



**UNIVERSITÀ⁹ DEGLI STUDI DI
NAPOLI FEDERICO II**

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

Software Architecture Design

Documentazione Team A5

Anno Accademico 2023/24

Prof.ssa: **Anna Rita Fasolino**

Membri del team:

Gianmarco Cascone
Giovanni Liguori
Francesco Riccio

Contents

1	Introduzione	1
1.1	Descrizione del progetto	2
1.2	Descrizione del requisito assegnato	3
1.3	Descrizione del requisito R9	5
1.3.1	Database analizzati	6
1.4	Descriiizione del requisito R11	9
1.4.1	Database analizzati	11
2	Metodologie di sviluppo	13
2.1	SCRUM	14
2.2	Sprint Backlog	15
2.3	Divisione ed organizzazione del lavoro	15
2.4	Strumenti utilizzati	17
2.4.1	GitHub	17
2.4.2	Visual Paradigm	19
2.4.3	Microsoft Teams	20
2.4.4	Trello	20
2.4.5	Notion	21
2.4.6	Visual Studio Code	22

2.4.7	Java Spring	23
2.4.8	Spring Boot	24
2.4.9	Maven	25
2.4.10	Go	26
2.4.11	Postman	27
2.4.12	HTTPie	29
2.4.13	DBgate	30
3	Stato iniziale del progetto	33
3.1	Architettura del sistema	34
3.1.1	Workflow	36
3.2	Stato del task T4	38
3.2.1	Architettura Game Repository	38
3.2.2	Swagger	44
3.2.3	Documentazione API	46
3.2.4	CreateGame sequence diagram	49
3.2.5	UpdateGame sequence diagram	51
3.2.6	UpdateTurn sequence diagram	53
3.3	Stato del Task T5	55
4	Modifiche al componente T4	57
4.1	Modifiche al diagramma ER	57
4.1.1	Tabella PlayerGame	58
4.1.2	Tabella Player	59
4.1.3	Tabella Game	59
4.1.4	Tabella Round	60
4.1.5	Tabella Turn	60

4.1.6	Tabella Metadata	60
4.2	Modifiche al codice	61
4.2.1	Modifiche al file model.go	61
4.2.2	Modifiche al file Game.java in T5	63
4.2.3	UpdateGame in game/service.go	64
4.3	Modifiche ai diagrammi di sequenza	66
4.3.1	Diagramma di sequenza di UpdateGame	66
4.3.2	Diagramma di sequenza di UpdateTurn	68
4.4	Altre modifiche	71
4.4.1	API diagram	71
4.4.2	System Model Domain	72
5	Requisito R9	74
5.1	Storico partite	74
5.1.1	Data Partita	75
5.1.2	Durata	75
5.1.3	Classe Testata, Robot e Difficoltà	77
5.1.4	Page Report	77
5.1.5	Ricezione e Salvataggio delle variabili Class, Robot e Difficulty	79
5.1.6	Vincitore	83
5.1.7	LOC Coverage Giocatore e Robot	83
5.2	Classifica giocatori	85
5.2.1	Nome Giocatore	85
5.2.2	Partite Giocate	86
5.2.3	Partite vinte	87

6 Requisito R11	89
6.1 Elenco dei giocatori iscritti	89
6.1.1 Nome, cognome, corso di studi	90
6.1.2 Numero di classi testate e Numero di partite giocate	91
6.1.3 Tempo totale delle partite giocate	92
6.2 Elenco delle classi disponibili	94
6.2.1 Nome	94
6.3 Altre considerazioni	95
6.3.1 Elenco giocatori iscritti	96
6.3.2 Elenco classi disponibili	96
6.3.3 Classifica giocatori	97
7 Futuri sviluppi : Rimozione Round	98
7.1 Introduzione	98
7.1.1 Motivazioni della rimozione di Round	100
7.2 Modifiche alla strutture del progetto	101
7.3 Aggiornamenti ai servizi REST	103
7.3.1 Retrieve a Game REST Service	104
7.3.2 Update Game REST Service	104
7.3.3 Create Turns REST Service	105
7.3.4 Retrieve Turns REST Service	106
7.4 Conclusioni	106
8 Consigli e Risoluzione dei Problemi Installazione	108
8.1 Introduzione	108
8.2 Compilazione del Progetto con Maven	109

8.2.1	Cos'è Maven?	109
8.2.2	Utilizzo di Maven per la Compilazione	109
8.3	Gestione di Docker per una Disinstallazione Pulita . .	110
8.3.1	Il Ruolo di Docker	110
8.3.2	Procedura per una Disinstallazione Pulita	110
8.4	Debugging con gli Strumenti di Sviluppo di Chrome .	111
8.4.1	Il Tasto F12 e gli Strumenti di Sviluppo	111
8.4.2	Analisi delle Chiamate API	112
8.5	Il Ruolo Essenziale di Node.js	113
8.5.1	Introduzione a Node.js	113
8.5.2	Node.js e VSCode	113
8.6	Conversione degli End of File (EOF)	114
8.6.1	Problemi con CRLF e LF	114
8.7	Soluzione	114
9	Installazione	116
9.1	Docker e WSL	116
9.2	Deployment dell'applicazione	117
9.2.1	Download applicazioni necessarie	117
9.2.2	Installazione	118
9.2.3	Configurazione MongoDB	118
9.3	Utilizzo dell'applicazione	119
9.3.1	Registrazione admin	120
9.3.2	Caricamento classi	120
9.3.3	Registrazione utente	122

10 Glossario	124
10.1 Game	124
10.2 Player	124
10.3 Robot	124
10.4 Round	125
10.5 Turn	125

Chapter 1

Introduzione

ENACTEST (European iNnovative AllianCe for TESTing) rappresenta un'iniziativa pionieristica che mira a rivoluzionare il campo dell'ingegneria e della validazione di qualità in Europa. Al centro di questo progetto c'è l'obiettivo di creare una rete collaborativa tra istituzioni accademiche, industrie e organizzazioni di ricerca, con l'intento di sviluppare e implementare metodologie di test innovative e all'avanguardia. ENACTEST si propone di affrontare le sfide emergenti nel settore del testing, integrando tecnologie avanzate come l'intelligenza artificiale, il machine learning e i sistemi cyber-fisici. Questo progetto non solo mira a elevare gli standard di qualità e affidabilità dei prodotti europei, ma anche a promuovere una cultura di condivisione del sapere e di collaborazione transfrontaliera. Con ENACTEST, l'Europa si pone all'avanguardia nel definire il futuro del testing, puntando a risultati che possano beneficiare l'intero ecosistema industriale e accademico.

del continente. Attraverso ENACTEST, si auspica di suscitare 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. Il progetto si pone come un importante passo avanti nel migliorare la qualità delle applicazioni web e nell'educare la comunità di sviluppatori sulla rilevanza fondamentale del testing nel mondo dello sviluppo software.

1.1 Descrizione del progetto

Il progetto in questione si colloca all'interno del corso di Software Architecture Design nell'anno accademico 2023/24, con l'intento di fornire un'esperienza educativa e interattiva agli studenti attraverso lo sviluppo di un gioco incentrato sull'importanza delle pratiche di testing nel ciclo di vita dello sviluppo software. L'obiettivo fondamentale di questa iniziativa è di sollevare consapevolezza e promuovere un approccio critico verso le metodologie di testing, essenziali per garantire la qualità e l'affidabilità del software.

Il gioco si propone come strumento didattico innovativo, progettato per inserirsi efficacemente nel curriculum accademico del corso di Software Architecture Design. Attraverso un approccio ludico, mira a stimolare l'interesse degli studenti verso le pratiche di testing, una componente cruciale ma spesso sottovalutata dello sviluppo software. La sfida principale consiste nel testare classi scritte in Java, valutando l'efficacia e

l'efficienza dei test implementati dai partecipanti.

La versione del progetto da cui siamo partiti è "T10-G40".

link github: <https://github.com/Testing-Game-SAD-2023/T10-G40>

1.2 Descrizione del requisito assegnato

Il lavoro del nostro gruppo si è concentrato sulla precisa implementazione e verifica di un requisito critico assegnato, che ha delineato un obiettivo chiaro e articolato per il nostro progetto. Questo obiettivo era incentrato sulla necessità di garantire che le informazioni raccolte e memorizzate nel database associato al componente T4 del nostro sistema, riguardanti ogni singola partita e ciascun turno di gioco fino al termine della partita stessa, fossero complete e adeguatamente strutturate per non solo soddisfare i requisiti di base ma anche per appoggiare la realizzazione di funzionalità avanzate. In particolare, il requisito punta a verificare l'adeguatezza delle informazioni salvate per supportare gli ulteriori Requisiti R9 ed R11, i quali riguardano rispettivamente la gestione dello storico delle partite degli utenti e la creazione di un cruscotto per l'amministratore, due aspetti fondamentali per arricchire l'esperienza utente e promuovere un ambiente di gioco competitivo.

Per assicurare che questo obiettivo fosse raggiunto, il nostro team ha intrapreso un'approfondita analisi e revisione dei componenti T4 e T5 del progetto, introducendo modifiche significative e ottimizzazioni mi-

- R3) Verificare che le informazioni salvate nel database di T4 per ogni partita ad ogni turno di gioco e al momento della fine della partita siano tutte quelle necessarie a realizzare anche i Requisiti aggiuntivi per la gestione dello storico delle partite Utente e la creazione di classifiche di giocatori (v. requisiti R9 ed R11).

Figure 1.1: Task assegnato

rate. Queste modifiche hanno riguardato sia l'architettura del database che le logiche di programmazione impiegate per la raccolta e la gestione delle informazioni di gioco. Sono stati specificamente migliorati gli schemi di database per assicurare che ogni dettaglio relativo alle partite - inclusi i dati di ogni turno, i punteggi finali, le statistiche di gioco e le interazioni tra i giocatori - fosse accuratamente catturato e memorizzato in modo da facilitare analisi complesse e la generazione di insight significativi.

Inoltre, abbiamo implementato nuove funzioni e procedure per il trattamento e l'analisi dei dati, garantendo che le informazioni raccolte soddisfassero pienamente i requisiti R9 e R11. Questo include la capacità di tracciare con precisione il percorso di ogni giocatore attraverso le diverse partite, consentendo una gestione dettagliata dello storico delle partite e la creazione dinamica di classifiche basate su vari criteri, come il numero di vittorie, la frequenza di gioco e le performance complessive.

Tutte le modifiche apportate ai componenti T4 e T5 sono state guidate dalla necessità di rispondere efficacemente al requisito assegnato.

Questo sforzo ha richiesto una collaborazione stretta tra i membri del team, un approccio metodico alla risoluzione dei problemi e un impegno costante nel testare e rifinire le soluzioni proposte per garantire che il risultato finale soddisfacesse le aspettative. Il dettaglio di queste analisi, le scelte progettuali adottate e i risultati ottenuti saranno esposti nei capitoli successivi, dove verrà dimostrato come le modifiche implementate abbiano contribuito in modo significativo a realizzare un sistema robusto, flessibile e in grado di supportare i requisiti avanzati del progetto.

1.3 Descrizione del requisito R9

Nel documento visualizzato nell’immagine 1.2, viene presentato il Requisito R9, che riguarda l’implementazione di nuove funzionalità, attivabili dall’utente dopo il processo di login. La descrizione è divisa in due sezioni principali.

La prima sezione (a) riguarda la possibilità per l’utente di accedere a un registro storico delle partite precedentemente giocate. Per ogni partita, il sistema è tenuto a fornire una serie di dettagli comprensivi di data e durata, classe testata, l’identità del robot sfidato, il vincitore della partita e il livello di copertura del codice raggiunto, sia dall’utente che dal robot.

La seconda sezione (b) si concentra sull’introduzione di nuove modalità di gioco. Viene data la scelta di iniziare una partita base, che appare

essere già una funzionalità presente nell'app, oppure di optare per una modalità più competitiva, "Partita Contro tutti i Robot".

Infine, un elemento aggiuntivo è la richiesta di implementare una vista che permetta agli utenti di visualizzare la classifica totale dei giocatori, ordinata in base al numero di partite giocate e vinte.

Requisito R9

R9) Modificare la app in modo che, una volta effettuato il login da parte del Giocatore, gli sia consentito di scegliere fra più opzioni:

<p>a) Vuoi visualizzare lo storico delle partite che hai giocato?</p> <ul style="list-style-type: none"> • In tal caso mostrare l'elenco delle partite, dove ogni partita riporta la Data, La durata complessiva della Partita, la Classe Testata, il Robot sfidato, il Vincitore (Giocatore/ Robot), la LOC coverage raggiunta dal Giocatore e quella del Robot <p>b) Vuoi giocare una Nuova Partita?</p> <ul style="list-style-type: none"> • Prevedere la possibilità di scegliere di Giocare fra due tipi di Partite: <ol style="list-style-type: none"> 1) Partita Base (già disponibile): Il sistema mostra l'elenco delle classi, dei Robot, e dei livelli disponibili (v. UI già esistente) e accetta le scelte utente 2) Partita Contro tutti i Robot (v. requisito R10): sistema mostra solo l'elenco delle classi disponibili e dei relativi Robot disponibili e accetta la scelta utente 	<p>a) Vuoi vedere la classifica totale dei giocatori?</p> <ul style="list-style-type: none"> • Mostrare Classifica dei Giocatori ordinata in base al numero di partite giocate e vinte <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>Giocatore</th><th>Partite Giocate</th><th>Partite Vinte</th></tr> </thead> <tbody> <tr> <td>Pippo</td><td>5</td><td>3</td></tr> <tr> <td>Pluto</td><td>3</td><td>1</td></tr> <tr> <td>...</td><td></td><td></td></tr> </tbody> </table>	Giocatore	Partite Giocate	Partite Vinte	Pippo	5	3	Pluto	3	1	...		
Giocatore	Partite Giocate	Partite Vinte											
Pippo	5	3											
Pluto	3	1											
...													

Figure 1.2: Requisito R9

1.3.1 Database analizzati

Per analizzare i dati correlati al requisito R9, abbiamo esplorato il database del componente T4 e del componente T23 con l'obiettivo di verificarne la validità. Questa operazione ci ha consentito di ottenere una panoramica dettagliata delle informazioni rilevanti per la richiesta di implementazione di nuove funzionalità.

Mediante l'impiego di DBgate, abbiamo accesso ai database inserendo le seguenti credenziali per il componente T23:

- **Connection type:** MySQL
- **Connection mode:** Host and port
- **Server:** localhost
- **Port:** 3306
- **Display name:** T23

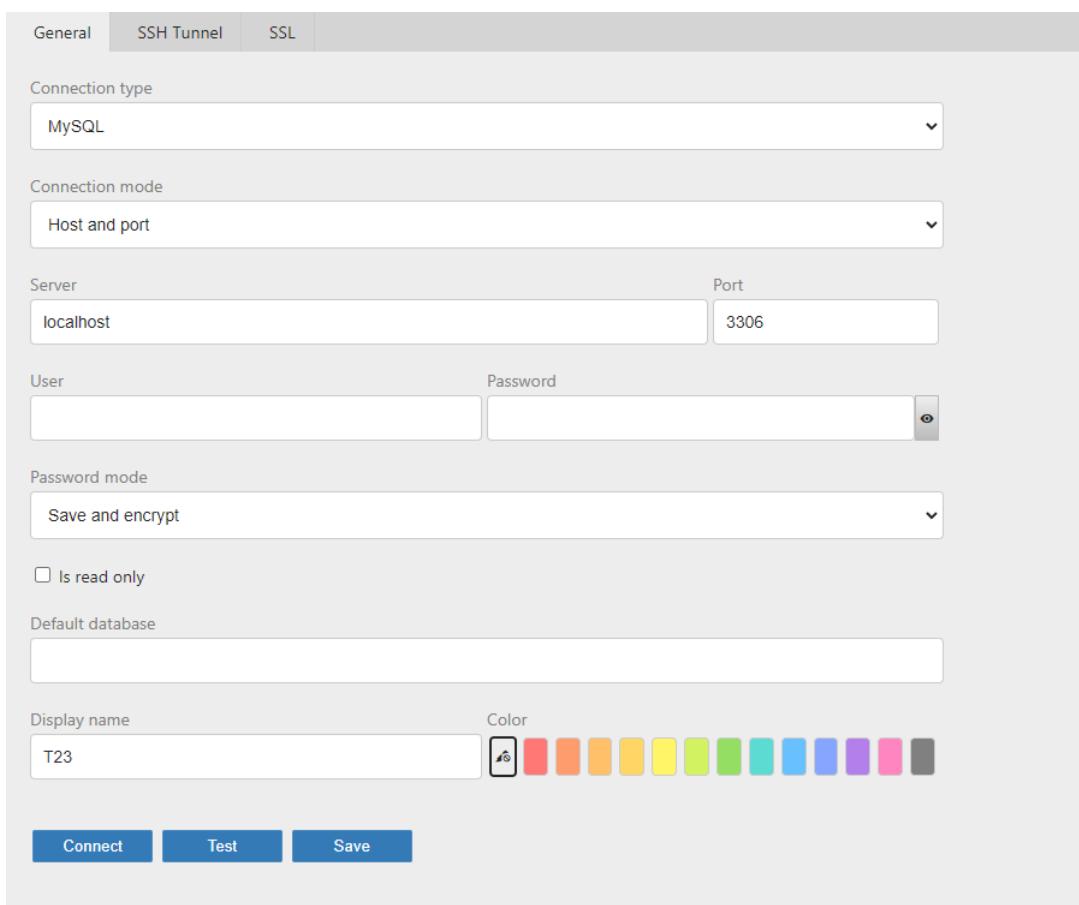


Figure 1.3: accesso al database T23

Di seguito le credenziali utilizzare per entrare nel database del componente T4:

- **Connection type:** PostgreSQL
- **Connection mode:** Host and port
- **Server:** localhost
- **Port:** 5432
- **User:** postgres
- **Password:** postgres

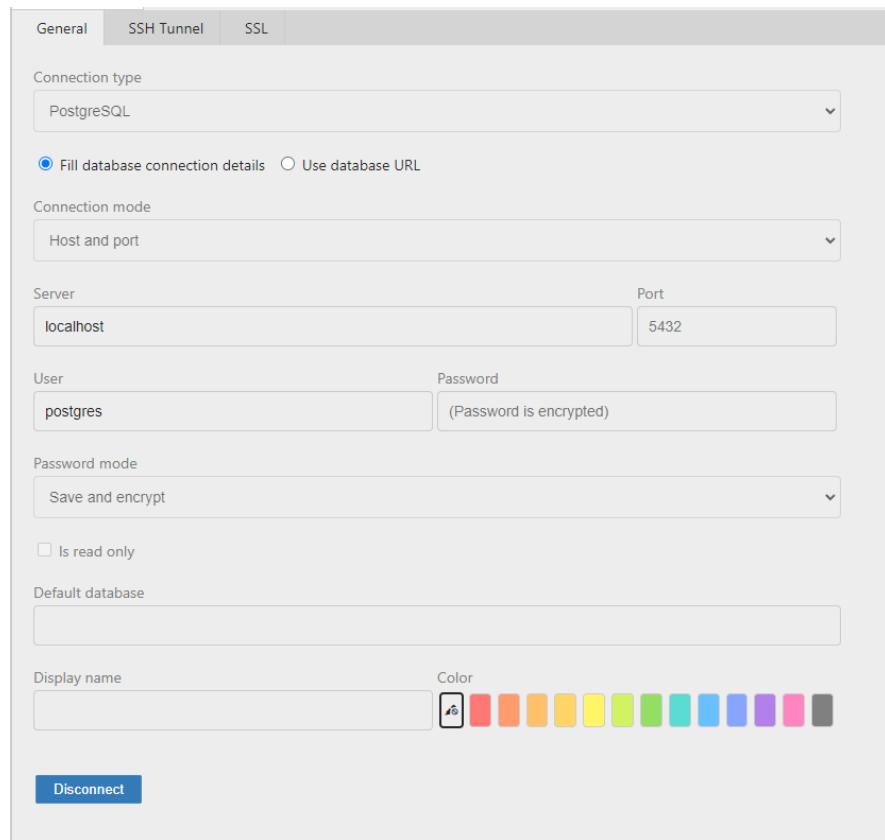


Figure 1.4: accesso al database T4

1.4 Descrizione del requisito R11

Nel presente capitolo si procede con l'analisi del Requisito R11 1.5, il quale stabilisce le specifiche per una nuova interfaccia di amministrazione del sistema, concepita per fornire all'amministratore un cruscotto dalle molteplici funzionalità di interrogazione. Questo strumento è stato ideato per consentire l'accesso a un'ampia gamma di informazioni strutturate che facilitano la gestione e il monitoraggio dell'attività degli utenti all'interno del sistema.

Requisito R11

Prevedere per l'amministratore del sistema un cruscotto che dia accesso a diverse funzionalità di interrogazione per ottenere:

- Elenco dei giocatori iscritti riportando per ognuno: Nome, Cognome, Corso di Studi a cui è iscritto, Numero di Classi testate, Numero di Partite giocate, Tempo totale delle partite giocate
- Elenco dei Giocatori iscritti, ordinato per Tipo di Corso (BSc o Master)
- Elenco delle Classi disponibili, riportando tutti i dettagli delle Classi (Nome, LOC, Numero di Metodi...)
- Una Classifica dei Giocatori ordinata in base al numero di partite giocate
- Una Classifica dei Giocatori ordinata in base al numero di partite giocate e vinte
- ...

Figure 1.5: Requisito R11

In dettaglio, il Requisito R11 si articola in diverse richieste:

- Elenco Giocatori Iscritti: Si prevede la creazione di un elenco dettagliato dei giocatori, riportando per ciascuno nome, cognome, corso di studi seguito, il numero di classi testate, il numero

totale delle partite giocate e il tempo complessivo impiegato nelle partite. Questa sezione del cruscotto è essenziale per ottenere una panoramica immediata della progressione degli utenti.

- **Ordinamento per Corso di Studio:** Si richiede inoltre che l'elenco dei giocatori possa essere ordinato in base al tipo di corso di studio - BSc o Master - offrendo così una visione differenziata e stratificata del coinvolgimento degli utenti in relazione al loro background accademico.
- **Elenco delle Classi Disponibili:** Una sezione del cruscotto dovrà elencare tutte le classi disponibili all'interno del sistema, fornendo dettagli essenziali quali il nome, il numero di Linee di Codice (LOC) e il numero di metodi associati a ciascuna classe, permettendo all'amministratore di avere un quadro completo delle risorse di gioco a disposizione.
- **Classifiche dei Giocatori:** Sono previste due diverse classifiche dei giocatori: una ordinata in base al numero di partite giocate e l'altra in base al numero di partite giocate e vinte. Questo aspetto introduce una componente competitiva all'interno del sistema.

1.4.1 Database analizzati

Per analizzare i dati correlati al requisito R11, abbiamo esplorato il database del componente T1 con l'obiettivo di verificarne la validità. Questa operazione ci ha consentito di ottenere una panoramica dettagliata delle informazioni rilevanti per la richiesta di implementazione di nuove funzionalità.

Mediante l'impiego di DBgate, abbiamo accesso ai database inserendo le seguenti credenziali per il T1:

- **Connection type:** MongoDB
- **Connection mode:** Host and port
- **Server:** localhost
- **Port:** 27017
- **efault database:** manvsclass
- **Display name:** T1

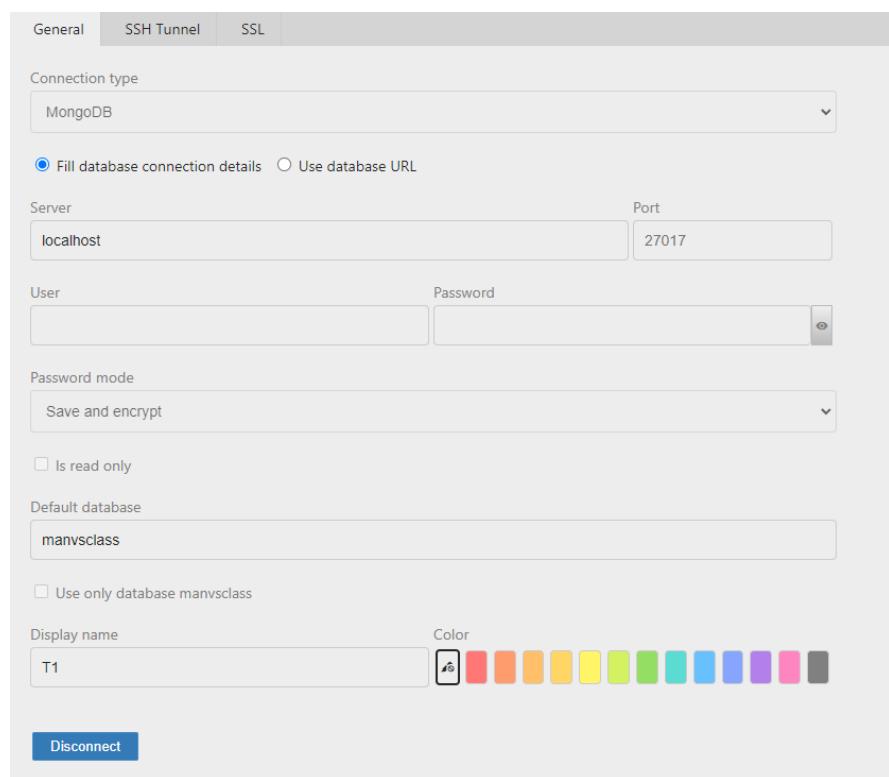


Figure 1.6: accesso al database T1

Chapter 2

Metodologie di sviluppo

Per la realizzazione del progetto proposto, abbiamo scelto di adottare il processo di sviluppo Agile. Questa decisione è stata influenzata dalla nostra esperienza acquisita durante il corso, che ci ha permesso di familiarizzare con questo nuovo approccio operativo, rendendo il nostro lavoro notevolmente più efficiente. L'approccio Agile è caratterizzato da un approccio iterativo, il cui scopo principale è consentire al team di sviluppo di organizzare le attività in modo rapido ed efficiente, rispetto a metodologie tradizionali come il modello a cascata (waterfall). Invece di seguire fasi di sviluppo lunghe e sequenziali, l'approccio Agile ci ha consentito di lavorare attraverso iterazioni, chiamate "sprint". Alla fine di ciascun sprint, vengono aggiunti incrementi al prodotto finale, introducendo nuove funzionalità di valore per il cliente. Inizialmente, abbiamo adottato un approccio tradizionale basato sull'analisi dei requisiti funzionali e non funzionali. Successivamente, abbiamo deciso di

orientarci verso l'approccio Agile. In particolare, abbiamo optato per l'uso delle metodologie SCRUM e abbiamo creato un product backlog contenente le storie utente che avremmo sviluppato successivamente in codice.

2.1 SCRUM

Scrum è un metodo Agile ampiamente utilizzato nell'ambito della gestione dei progetti, particolarmente adatto per affrontare sfide complesse e innovative. Si tratta di un framework che fornisce una guida dettagliata sulle pratiche da seguire per suddividere efficacemente il processo di gestione di un progetto. L'approccio Scrum si basa sulla suddivisione del lavoro in cicli chiamati "sprint," ognuno dei quali di solito ha una durata limitata, tipicamente compresa tra 2 e 4 settimane. Questi sprint sono fondamentali per concentrarsi sulle esigenze degli stakeholder e gestire parallelamente lo sviluppo con iterazioni frequenti. Durante ogni sprint, il team si impegna a sviluppare una parte del prodotto, consentendo così un rapido adattamento alle richieste dei clienti e ai cambiamenti nel contesto del progetto. L'approccio Scrum promuove la collaborazione, la trasparenza e l'auto-organizzazione del team, contribuendo a migliorare l'efficienza e la qualità complessiva del processo di sviluppo.

2.2 Sprint Backlog

Le riunioni organizzative si sono dimostrate estremamente utili nel processo di pianificazione degli Sprint Backlog. Durante queste sessioni, il team ha potuto definire chiaramente le attività da svolgere all'interno di ciascuno sprint e distribuire i compiti tra i membri del team in modo equo e efficiente. Inoltre, è stata possibile stabilire scadenze ben definite per il completamento di ciascun obiettivo, basandosi sulle date delle revisioni stabilite. Inizialmente, le scadenze sono state fissate ogni due settimane nel periodo da settembre a dicembre, consentendo un approccio graduale e pianificato al lavoro. Tuttavia, con l'avvicinarsi della data di presentazione del progetto, è diventato evidente l'esigenza di accelerare il ritmo di lavoro per garantire un raffinamento più approfondito degli artefatti. Di conseguenza, il lavoro all'interno del team si è intensificato in modo da garantire un prodotto valido.

2.3 Divisione ed organizzazione del lavoro

Al fine di organizzare al meglio il lavoro, abbiamo schedulato periodicamente ciascuno di questi eventi:

- Iteration Planning: è una riunione della durata di circa 1 ora svolta all'inizio di ogni iterazione. Con questa riunione si ha l'importante compito di stabilire come migliorare il proprio prodotto e, soprattutto, di attribuire la priorità alle varie storie utente e

di schedulare i vari compiti da svolgere. Questa riunione è anche tipicamente nota col nome di Sprint Planning.

- Coordination Meeting: questo tipo di riunione, invece, nasce per essere di breve durata, ovvero all'incirca 15 minuti, in modo tale da essere effettuata giornalmente col fine di sincronizzare il lavoro svolto dai vari membri del team e di aiutarsi insieme per superare eventuali ostacoli riscontrati. Essa rappresenta a tutti gli effetti quella che si può definire come “riunione di allineamento”. Questa riunione è tipicamente nota col nome di Daily Scrum;
- Sprint Review: questa riunione è generalmente di tipo informale e ha una durata che può andare dai 30 ai 60 minuti; viene svolta alla fine di un’iterazione e in essa si presenta il risultato raggiunto, magari facendo uso di una demo col fine di raccogliere i primi feedback;
- Sprint Retrospective: è una riunione della durata di circa 1 ora, anch’essa svolta alla fine di un’iterazione e viene svolta successivamente alla Sprint Review così i membri del team hanno l’opportunità di discutere su cosa sia andato bene o cosa sia andato male dopo essersi interfacciati con i clienti. L’obiettivo è quello di migliorare sempre di più la soddisfazione e di supportare, al tempo stesso, un buon rapporto lavorativo all’interno del team.

