



UNIVERSITÀ^{DEGLI} STUDI DI
NAPOLI FEDERICO II

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

Software Architecture Design

Documentazione Team A3 Task T4

Anno Accademico 2023/2024

Prof.ssa: **Anna Rita Fasolino**

Membri del team:

Valerio Domenico Conte M63001606

Nike Di Giacomo M63001641

Alessandro Falino M63001658

Abstract

Questo elaborato descrive le metodologie, le fasi di sviluppo e gli artefatti prodotti durante il corso di Software Architecture Design, anno accademico 2023/24, per la realizzazione di uno specifico task all'interno del progetto **ENACTEST**. In particolare, il progetto ha come obiettivo la creazione e lo sviluppo di un gioco con lo scopo di promuovere l'attività di testing e permettere ai partecipanti di cimentarsi in sfide contro sistemi automatizzati ed eventualmente altri giocatori.

L’elaborato è stato suddiviso nei seguenti capitoli:

- **Capitolo 1:** in questa introduzione vengono presentati il progetto, il requisito da soddisfare, la suddivisione del lavoro e lo stato del progetto che ha rappresentato il nostro punto di partenza;
- **Capitolo 2:** qui vengono illustrate le metodologie, gli strumenti e le tecnologie che ci hanno accompagnato nel nostro lavoro di sviluppo software;
- **Capitolo 3:** in questo capitolo viene approfondito il requisito funzionale che abbiamo dovuto sviluppare nel corso del nostro lavoro, illustrando le modifiche che abbiamo apportato alle componenti di interesse del progetto;
- **Capitolo 4:** questo capitolo approfondisce l’organizzazione del lavoro all’interno del team, in particolare spiega come è stato applicato Scrum e quali sono stati i risultati di ogni iterazione;
- **Capitolo 5:** in tale capitolo viene fornita una guida dettagliata per effettuare l’installazione dell’applicazione e per configurala correttamente affinché possa essere utilizzata;
- **Capitolo 6:** qui troviamo un glossario dei termini specifici di maggiore rilievo all’interno della documentazione.

Indice

1	Introduzione	1
1.1	Descrizione del progetto	2
1.2	Descrizione del requisito assegnato	3
1.2.1	Requisito R2	4
1.3	Suddivisione del lavoro	5
1.4	Stato iniziale del progetto	7
1.4.1	Stato del task T4	10
1.4.2	Stato del task T5	19
2	Metodologie di sviluppo	24
2.1	Scrum	25
2.1.1	Daily Scrum e Sprint Backlog	26
2.2	Strumenti	27
2.2.1	Microsoft Teams	27
2.2.2	Github	28
2.2.3	Visual Paradigm	29
2.2.4	MiroBoard	30
2.3	Tecnologie	31
2.3.1	Visual Studio Code	31

2.3.2	Java Spring e Spring Boot	32
2.3.3	Maven	33
2.3.4	Go	34
2.3.5	Postman	35
3	Requisito R2	37
3.1	Analisi preliminare e problematiche	37
3.2	Refactoring e nuove implementazioni in T4	39
3.2.1	model	45
3.2.2	api	45
3.2.3	Aggiornamento dell'API Diagram	47
3.3	Refactoring e nuove implementazioni in T5	52
3.3.1	Game.java	54
3.3.2	GameDataWriter.java	55
3.3.3	saveGameCSV	56
3.3.4	updateGameCSV	59
3.3.5	createNewTurnCSV	59
3.3.6	GUIController.java	59
3.3.7	receiveGameVariables	60
3.3.8	saveGame	61
3.3.9	updateGame	63
3.3.10	Aggiornamento della documentazione	65
3.4	Ricompilazione del progetto	69
3.5	Testing	70
4	Organizzazione del lavoro	77
4.1	Introduzione a Scrum	77

4.1.1	Adozione di Scrum per il nostro progetto	77
4.2	Prima iterazione	78
4.3	Seconda iterazione	78
4.4	Terza iterazione	80
4.5	Quarta iterazione	81
5	Installazione	83
5.1	Docker e WSL	83
5.2	Deployment dell'applicazione	84
5.2.1	Download applicazioni necessarie	84
5.2.2	Installazione	85
5.2.3	Configurazione MongoDB	85
5.2.4	Cosa fare se MongoDB non funziona	86
5.3	Utilizzo dell'applicazione	88
5.3.1	Registrazione admin	88
5.3.2	Caricamento classi	88
5.3.3	Registrazione utente	90
6	Glossario	92
6.1	Game	92
6.2	Player	92
6.3	Robot	92
6.4	Round	93
6.5	Turn	93

Capitolo 1

Introduzione

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

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

L'approccio innovativo di ENACTEST si concretizza attraverso la

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

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

1.1 Descrizione del progetto

Man vs Automated Testing Tools Challenges è un gioco educativo nel quale gli studenti si confrontano con strumenti di testing automatizzati, Randoop ed Evosuite, capaci di generare casi di test JUnit. I task consistono nello sviluppo di casi di test di unità per classi Java che devono raggiungere uno specifico obiettivo. I possibili obiettivi sono la copertura delle LOC, dei Branch, delle Decisioni, delle Eccezioni, di Weak Mutation. I casi di test sono da implementare in JUnit 4. Per l'implementazione è stato scelto un approccio basato dunque sulla Gamification, che consenta dunque la competizione con strumenti di generazione automatica, come specificato precedentemente. Sulla piatta-

taforma Github è disponibile il progetto e la documentazione fornita da ciascun gruppo partecipante, incaricato di implementare requisiti funzionali accorpati in task. Per un’analisi più approfondita e per una piena comprensione si consiglia dunque di consultare la documentazione messa a disposizione da ciascun gruppo che ha contribuito alla realizzazione della prima edizione del gioco al seguente link: Testing Game.

1.2 Descrizione del requisito assegnato

Prima di addentrarci nella descrizione specifica del requisito assegnato, è bene evidenziare dunque l’obiettivo generale: realizzare un’applicazione per consentire agli studenti di fare addestramento su Task di Unit Testing. Analizziamo sinteticamente i requisiti generali sui Task di Unit Testing:

- L’applicazione deve permettere allo studente di ottenere una classe da poter testare, scrivere casi di test in J-Unit, compilari e quantificare gli obiettivi raggiunti.
- L’applicazione deve consentire allo studente la generazione automatica dei test per una classe Java sottoposta. Tali casi di test saranno compilati ed eseguiti e il risultato sarà la copertura degli obiettivi raggiunta. Sarà possibile operare a diversi livelli di difficoltà(Easy, Medium, Hard).

- L'applicazione deve inoltre consentire il confronto tra i risultati dei test generati dallo studente e quelli generati dallo strumento automatico.
- L'applicazione deve permettere la registrazione al fine di giocare in autonomia o allo scopo di consentire la sfida.

1.2.1 Requisito R2

Per consentire l'implementazione del requisito assegnatoci, è stato necessario lo studio e la comprensione del **task T4, "Requisiti sul mantenimento delle Partite giocate"**. Nello specifico:

Per ogni partita giocata dal giocatore il sistema deve mantenere lo storico di tale partita, memorizzando l'Id del giocatore, il tipo di partita (primo scenario, secondo scenario...) la data e l'ora di inizio e di termine della partita, la classe testata, l'insieme dei casi di test creati in ogni turno e i relativi risultati, nonché il Robot con cui si è giocato e i casi di test creati dal Robot con i relativi risultati. Ogni partita dovrà essere fatta di più turni.

La documentazione messa a disposizione per la fase preliminare di studio è consultabile al seguente link per un'analisi approfondita del punto di partenza: Documentazione T4. L'obiettivo principale del nostro gruppo consiste nell'implementare il requisito R2, riportato di seguito:

Verificare e gestire la coerenza dei dati della partita del giocatore attualmente salvati nel File System di T8 con quelli mantenuti nel database del gioco in T4. Eventualmente rivedere le API offerte da T4 per consentire di salvare anche i dati relativi ai Turni di gioco in T4.

1.3 Suddivisione del lavoro

Questo lavoro è stato preceduto, come già evidenziato in precedenza, da una revisione del lavoro dei nostri colleghi che hanno integrato i vari componenti dell’architettura globale e documentato tutto come riportato al seguente link: Documentazione Progetto. Il lavoro è stato inizialmente suddiviso nei seguenti task con corrispondenti stime di persone necessarie per portare ciascuno a completamento, nelle due settimane dateci per portare a termine il lavoro:

- **Task per la prima iterazione:**

- Analisi dell’architettura di partenza, partendo dalla documentazione generale del progetto.
- Analisi del Task specifico (T4) mediante studio della relativa documentazione. [3 persone]

- **Task per la seconda iterazione:**

- Modifiche al diagramma ER per far fronte alle nostre esigenze.

- Studio del Task T5, al fine di capire come implementare il confronto (per valutare la coerenza) tra i dati presenti nel Game Repository, relativo al task T4 e i dati salvati nel FileSystem.
- Realizzazione di diagrammi UML al fine di documentare le scelte che si stessero intraprendendo. [2 persone]

- **Task per la terza iterazione:**

- Refactoring del codice del componente T4, al fine di adattare le API messe a disposizione per consentire il salvataggio dei dati relativi ai turni di gioco.
- Implementazione di un metodo ad hoc che consentisse il salvataggio delle informazioni relative a un Game nell'apposito file CSV, all'interno del task T5. [2 persone]

- **Task per la quarta iterazione:**

- Stesura di una nuova documentazione, più dettagliata e organizzata. [2 persone]
- Ulteriori modifiche del codice di T4 al fine di adattare le API messe a disposizione per consentire il salvataggio dei dati relativi a games, rounds e turn, in particolare creazione e aggiornamento di questi. [1 persona]

- Miglioramento del metodo che consente il salvataggio delle informazioni di un Game in file system. [1 persona]
- Implementazione di un metodo per gestire il passaggio tra diversi turni, assicurando ancora una volta coerenza tra dati salvati in Database e in File System. [1 persona]

1.4 Stato iniziale del progetto

Per comprendere il lavoro da noi svolto è opportuno prestare attenzione a ciò che abbiamo avuto "in eredità" dai colleghi che hanno lavorato per primi a questo progetto. Prima di fare qualsiasi considerazione sul progetto in sé è necessario soffermarsi su un aspetto fondamentale: l'installazione del programma. È stato abbastanza problematico riuscire a installare tale programma: fin da subito abbiamo notato che le guide messe a disposizione non fossero esaustive, motivo per il quale abbiamo riscontrato così tanta difficoltà in questa fase preliminare che fortunatamente siamo riusciti poi a superare. Forniamo ora una vista generale sul sistema nel complesso attraverso il diagramma Componenti e Connettori in Figura 1.1. Tenendo presente tale diagramma, il nostro lavoro si concentrerà al livello dei componenti **Game Repository** e **Student Repository** e delle interfacce da essi esposte.

Di seguito analizziamo più nel dettaglio i componenti di tale diagramma:

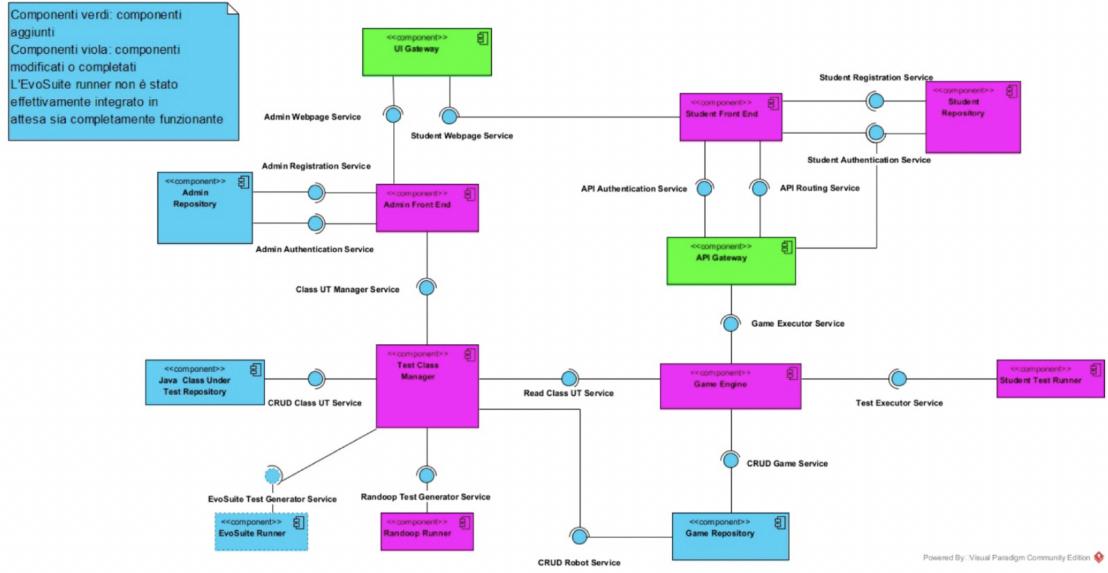


Figura 1.1: Diagramma Componenti e Connettori

- **Admin Front End:** è un componente che utilizza l’interfaccia Admin Features, esposta dalla API Gateway. Consente di fornire una schermata personalizzata per l’admin che avrà a disposizione diverse funzionalità proprie della sola figura dell’amministratore.
- **Student Front End:** è un componente che utilizza l’interfaccia Player Features, esposta dalla API Gateway. Ha lo scopo di consentire ad un giocatore di intraprendere una partita, mediante l’utilizzo di una pagina web.
- **API Gateway:** è un componente che espone diverse interfacce (Admin Features, Player Features) e ne utilizza di altre messe a disposizione dagli altri componenti (Registration Service, Authentication Service, CRUD Class UT, Class UT Manager,

Round Executor). Esso si occupa di indirizzare le richieste ai servizi giusti e di effettuare le operazioni di autenticazione e autorizzazione. Ed è l'unico punto di contatto tra back-end e front-end.

- **Student Repository:** Questo componente è responsabile di salvare i dati relativi alle partite giocate da uno studente, ed espone inoltre due interfacce: Registration Service e Authentication Service.
- **Java Class Under Test Repository:** Questo componente salva le classi testate, ed espone un'interfaccia, che identifica le funzionalità CRUD esposte dalla UT.
- **Test Class Manager:** Questo componente è responsabile dell'esecuzione del testing per ogni livello di difficoltà del gioco interagendo con il Test Run Environment.
- **Game Engine:** Questo componente che rappresenta l'utente utilizzatore del servizio che si occupa di implementare le logiche di gioco utilizza l'interfaccia Test Executor e l'interfaccia CRUD Game.
- **Game Repository:** il componente ha la responsabilità di salvare i risultati dei test.

- **Evosuite Runner:** è un componente che consente la generazione dei livelli del robot Evosuite.
- **Randoop Runner:** è un componente che consente la generazione dei livelli del robot Randoop.
- **Student Test Runner:** è un componente che identifica il motore di analisi delle analisi scritte dal giocatore e che genera i risultati del test in termini di metriche di coverage.

1.4.1 Stato del task T4

Il requisito assegnatoci richiede di intervenire sul task T4, pertanto di seguito presenteremo la sua struttura per una maggiore comprensione.

Il componente T4 fornisce una API REST per gestire le informazioni che riguardano i giocatori, i robot e le partite. In particolare, T4 è composto da:

- **Rest Server:** componente principale dell'applicazione, implementa l'API suddetta ed è stato realizzato in linguaggio **Go**;
- **Postgres:** database relazionale necessario al funzionamento dell'applicazione;
- **Prometheus:** permette l'estrazione delle metriche dal Rest Server con un sistema di monitoraggio periodico;

- **Grafana:** elabora le metriche estratte da Prometheus per la creazione di dashboard grafiche che consentono la visualizzazione delle performance dell'applicazione.

Innanzitutto, visualizziamo il diagramma delle classi relativo al task in esame presente in Figura 1.2. Come si può notare sono presenti diversi packages che fanno riferimento alle entità di interesse, cioè Game, Robot, Round e Turn, e un package Model, con il quale ciascun package interagisce tramite delle relazioni d'uso dato che si occupa della logica di business e data access. Ciascuna entità (in verde) implementa i pattern DTO (Data Transfer Object) e Dependency Injection e sono tra loro indipendenti, mentre lo schema della base dati, rappresentante gli attributi delle entità e le relazioni tra di esse, è situato nel package Model (in rosa).

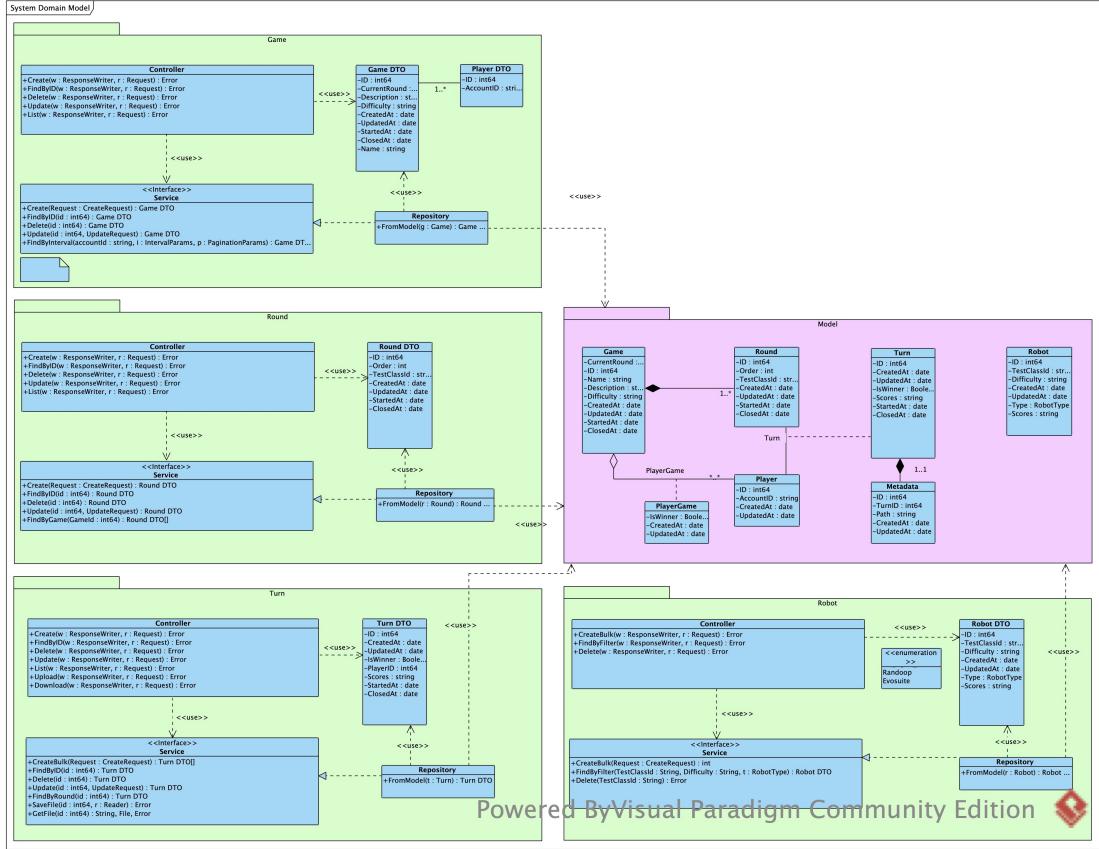


Figura 1.2: Diagramma delle Classi del Game Repository

Analizzando la directory del progetto, abbiamo individuato tre cartelle principali, strutturate secondo quanto detto:

- **model:** in cui ci sono le strutture che servono per interagire con il database;
- **api:** in cui per ogni entità sono implementati gli endpoint HTTP con le relative operazioni sul database;
- **postman:** in cui si trovano la specifica OpenAPI e la collection Postman.

