



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

ELABORATO DI SOFTWARE ARCHITECTURE DESIGN

Progetto ENACTEST: Classifiche

Anno Accademico 2024/2025

Gruppo B9 - Task #R2

Esposito Fabio

Matricola: M63001806

fabio.esposito46@studenti.unina.it

Grandioso Nicola

Matricola: M63001814

n.grandioso@studenti.unina.it

Iovine Giovanna

Matricola: M63001821

giovan.iovine@studenti.unina.it

Fontanella Gianluca

Matricola: M63001818

gianlu.fontanella@studenti.unina.it

Contatti:

Link Repository GitHub del progetto: **<https://github.com/Gianlucaf38/A13>**

Indice

1 Punto di partenza e Obiettivi del Progetto	1
1.1 Descrizione versione di partenza	2
1.2 Nuovi requisiti assegnati	5
1.3 Processo di sviluppo seguito	6
2 Analisi dei Requisiti	7
2.1 Glossario	7
2.2 Revisione dei Requisiti	8
2.2.1 Requisiti Funzionali	9
2.2.2 Requisiti sui Dati	10
2.3 Storie Utente e Criteri di Accettazione	11
2.4 Use Case Diagram	15
2.5 Diagrammi dinamici	16
3 Analisi dell'impatto dei requisiti assegnati sul progetto esistente	18
4 Progettazione della soluzione per realizzare i requisiti richiesti	21
4.1 Descrizione delle decisioni di progetto realizzate	21

4.2	Composite Diagrams dell'architettura realizzata	22
4.3	Descrizione delle nuove interfacce realizzate e modifica di quelle precedenti	23
4.3.1	REST API GET \player	23
4.4	Package Diagrams	27
4.5	Diagrammi dinamici dell'architettura	31
4.5.1	Sequence Diagrams	31
4.6	Modifiche ai database	34
4.7	Eventuali issue rilevate e corrette durante lo sviluppo .	35
4.8	Varie	36
4.8.1	Tools utilizzati	36
4.8.2	Frameworks utilizzati	36
4.8.3	API utilizzate	38
4.8.4	Linguaggi utilizzati	38
5	Implementazione e struttura del progetto modificato	40
5.1	Lista dei moduli aggiunti e modificati	40
5.1.1	package T4-G18	40
5.1.2	package T5-G2	41
5.2	Spiegazione modifiche e refactoring dei componenti es- istenti	43
5.2.1	T4	43
5.2.2	T5	44
5.3	Deployment	47
5.4	Foto del risultato ottenuto	48
6	Testing	51

6.1	Testing della GUI	51
6.2	Testing API	56
6.3	Testing TRIGGER	59
7	Sviluppi Futuri	65
7.1	Tipi di classifiche	65
7.2	Migliorare le performance	67
7.2.1	Problema riscontrato	67
7.2.2	Soluzioni proposte	67
7.3	Guida ad implmentazioni future	68

Chapter 1

Punto di partenza e Obiettivi del Progetto

Il Task sviluppato e implementato dal nostro team si inserisce all'interno di un'iniziativa più ampia legata al programma ERASMUS, con una durata triennale, denominata ENACTEST (European iNNovative Al-lianCe for TESTing educaTion). Per raggiungere tale obiettivo, è stata adottata la strategia innovativa della gamification, che consiste nell'integrare elementi caratteristici dei giochi in contesti non ludici, come suggerisce il termine stesso. Questo approccio punta a rendere l'apprendimento più coinvolgente ed efficace. Come risultato è stato ideato e sviluppato il gioco interattivo “Man vs Automated Testing Tools Challenges”, in cui gli studenti, indicati come players, competono tra loro utilizzando test progettati manualmente tramite il framework JUnit. Questi test vengono messi a confronto con quelli generati auto-

maticamente da strumenti come Randoop o EvoSuite, capaci di creare test autonomamente. La sfida viene considerata vinta dal partecipante che riesce a soddisfare un determinato obiettivo di copertura, dimostrando così l'efficacia del proprio approccio.

L'applicazione finale consentirà agli studenti di ottenere una classe da testare, identificata come Class Under Test (CUT); scrivere casi di test JUnit per la CUT, utilizzando un template precompilato; compilare ed eseguire i test prodotti; calcolare una misura di copertura utilizzando strumenti come Emma o JaCoCo; confrontare i risultati dei test generati dal Player con quelli creati dai tool automatici, includendo viste di riepilogo personalizzate; mantenere uno storico delle partite giocate da ciascun giocatore, con relativi esiti, per generare statistiche sull'uso dell'applicazione.

1.1 Descrizione versione di partenza

Il pattern architetturale utilizzato nell'applicazione ENACTEST è MVC – Model View Controller su uno stile architetturale di tipo Client – Server. Inoltre, l'architettura si basa su microservizi, scelta effettuata per garantire: agilità nello sviluppo, in quanto i team possono lavorare in piena autonomia, concentrandosi sulla progettazione, sviluppo, testing e distribuzione di un singolo servizio alla volta. Questo approccio riduce la necessità di interfacciarsi o collaborare costantemente con altri team, poiché ogni servizio è completamente indipendente dagli al-

CHAPTER 1. PUNTO DI PARTENZA E OBIETTIVI DEL PROGETTO

tri; scalabilità del sistema poiché i microservizi permettono di scalare in modo specifico i singoli componenti, evitando di dover intervenire sull'intera applicazione. Questo consente una gestione più efficiente delle risorse e un miglioramento delle prestazioni complessive; semplificazione del rilascio e della manutenzione dal momento che, grazie alla loro indipendenza, i microservizi possono essere rilasciati e aggiornati singolarmente, riducendo i tempi e i rischi legati alle modifiche. Le dimensioni ridotte dei servizi facilitano inoltre la loro gestione; infine, grazie alla suddivisione in microservizi, ci sono numerosi vantaggi in termini di sperimentazione tecnologica perché viene garantita la possibilità di testare e integrare facilmente nuove tecnologie all'interno del sistema senza compromettere il funzionamento complessivo.

La versione del progetto da cui siamo partiti è A13. Di cui seguito si riportano dei diagrammi al fine di mostrare in modo rapido ed intuitivo lo stato iniziale dell'applicazione su cui andremo a lavorare.

CHAPTER 1. PUNTO DI PARTENZA E OBIETTIVI DEL PROGETTO

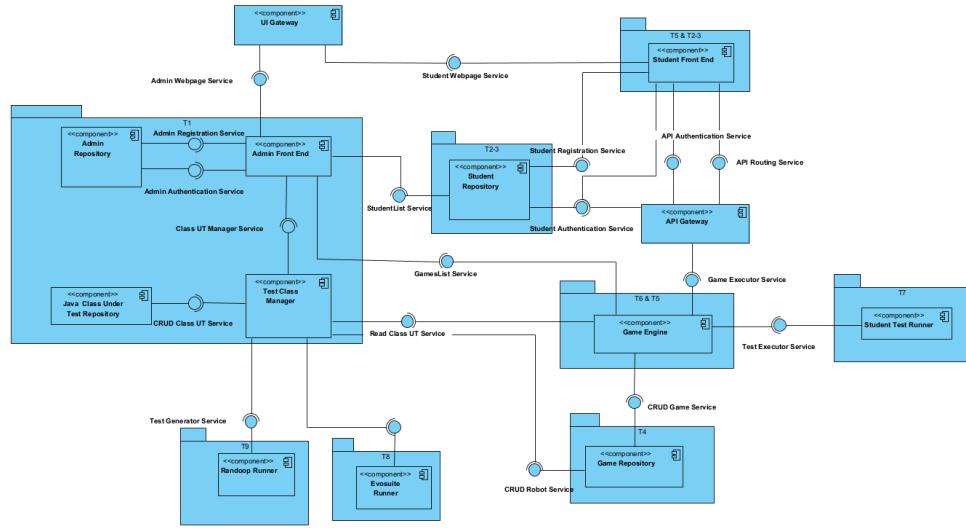


Figure 1.1: Composite Diagram di partenza

Come possiamo notare, troviamo 9 container al cui interno sono rappresentati i principali componenti necessari al funzionamento dell'applicazione.

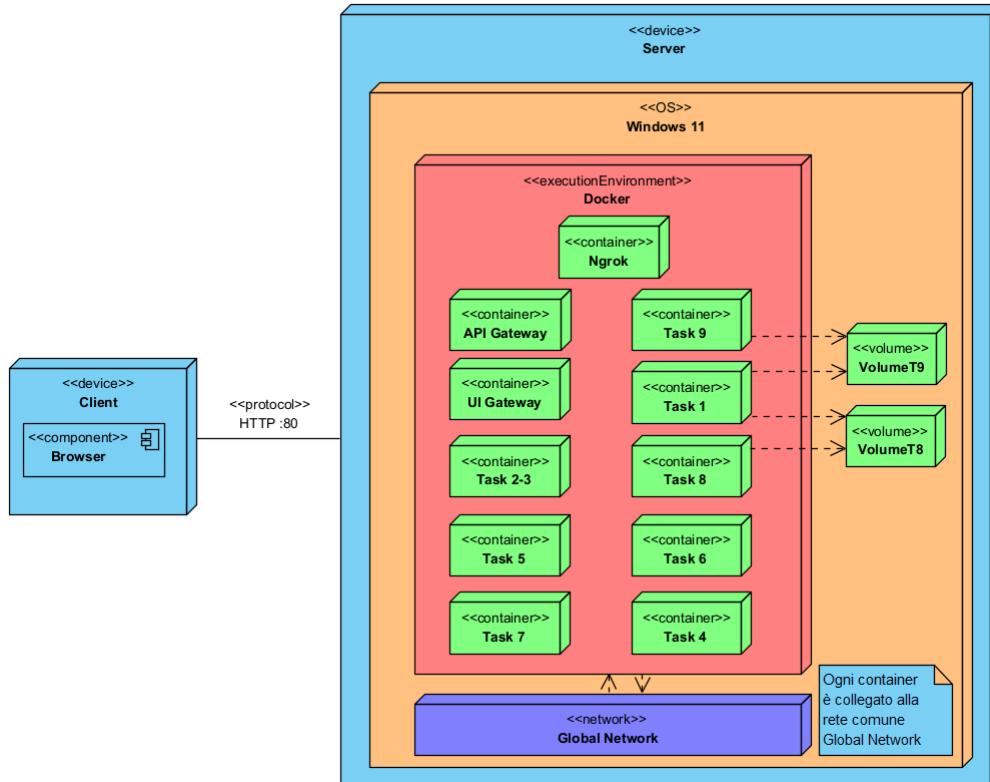


Figure 1.2: Diagramma di deploy di partenza

Il deployment diagram riportato contiene i principali componenti che rappresentano i diversi container necessari all'esecuzione dell'applicazione, ai fini del nostro task non è necessario andare a toccare tali componenti.

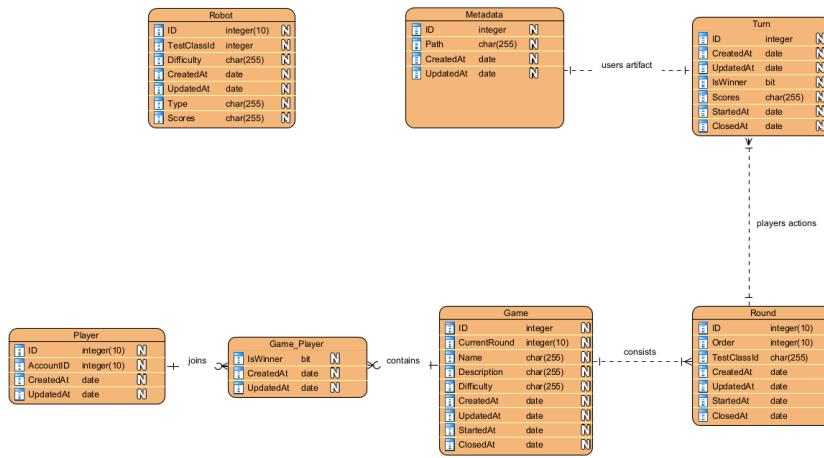


Figure 1.3: Modello ER di partenza del database in T4

Questa è la condizione iniziale del database che contiene parte delle informazioni che sono state utilizzate per la realizzazione delle classifiche.

1.2 Nuovi requisiti assegnati

Il Task la cui realizzazione è stata assegnata al nostro team è lo sviluppo e l'implementazione di una Classifica dei giocatori che utilizzano l'applicazione. La classifica deve rispettare i seguenti requisiti:

Task da svolgere- R2 (Classifiche)

Task	Descrizione	Team
Task R2	Gestire la creazione e visualizzazione di Classifiche dei vari giochi, verosimilmente da visualizzare attraverso la homepage Esempi Di classifiche: classifica delle partite giocate, partite vinte, classi testate.....	
Impatto Stimato	T5	

Figure 1.4: Task #R2 - Classifiche

1.3 Processo di sviluppo seguito

Per portare a termine il Task, il nostro Team di Sviluppo ha scelto di adottare una metodologia di lavoro agile basata sul framework SCRUM, progettato per gestire in maniera iterativa e incrementale il ciclo di sviluppo del software. Questo approccio si esprime al meglio attraverso gli Sprint, che rappresentano le unità fondamentali di lavoro in SCRUM e che solitamente hanno una durata compresa tra una e quattro settimane.

Ogni Sprint inizia con una riunione di pianificazione, durante la quale vengono definiti gli obiettivi e stimati i tempi necessari per raggiungerli. Al termine di ogni Sprint, il Team è tenuto a consegnare una versione del prodotto che sia funzionale e contenga i progressi stabiliti durante la fase di pianificazione. Questo metodo consente di lavorare in modo incrementale, con obiettivi chiari e verificabili per ogni fase dello sviluppo.

Chapter 2

Analisi dei Requisiti

A partire dalla specifica assegnata si procede individuando tutti i requisiti necessari, per la modifica richiesta.