2.4 Strumenti utilizzati

Nel corso dell' implementazione il nostro gruppo ha utilizzato i seguenti strumenti per lavorare al meglio:

2.4.1 GitHub

GitHub è una piattaforma cruciale nel mondo dello sviluppo software, offrendo servizi di hosting per il controllo delle versioni dei progetti. Questa potente piattaforma consente agli utenti di collaborare in modo efficace su progetti di sviluppo, tenendo un registro delle modifiche apportate ai file e semplificando la gestione del flusso di lavoro di sviluppo attraverso un'unica interfaccia utente intuitiva. GitHub offre un'ampia gamma di integrazioni e strumenti di automazione che contribuiscono a rendere il lavoro degli sviluppatori più efficiente e permettono un processo di sviluppo più fluido e collaborativo. In sintesi, GitHub rappresenta una risorsa fondamentale per la comunità dello sviluppo software, facilitando la collaborazione e la gestione dei progetti in modo efficace. Ecco alcune delle sue funzionalità che sono state utilizzate durante lo sviluppo:

- Repository: Un repository, comunemente abbreviato come "repo" è un contenitore in cui vengono archiviati tutti i file di un progetto, compresi il codice sorgente, la documentazione e altri file di supporto. I repository possono essere sia pubblici che privati,

a seconda delle esigenze e della visibilità desiderata.

- Branch: I branch rappresentano le "diramazioni" del codice sorgente all'interno di un repository. Consentono agli sviluppatori di lavorare su nuove funzionalità o correzioni di bug senza influire sul ramo principale del progetto. Questo approccio facilita la gestione delle modifiche parallele e la separazione tra il lavoro in corso e il codice stabile.
- Pull Request: Una pull request è una proposta di modifica al codice sorgente all'interno di un repository. Gli sviluppatori possono aprire una pull request per iniziare una discussione e permettere agli altri membri del team di esaminare, commentare e valutare le modifiche proposte. Questo processo di revisione consente di garantire la qualità del codice prima che venga unito al ramo principale del progetto. Le pull request sono uno strumento essenziale per il controllo di qualità e la collaborazione efficace.
- Collaborazione: GitHub fornisce una serie di strumenti per agevolare la collaborazione tra i membri del team di sviluppo. Gli sviluppatori possono commentare direttamente sul codice, partecipare a discussioni sulle issue, e collaborare in modo efficiente grazie a funzionalità come notifiche e menzioni. Questi strumenti aiutano a mantenere una comunicazione chiara all'interno

del team e a garantire che tutti siano coinvolti e informati sullo stato del progetto.

2.4.2 Visual Paradigm

Un elemento di estrema rilevanza nel processo di scrittura del nostro progetto è stato l'utilizzo di Visual Paradigm, un'applicazione che ha svolto un ruolo fondamentale nel consentirci di visualizzare e comprendere i diagrammi UML creati dai nostri colleghi in precedenza. Naturalmente, abbiamo sfruttato questo strumento anche per generare nuovi diagrammi e apportare modifiche a quelli preesistenti, dove necessario. Questi diagrammi sono presentati nel presente documento. Visual Paradigm rappresenta una suite di strumenti progettati per fornire supporto in tutte le fasi del ciclo di sviluppo del software, comprendendo dalla modellazione e progettazione all'implementazione e alla manutenzione. La sua ampia diffusione nell'ambito dell'ingegneria del software lo rende una risorsa cruciale, offrendo una vasta gamma di funzionalità per aiutare gli sviluppatori a concepire, documentare e gestire progetti software di complessità variabile.

Le principali funzionalità offerte da Visual Paradigm sono la modellazione visuale e i diagrammi UML. Infatti questo strumento mette a disposizione gli strumenti necessari per creare modelli visivi, impiegando notazioni standard come UML (Unified Modeling Language). Ciò permette di rappresentare in modo chiaro e comprensibile i con-

cetti e le relazioni presenti all'interno del sistema software. Inoltre Visual Paradigm supporta un'ampia gamma di diagrammi UML, quali i diagrammi dei casi d'uso, diagrammi delle classi, diagrammi delle sequenze, diagrammi delle attività, e molti altri. Questi diagrammi risultano fondamentali per descrivere e visualizzare aspetti specifici di un sistema software, offrendo una rappresentazione visuale chiara e concisa.

2.4.3 Microsoft Teams

L'applicazione Microsoft Teams ha permesso al team sia di comunicare nelle fasi di sviluppo degli artefatti tramite videochiamate di gruppo, sia di utilizzare la chat del canale appositamente creato per lo scambio rapido di file e messaggi.

2.4.4 Trello

Trello è un'applicazione di gestione dei progetti che si distingue per la sua semplicità ed efficacia. Con Trello, gli utenti possono organizzare le attività in modo intuitivo attraverso schede, liste e bacheche. Ogni scheda rappresenta una specifica attività o compito all'interno del progetto, mentre le liste permettono di suddividere il lavoro in fasi o categorie. Le bacheche fungono da contenitori che ospitano schede e liste, offrendo una panoramica completa del progetto.

Una delle caratteristiche chiave di Trello è la sua altissima person-

alizzazione. Gli utenti possono aggiungere etichette alle schede per categorizzarle, impostare scadenze per tenere traccia dei tempi e assegnare compiti ai membri del team. Inoltre, è possibile commentare le schede per comunicare e condividere informazioni importanti in modo chiaro e trasparente con i collaboratori.

Trello è una soluzione flessibile che può essere utilizzata in una varietà di contesti, dal project management all'organizzazione di attività personali. La sua interfaccia intuitiva e le numerose funzionalità rendono Trello una scelta popolare per coloro che cercano un modo efficace per gestire progetti e attività.

2.4.5 Notion

Per la creazione e la gestione dello Sprint Backlog, è stato impiegato con successo anche il software Notion 2.1. Questo strumento si è dimostrato estremamente efficace nella pianificazione delle riunioni, nella segnalazione degli aggiornamenti sul progresso dello sviluppo dei task e nell'offrire una repository condivisa per l'archiviazione di tutti i tipi di documenti relativi al progetto.

Grazie a Notion, il team ha potuto organizzare le attività in modo strutturato, garantendo una comunicazione chiara e un accesso agevole a tutte le risorse necessarie per il progetto. La versatilità e l'interfaccia intuitiva di Notion hanno reso la gestione dello Sprint Backlog un processo efficiente e collaborativo, contribuendo in modo significativo al

successo complessivo del progetto.

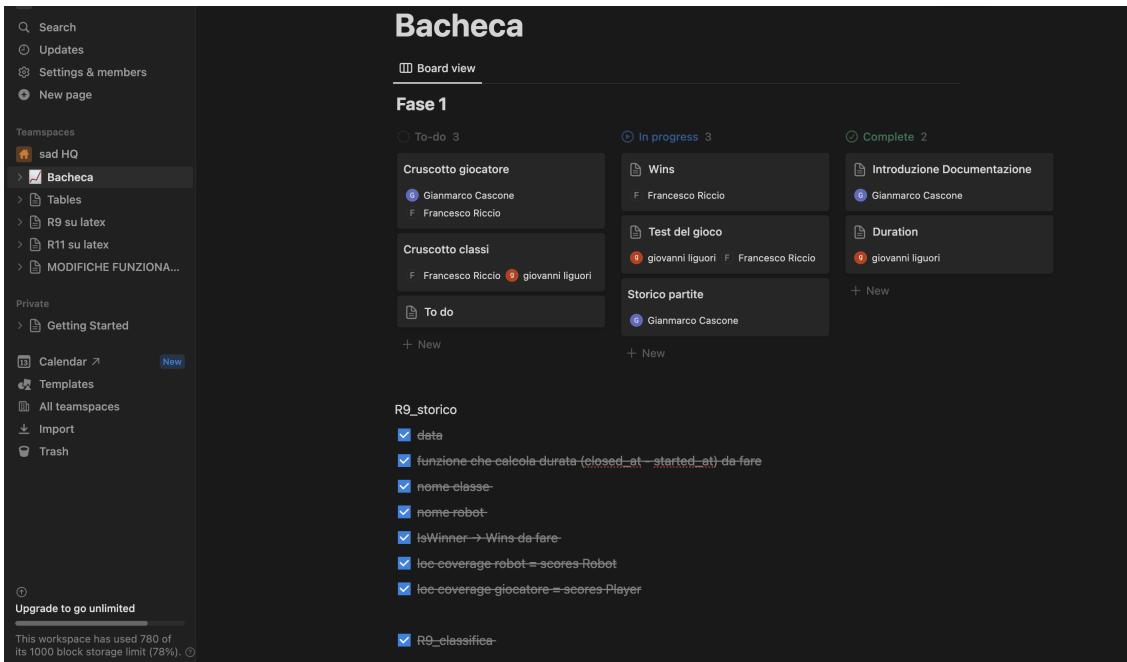


Figure 2.1: Screenshot di organizzazione notion

2.4.6 Visual Studio Code

Visual Studio Code, comunemente noto come VS Code, si presenta come un editor di codice sorgente e un ambiente di sviluppo che gode di una vasta popolarità tra gli sviluppatori. Le sue qualità principali includono:

Visual Studio Code è noto per la sua leggerezza e velocità, garantendo un'esperienza fluida agli utenti indipendentemente dall'hardware in uso.

La sua estensibilità rappresenta un punto forte: supporta numerosi linguaggi di programmazione e permette di personalizzare l'editor at-

traverso l'installazione di estensioni dal Marketplace di VS Code. Questa flessibilità consente agli sviluppatori di adattare l'ambiente alle loro specifiche esigenze e di integrare il supporto per vari framework e tecnologie.

Oltre ad essere un editor di testo avanzato, VS Code agisce come un ambiente di sviluppo integrato (IDE), dotato di strumenti essenziali. Tra questi troviamo un terminale integrato, un debugger per il debugging in tempo reale e una stretta integrazione con Git, semplificando notevolmente il controllo di versione e migliorando la collaborazione tra membri del team.

2.4.7 Java Spring

Java Spring e Spring Boot hanno svolto un ruolo fondamentale nel contesto di un progetto specifico, fornendo un solido fondamento per lo sviluppo di applicazioni enterprise di successo. Questi due framework hanno contribuito in modo significativo alla realizzazione del progetto, consentendo agli sviluppatori di affrontare sfide complesse e di raggiungere gli obiettivi prefissati.

Nel corso del progetto, Java Spring ha dimostrato la sua affidabilità e versatilità. Grazie a Spring, è stato possibile adottare pattern di progettazione consolidati come il Model-View-Controller (MVC) per lo sviluppo di applicazioni web ben strutturate. L'integrazione con tecnologie chiave come JDBC (Java Database Connectivity) e JMS

(Java Message Service) ha semplificato notevolmente l’interazione con database, middleware e altri servizi. Inoltre, la progettazione orientata all’interfaccia e l’uso di oggetti mock per il testing hanno reso possibile una rigorosa verifica delle funzionalità delle applicazioni, garantendo la loro qualità e affidabilità.

2.4.8 Spring Boot

Spring Boot, in particolare, ha giocato un ruolo chiave nell’accelerare lo sviluppo del progetto. La sua auto-configurazione ha ridotto notevolmente la quantità di configurazione manuale richiesta agli sviluppatori, consentendo loro di concentrarsi sulla logica di business dell’applicazione. L’inclusione di server incorporati come Tomcat e la disponibilità di modelli preconfigurati per diversi tipi di progetti hanno semplificato l’avvio di nuovi progetti e la gestione delle dipendenze del progetto. Inoltre, la comunità di sviluppatori attiva e la vasta documentazione disponibile online hanno fornito agli sviluppatori risorse essenziali per affrontare le sfide specifiche del progetto. L’abilità di integrare facilmente altre tecnologie e librerie ha permesso di adattare le applicazioni alle esigenze specifiche del contesto del progetto, garantendo la loro efficacia e successo.

In definitiva, Java Spring e Spring Boot hanno svolto un ruolo cruciale nel progetto, offrendo strumenti potenti e versatili per la creazione, la configurazione e la distribuzione di applicazioni di successo. La com-

binazione di queste due piattaforme ha contribuito a raggiungere gli obiettivi del progetto in modo efficiente ed efficace.

2.4.9 Maven

Maven è una potente e ampiamente utilizzata strumento di gestione delle dipendenze e di build per la creazione e la gestione dei progetti software in ambiente Java. La sua versatilità e le numerose funzionalità offerte lo rendono uno strumento essenziale per gli sviluppatori Java.

Una delle principali funzionalità di Maven è la gestione delle dipendenze. Maven semplifica notevolmente il processo di gestione delle librerie e delle dipendenze del progetto, consentendo agli sviluppatori di dichiarare le dipendenze nei file di configurazione del progetto (come il file "pom.xml") e successivamente scaricando automaticamente le librerie necessarie da repository centralizzati, come il repository Maven Central. Questo approccio assicura che il progetto abbia sempre accesso alle versioni corrette delle librerie di terze parti, migliorando la stabilità e la coerenza del progetto.

Inoltre, Maven offre funzionalità di build automatizzate. Gli sviluppatori possono definire facilmente le fasi di build, i goal e i plugin desiderati nel file di configurazione "pom.xml", consentendo l'esecuzione di compiti di build complessi con un singolo comando. Questa capacità semplifica l'automazione delle attività di compilazione, test, pacchet-

tizzazione e distribuzione del progetto.

Maven favorisce anche la standardizzazione dei progetti Java attraverso la sua struttura di directory predefinita. Questa struttura standardizzata facilita la navigazione e la gestione del codice sorgente, migliorando la comprensione e la manutenzione del progetto. Inoltre, Maven supporta la generazione automatica della documentazione, il che contribuisce a mantenere la documentazione aggiornata e coerente con il codice.

2.4.10 Go

Il linguaggio di programmazione Go, spesso abbreviato in Golang, è stato impiegato con successo per l'implementazione di specifici task all'interno del progetto. Go è noto per la sua efficienza, la sua semplicità e la sua velocità di esecuzione, rendendolo una scelta ideale per una serie di compiti specifici all'interno del contesto del progetto.

- **Efficienza e Concorrenza:** Una delle caratteristiche distintive di Go è la sua gestione avanzata della concorrenza. Il linguaggio offre supporto nativo per le goroutine, che sono unità leggere di esecuzione. Queste goroutine semplificano la scrittura di codice concorrente e parallelo, consentendo l'elaborazione di compiti in modo efficace e scalabile. Ciò può essere particolarmente utile per compiti come l'elaborazione di dati in parallelo o la gestione di richieste concorrenti.

- **Velocità di Esecuzione:** Go è noto per la sua velocità di esecuzione, che lo rende adatto per task che richiedono prestazioni elevate. Grazie al suo runtime altamente ottimizzato e alla sua compilazione efficiente, Go può essere utilizzato per sviluppare applicazioni reattive e performanti.
- **Facilità di Deploy:** Go compila il codice in un singolo binario eseguibile, che semplifica notevolmente la distribuzione delle applicazioni. Questo è particolarmente utile per task come la creazione di microservizi o applicazioni containerizzate, in cui la facilità di deploy è cruciale.

2.4.11 Postman

Postman è uno strumento essenziale nel processo di sviluppo, test e documentazione delle API. Questa piattaforma offre una serie di funzionalità fondamentali che semplificano notevolmente il lavoro degli sviluppatori e dei team responsabili delle API, eccone alcune:

- **Creazione e Test di Richieste API:** Postman consente agli sviluppatori di creare facilmente richieste HTTP o HTTPS per interagire con API e servizi web. Questo include la possibilità di specificare i metodi HTTP (come GET, POST, PUT, DELETE), gli header, i parametri, i dati di input e molto altro. Gli sviluppatori possono inviare queste richieste e ricevere risposte direttamente dall'interfaccia di Postman per testarne il funzionamento.

- Automazione dei Test: Postman offre un ambiente per la scrittura e l'esecuzione di test automatizzati sulle API. Gli sviluppatori possono definire asserzioni e verifiche per verificare che le risposte delle API siano conformi alle specifiche previste. Questi test automatizzati possono essere eseguiti in modo regolare per garantire che le API rimangano coerenti e senza errori nel tempo.
- Raccolta di Richieste: Postman consente di organizzare e gestire gruppi di richieste in raccolte. Questo rende più agevole l'organizzazione delle richieste relative a uno specifico progetto o API. Le raccolte possono essere condivise tra i membri del team per garantire la coerenza nell'utilizzo delle API.
- Collaborazione: Postman offre funzionalità di collaborazione che permettono ai membri del team di lavorare insieme su collezioni di richieste e test. Gli sviluppatori possono condividere le proprie raccolte, collaborare su scenari di test e condividere l'accesso alle API.
- Generazione di Documentazione: Postman consente di generare automaticamente documentazione dettagliata delle API basata sulle raccolte di richieste. Questo semplifica la creazione di documentazione chiara e completa per le API, facilitando la comprensione e l'adozione da parte di altri sviluppatori o utenti.
- Variabili e Ambienti: Postman permette di definire variabili e

ambienti, consentendo una maggiore flessibilità nella configurazione delle richieste. Questo è particolarmente utile per gestire diverse configurazioni per ambienti di sviluppo, test e produzione.

- Monitoraggio delle API: Postman offre funzionalità avanzate di monitoraggio delle API che consentono di tracciare le prestazioni e il comportamento delle API in produzione, identificando eventuali problemi o rallentamenti.

2.4.12 HTTPie

HTTPie è un potente strumento per la progettazione e il testing di API che offre una serie di funzionalità chiave per semplificare il processo di sviluppo e debug delle API. Ecco alcune delle sue principali funzionalità:

- Sintassi Leggibile: HTTPie utilizza una sintassi leggibile e intuitiva per creare e inviare richieste HTTP alle API. Gli sviluppatori possono specificare facilmente i metodi HTTP, gli header, i parametri e i dati di input utilizzando un linguaggio naturale e conciso.
- Output Formattato: HTTPie offre un'ampia gamma di opzioni per formattare e visualizzare le risposte delle API in modo leggibile. Gli utenti possono selezionare tra diversi formati di output,

tra cui JSON, XML e HTML, per facilitare l'analisi delle risposte delle API.

- Gestione delle Sessioni: HTTPie permette di gestire facilmente le sessioni per mantenere lo stato tra le richieste. Questo è utile per l'autenticazione e l'interazione con API che richiedono la conservazione dello stato.
- Completamento Automatico: HTTPie offre il completamento automatico dei comandi e delle opzioni, facilitando la creazione di richieste API senza errori. Gli sviluppatori possono risparmiare tempo e minimizzare gli errori di sintassi.
- Scripting e Automazione: HTTPie può essere utilizzato per scrivere script e automatizzare il testing delle API. Questa funzionalità è utile per eseguire test automatizzati e integrazione continua.
- Estensibilità: HTTPie è altamente estensibile e consente agli sviluppatori di creare plugin personalizzati per estendere le funzionalità base del tool, adattandolo alle esigenze specifiche del progetto.

2.4.13 DBgate

DBGate è uno strumento moderno e versatile per il lavoro con i database, progettato per soddisfare le esigenze degli sviluppatori, degli analisti

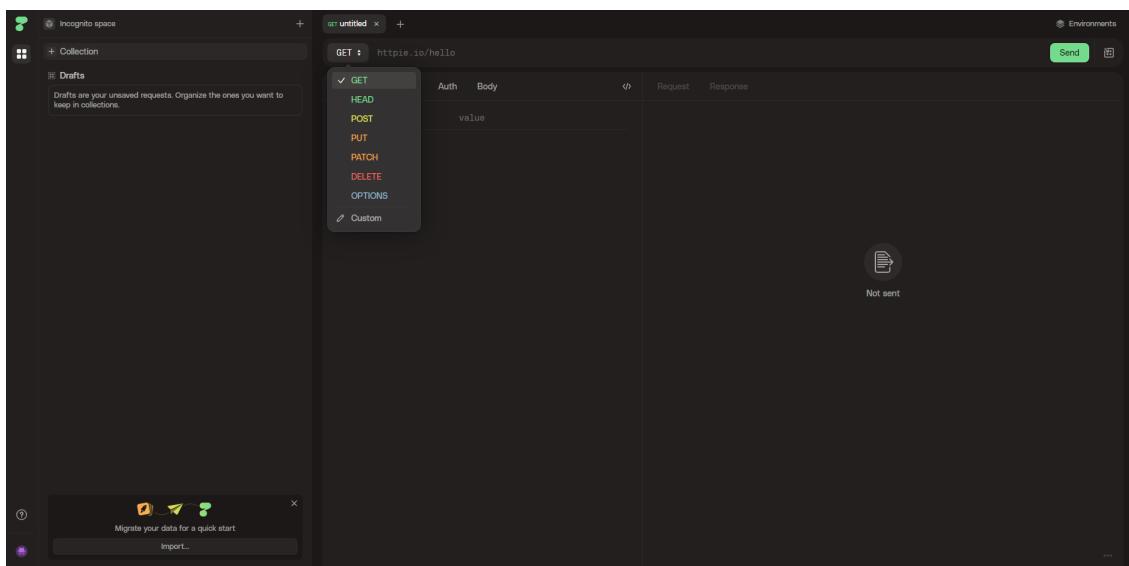


Figure 2.2: Screenshot Httpie

di dati e degli amministratori di database. Ecco alcune delle sue principali caratteristiche e funzionalità:

- Interfaccia Utente Intuitiva: DBGate offre un’interfaccia utente pulita e facile da usare. Questo rende semplice la navigazione tra diverse funzioni e database, consentendo agli utenti di lavorare in modo efficiente senza una ripida curva di apprendimento.
- Supporto Multi-Database: Una delle principali caratteristiche di DBGate è il suo supporto per una vasta gamma di sistemi di gestione dei database (DBMS), come MySQL, PostgreSQL, SQL Server, MongoDB e molti altri. Ciò consente agli utenti di lavorare con diversi database utilizzando un unico strumento.
- Query Editor Avanzato: DBGate fornisce un editor di query potente e flessibile con funzionalità di completamento automatico,

evidenziazione della sintassi e formattazione del codice. Questo aiuta gli utenti a scrivere e ottimizzare le query SQL in modo più efficiente.

- Visualizzazione e Gestione dei Dati: Gli utenti possono visualizzare, inserire, aggiornare e cancellare facilmente i dati direttamente dall'interfaccia. DBGate offre anche funzionalità avanzate di filtro e ordinamento per gestire grandi quantità di dati.
- Progettazione del Database: DBGate include strumenti per la progettazione visiva del database, che consentono agli utenti di creare e modificare tabelle, relazioni e altri oggetti del database in modo intuitivo, senza scrivere codice SQL.

Chapter 3

Stato iniziale del progetto

Per comprendere a pieno il nostro lavoro, è essenziale prestare attenzione all'eredità ricevuta dai colleghi che hanno inizialmente lavorato su questo progetto. Prima di addentrarci nell'analisi dettagliata del progetto stesso, dobbiamo focalizzarci su un aspetto fondamentale: l'installazione del programma. È stato un processo piuttosto problematico fin dall'inizio, poiché abbiamo notato fin da subito che le guide disponibili erano insufficienti. Questo ha portato a notevoli difficoltà nella fase iniziale, ma fortunatamente siamo riusciti a superarle con successo.

Per soddisfare il nostro requisito abbiamo esteso il nostro studio a tutti i database del gioco e non solo al database T4, che e' stato pero' comunque quello su cui abbiamo lavorato maggiormente. In seguito analizzeremo la sua struttura.

3.1 Architettura del sistema

La tipologia di architettura del gioco e' basata su microservizi. Le architetture basate su microservizi rappresentano una metodologia di sviluppo software che segmenta un'applicazione in componenti più piccoli e autonomi. Ogni microservizio funziona in modo indipendente, gestendo determinate funzioni dell'applicazione e interagendo con gli altri componenti tramite API ben definite. Questo approccio facilita sia la scalabilità che la manutenzione del sistema, offrendo numerosi vantaggi.

Tra i benefici principali, le architetture a microservizi incrementano la flessibilità, consentendo lo sviluppo, il deployment e la scalabilità di ciascun servizio in modo indipendente. Questa caratteristica supporta un'efficace gestione e aggiornamento del sistema, permettendo lo sviluppo simultaneo di diverse funzionalità. Inoltre, queste architetture migliorano la resilienza dell'applicazione, poiché il malfunzionamento di un singolo microservizio non compromette l'intero sistema. L'adozione di microservizi permette inoltre la diversificazione delle tecnologie utilizzate, consentendo di selezionare linguaggi e tecnologie ottimali per ogni servizio. Questo aspetto promuove l'integrazione di nuove soluzioni tecnologiche senza impattare sull'intera architettura. Nonostante i vantaggi, la gestione di architetture a microservizi richiede una cura particolare per quanto riguarda la comunicazione inter-servizio e la sicurezza. Tuttavia, la loro capacità di offrire flessibilità e scal-

abilità rende le architetture a microservizi un'opzione valida per lo sviluppo di applicazioni dinamiche e complesse.

L'architettura adottata per il sistema si basa sull'utilizzo del Gateway Pattern. Questo modello si caratterizza per l'introduzione di componenti che fungono da unici punti di ingresso verso i microservizi. Questa struttura permette di centralizzare funzionalità critiche come autenticazione, autorizzazione, instradamento delle richieste, bilanciamento del carico e aggregazione delle API, operando in maniera simile a un facade, che rende l'interazione con le API più flessibile ed efficiente. Per quanto riguarda l'API Gateway, esso è stato sviluppato impiegando Spring Boot in combinazione con Netflix Zuul, operando come un reverse proxy. Questo componente si incarica di dirigere il traffico, gestendo le questioni di autenticazione e autorizzazione associate alle REST API. Ciò ha consentito di centralizzare l'accesso alle API sotto il percorso URL "/api/", conferendo uniformità all'interfaccia di programmazione.

Per il UI Gateway, si è scelto di utilizzare Nginx, creando anch'esso un reverse proxy. La presenza di questo gateway è cruciale per la struttura del sistema: esso unifica le richieste sul porto 80, che è lo standard per il protocollo HTTP, e si pone in ascolto. Questo non solo semplifica la gestione delle richieste in arrivo, dirigendole ai vari microservizi, ma migliora anche la scalabilità e la distribuzione del sistema. In questo modo, il gateway fornisce un'interfaccia unificata verso l'esterno, raf-

forzando la sicurezza e la capacità di espansione dell'architettura.

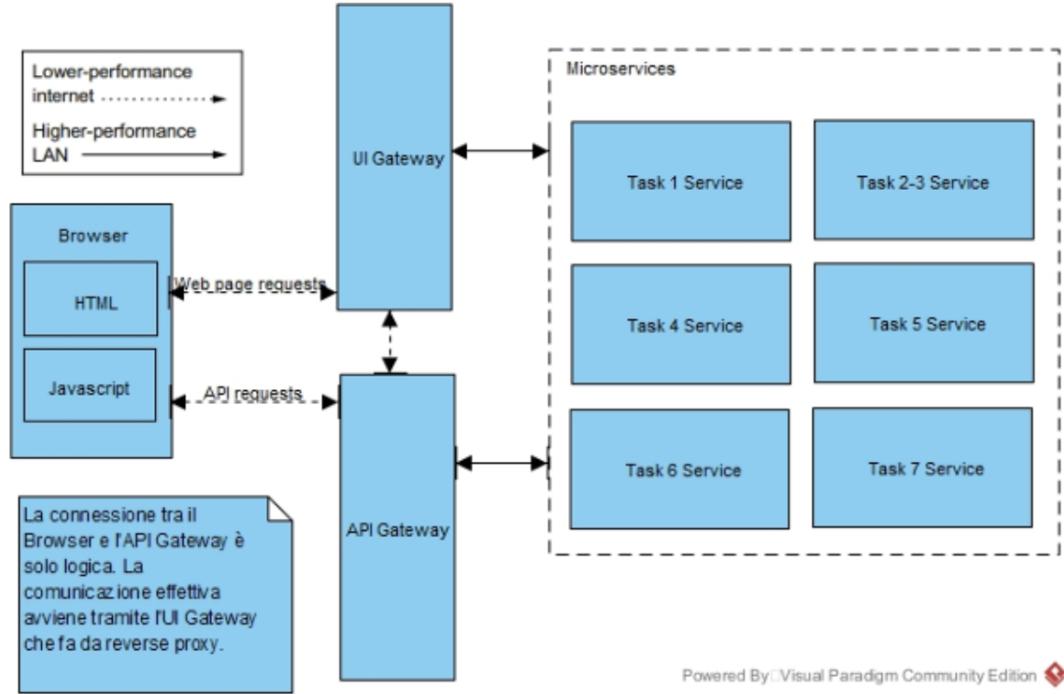


Figure 3.1: Architettura del sistema

3.1.1 Workflow

Per comprendere a pieno il funzionamento dell'applicazione, analizzeremo il workflow diagram di alto livello.

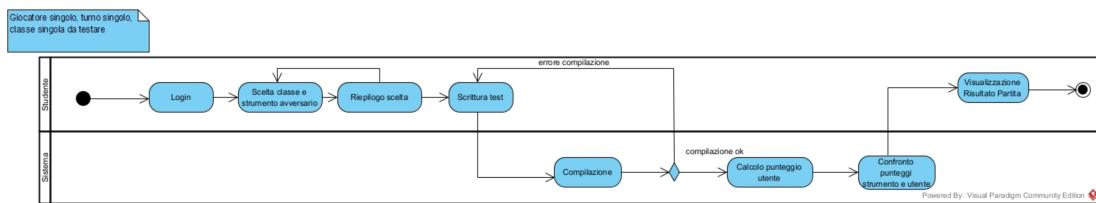


Figure 3.2: Workflow diagram

Il flusso di lavoro del gioco si innesca con il login del giocatore, il quale

poi seleziona la classe da testare e il robot contro cui competere. Confermata la scelta, il giocatore procede a scrivere i suoi test nell'editor dedicato. Il passo successivo è la compilazione del test, seguita da un controllo dei risultati e dal calcolo del punteggio dell'utente.

Per determinare il punteggio finale, vengono utilizzati gli strumenti Jacoco ed EvoSuite. Dopo, il sistema mette a confronto i punteggi ottenuti sia dal giocatore che dall'avversario robot nel turno corrente e li visualizza sullo schermo.

In base al risultato, se il giocatore vince, il gioco si conclude e gli viene assegnato il punteggio. In caso di sconfitta, però, il giocatore ha la possibilità di continuare a giocare fino a conseguire una vittoria. Quindi, l'applicazione non prevede un'uscita definitiva per sconfitta in una partita completa, ma gestisce solo la sconfitta in un singolo turno. Il caso di studio attuale è incentrato sull'implementazione di un unico scenario di gioco. In questo scenario, come mostrato nel context diagram che abbiamo ereditato dai colleghi che hanno implementato le versioni precedenti, un giocatore può ingaggiare una partita contro un avversario autonomo, che può essere Randoop o EvoSuite, e durante la partita ha l'opportunità di mettere alla prova una classe di sua scelta.