Riportiamo ora una vista di alto livello sull’architettura del task T4 attraverso il diagramma Componenti e Connettori in Figura 1.3. Individuiamo quattro parti dell’applicazione indipendenti tra loro, cioè le entità di interesse, rappresentate dalle fasce orizzontali rosa; ogni entità ha a sua disposizione un proprio Controller che interagisce con un Service, sfruttandone l’interfaccia esposta. A loro volta, i vari componenti Service interagiscono con un componente ORM (Object-Relational Mapping) che implementa la logica di accesso dati verso il database relazionale Postgres. Altri due componenti sono Prometheus e Grafana: il primo monitora alcune metriche prestazionali dell’applicazione, il secondo consente la visualizzazione di tali metriche. Notiamo che le implementazioni delle classi Service, astratte mediante le relative interfacce, e l’indipendenza tra le diverse entità rappresentano scelte atte a garantire un basso accoppiamento e una migliore testabilità dell’applicazione, in quanto si possono così testare indipendentemente i componenti Controller e Service.

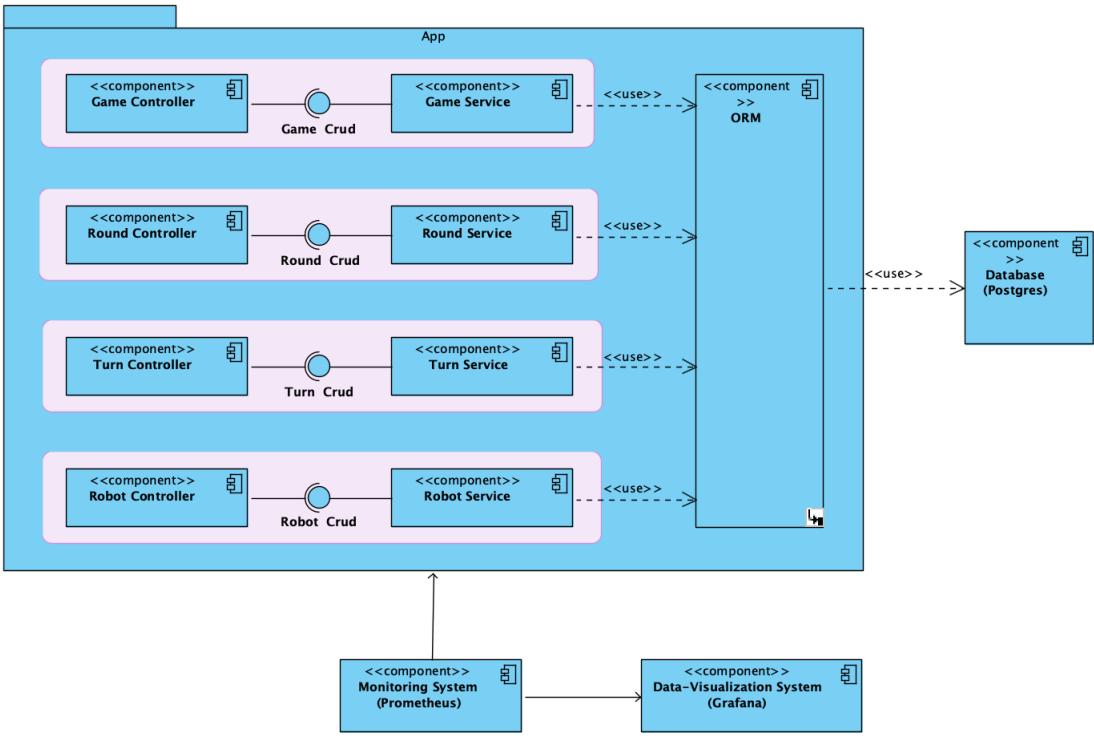


Figura 1.3: Diagramma C&C del Game Repository

In Figura 1.4 possiamo osservare lo stato del diagramma Entity-Relationship antecedente al nostro intervento; chiaramente ritroviamo le entità Player, Game, Round e Turn, inoltre c'è una classe associativa PlayerGame per realizzare una relazione molti a molti tra Player e Game, un'altra entità per rappresentare i Metadata e, infine, un'entità Robot che qui non ha alcun legame col resto. Le modifiche che verranno apportate partiranno proprio dalla correzione del diagramma ER visto, procedendo poi con la modifica della base dati, del Model e, infine, delle API realizzate per i vari componenti.

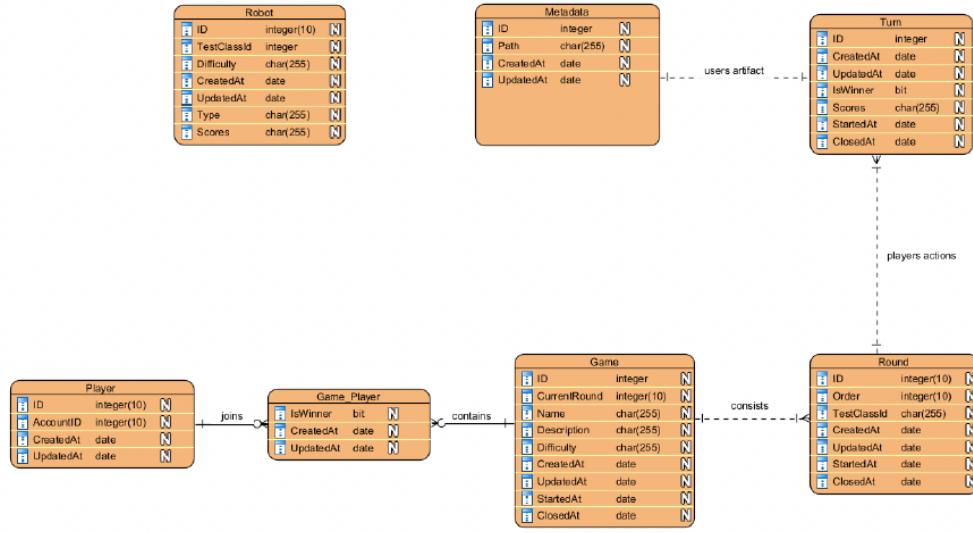


Figura 1.4: Diagramma ER del Game Repository

Ciascuna entità, come si può notare, possiede i campi **CreatedAt** e **UpdatedAt**, utilizzati per tracciare la data di creazione e modifica. Inoltre, le entità in cui si tiene tracciare lo stato di apertura e chiusura posseggono inoltre gli attributi **StartedAt** e **ClosedAt**. È stata effettuata l'analisi dettagliata degli attributi per ciascuna entità:

- **Game**

- ID: identificativo univoco
- Current Round: indica il numero del round corrente
- Name: nome della partita
- Description: descrizione della partita
- Difficulty: grado di difficoltà della partita

- **Player**

- ID: identificativo univoco
- AccountID: identificativo fornito dal componente Student Repository, che contiene i dati relativi ai giocatori

- **Game Player**

- IsWinner: valore booleano che indica il vincitore finale della partita

- **Round**

- ID: identificativo univoco
- Order: indica l'ordine dei round all'interno della partita (primo, secondo etc...)
- TestClassId: Identificativo della classe da testare presente nel Class UT Repository

- **Turn**

- ID: identificativo univoco
- IsWinner: valore booleano che indica il vincitore del singolo round
- Scores: i punteggi relativi al giocatore che ha completato un round, calcolati dal componente di testing e compilazione.

- **Metadata**

- ID: identificativo univoco

- Path: il percorso relativo nel file system in cui sono memorizzati i file delle classi testate dall’utente.

Di rilevanza per il nostro lavoro è l'**API REST** offerta da T4, fondamentale per l’elaborazione e il mantenimento dei dati relativi a giocatori, robot e partite. In particolare, sono state realizzate in Go quattro API che attraverso un Service si occupano di gestire le transazioni e le query verso il database Postgres; tali operazioni vengono richieste tramite i metodi CRUD HTTP dagli altri componenti del progetto, sono poi gestite dal Controller delle API e indirizzate verso il Service, che poi restituisce i propri output al Controller. Queste API riguardano le entità Game, Round, Turn e Robot, sono state testate tramite Postman (nel progetto è presente l’apposita Collection) ed è possibile consultare la loro documentazione al seguente link: <https://pkg.go.dev/github.com/alarmfox/game-repository@v0.0.4-rc/api>. Possiamo servirci del diagramma delle API in Figura 1.5 per approfondire il funzionamento di tutti i metodi esposti dalle interfacce, visualizzando per esempio le richieste HTTP che li attivano, le risposte e gli errori che possono eventualmente generare.

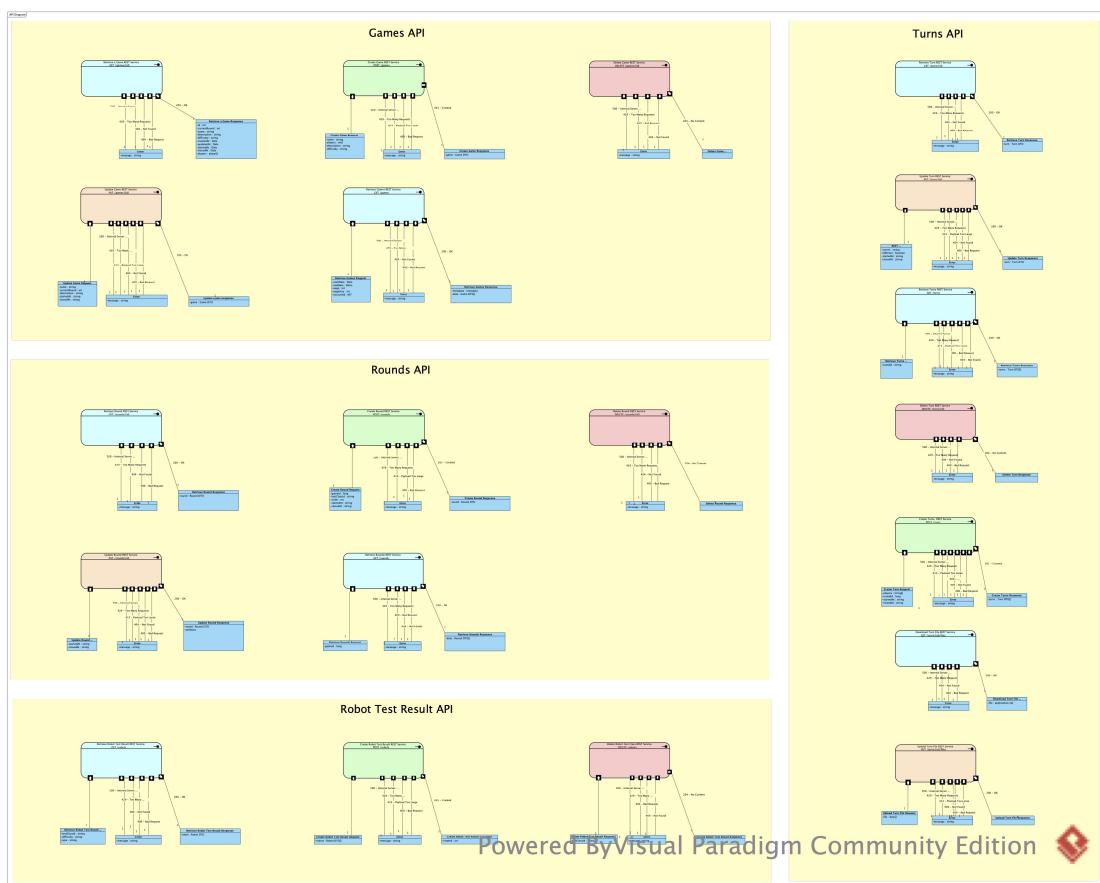


Figura 1.5: Diagramma delle API del Game Repository

1.4.2 Stato del task T5

Il nostro lavoro si è concentrato anche sul task T5, dove abbiamo trovato le classi che si occupano del salvataggio dei dati relativi alle partite; qui siamo dovuti intervenire per garantire la coerenza tra i dati salvati all'interno del **Game Repository** (database di T4) e quelli salvati nello **Student Repository** (file system di T5). Preliminariamente abbiamo analizzato la documentazione del T5 per comprenderne innanzitutto la struttura. Il task T5 fornisce l'API per la gestione dei dati inerenti alla creazione e l'avvio di una partita, in particolare realizza il front-end dell'applicazione che permette all'utente di autenticarsi e di iniziare una partita e del back-end necessario per effettuare gli opportuni salvataggi in database e su file system. Si compone di una serie di classi Java che realizzano la logica di business e di controllo, e di una web-app realizzata in HTML, CSS, Javascript usando il framework Bootstrap che fornisce invece la logica di presentazione. L'architettura del T5 ricalca dunque il pattern **MVC**, troviamo infatti in Figura 1.6 il Class Diagram dello Student Repository, dove è possibile distinguere i packages Model, View e Controller e inoltre dei packages riferiti alle interfacce.

CAPITOLO 1. INTRODUZIONE

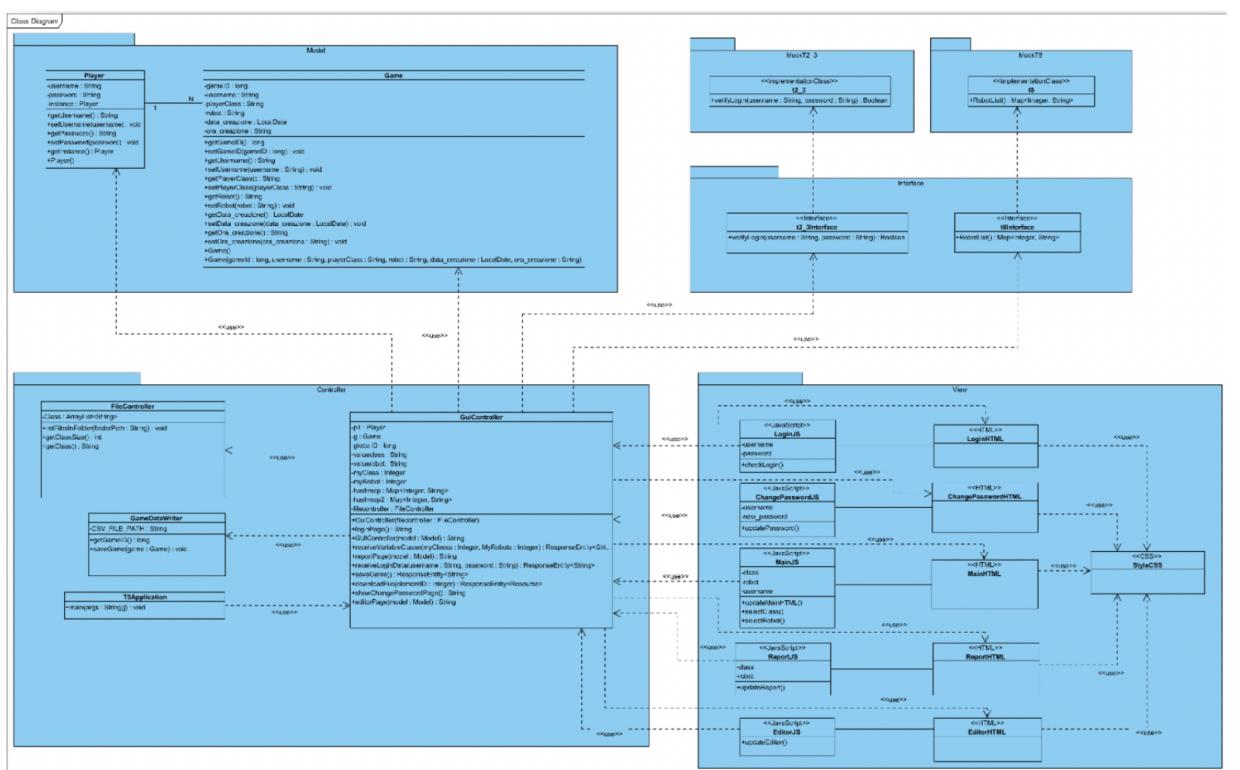


Figura 1.6: Diagramma delle classi dello Student Repository

Vediamo di cosa si occupano di preciso le varie parti:

- **Model:** gestisce la logica di business e definisce le entità di interesse, consentendo dunque l'accesso ai dati necessari e l'aggiornamento delle viste; il Model è stato realizzato in **Java** e le entità da esso definite sono lo studente che effettuerà la partita (*Player*), la classe da testare (*ClassUT*) e la partita stessa (*Game*);
- **View:** gestisce la logica di presentazione e coincide con il front-end della *Web-App* che consente al giocatore di effettuare registrazione e login, scegliere classe e robot da sfidare e di iniziare una nuova partita; è stato realizzato utilizzando il framework **Bootstrap**, che agevola la creazione di siti web moderni e responsivi usando le tecnologie **Javascript**, **HTML** e **CSS**;
- **Controller:** gestisce la logica di controllo, elaborando i dati di input provenienti dall'applicazione e aggiornando il Model quando necessario; è costituito da due importanti classi **Java**: *GUI-Controller*, che grazie al framework **Spring** si occupa di gestire le operazioni CRUD definendo diversi metodi per ogni route e operazione, e *GameDataWriter*, che definisce i metodi per l'interazione con Game Repository e Student Repository.

Possiamo comprendere meglio le operazioni di cui T5 si occupa complessivamente visualizzando il Sequence Diagram in Figura 1.7. Analizzando il codice che realizza queste operazioni, ci siamo resi conto che la logica di salvataggio dei dati della partita su file system in realtà ancora non era stata implementata, per cui una volta compresa la struttura dello Student Repository abbiamo apportato le opportune modifiche al codice. Le modifiche di cui parleremo più nel dettaglio nei prossimi capitoli fanno riferimento alla necessità di effettuare il salvataggio delle informazioni relative alle partite all'interno di un file CSV nello Student Repository, assicurandosi inoltre che tali informazioni siano coerenti con quelle che vengono salvate nel database del Game Repository. Discuteremo successivamente di come tale funzionalità sia stata implementata. Maggiori dettagli sul funzionamento dello Student Repository possono essere trovati nella documentazione realizzata dai nostri precedenti colleghi e consultabile al seguente link: Documentazione T5.