2.1 Glossario

Termine	Definizione	Sinonimi
Player	Un utente che si appresta a giocare tramite una delle modalità proposte, che deve essere registrato e autenticato	Giocatore
Games	Partita singola avviata da un giocatore contro uno dei robot disponibili	Partita

2.2 Revisione dei Requisiti

1. Il giocatore deve poter accedere a una pagina che mostri la classifica globale dei giocatori.
2. Il giocatore deve poter ordinare la classifica in base al punteggio totale accumulato dai giocatori.
3. Il giocatore deve poter ordinare la classifica in base al numero totale di partite vinte dai giocatori.
4. Il giocatore deve poter vedere la propria posizione in classifica.
5. Il sistema deve calcolare la posizione del giocatore autenticato sulla base dei dati correnti.
6. Il sistema deve generare la classifica ordinando i giocatori in base al punteggio totale accumulato.
7. Il sistema deve generare la classifica ordinando i giocatori in base al numero di partite vinte, se richiesto dal giocatore.
8. All'apertura della classifica, il sistema deve mostrare solo i primi 10 giocatori.
9. Il giocatore deve poter espandere la visualizzazione in incrementi di 10 giocatori per volta.

10. Ogni riga della classifica deve mostrare posizione, nome e cognome, punteggio o numero delle partite vinte del giocatore.
11. Il sistema non deve inserire in classifica i giocatori con un punteggio pari a 0.
12. Il sistema non deve inserire in classifica i giocatori con 0 partite vinte.

Stilata la lista di requisiti, possiamo dividerli in due categorie: Requisiti funzionali e Requisiti sui dati.

2.2.1 Requisiti Funzionali

1. Il giocatore deve poter accedere a una pagina che mostri la classifica globale dei giocatori.
2. Il giocatore deve poter ordinare la classifica in base al punteggio totale accumulato dai giocatori.
3. Il giocatore deve poter ordinare la classifica in base al numero totale di partite vinte dai giocatori.
4. Il giocatore deve poter vedere la propria posizione in classifica.
5. Il sistema deve calcolare la posizione del giocatore autenticato sulla base dei dati correnti.

6. Il sistema deve generare la classifica ordinando i giocatori in base al punteggio totale accumulato.
7. Il sistema deve generare la classifica ordinando i giocatori in base al numero di partite vinte, se richiesto dal giocatore.
8. All'apertura della classifica, il sistema deve mostrare solo i primi 10 giocatori.
9. Il giocatore deve poter espandere la visualizzazione in incrementi di 10 giocatori per volta.

2.2.2 Requisiti sui Dati

1. Ogni riga della classifica deve mostrare username, punteggio e numero delle partite giocate del giocatore.
2. Il sistema non deve inserire in classifica i giocatori con un punteggio pari a 0.
3. Il sistema non deve inserire in classifica i giocatori con 0 partite giocate.

Per validare la lista di requisiti individuati, al fine di renderci più chiaro possibile le finalità e le modalità dello sviluppo delle funzionalità da integrare all'applicativo, si procede con la scrittura delle storie utente e i rispettivi criteri di accettazione.

2.3 Storie Utente e Criteri di Accettazione

Le User Stories sono uno strumento utilizzato nello sviluppo software agile per descrivere le funzionalità o i requisiti di un sistema dal punto di vista dell'utente finale. Sono brevi descrizioni che catturano chi utilizzerà una determinata funzionalità, cosa deve poter fare e perché è importante.

I criteri di accettazione sono un insieme di condizioni che definiscono quando una funzionalità o una User Story è considerata completa e soddisfacente. Rappresentano gli standard che il prodotto deve rispettare affinché il team di sviluppo e gli stakeholder siano d'accordo che il lavoro sia stato svolto correttamente.

Riportiamo di seguito le storie utente e i criteri di accettazione di ognuna di essa:



Figure 2.1: Storia Utente #1

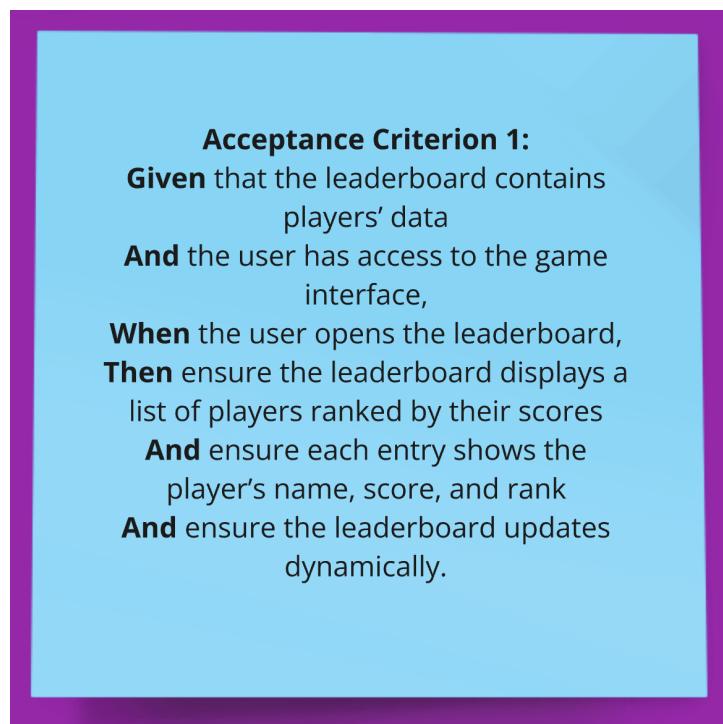


Figure 2.2: Criterio di Accetazione #1



Figure 2.3: Storia Utente #2

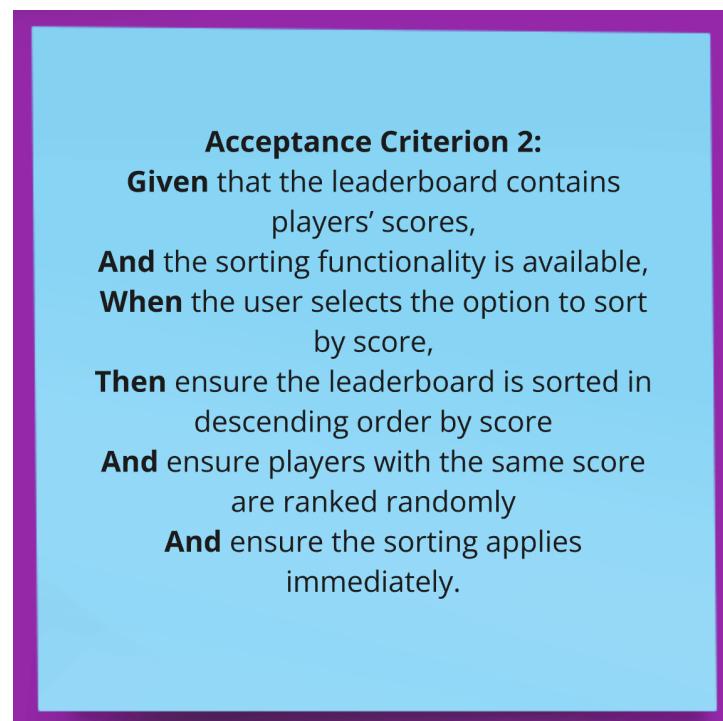


Figure 2.4: Criterio di Accettazione #2

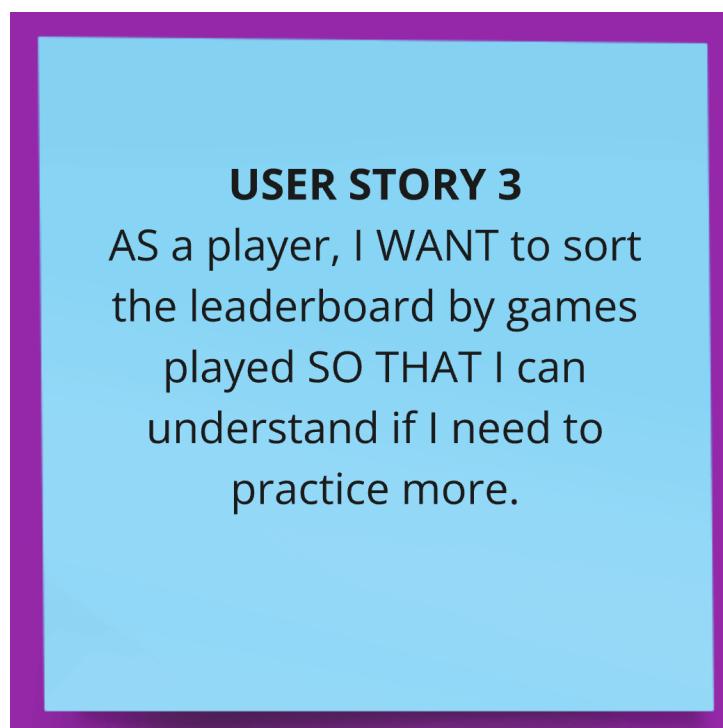


Figure 2.5: Storia Utente #3

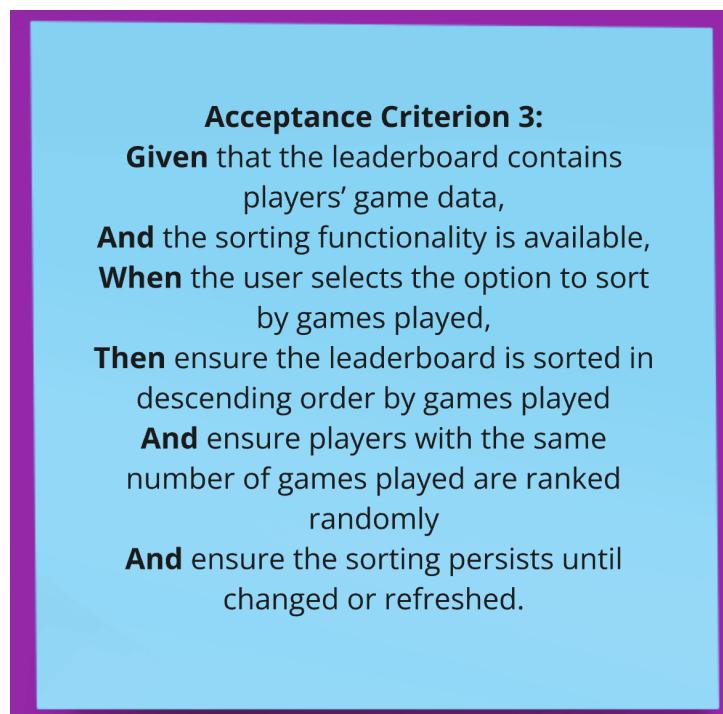


Figure 2.6: Criterio di Accettazione #3

2.4 Use Case Diagram

In questa sezione riportiamo il diagramma dei casi d'uso sviluppato. Possiamo notare come quest'ultimo consta di un unico caso d'uso "Visualizza Classifica dei Giocatori" che si specializza in due casi d'uso: "Visualizza Classifica Ordinata per Punteggio" e "Visualizza Classifica Ordinata per Partite Giocate"; questi ultimi due rappresentano le due diverse modalità con cui sarà possibile visualizzare la classifica dei players all'interno della pagina di interesse. Ricordiamo che la modalità di visualizzazione ordinata per punteggio è la modalità standard con la quale viene visualizzata la classifica.

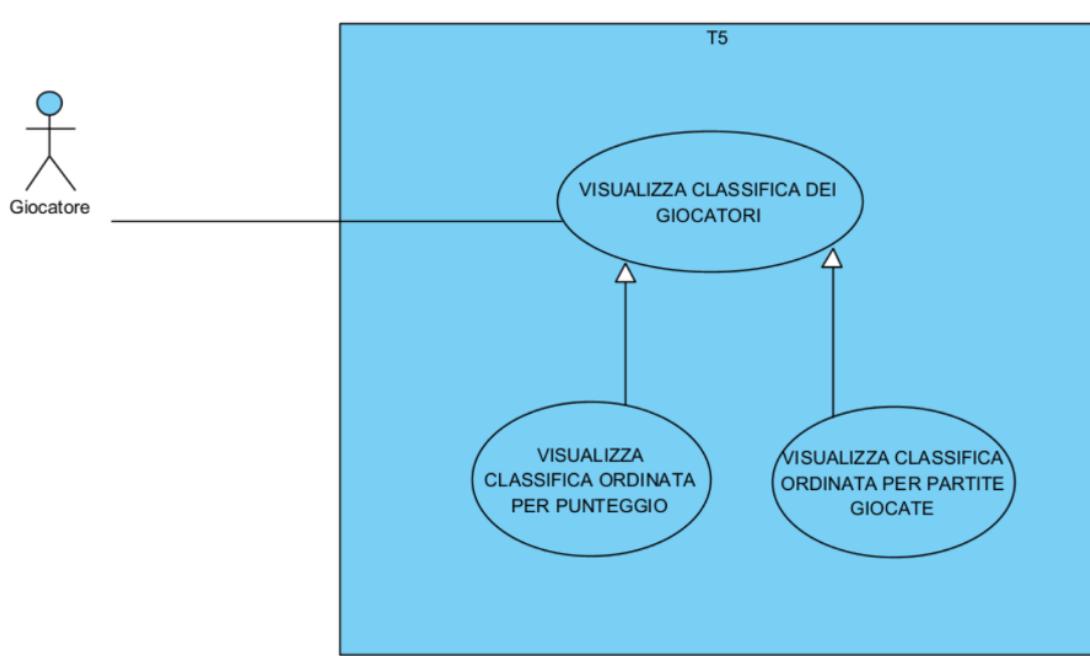


Figure 2.7: Diagramma dei Casi d'Uso

2.5 Diagrammi dinamici

Introduciamo in questa sezione un primo sequence diagram di analisi che analizza innanzitutto la struttura del codice di partenza; in effetti è risultato necessario conoscere quale fossero le chiamate necessarie a realizzare il pattern architetturale MVC. Dall'analisi effettuata si è osservato l'insieme delle chiamate effettuate dal controller per ottenere il model a partire dai dati conservati nei diversi database contenuti in T4 e T23 che viene poi passato alla view (pagina HTML gestita tramite JavaScript) per la visualizzazione del model. In questo primo approccio di progettazione l'idea è stata quella di fare sì che la view ottenessse il model costituito da 2 diverse liste, la lista dei nomi e cognomi dei player e la lista dei punteggi dei player, il problema risultava essere quello di dover inserire la business logic nella view, andando così a violare il pattern architetturale scelto.

