi, la classe GameDataWriter non solo gestisce il salvataggio delle informazioni di gioco, ma si occupa anche di aggiornare le informazioni del giocatore nel database, assicurandosi che il nome del giocatore sia correttamente memorizzato e associato al suo ID.

```

78     JSONObject round = new JSONObject();
79     round.put("gameId", game_id);
80     round.put("testClassId", game.getClasse());
81     round.put("name","startedAt", time);
82     //-----MODIFICA A9-----
83     JSONObject newPlayerObj = new JSONObject();
84     newPlayerObj.put("name",game.getPlayerName());
85
86     String newPlayerUrl ="http://t4-g18-app-1:3000/players/" + game.getPlayerId();
87
88     HttpPut httpPutPlayer = new HttpPut(newPlayerUrl);
89     StringEntity jsEntityPlayer = new StringEntity(newPlayerObj.toString(),ContentType.APPLICATION_JSON);
90     httpPutPlayer.setEntity(jsEntityPlayer);
91
92     HttpResponse newPlayerResponse = httpClient.execute(httpPutPlayer);
93     int newPlayerStatusCode = newPlayerResponse.getStatusLine().getStatusCode();
94     //----- FINE MODIFICA A9-----
95     httpPost = new HttpPost(uri:"http://t4-g18-app-1:3000/rounds");
96     jsonEntity = new StringEntity(round.toString(), ContentType.APPLICATION_JSON);
97
98     httpPost.setEntity(jsonEntity);
99

```

Figure 9.11: GameDataWriter - modifiche effettuate

#### 9.4.3 Game



Figure 9.12: Game modifiche effettuate - Diagramma delle Classi

La classe Game è stata modificata aggiungendo l'attributo playerName e il metodo getPlayerName per la gestione del nome del giocatore

#### 9.4.4 Codice: Game

La classe Game è progettata per rappresentare una partita all'interno dell'applicazione. Il suo costruttore è stato aggiornato per includere un nuovo parametro, playerName. Questa modifica implica che, per istanziare un oggetto Game, ora è necessario fornire il nome del giocatore, oltre alle informazioni preesistenti quali playerId, description, name e difficulty. L'inclusione del playerName come un attributo fondamentale dell'oggetto Game sottolinea l'importanza di associare ogni partita con l'identità specifica del giocatore.

In aggiunta, è stato implementato un metodo getter, getPlayerName(), che fornisce l'accesso al nome del giocatore associato a un'istanza di Game. Questa funzionalità consente un facile recupero del nome del giocatore, mantenendo il principio dell'incapsulamento in programmazione orientata agli oggetti. Nel complesso, queste modifiche migliorano significativamente la gestione delle informazioni relative ai giocatori negli oggetti Game, facilitando la memorizzazione e l'accesso ai dettagli specifici del giocatore per ogni partita.

```

1  public Game(int playerId, String playerName, String description, String name, String difficulty) {
2      this.playerId = playerId;
3      this.playerName = playerName; //aggiunta A9
4      this.description = description;
5      this.name = name;
6      this.difficulty = difficulty;
7      this.classe = "";
8  }
9
10 public int getPlayerId() {
11     return playerId;
12 }
13
14 public String getPlayerName(){ //aggiunta A9
15     return playerName;
16 }
17
18 public void setPlayerId(int playerId) {
19     this.playerId = playerId;
20 }
21
22 public long getId() {
23     return id;
24 }
```

Figure 9.13: Game - modifiche effettuate

## 9.5 Pagine create

- Pagina Main:

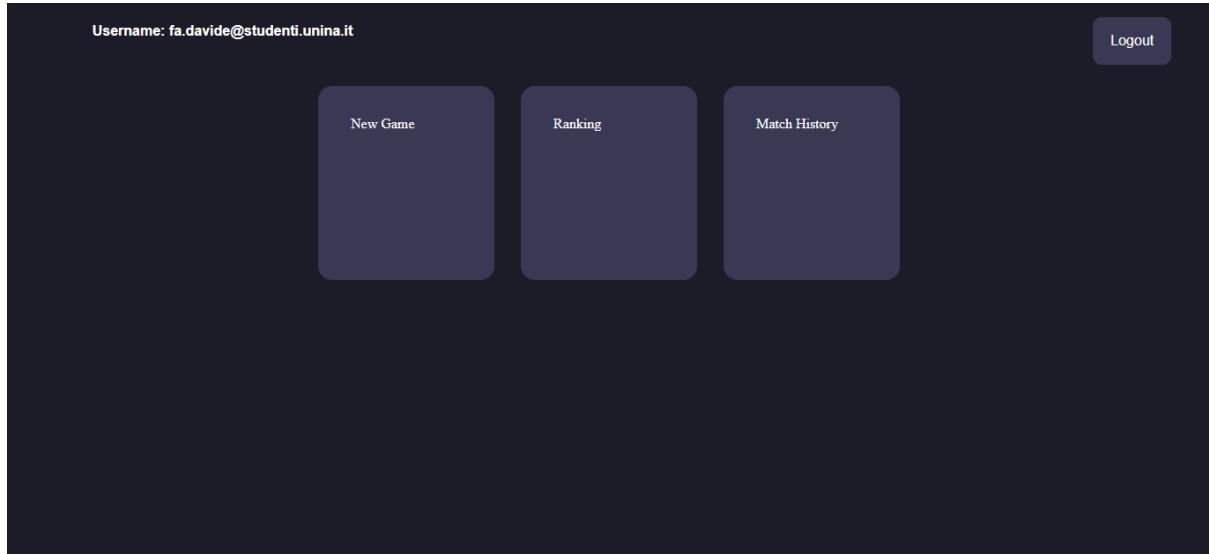


Figure 9.14: Layout Pagina Main

- Pagina Visualizza Classifica:

ID Giocatore	Nome Giocatore	Partite Vinte	Partite Effettuate
1	Fabio	30	36
7	FabioProva	7	7
3	Fabi	6	11
2	prova	5	7
11	wecafap	4	4
8	ProvaEmailProv	2	2
12	moxicFabioProva	2	2
6	FabioDe	1	1
13	gawadProva	1	2
5	FabioDavide	1	1

Figure 9.15: Layout Classifica

## CHAPTER 9. MODIFICHE AL TASK5

- Pagina Visualizza Storico:

The screenshot shows a dark-themed web application interface. At the top left, it displays the username "fa.davide@studenti.unina.it". On the top right, there is a "Logout" button. Below this, the text "Storico Partite Utente: Fabi" is displayed. The main content is a table titled "Visualizza Storico" with the following columns: ID Partita, Inizio Partita, Fine Partita, Classe Testata, Robot Sfidato, Risultato Partita, Difficoltà, and Scores. The table contains 10 rows of data, each representing a game record with its details.