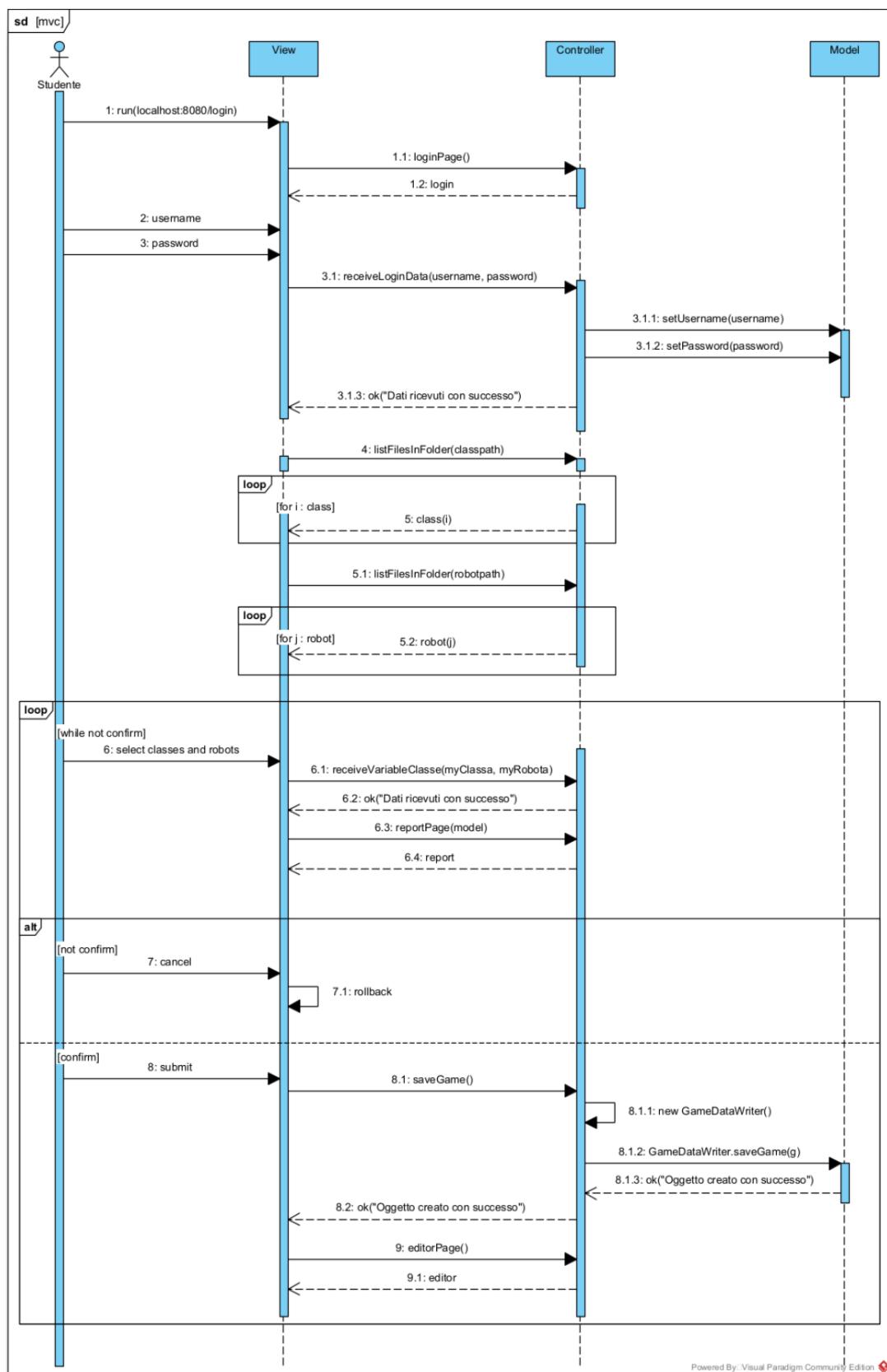


Figura 1.7: Diagramma di sequenza della creazione di un game

Capitolo 2

Metodologie di sviluppo

Il primo passo per la realizzazione del task assegnato è stato l'introduzione e l'adozione di **Scrum**, una metodologia Agile incentrata sulla collaborazione di team per raggiungere obiettivi comuni, largamente usata nell'ambito dello sviluppo software. La riunione iniziale del team ha avuto come obiettivo l'analisi della traccia assegnata e la pianificazione di un backlog per gestire eventuali scadenze e stabilire l'organizzazione interna del team (assegnamento dei ruoli, pianificazione dei compiti...). Introduciamo dunque cos'è Scrum, per poi presentare quali sono state le scelte riguardanti gli strumenti e le tecnologie di sviluppo del software.

2.1 Scrum

Scrum è una metodologia Agile utilizzata per la gestione del ciclo di sviluppo software. Scrum è stato introdotto negli anni '90 ed è diventato uno dei metodi più popolari per la gestione dei progetti agili. Questo framework si basa su alcuni principi fondamentali, tra cui:

- **Ruoli:** Scrum definisce tre ruoli principali, cioè Product Owner, Scrum Master e Team di Sviluppo; ognuno di questi ha responsabilità specifiche all'interno del processo;
- **Eventi:** Scrum prevede una serie di eventi definiti per facilitare l'ispezione e l'adattamento costanti. Gli eventi principali includono la Sprint Planning, la Daily Scrum, la Sprint Review e la Sprint Retrospective;
- **Artefatti:** Scrum utilizza diversi artefatti, come il Product Backlog, lo Sprint Backlog e il Product Increment, per tracciare e gestire il lavoro da svolgere durante il progetto;
- **Time-boxing:** gli eventi in Scrum sono time-boxed, il che significa che hanno una durata fissata. Ad esempio, si indica con *Sprint* un periodo di tempo fisso, di solito compreso tra una settimana e un mese, durante il quale il team lavora per consegnare un incremento del prodotto.

L’obiettivo di Scrum è fornire una struttura flessibile, ma allo stesso tempo disciplinata, per lo sviluppo di prodotti, consentendo una rapida risposta ai cambiamenti e una consegna iterativa di valore. Scrum è ampiamente utilizzato nell’ambito dello sviluppo software, ma può essere adattato e applicato a una varietà di contesti e settori.

2.1.1 Daily Scrum e Sprint Backlog

La principale difficoltà è sorta dall’analisi del dominio del problema da affrontare e, successivamente, abbiamo incontrato difficoltà anche nell’analisi della parte di progetto di nostra competenza; menzioniamo ad esempio l’ambiguità di alcune definizioni (round e turn), nonché la poca chiarezza su determinate meccaniche di gioco; altri problemi riscontrati riguardano l’aspetto software (installazione e utilizzo dell’app in primis). Di conseguenza, è stata necessaria dapprima un’analisi approfondita della struttura generale del software e solo successivamente è stato per noi possibile comprendere a fondo alcune scelte fatte relativamente al task di nostra competenza.

Per portare a termine questo processo di analisi sono state organizzate diverse call, effettuate su Microsoft Teams con cadenza variabile, durante le quali abbiamo avuto modo di stabilire l’organizzazione all’interno del team. Tutto ciò ha permesso a noi membri del team di conoscerci meglio, sotto il punto di vista sia professionale/accademico

sia personale, e di stabilire una pianificazione rigorosa delle attività. Successivamente a questi primi meeting organizzativi, abbiamo spesso lavorato da vicino in Università, soprattutto per la parte implementativa, effettuando per esempio Pair Programming.

Le riunioni hanno permesso inoltre la realizzazione degli Sprint Backlog. Durante tali incontri, si è proceduto infatti all’assegnazione dei task tra i membri del team e all’individuazione di scadenze per il conseguimento degli obiettivi. In particolare, nel periodo compreso tra ottobre e dicembre, le deadline sono state inizialmente fissate ogni due settimane; con l’avvicinarsi della data di presentazione del progetto, gli incontri si sono intensificati al fine di portare a compimento il lavoro e raffinare gli artefatti, raggiungendo una cadenza quotidiana.

2.2 Strumenti

Di seguito gli strumenti che ci hanno consentito di svolgere i nostri compiti e lavorare in gruppo secondo quanto indicato da Scrum.

2.2.1 Microsoft Teams

Microsoft Teams è una piattaforma che consente di gestire classi virtuali e di effettuare chiamate o scambiare messaggi tra utenti; è stata un’applicazione di importanza cruciale per facilitare la comunicazione

del team durante le fasi di sviluppo degli artefatti. Le videochiamate di gruppo hanno infatti consentito discussioni dettagliate ed esaustive, mentre l'utilizzo della chat di gruppo ha semplificato lo scambio di file e informazioni di vario tipo. La piattaforma ha contribuito in definitiva a mantenere costante la collaborazione.

2.2.2 Github

GitHub è una piattaforma di hosting di sviluppo software basata su Git, un sistema di controllo versione distribuito che consente agli sviluppatori di tenere traccia delle modifiche al codice sorgente durante lo sviluppo di software. GitHub aggiunge un livello di servizi di gestione del progetto e collaborazione alla base fornita da Git, rendendo più facile per gli sviluppatori collaborare su progetti, tenere traccia delle modifiche, gestire problemi e contribuire al codice. È stato fondamentale per supportare lo sviluppo parallelo grazie alle sue funzionalità:

- **Repository:** un repository (o repo) è un contenitore in cui vengono archiviati i file del progetto, inclusi il codice sorgente, la documentazione e altri file di supporto. I repository possono essere pubblici o privati;
- **Branch:** i branch sono "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;

- **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 discutere e rivedere le modifiche prima che vengano unite nel ramo principale.: Una pull request è una proposta di modifica al codice sorgente all'interno di un repository. Gli sviluppatori possono aprire una pull request per discutere e rivedere le modifiche prima che vengano unite nel ramo principale;
- **Collaborazione:** GitHub offre strumenti per la collaborazione tra membri del team. Gli sviluppatori possono commentare sul codice, partecipare alle discussioni sulle issue e collaborare efficacemente utilizzando funzionalità come le notifiche e le menzioni.

2.2.3 Visual Paradigm

Di fondamentale importanza per la stesura del nostro progetto è stato **Visual Paradigm**, applicazione che ci ha consentito di visualizzare e comprendere i diagrammi UML realizzati dai nostri colleghi in precedenza; naturalmente questo strumento è stato utilizzato anche per la stesura di nuovi diagrammi e la modifica di quelli precedenti (laddove necessario) che verranno presentati in questo documento.

Visual Paradigm è una suite progettata per supportare il processo di

sviluppo del software attraverso varie fasi, dalla modellazione e progettazione all'implementazione e manutenzione. È ampiamente utilizzato nel campo dell'ingegneria del software e offre una serie di strumenti e funzionalità per aiutare gli sviluppatori a progettare, documentare e gestire progetti software complessi.

- **Modellazione visuale:** fornisce strumenti per la creazione di modelli visivi utilizzando notazioni standard come UML (Unified Modeling Language) per rappresentare i concetti e le relazioni nel sistema software
- **Diagrammi UML:** supporta una vasta gamma di diagrammi UML, come diagrammi dei casi d'uso, diagrammi delle classi, diagrammi delle sequenze, diagrammi delle attività e molti altri fondamentali per descrivere e visualizzare aspetti specifici di un sistema software.

2.2.4 MiroBoard

MiroBoard è uno strumento di lavagna digitale che ha agevolato la collaborazione remota del team, che offre:

- **Piattaforma di collaborazione visiva:** offre un ambiente virtuale per la creazione di lavagne digitali, facilitando attività come brainstorming;

- **Strumenti di collaborazione:** fornisce una varietà di strumenti, inclusi pennarelli virtuali e forme;
- **Funzionalità di presentazione:** permette la creazione di presentazioni dinamiche direttamente sulla lavagna.

2.3 Tecnologie

Il nostro lavoro si è concentrato fondamentalmente sul garantire una corretta interazione tra parti diverse all'interno del sistema e tali parti si sono rivelate alquanto eterogenee tra di loro. Conseguenza di ciò è che abbiamo avuto modo di operare e di apprendere talvolta nuovi strumenti e tecnologie di sviluppo, in particolare framework e linguaggi di programmazione.

2.3.1 Visual Studio Code

Visual Studio Code (o **VS Code**) è un editor di codice sorgente largamente utilizzato in quanto leggero e versatile, nonché un IDE che offre un gran numero di funzionalità. Vediamo infatti quali sono i suoi principali vantaggi:

- **Leggerezza e velocità:** a prescindere dall'hardware, VS Code garantisce un'esperienza fluida ai propri utenti in quanto software leggero e reattivo;

- **Estensibilità:** VS Code supporta praticamente qualsiasi linguaggio e può essere arricchito di funzionalità installando le estensioni presenti nel suo Marketplace, in modo tale da soddisfare le esigenze specifiche di sviluppatori in diversi linguaggi di programmazione e per supportare vari framework e tecnologie;
- **Ambiente di sviluppo integrato:** come anticipato, VS Code non è un semplice editor ma anche un IDE, dotato di strumenti come terminale, debugger e Git; quest'ultimo strumento è di particolare rilevanza in quanto agevola notevolmente il version control.

2.3.2 Java Spring e Spring Boot

Java Spring è un framework di sviluppo di applicazioni Java open source che fornisce un modello completo di programmazione e configurazione per le moderne applicazioni Enterprise, su qualsiasi tipo di piattaforma di distribuzione. Grazie all'infrastruttura offerta da tale framework gli sviluppatori possono concentrarsi sulla logica di business delle applicazioni senza preoccuparsi degli specifici ambienti di distribuzione e delle configurazioni. Tra le sue principali caratteristiche troviamo:

- **Design patterns:** Spring promuove l'uso di pattern di progettazione come MVC (Model-View-Controller) per lo sviluppo di applicazioni web;

- **Integrazione:** Spring offre integrazione con tecnologie come JDBC, JMS... facilitando dunque l’interazione con database, middleware e altri servizi di vario genere;
- **Testabilità:** Spring facilita il testing delle applicazioni grazie alla progettazione orientata all’interfaccia e all’uso di oggetti mock per simulare comportamenti di altri oggetti in maniera controllata.

Spring Boot è un’estensione del framework Java Spring che agevola creazione, configurazione, sviluppo e distribuzione di applicazioni stand-alone; tra le sue features infatti troviamo l’inclusione di server incorporati come Tomcat, la possibilità di scegliere tra diversi templates preconfigurati per iniziare un nuovo progetto di un certo tipo (web, dati...) e auto-configurazione basata sulle dipendenze del progetto, riducendo in tal modo la quantità di configurazione manuale necessaria da parte degli sviluppatori.

2.3.3 Maven

Maven è un potente strumento di project management e di build per progetti Java. Fornisce uno standard per la gestione del ciclo di vita di un progetto software, dalla compilazione all’esecuzione dei test, alla distribuzione e alla gestione delle dipendenze. Vediamo alcune delle caratteristiche principali di Maven:

- **Gestione delle dipendenze:** Maven consente di specificare le dipendenze del progetto in un file di configurazione chiamato "pom.xml" (Project Object Model); Maven stesso si occuperà di scaricarle automaticamente dal Maven Central Repository o da repository personalizzati;
- **Ciclo di vita del progetto:** Maven definisce un ciclo di vita standard del progetto con fasi chiave come "clean", "compile", "test", "package", "install" e "deploy"; ogni fase del ciclo di vita rappresenta un passo specifico nel processo di sviluppo del software;
- **Maven Central Repository:** come anticipato, Maven sfrutta un repository centrale, situato online, che contiene una vasta gamma di librerie e framework Java; Maven scarica automaticamente le dipendenze da questo repository durante il processo di compilazione.

2.3.4 Go

Go è un linguaggio di programmazione open source sviluppato da Google, progettato per essere semplice, efficiente, conciso e altamente performante, soprattutto in contesti di programmazione concorrente. Il task T4 è stato realizzato utilizzando Go; ecco alcune delle caratteristiche principali e dei vantaggi di tale linguaggio:

- **Efficienza e prestazioni:** Go è noto per le sue performance elevate e l'efficienza nell'uso delle risorse, dunque rappresenta una validissima opzione per applicazioni che richiedono elevata concorrenza e scalabilità;
- **Gestione della concorrenza:** Go sfrutta un modello di concorrenza semplificato e dei thread leggeri (goroutine) che gli consentono gestire un gran numero di richieste simultanee, rendendolo adatto per applicazioni web ad alte prestazioni;
- **Facilità di distribuzione:** gli eseguibili Go sono statici, semplificando il processo di deployment delle applicazioni in quanto non devono essere ricompilati ogni volta vengono modificati.

2.3.5 Postman

Postman è uno strumento molto popolare utilizzato nello sviluppo e nella gestione di API; si tratta di un client API che consente agli sviluppatori di testare, sviluppare e documentare le API in modo efficiente. Ecco alcuni dei principali vantaggi di questa piattaforma:

- **Sviluppo e Debugging:** gli sviluppatori possono utilizzare Postman durante la fase di sviluppo per inviare rapidamente richieste alle API e verificare il comportamento delle risorse; inoltre è particolarmente utile durante la fase di debugging per identificare e risolvere eventuali problemi;

- **Testing:** Postman consente agli sviluppatori di inviare richieste HTTP alle API e ricevere le risposte, dunque grazie a esso è possibile testare le API in modo completo, esaminare i dati di risposta e verificare che le risorse siano correttamente implementate;
- **Documentazione:** Postman offre strumenti per la creazione di documentazione dettagliata delle API, rendendo più facile per gli sviluppatori e gli utenti capire come utilizzare correttamente le API;
- **Collaborazione:** Postman facilita la collaborazione tra membri del team di sviluppo, dato che rende possibile una facile condivisione di raccolte di richieste, ambienti di lavoro e documentazione delle API tra membri del team.

Capitolo 3

Requisito R2

Il requisito assegnatoci consiste nella progettazione e nello sviluppo delle funzionalità necessarie al mantenimento delle informazioni riguardanti le partite giocate da ciascun player. Di seguito è riportata la sua formulazione:

Requisito R2: *verificare e gestire la coerenza dei dati della partita del giocatore attualmente salvati nel File System di T8 con quelli mantenuti nel database del gioco in T4. Eventualmente rivedere le API offerte da T4 per consentire di salvare anche i dati relativi ai Turni di gioco in T4.*

3.1 Analisi preliminare e problematiche

I nostri primi passi per comprendere meglio il requisito sono stati la consultazione delle documentazioni dei task T4 e T8 realizzate in pre-

cedenza dai nostri colleghi e i numerosi tentativi di installazione del gioco. Questa seconda parte si è rivelata particolarmente ardua per noi in quanto l'hardware delle nostre macchine non permetteva o di installare il gioco o di eseguirlo senza alcuna problematica; inoltre il gioco stesso presentava diversi malfunzionamenti a livello software. Tale problematica ci ha rallentato notevolmente, in ogni caso siamo riusciti poi a superarla grazie al contributo dei team che hanno lavorato ai task T1 e T2-T3, che hanno risolto il problema del caricamento delle classi e della generazione dei livelli dei robot.

Un ulteriore motivo di rallentamento per il nostro team è stato il requisito stesso, in quanto consultando la documentazione del T8 e le cartelle del progetto relative a tale task ci siamo trovati davanti a una incongruenza, dato che nel File System del T8 non vengono salvati i dati relativi alle partite dei giocatori ma solo quelli relativi ai robot EvoSuite e le metriche di copertura delle classi testate. Analizzando meglio il progetto ci siamo successivamente resi conto che il File System di interesse per noi fosse quello del task T5 dato che lì avvengono i salvataggi dei dati relativi a giocatori, partite, round e turni; ciò è stato fondamentale al fine di soddisfare il requisito richiesto, cioè garantire la coerenza con i corrispettivi dati salvati nel database di T4. Abbiamo dunque consultato la documentazione del T5 per comprenderne la struttura, il pattern architetturale e le API esposte, iniziando poi il nostro lavoro di *refactoring* per migliorare manutenibilità e mo-

dularità del codice, correggendo inoltre le incongruenze di salvataggio, per poi implementare metodi nuovi laddove servisse ulteriore lavoro per garantire la coerenza tra i dati salvati. È stata necessaria inoltre la modifica preliminare della struttura del File System di T5 al fine di renderla più comprensibile e adatta alle modifiche richieste.