CHAPTER 2. ANALISI DEI REQUISITI

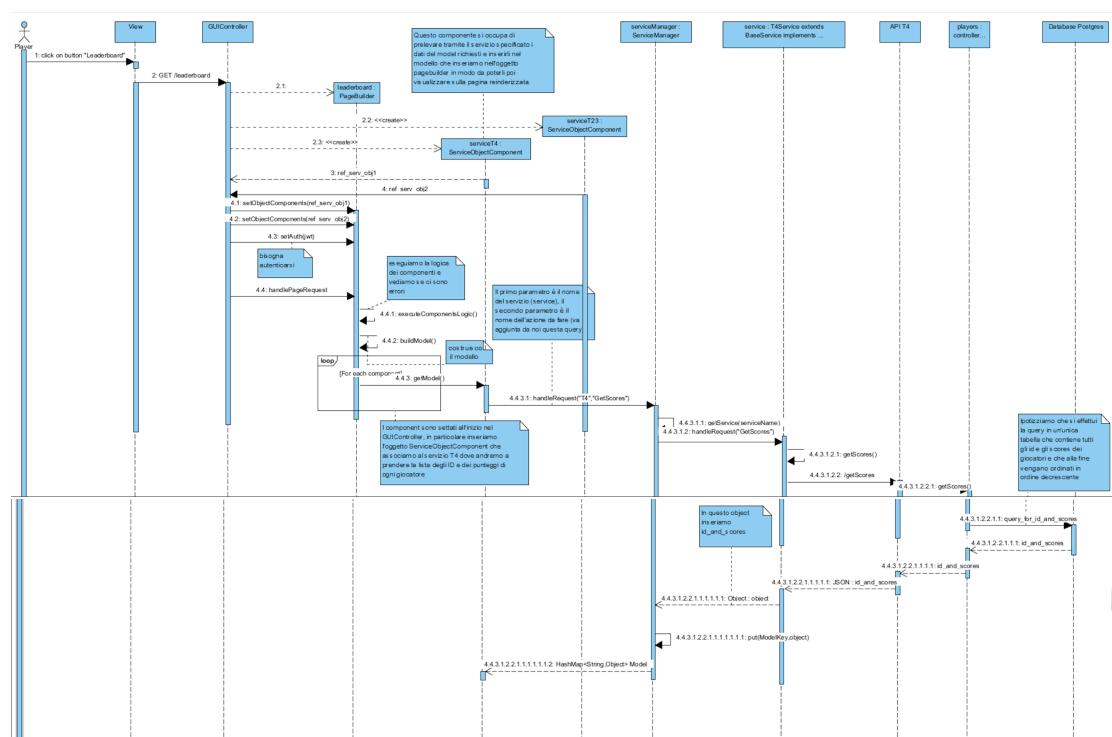


Figure 2.8: Sequence Diagram di Analisi

Chapter 3

Analisi dell'impatto dei requisiti assegnati sul progetto esistente

Abbiamo successivamente studiato e reso noto l'impatto del Task sui diversi componenti del sistema: l'implementazione delle funzionalità chiave del nostro Task impatta su tre componenti: **T23**, **T4**, e **T5** che procederemo a documentare di seguito:

- Iniziamo dal **T23**, container in cui è presente il servizio di autenticazione e registrazione dei players. Il servizio offre ai giocatori la possibilità di creare un account, inserendo informazioni quali nome, cognome, e-mail, password e il proprio percorso di studi. Una volta registrati, possono autenticarsi inserendo le proprie

credenziali. Per i giocatori già registrati, il servizio consente anche di impostare una nuova password nel caso in cui quella precedente fosse stata dimenticata. Inoltre, i giocatori autenticati hanno accesso all’area riservata per selezionare i parametri di gioco e possono effettuare il logout dal sistema quando necessario. Il servizio interagisce con un database MySQL utilizzato per gestire le informazioni relative ai giocatori dalla tabella “Students”, in particolare il Nome e il Cognome di ogni giocatore.

- Passando al **T4**, si tratta del servizio di Repository dei dati di gioco. Questo modulo interagisce con un database PostgreSQL, che si occupa di archiviare in modo strutturato i dati relativi alle partite. Il servizio offre le seguenti funzionalità al motore di gioco: consente di creare una nuova partita, memorizzando una serie di informazioni su di essa; permette di aggiornare la partita, recuperando la data e l’ora di inizio e fine; offre la possibilità di eliminare una partita esistente; genera contestualmente la partita appena creata, insieme ai round che la compongono; consente di aggiornare i vari round della partita; permette di creare, all’interno di ciascun round, i diversi turni associati ai partecipanti della partita; offre la possibilità di aggiornare tali turni; consente di recuperare, durante ciascun round, i punteggi ottenuti dai robot in base alla classe di test in corso.
- Infine, arriviamo al componente **T5**, qui viene gestito il front-end

CHAPTER 3. ANALISI DELL'IMPATTO DEI REQUISITI ASSEGNNATI SUL PROGETTO ESISTENTE

di avvio partita. Il servizio permette correttamente ai giocatori di accedere all'area riservata di selezione dei parametri di gioco, dove possono visualizzare le classi e i robot disponibili precedentemente caricati dagli amministratori. Consente inoltre loro di avviare una partita accedendo all'arena di gioco vera e propria. Il servizio offre ai giocatori due funzionalità chiave: permette loro di configurare le impostazioni del gioco e di accedere all'area di gioco, consentendo così di personalizzare la loro esperienza e di prendere pienamente parte al gioco.

Chapter 4

Progettazione della soluzione per realizzare i requisiti richiesti

4.1 Descrizione delle decisioni di progetto realizzate

Per aiutarci nella progettazione abbiamo scelto di fare uso di component diagrams e package diagrams, in modo da avere un'idea più chiara dei moduli del sistema da modificare mentre per rendere più agevole l'implementazione dei requisiti abbiamo sviluppato i sequence diagrams che danno una vista dinamica di ciò che accade nel sistema dal punto di vista comunicazionale tra i diversi moduli. In generale

l'idea principale è stata quella di ricalcare la struttura già esistente del progetto, quindi rimanere fedeli all'architettura MVC implementata, per rispettare questo vincolo è stato necessario aggiungere un altro servizio che si occupasse dell'elaborazione del model contenente i dati da passare alla view per la visualizzazione corretta della classifica.

4.2 Composite Diagrams dell'architettura realizzata

Si presenta di seguito il Composite Diagram dell'architettura realizzata, avendo evidenziato i componenti modificati durante l'implementazione del nostro task:

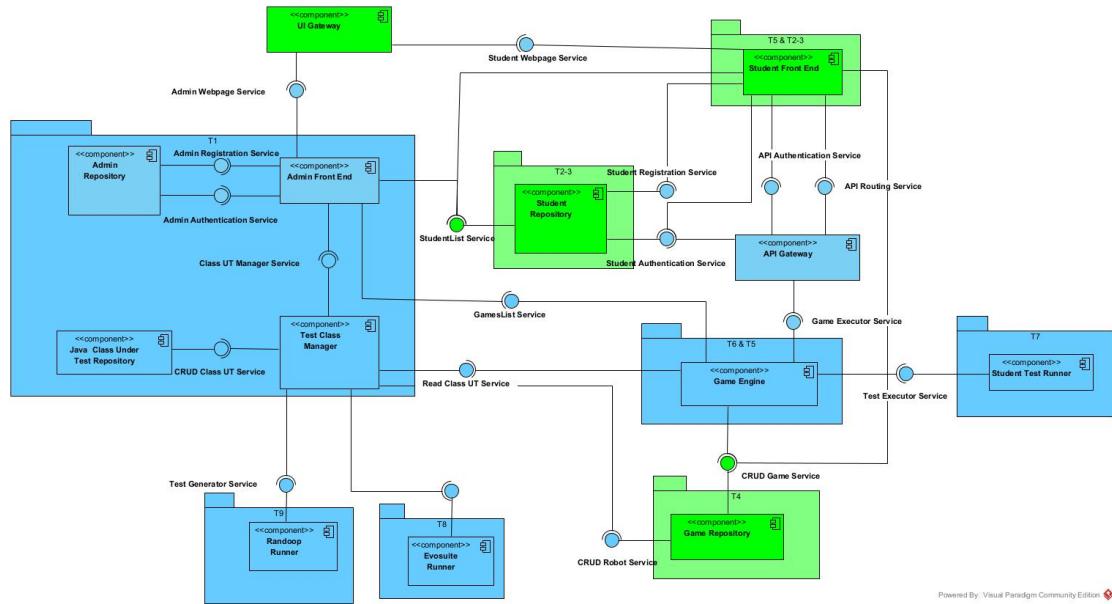


Figure 4.1: Composite Diagram

In particolare, per quanto riguarda il Game Repository, poiché contiene le informazioni relative ai punteggi e alle partite vinte, è stato nec-

essario modificarne il modello ER del database contenuto e l’interfaccia utilizzata dallo Student Front End contenuto nel package T5. Questa modifica ha permesso di fornire un’API che consente di ottenere la lista di tutti i punteggi e le partite vinte dei diversi giocatori.

Il package T23, invece, non ha subito modifiche, ma è stato utilizzato per prelevare i nomi e i cognomi dei giocatori contenuti nel database presente in esso. Inoltre, l’interfaccia fornita da T23 è stata resa disponibile anche al package T5. Altre modifiche sono state apportate all’UI Gateway, uno snodo cruciale per l’interfacciamento con l’utente.

4.3 Descrizione delle nuove interfacce realizzate e modifica di quelle precedenti

4.3.1 REST API GET \player

Per documentare l’api aggiunta è stato utilizzato Swagger. La documentazione dell’API che rispetta i requisiti OpenApi:

Tra le informazioni è riportato il nome dell’API ed altre informazioni descrittive. Tra queste anche la lista dei server, nel nostro caso l’ambiente di sviluppo locale. In un contesto reale, sarebbe utile aggiungere un server di produzione.

Sono poi riportate le risposte con alcuni esempi. Gli esempi forniti

```
1  openapi: 3.0.1
2  info:
3      title: Player API
4      description: API per gestire i dati dei giocatori.
5      version: "1.0.0"
6  servers:
7      - url: http://localhost:3000
8          description: Server locale per lo sviluppo
9  paths:
10     /player:
11         get:
12             summary: Ottieni i dettagli dei giocatori
13             description: Restituisce lista giocatori.
14             responses:
15                 '200':
16                     description: Lista dei giocatori
17                     content:
18                         application/json:
19                             schema:
20                                 type: array
21                                 items:
22                                     $ref: '#/components/schemas/Player'
23             examples:
24                 example1:
25                     summary: Lista di giocatori
26                     value:
27                         - id: 1
28                             accountID: "user123"
29                             createdAt: "2023-12-01T12:34:56Z"
30                             updatedAt: "2023-12-02T14:23:11Z"
31                             points: 100.5
32                             gamesWon: 10
33                         - id: 2
34                             accountID: "user456"
35                             createdAt: "2023-11-28T08:15:30Z"
36                             updatedAt: "2023-12-03T16:00:45Z"
37                             points: 85.0
38                             gamesWon: 8
39             '500':
40                 description: Errore interno del server
```

Figure 4.2: Doc API

aggiungono valore alla documentazione, illustrando dati realistici che gli sviluppatori possono utilizzare come riferimento. A supporto di questa descrizione c'è il seguente schema:

```
41 - components:
42 -   schemas:
43 -     Player:
44 -       type: object
45 -       properties:
46 -         id:
47 -           type: integer
48 -           format: int64
49 -           description: Identificativo univoco del giocatore
50 -         accountID:
51 -           type: string
52 -           description: Identificativo dell'account associato al giocatore
53 -         createdAt:
54 -           type: string
55 -           format: date-time
56 -           description: Data e ora di creazione del giocatore
57 -         updatedAt:
58 -           type: string
59 -           format: date-time
60 -           description: Data e ora dell'ultimo aggiornamento del giocatore
61 -         points:
62 -           type: number
63 -           format: float
64 -           description: Punteggio totale accumulato dal giocatore
65 -         gamesWon:
66 -           type: integer
67 -           format: int64
68 -           description: Numero di partite vinte dal giocatore
69
```

Figure 4.3: Doc API

Ogni proprietà nello schema è ben descritta. Ciò garantisce che i client comprendano il significato e l'utilizzo di ogni campo. L'utilizzo di '\$ref' per riferirsi allo schema 'Player' è una scelta per il riutilizzo del codice e per migliorare la manutenibilità. Inoltre non sono stati aggiunti endpoint CRUD (Create, Update, Delete) per completare l'implementazione RESTful poiché non necessari.

Questo perchè viene offerta una funzionalità unica e ben definita, ossia rappresenta un endpoint per l'integrazione con altre applicazione. Questa scelta è legata alla restrizione delle funzionalità offerte e per limitare le superfici d'attacco. L'unico funzionalità offerta è il prelievo dei dati sulle partite dei player. Un'altra soluzione per limitare gli attacchi potrebbe essere quella di limitare l'accesso a quest'endpoint soltanto a chi possiede un API key oppure un token JWT, in modo da proteggere le informazioni presenti nel database.

La descrizione di questa API deve essere aggiunta a quelle già presenti in T4 in modo da offrire un quadro completo sulle API offerte. Per una visualizzazione sintetica:

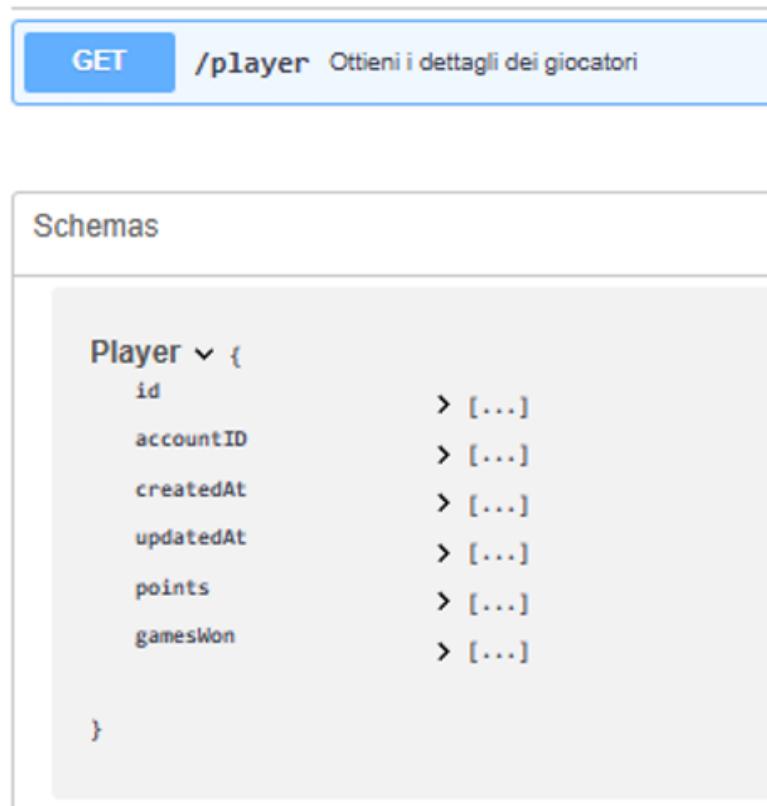


Figure 4.4: API Postman

4.4 Package Diagrams

Nel seguente package diagram abbiamo analizzato e rappresentato i package effettivamente interessati dal nostro intervento: T23, T4 e T5.