ID Partita	Inizio Partita	Fine Partita	Classe Testata	Robot Sfidato	Risultato Partita	Difficoltà	Scores
151	2024-01-19T17:36:09.831392Z	2024-01-19T17:42:30.337417Z			Vittoria		100
173	2024-01-22T09:54:04.924308Z	2024-01-22T09:55:49.358062Z			Vittoria		100
176	2024-01-22T11:52:12.070979Z	2024-01-22T11:55:44.823123Z			Vittoria		100
195	2024-01-26T12:34:02.247413Z	2024-01-26T12:36:04.603174Z	VCardBean	randoop	Sconfitta	3	77
197	2024-01-26T12:39:28.698683Z	2024-01-26T12:45:25.022298Z	Calcolatrice	randoop	Vittoria	1	100
204	2024-01-26T16:46:46.13121Z	2024-01-26T16:48:38.608909Z	VCardBean	evoSuite	Sconfitta	2	32
205	2024-01-26T16:49:23.538466Z	2024-01-26T16:52:07.882028Z	VCardBean	evoSuite	Sconfitta	2	9
206	2024-01-26T16:52:10.61515Z	2024-01-26T16:55:10.046151Z	VCardBean	evoSuite	Sconfitta	2	32
207	2024-01-26T16:55:13.959607Z	2024-01-26T16:57:08.336462Z	VCardBean	evoSuite	Sconfitta	2	32

Figure 9.16: Layout Storico

- Pagina Game Mode:

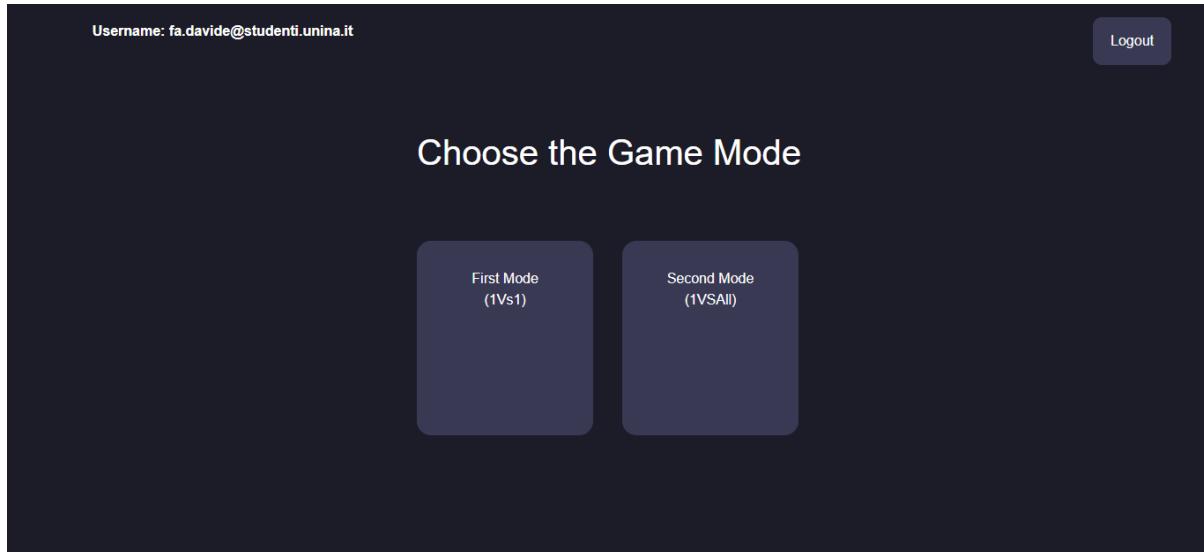


Figure 9.17: Layout scelta modalità di gioco

- Pagina All Robots:

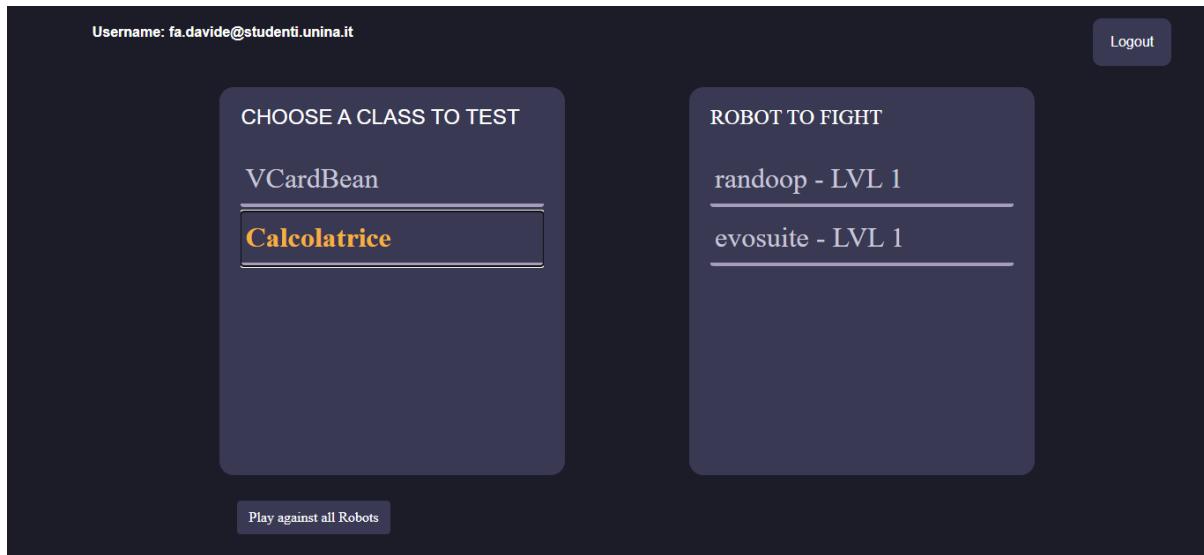


Figure 9.18: Layout scelta classe con tutti i robot

- Pagina Report:

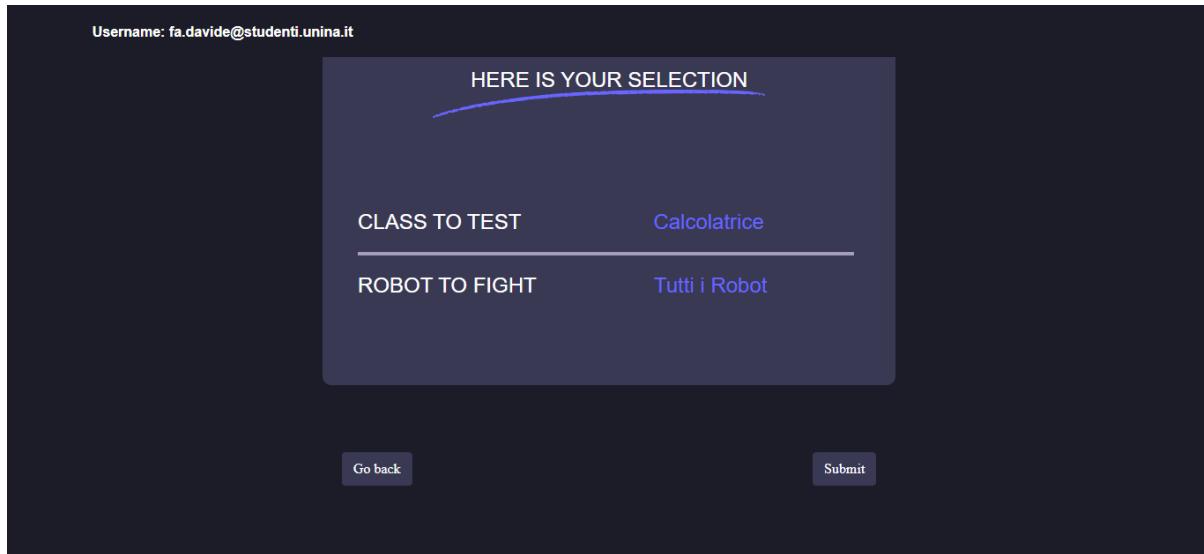


Figure 9.19: Layout report contro tutti i Robot

# Chapter 10

## Modifiche al Task 6

### 10.1 Diagramma dei package

Un **diagramma dei package** è un tipo di diagramma UML utilizzato per rappresentare e organizzare i diversi elementi di un sistema software in "package". Ogni package raggruppa elementi correlati, come classi, interfacce e altri package, secondo la loro funzionalità, livelli di accesso o utilizzo all'interno del sistema. Questo tipo di diagramma aiuta a visualizzare l'architettura del software, delineando una struttura modulare che facilita la comprensione delle dipendenze e delle relazioni tra i componenti.

Analizzando il diagramma fornito, possiamo osservare una struttura organizzata in tre aree principali: Editor, Controller e ExternalService. All'interno dell'area "Editor", troviamo package che rappresentano componenti dell'interfaccia utente, come "Windows", "Buttons" e "Code", che indicano la separazione delle responsabilità nell'ambiente di editing. I package come "myController.java" e "mainController.java" all'interno della sezione "Controller" suggeriscono l'esistenza di classi Java destinate a gestire la logica di business e l'interazione tra l'utente e il sistema. Infine, la sezione "ExternalService" comprende componenti come "GameEngine" e "GameInfoRepository", che possono essere responsabili per la gestione delle regole di gioco e la persistenza dei dati, rispettivamente. L'uso di package come "RobotTestGenerator" con elementi "EvoSuite" e "Randoop" implica inoltre l'integrazione di strumenti di test automatici all'interno del sistema.

In sintesi, questo diagramma dei package presenta una chiara divisione delle funzionalità del sistema, facilitando la comprensione di come gli elementi interagiscono all'interno dell'architettura software.