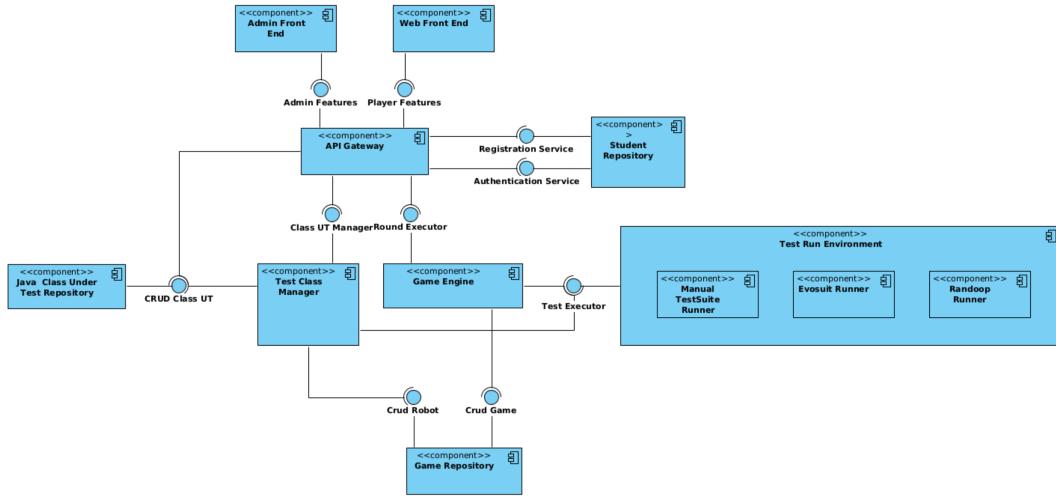


Figure 3.3: Context diagram

3.2 Stato del task T4

3.2.1 Architettura Game Repository

L’architettura del componente è ispirata al modello Domain Driven Design (DDD) in cui ogni entità risulta indipendente dalle altre. Come mostrato in Figura 3.4, ogni elemento logico dell’applicazione (individuato nel diagramma in Figura 3.6), ha a sua disposizione un Controller che utilizza un Service. Questi due componenti (realizzati per ogni entità) interagiscono con un ORM che implementa lo schema relazionale sul database.

Si noti che le fasce orizzontali rosa sono tra loro indipendenti e che l’implementazione delle classi Service sono astratte da un’interfaccia. Questa scelta, non solo garantisce un basso accoppiamento tra le entità, ma ne favorisce anche la testabilità in quanto fornendo un’imple-

entazione mock del service è possibile testare indipendentemente le classi Controller e Service. Lo stile Domain Driven è un'estensione concettuale di quello esagonale e può essere utilizzato per implementare microservizi secondo il principio SOLID. I domini applicativi rappresentati in Figura 3.4 sono stati raffinati nel System Domain Model Figura 3.5. Si noti che ciascuna entità (in verde) implementa sia il pattern DTO che Dependency Injection e che inoltre, sono tra loro indipendenti; invece le relazioni della base dati sono rappresentate nel package Model (in rosa).

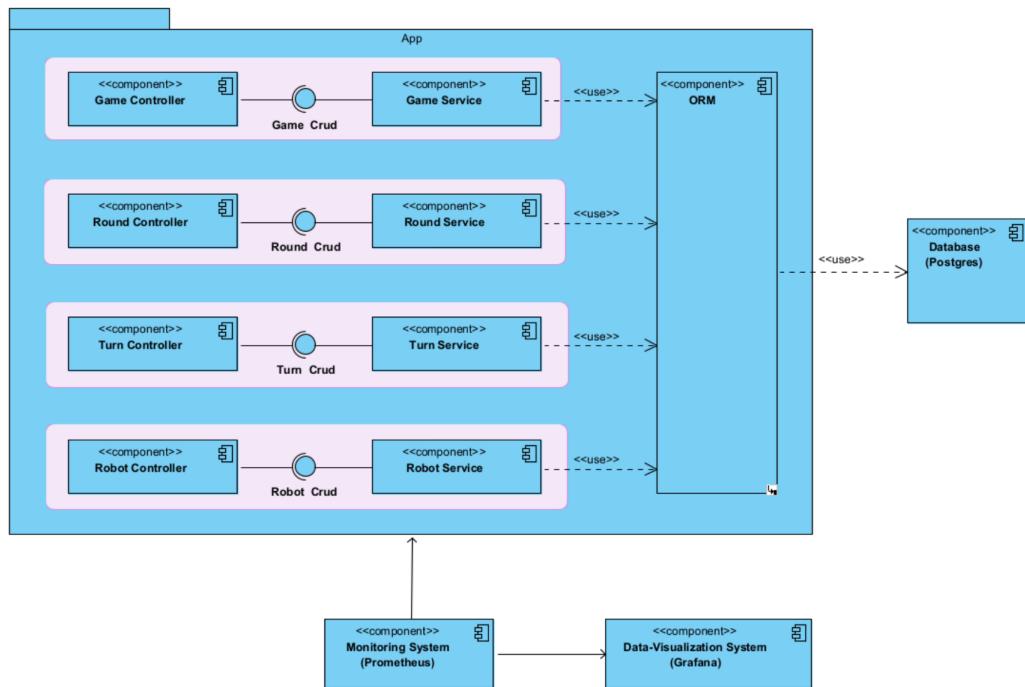


Figure 3.4: Architettura Game Repository

Il componente T4 offre un'API REST progettata per gestire infor-

mazioni relative a giocatori, robot e partite. In particolare questo è composto da 4 componenti:

1. Rest Server: Questo è il componente principale dell'applicazione ed è stato sviluppato utilizzando il linguaggio di programmazione Go. Implementa l'API necessaria per gestire le operazioni dell'applicazione.
2. Postgres: Si tratta di un database relazionale fondamentale per il funzionamento dell'applicazione. Conserva i dati e fornisce una struttura organizzata per l'accesso e la gestione delle informazioni.
3. Prometheus: Prometheus è utilizzato per estrarre metriche dal Rest Server attraverso un sistema di monitoraggio periodico. Questo componente aiuta a monitorare le prestazioni dell'applicazione e a raccogliere dati statistici rilevanti.
4. Grafana: Grafana viene utilizzato per elaborare le metriche estratte da Prometheus e creare dashboard grafiche. Queste dashboard consentono la visualizzazione delle performance dell'applicazione in modo intuitivo e accessibile.

In figura 3.5 possiamo osservare il system domain model relativo al T4. Il sistema è diviso in quattro aree principali, ognuna con i propri componenti DTO (Data Transfer Object), controllori, servizi e repository:

CHAPTER 3. STATO INIZIALE DEL PROGETTO

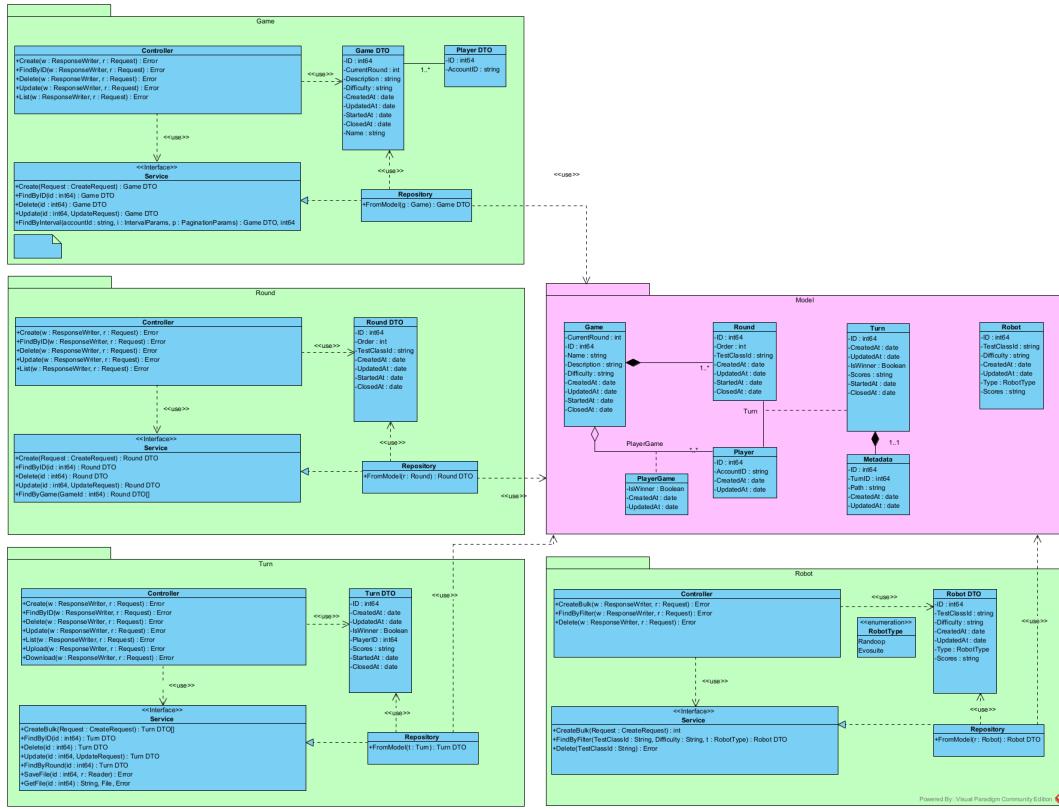


Figure 3.5: System Domain Model

- **Game:** Quest'area gestisce le informazioni relative al gioco stesso, include le operazioni CRUD (Create, Read, Update, Delete) sui giochi. I DTO includono campi come ID, nome, descrizione, e date di creazione e chiusura. Il servizio corrispondente comunica con un repository per salvare i dati dei giochi.
- **Round:** Simile all'area Game, gestisce i round all'interno di un gioco. I DTO per un round includono ID, descrizione e date di creazione e chiusura. Anche in questo caso, esiste un servizio che gestisce le interazioni con il repository dei round.
- **Turn:** Questa sezione si occupa della gestione dei turni individ-

uali all'interno di un round. I DTO del turno hanno campi per l'ID, il giocatore, se il turno è stato saltato, e date di creazione e chiusura. Vi è anche un servizio che gestisce i turni e interagisce con il relativo repository.

- Robot: L'ultima sezione gestisce i robot, che possono essere entità che giocano automaticamente o assistono i giocatori. Il DTO del robot include un numero identificativo, una difficoltà, e informazioni sulla creazione e l'update. Anche qui, esiste un servizio dedicato che opera con un repository di robot.

L'intero sistema sembra costruito seguendo il principio della separazione delle responsabilità, dove ogni controller riceve richieste, le passa ai servizi per la logica di business, e poi i servizi interagiscono con i repository per la persistenza dei dati. Questo design promuove la manutenibilità e la scalabilità, consentendo ai vari componenti di essere modificati indipendentemente dagli altri.

In figura 3.6 possiamo osservare il diagramma ER (Entity-Relationship) che mostra come sono correlate tra loro le entità Robot, Player, Metadata, Turn, Game, e Round, e la relazione tra Giocatore e Gioco attraverso una tabella di associazione Game_Player.

- Robot: Quest'entità include attributi come ID, TestClassId, Difficoltà, Date di creazione e aggiornamento, Tipo e Punteggi.
- Player: Quest'entità rappresenta i giocatori con attributi per ID,

CHAPTER 3. STATO INIZIALE DEL PROGETTO

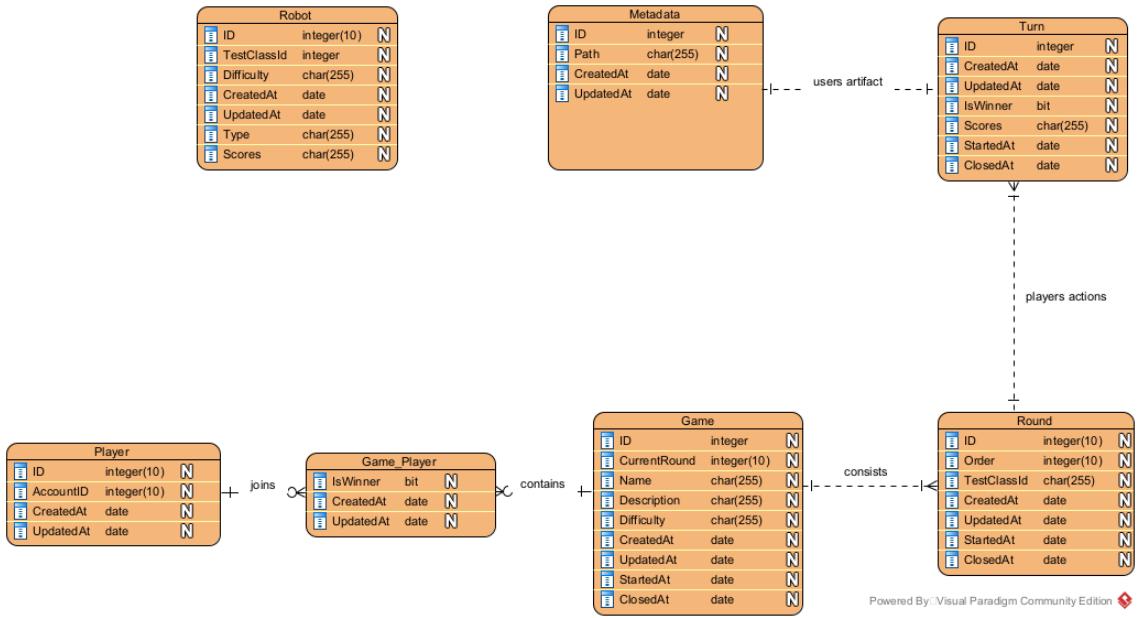


Figure 3.6: Diagramma ER

AccountId, e le date di creazione e aggiornamento.

- **Metadata**: Un'entità ausiliaria che tiene traccia di informazioni aggiuntive, relative ai turni, con attributi come Path e Date di creazione e aggiornamento.
- **Turn**: Quest'entità tiene traccia dei singoli turni di gioco con attributi per ID, Nome, Date di creazione e aggiornamento, se il giocatore è il vincitore, Punteggi, e Date di inizio e chiusura.
- **Game**: Rappresenta l'entità di gioco con attributi come ID, Round corrente, Nome, Descrizione, Difficoltà, e Date di creazione, inizio e chiusura.
- **Round**: Quest'entità rappresenta i round individuali all'interno di un gioco, con attributi per ID, Ordine, TestClassId, e Date di

creazione, aggiornamento, inizio e chiusura.

- Game_Player: Una tabella di associazione che collega i giocatori ai giochi, indicando se il giocatore è il vincitore e con attributi per le date di creazione e aggiornamento.

Ogni entità ha indicazioni sulla necessità (N) dei campi, suggerendo quali attributi sono obbligatori. Le relazioni tra le entità mostrano come i turni si basano su Game e Round (indicati con linee tratteggiate), mentre Game_Player crea una relazione multi-a-molti tra Player e Game, permettendo di registrare quali giocatori partecipano a quali giochi e se hanno vinto. Questo schema ER è tipico di applicazioni di gioco in cui si deve tracciare in dettaglio lo stato e l'evoluzione di ogni partita e turno.

3.2.2 Swagger

Nel corso dello scorso anno, il team ha effettuato un'integrazione mirata dell'interfaccia Swagger all'interno dell'applicativo, con l'obiettivo di facilitare la corretta integrazione con altri moduli dell'architettura complessiva. L'adozione di Swagger è stata una scelta strategica per documentare in modo chiaro e accessibile i servizi forniti dal componente sviluppato, aderendo ai principi della specifica OpenAPI. Questa specifica rappresenta un punto di riferimento nell'ambito dell'integrazione di componenti disparati che interagiscono attraverso API HTTP.

Grazie all’interfaccia Swagger, è stato possibile dettagliare ogni aspetto delle API esposte, rendendo più agevoli le fasi di testing e di integrazione del Game Repository. Questa soluzione ha permesso di

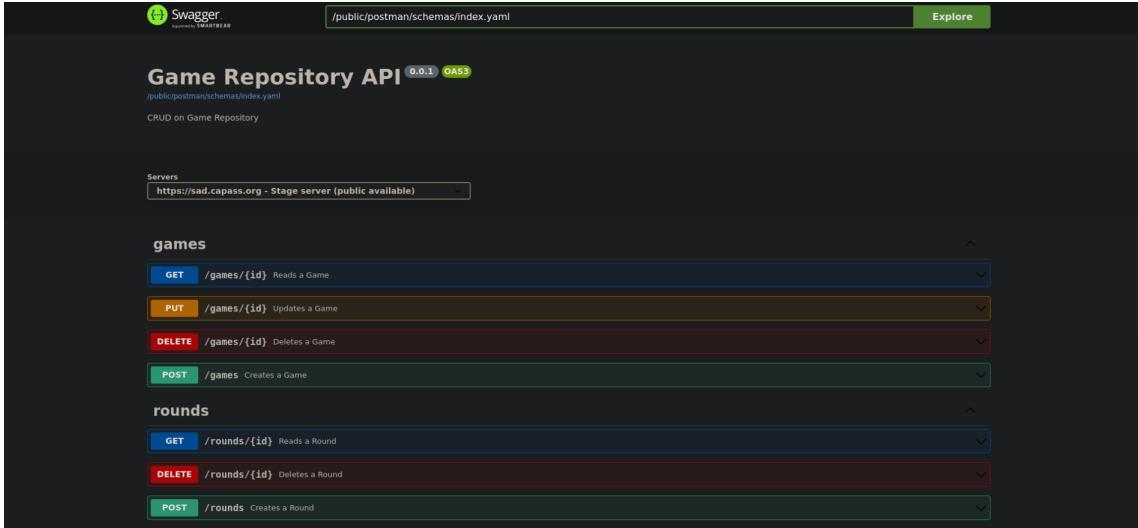


Figure 3.7: Swagger UI

illustrare con precisione informazioni chiave come i percorsi (path), i parametri, i dettagli delle richieste (request) e le potenziali risposte (response) fornite dai servizi. Per la progettazione degli endpoint secondo la specifica OpenAPI, è stato utilizzato Postman, un’operazione che ha permesso di concretizzare la documentazione API in modo strutturato. Successivamente, la specifica è stata integrata tramite GitHub nel progetto principale, assicurando così che le informazioni fossero facilmente reperibili e aggiornate. In aggiunta, si è provveduto a realizzare un’integrazione specifica nel progetto che consentisse di esporre la Swagger UI, la quale è stata configurata per servire sia la specifica OpenAPI che la collection di Postman all’atto della configu-

razione dell'applicazione con il flag 'enableSwagger' impostato su True. Questa implementazione ha significativamente contribuito alla standardizzazione della documentazione e dell'interfacciamento dei servizi API del Game Repository.

3.2.3 Documentazione API

L'API REST sviluppata da T4 è cruciale per la manipolazione e conservazione dei dati concernenti i giocatori, i robot e le partite. Realizzate nel linguaggio di programmazione Go, le quattro API interagiscono con un database Postgres per gestire transazioni e richieste di interrogazione. Le operazioni CRUD (Create, Read, Update, Delete) sono invocate attraverso metodi HTTP dai componenti integranti del progetto. Il Controller delle API riceve queste richieste, le smista al Service appropriato e poi rimanda i risultati al Controller.

Le API sono state implementate per le entità principali del sistema: Game, Round, Turn e Robot. L'efficacia di queste API è stata confermata attraverso test eseguiti con l'ausilio di Postman. La figura 3.8, che rappresenta il diagramma API serve come punto di riferimento visivo per comprendere meglio il funzionamento di questi metodi.

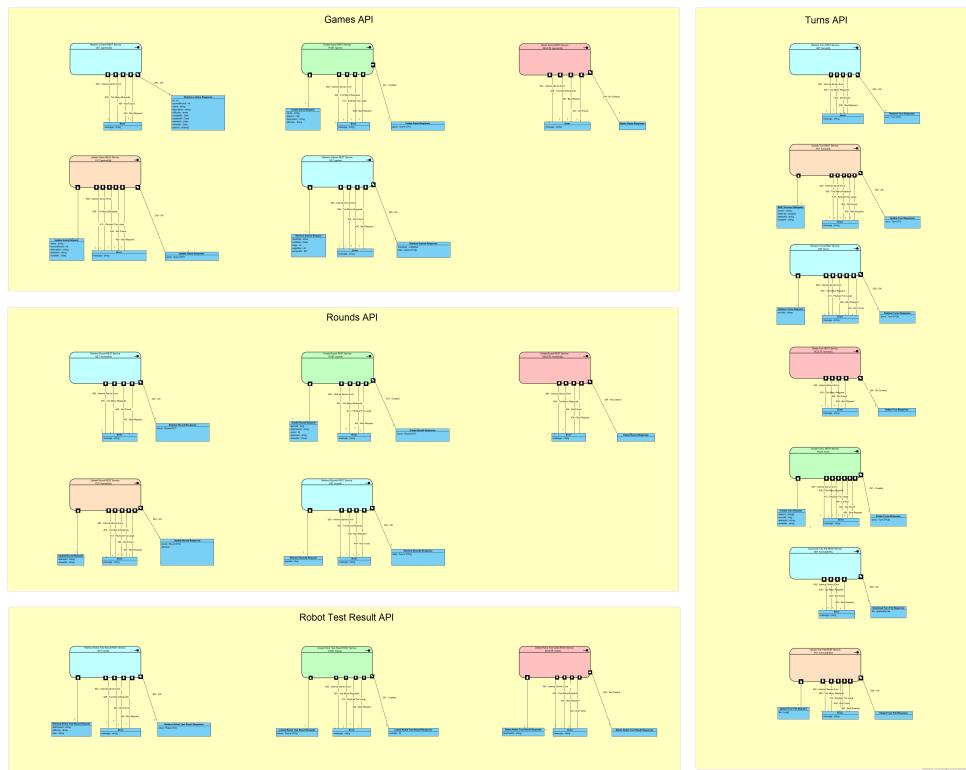


Figure 3.8: API Diagram

Il nostro gruppo si e' soffermato in particolare su 3 APIs:

- **CreateGame:** Questa funzione API consente la creazione di un nuovo gioco. Richiede dettagli come il nome del gioco, una descrizione e il livello di difficoltà. Una volta che il gioco è stato creato, gli viene assegnato un ID univoco, che può essere utilizzato per ulteriori operazioni come aggiungere giocatori, iniziare nuovi round, ecc.

CHAPTER 3. STATO INIZIALE DEL PROGETTO

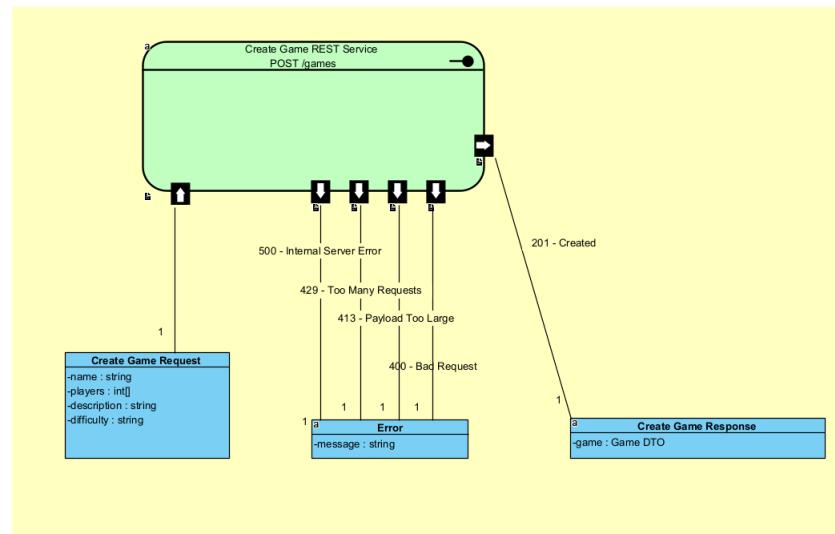


Figure 3.9: API CreateGame

- **UpdateGame** : L'API UpdateGame permette agli utenti di aggiornare le informazioni riguardanti un gioco specifico. Ciò può includere la modifica della descrizione del gioco, l'aggiornamento del round corrente o la chiusura di un gioco che è terminato. Le richieste di aggiornamento richiedono l'ID del gioco e i dettagli aggiornati da applicare.

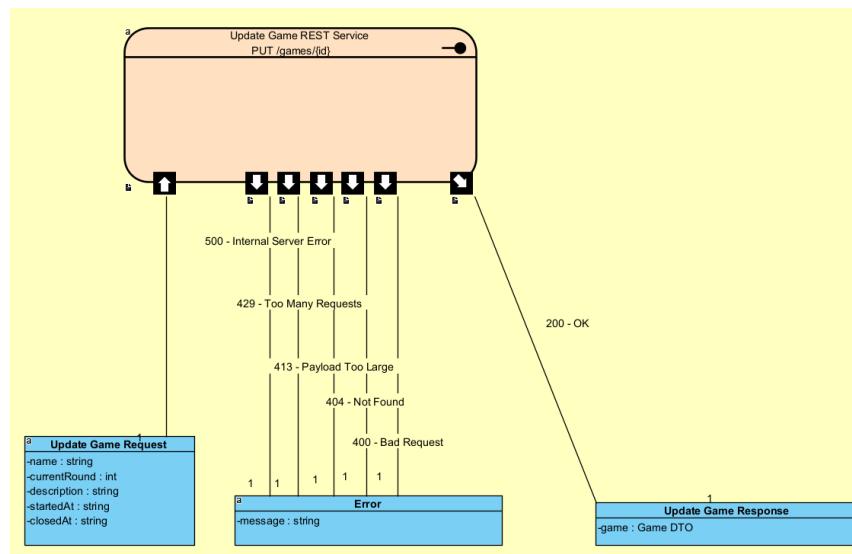


Figure 3.10: API UpdateGame

- **UpdateTurn** : Questa specifica API si occupa dell'aggiornamento dello stato di un turno all'interno di un gioco. Le operazioni tipiche possono includere la modifica di chi è il giocatore corrente, se il turno è stato completato, e altri metadati associati. Le richieste a questa API richiedono l'ID del turno e i nuovi dati che devono essere scritti nel database.

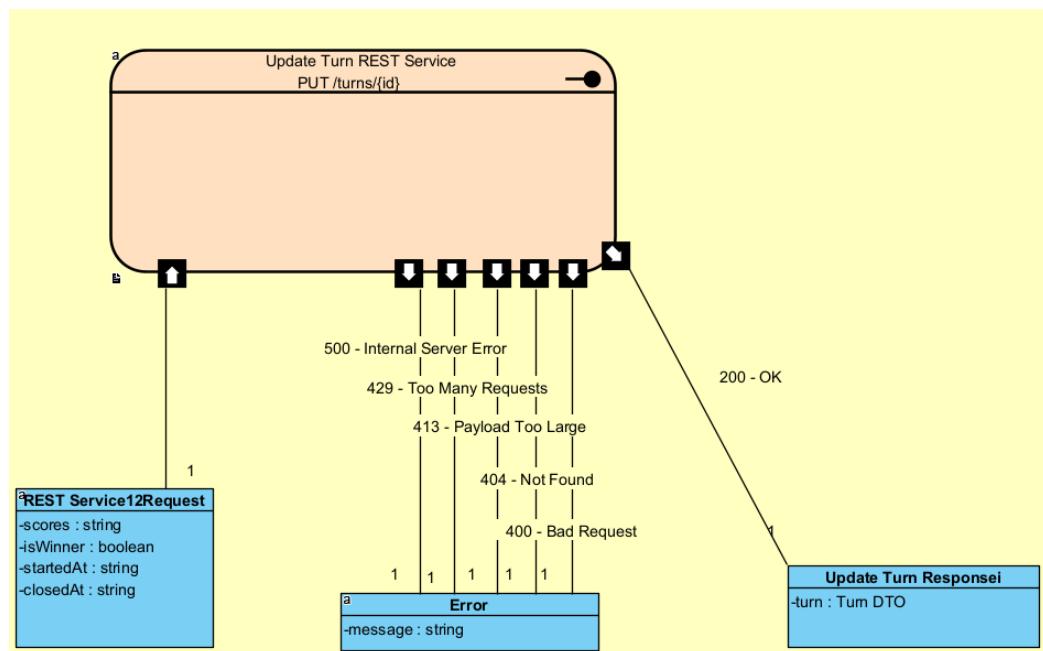


Figure 3.11: API UpdateTurn

Possiamo comprendere meglio le operazioni visualizzando i Sequence Diagrams.

3.2.4 CreateGame sequence diagram

Il diagramma di sequenza rappresentato nell'immagine 3.12 descrive il flusso di interazioni che avviene durante la creazione di un nuovo gioco

all'interno di un'applicazione di gioco. Il processo inizia con un attore che invia una richiesta di creazione gioco (Create) al Controller. Il Controller, a sua volta, inoltra la richiesta al Service associato tramite il metodo Create Game Request, che accetta un oggetto GameDTO (Data Transfer Object) come parametro.