3.2 Refactoring e nuove implementazioni in T4

Le API messe a disposizione sono state modificate al fine di implementare il requisito richiesto. In primo luogo però abbiamo provveduto alla modifica di alcuni diagrammi messi a disposizione dai colleghi che hanno lavorato al T4 precedentemente. In particolare, è stato necessario modificare in primo luogo l'Activity Diagram che rappresenta il flusso delle partite. Dato che, nella nostra versione, abbiamo previsto l'esistenza di un unico Round (posto dunque al valore di default 1 e non eliminato del tutto per non mettere in difficoltà gli altri gruppi, dato che la totale eliminazione avrebbe comportato delle ricadute sull'intero progetto). Tale Round, sarà diviso in più turni che saranno giocati dal giocatore e nel quale il giocatore testerà una classe, il turno terminerà nel momento in cui il giocatore, una volta finita la stesura della classe di test deciderà di sfidare il robot. Dunque, ciascun turno è caratterizzato da un vincitore. Per la determinazione del vincitore

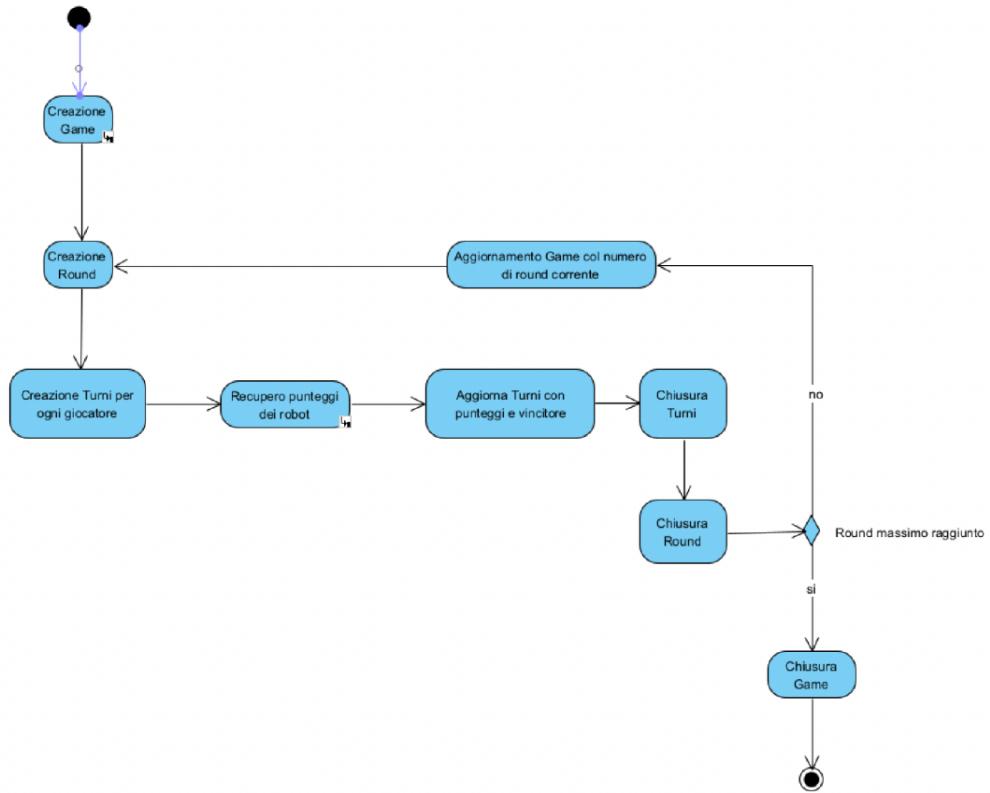


Figura 3.1: Diagramma di attività originale: flusso di una partita

dell’intera partita, si effettuerà una media dei punteggi, che sarà poi attribuita al giocatore vincitore. In Figura 3.1 e in Figura 3.2 si riportano rispettivamente la versione originale e la versione aggiornata dell’Activity Diagram che descrive il flusso di una partita.

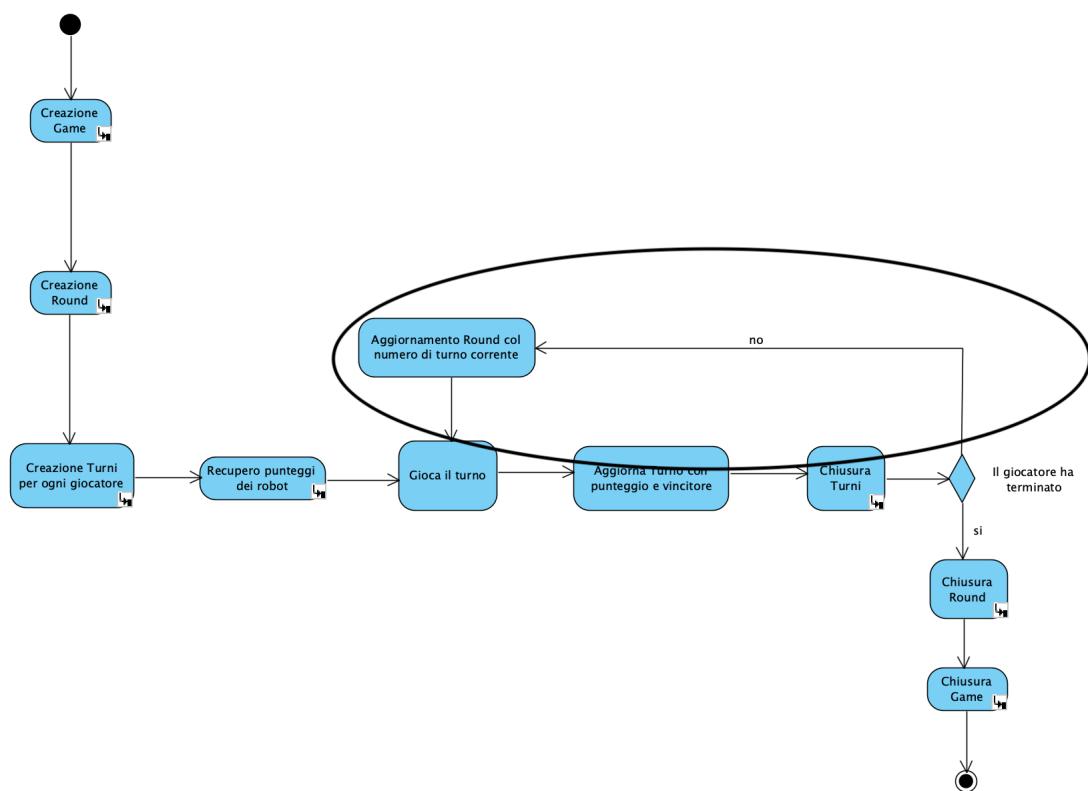


Figura 3.2: Diagramma di attività aggiornato: flusso di una partita

Successivamente è stata necessaria la modifica del diagramma ER, descritto nel Capitolo 1 di questo documento (rimandiamo al sottoparagrafo 1.4.1: Stato del task T4). Le modifiche sono state necessarie perché in primo luogo abbiamo notato che nella versione originale l'entità **Robot** non era collegata ad alcuna altra entità, quando in realtà questa è fondamentale, pertanto abbiamo previsto una relazione "**1 a N**" tra le entità **Robot** e **Partita** dato che un Robot può giocare più partite, mentre una partita può essere giocata da uno e un solo Robot. Inoltre, è stato necessario aggiungere un'associazione "**uno a uno**" tra le entità **Player** e **Turn**, dato che un turno può essere giocato da un solo player non essendo prevista la modalità multi giocatore e, inoltre, un giocatore può giocare un solo turno alla volta.

Infine, abbiamo provveduto alla modifica di diversi attributi alle entità, come si può evincere dalla Figura 3.3. In particolare, è stato inserito un attributo **Round** all'interno dell'entità **Game**, al quale si attribuirà sempre il valore di default "1". Inoltre, abbiamo deciso di eliminare l'attributo **Order** dall'entità **Round** dato che non abbiamo più la necessità di tener contezza dell'ordine di esecuzione dei Round, essendo ora unico.

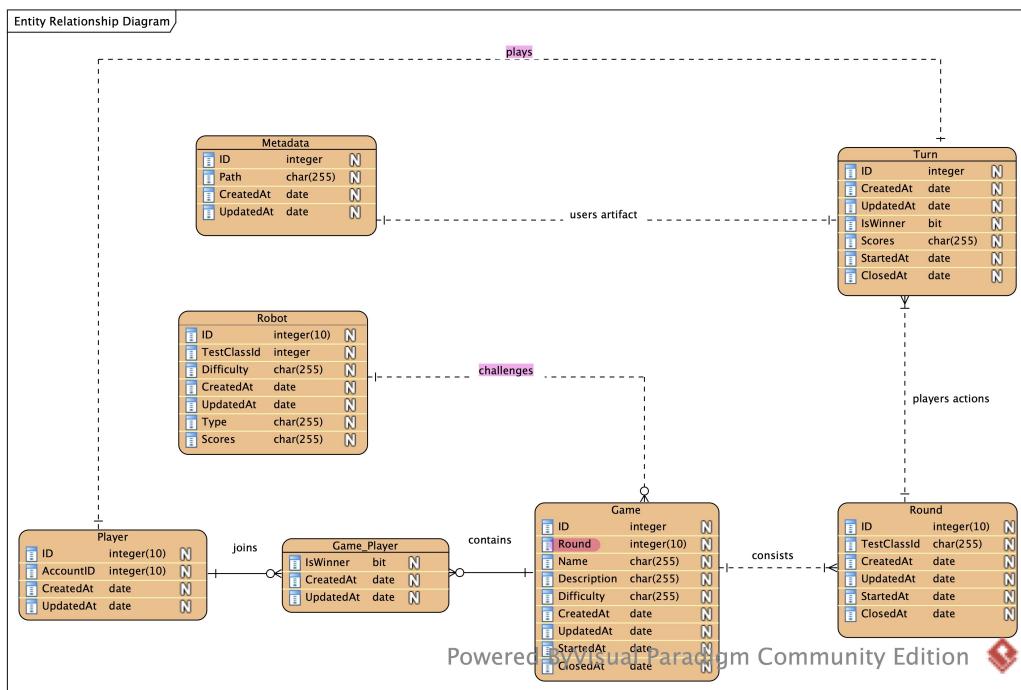


Figura 3.3: Diagramma ER aggiornato

Il System Domain Model è stato modificato in seguito dei precedenti cambiamenti attuati al diagramma ER. Le modifiche sono state apportate a livello del **Package Model** e inoltre sono stati modificati i **DTO** dei vari package per adattarli alla nuova versione del diagramma ER. Come è possibile notare, in entrambe le versioni del Package Model, abbiamo previsto la presenza della classe associativa **Player-Game**, questa scelta non è casuale, dato che nonostante ad oggi il gioco non preveda la modalità multi-giocatore, non abbiamo voluto precludere la possibilità di implementarla a coloro che in futuro vi lavoreranno. Nella Figura 3.4 è possibile osservare la versione aggiornata del diagramma.

Tenendo presente questi cambiamenti, abbiamo modificato di con-

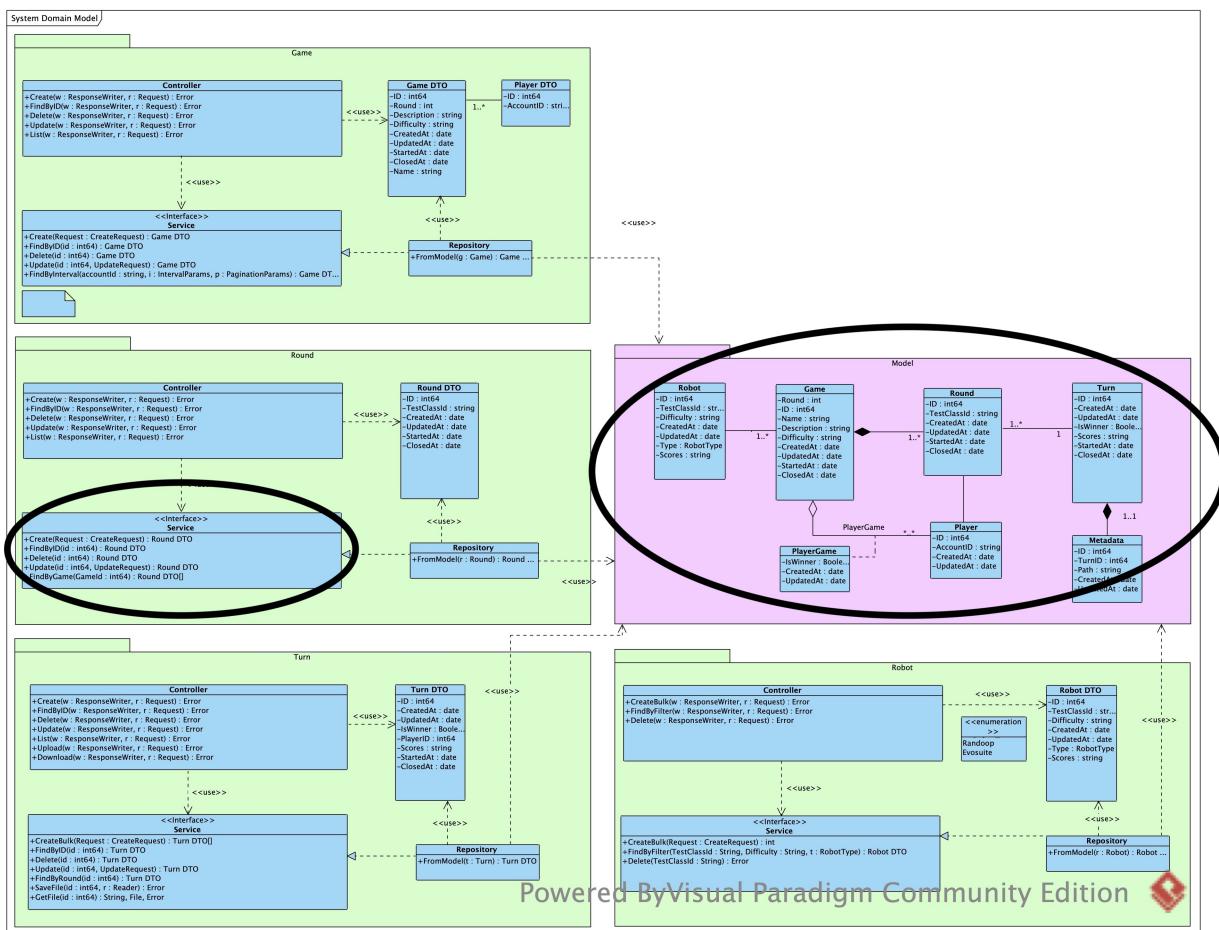


Figura 3.4: System Domain Model

seguenza il codice. In particolare, siamo partiti dall'adattare il Model alle novità, proseguendo poi con la correzione delle API ove necessario. Le cartelle e i file di cui si parlerà sono presenti nella directory `/T4-G18` del nostro progetto.

3.2.1 model

Nel file `model.go` presente in questa cartella vengono definite le tabelle e le relazioni presenti nel Game Repository e che vengono memorizzate dunque nel database Postgres. Abbiamo dovuto modificare gli attributi relativi a Game, Turn, Round e Metadata. Per Round abbiamo dovuto solo rimuovere l'attributo Order, mentre nelle Figure a seguire possiamo visualizzare le modifiche delle altre entità.

```
type Game struct {
    ID      int64      `gorm:"primaryKey;autoIncrement"`
    Name   string     `gorm:"not null"`
    Round  int        `gorm:"default:1"`
    Class  string     `gorm:"not null"`
    Description sql.NullString `gorm:"default:null"`
    Difficulty string    `gorm:"not null"`
    CreatedAt time.Time `gorm:"autoCreateTime"`
    UpdatedAt time.Time `gorm:"autoUpdateTime"`
    StartedAt *time.Time `gorm:"default:null"`
    ClosedAt  *time.Time `gorm:"default:null"`
    Players   []Player   `gorm:"many2many:player_games;foreignKey:PlayerID;constraint:OnUpdate:CASCADE,OnDelete:CASCADE"`
    Robot    int64      `gorm:"default:null"`
    // Future implementazione di partite con più rounds
    // Rounds   []Round    `gorm:"foreignKey:GameID;constraint:OnUpdate:CASCADE,OnDelete:CASCADE"`
}
```

Figura 3.5: Game model.go

3.2.2 api

Aggiornato il model, abbiamo dovuto aggiornare le API definite in questa cartella, aggiungendo i nuovi attributi necessari per gli oggetti

```

type Turn struct {
    ID      int64      `gorm:"primaryKey"`
    RoundID int64      `gorm:"primaryKey"`
    CreatedAt time.Time `gorm:"autoCreateTime"`
    UpdatedAt time.Time `gorm:"autoUpdateTime"`
    StartedAt *time.Time `gorm:"default:null"`
    ClosedAt *time.Time `gorm:"default:null"`
    Metadata Metadata   `gorm:"foreignKey:TurnID,RoundID;references:ID,RoundID"`
    Scores  string     `gorm:"default:null"`
    IsWinner bool       `gorm:"default:false"`
    PlayerID int64     `gorm:"index:idx_playerturn;not null"`
}
    
```

Figura 3.6: Turn model.go



Figura 3.7: CreateRequest game.go aggiornata

trasferiti e per effettuare le richieste di creazione e modifica in database. Di seguito sono riportate le modifiche apportate alle API offerte dal task T4. Per quanto riguarda l'API di Game, come possiamo vedere in Figura 3.7, è stata modificata la struct **CreateRequest** attraverso l'aggiunta di un attributo *StartedAt*, di tipo *date*, e la sostituzione dell'attributo **description** con un altro di tipo *String*, cioè **class**.

Nell'API di Round invece sono stati rimossi i riferimenti al vecchio attributo *Order* che l'entità aveva inizialmente (Figura 3.8).

```

type Round struct {
    ID      int64      `json:"id"`
    TestClassId string    `json:"testClassId"`
    GameID   int64      `json:"gameId"`
    CreatedAt time.Time `json:"createdAt"`
    UpdatedAt time.Time `json:"updatedAt"`
    StartedAt *time.Time `json:"startedAt"`
    ClosedAt  *time.Time `json:"closedAt"`
}

type CreateRequest struct {
    GameId      int64      `json:"gameId"`
    TestClassId string    `json:"testClassId"`
    StartedAt   *time.Time `json:"startedAt,omitempty"`
    ClosedAt    *time.Time `json:"closedAt,omitempty"`
}
    
```

Figura 3.8: Modifiche in round.go

Infine, a livello dell'API di Turn, sono state aggiornate la **CreateRequest** e la **UpdateRequest** con i dati necessari per salvataggio e aggiornamento dei turni in database secondo la nuova gestione del meccanismo di gioco. In Figura 3.9 possiamo visualizzare tali modifiche.

```
type Turn struct {
    ID        int64      `json:"id"`
    IsWinner bool       `json:"isWinner"`
    CreatedAt time.Time `json:"createdAt"`
    UpdatedAt time.Time `json:"updatedAt"`
    PlayerID int64      `json:"playerId"`
    RoundID  int64      `json:"roundId"`
    Scores   string     `json:"scores"`
    StartedAt *time.Time `json:"startedAt"`
    ClosedAt  *time.Time `json:"closedAt"`
}

type CreateRequest struct {
    RoundId  int64      `json:"roundId"`
    Players  []string   `json:"players"`
    StartedAt *time.Time `json:"startedAt,omitempty"`
    ClosedAt  *time.Time `json:"closedAt,omitempty"`
    ID       int64      `json:"id"`
}

func (CreateRequest) Validate() error {
    return nil
}

type UpdateRequest struct {
    RoundId  int64      `json:"roundId"`
    Scores   string     `json:"scores,omitempty"`
    IsWinner bool       `json:"isWinner"`
    StartedAt *time.Time `json:"startedAt,omitempty"`
    ClosedAt  *time.Time `json:"closedAt,omitempty"`
}
```

Figura 3.9: Modifiche in turn.go

3.2.3 Aggiornamento dell'API Diagram

Complessivamente, possiamo osservare le modifiche apportate all'API REST esposta da T4 nel nuovo **API Diagram**. Tenendo presente la versione precedente (Figura 1.5), vediamo singolarmente le operazioni che sono cambiate. Per ogni operazione illustreremo scopo, dati di input e output, evidenziando in grassetto quelli che abbiamo aggiunto o modificato.