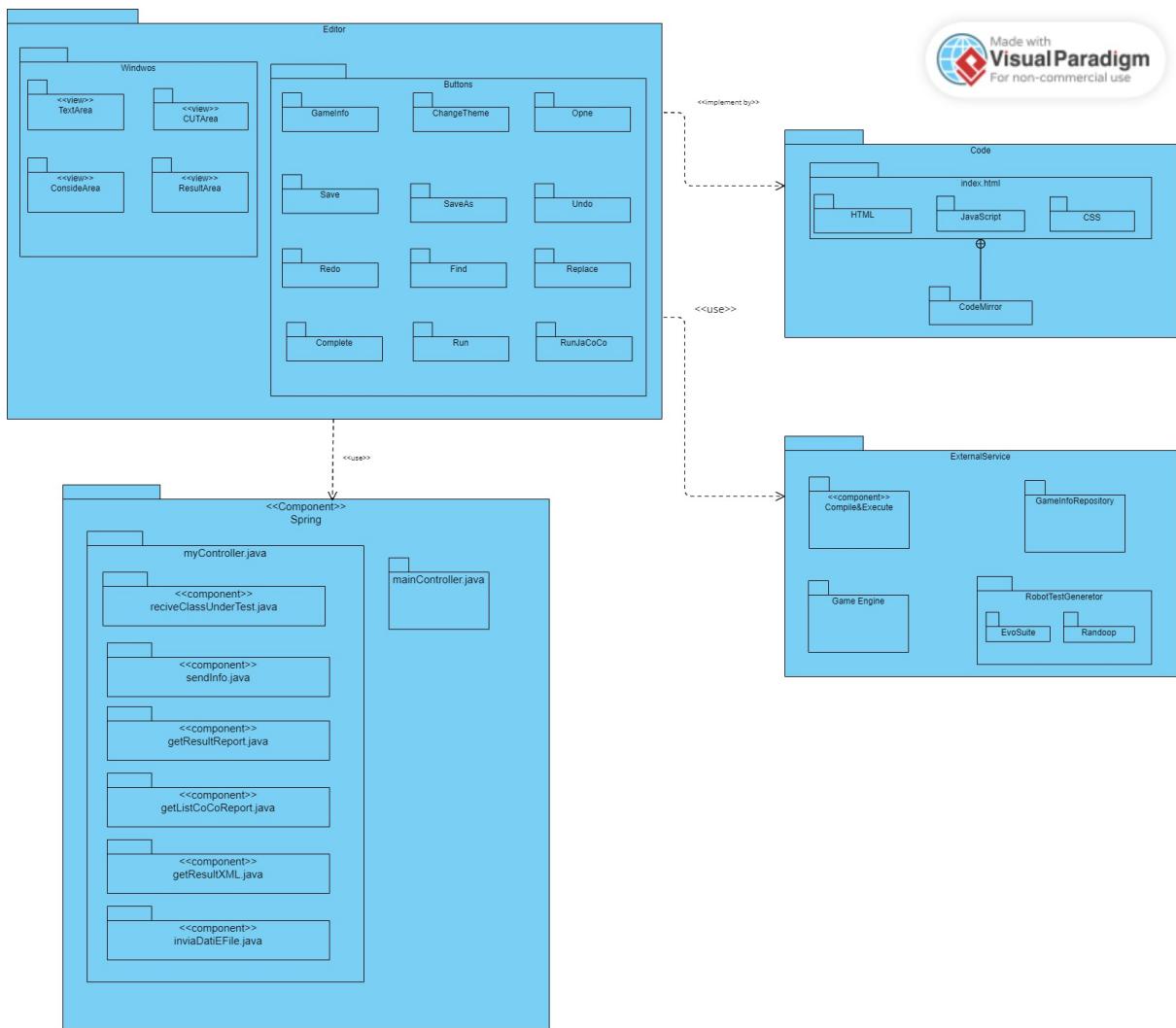


Figure 10.1: Diagramma dei package T6

Questo diagramma è stato il punto di partenza per la comprensione e la successiva modifica del Task6. Purtroppo nella documentazione del Task6 non era presente un diagramma delle classi. Si è quindi proceduto a sviluppare solo i diagrammi delle classi modificate.

## 10.2 MyController

All'interno del component spring siamo andati a modificare la classe MyController.java.

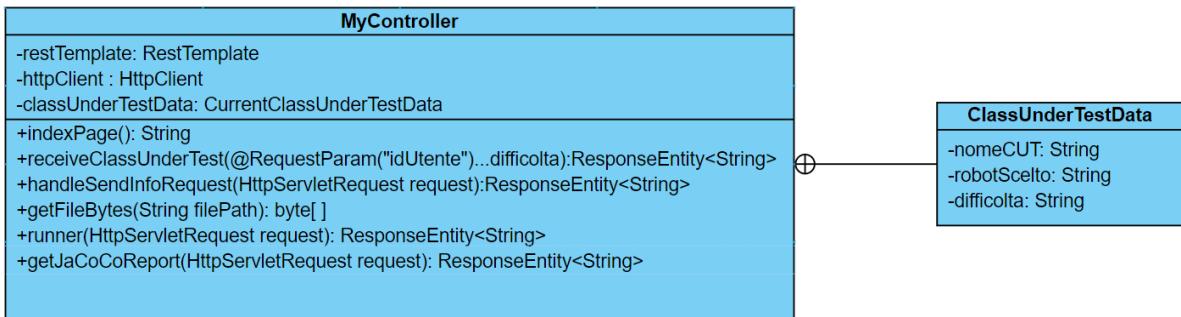


Figure 10.2: Classe MyController

La funzione modificata è runner, sono state aggiunte le tre variabili di seguito:

- `nomeCUT`: Identifica una "Class Under Test". Viene utilizzato in `receiveClassUnderTest()` per selezionare e scaricare una classe da un servizio esterno per il testing.
- `robotScelto`: Rappresenta il robot di testing selezionato. Questo parametro è cruciale nella funzione `runner()` per orchestrare i test, in combinazione con `nomeCUT` e `difficoltà`.
- `difficoltà`: Indica il livello di difficoltà del test. In `runner()`, determina le condizioni di esecuzione e valutazione dei test.

Questa struttura è necessaria per ottenere poi le informazioni nello storico.

## 10.3 Codice: MyController

In questa sezione, abbiamo proceduto con aggiornamenti mirati alla tabella “Turn” del database (T4). L’obiettivo principale è stato quello di registrare nella tabella tutti quei parametri di gioco che, nonostante fossero disponibili, non venivano salvati nel database e, di conseguenza, non erano visualizzabili nell’applicazione.

I parametri specifici che abbiamo deciso di raccogliere al termine di ogni turno di gioco includono la “Classe Testata”, il “Robot Sfidato” e il livello di “Difficoltà”. Queste aggiunte sono state

pensate per migliorare la raccolta dei dati storici delle partite. L'obiettivo era quello di ottenere un archivio delle partite più completo e dettagliato, migliorando così la qualità e l'utilità delle informazioni disponibili.

Di seguito vengono illustrate le parti del codice modificato

```
private static class ClassUnderTestData{
    private String nomeCUT;
    private String robotScelto;
    private String difficolta;
}

private ClassUnderTestData currentClassUnderTestData;
@Autowired
public MyController(RestTemplate restTemplate) {
    this.restTemplate = restTemplate;
}

@GetMapping("/")
public String indexPage() {
    return "index";
}

@GetMapping("/receiveClassUnderTest")
public ResponseEntity<String> receiveClassUnderTest(
    @RequestParam("idUtente") String idUtente,
    @RequestParam("idPartita") String idPartita,
    @RequestParam("idTurno") String idTurno,
    @RequestParam("nomeCUT") String nomeCUT,
    @RequestParam("robotScelto") String robotScelto,
    @RequestParam("difficolta") String difficolta) {
    try {
```

Figure 10.3: Class UnderTestData

## CHAPTER 10. MODIFICHE AL TASK 6

---

```
250
251
252 //Codice Aggiunto
253 String nomeCUT = currentClassUnderTestData.nomeCUT;
254 String robotScelto = currentClassUnderTestData.robotScelto;
255 String difficolta = currentClassUnderTestData.difficolta;
256
257 // conclusione e salvataggio partita
258 // chiusura turno con vincitore
259 HttpPut httpPut = new HttpPut("http://t4-g18-app-1:3000/turns/" + String.valueOf(request.getParameter(name:"turnId")));
260
261 obj = new JSONObject();
262 obj.put(name:"scores", String.valueOf(userScore));
263
264 if (roboScore > userScore) {
265     obj.put(name:"isWinner", value:false);
266 } else {
267     obj.put(name:"isWinner", value:true);
268 }
269 String time = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
270 obj.put(name:"closedAt", time);
271 obj.put(name:"testClass", nomeCUT);
272 obj.put(name:"robot", robotScelto);
273 obj.put(name:"difficulty", difficolta);
274
275 jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);
276
277 httpPut.setEntity(jsonEntity);
278
279 response = httpClient.execute(httpPut);
280 httpPut.releaseConnection();
281
282
283
284
285
286
287 }
```