All'interno del Service, si verifica una logica di validazione e di elabo-

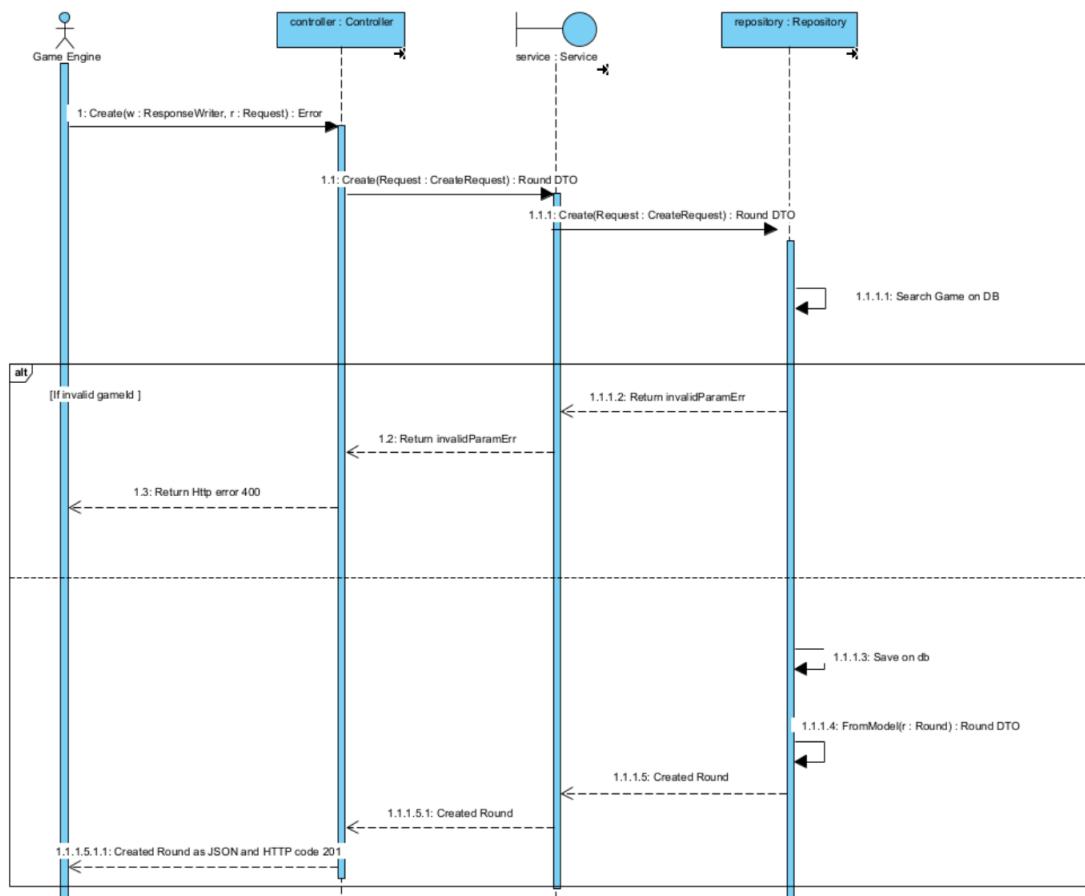


Figure 3.12: Diagramma di sequenza di CreateGame

razione, seguita dall'interazione con il Repository per salvare effettivamente i dati del gioco nel database. Durante questo processo, il Service può interagire con altre entità come Player e Game, come mostrato nel

diagramma. Una volta completato il salvataggio dei dati del gioco, il Repository ritorna un oggetto Game al Service, che a sua volta lo trasforma in un GameDTO da restituire al Controller. Se la creazione del gioco ha successo, il Controller restituisce una risposta al chiamante con un codice di stato HTTP 201, indicando che la risorsa (il gioco) è stata creata con successo, e l'oggetto GameDTO come corpo della risposta in formato JSON. Se invece la richiesta iniziale non contiene dati validi per la creazione di un gioco (ad esempio, mancano dati del giocatore), il Controller restituisce un errore 400, che sta ad indicare una "Bad Request". Questo diagramma mette in evidenza le responsabilità di ciascun componente all'interno del flusso di creazione di un gioco e come essi interagiscono tra loro seguendo un ordine ben definito per elaborare la richiesta.

3.2.5 UpdateGame sequence diagram

Il diagramma di sequenza rappresentato in figura 3.13 illustra il processo di aggiornamento di un gioco all'interno di un'applicazione gestita da un motore di gioco (Game Engine). Il flusso inizia con il Controller che riceve una richiesta di aggiornamento (Update), che include un ResponseWriter e una Request. Il Controller invoca quindi il metodo Update del Service, passando un ID di gioco e una UpdateRequest. Il Service esegue un'operazione Update sul Repository, passando gli stessi parametri. Il Repository inizia la sua sequenza di eventi con

CHAPTER 3. STATO INIZIALE DEL PROGETTO

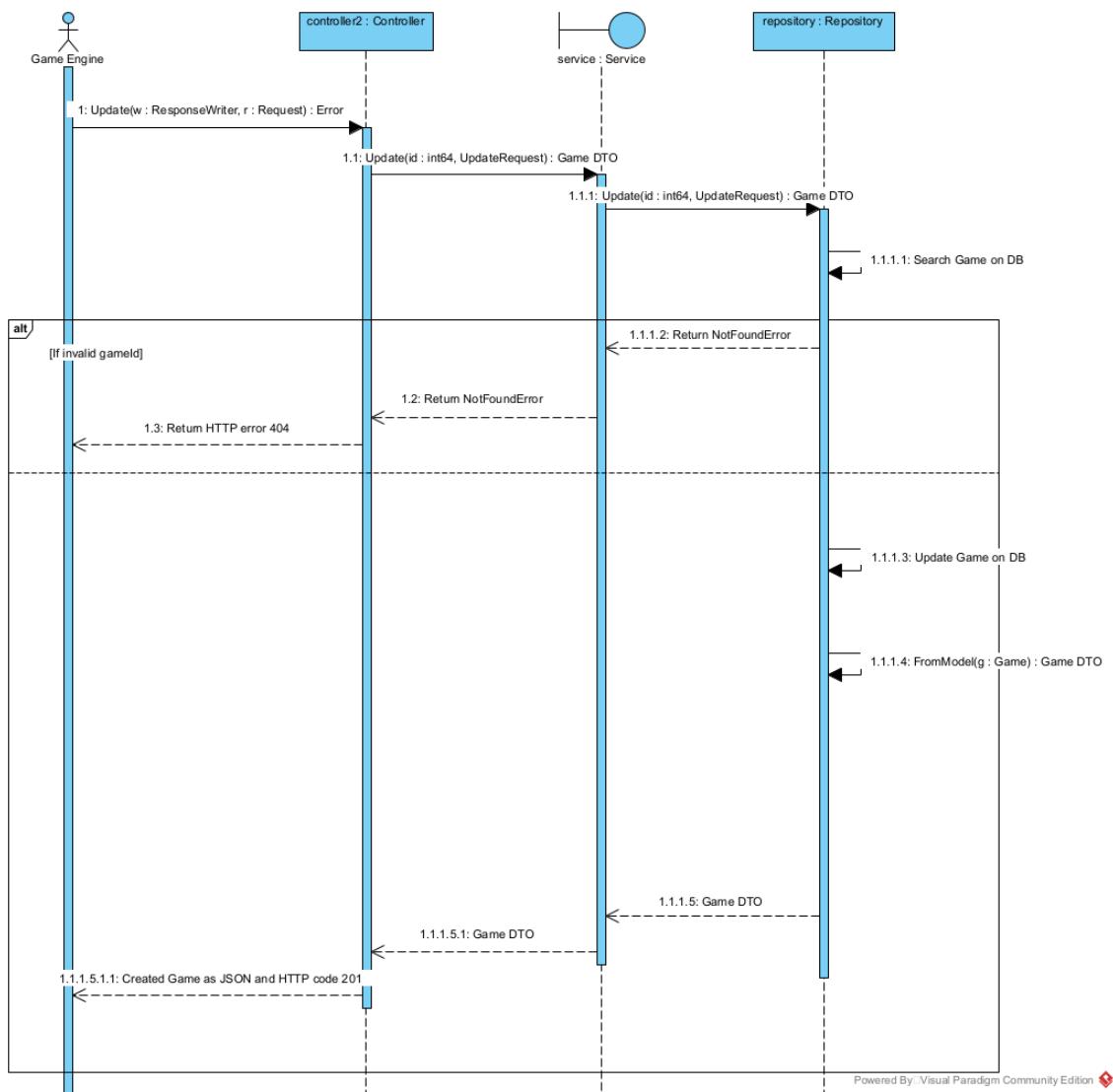


Figure 3.13: Diagramma di sequenza di `UpdateGame`

la ricerca del gioco nel database (DB). Se il gioco con l'ID specificato non viene trovato, il Repository restituisce un `NotFoundError`, che viene propagato indietro al Service e poi al Controller, il quale a sua volta risponde alla richiesta originale con un errore HTTP 404, indicando che la risorsa richiesta non è stata trovata. Se il gioco esiste, il Repository procede con l'aggiornamento del gioco nel database.

Dopo l'aggiornamento, converte i dati aggiornati in un Game DTO (Data Transfer Object), che viene passato indietro al Service e quindi al Controller. Infine, il Controller risponde alla richiesta originale con il Game DTO convertito in JSON e invia una risposta HTTP 200, indicando che l'operazione è stata eseguita con successo.

In sintesi, il diagramma descrive il flusso di un'operazione di aggiornamento standard in un'architettura di sistema basata su microservizi, con un chiaro meccanismo di gestione degli errori e di conferma del successo dell'operazione.

3.2.6 UpdateTurn sequence diagram

Il diagramma di sequenza in figura 3.14 fornisce una rappresentazione visiva del processo di aggiornamento di un turno gestito da un'entità denominata Game Engine2. Il flusso inizia con il Controller che riceve una richiesta HTTP per aggiornare un turno (Update), che passa poi al Service corrispondente. Il Service elabora la richiesta, invocando il metodo Update del Repository con un identificativo di turno e una UpdateRequest. Il Repository prima tenta di localizzare il turno specifico nel database. Se il turno non è presente, restituisce un NotFoundError. Questo errore si propaga al Service e successivamente al Controller, che risponde alla richiesta iniziale con un codice di stato HTTP 404, indicando che il turno richiesto non è stato trovato. Qualora il turno sia trovato, il Repository procede con l'aggiornamento del record nel

CHAPTER 3. STATO INIZIALE DEL PROGETTO

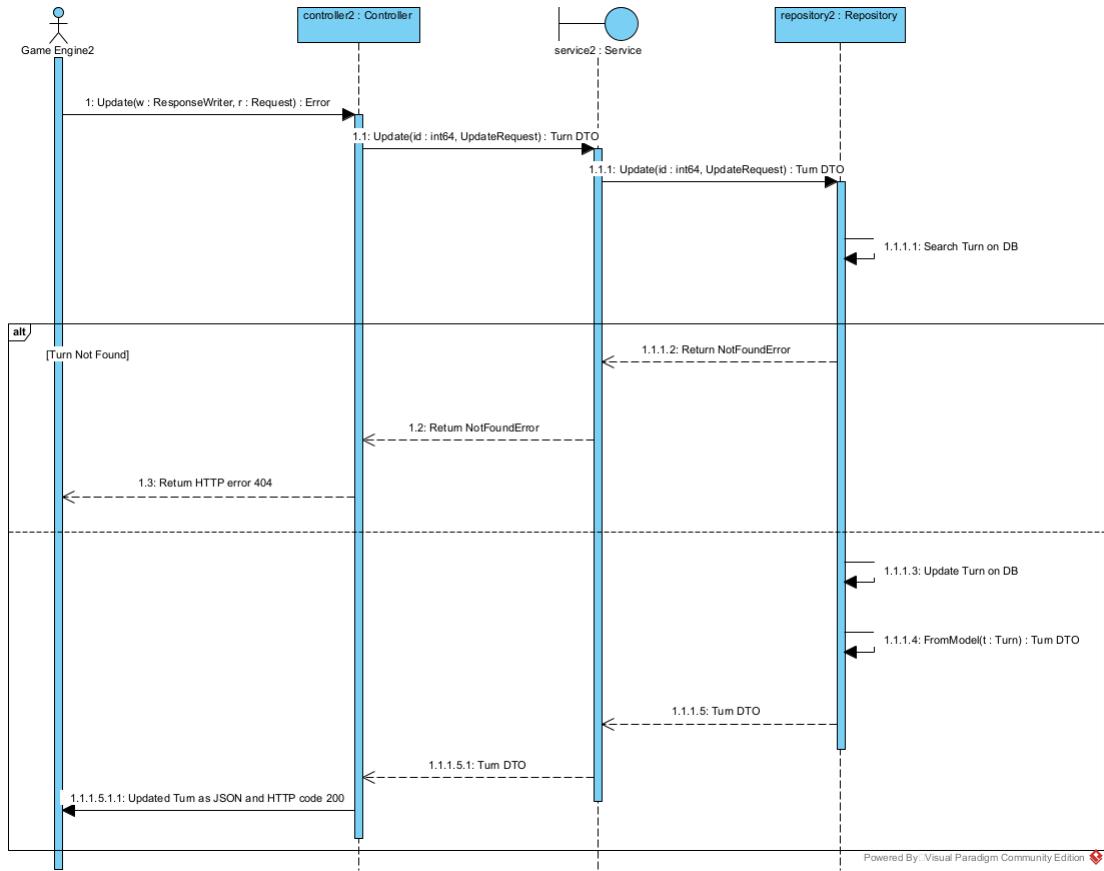


Figure 3.14: Diagramma di sequenza di `UpdateTurn`

database. Successivamente, trasforma i dati aggiornati in un `Turn DTO` (Data Transfer Object). Questo `DTO` viene inviato indietro al Service e poi al Controller, che finalizza il processo inviando una risposta HTTP con codice 200, insieme al `Turn DTO` aggiornato, indicando che l'operazione di aggiornamento è stata completata con successo. Questo diagramma di sequenza evidenzia il percorso di elaborazione standard per le richieste di aggiornamento di un turno, incluse le interazioni tra i vari componenti del sistema e la gestione degli errori.

3.3 Stato del Task T5

Il nostro studio si è esteso anche al task T5, focalizzandoci sulle classi responsabili della gestione dei dati relativi alle partite. Abbiamo identificato e risolto questioni di coerenza tra i dati memorizzati nel Game Repository (database del task T4) e quelli archiviati nello Student Repository (sistema di file del task T5). Inizialmente, abbiamo esaminato la documentazione del T5 per comprendere la sua architettura. Il task T5 offre un'API per gestire i dati legati alla creazione e all'avvio di partite, coprendo sia il front-end, che permette agli utenti di autenticarsi e iniziare una partita, sia il back-end, che si occupa di salvare i dati nel database e nel file system. Il sistema è implementato tramite classi Java per la logica di business e di controllo, e una web-app costruita con HTML, CSS, JavaScript e il framework Bootstrap per la logica di presentazione. L'architettura del T5 segue il pattern MVC, con il Class Diagram dello Student Repository che mostra chiaramente la divisione in pacchetti Model, View, Controller, oltre a pacchetti dedicati alle interfacce.

Possiamo comprendere meglio le operazioni di cui T5 si occupa complessivamente visualizzando il Sequence Diagram in Figura 3.15.

CHAPTER 3. STATO INIZIALE DEL PROGETTO

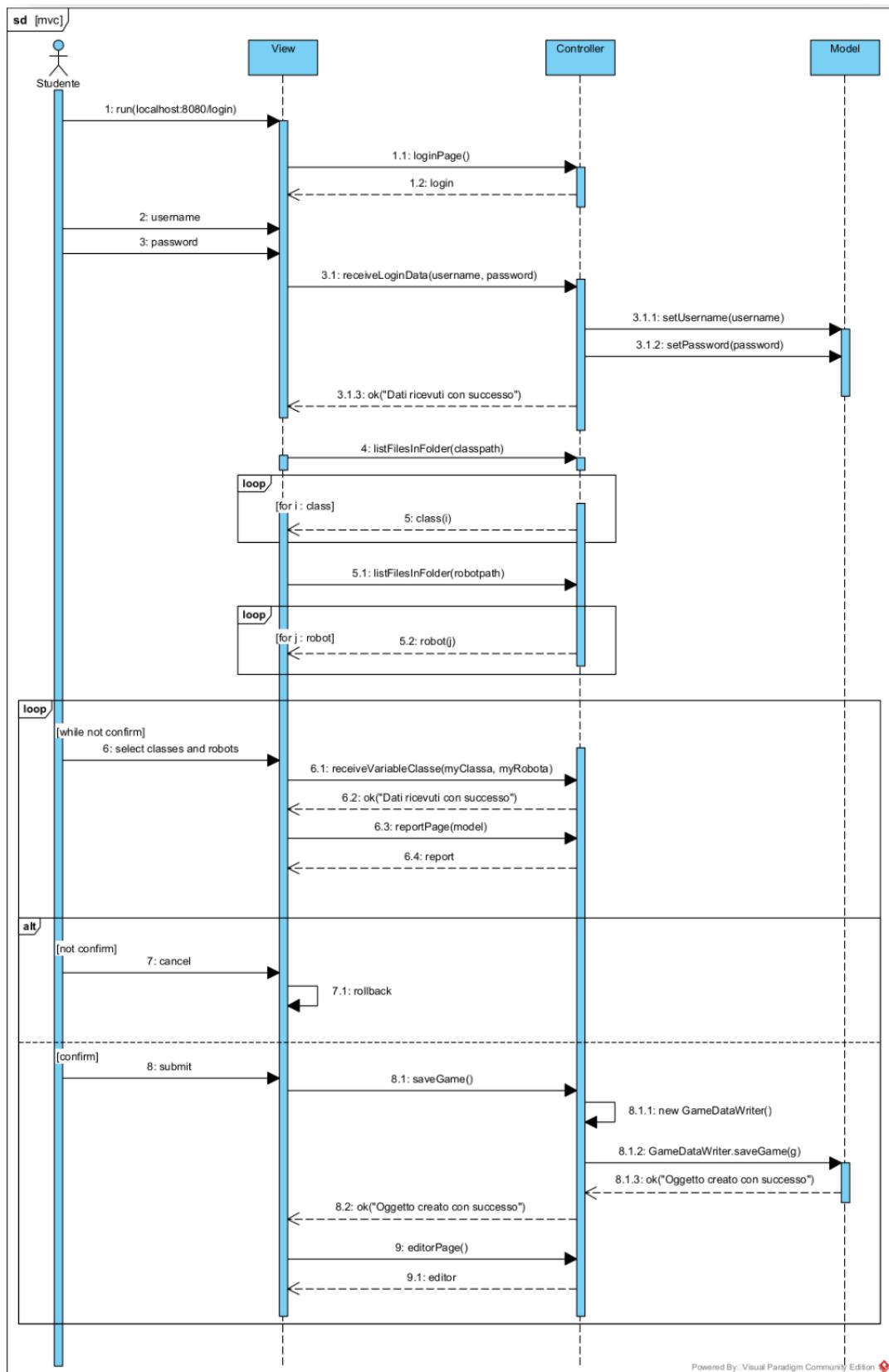


Figure 3.15: Diagramma di sequenza

Chapter 4

Modifiche al componente T4

Per soddisfare i requisiti R9 e R11, relativi alla gestione dello storico delle partite e alla creazione di classifiche di giocatori, abbiamo approntato le seguenti modifiche essenziali al codice del componente T4 ed in parte a quello del T5. Tutte le modifiche verranno poi approfondite nei capitoli seguenti.

4.1 Modifiche al diagramma ER

La modifica di un diagramma ER è un passo fondamentale nel processo di sviluppo di un'applicazione o sistema informativo, in quanto riflette una comprensione più profonda o aggiornata della struttura dei dati e delle relazioni all'interno del dominio del problema.

Nella transizione dalla vecchia versione del diagramma ER a quella nuova, sono state apportate alcune modifiche sia ai campi delle tabelle che alle dipendenze tra esse:

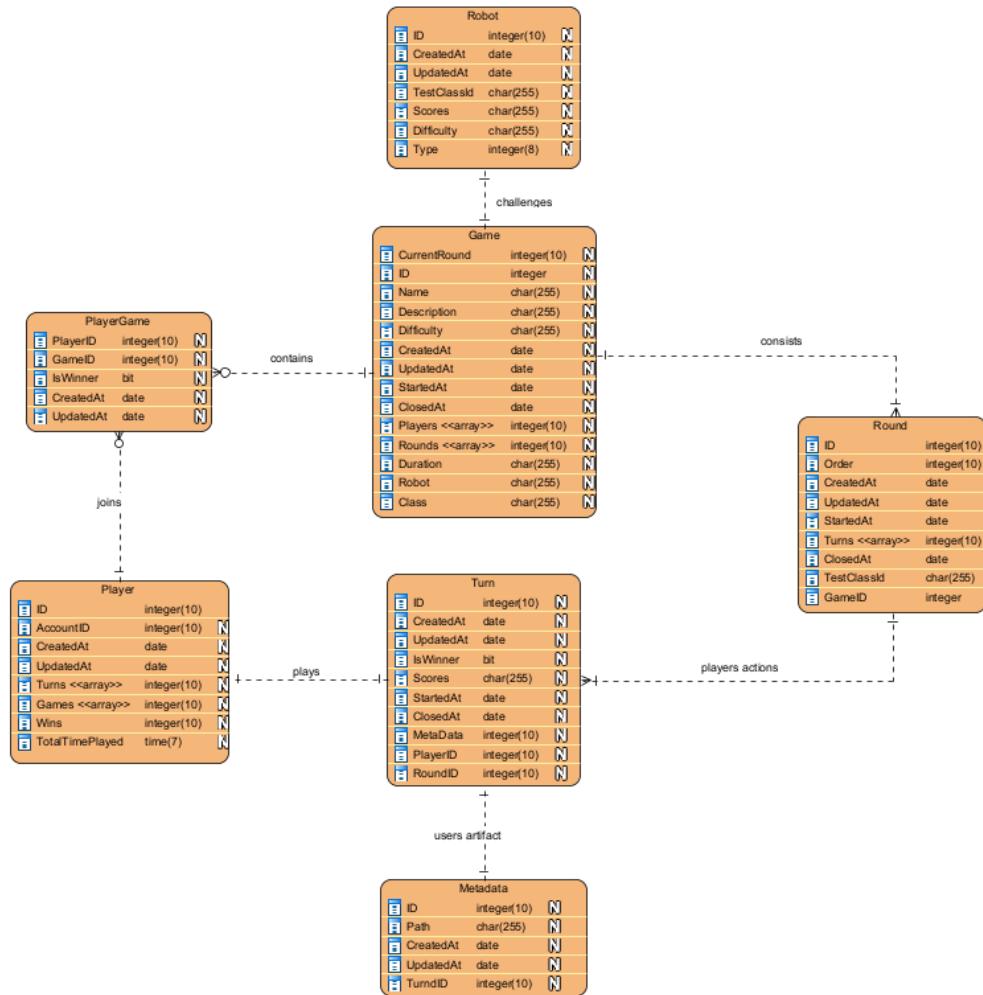


Figure 4.1: Diagramma ER

4.1.1 Tabella PlayerGame

Nella vecchia versione, la tabella Game_Player collega Player e Game.

Nella nuova versione, questa è stata aggiornata, cambiandole innanzi-

tutto il nome che era errato e aggiungendo i campi PlayerID e GameID al fine di poter identificare giocatori e partite tramite questa tabella.

4.1.2 Tabella Player

La tabella Player nella nuova versione include quattro nuovi campi:

- Turns[] e Games[], due vettori che contengono rispettivamente i turni e le partite;
- Wins, che indica il numero totale di partite vinte;
- TotalTimePlayed, tramite il quale si puo' ora tenere traccia del tempo totale giocato.

Inoltre sono state inserite due nuove relazioni: la prima e' una relazione uno a molti con la tabella PlayerGame e la seconda e' una relazione uno ad uno con la tabella Turn.

4.1.3 Tabella Game

Nella tabella Game sono stati aggiunti vari campi:

- Players[] e Rounds[], due array che contengono rispettivamente i giocatori della partita e tutti i round.
- Duration, che contiene informazioni relative alla durata della partita;

- Robot, che identifica il robot sfidato nel corso della partita;
- Class, che contiene la classe utilizzata per giocare;

E' stata anche inserita una nuova relazione, di tipo uno ad uno con la tabella Robot.

4.1.4 Tabella Round

Nella tabella round sono stati aggiunti tre campi:

- Turns[], che è un array che contiene tutti i Turni che sono stati fatti nel corso del round;
- TestClassId, che contiene l'Identificativo della classe che corrisponde al nome della classe stessa;
- GameID, contiene l'ID della partita che si sta giocando.

4.1.5 Tabella Turn

Nella tabella Turn sono stati aggiunti PlayerID e RoundID, che contengono gli identificativi del player che sta giocando e del round.

4.1.6 Tabella Metadata

Nella tabella Metadata è stato aggiunto l'attributo TurnID, ossia l'identificativo del turno. Queste modifiche riflettono una complessiva

evoluzione del sistema verso una maggiore flessibilità, dettaglio e scalabilità.

4.2 Modifiche al codice

Quando apportiamo modifiche a un diagramma ER, è essenziale che il codice sorgente dell'applicazione venga aggiornato di conseguenza per garantire che la base di dati e le interazioni con essa siano allineate con il nuovo modello concettuale. Questo processo di aggiornamento del codice in risposta alle modifiche del diagramma ER è cruciale per mantenere l'integrità dei dati, la coerenza delle operazioni e l'efficienza complessiva del sistema.

4.2.1 Modifiche al file model.go

Abbiamo apportato significative revisioni al file model.go per garantire una completa coerenza con le recenti modifiche apportate al diagramma ER.

Struct Game

- Campo Duration: È stato aggiunto un nuovo campo Duration di tipo string. Questo campo contiene la durata della partita.
- Campo Robot: È stato aggiunto un campo Robot di tipo string. Questo campo tiene traccia del robot sfidato durante la partita.

- Campo Class: È stato aggiunto un campo Class di tipo string.
Questo campo si riferisce alla classe testata durante la partita.

```
8  type Game struct {
9      CurrentRound int           `gorm:"default:1"`
10     ID          int64         `gorm:"primaryKey;autoIncrement"`
11     Name        string        `gorm:"default:null"`
12     Description sql.NullString `gorm:"default:null"`
13     Difficulty  string        `gorm:"default:null"`
14     CreatedAt   time.Time    `gorm:"autoCreateTime"`
15     UpdatedAt   time.Time    `gorm:"autoUpdateTime"`
16     StartedAt   *time.Time   `gorm:"default:null"`
17     ClosedAt   *time.Time   `gorm:"default:null"`
18     Rounds      []Round      `gorm:"foreignKey:GameID;constraint"`
19     Players     []Player     `gorm:"many2many:player_games;forei"`
20     Duration    string        `gorm:"default:null" // aggiunto`
21     Robot       string        `gorm:"default:null" // aggiunto`
22     Class       string        `gorm:"default:null" // aggiunto`
23 }
```

Figure 4.2: Struct Game

Struct Player

- Campo Wins, di tipo int64 per tenere traccia del numero totale di partite vinte da un giocatore.
- Campo TotalTimePlayed, di tipo time.Duration per registrare il tempo totale giocato. Il tempo è registrato in nanosecondi e necessita di conversione quando viene recuperato dal database.

```
type Player struct {
    ID          int64      `gorm:"primaryKey;autoIncrement"`
    AccountID   string     `gorm:"unique"`
    CreatedAt   time.Time  `gorm:"autoCreateTime"`
    UpdatedAt   time.Time  `gorm:"autoUpdateTime"`
    Turns       []Turn     `gorm:"foreignKey:PlayerID"`
    Games       []Game     `gorm:"many2many:player_games"`
    Wins        int64      `gorm:"default:0" //`
    TotalTimePlayed time.Duration `gorm:"default:0" //`
}
```

Figure 4.3: Struct Player

Struct PlayerGame

Il campo PlayerID era prima dichiarato come string, ora è stato corretto a int64

```
29 type PlayerGame struct {
30     PlayerID  int64      `gorm:"primaryKey" //aggiunto messo a int`
31     GameID    int64      `gorm:"primaryKey"`
32     CreatedAt time.Time `gorm:"autoCreateTime"`
33     UpdatedAt time.Time `gorm:"autoUpdateTime"`
34     IsWinner  bool       `gorm:"default:false"`
35 }
```

Figure 4.4: Struct PlayerGame

4.2.2 Modifiche al file Game.java in T5

Game.java rappresenta la struttura della tabella nel contesto di un'applicazione Java. Questa classe è stata aggiornata per riflettere le modifiche fatte nella struttura Game definita nel file model.go. L'aggiorname-

nto assicura che entrambe le rappresentazioni della tabella Game siano coerenti tra loro.

L'unico campo che è stato aggiunto è robot, di tipo stringa.

```
5  public class Game {  
6      private int playerId;  
7      private long id;  
8      private String description;  
9      private String name;  
10     private String difficulty;  
11     private LocalDate data_creazione;  
12     private String ora_creazione;  
13     private String classe;|  
14     private String robot; // aggiunto
```

Figure 4.5:

4.2.3 UpdateGame in game/service.go

Abbiamo notato che il parametro 'IsWinner' nella tabella 'PlayerGame' non veniva aggiornato correttamente dopo la vittoria in una partita. Di conseguenza, abbiamo apportato delle modifiche alla funzione 'UpdateGame' al fine di risolvere questo problema.

Di seguito il codice modificato:

CHAPTER 4. MODIFICHE AL COMPONENTE T4

```
46 func (gs *Repository) Update(id int64, r *UpdateRequest) (Game, error) {
47
48     var (
49         game model.Game = model.Game{ID: id}
50         err   error
51     )
52
53     // Aggiorna il gioco con i nuovi valori
54     err = gs.db.Model(&game).Updates(r).Error
55     if err != nil {
56         return Game{}, api.MakeServiceError(err)
57     }
58
59     // Ricarica il gioco per ottenere i dati aggiornati, inclusi StartedAt e ClosedAt
60     err = gs.db.First(&game, id).Error
61     if err != nil {
62         return Game{}, api.MakeServiceError(err)
63     }
64
65     // Controlla se StartedAt che ClosedAt sono non nulli
66     if game.StartedAt != nil && game.ClosedAt != nil {
67
68         // Trova il Round associato al Game
69         var round model.Round
70         err := gs.db.Where("game_id = ?", id).First(&round).Error
71         if err != nil {
72             fmt.Println("Errore nella ricerca del Round:", err)
73         }
74
75         // Trova il Turn associato al Round
76         var turn model.Turn
77         err = gs.db.Where("round_id = ?", round.ID).First(&turn).Error
78         if err != nil {
79             fmt.Println("Errore nella ricerca del Turn:", err)
80         }
81
82         // Trova il Player associato al Turn
83         var player model.Player
84         err = gs.db.First(&player, turn.PlayerID).Error
85         if err != nil {
86             fmt.Println("Errore nella ricerca del Player:", err)
87         }
88
89         // Controlla se il turno è stato vinto
90         if turn.IsWinner {
91             // Crea un oggetto PlayerGame con le chiavi primarie impostate
92             playerGame := model.PlayerGame{
93                 PlayerID: turn.PlayerID,
94                 GameID:   id,
95             }
96
97             // Aggiorna il campo IsWinner a true per l'oggetto specificato
98             err = gs.db.Model(&playerGame).Update("IsWinner", true).Error
99             if err != nil {
100                 fmt.Println("Errore nell'aggiornamento di IsWinner in PlayerGame:", err)
101             }
102         }
103     }
104 }
```

Figure 4.6: Funzione UpdateGame in service.go

Ora la funzione controlla che la partita sia conclusa e vinta, se la partita è effettivamente vinta cerca un'entità PlayerGame tramite PlayerID e GameID, poi aggiorna PlayerGame.IsWinner.