Game: la creazione del Game viene effettuata tramite una POST

alla route `/games`, i dati in ingresso necessari sono quelli visti nella CreateRequest, per cui in Figura 3.10 possiamo avere un riscontro delle modifiche suddette. Presentiamo gli input più nel dettaglio:

- **name:** stringa contenente il nome della partita;
- **players:** array di stringhe contenente i riferimenti ai giocatori che partecipano alla partita;
- **class:** stringa contenente il nome della classe da testare;
- **difficulty:** stringa contenente la difficoltà del livello scelto per il robot da sfidare;
- **startedAt:** data contenente l'orario di inizio della partita.

Se l'operazione va a buon fine, viene restituito il corrispondente oggetto DTO del Game appena creato, utile per poter conoscere il suo identificativo in database. In caso contrario viene restituito errore.

Round: la creazione del Round viene effettuata tramite una POST alla route `/rounds`, i dati in ingresso necessari sono quelli visti nella CreateRequest, per cui in Figura 3.11 possiamo avere un riscontro delle modifiche suddette. Presentiamo gli input più nel dettaglio:

- **gameId:** numero *long* contenente l'identificativo della partita, in modo tale che il round possa essere associato univocamente alla partita in cui si trova;

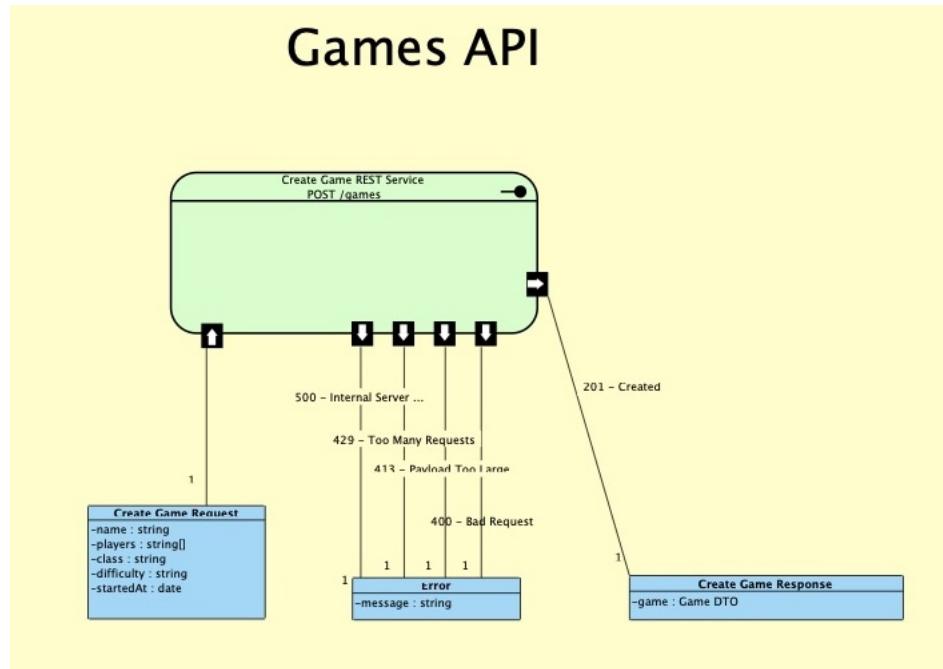


Figura 3.10: Create Game aggiornata

- `testClassId`: stringa contenente il riferimento alla classe da testare;
- `startedAt`: data contenente l’orario di inizio del round;
- `closedAt`: data contenente l’orario di fine del round.

Non c’è più l’intero *Order* che in origine avrebbe dovuto identificare la posizione del round all’interno della sequenza di Round che si sarebbero dovuti succedere in un Game. Se l’operazione va a buon fine, viene restituito il corrispondente oggetto DTO del Round appena creato, utile per poter conoscere il suo identificativo in database. In caso contrario viene restituito errore.

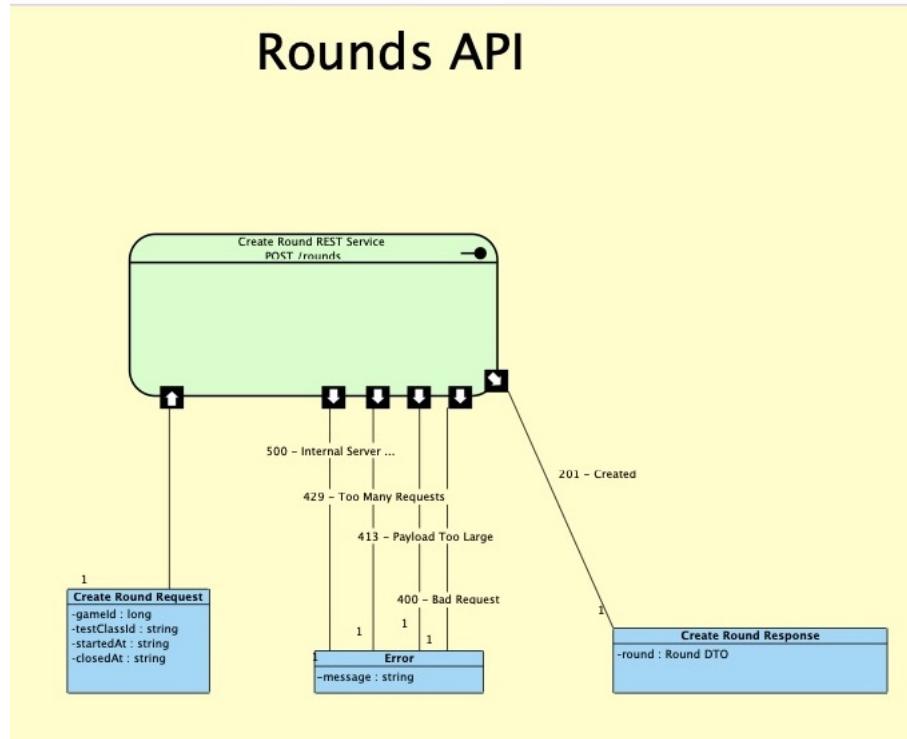


Figura 3.11: Create Round aggiornata

Turn: la creazione del Turn viene effettuata tramite una POST alla route `/turns`, i dati in ingresso necessari sono quelli visti nella CreateRequest, per cui in Figura 3.12 possiamo avere un riscontro delle modifiche suddette. Presentiamo gli input più nel dettaglio:

- **id:** numero *long* contenente la posizione del turno nel round;
- **roundId:** numero *long* contenente l'identificativo del round, in modo tale che il turn possa essere associato univocamente al round in cui si trova;
- **players:** array di stringhe contenente i riferimenti ai giocatori che partecipano al turno;
- **startedAt:** stringa contenente l'orario di inizio del turno;

- **closedAt:** stringa contenente l'orario di fine del turno.

Se l'operazione va a buon fine, viene restituito il corrispondente oggetto DTO del Turn appena creato, più specificamente un array contenente un singolo Turn che abbiamo lasciato invariato per le future implementazioni. In caso contrario viene restituito errore.

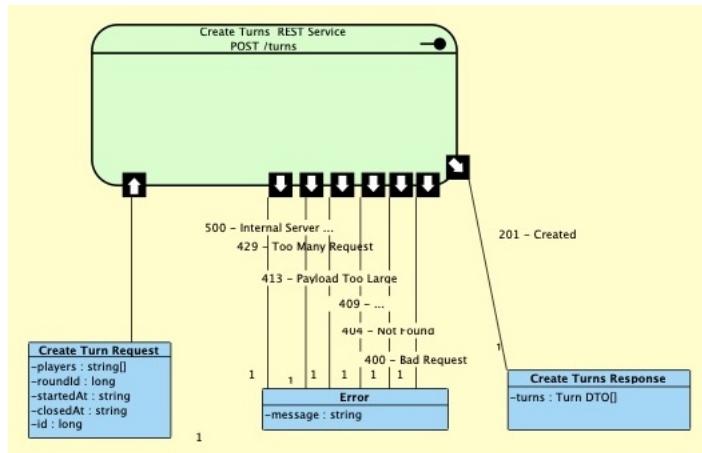


Figura 3.12: Create Turn aggiornata

L'aggiornamento del Turn viene effettuato tramite una PUT alla route `/turns/id`, i dati in ingresso necessari sono quelli visti nella UpdateRequest, per cui in Figura 3.13 possiamo avere un riscontro delle modifiche sudette. Presentiamo gli input più nel dettaglio:

- **id:** numero intero contenente la posizione del turno nel round;
- **roundId:** numero *long* contenente l'identificativo del round a cui il turn appartiene;
- **scores:** stringa contenente il punteggio realizzato dal giocatore al termine di quel turno;

- isWinner: booleano che indica se il giocatore è riuscito a battere il robot o meno;
- startedAt: stringa contenente l'orario di inizio del turno;
- closedAt: stringa contenente l'orario di chiusura del turno;

Se l'operazione va a buon fine, viene restituito il corrispondente oggetto DTO del Turn appena creato. In caso contrario viene restituito errore.

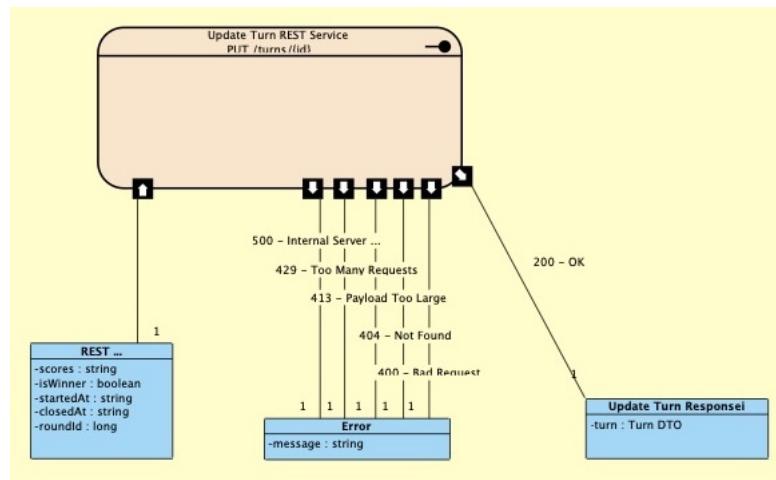


Figura 3.13: Update Turn aggiornata

3.3 Refactoring e nuove implementazioni in T5

Prima di agire sul codice, c'è stato bisogno di una riorganizzazione della directory dello Student Repository. In Figura 3.14 osserviamo la precedente gerarchia delle cartelle all'interno del File System dello Student Repository, mentre in Figura 3.15 vediamo la nuova struttura

della cartella StudentLogin, che risulta migliorata in quanto qui ora vengono create di volta in volta apposite cartelle sulla base degli id del giocatore, della partita, del round e del turno, evitando sovrascritture di file indesiderate che invece si verificavano precedentemente.

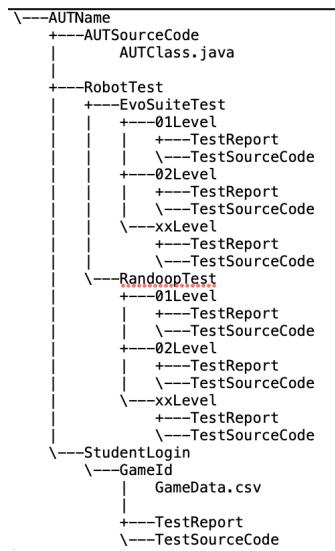


Figura 3.14: Directory dello Student Repository vecchia

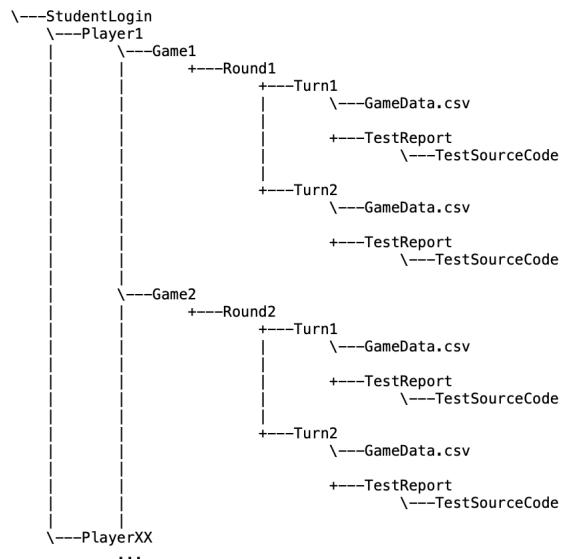


Figura 3.15: Directory StudentLogin aggiornata

3.3.1 Game.java

Le prime modifiche effettuate riguardano il **Model**; in particolare, per garantire la coerenza dei dati salvati in Database e quelli in File System, abbiamo dovuto modificare la classe **Game** presente nel package Model per adattarla alla struttura del Database di T4. Tenendo presente il Class Diagram di T5 visto prima in Figura 1.6 e il nuovo diagramma ER di T4 visto in Figura 3.3, abbiamo modificato di conseguenza la classe Game, come possiamo vedere in Figura 3.16. Anche qui dunque non sono più presente il campo *currentRound* bensì *round* che verrà impostato sempre a un valore di default (dato che il nostro scenario prevede solo un Round), mentre sono stati aggiunti i campi *playerId* e *robot*. In realtà, *playerId* sarebbe dovuto essere un vettore di id, ma abbiamo deciso di utilizzare un valore singolo poiché ci siamo concentrati sullo scenario single-player, lasciando l'introduzione dello scenario multi-player alle future implementazioni. Inoltre, per il momento, in *robot* viene salvata una stringa che può essere "randoop" oppure "evosuite", ma in futuro dovrà diventare un campo numerico che conterrà l'id in database dello specifico robot affrontato. Vediamo allora, in Figura 3.17, come è cambiato il Class Diagram del Model in seguito alla modifica della classe Game; le parti evidenziate rappresentano le novità.

```
public class Game {  
    private long id;  
    private String name;  
    private int round;  
    private String testedClass;  
    private String description;  
    private String difficulty;  
    private String createdAt;  
    private String updatedAt;  
    private String startedAt;  
    private String closedAt;  
    private long playerId; // Adattare per il multi-player  
    private String robot; // Adattare a long
```

Figura 3.16: Classe Game nel Model di T5 aggiornata

3.3.2 GameDataWriter.java

Una volta corretto il Model, la prima delle due classi del **Controller** che abbiamo modificato e migliorato è **GameDataWriter**, classe contenente i metodi che si occupano singolarmente di effettuare creazioni e aggiornamenti in database (T4) e in file system (T5). I metodi che presenteremo sono stati creati da zero oppure modificati per garantire la coerenza tra i salvataggi nelle due diverse posizioni e vengono utilizzati dall'altra classe che costituisce il Controller e che vedremo successivamente, cioè *GUIController*. Preliminariamente, abbiamo aggiunto a *GameDataWriter* due proprietà statiche:

```
private static String CSV_FILE_PATH = "AUTName/StudentLogin/";  
private static String CSV_FILE_NAME = "/GameData.csv";
```

che rappresentano delle parti fisse del percorso di salvataggio dei dati su file system dello Student Repository.

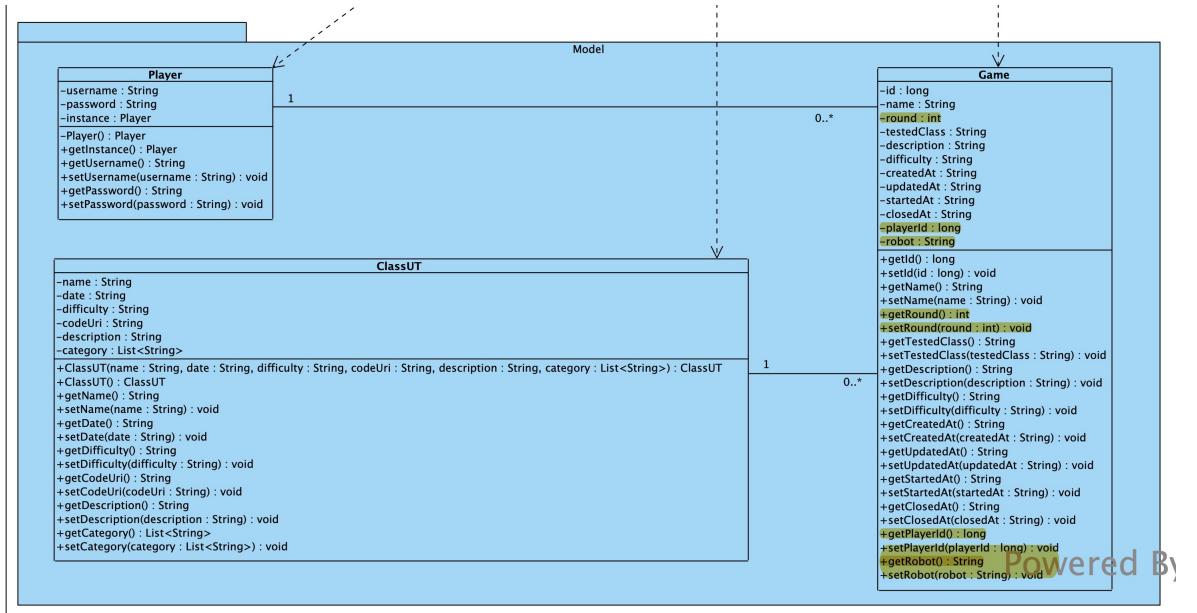


Figura 3.17: Class Diagram del Model in seguito alle nostre modifiche

3.3.3 saveGameCSV

Per effettuare il salvataggio su file system dei dati di una partita appena iniziata abbiamo implementato un nuovo metodo **saveGameCSV**, che si occupa di registrare i dati del game all'interno di un file CSV, che viene poi salvato in una directory che rispetta la struttura della nuova Student Repository. Innanzitutto, il metodo costruisce il percorso dove verrà salvato il file CSV sulla base degli id necessari, assicurandosi poi che tale percorso venga effettivamente creato in file system (Figura 3.18).