Figure 10.4: Modifiche al metodo Runner

# Chapter 11

## Deployment

Il diagramma sottostante, illustra la disposizione fisica dei vari componenti di un sistema software e di come essi interagiscono tra loro all'interno di un ambiente di esecuzione. Questo sistema software è ospitato su un server, il quale utilizza il sistema operativo Windows 11. Inoltre, il software è eseguito all'interno di un ambiente Docker. Come già descritto precedentemente essa è una piattaforma di containerizzazione. Pertanto, possiamo affermare che questo sistema è distribuito quindi su diversi container. Si è deciso di non utilizzare colorazioni diverse in quanto, non è stato modificato nulla, rispetto alla documentazione T11-G41 la quale abbiamo preso come riferimento di partenza. All'interno di esso, bisogna fare particolare attenzioni a diversi componenti:

1. **La rete.** Essa è condivisa dai container di tutti i Task, ciò permette di isolare i container dalla rete dell'host e al tempo stesso permettere la comunicazione tra di essi (si ottiene una rete virtuale): l'unico container che è esposto verso l'esterno è il Gateway che si occupa di gestire le richieste.
2. **Il Volume T8**, il quale è condiviso dai container dei Task 1 e 8, la parte del File System in comune è quella relativa alle classi e alla copertura generata da Evosuite.
3. **Il volume T9**, che come il Volume T8, è anch'esso condiviso da 2 container. In questo caso si vanno a considerare i container dei Task 1 e 9, questo perché entrambi i Task devono accedere alla stessa porzione di File System: quella relativa alle classi e alla copertura

generata da Randoop e EvoSuite.

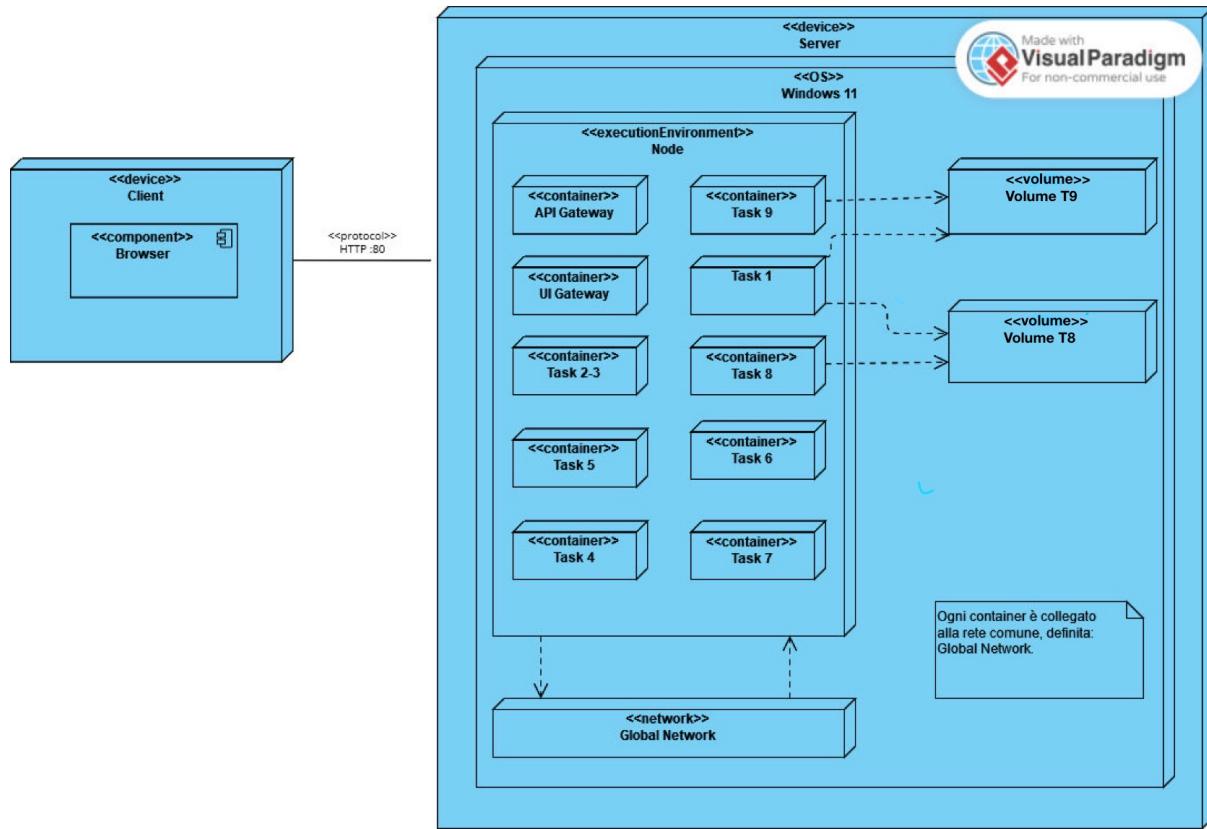


Figure 11.1: Deployment

# Chapter 12

## Testing

### 12.1 Testing End to End

L’obiettivo principale del Test End-to-End (E2E), noto anche come Test di Sistema, è quello di integrare efficacemente i nuovi componenti con quelli già esistenti all’interno del sistema, assicurandosi che l’applicazione funzioni correttamente nel suo insieme. Questo tipo di test valuta l’intera applicazione simulando scenari reali d’uso, al fine di verificare il corretto funzionamento di tutte le sue parti, che includono l’interfaccia utente, i servizi di backend, i database e le comunicazioni di rete.

L’obiettivo del test E2E è quello di convalidare il comportamento complessivo dell’applicazione, tenendo conto di aspetti fondamentali come le funzionalità, l’affidabilità, le prestazioni e la sicurezza. Lo scopo finale è identificare eventuali difetti o problemi che potrebbero sorgere durante l’interazione tra le diverse componenti dell’applicazione.

Per eseguire il test E2E, sono stati impiegati strumenti specifici. Tra questi, **Selenium** è stato utilizzato per il testing automatizzato dell’ applicazione web, **JUnit** è stato scelto come framework di testing e **Google Chrome** è stato usato come browser per accedere e testare l’applicazione. Questi strumenti aiutano a garantire un processo di testing efficace e accurato, contribuendo a migliorare la qualità e l’affidabilità dell’applicazione.