4.3 Modifiche ai diagrammi di sequenza

I sequence diagrams sono stati modificati coerentemente con le modifiche apportate. In particolare sono stati cambiati in gran parte UpdateGame e UpdateTurn per riflettere le modifiche fatte al codice delle rispettive funzioni, modifiche che verranno analizzate nel capitolo seguente.

4.3.1 Diagramma di sequenza di UpdateGame

Il nuovo sequence diagram di UpdateGame, (in Figura 4.7) e' ora strutturato nel seguente modo:

1. Un giocatore inizia il processo inviando una richiesta di aggiornamento (Update _ Request(id)) al repository.
2. Il repository riceve la richiesta e procede con la richiesta di ottenere il gioco dal database (FetchGameByID).
3. Il database esegue l'operazione di recupero (FetchingGame) e ritorna le informazioni del gioco al repository (ReturnGame).
4. Il repository, dopo aver ricevuto i dati del gioco, fa una richiesta di aggiornamento al database con i nuovi valori (RequestUpdate(New Values)).
5. Il database conferma l'aggiornamento (ConfirmingUpdate) e ritorna al repository indicando che il gioco è stato aggiornato (Re-

turnGameUpdated).

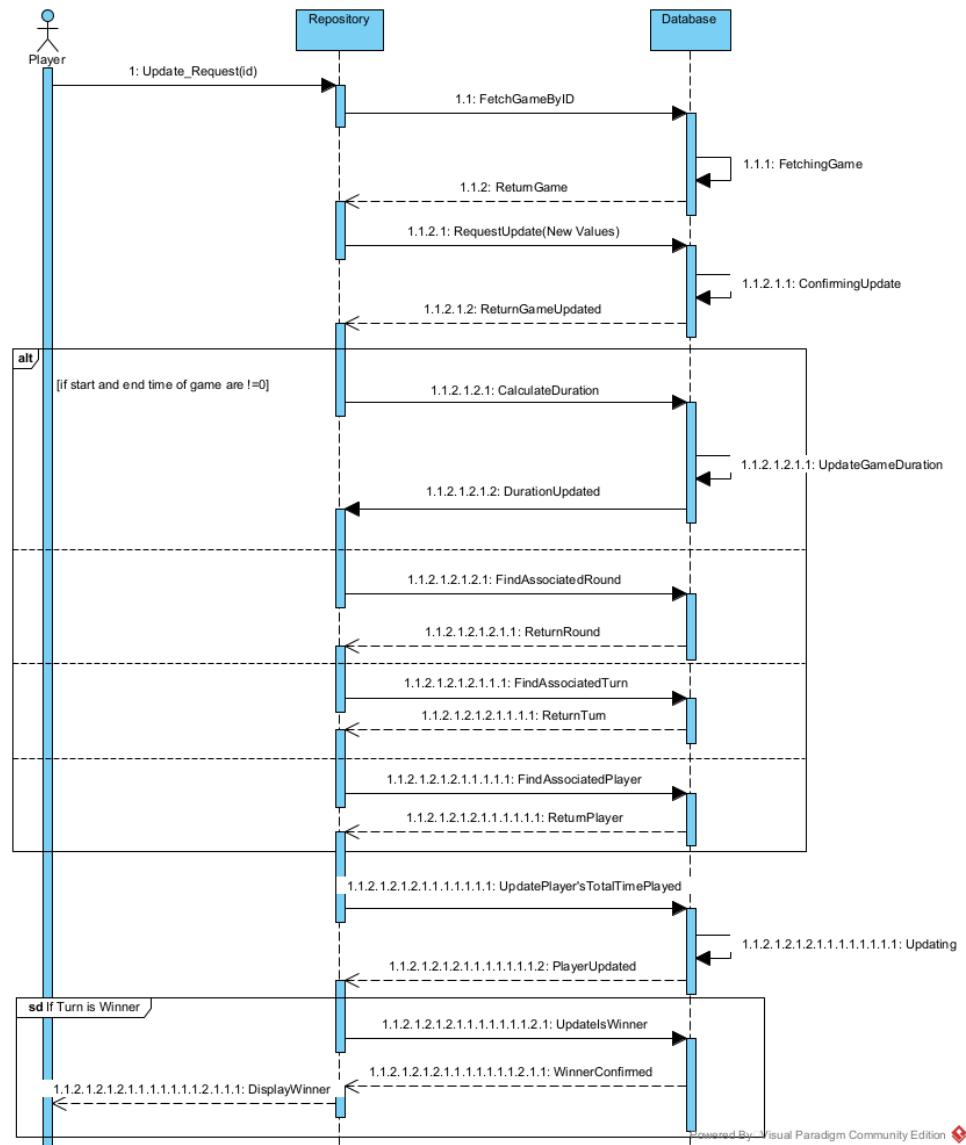


Figure 4.7: Diagramma di sequenza di UpdateGame

Dopo questo, c'è una valutazione condizionale alt che verifica se l'ora di inizio e di fine del gioco sono diverse da zero:

- Se è vero, il repository calcola la durata del gioco (CalculateDuration) e poi aggiorna questa durata nel database (UpdateGameDuration). Il database conferma l'aggiornamento della durata

(DurationUpdated).

Successivamente, il repository esegue una serie di operazioni per trovare e ritornare dati associati al gioco, come il round associato (FindAssociatedRound e ReturnRound), il turno associato (FindAssociatedTurn e ReturnTurn), e il giocatore associato (FindAssociatedPlayer e ReturnPlayer). Dopo aver recuperato queste informazioni:

- Il repository aggiorna il tempo totale giocato dal giocatore nel database (UpdatePlayer'sTotalTimePlayed), e il database conferma questo aggiornamento (PlayerUpdated).

Infine, c'è un'ulteriore valutazione condizionale che verifica se il turno corrente corrisponde a un vincitore:

- Se è vero, il repository aggiorna le informazioni del vincitore nel database (UpdatesIsWinner) e il database conferma che il vincitore è stato aggiornato (WinnerConfirmed). Poi, il vincitore viene mostrato (DisplayWinner).

4.3.2 Diagramma di sequenza di UpdateTurn

Il secondo diagramma di sequenza che è stato profondamente modificato è quello relativo a UpdateTurn. Il nuovo diagramma di sequenza (in Figura 4.8) è così composto:

1. Il giocatore inizia inviando una richiesta di aggiornamento (Update(id, r)) al repository.

2. Il repository elabora la richiesta eseguendo una query al database per recuperare il turno specificato (QueryTurnByID).
3. Il database risponde restituendo i dettagli del turno (ReturnTurn).

A questo punto, il diagramma introduce un punto di decisione alt per gestire gli errori di query:

- Se si verifica un errore di query, il repository restituisce un errore al giocatore (ReturnError).
- Se non c'è errore di query, il repository procede con l'aggiornamento del turno con i nuovi dati forniti (r), e il database conferma l'aggiornamento (Confirm Update).

Successivamente, c'è un altro punto di decisione alt per gestire gli errori di aggiornamento:

- Se si verifica un errore di aggiornamento, il repository restituisce di nuovo un errore al giocatore (ReturnError).

Infine, c'è un'ultima valutazione alt per determinare se il giocatore è il vincitore:

- Se il giocatore è il vincitore (IsWinner True), il repository incrementa il conteggio delle vittorie del giocatore (Increment Player Wins) e il database conferma l'incremento (Confirm Increment).

Il processo si conclude con il repository che restituisce al giocatore il turno aggiornato confermando che non ci sono stati errori (Return Updated Turn, No Error) o, in caso di errore durante l'incremento, restituendo un errore (ReturnError).

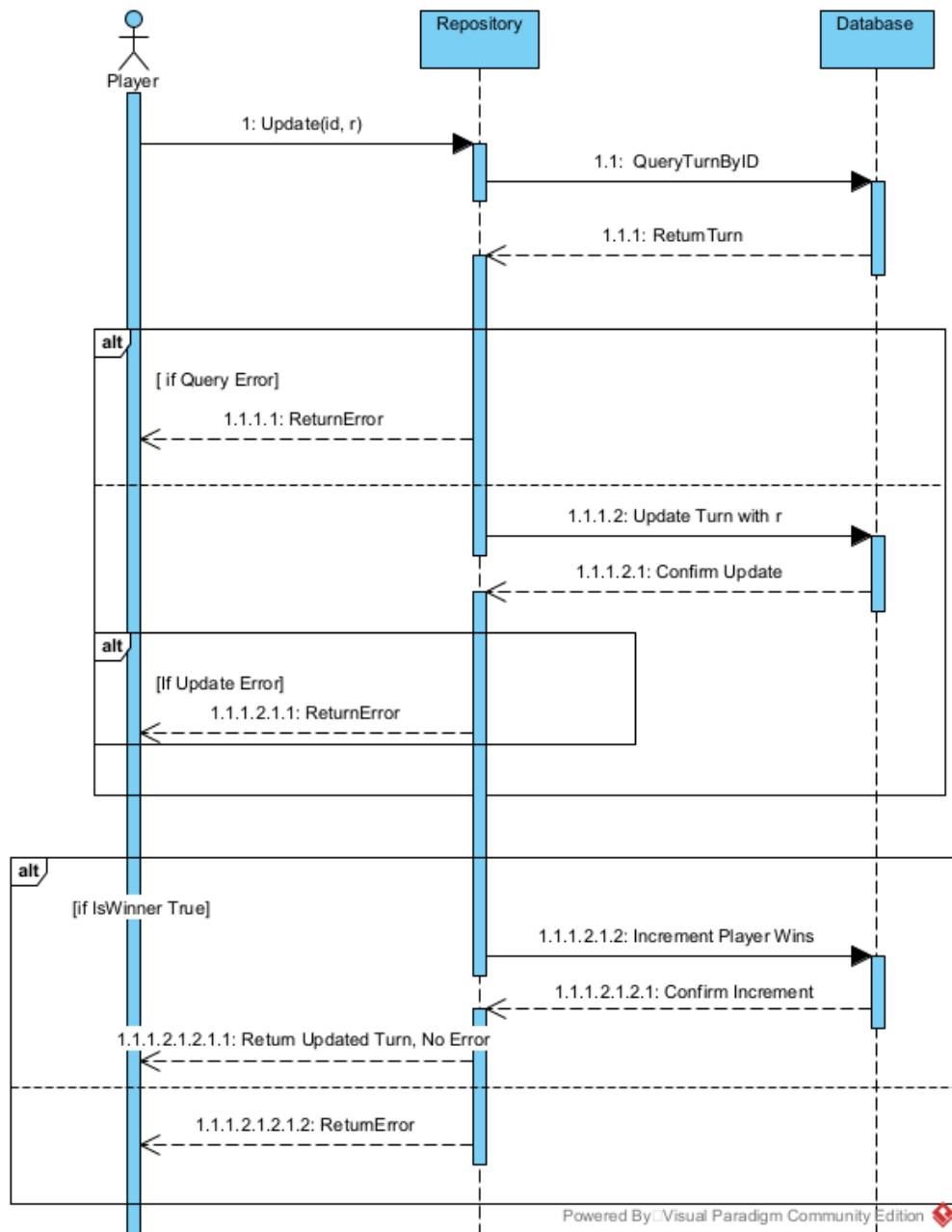


Figure 4.8: Diagramma di sequenza di `UpdateTurn`

4.4 Altre modifiche

4.4.1 API diagram

Abbiamo modificato anche l'API Diagram di CreateGame, in particolare all'interno della Request, aggiungendo due nuovi campi:

- Robot, di tipo string, per indicare il robot che verra' sfidato nella partita;
- Class, anche questo di tipo string, che rappresenta la classe che il player decide di testare all'interno della partita.

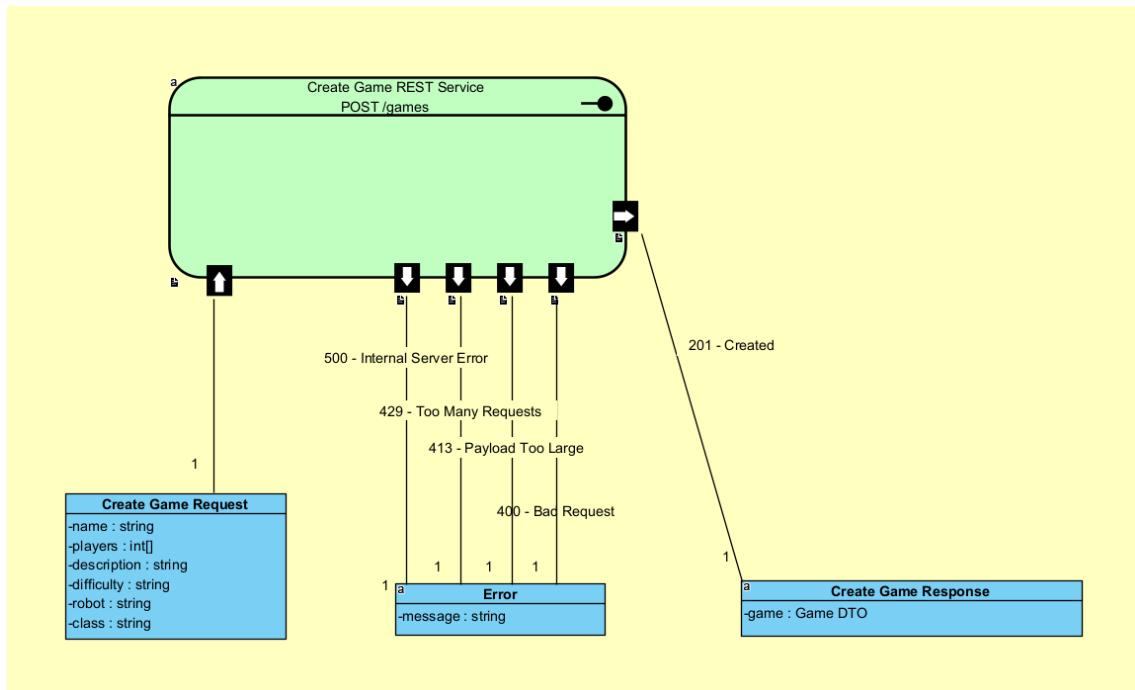


Figure 4.9: API diagram

In seguito sono state apportate modifiche anche al codice di CreateRequest:

```
31 v type CreateRequest struct {
32     Name      string      `json:"name"`
33     Players    []string    `json:"players"`
34     Description string     `json:"description"`
35     Difficulty string     `json:"difficulty"`
36     StartedAt *time.Time `json:"startedAt,omitempty"`
37     ClosedAt   *time.Time `json:"closedAt,omitempty"`
38     Robot      string     `json:"robot"` // aggiunto
39     Class      string     `json:"class"` // aggiunto
40 }
```

Figure 4.10: CreateRequest

4.4.2 System Model Domain

Abbiamo apportato delle modifiche al System Domain Model in modo che fosse coerente con il nuovo diagramma ER, queste sono state aperte a livello del package model e anche i DTO sono stati modificati.

CHAPTER 4. MODIFICHE AL COMPONENTE T4

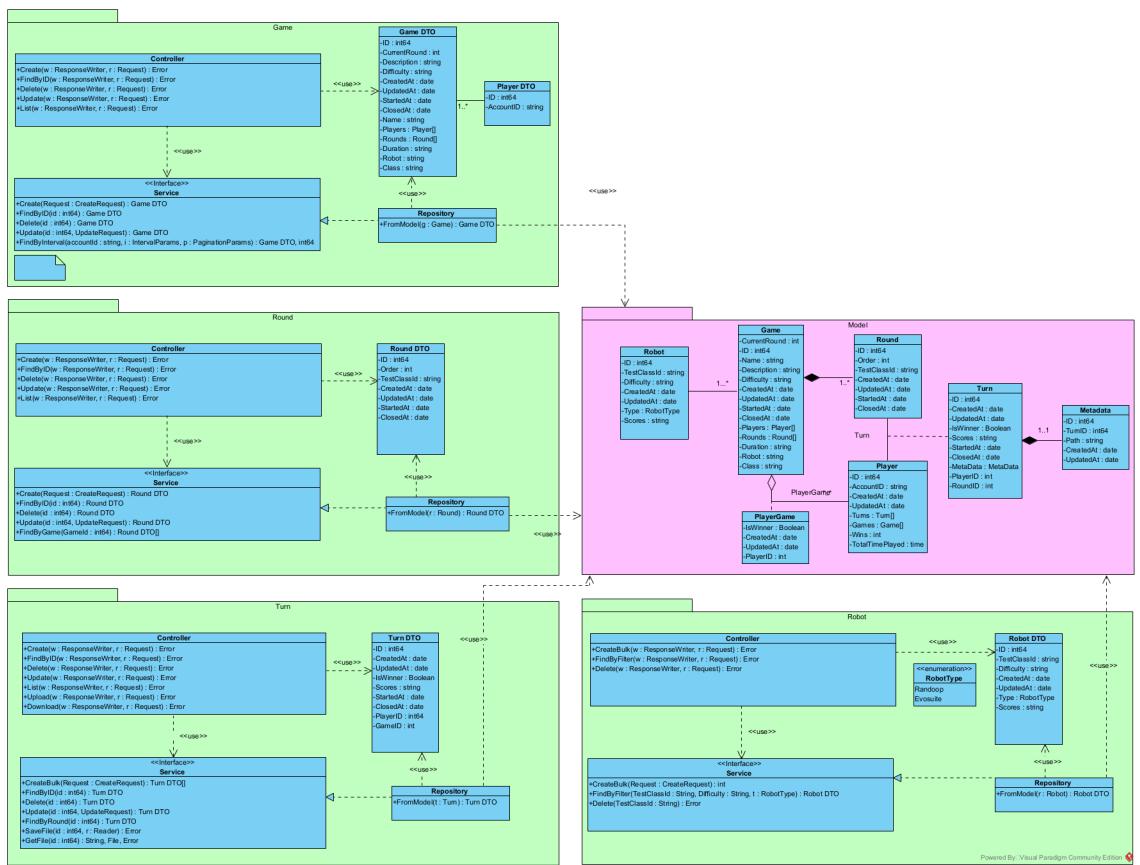


Figure 4.11: System Model Domain

Chapter 5

Requisito R9

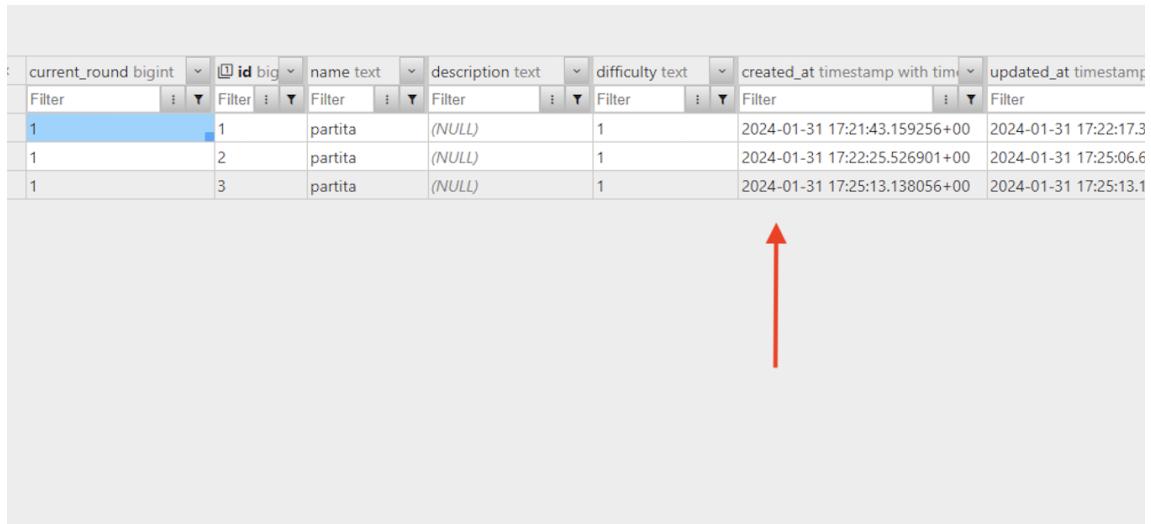
In questo capitolo, ci dedicheremo ad approfondire e fornire una giustificazione dettagliata delle modifiche implementate nel nostro progetto, come precedentemente accennato nel capitolo 4. Esamineremo l'importanza di ciascuna modifica, evidenziando come queste abbiano contribuito a migliorare il sistema in linea con gli obiettivi prefissati. Questo capitolo serve quindi a fornire una comprensione completa del valore aggiunto da queste modifiche, in termini di funzionalità.

5.1 Storico partite

L'obiettivo di questa sezione è descrivere in dettaglio le modifiche e le implementazioni effettuate per la gestione dello storico delle partite all'interno dell'applicazione.

5.1.1 Data Partita

La data di ogni partita è già presente nel database, sarà visualizzata all'utente. Uno screenshot del database T4 in cui è indicata la data:



current_round bigint	id big	name text	description text	difficulty text	created_at timestamp with time zone	updated_at timestamp
1	1	partita	(NULL)	1	2024-01-31 17:21:43.159256+00	2024-01-31 17:22:17.3
1	2	partita	(NULL)	1	2024-01-31 17:22:25.526901+00	2024-01-31 17:25:06.6
1	3	partita	(NULL)	1	2024-01-31 17:25:13.138056+00	2024-01-31 17:25:13.1

Figure 5.1: Screenshot database T4

5.1.2 Durata

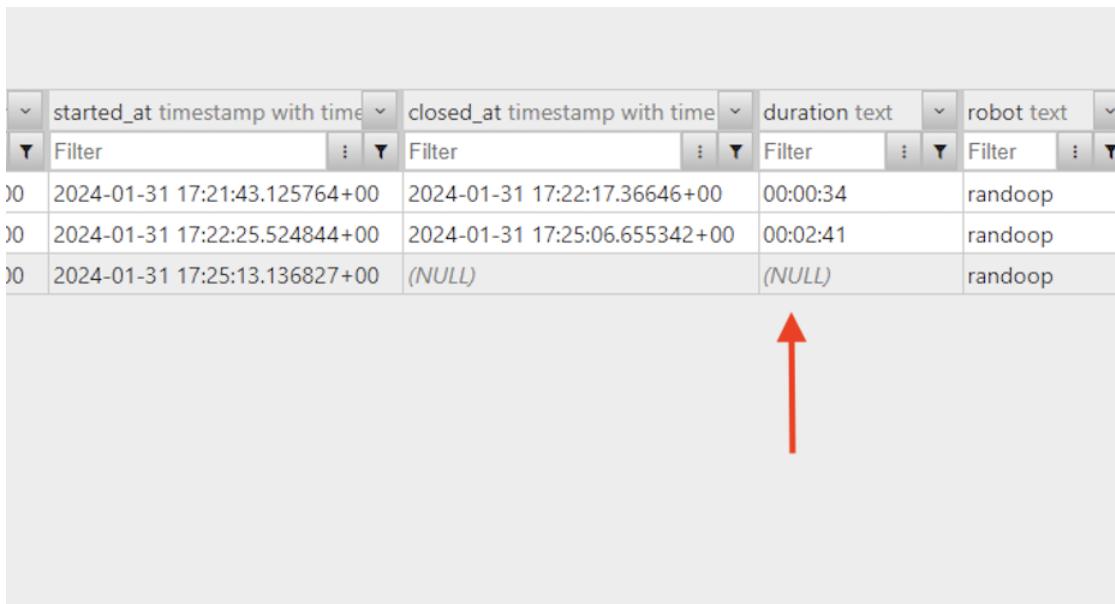
E' stato aggiunto un nuovo campo per tenere traccia della durata di ogni singola partita del giocatore.

In precedenza abbiamo già inserito uno screenshot del codice in cui è stato aggiunto questo nuovo campo. Di seguito la funzione in cui viene calcolata la durata:

```
100 func (gs *Repository) Update(id int64, r *UpdateRequest) (Game, error) {
101
102     var (
103         game model.Game = model.Game{ID: id}
104         err  error
105     )
106
107     // Aggiorna il gioco con i nuovi valori
108     err = gs.db.Model(&game).Updates(r).Error
109     if err != nil {
110         return Game{}, api.MakeServiceError(err)
111     }
112
113     // Ricarica il gioco per ottenere i dati aggiornati, inclusi StartedAt e ClosedAt
114     err = gs.db.First(&game, id).Error
115     if err != nil {
116         return Game{}, api.MakeServiceError(err)
117     }
118
119     // Controlla se StartedAt che ClosedAt sono non nulli
120     if game.StartedAt != nil && game.ClosedAt != nil {
121
122         duration := game.ClosedAt.Sub(*game.StartedAt)
123
124         // Calcola ore, minuti, secondi
125         hours := duration / time.Hour
126         duration -= hours * time.Hour
127         minutes := duration / time.Minute
128         duration -= minutes * time.Minute
129         seconds := duration / time.Second
130
131         // Crea una stringa nel formato HH:MM:SS
132         durationStr := fmt.Sprintf("%02d:%02d:%02d", hours, minutes, seconds)
133
134         game.Duration = durationStr
135
136         // Aggiorna nuovamente il gioco con la durata calcolata
137         err = gs.db.Model(&game).Update("Duration", durationStr).Error
138         if err != nil {
139             return Game{}, api.MakeServiceError(err)
140         }
141     }
142
143     return fromModel(&game), api.MakeServiceError(err)
144 }
```

Figure 5.2: Funzione che calcola la durata

Viene prima fatto il controllo sul game, si verifica che sia effettivamente concluso, in seguito si calcola la durata della partita e la si converte in stringa, nel database quindi viene salvata come stringa nel formato HH:MM:SS, in modo da essere piu' comprensibile.



	started_at timestamp with time	closed_at timestamp with time	duration text	robot text
▼	Filter	Filter	Filter	Filter
00	2024-01-31 17:21:43.125764+00	2024-01-31 17:22:17.36646+00	00:00:34	randoop
00	2024-01-31 17:22:25.524844+00	2024-01-31 17:25:06.655342+00	00:02:41	randoop
00	2024-01-31 17:25:13.136827+00	(NULL)	(NULL)	randoop

Figure 5.3: Screenshot database T4

5.1.3 Classe Testata, Robot e Difficoltà

Il campo Difficulty era già presente nella tabella Game, i campi Class e Robot, come detto in precedenza sono stati aggiunti. Il gioco era già predisposto a salvare questi dati, infatti dal front-end questi venivano inviati, purtroppo non veniva gestita la ricezione e il salvataggio di essi, esistevano delle funzioni (commentate) che provavano a gestire ciò ma non erano funzionanti. Nel prossimo paragrafo verranno approfondite le modifiche al componente T5 del progetto.

5.1.4 Page Report

Dal front-end le informazioni vengono inviate quando si accede alla pagina di report, la funzione in precedenza non gestiva alcuna informazione, semplicemente reindirizzava alla pagina di editor, ora fa una

richiesta di POST a '/sendRobotVariables' e cosi' invia le variabili Class, Robot e Difficulty. Di seguito la funzione redirectToPageReport():

```
79  function redirectToPagereport() {
80
81    console.log(classe);
82    console.log(robot);
83    console.log(difficulty);
84
85    if (classe && robot && difficulty) {
86
87      localStorage.setItem("classe", classe);
88      localStorage.setItem("robot", robot);
89      localStorage.setItem("difficulty", difficulty);
90
91      $.ajax({
92        url: 'http://localhost/api/sendRobotVariables', // L'URL del tuo endpoint sul server
93        type: 'POST', // Metodo HTTP da utilizzare
94        data: {
95          classe: classe,
96          robot: robot,
97          difficulty: difficulty
98        },
99        success: function (response) {
100          console.log('Dati inviati con successo');
101
102          window.location.href = "/report";
103        },
104        error: function (error) {
105          console.error('Errore nell invio dei dati');
106          alert("Dati non inviati con successo");
107        }
108      });
109    }
110  else {
111    alert("Seleziona una classe e un robot");
112    console.log("Seleziona una classe e un robot");
113  }
114}
115 }
```

Figure 5.4: Funzione redirectToPageReport

La route '/sendRobotVariables' è stata aggiunta anche al file application.yml presente nel container api_gateway.

```
33 |     sendRobotVariables-service:  
34 |         sensitiveHeaders:  
35 |             path: /api/sendRobotVariables/**  
36 |             url: http://t5-app-1:8080/sendRobotVariables
```

Figure 5.5: Route /sendRobotVariables

5.1.5 Ricezione e Salvataggio delle variabili Class, Robot e Difficulty

Ogni volta che c'è un operazione di POST su '/sendRobotVariables' viene chiamata la funzione receiveRobotVariables(), che legge i valori inviati (classe, robot, difficulty) e li salva in un oggetto g di tipo Game (Game classe di T5 non del database). Di seguito il codice della funzione:

```
232 | @PostMapping("/sendRobotVariables")  
233 | public ResponseEntity<String> receiveRobotVariable(@RequestParam("classe") String classe,  
234 |                                         @RequestParam("robot") String robot, @RequestParam("difficulty") String difficulty) {  
235 |  
236 |     System.out.println("classe ricevuta: " + classe);  
237 |     System.out.println("robot ricevuta: " + robot);  
238 |     System.out.println("difficoltà ricevuta: " + difficulty);  
239 |  
240 |     LocalTime oraCorrente = LocalTime.now();  
241 |     DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");  
242 |     String oraFormattata = oraCorrente.format(formatter);  
243 |  
244 |     g.setClasse(classe);  
245 |     g.setDifficulty(difficulty);  
246 |     g.setRobot(robot);  
247 |     g.setData_creazione(LocalDate.now());  
248 |     g.setOra_creazione(oraFormattata);  
249 |  
250 |     return ResponseEntity.ok(body:"Dati ricevuti con successo");  
251 | }
```

Figure 5.6: Funzione receiveRobotVariables

In seguito, quando l'utente passerà alla schermata di editor viene chiamata la funzione saveGame(), invocata su una POST sulla route '/save-data' (codice già esistente). La funzione saveGame() chiama la

funzione gameDataDriver.saveGame(g) passando l'oggetto g, ed è proprio questa ultima funzione che si occupa del salvataggio sul database.