Successivamente, il metodo procede con la creazione del file *GameData.csv* e la scrittura dei record contenenti le informazioni della partita su tale file. Se tutte le operazioni vanno a buon fine, il met-

```
public boolean saveGameCSV(Game game, int tid) {
    long pid = game.getPlayerId();
    long gid = game.getId();
    int rid = game.getRound();

    String playerID = "Player" + pid;
    String gameID = "Game" + gid;
    String roundID = "Round" + rid;
    String turnID = "Turn" + tid;

    // Al path bisogna aggiungere PlayerID/GameID/RoundID/TurnID e poi il nome del file
    String fileName = CSV_FILE_PATH + playerID + "/" + gameID + "/" + roundID + "/" + turnID + CSV_FILE_NAME;

    Path path = Paths.get(fileName);

    if (!Files.exists(path)) {
        try {
            Files.createDirectories(path.getParent());
        } catch (IOException e) {
            System.out.println("Errore durante la creazione della directory.");
            e.printStackTrace();
        }
    }
}
```

Figura 3.18: Creazione del percorso

do restituisce *true*, altrimenti *false*, e gli esiti vengono stampati su terminale per fini di debugging (Figura 3.19).

```
try {
    File file = new File(fileName);

    if (!file.exists()) {
        file.createNewFile();
    }

    FileWriter writer = new FileWriter(file);

    CSVFormat csvFormat = CSVFormat.Builder.create().setDelimiter(delimiter).build();
    CSVPrinter csvPrinter = new CSVPrinter(writer, csvFormat);

    csvPrinter.printRecord(
        ...values:"GameID",
        "Name",
        "Round",
        "Class",
        "Description",
        "Difficulty",
        "Createdat",
        "UpdatedAt",
        "StartedAt",
        "ClosedAt",
        "PlayerID",
        "Robot"
    );

    csvPrinter.printRecord(
        game.getId(),
        game.getName(),
        game.getRound(),
        game.getTestedClass(),
        game.getDescription(),
        game.getDifficulty(),
        game.getCreatedat(),
        game.getUpdatedAt(),
        game.getStartedat(),
        game.getClosedAt(),
        game.getPlayerId(),
        game.getRobot()
    );

    csvPrinter.flush();
    csvPrinter.close();
    writer.close();
    System.out.println(gameID + " " + turnID + " salvato correttamente in File System.");
    return true;
} catch (IOException e) {
    e.printStackTrace();
    System.out.println("Errore durante la scrittura del file CSV.");
    return false;
}
```

Figura 3.19: Creazione e scrittura di GameData.csv

3.3.4 updateGameCSV

Analogamente al metodo visto prima, **updateGameCSV** si occupa di aggiornare i dati di gioco riguardanti il turno appena concluso, per cui sovrascriverà il file `gameData.csv` con le nuove informazioni.

3.3.5 createNewTurnCSV

Il metodo **createNewTurnCSV** serve semplicemente a richiamare `saveGameCSV` quando è necessaria la creazione di un nuovo file `gameData.csv` che dovrà memorizzare le informazioni sul nuovo turno appena iniziato, incrementandone in primis l'id.

3.3.6 GUIController.java

La seconda classe che costituisce il Controller è **GUIController**, un'importante classe dove, tramite il framework Spring, sono stati realizzati i metodi che si occupano di gestire gli input e le scelte dello studente che vengono trasferiti utilizzando i metodi CRUD di HTTP, in particolare vi sono una serie di PostMapping e GetMapping per le route necessarie. Prima di agire sui metodi presenti, abbiamo introdotto alcune novità, in particolare in seguito alle nostre aggiunte tale classe è ora dotata delle seguenti proprietà private:

```
private GameDataWriter gameDataWriter = new GameDataWriter();  
private Game g = new Game();
```

utilizzate rispettivamente per accedere ai metodi di `GameDataWriter` e per istanziare un oggetto `Game`. Inoltre, come possiamo vedere in

Figura 3.20, abbiamo aggiunto un metodo per ottenere quando necessario una stringa contenente la data e l'orario corrente (formattati secondo la convenzione italiana). Queste aggiunte serviranno per i metodi presentati di seguito.

```
private String getCurrentDateTime() {
    Date currentDate = new Date();
    SimpleDateFormat dateFormat = new SimpleDateFormat(pattern:"dd/MM/yyyy-HH:mm:ss");
    String formattedDate = dateFormat.format(currentDate);

    return formattedDate;
}
```

Figura 3.20: Metodo per ottenere data e ora corrente

3.3.7 receiveGameVariables

Tra i diversi metodi incompleti che abbiamo trovato commentati in `GUIController`, abbiamo recuperato e ultimato `receiveGameVariables`, che viene eseguito ogni volta viene effettuata una POST alla route `/sendGameVariables`. Nel passaggio dalla pagina di scelta di classe e robot alla pagina di riepilogo delle scelte dove è possibile avviare la partita, viene effettuata tale richiesta dallo script `main.js` lato web-app e vengono così inviate le prime variabili che verranno salvate nell'oggetto `g`, cioè `robot`, `classe` e `difficulty`, come possiamo vedere in Figura 3.21. Inoltre, il metodo si occupa di salvare in `g` anche data e ora della creazione della partita.

```
@PostMapping("/sendGameVariables")
public ResponseEntity<String> receiveGameVariables(@RequestParam("classe") String classe,
    @RequestParam("robot") String robot, @RequestParam("difficulty") String difficulty) {

    System.out.println("Classe ricevuta: " + classe);
    System.out.println("Robot ricevuto: " + robot);
    System.out.println("Difficoltà ricevuta: " + difficulty);

    g.setTestedClass(classe);
    g.setRobot(robot);
    g.setDifficulty(difficulty);
    g.setName(name:"nome");

    g.setCreatedAt(getCurrentDateTime());

    return ResponseEntity.ok(body:"Dati ricevuti con successo");
}
```

Figura 3.21: Salvataggio delle variabili di gioco nell'apposito oggetto

3.3.8 saveGame

Altro metodo di interesse per noi è stato **saveGame**, che viene eseguito quando lo script **main.js** della web app effettua una richiesta **POST** alla route */save-data* per creare una nuova partita; abbiamo aggiunto a tale metodo la chiamata a *saveGameCSV* e abbiamo modificato opportunamente l'assegnamento di alcune proprietà di *g*. Come possiamo vedere in Figura 3.22, *saveGame* riceve in input il *playerId* e popola di conseguenza l'oggetto *g*, in particolare riempie i campi *playerId* e *startedAt* tramite i metodi setter definiti nel Model. Successivamente, effettua una chiamata al metodo omonimo *saveGame* definito in *GameDataWriter* affinche vengano creati dei nuovi game, round e turn in database, ottenendo poi i loro id, che vengono utilizzati per riempire ulteriormente i campi *id* e *round* di *g*. Procede poi a salvare le informazioni del game in File System chiamando il metodo

do *saveGameCSV* e infine restituisce come output una stringa JSON contenente gli ID di partita, round e turno precedentemente ottenuti, che verranno mantenuti dalla web-app come elementi locali.

```
@PostMapping("/save-data")
public ResponseEntity<String> saveGame(@RequestParam("playerId") long playerId,
                                         HttpServletRequest request) {

    if (!request.getHeader(name:"X-UserID").equals(String.valueOf(playerId)))
        return ResponseEntity.badRequest().body(body:"Unauthorized");

    g.setPlayerId(playerId);
    g.setStartedAt(getCurrentDateTime());

    JSONObject ids = gameDataWriter.saveGame(g);

    if (ids == null)
        return ResponseEntity.badRequest().body(body:"Bad Request");

    long gameID = ids.getLong(name:"game_id");
    int roundID = ids.getInt(name:"round_id");
    int turnID = ids.getInt(name:"turn_id");

    g.setId(gameID);
    g.setRound(roundID);

    boolean saved = gameDataWriter.saveGameCSV(g, turnID);

    if (!saved)
        return ResponseEntity.internalServerError().body(body:"Game not saved in filesystem");

    return ResponseEntity.ok(ids.toString());
}
```

Figura 3.22: Gestione della richiesta di creazione di una nuova partita

3.3.9 updateGame

Ogni volta che il player sfida il robot, si chiude un turno e ne comincia uno nuovo. Per gestire l'aggiornamento del turno concluso e la creazione di un nuovo turno, abbiamo realizzato un nuovo metodo in *GUIController*, cioè **updateGame**, impostando preliminarmente la nuova route apposita `/update-game`. Il funzionamento di tale metodo è analogo a quello di `saveGame`, viene infatti chiamata la funzione omonima `updateGame` in *GUIController* che aggiorna il database, poi `updateGameCSV` che invece aggiorna il File System. Inoltre avviene la chiamata a `createNextTurnCSV` per la creazione del nuovo turno in File System e viene restituito alla web-app l'id del turno appena creato. In Figura 3.23 possiamo visualizzare il funzionamento di `updateGame`.

```
@PostMapping("/update-data")
public ResponseEntity<String> updateGame(@RequestParam("playerId") long playerId,
    @RequestParam("turnId") int turnId,
    HttpServletRequest request) {

    if (!request.getHeader("X-UserID").equals(String.valueOf(playerId)))
        return ResponseEntity.badRequest().body(body:"Unauthorized");

    g.setUpdatedAt(getCurrentDateTime());

    boolean esito = false;

    esito = gameDataWriter.updateGame(g, turnId);

    if (!esito)
        return ResponseEntity.internalServerError().body(body:"Data not updated in database");

    esito = gameDataWriter.updateGameCSV(g, turnId);

    if (!esito)
        return ResponseEntity.internalServerError().body(body:"Game not updated in filesystem");

    esito = gameDataWriter.createNextTurnCSV(g, turnId);

    if (!esito)
        return ResponseEntity.internalServerError().body(body:"New turn not created in filesystem");

    JSONObject nextTurn = new JSONObject();

    nextTurn.put(name:"turn_id", turnId + 1);

    return ResponseEntity.ok(nextTurn.toString());
}
```

Figura 3.23: Gestione della richiesta di aggiornamento della partita

3.3.10 Aggiornamento della documentazione

Abbiamo creato due nuovi Sequence Diagram per rappresentare più nel dettaglio la creazione e l'aggiornamento dei dati delle partite: quello in Figura 3.24 rappresenta lo scenario del salvataggio dei dati in seguito a creazione e inizio di un game, mentre quello in Figura 3.25 mostra le operazioni necessarie per l'aggiornamento dei dati già presenti e la creazione del prossimo turno. Inoltre, in Figura 3.26, troviamo il Class Diagram aggiornato dello Student Repository dove mostriamo come sono stati modificati il Model e il Controller.

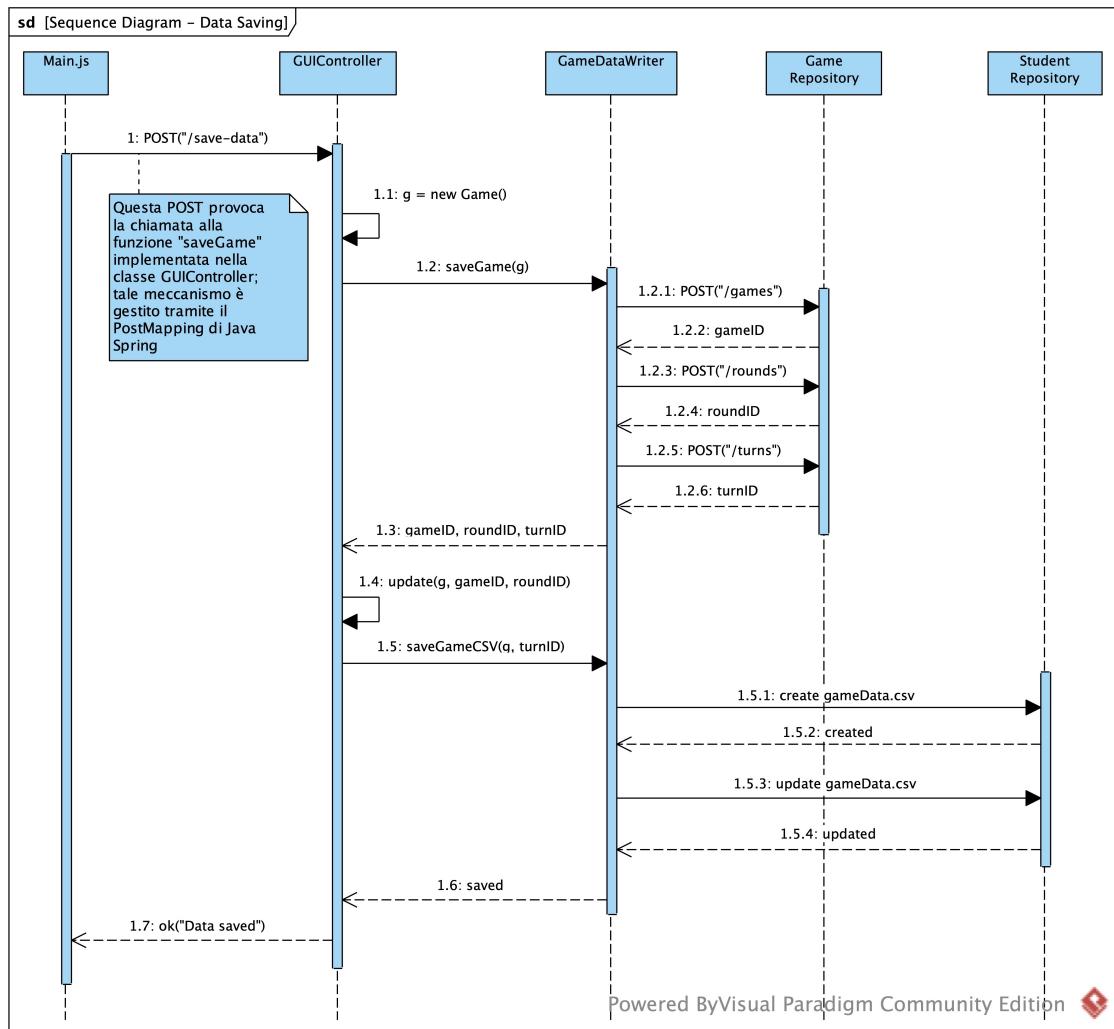


Figura 3.24: Sequence Diagram del salvataggio dati di un game

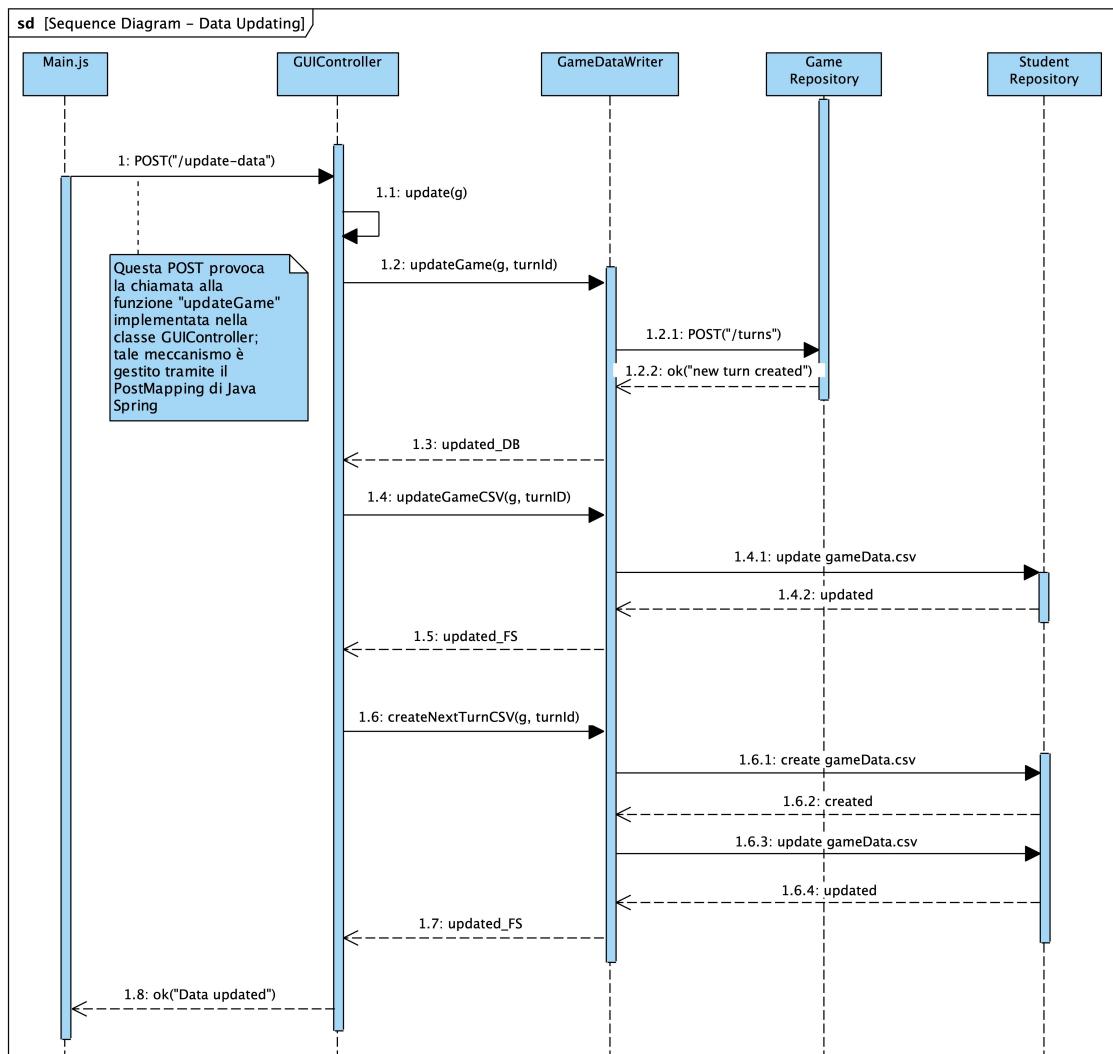


Figura 3.25: Sequence Diagram dell'aggiornamento dati di un game

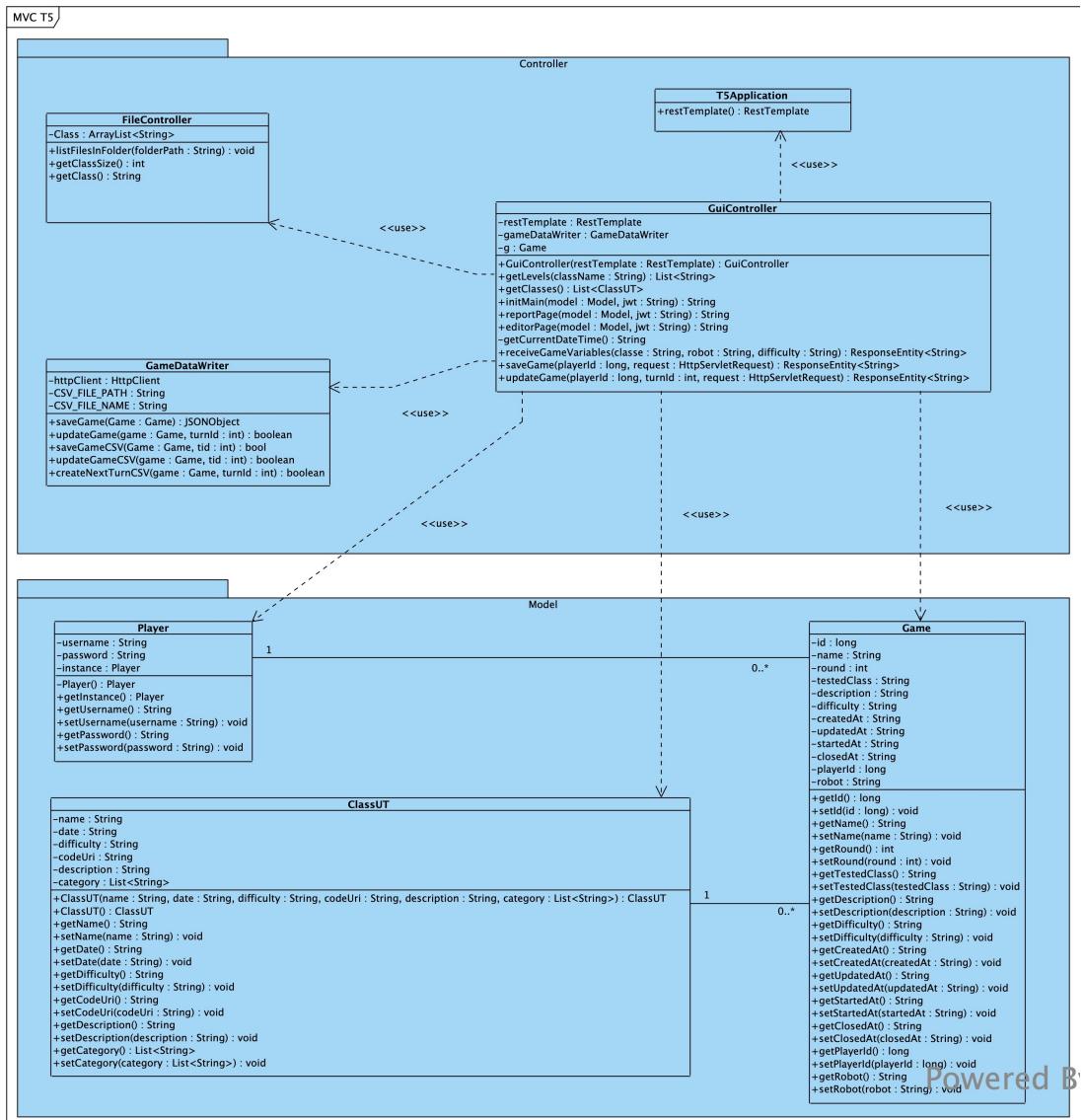


Figura 3.26: Class Diagram dello Student Repository aggiornato

3.4 Ricompilazione del progetto

Man mano che abbiamo effettuato le modifiche abbiamo dovuto anche ricompilare i componenti modificati e poi riavviare lo script di installazione dell'applicazione; informazioni dettagliate sull'installazione sono fornite in questa documentazione successivamente nel Capitolo 5.