Non è stata rappresentata l'intera struttura (con tutti i sottopackage e tutte le classi realmente presenti) di tutti i package, ma solo la parte rilevante per le modifiche e le implementazioni effettuate. In particolare, in giallo sono rappresentate le classi realizzate da zero (from scratch), mentre in verde quelle che hanno subito modifiche.

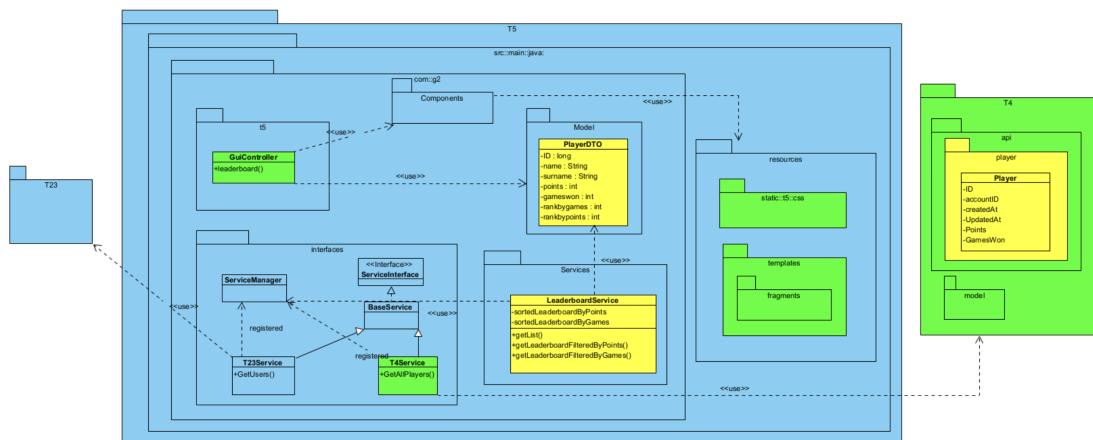


Figure 4.5: Package Diagram

Il package T23 non ha subito alcuna modifica, ma è stata utilizzata l'API `/students_list` fornita per recuperare i nomi e i cognomi di tutti i giocatori dal database MySQL in esso presente. Questa API è invocata tramite la classe ServiceManager, che registra il servizio T23Service, il quale consente di effettuare l'operazione richiesta. Il package T5 è quello maggiormente interessato dalle modifiche: nel package interfaces è stato aggiunto, nella classe T4Service,

un metodo per effettuare richieste GET dei dati di gioco contenuti nel database PostgreSQL di T4. Inoltre, nel package t5 è stato implementato il metodo leaderboard nella classe GuiController, associato alla rotta /leaderboard, che consente al GuiController di richiedere i dati necessari al model e passarli alla view. Poiché i dati provengono da database diversi è stato necessario realizzare un nuovo servizio, LeaderboardService, che permette di gestire e aggregare due classifiche ordinate (per punti e per partite vinte), utilizzando tre metodi per caricare e pre-elaborare i dati. Questi vengono poi passati dal controller al PageBuilder per costruire correttamente la pagina HTML leaderboard.html inserita nel package templates. Collegandoci alle modifiche visive, notiamo che è stato necessario anche aggiornare il package fragments affinchè si potesse visualizzare un opportuno pulsante per accedere alla sezione classifiche. Ritornando alla parte di backend, osserviamo che è stata implementata la classe PlayerDTO per trasferire solo le informazioni dei giocatori essenziali per la classifica. Inoltre, nel package resources, sono stati aggiunti la pagina HTML leaderboard.html, file CSS e JavaScript per migliorare l'usabilità e l'attrattività della pagina. Il package components, pur non avendo subito modifiche, è stato utilizzato per supportare il metodo GuiController. Infine, nel package T4 è stato modificato il file main.go, con l'aggiunta del mapping per la rotta /player, che invoca il controller del package player (da noi implementato). Questo controller richiama

CHAPTER 4. PROGETTAZIONE DELLA SOLUZIONE PER REALIZZARE I REQUISITI RICHIESTI

il servizio che preleva i dati dei giocatori dal database PostgreSQL, gestiti attraverso un nuovo model appositamente creato per includere i campi necessari. Il dettaglio delle modifiche del package T4 sono riportate nel package diagram di seguito riportato:

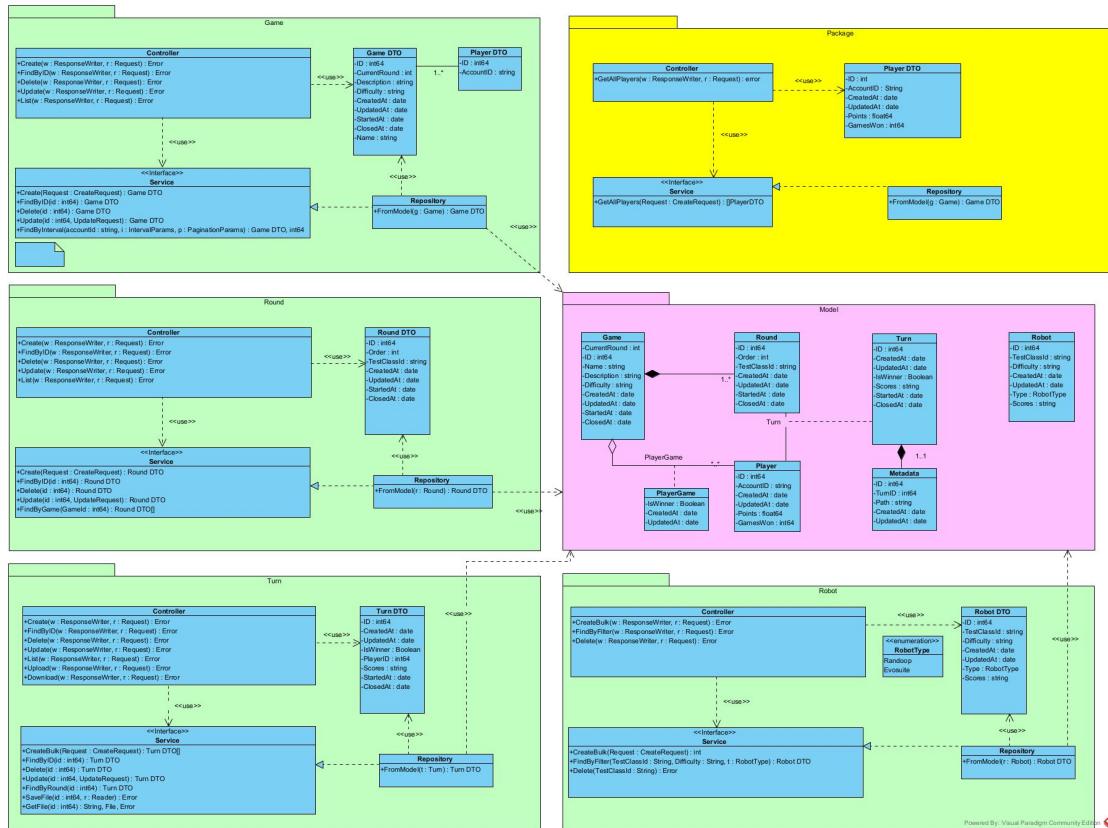


Figure 4.6: Package Diagram T4

In esso ritroviamo la struttura originariamente pensata per T4, il nostro intervento ha previsto l'aggiunto del package riportato in giallo e quindi la modifica del model per quanto riguarda PlayerDTO.

CHAPTER 4. PROGETTAZIONE DELLA SOLUZIONE PER REALIZZARE I REQUISITI RICHIESTI

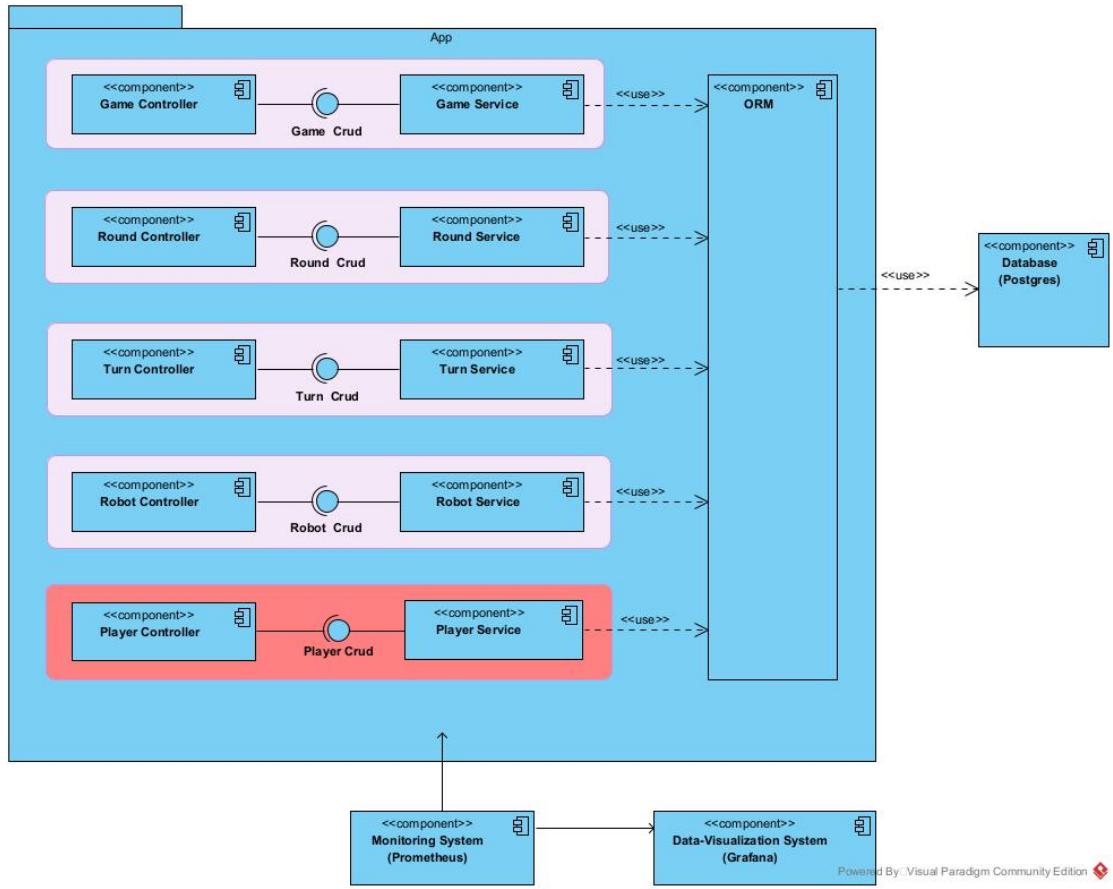


Figure 4.7: Architettura Game Repository

Focalizzandoci ancora sul package T4, notiamo che è stata completamente rispettata l'architettura per la gestione della repository già presente nel sistema. Questa prevede di non comunicare direttamente con il database, ma usare uno strato software (GORM) per evitare di realizzare query in SQL. I componenti da noi aggiunti sono indicati in ROSSO.

4.5 Diagrammi dinamici dell'architettura

4.5.1 Sequence Diagrams

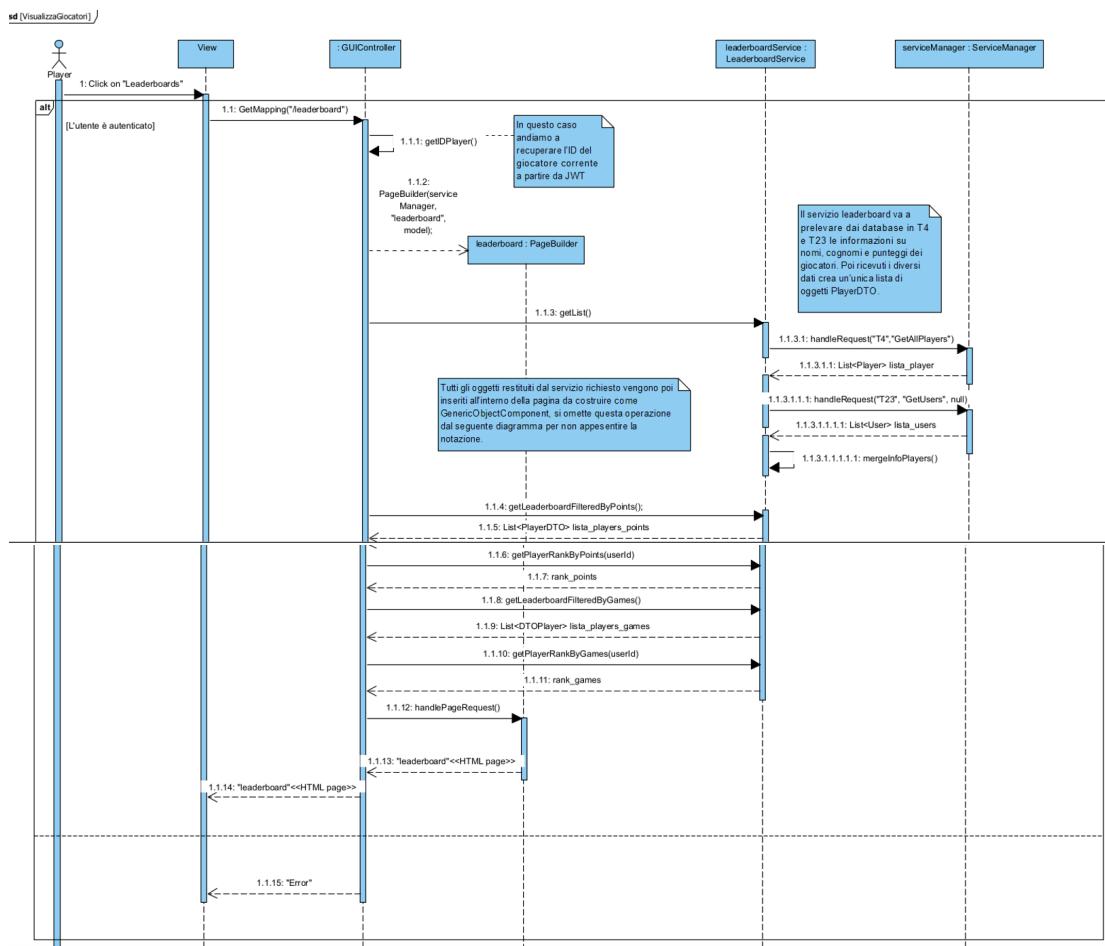


Figure 4.8: Sequence Diagram "VisualizzaGiocatori"