Di seguito il codice modificato:

```
73     public JSONObject saveGame(Game game) {
74         try {
75             String time = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
76             JSONObject obj = new JSONObject();
77
78             obj.put("difficulty", game.getDifficulty());
79             obj.put("name", game.getName());
80             obj.put("description", game.getDescription());
81             obj.put("startedAt", time);
82             obj.put("robot", game.getRobot()); // aggiunto
83             obj.put("class", game.getClasse()); // aggiunto
84
85             JSONArray playersArray = new JSONArray();
86             playersArray.put(String.valueOf(game.getPlayerId()));
87
88             obj.put("players", playersArray);
89
90             HttpPost httpPost = new HttpPost("http://t4-g18-app-1:3000/games");
91             StringEntity jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);
92
93             httpPost.setEntity(jsonEntity);
94
95             HttpResponse httpResponse = httpClient.execute(httpPost);
96             int statusCode = httpResponse.getStatusLine().getStatusCode();
97
98             if (statusCode > 299) {
99                 System.err.println(EntityUtils.toString(httpResponse.getEntity()));
100                return null;
101            }
102
103            HttpEntity responseEntity = httpResponse.getEntity();
104            String responseBody = EntityUtils.toString(responseEntity);
105            JSONObject responseObj = new JSONObject(responseBody);
106
107            Integer game_id = responseObj.getInt("id"); // salvo il game id che l'Api mi restituisce
```

Figure 5.7: Funzione saveGame

Questa funzione salva i campi di g di tipo Game in un JSON, e invia il JSON tramite richiesta http al componente T4. La richiesta è di tipo POST su "http://t4-g18-app-1:3000/games".

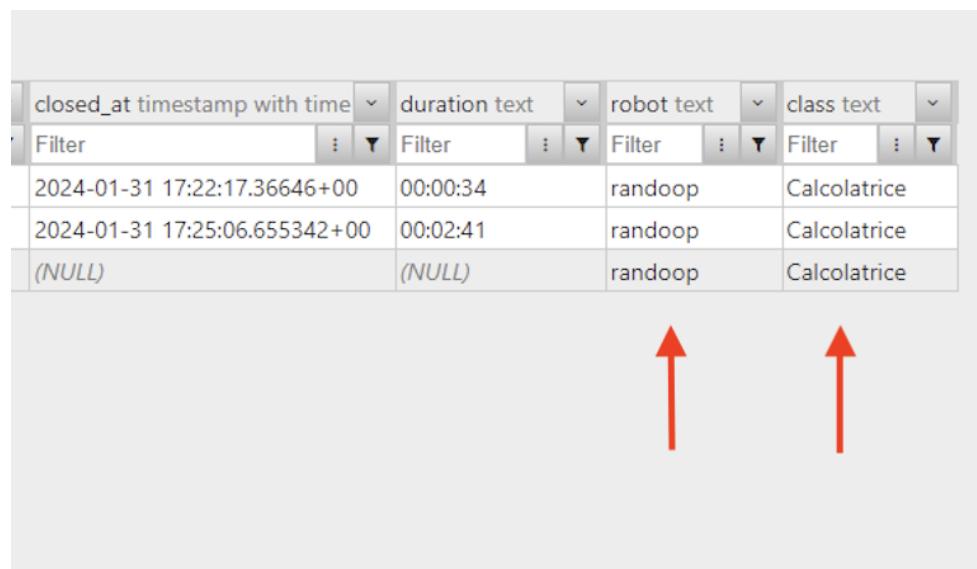
Per quanto riguarda il componente T4, la funzione che siamo andati a modificare è CreateGame, ovvero la funzione che permette di salvare i dati di una partita nel database vero e proprio. Di seguito i cambia-

menti:

```
22 func (gs *Repository) Create(r *CreateRequest) (Game, error) {
23     var (
24         game = model.Game{
25             Name:      r.Name,
26             StartedAt: r.StartedAt,
27             ClosedAt:   r.ClosedAt,
28             Class:     r.Class, // aggiunto
29             Robot:    r.Robot, // aggiunto
30             Difficulty: r.Difficulty, // aggiunto
31             Players:  make([]model.Player, len(r.Players)),
32         }
33     )
34     // detect duplication in player
35     if api.Duplicated(r.Players) {
36         return Game{}, api.ErrInvalidParam
37     }
38
39     for i, player := range r.Players {
40         game.Players[i] = model.Player{
41             AccountID: player,
42         }
43     }
44
45     err := gs.db.Transaction(func(tx *gorm.DB) error {
46         return tx.Create(&game).Error
47     })
48
49     if err != nil {
50         return Game{}, api.MakeServiceError(err)
51     }
52     game.Players = nil
53
54     return fromModel(&game), nil
55 }
```

Figure 5.8: Funzione Create

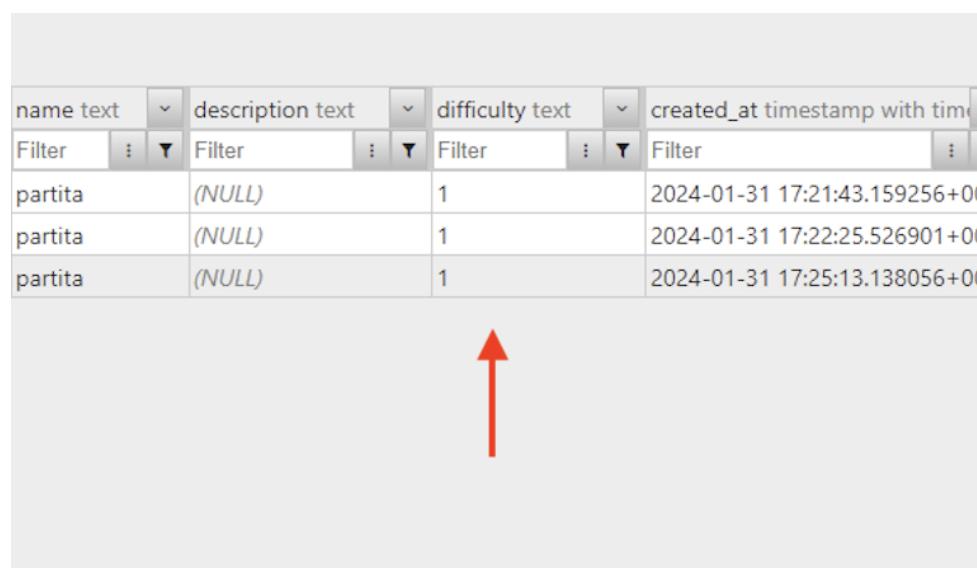
I dati ora vengono finalmente caricati correttamente sul database T4:



closed_at timestamp with time	duration text	robot text	class text
Filter	Filter	Filter	Filter
2024-01-31 17:22:17.36646+00	00:00:34	randoop	Calcolatrice
2024-01-31 17:25:06.655342+00	00:02:41	randoop	Calcolatrice
(NULL)	(NULL)	randoop	Calcolatrice

Figure 5.9: Tabella Games database T4

Il campo "difficulty" inizialmente si era pensato di salvarlo in database come stringa, del tipo "easy", "medium", "hard". In seguito abbiamo pensato di mantenerlo coerente con il concetto di livello del robot, quindi viene salvato come 1, 2 o 3.

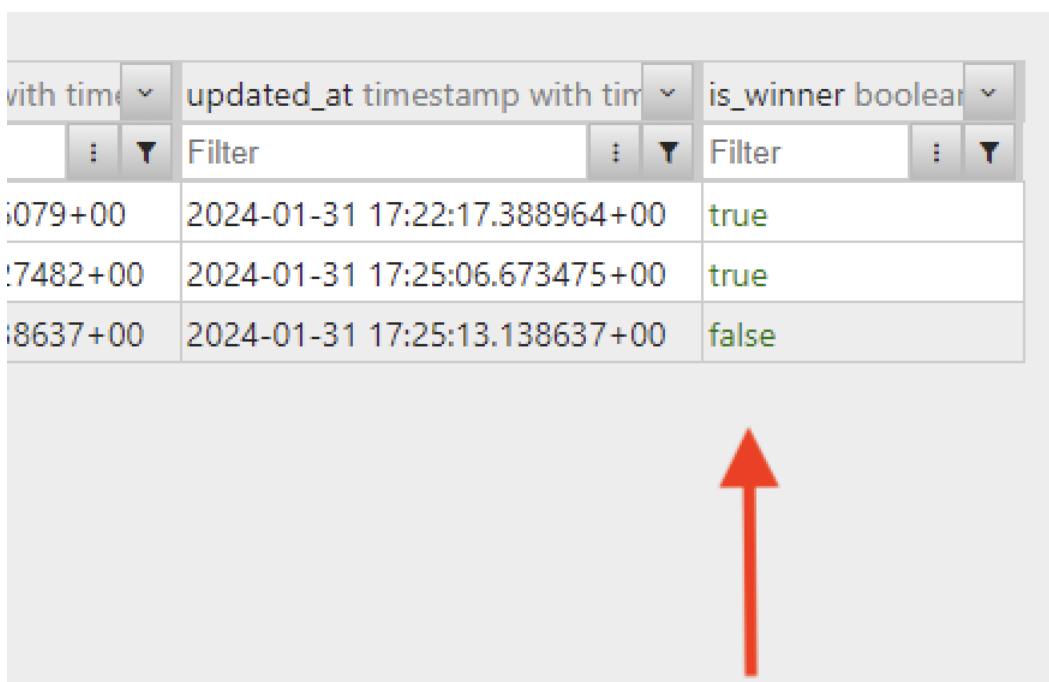


name text	description text	difficulty text	created_at timestamp with time
Filter	Filter	Filter	Filter
partita	(NULL)	1	2024-01-31 17:21:43.159256+00
partita	(NULL)	1	2024-01-31 17:22:25.526901+00
partita	(NULL)	1	2024-01-31 17:25:13.138056+00

Figure 5.10: Tabella Games database T4

5.1.6 Vincitore

Il campo isWinner è presente sia nella tabella Turn che in PlayerGame, tuttavia, originariamente veniva aggiornato solo nella tabella Turn quando un giocatore vinceva il turno. E' stata implementata una funzionalità per assicurare che il campo isWinner in PlayerGame rifletta correttamente il vincitore della partita, garantendo così che entrambi i campi siano aggiornati in modo coerente ogni volta che un giocatore vince il turno e, di conseguenza, la partita.



with time	updated_at timestamp with time	is_winner boolean
	Filter	Filter
!079+00	2024-01-31 17:22:17.388964+00	true
!7482+00	2024-01-31 17:25:06.673475+00	true
!8637+00	2024-01-31 17:25:13.138637+00	false

Figure 5.11: Tabella PlayerGame database T4

5.1.7 LOC Coverage Giocatore e Robot

Il campo Score, che rappresenta la LOC coverage del giocatore, si trova nella tabella Turns.

Figure 5.12: Tabella Turns database T4

Per quanto riguarda il robot, la LOC coverage è registrata nella tabella Robot.

Figure 5.13: Tabella Robots database T4

5.2 Classifica giocatori

Questa sezione si concentra sulla classifica dei giocatori, una funzionalità che permette agli utenti di visualizzare le proprie posizioni relative nel contesto della community di gioco.

5.2.1 Nome Giocatore

Nel contesto del nostro sistema di gestione dei dati, il nome del giocatore è un'informazione cruciale memorizzata nel database T23, non nel T4. Questa distinzione è fondamentale per disaccoppiare i ruoli dei nostri database, dove il T23 è specificamente progettato per conservare le informazioni personali e identificative dei giocatori, mentre il T4 è focalizzato sulla registrazione delle informazioni relative alle partite giocate. L'accesso ai nomi dei giocatori memorizzati in T23 avviene tramite un identificativo unico (ID) che è salvato nella tabella Player di T4. Questo ID funge da ponte tra i due database, permettendo di correlare le partite giocate (e altre informazioni di gioco registrate in T4) con i dati specifici del giocatore conservati in T23.

La separazione delle informazioni tra T23 e T4 è una scelta architettonica che mira a ottimizzare le prestazioni e la sicurezza del sistema. Mentre T23 gestisce dati sensibili e statici relativi ai giocatori, T4 si occupa di dati dinamici legati alle attività di gioco. Questo disaccoppiamento facilita la manutenzione del sistema, la scalabilità e l'integrità dei dati, garantendo che le informazioni personali dei giocatori siano

gestite in modo sicuro e separato dalle loro attività di gioco.

««	id int	email varchar(255)	name varchar	password varchar
	Filter	Filter	Filter	Filter
	1	giovanni@gmail.com	giovanni	\$2a\$10\$qw410jkc

Figure 5.14: Tabella Studenti database T23

5.2.2 Partite Giocate

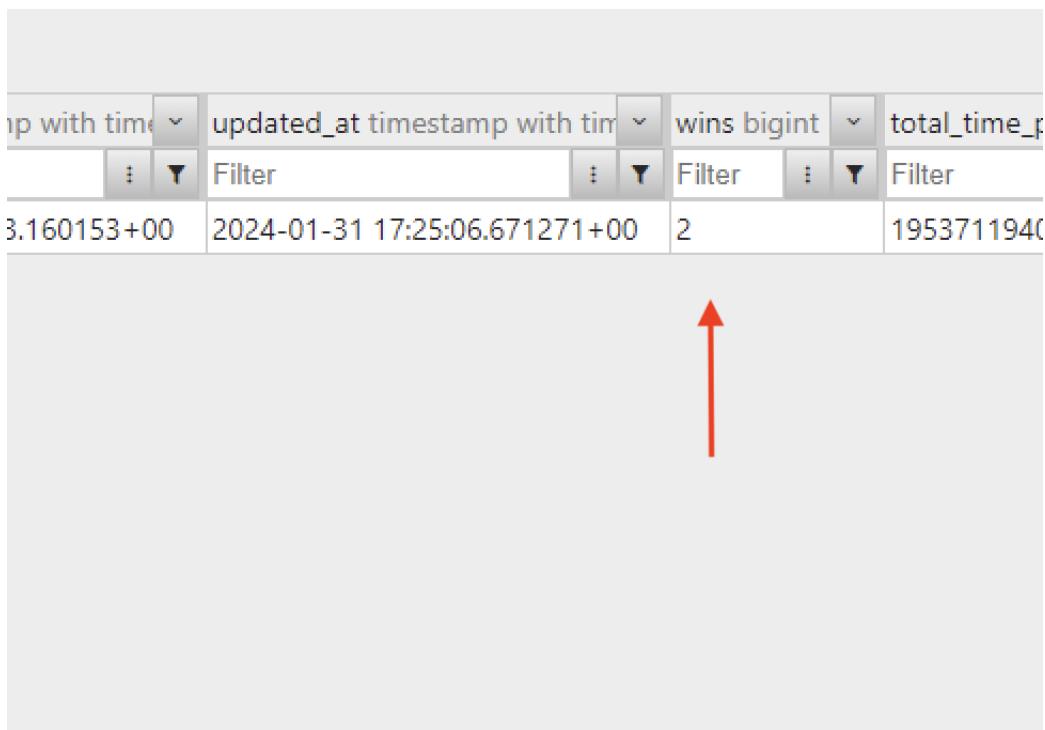
Il tracciamento delle partite giocate avviene attraverso la tabella PlayerGames: All’interno del nostro ecosistema di gioco, il conteggio delle partite giocate da ogni giocatore viene effettuato mediante un’accurata analisi della tabella PlayerGames. Questa tabella è progettata per registrare ogni partita giocata, associandola direttamente ai giocatori che hanno partecipato, attraverso una relazione chiave tra PlayerID e GameID. Ogni volta che un giocatore inizia una partita, una nuova voce viene creata nella tabella PlayerGames, contenente il suo PlayerID e il GameID corrispondente alla partita appena giocata.

Per determinare il numero totale di partite giocate da un giocatore specifico, il sistema conta il numero di voci presenti nella tabella Play-

erGames che corrispondono al PlayerID di quel giocatore. In pratica, questo significa che per ogni giocatore, l'ID unico associato al suo profilo viene utilizzato come chiave per aggregare tutte le partite a cui ha partecipato. Questo metodo fornisce una misura precisa e diretta dell'attività di gioco di ciascun utente.

5.2.3 Partite vinte

Abbiamo introdotto un meccanismo per registrare le vittorie accumulate dai giocatori nel corso del loro impegno nella piattaforma. Questo meccanismo si basa sull'utilizzo di un campo specifico, denominato Wins, all'interno della tabella Player nel database T4. Il campo Wins è progettato per tenere conto delle partite vinte da ogni giocatore.



id with time	updated_at timestamp with time	wins bigint	total_time
	Filter	Filter	Filter
3.160153+00	2024-01-31 17:25:06.671271+00	2	1953711940

Figure 5.15: Tabella Players database T23

Ogni volta che un giocatore vince una partita, il sistema procede automaticamente con l'aggiornamento del campo Wins per quel giocatore specifico. Questo processo di aggiornamento si attiva al termine di una partita, quando viene determinato il vincitore. A seguito della conferma della vittoria, il sistema identifica il PlayerID del giocatore vincente e incrementa di uno il valore del campo Wins associato a quel giocatore nella tabella Player. La funzione modificata è la seguente:

```
82 func (tr *Repository) Update(id int64, r *UpdateRequest) (Turn, error) {
83
84     var (
85         turn model.Turn = model.Turn{ID: id}
86         err   error
87     )
88
89     // Ottiene lo stato attuale del turno prima dell'aggiornamento
90     var currentTurn model.Turn
91     err = tr.db.First(&currentTurn, id).Error
92     if err != nil {
93         return Turn{}, api.MakeServiceError(err)
94     }
95
96     // Aggiorna il turno con i nuovi valori
97     err = tr.db.Model(&turn).Updates(r).Error
98     if err != nil {
99         return Turn{}, api.MakeServiceError(err)
100    }
101
102    // Controlla se il turno è stato aggiornato
103    // !currentTurn.IsWinner serve per vedere se prima non era già vinto il turno, visto che update la chiamiamo per il momento s
104    // in futuro potrebbe essere utile
105    if r.IsWinner /* && !currentTurn.IsWinner */ {
106        // Incrementa il conteggio delle vittorie del giocatore
107        err := tr.db.Model(&model.Player{}).Where("ID = ?", currentTurn.PlayerID).Update("Wins", gorm.Expr("Wins + ?", 1)).Error
108        if err != nil {
109            log.Printf("Errore nell'aggiornamento delle vittorie per il giocatore con ID %d: %v\n", currentTurn.PlayerID, err)
110        }
111    }
112
113 }
```

Figure 5.16: Funzione UpdateTurn

Chapter 6

Requisito R11

In questo capitolo, ci dedicheremo ad approfondire e fornire una giustificazione dettagliata delle modifiche implementate nel nostro progetto, come precedentemente accennato nel capitolo 4. Esamineremo l'importanza di ciascuna modifica, evidenziando come queste abbiano contribuito a migliorare il sistema in linea con gli obiettivi prefissati. Questo capitolo serve quindi a fornire una comprensione completa del valore aggiunto da queste modifiche, in termini di funzionalità.

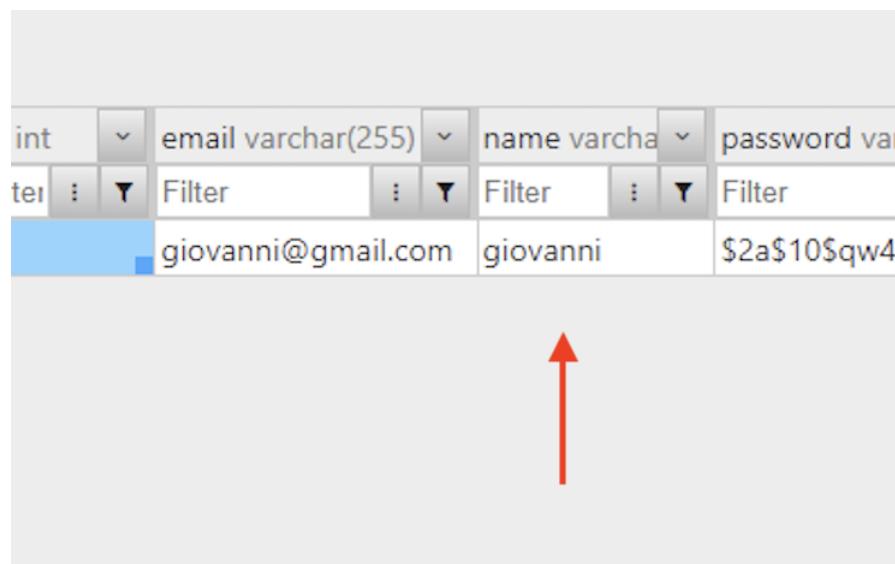
6.1 Elenco dei giocatori iscritti

L'obiettivo di questa sezione è descrivere in dettaglio le modifiche e le implementazioni effettuate per la gestione di un elenco dei giocatori iscritti.

6.1.1 Nome, cognome, corso di studi

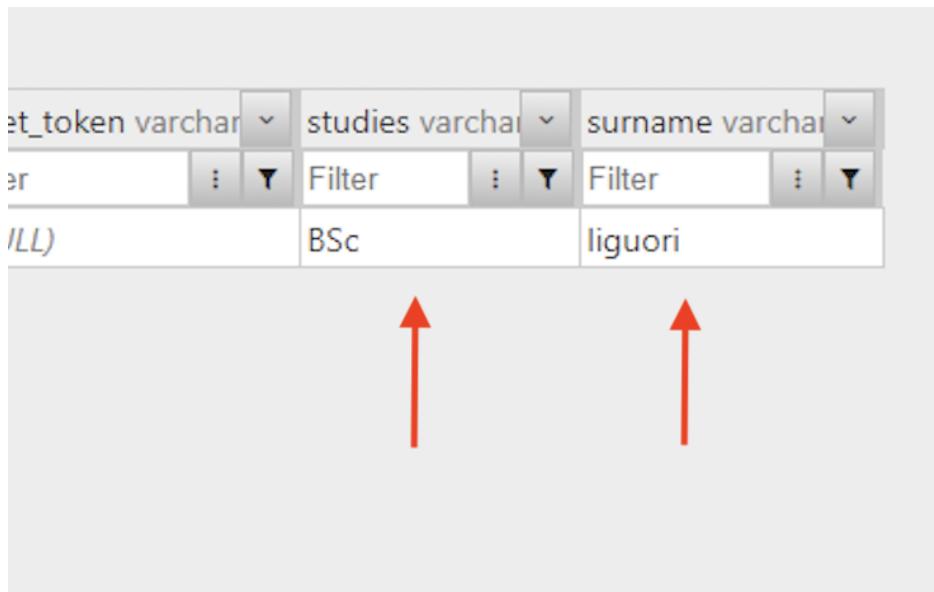
Le informazioni fondamentali, quali il nome, il cognome e il corso di studi degli utenti, vengono raccolte al momento della registrazione utente e successivamente salvate nel database del componente T1. Questo processo assicura che i dati essenziali siano prontamente disponibili per interrogazioni future.

Queste informazioni immesse al momento della registrazione, oltre a essere salvate, fungono da punto di partenza per la creazione dell'elenco dei giocatori iscritti. Di seguito screen del database T1:



int	email varchar(255)	name varchar	password var
ter	Filter	Filter	Filter
	giovanni@gmail.com	giovanni	\$2a\$10\$qw4

Figure 6.1: Nome in database T1



set_token	studies	surname
er	Filter	Filter
ILL)	BSc	Iliguori

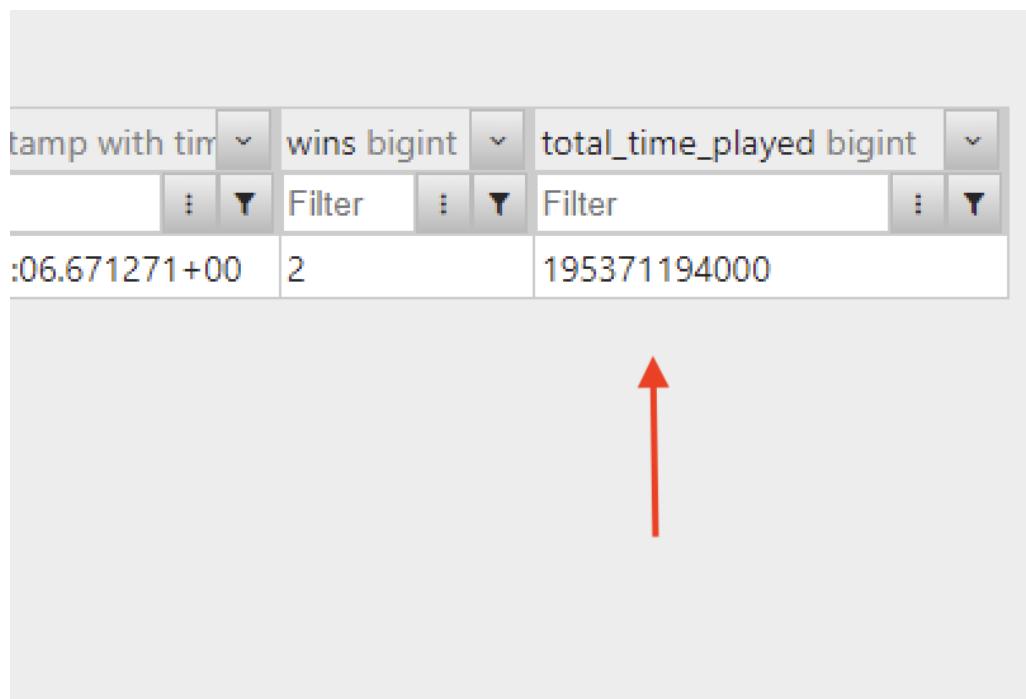
Figure 6.2: Cognome e corso di studi in database T1

6.1.2 Numero di classi testate e Numero di partite giocate

Approfondendo le funzionalità richieste dal cruscotto, si evidenzia la necessità di correlare il numero di classi testate direttamente con il numero di partite giocate, una metrica già esaminata nel capitolo precedente. La correlazione tra le due metriche è significativa: ogni partita giocata implica il test di una classe, e viceversa. Di conseguenza, il termine "numero di classi testate" può essere interpretato come sinonimo di "numero di partite giocate", offrendo un'interpretazione univoca. Questo dato, il numero di partite giocate, è stato già esplorato nel capitolo precedente e non necessita di ulteriori approfondimenti in questo contesto.

6.1.3 Tempo totale delle partite giocate

L'accurata misurazione e registrazione del tempo totale impiegato nelle partite giocate rappresenta un aspetto fondamentale per l'analisi dettagliata dell'attività degli utenti all'interno del sistema. Per consentire tale tracciamento, è stato introdotto un nuovo campo nella tabella player all'interno del database T4. Questo campo è dedicato alla conservazione del tempo complessivo che ogni giocatore ha speso giocando.



tamp with tim	wins bigint	total_time_played bigint
:06.671271+00	2	195371194000

Figure 6.3: Tabella Players in T4

Al fine di calcolare tale metrica, è stata sviluppata una funzione specifica che opera all'interno del sistema. Questa funzione è progettata per registrare la durata di ogni partita nel momento in cui essa si conclude. Tuttavia, a causa di limitazioni tecniche legate alla granularità e

al tipo di dato che il database T4 è in grado di gestire, il tempo viene memorizzato in nanosecondi. Sebbene questa unità di misura possa sembrare eccessivamente precisa e poco intuitiva per un'analisi umana diretta, essa è stata l'unica opzione praticabile data la restrizione di salvataggio del sistema.

```

53 func (gs *Repository) Update(id int64, r *UpdateRequest) (Game, error) {
54
55     var (
56         game model.Game = model.Game{ID: id}
57         err error
58     )
59
60     // Aggiorna il gioco con i nuovi valori
61     err = gs.db.Model(&game).Updates(r).Error
62     if err != nil {
63         return Game{}, api.MakeServiceError(err)
64     }
65
66     // Ricarica il gioco per ottenere i dati aggiornati, inclusi StartedAt e ClosedAt
67     err = gs.db.First(&game, id).Error
68     if err != nil {
69         return Game{}, api.MakeServiceError(err)
70     }
71
72     // Controlla se StartedAt che ClosedAt sono non nulli
73     if game.StartedAt != nil && game.ClosedAt != nil {
74
75         durationToAddPlayer := game.ClosedAt.Sub(*game.StartedAt)
76
77         // Trova il Round associato al Game
78         var round model.Round
79         err := gs.db.Where("game_id = ?", id).First(&round).Error
80         if err != nil {
81             fmt.Println("Errore nella ricerca del Round:", err)
82         }
83
84         // Trova il Turn associato al Round
85         var turn model.Turn
86         err = gs.db.Where("round_id = ?", round.ID).First(&turn).Error
87         if err != nil {
88             fmt.Println("Errore nella ricerca del Turn:", err)
89         }
90
91         // Trova il Player associato al Turn
92         var player model.Player
93         err = gs.db.First(&player, turn.PlayerID).Error
94         if err != nil {
95             fmt.Println("Errore nella ricerca del Player:", err)
96         }
97
98         // Aggiorna TotalTimePlayed per il Player
99         player.TotalTimePlayed += durationToAddPlayer
100        err = gs.db.Model(&player).Update("TotalTimePlayed", player.TotalTimePlayed).Error
101        if err != nil {
102            fmt.Println("Errore nell'aggiornamento di TotalTimePlayed:", err)
103        }
104    }
105
106
107    return fromModel(&game), api.MakeServiceError(err)
108 }
```

Figure 6.4: Funzione UpdateGame

6.2 Elenco delle classi disponibili

Questa sezione si dedica all'esplorazione delle modifiche e delle implementazioni intraprese per la creazione e la gestione di un elenco dettagliato delle classi disponibili nel sistema. L'elenco è progettato per fornire agli amministratori del sistema un quadro immediato delle risorse di gioco che gli utenti possono utilizzare, una funzionalità cruciale per la supervisione e l'ottimizzazione delle dinamiche interne del sistema.