### 12.1.1 Casi di Test

Di seguito sono riportati i vari test effettuati. I test implementati si sono concentrati esclusivamente sulle nuove modifiche apportate al sistema. Non è stata effettuata una nuova serie di test sulle componenti preesistenti, in quanto queste erano già state sottoposte a test in precedenza.

### 12.1.2 Setup

Le operazioni coinvolte nella fase di configurazione sono le seguenti:

1. Configurazione Iniziale del WebDriver
2. Apertura e Configurazione del Browser
3. Accesso all'applicazione Web e Login
4. Verifica della Redirezione Post-Login
5. Chiusura del Browser

```

33  @BeforeClass
34  public static void setDriver() {
35      System.setProperty("webdriver.chrome.driver", "C:\\\\Users\\\\Fabio.DESKTOP-IG5F0B6\\\\Desktop\\\\chromedriver.exe");
36  }
37
38  @Before
39  public void openBrowser(){
40      ChromeOptions options = new ChromeOptions();
41      options.setCapability(CapabilityType.UNEXPECTED_ALERT_BEHAVIOUR, UnexpectedAlertBehaviour.ACCEPT);
42      HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
43      chromePrefs.put("profile.default_content_settings.popups", 0);
44      chromePrefs.put("download.default_directory", "C:\\\\Users\\\\Fabio.DESKTOP-IG5F0B6\\\\Desktop");
45      options.setExperimentalOption(name:"prefs", chromePrefs);
46
47      driver = new ChromeDriver(options);
48      driver.manage().timeouts().implicitlyWait(timeout, TimeUnit.SECONDS);
49
50      driver.get(url:"http://localhost/login");
51      driver.findElement(By.id("email")).sendKeys(...keysToSend:"fabiodavide97@gmail.com");
52      driver.findElement(By.id("password")).sendKeys(...keysToSend:"Anonimo1!");
53      driver.findElement(By.cssSelector(cssSelector:"input[type=submit]")).click();
54
55      WebDriverWait wait = new WebDriverWait(driver, timeout);
56
57      String urlPaginaDiRedirezione = "http://localhost/main";
58      try {
59          wait.until(ExpectedConditions.urlToBe(urlPaginaDiRedirezione));
60      } catch(TimeoutException e) {
61          Assert.fail();
62      }
63  }
64
65  @After
66  public void closeBrowser(){
67      driver.close();
68  }

```

Figure 12.1: Setup

### 12.1.3 Classifica test

Questo test si concentra sulla verifica della navigazione e delle funzionalità della pagina Classifica, controllando gli elementi chiave come la tabella di classifica e il flusso di logout. Le operazioni coinvolte sono le seguenti:

1. Verifica dell'URL Corrente
2. Interazione con Elementi della Pagina
3. Attesa della Redirezione alla Pagina Classifica
4. Verifica della Visibilità della Tabella Classifica
5. Verifica della Presenza degli elementi nella Tabella
6. Logout dall'Applicazione

```

78 //TEST PAGINA CLASSIFICA
79 @Test
80 public void testClassifica() {
81
82     assertEquals(expected:"http://localhost/main", driver.getCurrentUrl());
83
84     driver.findElement(By.id(id:"downloadButton2")).click();
85
86     new WebDriverWait(driver, timeout).until(ExpectedConditions.urlToBe(url:"http://localhost/classifica"));
87
88     assertEquals(expected:"http://localhost/classifica", driver.getCurrentUrl());
89
90     new WebDriverWait(driver, timeout).until(ExpectedConditions.visibilityOfElementLocated(By.id(id:"classificaTable")));
91
92     List<WebElement> tableRows = driver.findElements(By.xpath(xpathExpression:"//table[@id='classificaTable']/tbody/tr"));
93     assertFalse(message:"La tabella non è popolata", tableRows.isEmpty());
94
95     WebElement logoutButton = new WebDriverWait(driver, timeout).until(ExpectedConditions.elementToBeClickable(By.id(id:"logoutButton")));
96     logoutButton.click();
97     new WebDriverWait(driver, timeout).until(ExpectedConditions.urlToBe(url:"http://localhost/login"));
98     assertEquals(expected:"http://localhost/login", driver.getCurrentUrl());
99 }

```

Figure 12.2: Classifica Test

### 12.1.4 Storico test

Questi test verificano le funzionalità di navigazione alla pagina storico, la corretta visualizzazione e il popolamento della tabella storico, oltre a testare la funzionalità di logout dall'applicazione. Le operazioni coinvolte sono le seguenti:

1. Verifica dell'URL Corrente

2. Interazione con Elementi della Pagina
3. Attesa della Redirezione alla Pagina Storico
4. Verifica della Visibilità della Tabella Storico
5. Verifica della Presenza degli elementi nella Tabella
6. Logout dall'Applicazione

```

99 //TEST PAGINA STORICO
100 @Test
101 public void navigateToStoricoPage() {
102
103     assertEquals(expected:"http://localhost/main", driver.getCurrentUrl());
104
105     WebElement storicoButton = driver.findElement(By.id("downloadButton3"));
106     storicoButton.click();
107
108     WebDriverWait wait = new WebDriverWait(driver, timeout);
109     wait.until(ExpectedConditions.urlContains(fraction:"http://localhost/storico"));
110 }
111
112 public void testStoricoPage() {
113     navigateToStoricoPage();
114
115
116     WebElement storicoTable = driver.findElement(By.id("storicoTable"));
117     Assert.assertTrue(storicoTable.isDisplayed());
118
119
120     WebDriverWait wait = new WebDriverWait(driver, timeout);
121     wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(cssSelector:"#storicoTable tbody tr")));
122
123     List<WebElement> rows = storicoTable.findElements(By.tagName(tagName:"tr"));
124     Assert.assertTrue(rows.size() > 0);
125
126 }
127
128 }
129

```

Figure 12.3: Storico Test

### 12.1.5 Nuova Partita test

Questi test verificano le funzionalità di navigazione all'interno dell'applicazione, partendo dalla pagina principale e arrivando alle pagine di scelta delle due modalità di gioco. Si concentrano sulla corretta navigazione e sul caricamento delle pagine desiderate, validando i cambiamenti degli URL per confermare che l'utente sia stato reindirizzato alla pagina corretta. Le operazioni coinvolte sono le seguenti:

1. Navigazione alla pagina Modalità di Gioco
2. Test della Navigazione alla Pagina che permette di scegliere la modalità 1VS1 (<http://localhost/choose>)

**3. Test della Navigazione alla Pagina che permette di scegliere la modalità 1VS-tutti ([http://localhost/all\\_robots](http://localhost/all_robots))**

**4. Logout dall'Applicazione**

```

140 //TEST PAGINA NUOVA PARTITA
141 public void navigateToGameMode() {
142     assertEquals(expected:"http://localhost/main", driver.getCurrentUrl());
143     driver.findElement(By.id("downloadButton1")).click();
144     new WebDriverWait(driver, timeout).until(ExpectedConditions.urlToBe(url:"http://localhost/game_mode"));
145     assertEquals(expected:"http://localhost/game\_mode", driver.getCurrentUrl());
146 }
147
148 @Test
149 public void testNavigateToChoose() {
150     navigateToGameMode();
151     driver.findElement(By.id("nuovaPartitaButton")).click();
152     new WebDriverWait(driver, timeout).until(ExpectedConditions.urlToBe(url:"http://localhost/choose"));
153     assertEquals(expected:"http://localhost/choose", driver.getCurrentUrl());
154 }
155
156 @Test
157 public void testNavigateToAllRobots() {
158     navigateToGameMode();
159     driver.findElement(By.id("classificaButton")).click();
160     new WebDriverWait(driver, timeout).until(ExpectedConditions.urlToBe(url:"http://localhost/all\_robots"));
161     assertEquals(expected:"http://localhost/all\_robots", driver.getCurrentUrl());
162 }
163

```

Figure 12.4: Nuova Partita Test

### 12.1.6 Scelta classi e robot test

Questo test verifica la funzionalità di interazione con i pulsanti nella pagina "Choose", controllando che i pulsanti delle classi e dei robot siano presenti e che rispondano correttamente ai clic, evidenziando la selezione effettuata dall'utente. Le operazioni coinvolte sono le seguenti:

1. **Navigazione alla Pagina Choose (<http://localhost/choose>)**
2. **Interazione e Verifica dei Pulsanti delle Classi**
3. **Interazione e Verifica dei Pulsanti dei Robot**
4. **Logout dall'Applicazione**

```

● 164 //TEST PAGINA CHOOSE
165 @Test
166 public void testButtonInteractions() {
167     driver.get(url:"http://localhost/choose");
168
169
170     List<WebElement> classButtons = driver.findElements(By.className(className:"lista-item"));
171     assertFalse(classButtons.isEmpty());
172     WebElement firstClassButton = classButtons.get(0);
173     firstClassButton.click();
174
175
176     assertTrue(firstClassButton.getAttribute(name:"class").contains("highlighted"));
177
178
179     WebElement firstRobotButton = driver.findElement(By.cssSelector(cssSelector:"button.lista-item.my-button"));
180     firstRobotButton.click();
181
182
183     assertTrue(firstRobotButton.getAttribute(name:"class").contains("highlighted"));
184
185 }
```

Figure 12.5: Scelta classi e robot test

### 12.1.7 Risultato del Test

Di seguito si riporta il risultato totale dei test effettuati con l'output di Maven

```

INFO: Detected dialect: W3C
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 81.44 s --
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:39 min
[INFO] Finished at: 2024-01-29T01:22:06+01:00
[INFO] -----
```

Figure 12.6: Output Test

# Chapter 13

## Guida all'installazione

### 13.1 Docker e WSL

Docker è una piattaforma di containerizzazione che consente la creazione, distribuzione e gestione di applicazioni in modo efficiente e isolato, garantendo portabilità indipendentemente dall'ambiente di esecuzione. In ambiente Windows, l'uso di Docker richiede la Windows Subsystem for Linux (WSL), che permette di eseguire un ambiente Linux dentro Windows. Docker crea immagini e container in ambienti isolati, facilitando l'esecuzione di servizi distinti e la loro intercomunicazione tramite mappatura dei porti. Questo approccio promuove modularità, scalabilità e semplifica la distribuzione delle applicazioni. Inoltre, Docker consente di includere file system locali nei container, attraverso percorsi assoluti, e di creare istanze di macchine virtuali, ottimizzando l'uso delle risorse del file system.

### 13.2 Installazione

#### 13.2.1 Passo 1

Per l'installazione di Docker Desktop sul proprio sistema operativo, seguire le istruzioni di download ufficiali. È possibile che, durante l'avvio di Docker, si verifichi l'errore "*Docker desktop - unexpected wsl error*". In tal caso, eseguire nel terminale il comando `wsl -shutdown` e poi riavviare il computer. Se il problema persiste, potrebbe essere necessario installare o aggiornare il Windows Subsystem for Linux (WSL) eseguendo il comando `wsl -install` nel

terminale. Assicurarsi di avere la versione più recente del WSL per garantire la compatibilità e il funzionamento ottimale di Docker Desktop.

### 13.2.2 Passo 2

Per l'installazione su Windows, è sufficiente avviare lo script `installer.bat`. Per macOS, nella cartella contenente il file `installer-mac.sh`, eseguire il comando `chmod +x installer-mac.sh` per renderlo eseguibile, seguito da `./installer-mac.sh` per avviare l'installazione. Il processo di installazione comporta:

1. Creazione della rete "global-network" utilizzata da tutti i container.
2. Creazione di volumi condivisi ("VolumeT9" per i Task 1 e 9 e "VolumeT8" per i Task 1 e 8).
3. Creazione dei singoli container in Docker Desktop.

Dopo l'installazione, è necessario avviare tutti i container inattivi, ad eccezione di "ui gateway", che va avviato solo dopo l'attivazione degli altri. Importante: il container del Task 9 si arresta automaticamente dopo l'avvio, essendo usato solo per popolare il "VolumeT9" condiviso con il Task 1.

### 13.2.3 Passo 3

Per configurare il container "manvsclass-mongo-db-1" come descritto nella documentazione del Task 1, segui questi passaggi nel Docker Desktop:

1. Apri il terminale del container.
2. Esegui il comando `mongosh`.
3. Inserisci i seguenti comandi in sequenza:
  - `use manvsclass`
  - `db.createCollection("ClassUT")`
  - `db.createCollection("interaction")`
  - `db.createCollection("Admin")`

- db.createCollection("Operation")
- db.ClassUT.createIndex({difficulty: 1})
- db.Interaction.createIndex({name: "text", type: 1})
- db.interaction.createIndex({name: "text"})
- db.Admin.createIndex({username: 1})

Dopo aver completato questi passaggi, l'applicazione sarà completamente configurata e accessibile sulla porta 80.