Sfruttando l'architettura a microservizi scelta per l'applicazione, l'idea è stata quella di aggiungere un servizio denominato Leaderboard-Service, in questo modo il GuiController fa riferimento al Leaderboard-Service per ottenere i dati necessari sui giocatori, ovvero il nome e cognome (conservati nel database MySQL T23) e il punteggio e il numero di partite vinte dai giocatori (conservati nel database PostgreSQL in

T4). In questo modo il servizio una volta recuperati i dati tramite il ServiceManager, si occupa di unire le informazioni (viene simboleggiato nel sequence dalla chiamata mergeInfoPlayers()) generando un singolo oggetto PlayerDTO per ogni giocatore, che contiene le informazioni su nome, cognome, punti e partite vinte per ogni giocatore, l'unione dei dati è fatta sulla base dell'id del giocatore, e poi unendo questi oggetti in un'unica lista di DTOPlayer. A questo punto il controller si limita a richiedere le diverse classifiche ordinate secondo i due criteri scelti e richiede la posizione del giocatore attualmente loggato nell'applicazione, questi dati vengono inseriti nel model come GenericObjectComponent e inviati alla pagina HTML leaderboard.html, questa si occupa di mostrare la classifica e permette di ordinarla secondo diversi criteri. Si omettono in questo sequence le specifiche chiamate per l'acquisizione dei dati riportate nel sequence precedente.

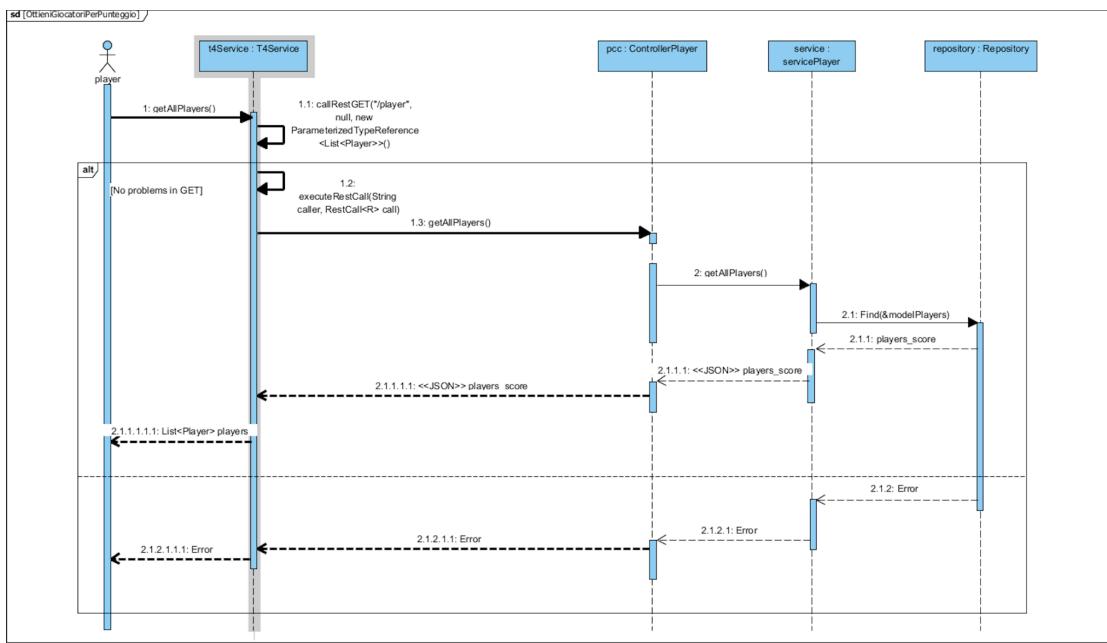


Figure 4.9: Sequence Diagram "OttieniGiocatoriPerPunteggio"

In questo sequence abbiamo progettato le modifiche da apportare al componente T4 affinchè si potessero prelevare i punteggi di tutti i giocatori che abbiano giocato quanto meno una partita. L'interazione parte da T5 dove abbiamo aggiunto T4Service che permette di invocare l'API fornita da T4, da noi realizzata, mappata sulla rottta /player. Il pattern utilizzato in questo caso è il pattern service->controller->model, si invia la richiesta al controller che a sua volta invoca il service per ottenere i dati conservati nel database secondo il model definito, per quanto riguarda l'interazione con il DB in realtà essa non è diretta ma avviene tramite GORM che permette di interagire programmaticamente, senza dover realizzare query in SQL.

4.6 Modifiche ai database

Infine per terminare l'analisi delle modifiche apportate si riporta anche il diagramma ER iniziale con le modifiche apportate (indicate in giallo). Notiamo che la modifica si limita ad aggiungere due attributi all'entità player.

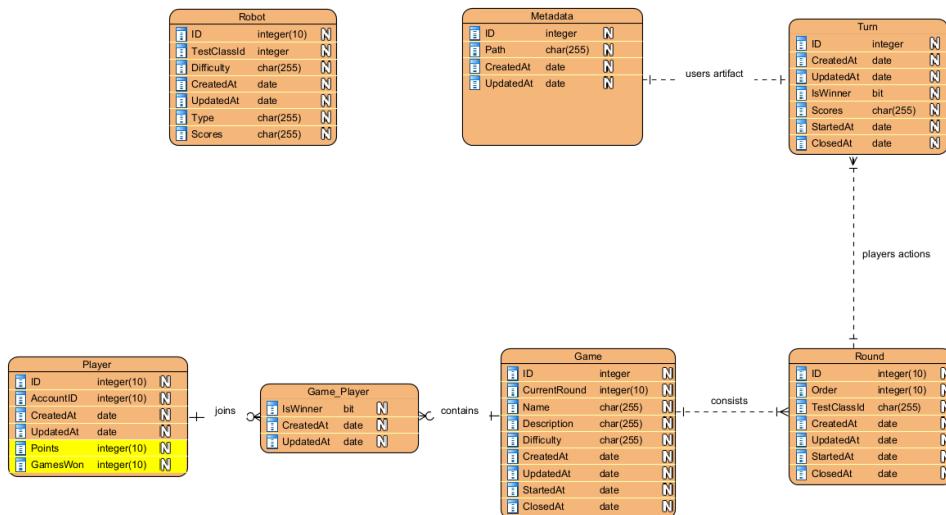


Figure 4.10: Database a valle delle modifiche effettuate

Un'ulteriore modifica riguardo il database consiste nell'aver inserito opportuni trigger per mantenere intatta la consistenza dei dati mantenuti, in particolare relativamente al corretto aggiornamento di punteggio e partite vinte ad ogni partita giocata da un giocatore.

4.7 Eventuali issue rilevate e corrette durante lo sviluppo

Si riporta in questa sezione le problematiche che abbiamo rilevato e corretto durante l'implementazione del task:

- Al termine di una partita, le informazioni circa quest'ultima non venivano salvate nel database. Per aggirare il problema e testare solo le nostra funzionalità, abbiamo fatto l'injection direttamente sul database. Nel frattempo dei nostri colleghi hanno scoperto la natura del bug e l'hanno risolto; si suppone quindi che nella versione finale del progetto il funzionamento sia corretto.
- Difficoltà a lavorare con bootstrap, di cui si fa uso in gran parte delle pagine html del progetto. Per poter realizzare un interfaccia libera dai vincoli del template si è deciso di scrivere da zero la pagina html senza farne uso.
- Dal momento che il calcolo del punteggio delle partite non è ancora stato implementato, questo valore risulta essere sempre nullo.

4.8 Varie

4.8.1 Tools utilizzati

Per la fase di progettazione, abbiamo fatto uso di strumenti collaborativi e visivi come **Trello**, per la gestione delle attività, **Visual Paradigm**, per la modellazione UML e la progettazione dei diagrammi, e **Miro**, per brainstorming e la creazione di mappe concettuali.

Durante la fase di implementazione, ci siamo avvalsi di **Visual Studio Code** come ambiente di sviluppo, **Docker** per la containerizzazione delle applicazioni e **ChatGPT** come supporto per l'ottimizzazione del codice e la risoluzione di problemi tecnici.

4.8.2 Frameworks utilizzati

Il framework in cui si inseriscono le nostre modifiche è “**Spring**”, un framework open-source che fornisce un supporto completo per la costruzione di applicazioni aziendali e web di alta qualità, modulari e facilmente scalabili. Spring è stato progettato per semplificare lo sviluppo di applicazioni Java riducendo la complessità attraverso tecniche come Dependency Injection (DI) e la programmazione orientata agli aspetti (AOP).

Inversion of Control (IoC) è un principio che affida al framework la gestione del controllo dell'applicazione. Questo significa che il framework si occupa di creare gli oggetti, inizializzarli e richiamare

i loro metodi, liberando così gli sviluppatori dalla necessità di gestire manualmente queste operazioni. In questo modo, gli sviluppatori possono concentrarsi sulla logica principale dell'applicazione, lasciando al framework gli aspetti tecnici legati alla gestione degli oggetti.

Dependency Injection (DI), invece, è un metodo specifico con cui Spring implementa il principio di IoC. Con la DI, le dipendenze di un oggetto (cioè gli altri oggetti di cui ha bisogno per funzionare) non vengono create direttamente da esso, ma vengono fornite dall'esterno, ad esempio attraverso un costruttore o un metodo setter. Questo approccio facilita la modularità e rende più semplice gestire e configurare le dipendenze all'interno dell'applicazione, migliorandone la manutenibilità.

In Spring, gli oggetti creati e gestiti dal framework sono chiamati bean e vengono definiti all'interno del progetto tramite configurazioni. L'IoC Container di Spring ha il compito di individuare i bean dichiarati e di occuparsi automaticamente dell'iniezione delle dipendenze necessarie per il loro corretto funzionamento. Questo approccio garantisce un elevato livello di flessibilità e semplifica la configurazione delle dipendenze, rendendo più efficiente la gestione degli oggetti e delle loro relazioni all'interno dell'applicazione.

4.8.3 API utilizzate

Le API utilizzate durante l'implementazione del Task sono diverse. Prima di tutto le **API REST**, un insieme di regole e convenzioni per progettare interfacce di programmazione che consentono a diverse applicazioni software di comunicare tra loro attraverso il protocollo HTTP. Queste sono un punto chiave del progetto in quanto è strutturato sul pattern Client-Server. REST impone, inoltre, un'interfaccia uniforme che semplifica l'interazione tra i sistemi. Ciò include l'utilizzo di risorse identificate da URI (Uniform Resource Identifier) e metodi HTTP standard. Una risorsa può essere rappresentata in diversi formati (ad esempio, JSON, XML, HTML). In particolare, i formati utilizzati sono **JSON** e **HTML**. Inoltre, abbiamo usato le API offerta da **GORM** (Google Object Relational Mapper), una libreria del linguaggio GO, che ci hanno permesso di lavorare con i database relazionali in GO in modo più semplice, mappando le tabelle del database a strutture dati GO (struct). Abbiamo utilizzato funzioni e metodi forniti da GORM per eseguire le operazioni CRUD.

4.8.4 Linguaggi utilizzati

Al fine di implementare le nuove funzionalità offerte dall'app abbiamo usato diversi **linguaggi di programmazione** quali Java e GO, dei **linguaggi di Markup** quali HTML e CSS e infine JavaScript come **linguaggio di scripting** orientato agli oggetti. In particolare, abbi-

CHAPTER 4. PROGETTAZIONE DELLA SOLUZIONE PER REALIZZARE I REQUISITI RICHIESTI

amo utilizzato Java e GO per implementare le funzionalità di back-end, HTML per il front-end, CSS per lo stile delle pagine e JavaScript per implementare la logica interattiva.

Chapter 5

Implementazione e struttura del progetto modificato

5.1 Lista dei moduli aggiunti e modificati

5.1.1 package T4-G18

In T4-G18\api è stato aggiunto il package **player** contenente i seguenti file:

- player.go
- controller.go
- service.go

In T4-G18\model\model.go sono state fatte le seguenti modifiche:

- nella tabella players sono stati aggiunti i campi **Points** per tenere traccia dei punti accumulati dai giocatori durante il game-play e **GamesWon** per tenere traccia del numero di partite vinte da ogni player.
- è stato aggiunto il trigger afterSave della tabella Game per aggiornare i punti accumulati dal player al termine di ogni partita giocata.

In T4-G18\main.go:

- è stata aggiunta la rotta /player per consentire l'accesso ai dati e alle funzionalità legate ai giocatori attraverso l'API.
- è stato aggiunto il controller (endpoint della rotta /player).
- è stato importato il package aggiunto "player".

In T4-G18\api\game\service.go è stata aggiornata la funzione di Update perché quando viene aggiornato un game e isWinner viene modificato anche il campo isWinner della tabella PlayerGame e aggiorna direttamente il numero di partite vinte in Player.

5.1.2 package T5-G2

In T5-G2\t5\src\main\java\com\g2\Interfaces\T4Service è stata aggiunta la chiamata rest GET GetAllPlayers() per ottenere tutti i gio-

CHAPTER 5. IMPLEMENTAZIONE E STRUTTURA DEL PROGETTO MODIFICATO

catori ordinati per punti. Questa chiamata si trova all'interno di una nuova azione “registerAction” che collega le operazioni di back-end con la grafica dell'interfaccia.