6.2.1 Nome

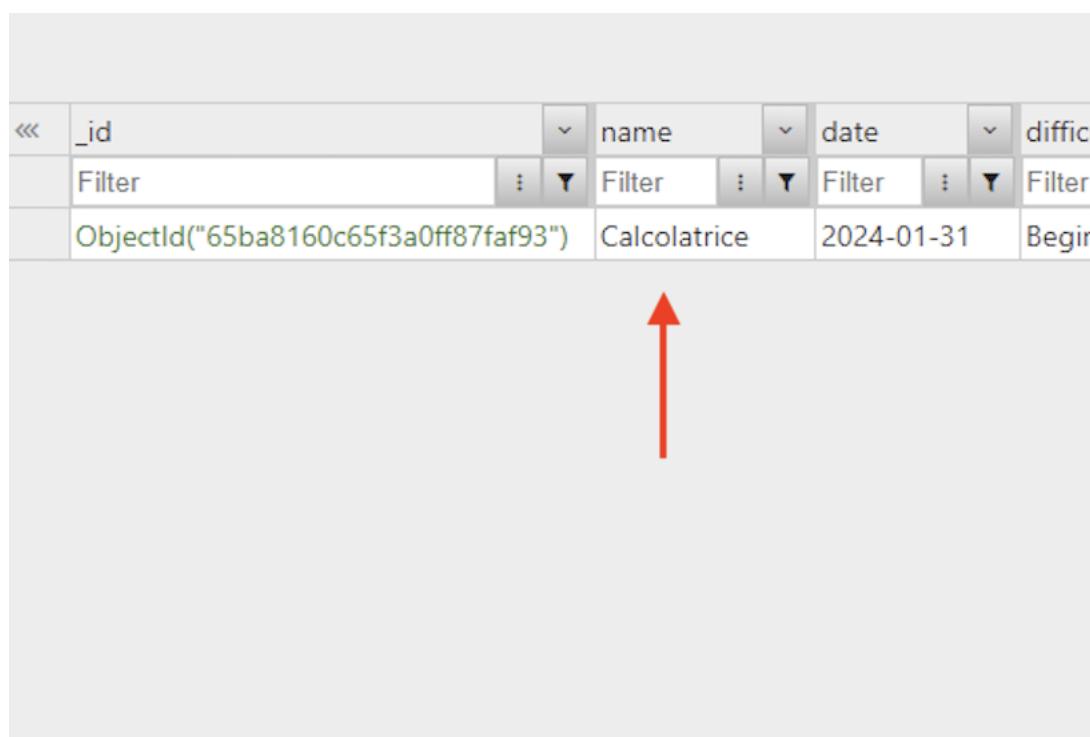
All'interno del database T1, il nome rappresenta una caratteristica distintiva di ogni classe, sebbene non funga da chiave primaria. Questo dato è essenziale per l'identificazione intuitiva di una classe da parte degli amministratori e degli utenti. Nonostante l'importanza del nome, ogni classe è effettivamente identificata all'interno del database da un ID univoco, il quale serve come chiave primaria per le operazioni di database, assicurando l'integrità e l'univocità delle informazioni.

Il nome di una classe, accanto al suo ID, viene registrato nel database T1 non appena la classe viene creata o aggiunta al sistema. Questa procedura garantisce che ogni nuova classe sia resa prontamente disponibile per essere inclusa nell'elenco delle classi, e che i dati siano mantenuti in uno stato coerente e attuale.

L'associazione tra l'ID univoco e il nome della classe permette di

evitare problemi di ambiguità che potrebbero emergere in presenza di classi con nomi simili o identici. In questo modo, gli amministratori possono sfruttare l'ID per operazioni tecniche specifiche, mentre il nome rimane il riferimento principale durante la consultazione dell'elenco delle classi.

Di seguito uno screen del database T1:



«`	_id	name	date	diffic
	Filter	Filter	Filter	Filter
	ObjectId("65ba8160c65f3a0ff87faf93")	Calcolatrice	2024-01-31	Begir

Figure 6.5: Tabella classUT database T1

6.3 Altre considerazioni

Nel contesto di un sistema informativo in continua evoluzione, è essenziale porre attenzione ad alcune considerazioni che emergono dalla gestione dei dati e dalla loro rappresentazione.

6.3.1 Elenco giocatori iscritti

Per quanto riguarda l'elenco dei giocatori iscritti, è importante sottolineare che il nostro ruolo si limita alla garantire la disponibilità e l'integrità dei dati all'interno del database. L'ordine con cui questi dati vengono poi presentati e organizzati nelle varie modalità di visualizzazione non è di nostra competenza. Ci assicuriamo che le informazioni siano accuratamente raccolte e memorizzate in modo che, quando si rende necessario un ordinamento o una particolare elaborazione, il sistema o gli amministratori possano effettuarlo sulla base di criteri che meglio si adattano alle esigenze del momento.

6.3.2 Elenco classi disponibili

Nel database T1, dove sono memorizzate le informazioni relative alle classi disponibili, si è optato per includere dati che forniscono una panoramica completa sulle caratteristiche di ogni classe. Sebbene le metriche come Linee di Codice (LOC) e il numero di metodi non siano presenti, altre informazioni come la difficoltà, la descrizione e la categoria sono disponibili e offrono un valore aggiunto significativo. Queste informazioni consentono agli utenti di comprendere meglio la natura e le sfide associate a ogni classe, contribuendo così a una migliore selezione e utilizzo delle stesse all'interno del sistema.

6.3.3 Classifica giocatori

La classifica dei giocatori è stata già trattata in maniera approfondita nel capitolo precedente. Pertanto, si rimanda a quella sezione per una discussione dettagliata riguardo la metodologia di calcolo e la presentazione dei dati relativi alla performance degli utenti. In sintesi, la classifica rappresenta un mezzo efficace per stimolare la competizione leale tra gli utenti e per promuovere un engagement più profondo con il sistema.

Chapter 7

Futuri sviluppi : Rimozione Round

7.1 Introduzione

Nella progettazione di sistemi software, specialmente nelle applicazioni di gioco, l'efficienza e la scalabilità sono obiettivi fondamentali. Il costante monitoraggio e rifinitura dell'architettura del database possono fornire miglioramenti significativi sia in termini di prestazioni che di manutenzione. Questo capitolo discute una tale strategia di ottimizzazione: la rimozione dell'entità **Round** dai nostri schemi di gioco e dal database.

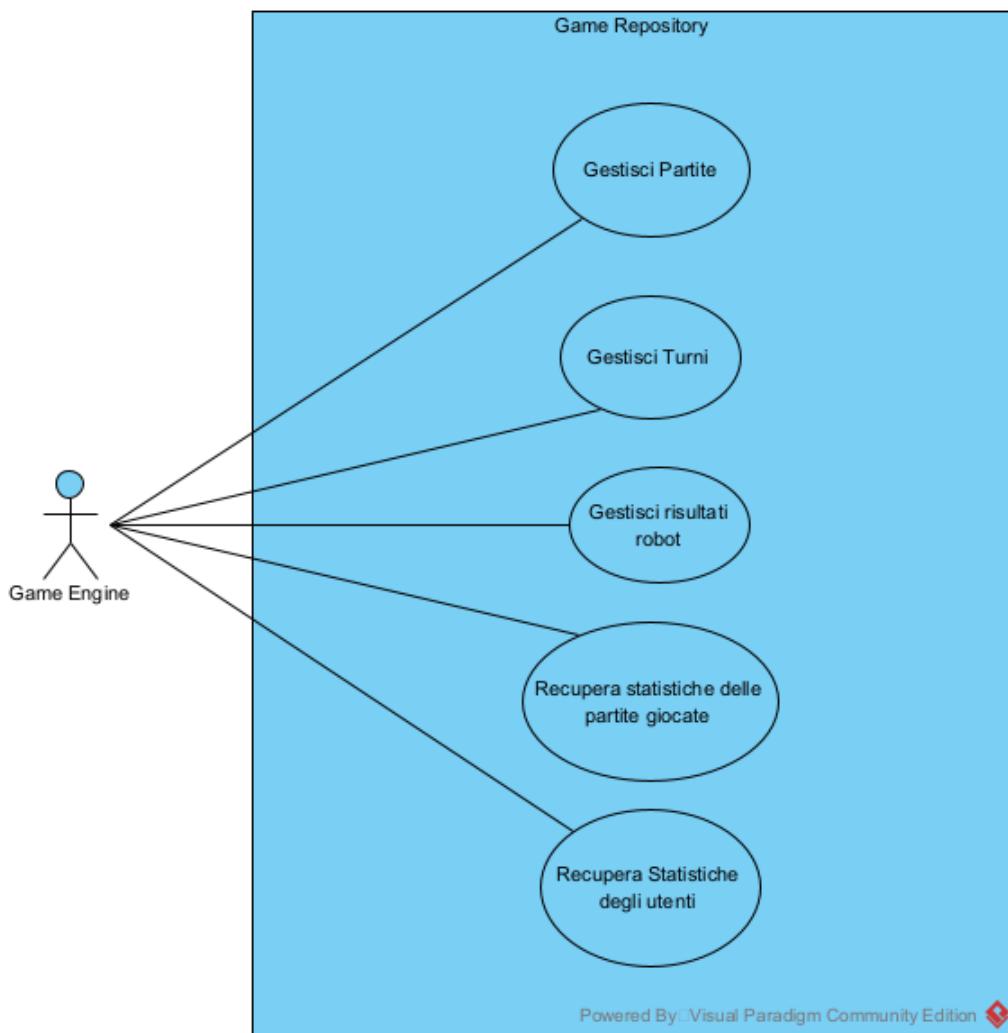


Figure 7.1: Use Case Diagram

Nel diagramma ER (7.2) si può notare che i campi e le relazioni che legavano Round con le altre entità sono stati rescissi. Le responsabilità che prima aveva sono state affidate a Turn, che ora si aggiorna in base agli eventi del gioco e tiene traccia delle azioni del giocatore.

Adesso Game non ha più un array di Rounds ma un array di Turns che registra il numero di turni effettuati dal giocatore.

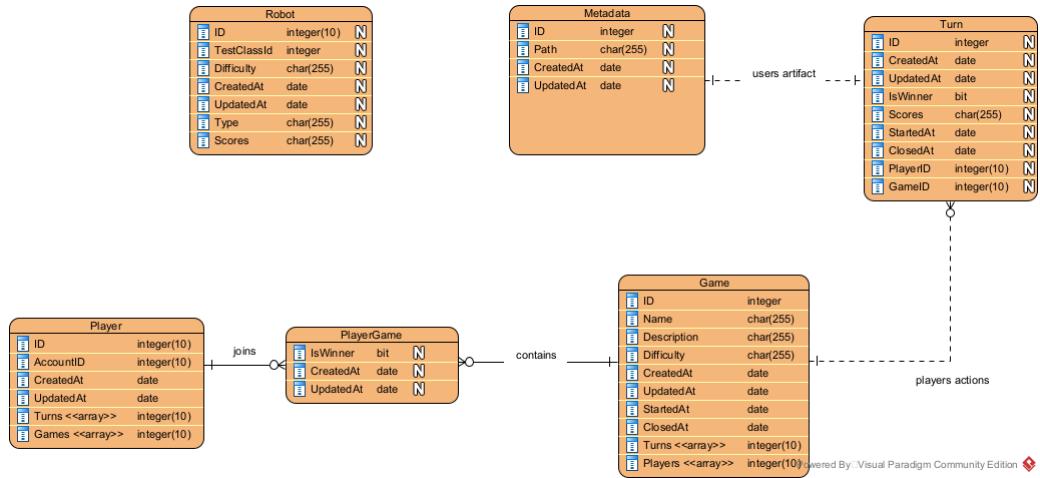


Figure 7.2: Diagramma ER

7.1.1 Motivazioni della rimozione di Round

L'entità Round è stata identificata come un candidato alla rimozione per diversi motivi. Primo tra questi è il potenziale di semplificazione dei processi di gioco, riducendo la granularità delle sessioni di gioco senza compromettere la qualità dell'esperienza utente. Infatti, il Round, originariamente concepito come una divisione logica del gioco in fasi distinte, è diventato effettivamente intercambiabile con Turn, rendendo obsoleta la sua esistenza autonoma. In secondo luogo, la sua rimozione dallo schema del database riduce il sovraccarico di storage e le operazioni di I/O, il che è particolarmente vantaggioso in un ambiente con un alto volume di transazioni e operazioni di gioco.

7.2 Modifiche alla struttura del progetto

Come si può notare dai seguenti diagrammi la struttura del T4 subirebbe dei cambiamenti in seguito alla rimozione di Round. Questo implica che il sistema diventa più semplice da mantenere stabile e da monitorare.

L’architettura della repository 7.3 e il system domain model 7.4 ne giovanono sensibilmente di questo cambiamento, in quanto permette un alleggerimento del sistema e quindi la possibilità di implementare nuove funzionalità che non minacciano la fluidità del gioco, come ad esempio una modalità multigiocatore basata su più turni o una sfida incrementale con dei robot che aumentano la difficoltà man mano che si vince (modalità arcade).

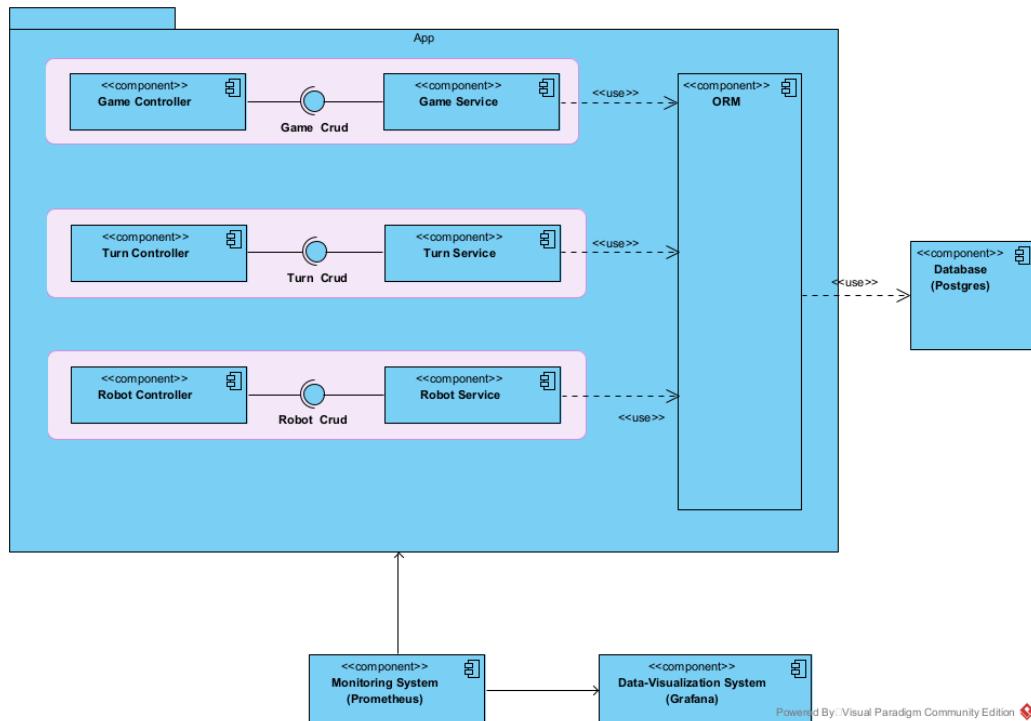


Figure 7.3: Architettura Game Repository senza Round

CHAPTER 7. FUTURI SVILUPPI : RIMOZIONE ROUND

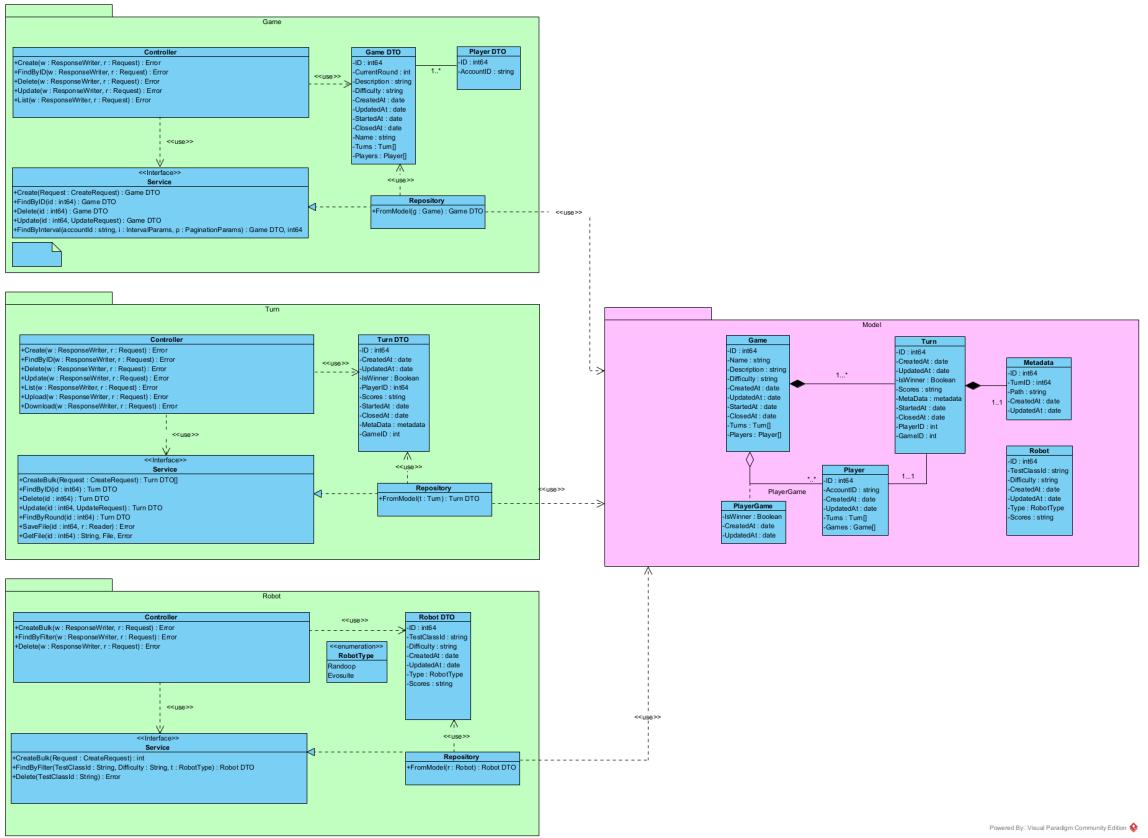


Figure 7.4: System Domain Model

Come si nota nel diagramma di flusso 7.5 adesso Turn diventa la principale entità su cui si appoggia Game. La logica è che in una partita ci possono più turni, in base al numero di giocatori.

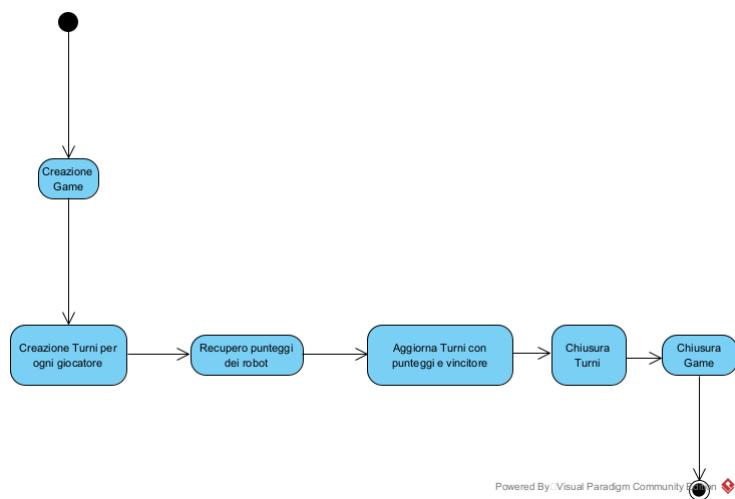


Figure 7.5: Diagramma del flusso della partita

7.3 Aggiornamenti ai servizi REST

Con la rimozione di Round, i servizi REST esistenti per il gioco e i turni subirebbero cambiamenti significativi. Gli aggiornamenti verrebbero attuati per semplificare le interazioni con il database e ridurre il traffico di rete.

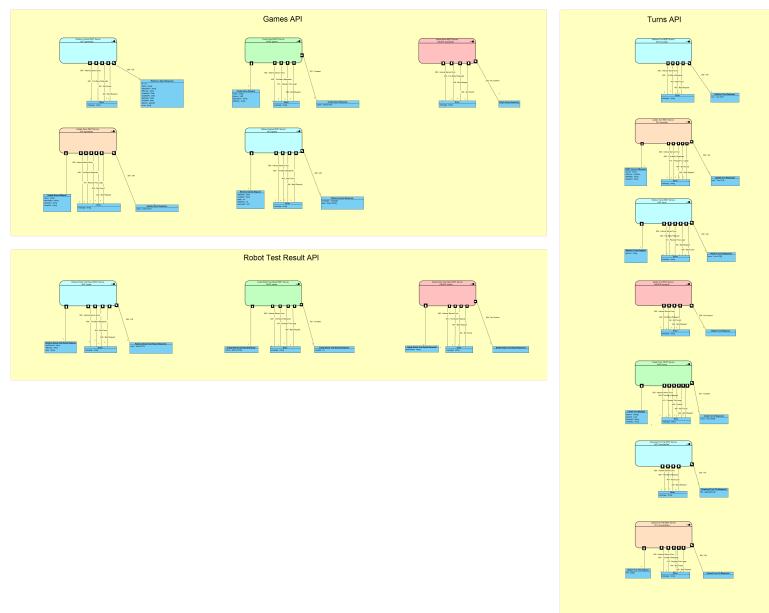


Figure 7.6: API Diagram

7.3.1 Retrieve a Game REST Service

Il servizio **Retrieve a Game** verrebbe aggiornato per riflettere la rimozione di 'Round'. Fornisce informazioni sui turni direttamente all'interno della risposta del gioco, eliminando la necessità di chiamate API aggiuntive per recuperare i dati del round.

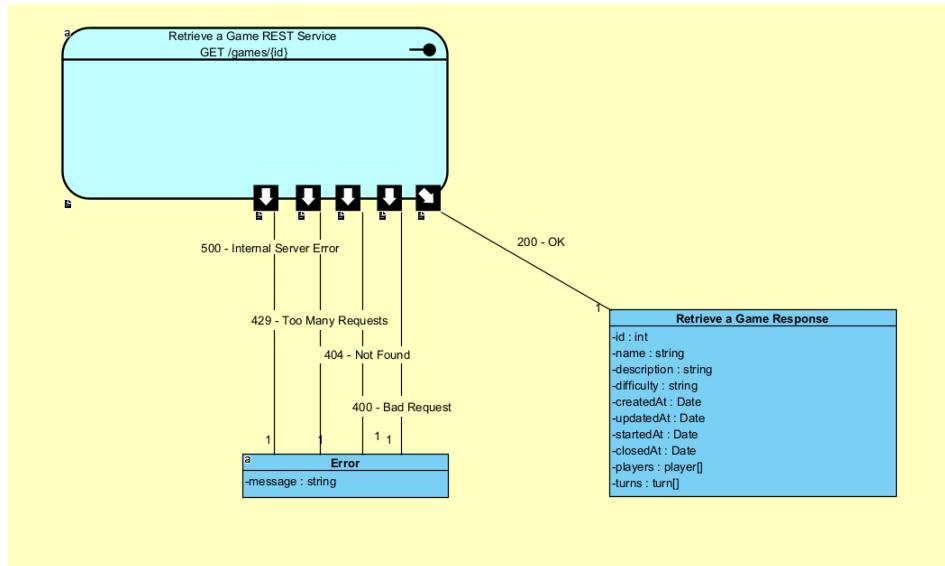


Figure 7.7: Funzione Retrieve Game

7.3.2 Update Game REST Service

Il servizio per aggiornare i dettagli di una partita rifletterebbe direttamente gli aggiornamenti dei turni, evitando qualsiasi confusione potenzialmente derivante dalla precedente struttura che includeva Round.

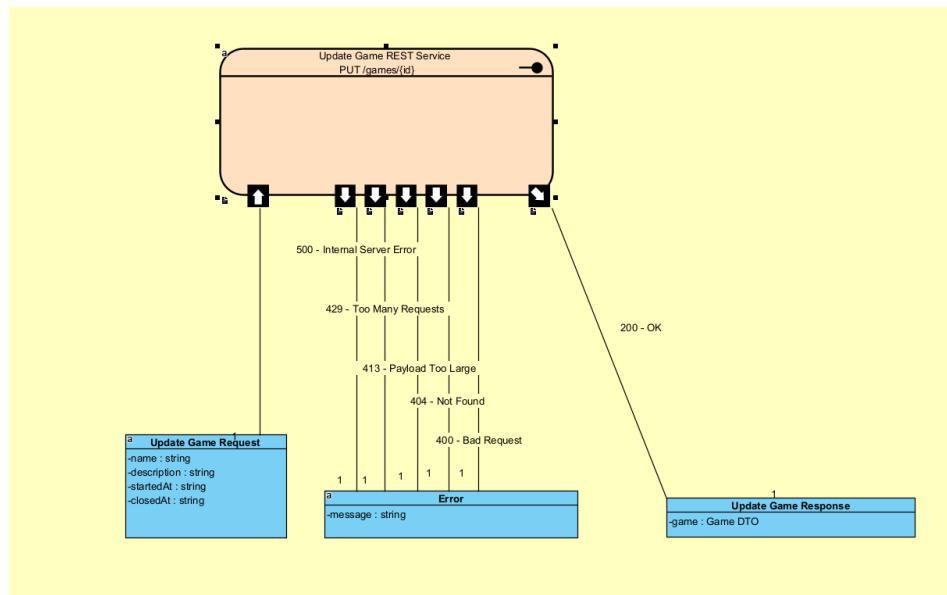


Figure 7.8: Funzione Update Game

7.3.3 Create Turns REST Service

Il servizio **Create Turns** gestirebbe in modo più efficiente la creazione e l'aggiornamento dei turni, con un focus sulla reattività e sull'efficienza.

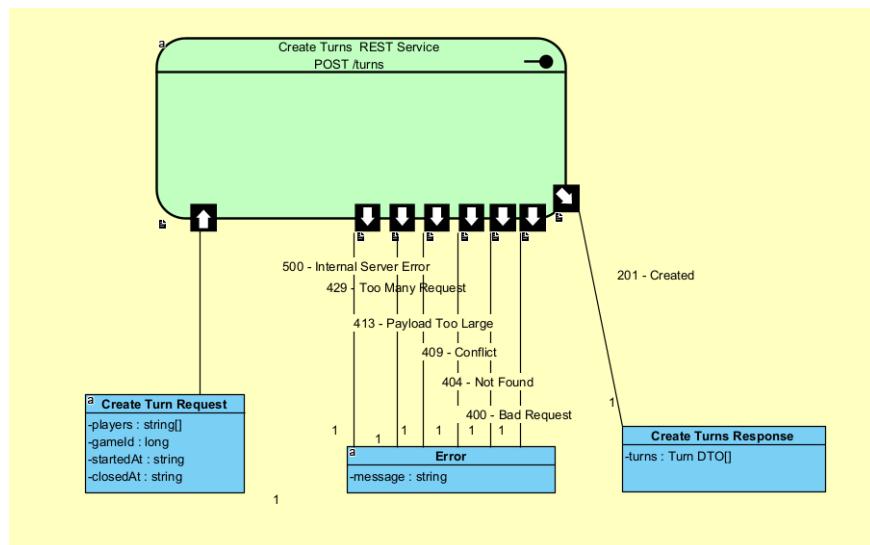


Figure 7.9: Modifiche alla funzione Create di Turn

7.3.4 Retrieve Turns REST Service

Il servizio **Retrieve Turns** continuerebbe a funzionare come prima, ma potrebbe gestire un carico di lavoro più leggero grazie all'eliminazione di 'Round'. Questo si traduce in un miglioramento delle prestazioni complessive e in una riduzione del carico sul server.

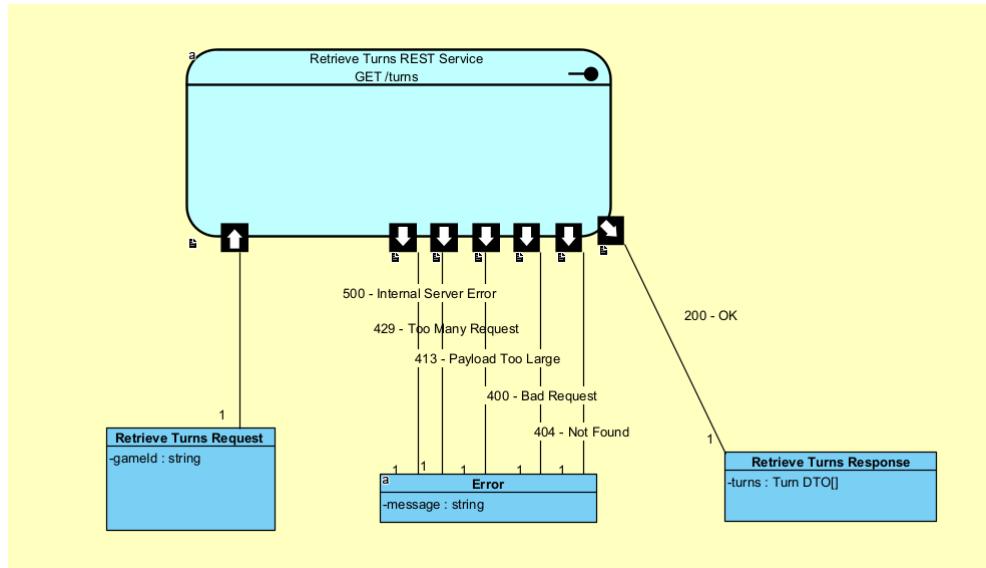


Figure 7.10: Modifiche alla funzione Retrieve di Turn

7.4 Conclusioni

La rimozione di Round rappresenta un passo in avanti verso un'architettura di gioco più snella e focalizzata. Questo capitolo ha esplorato gli aggiornamenti necessari e fornito una visione olistica di come queste modifiche influenzano l'interazione dell'utente, le prestazioni del server, e la manutenzione del sistema.

Questa strategia di ottimizzazione non è stata implementata poiché

sconvolgerebbe l'architettura mettendo in difficoltà gli altri gruppi del progetto. Nel contesto della progettazione dei sistemi software, soprattutto nelle applicazioni di gioco, l'efficienza e la scalabilità sono obiettivi fondamentali. La mancanza di coordinamento con gli altri gruppi potrebbe portare a conflitti nell'integrazione delle modifiche. Se il nostro team procedesse con questa modifica senza coinvolgere adeguatamente gli altri gruppi, potremmo trovarci in una situazione in cui le nostre modifiche entrano in conflitto con le loro, causando ritardi e complicazioni nel processo di sviluppo complessivo.

Chapter 8

Consigli e Risoluzione dei Problemi Installazione

8.1 Introduzione

Il processo di sviluppo e installazione di un progetto di gioco può presentare diverse sfide tecniche. Affrontare queste sfide richiede una comprensione dettagliata degli strumenti e delle tecnologie impiegate. In questo capitolo, esploreremo approfonditamente come superare i problemi comuni incontrati durante l'installazione e lo sviluppo del nostro gioco, fornendo consigli pratici e procedure dettagliate.