### 13.2.4 Utilizzo

Una volta completata l'installazione e avviati i container, è possibile procedere alla registrazione attraverso la schermata di accesso. Prima di iniziare a giocare, un'azione cruciale è l'inserimento di una classe di test, compito riservato all'amministratore. Per questo, è fondamentale che l'amministratore si registri al sistema. Per facilitare questo processo, vengono fornite istruzioni dettagliate su come registrarsi e caricare la classe di test. Questa guida passo-passo assicura che l'amministratore possa eseguire con efficacia tali operazioni, indispensabili per il corretto funzionamento dell'applicazione.

#### Registrazione Admin

Per registrarsi come amministratore alla prima visita dell'applicazione, è necessario accedere alla schermata di registrazione tramite l'URL <http://localhost/registraAdmin>. In questa pagina web, l'amministratore può inserire i propri dati personali, come nome, cognome, username e password. Questo processo consente di creare un account amministratore per la gestione dell'applicazione.

#### Registrazione utente

Per accedere all'applicazione al primo utilizzo, l'utente deve visitare la pagina di login tramite l'URL <http://localhost/login>. Nella schermata di login, cliccando su “Non sei ancora registrato? Registrati.” si apre il modulo di registrazione. Qui, l'utente può inserire nome, cognome, email e password. È importante notare che la password deve soddisfare criteri specifici:

contenere almeno un carattere speciale, una lettera maiuscola, una minuscola, e un minimo di 8 caratteri. Dopo la registrazione e il successivo login, l'utente potrà scegliere tra le classi di test caricate dall'amministratore, sottoporle al sistema e sfidare il robot nel gioco per ottenere un livello di copertura del test maggiore.

# Chapter 14

## Glossary

### **AGILE**

Metodologia di sviluppo software che promuove lo sviluppo incrementale e iterativo.

### **API Gateway**

Server che funge da front-end API, ricevendo richieste API, applicando politiche di sicurezza e passando richieste al servizio back-end.

### **Bootstrap**

Framework open-source per lo sviluppo front-end per la realizzazione di siti web responsivi.

### **Chi**

Un router leggero e idiomático per la costruzione di servizi HTTP in Go.

### **Classifica**

Elenco ordinato di giocatori basato su criteri come partite giocate e vinte.

### **Controller**

Componente in architettura software, specialmente nel pattern MVC, che gestisce richieste in entrata e interazioni tra Model e View.

**Copertura delle linee di codice (LOC)**

Metrica per valutare le parti del codice eseguite durante il testing.

**CRUD**

Acronimo per Create, Read, Update, Delete, le quattro funzioni base della persistenza dei dati nel software.

**Daily SCRUM**

Riunione quotidiana nell'ambito della metodologia AGILE per discutere lo stato del progetto.

**Database**

Collezione sistematica di dati che supporta storage e manipolazione.

**ER Diagram (Entità-Relazione)**

Tecnica di modellazione dati che illustra entità di un sistema informativo e le loro relazioni.

**Game Engine**

Ambiente software per la costruzione di videogiochi.

**Game Repository**

Componente che archivia dati relativi al gioco, come statistiche e dati storici.

**GitHub**

Piattaforma per il versionamento e la gestione del codice sorgente con Git.

**Golang (Go)**

Linguaggio di programmazione open-source per programmazione di sistema e gestione database.

**Hook AfterSave**

Hook eseguito dopo il salvataggio di un record in contesti di database.

**Interfaccia Utente (UI)**

Parte di un sistema software con cui l'utente interagisce.

**JUnit**

Framework di testing per Java.

**JWT (JSON Web Token)**

Mezzo compatto per rappresentare asserzioni tra due parti.

**Maven**

Strumento di gestione e automazione del build per progetti Java.

**Metodologia**

Insieme di pratiche, tecniche e procedure per raggiungere obiettivi specifici.

**Microsoft Teams**

Piattaforma di collaborazione con chat, videoconferenze, storage di file e altre integrazioni.

**Miro**

Strumento di lavagna digitale per collaborazione e brainstorming.

**MongoDB**

Database NoSQL per la gestione di dati strutturati e non.

**MySQL**

Sistema di gestione di database relazionali open-source basato su SQL.

**NGINX**

Web server utilizzabile anche come reverse proxy, load balancer e cache HTTP.

**Pattern MVC (Model-View-Controller)**

Pattern architetturale per implementare interfacce utente.

**Player Repository**

Componente che gestisce dati dei giocatori in un'applicazione di gioco.

**PostgreSQL**

Sistema di gestione di database relazionali per gestire carichi di lavoro estesi e supportare transazioni complesse.

**Progettazione Architetturale**

Processo per concepire soluzioni strutturali in un progetto software.

**Requisiti Funzionali**

Specifiche che descrivono funzioni di un sistema software.

**Randoop**

Strumento per generazione automatica di test per Java.

**Robot**

Strumenti per generazione automatica di test in questa applicazione.

**Scalabilità**

Capacità di un sistema di gestire efficacemente l'aumento del carico di lavoro.

**Sprint Backlog**

Elenco di elementi selezionati dal backlog del prodotto per lo Sprint.

**Spring Boot**

Framework basato su Java per la creazione di applicazioni stand-alone e microservizi.

**Storico delle Partite**

Registro delle partite giocate con dettagli su ogni partita.

**Test Case**

Insieme di condizioni o variabili per determinare il funzionamento corretto di un sistema software.

**Testing**

Processo di esecuzione di un sistema per identificare difetti o carenze.

**Token JWT (JSON Web Token)**

Standard JSON per creare token di accesso che affermano alcune asserzioni.

**Turno**

Ogni tentativo di sottoporre un risultato in una partita in questa applicazione.

**UML (Unified Modeling Language)**

Linguaggio di modellazione per visualizzare design di un sistema.

**User Stories**

Descrizioni brevi delle funzionalità software dal punto di vista dell'utente.

**Visual Paradigm**

Strumento UML per modellazione aziendale e gestione processi di sviluppo.

**Visual Studio Code**

Editor di codice sorgente che supporta vari linguaggi di programmazione.

**Zuul**

Servizio di bordo open-source per routing dinamico, monitoraggio e sicurezza.