In T5-G2\t5\src\main\java\com\g2\Model sono stati aggiunti i seguenti file:

- Player.java
- PlayerDTO.java un dto del giocatore che contiene id, nome, cognome, punti e partite giocate

In T5-G2\t5\src\main\java\com\g2\Service è stato aggiunto il file LeaderboardService.java, un nuovo servizio dedicato alla gestione della logica di business per la generazione delle classifiche. Questo Service viene chiamato nella rotta /leaderboard del GUI Controller per essere mostrato poi nella page della classifica.

In T5-G2\t5\src\main\java\com\g2\t5\GuiController.java sono state fatte le seguenti modifiche:

- è stata riscritta la rotta \leaderboard
- è stato importato Service.LeaderboardService
- è stato aggiunto l'attributo Leaderboard (@Autowired)

In T5-G2\t5\src\main\resources\templates\fragments è stato modificato il file navbar.html per aggiungere il bottone "Classifica".

In T5-G2\t5\src\main\resources\templates è stata riscritta la pagina leaderboard.html

In T5-G2\t5\src\main\resources\static\t5\css è stato aggiunto il file leaderboard.css

In T5-G2\t5\src\main\resources\static\t5\js è stato aggiunto il file leaderboard.js

In T5-G2\t5\src\main\resources\static\t5\images è stata aggiunta foto7.jpeg ovvero l'immagine usata come sfondo nella pagina delle classifiche.

In T5-G2\t5\src\main\resources\lang sono stati modificati i file per la gestione del meccanismo di internazionalizzazione aggiungendo le etichette presenti nella pagina html.

5.2 Spiegazione modifiche e refactoring dei componenti esistenti

I package impattati dalle modifiche come visto sono il T4 e il T5

5.2.1 T4

In questo package troviamo il database dove vengono salvate le informazioni circa ciascuna partita giocata. Per associare a ciascuna partita l'id del giocatore è presente una tabella player che ha come unico campo gli id dei giocatori. Abbiamo modificato quest'ultima aggiun-

gendo i campi relativi al numero di partite vinte e al punteggio totale raggiunto da ciascun giocatore. Per effettuare questa modifica è stato modificato il file `model.go`. Ogni volta che una partita viene salvata nella tabella `game` vogliamo che tali campi si aggiornino automaticamente, pertanto si è pensato di aggiungere un trigger. Per effettuare la modifica alla tabella `player` e aggiungere il trigger è stato modificato il file **`model.go`**. Vogliamo quindi fare una query su questa tabella del database per ottenere le informazioni necessarie a costruire una classifica. A tal fine per lavorare in accordo con il pattern MVC usato anche per la gestione di questo database abbiamo aggiunto il **package player** con le classi, `player.go`, `controller.go` e `service.go`. Nel modulo **`main.go`** abbiamo aggiunto una nuova rotta, "`player`" il cui endpoint è il modulo "controller.go" del package "player". Il "service.go" si occupa di effettuare una query che preleva tutti i giocatori dalla tabella "player" mentre in "player.go" i dati dei giocatori prelevati vengono trasformati in formato JSON. Per far sì che quando un game viene aggiornato, si abbia consistenza anche con i valori di punti e partite vinte della tabella `players` è stato modificato "service.go" nel package `game` aggiornando la funzione `update`.

5.2.2 T5

In T5 abbiamo la necessità di richiamare l'API offerta da T4 per recuperare la lista di tutti i giocatori con le relative informazioni. A tal è

stata aggiunta una chiamata REST "GetAllPlayers()" all'interno del modulo T4Service.

Per gestire gli oggetti player è stato poi necessario aggiungere una classe Player e una classe PlayerDTO all'interno del package Model.

Abbiamo sviluppato un MockUp della leaderboard per capire quali linee guida seguire per il design grafico. Dopodichè abbiamo realizzato la pagina html della leaderboard chiamata "leaderboard.html".

All'interno del package T5 abbiamo aggiunto una chiamata REST "GetAllPlayers()" all'interno del package T4 Service per recuperare tutti i players da T4.

In T4 si hanno le informazioni relative alle partite, ma non le informazioni relative agli utenti che invece sono conservate in un database gestito in T23. L'applicazione si propone come sussidio didattico, per cui piuttosto che visualizzare un id numerico, per identificare il giocatore in classifica, si ritiene necessario visualizzare nome e cognome di ciascuno studente giocatore. Per ottenere tale informazione va fatta una chiamata a T23, questa API era già presente nel progetto di partenza per cui è stata solo utilizzata. A questo punto va costruito l'oggetto da passare alla view, contenente un appropriato mapping delle informazioni provenienti dalle due chiamate. Per non violarre i principi del pattern MVC, tale responsabilità è stata affidata a una classe Service chiamata LeaderboardService che abbiamo aggiunto al package service.

Si potrebbe pensare che avere un identificativo univoco possa risolvere il problema dell’omonimia, non gestito identificando gli studenti giocatori con nome e cognome. Tuttavia, la pagina che si vuole costruire presenta un riquadro apposito per conoscere la propria posizione (utente autenticato).

Si vuole costruire una vista che consenta di vedere una classifica ordinata per punti e una ordinata per partite vinte.

E’ stato poi necessario modificare la rotta leaderboard nel GUIController, già presente nella versione di partenza del progetto, affinchè fossero passati alla pagina html i dati necessari secondo le scelte di progetto da noi effettuate: lista dei giocatori ordinata per punti, lista giocatori ordinata per partite vinte, posizione in classifica dell’utente autenticato per le due modalità di visualizzazione.

Per garantire modularità del codice nella realizzazione della vista sono state divise pagina html, css e javascript.

La pagina leaderboard.html è stata completamente riscritta per migliorare il più possibile l’esperienza utente. E’ stato previsto un toggle per switchare tra le due classifiche visualizzabili. Per mostrare 10 giocatori alla volta in classifica, muoversi attraverso quest’ultima e cercare i giocatori è stata usata la libreria DataTable. Per ricavare la posizione dell’utente autenticato è stato l’id passato dal controller che ricava dal cookie.

E’ stato aggiunto un file leaderboard.css per gestire lo stile della

pagina. Per gestire la dinamicità della pagina, grazie all'uso del toggle, è stato invece aggiunto un file leaderboard.js.

Tutte le etichette nella pagina hanno un comportamento dinamico, sfruttando non solo javascript ma anche il meccanismo di internazionalizzazione offerto da Thymeleaf.

Inoltre, per migliorare al massimo la gamification, e rendere il gioco quanto più attrattivo possibile, si è pensato di introdurre un'animazione sulla pagina che parte quando il giocatore autenticato raggiunge il primo posto in classifica per una delle due modalità previste. Vogliamo festeggiare con gli studenti i loro piccoli traguardi, e per questo abbiamo programmato un'animazione grafica che prevede la simulazione di una cascata di coriandoli sullo schermo.. Per fare ciò è stata usata la libreria canvas-confetti, il cui riferimento è nella pagina html, la cui gestione invece è nella pagina javacsript.

5.3 Deployment

Per rappresentare la disposizione fisica dei componenti hardware e software in un sistema distribuito si rende necessario un diagramma di deploy. Non avendo apportato modifiche in merito si fa riferimento a quello presentato nella versione A13 del progetto, da cui siamo partiti, riportato nel primo capitolo di questa documentazione. Come fatto per le precedenti versioni del progetto è stato usato il meccanismo di "tunneling" di ngrok per esporre il servizio in rete. Ciò consente di

esporre un server che sta girando sulla una macchina locale a Internet, creando un tunnel sicuro che permette l'accesso esterno a una porta specifica sulla nostra macchina personale. In questo modo abbiamo testato il corretto funzionamento dell'applicazione con la nuova funzionalità aggiunta.

Per ottenere il link fornito da ngrok è possibile contattare il responsabile del gruppo.

5.4 Foto del risultato ottenuto



Figure 5.1: Navbar

CHAPTER 5. IMPLEMENTAZIONE E STRUTTURA DEL PROGETTO MODIFICATO

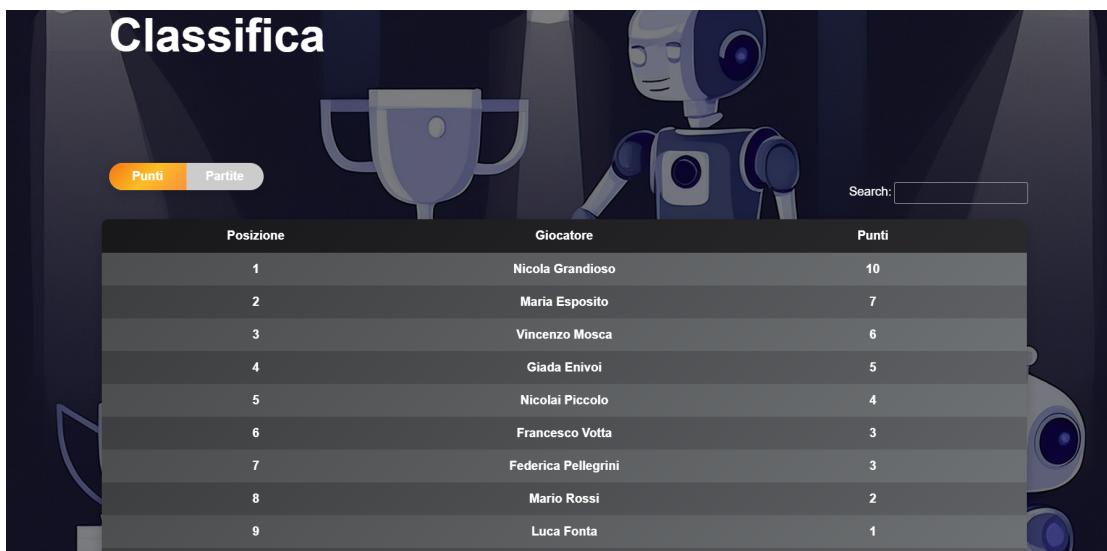


Figure 5.2: Landing page "Classifica"

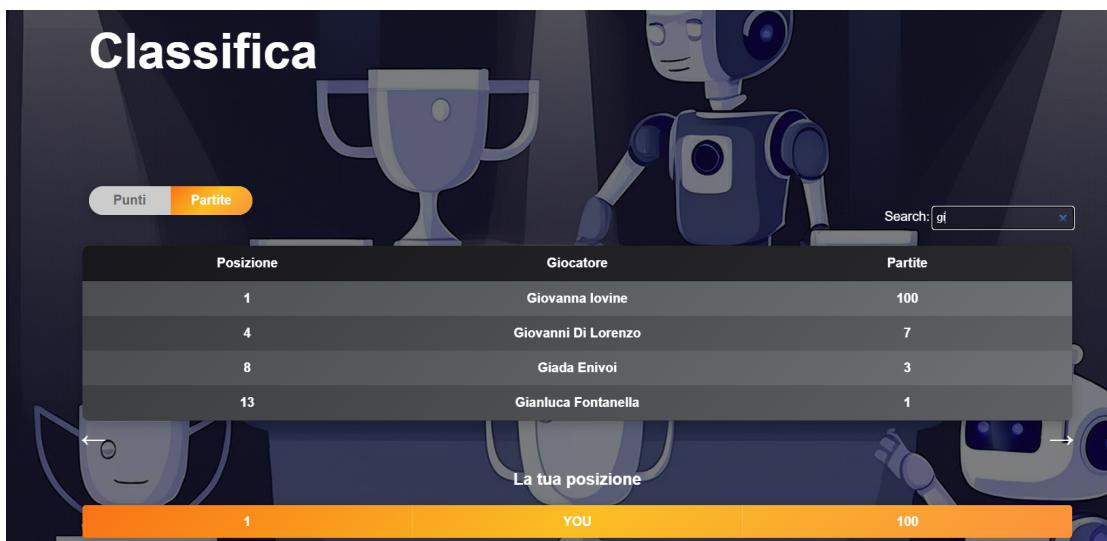
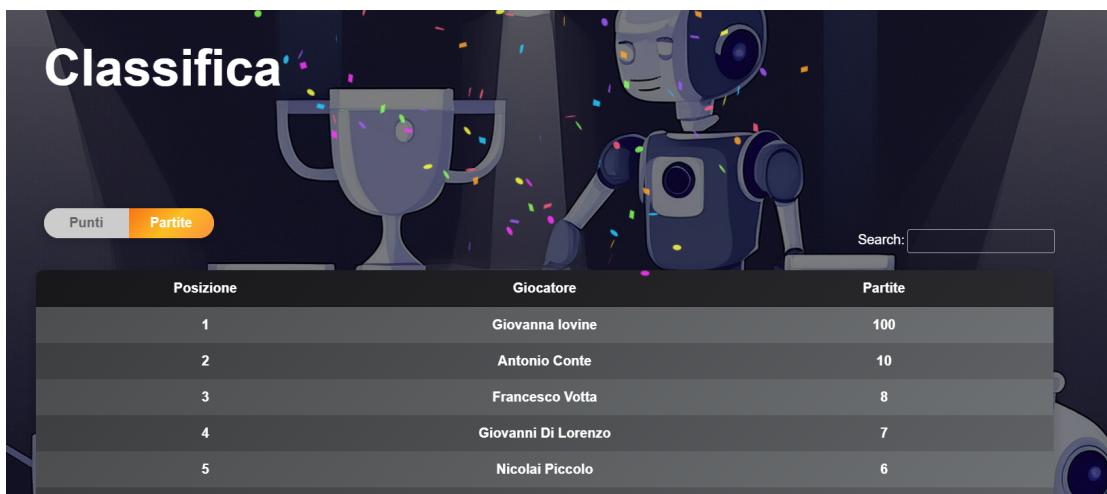


Figure 5.3: Foto funzionamento toggle e barra di ricerca

CHAPTER 5. IMPLEMENTAZIONE E STRUTTURA DEL PROGETTO MODIFICATO



The screenshot shows a game classification interface. At the top left, the word "Classifica" is displayed in large white letters. To its right is a silver trophy with confetti falling around it. Further right is a blue and grey robot head. Below the title is a navigation bar with two tabs: "Punti" (Points) and "Partite" (Matches), with "Partite" being the active tab. To the right of the tabs is a search bar labeled "Search: []". The main area is a table with five rows, showing player statistics. The columns are "Posizione" (Position), "Giocatore" (Player), and "Partite" (Matches). The data is as follows:

Posizione	Giocatore	Partite
1	Giovanna Iovine	100
2	Antonio Conte	10
3	Francesco Volta	8
4	Giovanni Di Lorenzo	7
5	Nicolai Piccolo	6

Figure 5.4: Foto sparacoriandoli

Chapter 6

Testing

6.1 Testing della GUI

Per testare il corretto funzionamento dell’interfaccia grafica è stato effettuato un testing manuale relativamente al quale, si riporta di seguito la test suite.