8.2 Compilazione del Progetto con Maven

8.2.1 Cos'è Maven?

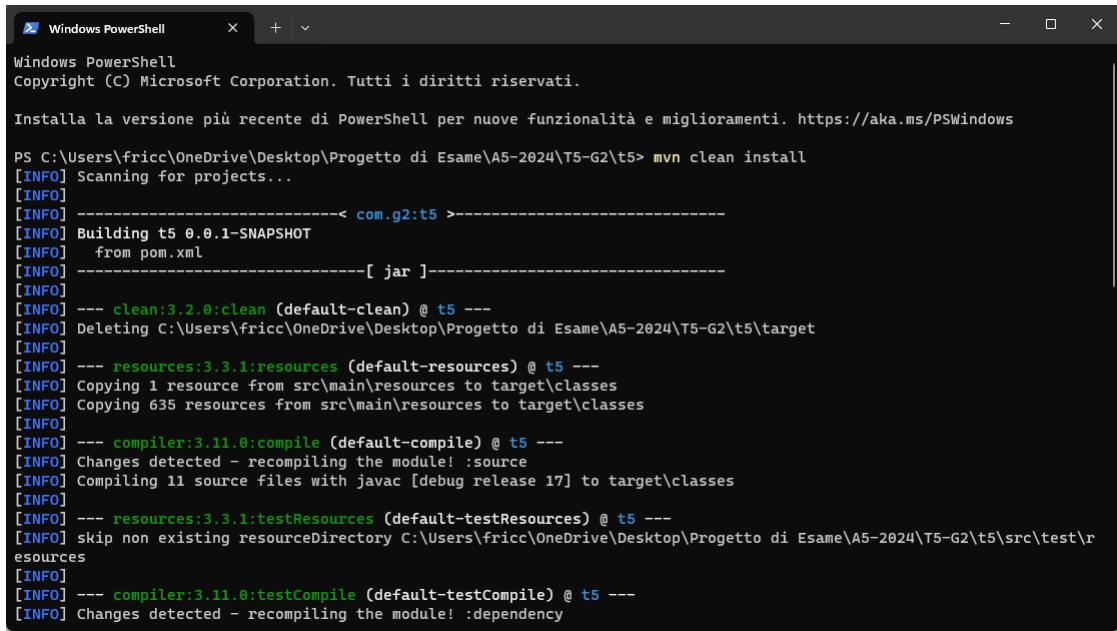
Apache Maven è uno strumento di gestione di progetti software basato sul concetto di Project Object Model (POM). La sua capacità di gestire dipendenze e l'automazione di build rende Maven essenziale per progetti complessi.

8.2.2 Utilizzo di Maven per la Compilazione

Per compilare il progetto, è necessario eseguire il comando ‘mvn clean install’ all’interno della directory del progetto dove sono state effettuate le modifiche. Questo processo comporta due fasi principali:

- **Clean:** Rimuove i file generati in precedenti compilazioni, assicurando che la build parta da una base pulita.
- **Install:** Compila il codice sorgente del progetto, esegue i test e pacchetta il codice compilato in formati distribuibili, come JAR o WAR.

L’importanza di questo comando risiede nella sua capacità di garantire che tutte le modifiche siano correttamente integrate e che il progetto sia libero da errori di compilazione prima di procedere con ulteriori fasi di sviluppo o deployment.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.

Installa la versione più recente di PowerShell per nuove funzionalità e miglioramenti. https://aka.ms/PSWindows

PS C:\Users\fricc\OneDrive\Desktop\Progetto di Esame\A5-2024\T5-G2\t5> mvn clean install
[INFO] Scanning for projects...
[INFO] [INFO] < com.g2:t5 >
[INFO] Building t5 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] [INFO] [ jar ]
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ t5 ---
[INFO] Deleting C:\Users\fricc\OneDrive\Desktop\Progetto di Esame\A5-2024\T5-G2\t5\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ t5 ---
[INFO] Copying 1 resource from src\main\resources to target\classes
[INFO] Copying 635 resources from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ t5 ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 11 source files with javac [debug release 17] to target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ t5 ---
[INFO] skip non existing resourceDirectory C:\Users\fricc\OneDrive\Desktop\Progetto di Esame\A5-2024\T5-G2\t5\src\test\resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ t5 ---
[INFO] Changes detected - recompiling the module! :dependency
```

Figure 8.1: Utilizzo del comando mvn clean install

8.3 Gestione di Docker per una Disinstallazione Pulita

8.3.1 Il Ruolo di Docker

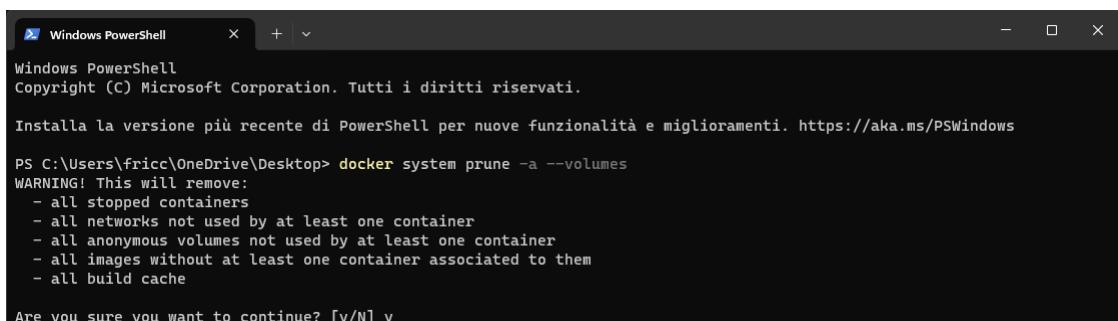
Docker gioca un ruolo cruciale nell'esecuzione del gioco. Assicura che il gioco funzioni consistentemente attraverso diverse piattaforme e configurazioni.

8.3.2 Procedura per una Disinstallazione Pulita

Per mantenere un ambiente di sviluppo pulito e ottimizzato, è fondamentale eseguire periodicamente il comando ‘docker system prune -a –volumes’. Questo comando rimuove:

- Tutti i container inutilizzati.
- Immagini non utilizzate.
- Reti non utilizzate.
- Volumi non collegati a container attivi.

Prima di eseguire questo comando, assicurarsi di aver salvato i dati importanti, poiché la sua esecuzione comporta la perdita di dati non salvati nei container e nei volumi.



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "docker system prune -a --volumes" is being run. The output shows a warning message: "WARNING! This will remove:" followed by a list of items: "all stopped containers", "all networks not used by at least one container", "all anonymous volumes not used by at least one container", "all images without at least one container associated to them", and "all build cache". At the bottom of the window, the question "Are you sure you want to continue? [y/N] y" is displayed.

Figure 8.2: Terminale con il comando per una disinstallazione pulita

8.4 Debugging con gli Strumenti di Sviluppo di Chrome

8.4.1 Il Tasto F12 e gli Strumenti di Sviluppo

Premendo **F12** su Google Chrome, si aprono gli strumenti di sviluppo, indispensabili per il debugging del gioco. Questi strumenti offrono una varietà di funzionalità:

- **Console:** Per eseguire comandi JavaScript e visualizzare messaggi di log.
- **Network:** Per monitorare le richieste di rete, visualizzare i dettagli delle chiamate API, risorse caricate, e analizzare le prestazioni.

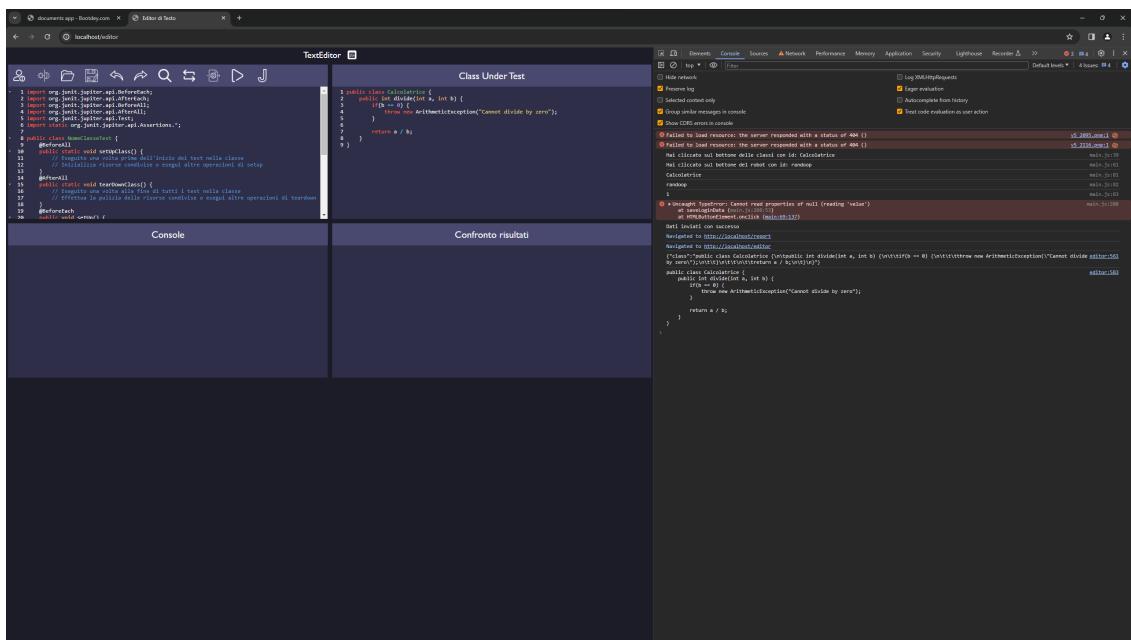


Figure 8.3:

8.4.2 Analisi delle Chiamate API

La sezione Network è particolarmente utile per analizzare le chiamate API. Qui, gli sviluppatori possono vedere i dettagli di ogni richiesta e risposta, compresi header, payload, e codici di stato. Questo è cruciale per identificare errori o colli di bottiglia nelle comunicazioni tra il client e il server.

CHAPTER 8. CONSIGLI E RISOLUZIONE DEI PROBLEMI INSTALLAZIONE

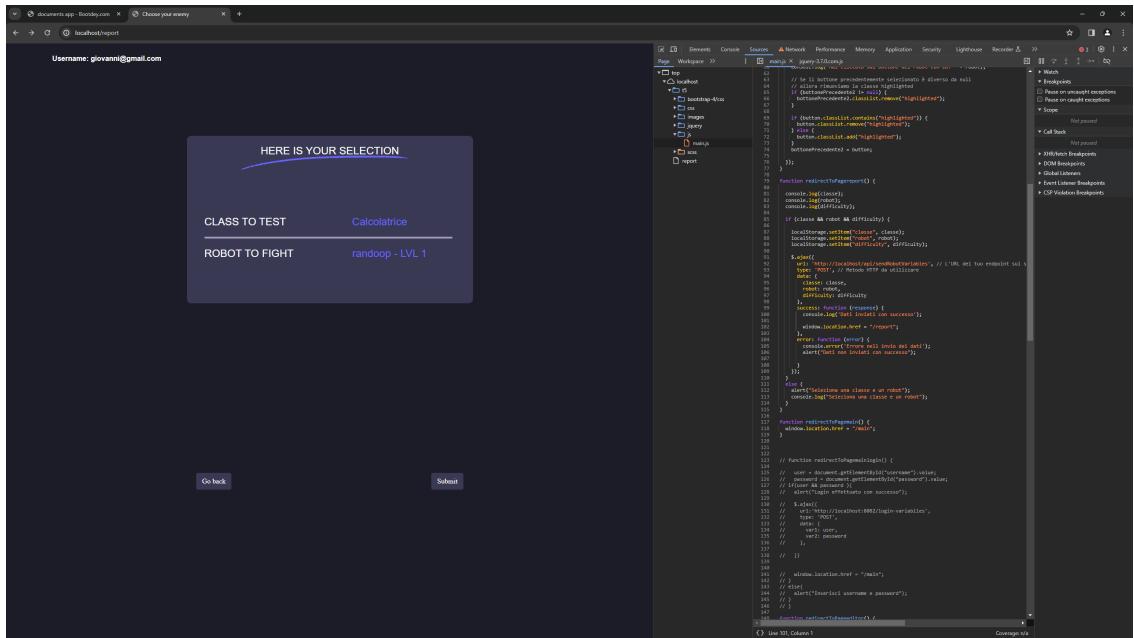


Figure 8.4:

8.5 Il Ruolo Essenziale di Node.js

8.5.1 Introduzione a Node.js

Node.js è un ambiente di esecuzione JavaScript lato server che consente di eseguire codice JavaScript fuori dal browser. La sua natura asincrona e basata su eventi lo rende ideale per sviluppare applicazioni web veloci e scalabili.

8.5.2 Node.js e VSCode

In combinazione con Visual Studio Code (VSCode), Node.js migliora significativamente lo sviluppo. Grazie all'integrazione con Node.js, VSCode offre funzionalità di autocompletamento per i file target, facili-

tando lo sviluppo e riducendo il margine di errore. Questo ambiente integrato aiuta a mantenere un flusso di lavoro agile e reattivo, accelerando lo sviluppo di funzionalità e la risoluzione di bug.

8.6 Conversione degli End of File (EOF)

8.6.1 Problemi con CRLF e LF

I sistemi operativi differiscono nel modo in cui gestiscono la fine delle linee nei file di testo. Windows utilizza CRLF, mentre Unix/Linux e macOS utilizzano LF. Questa discrepanza può causare problemi, soprattutto quando si eseguono script in ambienti Docker basati su Linux.

8.7 Soluzione

Prima di procedere con l'installazione di Docker, è vitale convertire gli EOF dei file di installazione in LF. Questo può essere fatto manualmente con editor che supportano la conversione di EOF, come Visual Studio Code, o attraverso strumenti da riga di comando disponibili su sistemi Unix/Linux.

I file che potrebbero richiedere queste modifiche nel progetto sono:

- T1-G11\applicazione\manvsclass

CHAPTER 8. CONSIGLI E RISOLUZIONE DEI PROBLEMI INSTALLAZIONE

- T8-G21\Progetto_SAD_GRUPPO21_TASK8\Progetto_def\opt_livelli\Prototipo2.0

```
$ installazione.sh
C: > Users > giova > Desktop > progetto round > A5-2024 > T1-G11 > applicazione > manvsclass > $ installazione.sh
1 #installiamo i programmi necessari
2 apt update
3 apt-get -y install software-properties-common
4 apt-add-repository universe
5 apt-get update
6 apt-get -y install maven openjdk-8-jdk git unzip nodejs
7
8 sleep 3
9
10 echo "le versioni di java e javac sono le seguenti, è necessaria la versione 1.8: "
11
12 update-alternatives --set java /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
13 update-alternatives --set javac /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/javac
14
15 java -version
16 javac -version
17
18 sleep 2
19
20
21 wget https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-1.0.6.jar
22
23 wget https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-standalone-runtime-1.0.6.jar
24
25 java -jar evosuite-1.0.6.jar
26
27 sleep 2
28
```

Ln 32, Col 1 Spaces: 4 CRLF Script

Figure 8.5: CRLF

```
$ installazione.sh
C: > Users > giova > Desktop > progett LF CRLF
1 #installiamo i programmi necessari
2 apt update
3 apt-get -y install software-properties-common
4 apt-add-repository universe
5 apt-get update
6 apt-get -y install maven openjdk-8-jdk git unzip nodejs
7
8 sleep 3
9
10 echo "le versioni di java e javac sono le seguenti, è necessaria la versione 1.8: "
11
12 update-alternatives --set java /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
13 update-alternatives --set javac /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/javac
14
15 java -version
16 javac -version
17
18 sleep 2
19
20
21 wget https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-1.0.6.jar
22
23 wget https://github.com/EvoSuite/evosuite/releases/download/v1.0.6/evosuite-standalone-runtime-1.0.6.jar
24
25 java -jar evosuite-1.0.6.jar
26
27 sleep 2
28
```

Ln 32, Col 1 Spaces: 4 UTF-8 CRLF Shell Script

Figure 8.6: LF

Chapter 9

Installazione

Introduciamo alcune conoscenze preliminari per poi guidare i lettori nell'installazione vera e propria dell'applicazione.

9.1 Docker e WSL

Docker è una piattaforma che consente la creazione, la distribuzione e la gestione efficiente e isolata di applicazioni container. Tutto ciò assicura la massima portabilità dell'applicazione software, indipendentemente dall'ambiente di esecuzione sottostante. Per effettuare l'esecuzione di Docker in ambiente Windows è necessario l'impiego della WSL (Windows Subsystem for Linux), che permette di eseguire un sistema Linux all'interno di Windows, consentendo agli utenti di utilizzare strumenti e applicazioni Linux sulla propria macchina Windows.

L'utilizzo di Docker permette di creare container, volumi e immagini che operano in ambienti isolati, consentendo l'esecuzione di servizi separati e la comunicazione tra di essi attraverso il mapping dei porti specifici. L'approccio adottato da Docker, quindi, favorisce la modularità e la scalabilità delle applicazioni e semplifica la loro distribuzione, ottimizzando l'efficienza e la gestione delle risorse. È stato possibile includere un file system locale all'interno del container, utilizzando percorsi assoluti per accedervi in combinazione con l'artefatto prodotto. Da questa configurazione è stata creata un'istanza di macchina virtuale, favorendo l'utilizzo delle risorse del file system senza dovervi rinunciare.

9.2 Deployment dell'applicazione

Di seguito sono riportati i vari passi per installare l'applicazione sulla propria macchina.

9.2.1 Download applicazioni necessarie

Scaricare e installare Docker Desktop. Se si sta operando in ambiente Windows, installare preliminarmente WSL; per farlo basta aprire una nuova finestra del terminale e lanciare il comando "`wsl –install`".

9.2.2 Installazione

Si deve avviare lo script "installer.bat" se si sta usando Windows oppure "installermac.sh" nel caso si utilizzi macOS o Linux; su MacOS eseguire preliminarmente da terminale, nella cartella dove è presente il file "installermac.sh", il comando "chmod +x installermac.sh" per renderlo eseguibile, e poi "./installermac.sh" per eseguirlo. Tale installazione porterà alla:

- Creazione della rete "global-network" comune a tutti i container;
- Creazione del volume VolumeT9 comune ai Task T1 e T9 e del VolumeT8 comune ai Task T1 e T8;
- Creazione dei singoli container in Docker Desktop. Il container relativo al Task T9 (Progetto-SAD-G19-master) si sosponderà autonomamente dopo l'avvio, dato che viene utilizzato solo per popolare il volume VolumeT9 condiviso con il Task T1.

Dopo l'installazione, controllare dunque che tutti gli altri container eccetto quello del T9 siano funzionanti, in caso contrario riavviarli manualmente lasciando per ultimo "ui gateway", il quale dovrà essere avviato solo dopo l'esecuzione di tutti gli altri.

9.2.3 Configurazione MongoDB

Si deve configurare il container "manvsclass-mongo db-1", per fare ciò bisogna eseguire le seguenti operazioni:

- da Docker Desktop, posizionarsi all'interno del terminale del container "manvsclass-mongo db-1";
- eseguire il comando "mongosh";
- eseguire in maniera sequenziale i seguenti comandi:
 1. use manvsclass
 2. db.createCollection(ClassUT);
 3. db.createCollection(interaction);
 4. db.createCollection(Admin);
 5. db.createCollection(Operation);
 6. db.ClassUT.createIndex(difficulty: 1)
 7. db.Interaction.createIndex(name: text, type: 1)
 8. db.interaction.createIndex(name: text)
 9. db.Admin.createIndex(username: 1)

Una volta effettuati tutti i passaggi l'applicazione è completamente configurata e raggiungibile sul port 80.

9.3 Utilizzo dell'applicazione

Prima di iniziare a giocare, è necessario caricare una classe di test da sottoporre a verifica. Questa operazione può essere eseguita esclusivamente dall'utente amministratore. Pertanto, è essenziale che

l'amministratore si registri al sistema in modo da poter successivamente caricare la classe di test. Di seguito sono fornite le istruzioni su come caricare la classe di test ed effettuare la registrazione dell'utente player attraverso una procedura guidata.

9.3.1 Registrazione admin

L'amministratore, in caso di primo utilizzo all'applicazione, si deve registrare nella schermata di registrazione alla quale si può accedere al seguente URL:

`http://localhost/registraAdmin.`

Apparirà una pagina web in cui è possibile inserire nome, cognome, username e password.

9.3.2 Caricamento classi

Una volta che l'admin si è registrato e autenticato, si troverà sulla pagina in Figura 9.1, qui potrà visualizzare tutte le classi caricate.

CHAPTER 9. INSTALLAZIONE

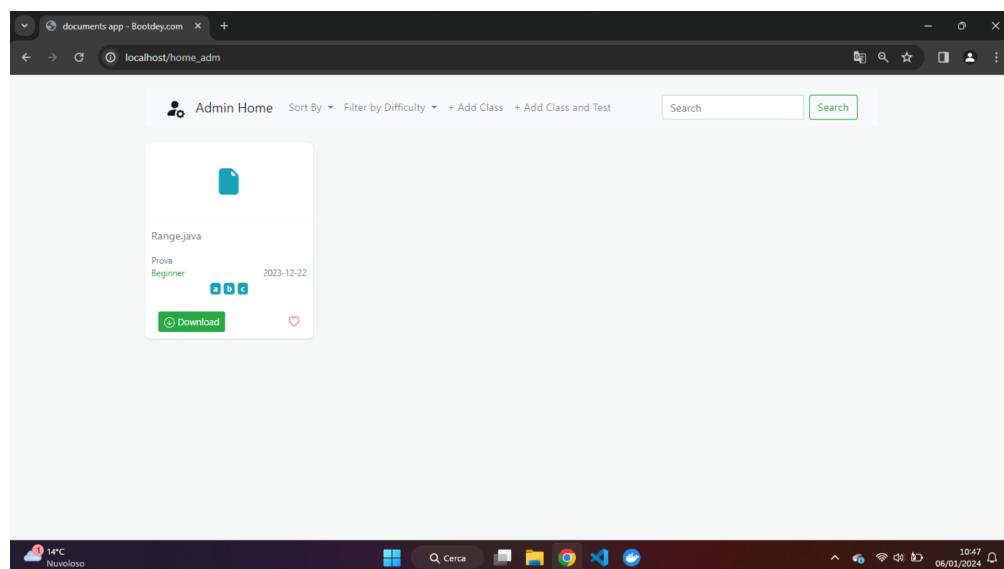


Figure 9.1: Admin home

In seguito visualizzeremo la pagina in Figura 9.2, dove possiamo compilare i diversi campi relativi a nome, difficoltà, categorie... della classe.

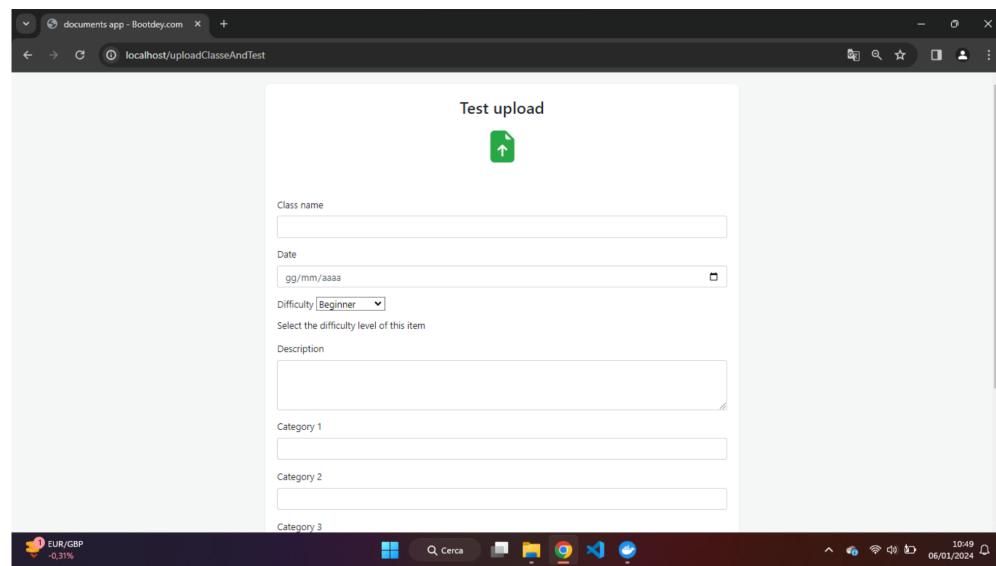


Figure 9.2: Upload classe

9.3.3 Registrazione utente

Se l'utente è al primo utilizzo dell' applicazione, deve recarsi all'URL:

`http://localhost/register`

per accedere alla schermata di registrazione; a questo punto è possibile effettuare la registrazione inserendo il proprio nome, cognome, email, e password. La password deve contenere almeno un carattere speciale, una lettera maiuscola, una minuscola, e deve contenere un minimo di 8 caratteri. Una volta registrato, l'utente può effettuare il login e scegliere una classe di test tra quelle caricate dall'amministratore, insieme al livello di difficoltà del robot da testare, come vediamo in Figura 9.3, per poi confermare la scelta e iniziare a giocare. L'obiettivo è provare a battere il robot e raggiungere un livello di copertura dei test sempre maggiore. Possiamo visualizzare la schermata di gioco in Figura 9.4.

CHAPTER 9. INSTALLAZIONE

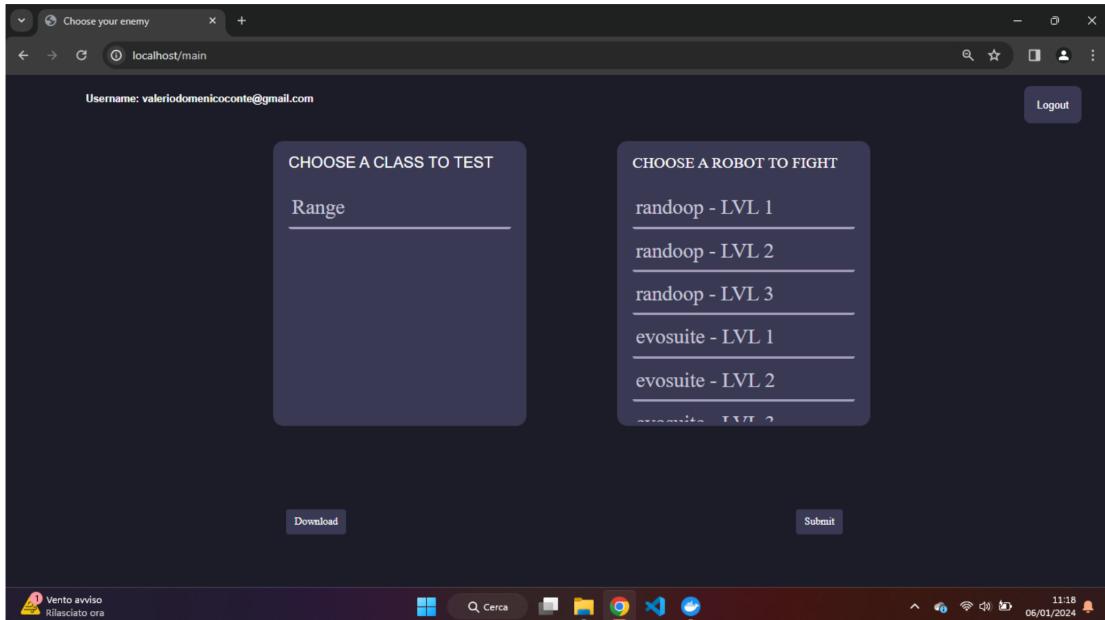


Figure 9.3: Schermata di report

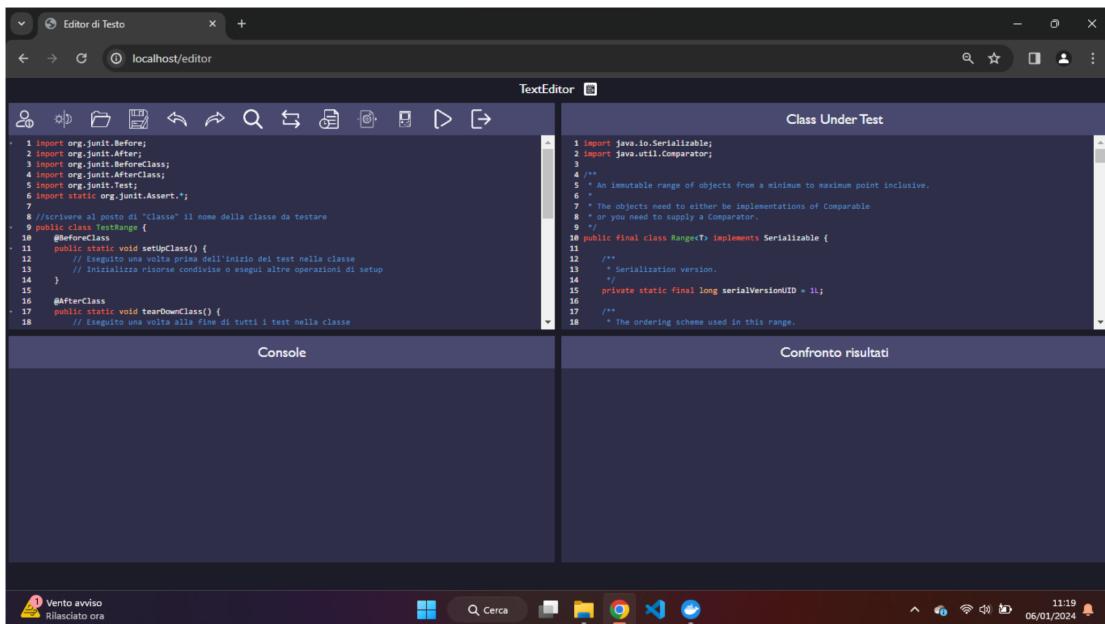


Figure 9.4: Schermata di editor

Chapter 10

Glossario

10.1 Game

Il game (oppure partita, gioco) e` l'entita` principale del sistema; un game e` composto da un round, che a sua volta si compone di molteplici turni.

10.2 Player

Il player (oppure giocatore) e` l'entita` che puo` giocare un partita, sfidando uno dei due possibili robot in ogni turno del round.

10.3 Robot

Il robot e` l'entita` affrontata dal giocatore durante un game; ci sono due possibili robot: Randoop ed Evosuite.

10.4 Round

Il round e` l'entita` che contiene le informazioni di gioco principali. Ogni game e` composto da un numero finito di round, che noi abbiamo impostato pari a uno, durante il quale ciascun giocatore effettua le proprie giocate all'interno dei rispettivi turni.

10.5 Turn

Il turn (oppure turno) e` l'unita` elementare di un round e rappresenta le azioni di un giocatore, in particolare la classe da lui scritta per sfidare il robot; tale entita` contiene dunque informazioni sull'esito dell'esecuzione dei test, i risultati della sfida contro il robot e le classi generate per risolvere il problema.