Per quanto riguarda T4, essendo realizzato in linguaggio **Go**, questo componente risulta costituito da eseguibili statici che non devono essere ricompilati manualmente a ogni modifica, dunque i cambiamenti sono visualizzabili con la semplice reinstallazione dell'app. Suggeriamo di eliminare da Docker Desktop almeno container, immagini e volumi legati a T4 prima di riavviare lo script di installazione per poter visualizzare correttamente tutte le modifiche apportate; in generale suggeriamo una reinstallazione pulita del gioco.

Il componente T5 invece è gestito tramite **Maven**, essendo realizzato in buona parte in linguaggio **Java**, dunque per visualizzare le modifiche bisogna innanzitutto aggiornare l'artefatto **t5-0.0.1-SNAPSHOT.jar** ricompilando il progetto Maven presente in T5. Per farlo, preliminarmente bisogna aver installato Maven sulla propria macchina (consultare il seguente link: <https://maven.apache.org>). Bisogna poi aprire una nuova finestra del terminale, spostarsi nella cartella contenente il progetto dell'app e poi nella cartella *T5-G2/t5*, dove è presente il file

di configurazione del progetto Maven **pom.xml**. Supponendo che la cartella del progetto si chiami *T4-A3_2.0* e si trovi sul nostro Desktop, da terminale dovremmo eseguire i seguenti comandi:

```
cd Desktop/T4-A3_2.0/T5-G2/t5  
mvn clean install
```

Al termine dell'esecuzione il nuovo artefatto sarà pronto e potrà essere installato su Docker Desktop facendo partire lo script di installazione del progetto presente nella cartella principale del progetto *T4-A3_2.0*. Anche in tal caso, rimuovere almeno container, immagini e volumi legati a T5 prima di deployare su Docker il nuovo artefatto tramite lo scritto, si consiglia comunque un'installazione pulita cancellando tutto da Docker Desktop.

3.5 Testing

Per testare le novità da noi introdotte, abbiamo deciso di effettuare **testing di integrazione**. Ci siamo infatti concentrati sul verificare il funzionamento dei metodi da noi implementati all'interno della web-app stessa per capire se ci fosse una corretta interazione del componente T5 con l'API REST realizzata dal componente T4.

Prima di procedere con il test di integrazione, abbiamo verificato che le route da noi create e/o utilizzate rispondessero come da noi previsto

quando vengono effettuate determinate richieste HTTP. A tal proposito, abbiamo sfruttato **Postman**, disabilitando momentaneamente il filtro autenticazione realizzato nel componente *api_gateway* tramite *Zuul*. I nostri test delle route non sono altro che delle richieste HTTP di POST contenenti nel request body i parametri che la route si aspetta; le nostre route erano configurate correttamente e hanno risposto alle nostre richieste con stato *200:OK*. In Figura 3.27 possiamo osservare la POST effettuata verso la route */api/sendGameVariables* da noi configurata.

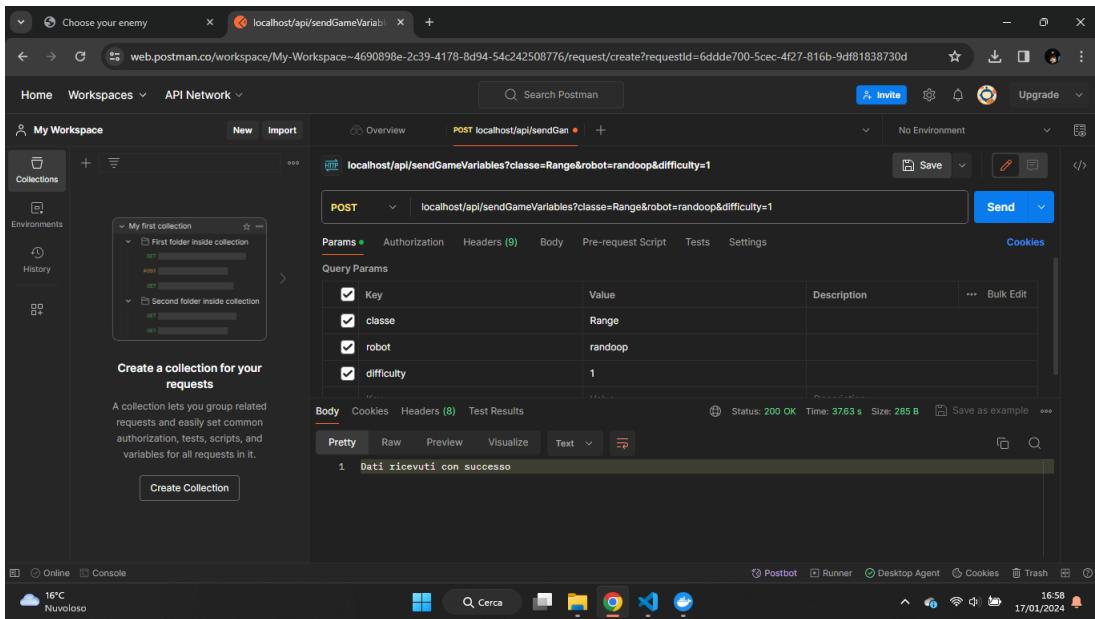


Figura 3.27: POST dei dati inerenti a classe e robot usando Postman

Successivamente, per eseguire i test di integrazione, ci siamo serviti di **Selenium**, una suite di strumenti ampiamente utilizzata per l’automazione dei test funzionali su applicazioni web, consentendoci di automatizzare le interazioni con il browser per eseguire test ripetibili, migliorare la qualità del software e accelerare i processi di sviluppo.

luppo. Selenium fornisce un linguaggio di dominio (*DSL*), in quanto, per automatizzare l’interazione con il browser, c’è bisogno delle features che fornisce un *DSL*, come ad esempio l’astrazione. In questo modo è possibile scrivere codice con Selenium indipendentemente dal browser settato nel *WebDriver*. Inoltre, un *DSL* risulta utile per la scrittura di test su alcuni tra i maggiori linguaggi di programmazione, tra cui Java. Il primo passo per utilizzarla è stato consultare la documentazione di tale suite (visitabile al seguente link: <https://www.selenium.dev/documentation/>). Per integrare Selenium nel nostro progetto ed eseguire i nostri test dunque abbiamo modificato il file di configurazione Maven **pom.xml** presente in T5, aggiungendo le diverse dipendenze di Selenium e la dipendenza del *WebDriver*, come è possibile vedere in Figura 3.28.

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-chrome-driver</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>4.4.3</version>
</dependency>
</dependencies>
```

Figura 3.28: Integrazione di Selenium attraverso Maven

Il passo successivo è stato creare **T5ApplicationTest.java** nell’apposita cartella */src/test* che Maven genera per le classi di test. Il test introdotto crea un’istanza di **ChromeDev** in cui viene caricata la

schermata di login, vengono immessi username e password (nel codice bisogna cambiare quelli presenti con le credenziali che l’utente reale ha usato per autenticarsi) e viene premuto il tasto di login. Tramite `wait.until()`, si attende il caricamento della pagina `main`, e dopodiché si fa corrispondere la stringa `ExpectedNextPage` al titolo della pagina che ci si aspetta venga caricata e la stringa `actualNextPage` al `driver.getTitle()`, così da ottenere il titolo della pagina attuale. Successivamente, viene effettuata una chiamata al metodo `assertTrue()`. Se `actualNextPage` è uguale a `expectedNextPage`, l’asserzione `assertTrue` risulterà essere vera e il test avrà esito positivo (Figura 3.29)

```

22  @RunWith(SpringRunner.class)
23  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)
24  public class T5ApplicationTests {
25
26      public static WebDriver driver;
27      public static Duration timeout = Duration.ofMillis(20000);
28
29  }
30  @Test
31  public void startGameTest() {
32      System.setProperty(key:"webdriver.chrome.driver", value:"src/test/java/drivers/chromedriver.exe");
33
34      ChromeOptions options = new ChromeOptions();
35      options.setBinary("src/test/java/chrome-win64/chrome.exe");
36
37      HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
38      chromePrefs.put(key:"profile.default_content_settings.popups", value:0);
39      options.setExperimentalOption("prefs", chromePrefs);
40
41      driver = new ChromeDriver(options);
42
43      driver.manage().timeouts().implicitlyWait(timeout);
44      driver.manage().window().maximize();
45
46      driver.get("http://localhost/login");
47      driver.findElement(By.id("email")).sendKeys("your@email.com");
48      driver.findElement(By.id("password")).sendKeys("yourPassword");
49      driver.findElement(By.cssSelector("input[type=submit]")).click();
50
51      WebDriverWait wait = new WebDriverWait(driver, timeout);
52
53      String urlMain = "http://localhost/main";
54
55      try {
56          wait.until(ExpectedConditions.urlToBe(urlMain));
57      } catch(TimeoutException e) {
58          Assert.fail();
59      }
60  }

```

Figura 3.29: SaveGameTest (prima parte)

Dopodiché, lo stesso procedimento viene ripetuto per:

- il passaggio dalla schermata *main* a *report*, in tal caso il tool si occupa di selezionare classe e robot e poi di cliccare sul pulsante *Submit*; ci permette di verificare la corretta esecuzione della POST verso */sendGameVariables*;
- il passaggio dalla schermata *report* a *editor*, in tal caso il tool si occupa solo di cliccare sul pulsante *Submit*; ci permette di verificare la corretta esecuzione della POST verso */save-data*.

Nel caso in cui la pagina dove si giunge sia diversa da quella attesa, dunque in caso di errore nell'invio dei dati durante la navigazione, l'asserzione sarà falsa e il test avrà esito negativo, altrimenti positivo (Figura 3.30).

```
29
30     String urlReport = "http://localhost/report";
31
32     driver.findElement(By.id("0")).click();
33     driver.findElement(By.id("0-1")).click();
34
35     WebElement submitButton = driver.findElements(By.cssSelector(".custom-button")).get(index:1);
36
37     new WebDriverWait(driver, Duration.ofSeconds(seconds:20)).until(ExpectedConditions.elementToBeClickable(submitButton));
38
39     submitButton.click();
40
41     new WebDriverWait(driver, Duration.ofSeconds(seconds:20)).until(ExpectedConditions.urlToBe(urlReport));
42
43     String urlEditor = "http://localhost/editor";
44
45     submitButton = driver.findElements(By.cssSelector(".custom-button")).get(index:1);
46
47     new WebDriverWait(driver, Duration.ofSeconds(seconds:20)).until(ExpectedConditions.elementToBeClickable(submitButton));
48
49     submitButton.click();
50
51     new WebDriverWait(driver, Duration.ofSeconds(seconds:20)).until(ExpectedConditions.urlToBe(urlEditor));
52
53     Assert.assertEquals(message:"Test fallito: i dati non sono stati inviati.", driver.getCurrentUrl(), urlEditor);
54
55 }
```

Figura 3.30: SaveGameTest (seconda parte)

Per replicare i test, scaricare preliminarmente la versione di ChromeDev presente al link <http://googlechromelabs.github.io/chrome-for-testing/#dev> per il proprio sistema operativo e modificare il percorso specificato in `options.setBinary()` facendo riferimento alla posizione corretta dell'eseguibile *chrome* appena scaricato sulla propria macchina. Quando si è pronti per eseguire il test, è necessario lanciare il comando *mvn test* dal terminale dopo che ci si è posizionati nella cartella */T5-G2/t5* oppure, da VSCode, cliccando sull'apposito pulsante triangolare verde per eseguire i test. In Figura 3.31 possiamo osservare il test che abbiamo realizzato e superato con successo per assicurarci

CAPITOLO 3. REQUISITO R2

The screenshot shows a Java IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Editor:** T5ApplicationTests.java (selected), main.html, main.js, GuiController.java, main.css, AuthenticationFilter.java.
- Code:** A snippet of Java code for a Selenium test. It includes imports for WebDriver, Duration, and ChromeOptions. It sets up a driver using ChromeDriver and maximizes the window. The test starts at <http://localhost/login>.
- Toolbars:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, TEST RESULTS, PORTS.
- Test Results:** A sidebar showing a list of test runs:
 - > Test run at 1/18/2024, 1:22:46 AM
 - Test run at 1/18/2024, 1:21:43 AM
 - > ○ Test run at 1/17/2024, 11:10:17 PM
 - > ○ Test run at 1/17/2024, 11:06:28 PM
 - > ○ Test run at 1/17/2024, 11:04:58 PM
- Status Bar:** Ln 26, Col 36, Tab Size: 4, UTF-8, CRLF, Java, 01:32, 18/01/2024.
- Bottom Icons:** Ultim'ora, Nottizie.

Figura 3.31: Test effettuato con Selenium

del corretto salvataggio dei dati nel passaggio da una vista all'altra dell'applicazione.

Capitolo 4

Organizzazione del lavoro

4.1 Introduzione a Scrum

Scrum è un framework Agile focalizzato sulla gestione dei progetti e lo sviluppo del software; è basato su principi e valori definiti nel "Manifesto Agile".

4.1.1 Adozione di Scrum per il nostro progetto

Nel contesto del progetto, abbiamo adottato il framework Scrum come metodologia di gestione e sviluppo per l'integrazione di nuove funzionalità. Il requisito di nostra competenza è stato sviluppato in quattro iterazioni, con un numero variabile di Sprint, per una durata complessiva di circa 3 mesi di lavoro. Ogni iterazione è iniziata con un Iteration Planning e ogni Sprint ha avuto un proprio Sprint Planning, suddividendo il lavoro in task assegnati ai membri del team. Abbiamo

utilizzato piattaforme come Miro per delineare i task da suddividerci e organizzare il lavoro di gruppo.

4.2 Prima iterazione

Durante la prima iterazione sono stati svolti i task relativi all’analisi della documentazione e la revisione del codice dei vari componenti del progetto, in particolare il componente T4. Ci siamo concentrati inoltre sull’analisi del componente T8, che solo successivamente si è rivelato non utile ai nostri fini; ciò è conseguenza della formulazione del nostro requisito, secondo cui siamo stati indirizzati ad analizzare T8 per trovare un File System che dovesse contenere i dati relativi alle partite.

4.3 Seconda iterazione

All’inizio di questa iterazione abbiamo tentato l’installazione del software, riscontrando numerosi problemi dovuti sia all’inadeguatezza dell’hardware delle nostre macchine sia a malfunzionamenti del software, in particolare non eravamo in grado di giocare delle partite in quanto non avveniva il caricamento delle classi e dei livelli dei robot. Risolta questa problematica grazie ai colleghi dei gruppi A1 e A8, abbiamo avuto un ulteriore intoppo in quanto non avveniva la generazione delle classi da testare; abbiamo risolto tale problematica effettuando

CAPITOLO 4. ORGANIZZAZIONE DEL LAVORO

un'installazione pulita da zero dell'applicazione. In Figura 4.1 sono riportate le attività che ci siamo prefissati di svolgere durante questa seconda iterazione.

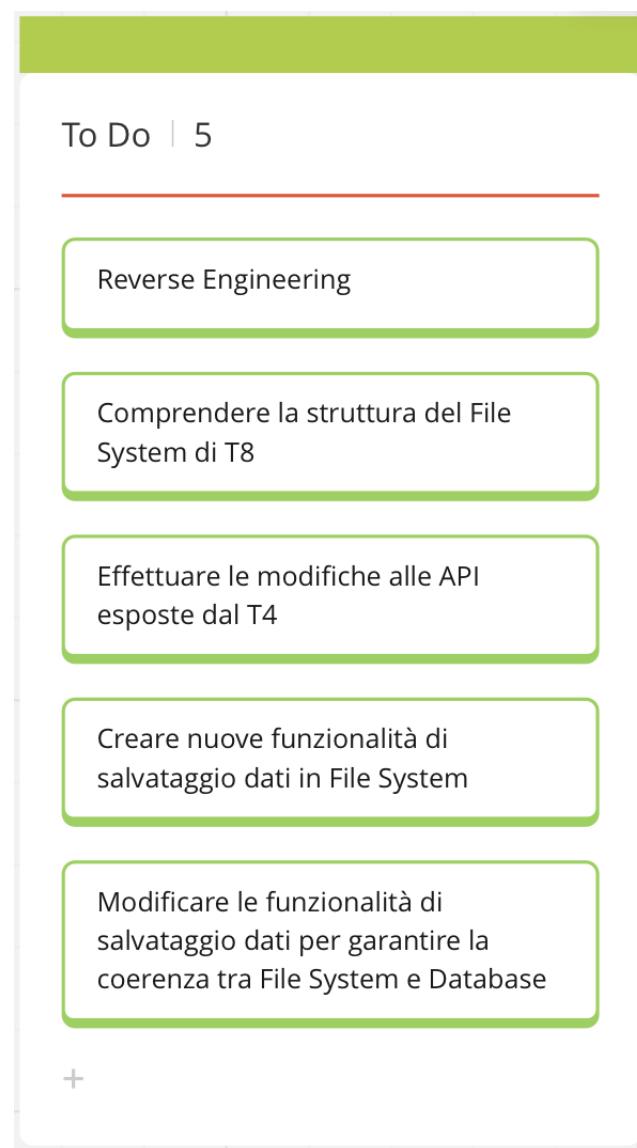


Figura 4.1: To-Do List per la seconda iterazione

4.4 Terza iterazione

Nelle successive due settimane ci siamo focalizzati sul comprendere dove apportare le modifiche necessarie, in quanto ci siamo resi conto che il File System di T8 serviva per il salvataggio dei soli dati riferiti al robot Evosuite e non per quelli delle partite. È stata necessaria una seconda analisi della struttura del progetto in generale, che ci ha fatto comprendere che le modifiche da effettuare dovevano esser realizzate a livello dello **Student Repository**, gestito dal task T5. Pertanto, abbiamo analizzato il componente, modificandone innanzitutto il Model e poi il Controller, in particolare abbiamo implementato il metodo **SaveGameCSV** che consente il salvataggio su File System dei dati relativi alle partite giocate, e abbiamo modificato la classe **GUIController** per garantire la coerenza tra dati salvati nei diversi componenti. La lista delle attività da svolgere durante la terza iterazione è riportata in Figura 4.2.

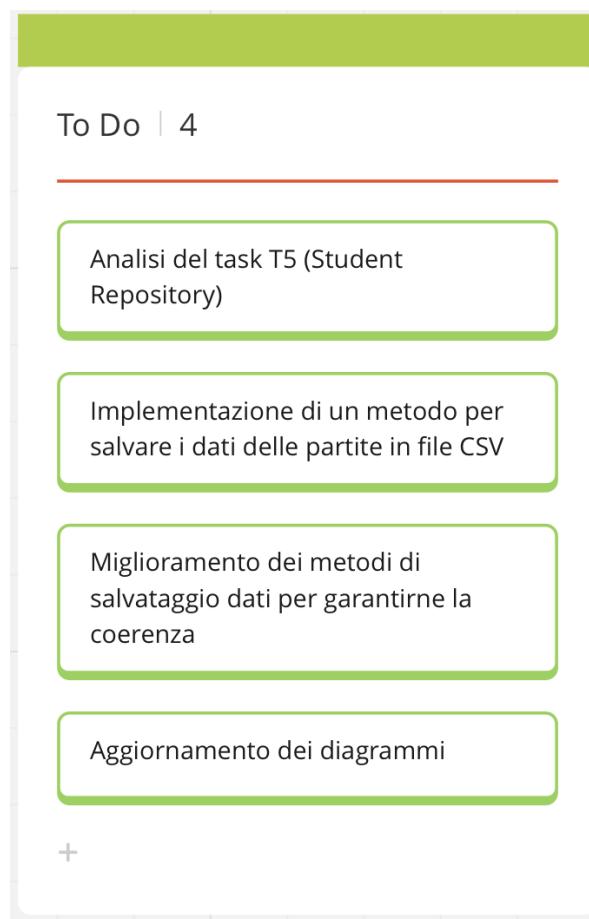


Figura 4.2: To-Do List per la terza iterazione

4.5 Quarta iterazione

Al termine della terza iterazione, nonostante i progressi importanti che erano stati realizzati dal punto di vista dell'implementazione del requisito, abbiamo dovuto far fronte a una documentazione incompleta e confusionaria. Abbiamo deciso dunque di prenderci altro tempo per migliorare e completare il lavoro svolto fino a quel momento, per quanto riguarda il codice ma soprattutto la documentazione. Durante questa quarta iterazione dunque siamo stati in grado di migliorare il salvataggio e l'avanzamento dei turni durante una partita, aumentan-

CAPITOLO 4. ORGANIZZAZIONE DEL LAVORO

do inoltre il numero di informazioni salvate in File System. Inoltre, abbiamo riorganizzato la documentazione per renderla più chiara, in quanto in precedenza non era ben comprensibile il punto di partenza del nostro lavoro e ciò che effettivamente avevamo realizzato, non erano presenti importanti diagrammi UML oppure non erano del tutto corretti. Infine, abbiamo modificato anche il progetto GitHub per renderlo coerente con il nostro lavoro, modificando il ReadMe e aggiornando la documentazione presente. In Figura 4.3 troviamo le attività che ci siamo prefissati di portare a termine durante quest'ultima iterazione.

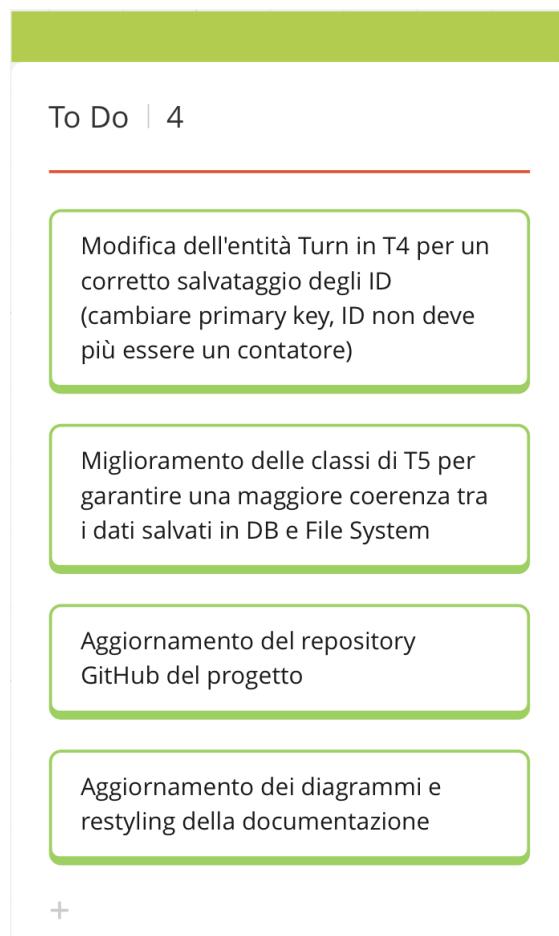


Figura 4.3: To-Do List per la quarta iterazione

Capitolo 5

Installazione

L'installazione di questo software rappresenta una novità per chi, come noi, non ha mai utilizzato prima Docker. Introduciamo dunque alcune conoscenze preliminari per poi guidare i lettori nell'installazione vera e propria dell'applicazione.

5.1 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.

5.2 Deployment dell'applicazione

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

5.2.1 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".

5.2.2 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 sospenderà autonomamente dopo l'avvio, dato che viene utilizzato solo per popolare il volume VolumeT9 condiviso con il Task T1. Dopo l'installazione, controllare dunque che tutti gli altri container eccetto quello del T9 siano funzionanti, in caso contrario riavviarli manualmente lasciando per ultimo "ui gateway", il quale dovrà essere avviato solo dopo l'esecuzione di tutti gli altri.

5.2.3 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:

```
use manvsclass
db.createCollection(ClassUT);
db.createCollection(interaction);
db.createCollection(Admin);
db.createCollection(Operation);
db.ClassUT.createIndex(difficulty: 1)
db.Interaction.createIndex(name: text, type: 1)
db.interaction.createIndex(name: text)
db.Admin.createIndex(username: 1)
```

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

5.2.4 Cosa fare se MongoDB non funziona

Durante i numerosi tentativi di installazione dell'applicazione abbiamo riscontrato un problema con MongoDB, in particolare la versione impostata di default Mongo 6.0.6 non era supportata dalle nostre macchine. Tale problematica abbiamo scoperto essere legata all'assenza dell'istruzione AVX nelle nostre CPU; siamo riusciti a risolverla effettuando un

```
    - volumet9:/volumet9
    - VolumeT8:/VolumeT8
networks:
    - global-network

mongo_db:
    image: "mongo:6.0.6"
    restart: always
    ports:
        - 27017:27017
    volumes:
```

Figura 5.1: Configurazione originale di MongoDB

```
    - volumet9:/volumet9
    - VolumeT8:/VolumeT8
networks:
    - global-network

mongo_db:
    image: "mongo:4.4.18"
    restart: always
    ports:
        - 27017:27017
volumes:
```

Figura 5.2: Configurazione modificata di MongoDB

downgrade della versione di MongoDB, portandola alla 4.4.18. Per effettuare tale operazione bisogna recarsi all'interno del progetto nella cartella "T1-G11/applicazione/manvsclass" e modificare il file di configurazione Docker chiamato "docker-compose.yml" cambiando la versione di Mongo. Prima delle modifiche Mongo risulta configurato come in Figura 5.1, mentre dopo la modifica dovrà essere configurato come in Figura 5.2. Dopo aver salvato il file, eliminare tutti i container, le immagini e i volumi da Docker ed effettuare un'installazione pulita dell'applicazione seguendo i passi sopra. La procedura resta invariata, tuttavia il comando da digitare nel terminale di Mongo al passo tre non sarà "mongosh" ma semplicemente "mongo".

5.3 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.

5.3.1 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.

5.3.2 Caricamento classi

Una volta che l'admin si è registrato e autenticato, si troverà sulla pagina in Figura 5.3; qui potrà visualizzare tutte le classi caricate. A questo punto ci sono due diverse opzioni, caricare la singola classe

CAPITOLO 5. INSTALLAZIONE

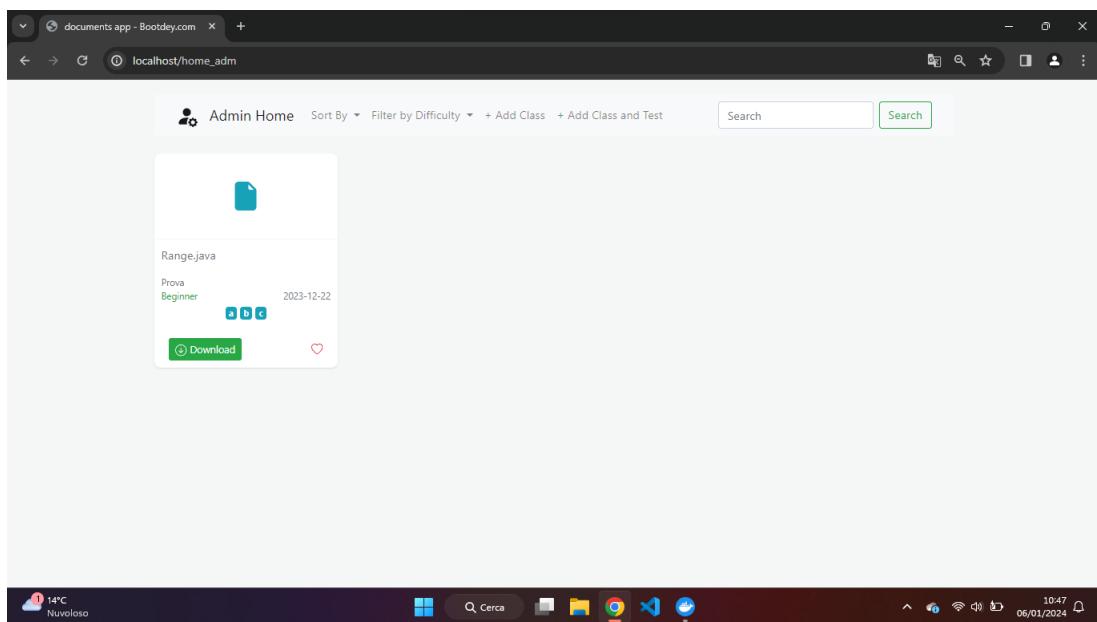


Figura 5.3: Pagina home degli admin

oppure classe e test già pronti, come vediamo nella barra di navigazione. Per velocizzare l'elaborazione delle classi caricate suggeriamo la seconda opzione, per cui clicchiamo su "Add Class and Test" e visualizzeremo la pagina in Figura 5.4, dove possiamo compilare i diversi campi relativi a nome, difficoltà, categorie... della classe ed effettuare il caricamento della classe Java e dei file zip contenenti i test Evosuite e Randoop di quella classe.

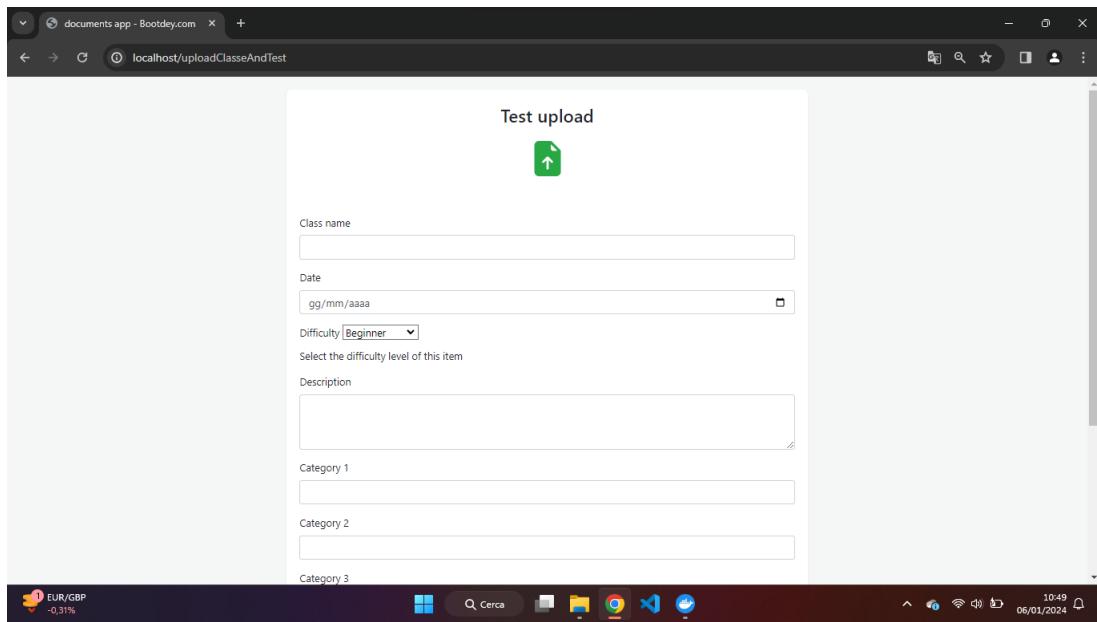


Figura 5.4: Pagina per effettuare upload di classi e test

5.3.3 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 5.5, per poi confermare la scelta e iniziare a giocare. L’obbiettivo è provare a battere il robot e raggiungere un livello di copertura del test sempre maggiore. Possiamo visualizzare la schermata di gioco in Figura 5.6.

CAPITOLO 5. INSTALLAZIONE

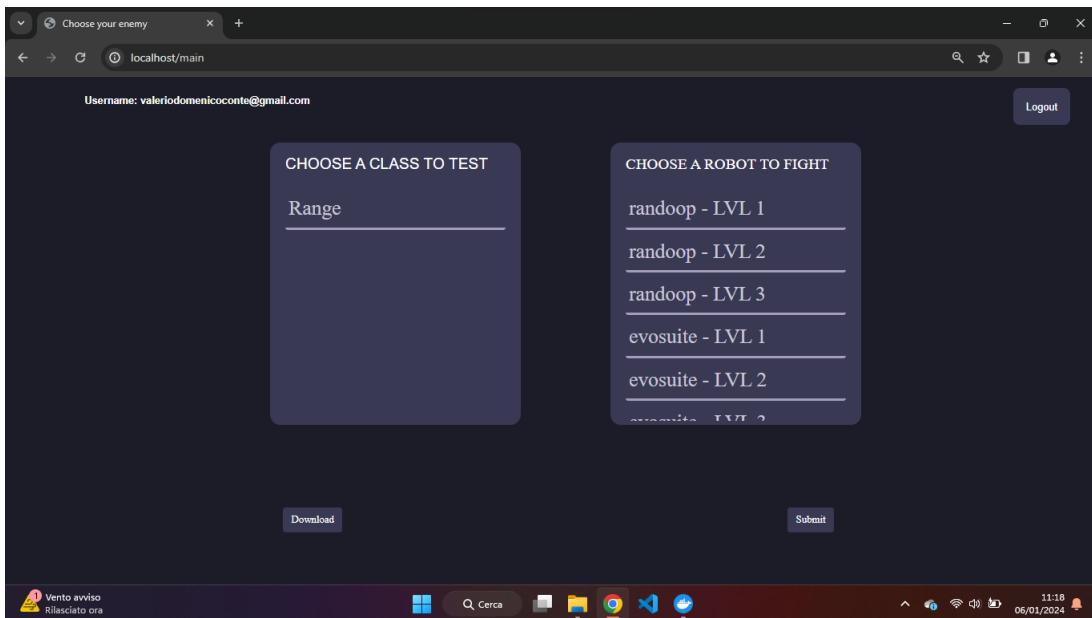


Figura 5.5: Pagina di scelta della classe e del robot da sfidare

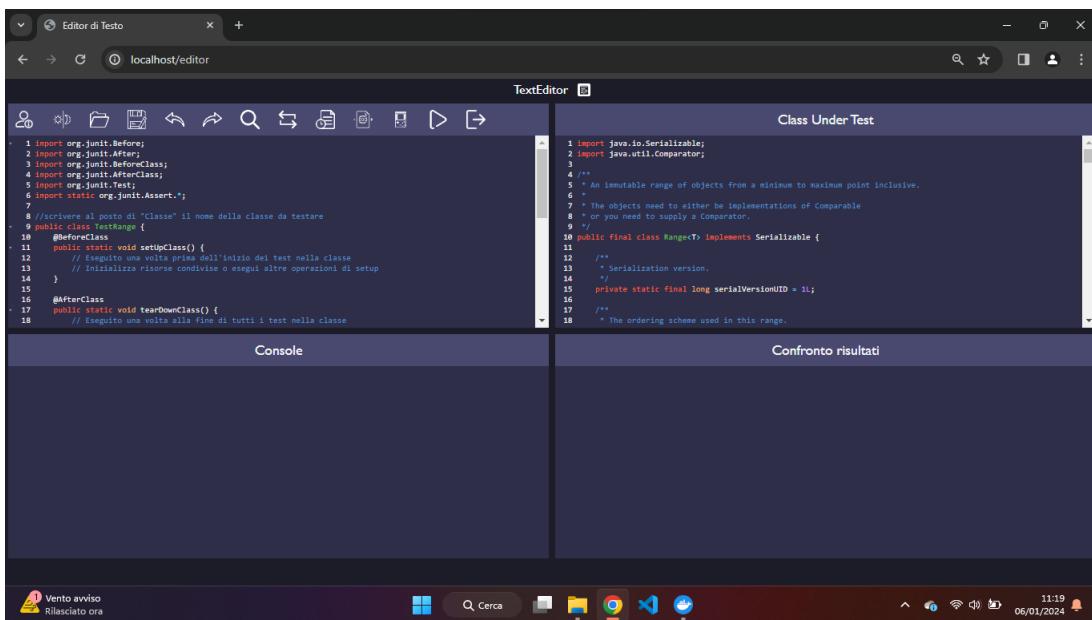


Figura 5.6: Pagina di gioco

Capitolo 6

Glossario

6.1 Game

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

6.2 Player

Il *player* (oppure *giocatore*) è l'entità che può giocare un partita, sfidando uno dei due possibili robot in ogni turno del round.

6.3 Robot

Il *robot* è l'entità affrontata dal giocatore durante un game; ci sono due possibili robot: Randoop ed Evosuite.

6.4 Round

Il *round* è l’entità che contiene le informazioni di gioco principali. Ogni game è 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.

6.5 Turn

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