Per ciascun caso di test si specifica l’obiettivo, i passaggi di input effettuati, un confronto tra output atteso e output ottenuto e le precondizioni che hanno dato vita a tale risultato. Rispetto a ciò si decreta se ciascun caso di test ha avuto esito positivo o meno.

Si è scelto di non considerare le post condizioni, poichè si tratta di una vista dinamica, tramite la quale non si ha la possibilità di modificare in alcun modo lo stato del sistema.

TEST SUITE

ID Test	Descrizione	Precondizioni	Input	Output atteso	Output ottenuto	Esito
TC001	Visualizzazione classifica in lingua italiana Verificare che tramite il pulsante "Classifica" sulla navbar venga visualizzata una classifica dei giocatori ordinata in base ai punti di questi ultimi.	L'utente ha effettuato il login. Vi è almeno un giocatore che ha almeno un punto. Al termine di ogni partita, i dati relativi a quest'ultima vengono correttamente salvati sul database. Aver selezionato la lingua italiana.	Cliccare "Classifica" dalla navbar	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Punti". Tutte le intestazioni sono in italiano, la classifica è correttamente ordinata.	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Punti". Tutte le intestazioni sono in italiano, la classifica è correttamente ordinata.	PASS
TC002	Visualizzazione della posizione dell'utente corrente in un riquadro sotto la classifica Verificare che sotto alla classifica sia possibile visualizzare in modo veloce e intuitivo la propria posizione in un riquadro dedicato.	L'utente ha effettuato il login. Al termine di ogni partita, i dati relativi a quest'ultima vengono correttamente salvati sul database.	Cliccare "Classifica" dalla navbar	Visualizzazione di un riquadro sotto la classifica con la propria posizione e i propri punti.	Visualizzazione di un riquadro sotto la classifica con la propria posizione e i propri punti.	PASS

TC003	Visualizzazione classifica tramite il meccanismo di internazionalizzazione Verificare che la classifica venga visualizzata correttamente anche nelle altre lingue previste	L'utente ha effettuato il login. Vi è almeno un giocatore che ha almeno un punto. Al termine di ogni partita, i dati relativi a quest'ultima vengono correttamente salvati sul database. Aver selezionato la lingua inglese o spagnola.	Scegliere la lingua inglese/spagnola. Cliccare "Leaderboard"/"Tabla de clasificación" dalla navbar.	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Punti". Tutte le intestazioni sono nella lingua scelta.	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Punti". Tutte le intestazioni sono nella lingua scelta.	PASS
TC004	Funzionamento toggle per switchare tra le due modalità di visualizzazione previste Verificare che cliccando su "Partite" nel toggle, si colori questa metà ad indicare la modalità selezionata, e venga mostrata la classifica correttamente ordinata per partite vinte.	L'utente ha effettuato il login. L'utente sta visualizzando la pagina della classifica. Vi è almeno un giocatore che ha vinto almeno una partita. Al termine di ogni partita, i dati relativi a quest'ultima vengono correttamente salvati sul database.	Switch del toggle	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Partite", e del riquadro relativo all'utente corrente. Il toggle presenta la voce "Partite" colorata. La classifica è correttamente ordinata per partite vinte.	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Partite", e del riquadro relativo all'utente corrente. Il toggle presenta la voce "Partite" colorata. La classifica è correttamente ordinata per partite vinte.	PASS

TC005	Funzionamento toggle per tornare alla modalità "Punti" Verificare che cliccando su "Punti" nel toggle, si colori questa metà ad indicare la modalità selezionata, e venga mostrata la classifica correttamente ordinata per punti.	L'utente ha effettuato il login. L'utente sta visualizzando la pagina della classifica ordinata per partite vinte. Vi è almeno un giocatore che ha almeno un punto. Al termine di ogni partita, i dati relativi a quest'ultima vengono correttamente salvati sul database.	Switch del toggle	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Punti", e del riquadro relativo all'utente corrente. Il toggle presenta la voce "Punti" colorata. La classifica è correttamente ordinata per punti.	Visualizzazione di una tabella con tre campi: "Posizione", "Giocatore", "Punti", e del riquadro relativo all'utente corrente. Il toggle presenta la voce "Punti" colorata. La classifica è correttamente ordinata per punti.	PASS
TC006	Classifica mostrata ad un utente che ha zero punti Verificare che un utente con zero punti non compaia in classifica	L'utente ha effettuato il login. L'utente ha zero punti.	Cliccare "Classifica" dalla navbar	L'utente non viene mostrato in classifica e nel riquadro personale sottostante, realtivamente alla voce "posizione" non abbiamo alcun valore, mentre in corrispondenza della voce "punti" viene indicato il valore 0.	L'utente non viene mostrato in classifica e nel riquadro personale sottostante, realtivamente alla voce "posizione" non abbiamo alcun valore, mentre in corrispondenza della voce "punti" viene indicato il valore 0.	PASS

TC007	Classifica mostrata ad un utente che non ha vinto alcuna partita Verificare che un utente con zero partite vinte non compaia in classifica	L'utente ha effettuato il login. L'utente non ha vinto alcuna partita.	Cliccare "Classifica" dalla navbar. Switchare alla modalità "Partite" tramite il toggle.	L'utente non viene mostrato in classifica e nel riquadro personale sottostante, realtivamente alla voce "posizione" non abbiamo alcun valore, mentre in corrispondenza della voce "partite" viene indicato il valore 0.	L'utente non viene mostrato in classifica e nel riquadro personale sottostante, realtivamente alla voce "posizione" non abbiamo alcun valore, mentre in corrispondenza della voce "partite" viene indicato il valore 0.	PASS
TC008	Ricerca utente presente in classifica Verificare che sia possibile cercare un utente presente in classifica, tramite l'identificativo previsto (nome e cognome).	L'utente ha effettuato il login. L'utente sta visualizzando la pagina della classifica. L'utente cercato è presente in classifica.	Nella barra "Search" scrivere il nome da cercare.	La tabella che costituisce la classifica mostra i valori che corrispondono alla chiave di ricerca.	La tabella che costituisce la classifica mostra i valori che corrispondono alla chiave di ricerca.	PASS
TC009	Ricerca utente non presente in classifica Verificare che venga mostrata una classifica vuota se non è presente in classifica l'utente cercato, tramite l'identificativo previsto (nome e cognome).	L'utente ha effettuato il login. L'utente sta visualizzando la pagina della classifica. L'utente cercato non è presente in classifica.	Nella barra "Search" scrivere il nome da cercare.	La classifica è vuota.	La classifica è vuota e presenta il messaggio "No matching records"	PASS

TC010	Visualizzazione di 10 giocatori alla volta Verificare che sia possibile scorrere tramite freccette la classifica visualizzando in una pagina 10 giocatori alla volta.	L'utente ha effettuato il login. L'utente sta visualizzando la pagina della classifica.	Click delle frecce per vedere i successivi/precedenti giocatori in classifica	Con la freccetta ">" la classifica mostra i successivi 10 giocatori, con la freccetta "<" la classifica mostra i precedenti 10 giocatori.	Con la freccetta ">" la classifica mostra i successivi 10 giocatori, con la freccetta "<" la classifica mostra i precedenti 10 giocatori.	PASS
-------	---	---	---	---	---	------

6.2 Testing API

E' stata aggiunta un'API che restituisce una lista di oggetti JSON che rappresentano tutti i giocatori presenti nella tabella "player" del database in T4, con i relativi campi.

Per testare il correttamento funzionamento di quest'ultima è stato usato Postman facendo riferimento alla test suite riportata di seguito.

TEST SUITE

ID Test	Descrizione	Precondizioni	Output atteso	Output ottenuto	Esito
TC001	Tabella players restituita correttamente Viene restituita la tabella dei giocatori con tutte le informazioni correttamente	L'id di ogni utente registrato viene salvato nel database. Al termine di ogni partita i dati vengono salvati in database.	Lista json di giocatori con le relative informazioni.	Lista json di giocatori con le relative informazioni.	PASS

TC002	Database vuoto Se il database è vuoto viene restituito un oggetto vuoto.	Il database è vuoto	Viene restituito un oggetto vuoto.	Viene restituito un oggetto vuoto.	PASS
TC003	Database non trovato Se il database non viene trovato, i dati richiesti non possono essere ottenuti e ci si aspetta un messaggio di errore	Database non collegato correttamente	Errore Connection Refused	Error: ECONNREFUSED	PASS

I seguenti test case sono stati aggiunti alla collection di test Postman presente in T4, in modo da offrire un quadro completo per i test dell'applicazione.

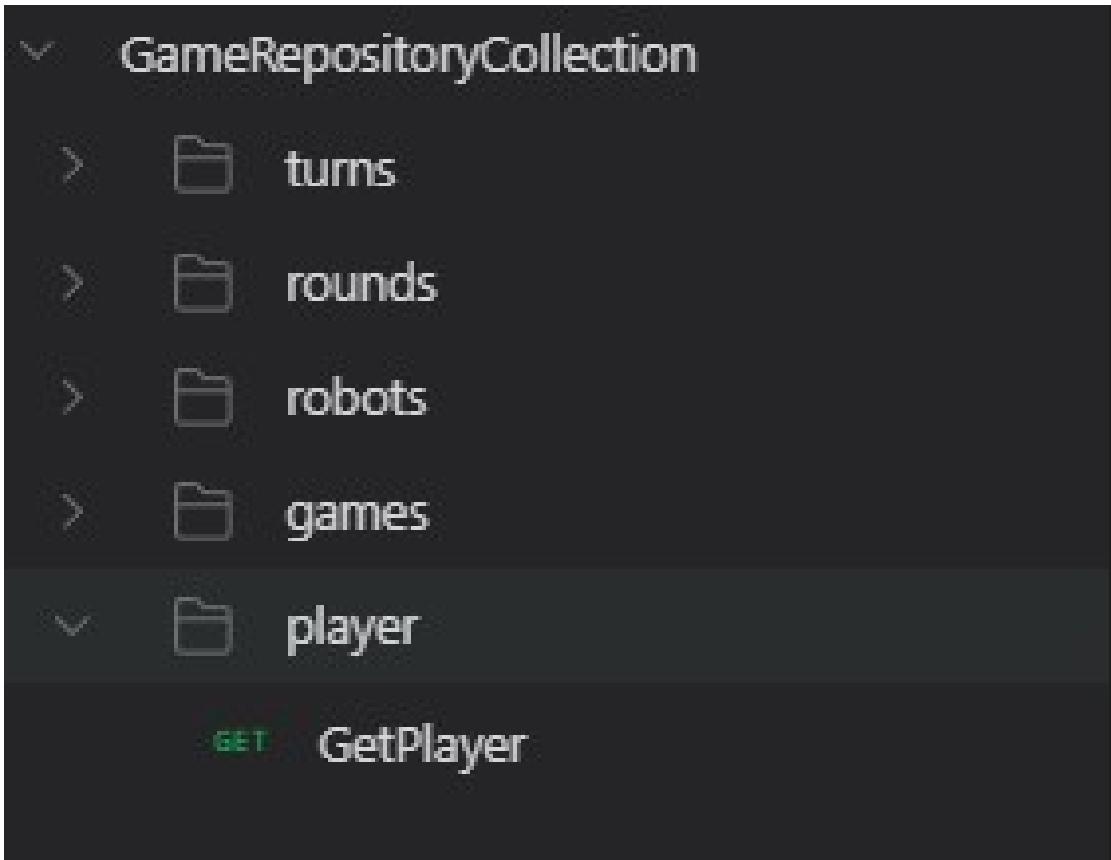


Figure 6.1: Game Repository Collection Test Postman.

Test Case 001

```

[{"id": "101", "accountId": "101", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 917, "gameSession": 322}, {"id": "102", "accountId": "102", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 577, "gameSession": 18}, {"id": "103", "accountId": "103", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 838, "gameSession": 394}, {"id": "104", "accountId": "104", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 436, "gameSession": 282}, {"id": "105", "accountId": "105", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 163, "gameSession": 198}, {"id": "106", "accountId": "106", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 489, "gameSession": 141}, {"id": "107", "accountId": "107", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 167, "gameSession": 483}, {"id": "108", "accountId": "108", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 123, "gameSession": 268}, {"id": "109", "accountId": "109", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 176, "gameSession": 463}, {"id": "110", "accountId": "110", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 171, "gameSession": 17}, {"id": "111", "accountId": "111", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 489, "gameSession": 499}, {"id": "112", "accountId": "112", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 507, "gameSession": 150}, {"id": "113", "accountId": "113", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 988, "gameSession": 480}, {"id": "114", "accountId": "114", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 597, "gameSession": 380}, {"id": "115", "accountId": "115", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 383, "gameSession": 398}, {"id": "116", "accountId": "116", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 779, "gameSession": 295}, {"id": "117", "accountId": "117", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 547, "gameSession": 180}, {"id": "118", "accountId": "118", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 452, "gameSession": 117}, {"id": "119", "accountId": "119", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 310, "gameSession": 213}, {"id": "120", "accountId": "120", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 734, "gameSession": 65}, {"id": "121", "accountId": "121", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 304, "gameSession": 122}, {"id": "122", "accountId": "122", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 123, "gameSession": 123}, {"id": "123", "accountId": "123", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 148, "gameSession": 226}, {"id": "124", "accountId": "124", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 593, "gameSession": 305}, {"id": "125", "accountId": "125", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 226, "gameSession": 120}, {"id": "126", "accountId": "126", "createdAt": "2001-01-01T00:00:00Z", "updatedAt": "2001-01-01T00:00:00Z", "points": 190, "gameSession": 190}

```

Figure 6.2: Testing del TC001 tramite Postman

Test Case 002

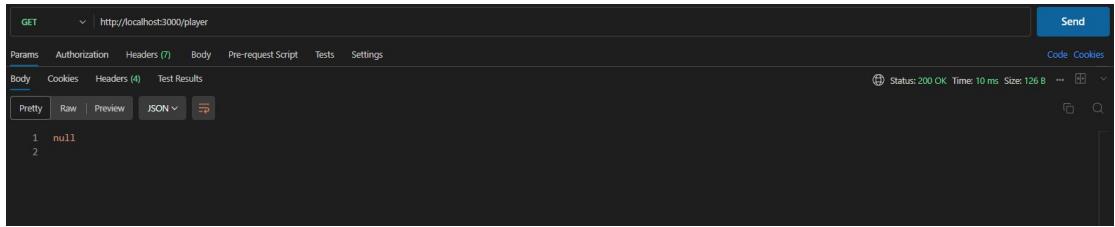


Figure 6.3: Testing del TC002 tramite Postman

Test Case 003

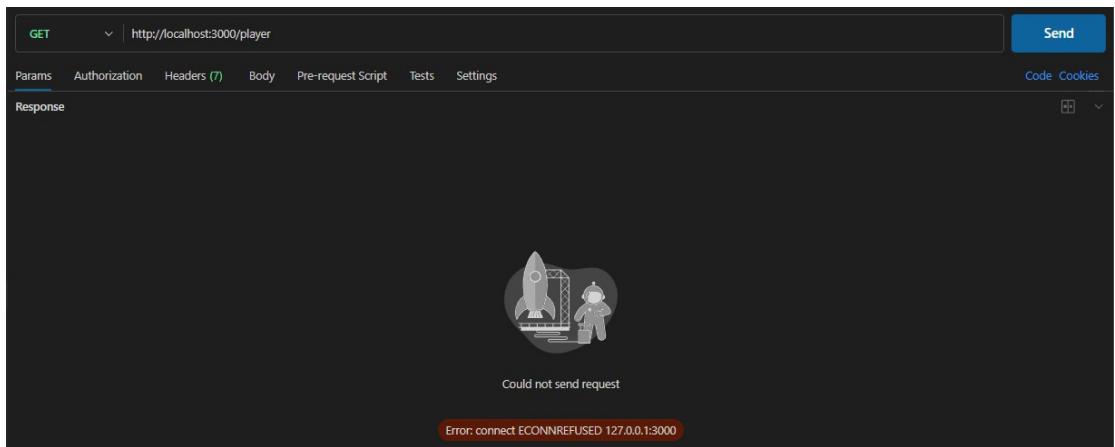


Figure 6.4: Testing del TC003 tramite Postman

6.3 Testing TRIGGER

Siccome sono stati aggiunti dei trigger nel database T4 PostgreSQL è necessario verificare il loro funzionamento. Essi sono stati inseriti grazie all'ausilio di GORM. I trigger inseriti con GORM non sono regole o script SQL associate a tabelle che vengono eseguiti automaticamente, infatti non implementano la logica dei trigger nativi come fa un database. GORM infatti consente di utilizzare i cosiddetti "hook" che

sono funzioni definite nell'applicazione. Questi hook sono eseguiti solo quando l'applicazione interagisce con il database attraverso GORM. Per cui se i dati vengono modificati direttamente nel database, GORM non è coinvolto e quindi i suoi hook non vengono attivati. Infatti per verificare le correttezza di quest'ultimi sono inefficienti le modifiche direttamente da database ma è necessario richiamare quelle parti di codice che utilizzano GORM, nel nostro caso richiamando le API che espone T4.

I trigger inseriti sono due.

Trigger 1

Il primo è stato inserito in T4-G18/model/model.go. Ogni qual volta viene aggiornata una riga della tabella Game verrà aggiornato il campo Points del player (o più player) associato a quel Game sommando il punteggio precedente con lo Score conseguito in quel Game. Per testare questo trigger è necessario fare una PUT sul game relativo a quel giocatore con uno Score >0 per vedere se tale modifica si riflette anche nella tabella Player. E' necessaria una PUT poichè dobbiamo aggiornare una riga già esistente. La seguente chiamta REST HTTP PUT è stata fatta con l'ausilio di CURL.

```
curl -X PUT http://localhost:3000/games/{id}
```

```
-H "Content-Type: application/json"
```

```
-d "{\"score\": value}"
```

Questa riportata è una chiamata generica, i valori "id" e "value" si intendano da sostituire.

Esempio Trigger 1

Verifichiamo il funzionamento del trigger sulla riga con id 1 della tabella Game.

	Q	*id	bigint	name	text	username	text	current_round	bigint	description	text	difficulty	text	score	numeric	is_winner	boolean
			Filter		Filter			Filter		Filter		Filter		Filter		Filter	
		>	1		name		(NULL)		1		Sfida		difficulty		0		false

Figure 6.5: Game con id 1 prima della chiamata PUT.

	Q	*id	bigint	account_id	text	points	numeric	games_won	bigint	created_at	timestamp with time z	updated_at	timestamp with time z
			Filter		Filter		Filter		Filter	Filter		Filter	
		>	1001		1001		0		0		0001-01-01 00:00:00+00		2024-12-12 08:29:36.35916-

Figure 6.6: Player collegato a quel Game prima dell'effetto del trigger.

Facciamo ora chiamata PUT per aggiornare lo score del Game con id 1.

```
curl -X PUT http://localhost:3000/games/1
-H "Content-Type: application/json"
-d "{\"score\": 10}"
```

Questa modifica si rifletterà su Game:

	*id bigint	name text	username text	current_round bigint	description text	difficulty text	score numeric	is_winner boolean
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
>	1	name	(NULL)	1	Sfida	difficulty	10	false

Figure 6.7: Game con id 1 dopo la chiamata PUT.

Ed attiverà il trigger aggiornando il campo points del Player corrispondente:

	*id bigint	account_id text	points numeric	games_won bigint	created_at timestamp with time z	updated_at timestamp with time z
	Filter	Filter	Filter	Filter	Filter	Filter
>	1001	1001	10	0	0001-01-01 00:00:00+00	2024-12-12 08:34:12.49518:

Figure 6.8: Player collegato a quel Game dopo l'effetto del trigger.

Trigger 2

Il secondo trigger è stato inserito in T4-G18/api/game/service.go. Questo trigger verifica se il Game appena modificato ha il campo isWinner pari a True, andando di conseguenza ad aggiornare il campo isWinner della tabella PlayerGame (modifica non strettamente necessaria ma aggiunta poichè non è stata prevista nelle precedenti implementazioni) e incrementando il numero di partite vinte del player (o più player) che ha giocato. Per testare questo trigger è necessario fare una PUT sul game relativo a quel giocatore con il campo isWinner impostato a True per vedere se tale modifica si riflette anche nella tabella Player.

```
curl -X PUT http://localhost:3000/games/{id}
-H "Content-Type: application/json"
-d "{\"isWinner\": {Boolean_Value}}"
```

Questa riportata è una chiamata generica, i valori "id" e "Boolean Value" si intendano da sostituire.

Esempio Trigger 2

Verifichiamo il funzionamento del trigger2 sulla riga con id 1 della tabella Game.

	*id bigint	name text	username text	current_round bigint	description text	difficulty text	score numeric	is_winner boolean
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
>	1	name	(NULL)	1	Sfida	difficulty	10	false

Figure 6.9: Game con id 1 prima della chiamata PUT.

	*id bigint	account_id text	points numeric	games_won bigint	created_at timestamp with time z	updated_at timestamp with time z
	Filter	Filter	Filter	Filter	Filter	Filter
>	1001	1001	10	0	2001-01-01 00:00:00+00	2024-12-12 08:34:12.495183

Figure 6.10: Player collegato a quel Game prima dell'effetto del trigger.

	*player_id text	*game_id bigint	created_at timestamp with time z	updated_at timestamp with time z	is_winner boolean
	Filter	Filter	Filter	Filter	Filter
	1001	1	2024-12-11 15:13:27.451493	2024-12-11 16:26:19.807171	false

Figure 6.11: PlayerGame collegato a quel Game/Player prima dell'effetto del trigger.

Facciamo ora chiamata PUT per aggiornare il campo isWinner del Game con id 1.

```
curl -X PUT http://localhost:3000/games/1
-H "Content-Type: application/json"
-d "{\"isWinner\": true}"
```

Questa modifica si rifletterà su Game:

* id	name	username	current_round	description	difficulty	score	is_winner
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	name	(NULL)	1	Sfida	difficulty	10	true

Figure 6.12: Game con id 1 dopo la chiamata PUT.

Ed attiverà il trigger incrementando il campo GamesWon del Player corrispondente:

* id	account_id	points	games_won
Filter	Filter	Filter	Filter
1001	1001	10	1

Figure 6.13: Player collegato a quel Game dopo l'effetto del trigger.

e aggiornando la riga della tabella PlayerGame corrispondente:

Q	* player_id	* game_id	created_at	updated_at	is_winner
Filter	Filter	Filter	Filter	Filter	Filter
>	1001	1	2024-12-11 15:13:27.45149	2024-12-12 08:39:28.36439	true

Figure 6.14: PlayerGame collegato a quel Game dopo l'effetto del trigger.

Chapter 7

Sviluppi Futuri

7.1 Tipi di classifiche

Il task sviluppato e implementato dal nostro team presenta ampie possibilità di miglioramento e ampliamento. Una possibile evoluzione potrebbe includere l'aggiunta di diversi tipi di classifiche.

Classifiche nazionali, che suddividono i giocatori in base al loro paese di appartenenza, permettendo così una competizione su scala geografica.

Classifica dedicata ai team di giocatori, in cui i gruppi accumulano punti completando missioni o raggiungendo obiettivi comuni, favorendo la competizione tra squadre.

Classifica periodica, in analogia con la presenza di missioni periodiche, si potrebbe avere una classifica giornaliera, settimanale o mensile.

Classifica relativa al robot sfidato, rispetto alla possibilità di legare anche quest'informazione al game che viene salvato nel database al termine di ogni partita.

Una prima idea per gestire questa moltitudine di classifiche potrebbe essere quella di sostituire il toggle con un menù a tendina, o prevedendo la possibilità di selezionare la modalità semplicemente cliccando sull'header della classifica. Tuttavia, nello scenario di un continuo ampliamento del gioco e quindi di più modalità di gioco in cui ciascuna delle quali potrebbe avere una classifica relativa, risulterebbero eccessivamente appesantite la navbar e la pagina. Per garantire invece un alto livello di usabilità, cliccando sul tasto classifica nella navbar si potrebbe accedere a una pagina che attraverso tasti grandi e intuitivi (come quelli proposti dal gruppo B16, per l'interfaccia dell'amministratore), permette di scegliere il tipo di classifica che si vuole visualizzare. La pagina a cui poi si accede resterebbe la stessa che proponiamo in questa versione del progetto, con il toggle per switchare tra le due modalità di interazione. Avremmo così ottenuto una soluzione semplice, intuitiva e interattiva.

Analogamente all'idea dei coriandoli, è possibile ideare altre forme di "ricompense" che, attraverso principi di gamification, possano migliorare significativamente l'esperienza di apprendimento dell'utente.

Sarebbe poi necessario, ai fini della corretta gestione dei casi di errore, prevedere l'implementazione di una pagina html da mostrare all'utente nel caso di problematiche riscontrate.

7.2 Migliorare le performance

7.2.1 Problema riscontrato

Il gioco nasce con lo scopo di aiutare gli studenti dei corsi di Software Testing ad imparare tramite la gamification. Immaginando un grande successo di questo progetto e quindi un vasto uso da parte degli studenti di tutta Europa, ci si ritrovrebbe presto con un gran numero di studenti registrati, che giocano solo per un periodo limitato di tempo. Ci si augura infatti che il gioco aiuti gli studenti a superare gli esami e a non averne più bisogno il più presto possibile. Di fronte a questo scenario, se ogni volta venisse fatta una query per recuperare gli identificativi di tutti gli studenti che hanno giocato almeno una partita, ci si ritroverebbe ad aver a che fare con un numero di dati via via crescente, che però non desta interesse all'utente corrente.

7.2.2 Soluzioni proposte

A tal fine, si è pensato di implementare un meccanismo per cui dopo un certo periodo di tempo di inattività i punti vengono azzerati, in

modo che individuata la lista di giocatori che ha almeno un punto dal database in T4, viene fatta una query con gli id di questi ultimi al database in T23, per richiedere solo gli identificativi (nome e cognome) necessari.

Se invece volessimo mantenere i punteggi degli studenti, per poter avere uno storico, utile per report di confronto dei vari anni, si può pensare di avere una classifica annuale, utilizzando l'informazione data dal timestamp di ciascuna partita. Analogamente si individuano gli id dei giocatori che attivi nell'anno corrente con una query al database in T4 e si fa una query per ottenere i soli identificativi di questi ultimi dal database di T23.

Per la query al database in T23 si prevede di utilizzare l'API realizzata dal gruppo B16, per ottenere le informazioni degli studenti appartenenti a un team fornendo la lista di ID di questi ultimi. Nel nostro caso si vuole invece passare la lista di id degli studenti da mostrare in classifica per migliorare l'efficienza.

7.3 Guida ad implementazioni future

Per aggiungere nuove tipologie di classifiche, anche discusse nelle sezioni precedenti, dobbiamo andare a modificare alcuni moduli. Il modulo principale dove vengono fatte le query al database e dove avviene il filtraggio dei dati è **LeaderboardService** in T5 all'interno del package Service. In questo modulo è presente un metodo "GetList()" che va

a fare la join dei dati dal database T23 e dal database T4, ottenendo così una lista di elementi di tipo PlayerDTO. Per aggiungere un ulteriore filtro, bisogna un aggiungere un metodo che filtri questa lista di PlayerDTO secondo i criteri scelti e un attributo della classe che contenga la nuova classifica. Il metodo aggiunto deve essere richiamato in GetList() in modo tale che assegni la lista che restituisce all'attributo creato. Invece per sapere il rank, bisogna modificare PlayerDTO aggiungendo un campo che contenga il rank (un intero) e modificare il metodo getPlayerRank(), aggiungendo una logica che calcoli il rank secondo i nuovi criteri adottati. Nel codice sono presenti commenti che spiegano i metodi citati.