



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

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

Elaborato finale in **Software Architecture Design**

Implementazione dei requisiti R1 ed R8

Anno Accademico 2023-24

Gruppo A12:

Francesco Pio Manna

matr. M63001485

Davide Landolfi

matr. M63001524

Claudio Pisa

matr. M63001512

Indice

Introduzione	1
1 Analisi preliminare del progetto esistente	2
1.1 Descrizione architettura	2
1.2 Workflow complessivo	3
1.3 Analisi dei requisiti	5
1.3.1 Use case diagram	8
1.3.2 Scenari	9
1.3.3 Activity diagram	11
2 Progettazione	12
2.1 Architettura a microservizi	12
2.2 Diagramma dei componenti	14
2.3 Composite structure diagram	16
3 Implementazione	18
3.1 Metodologia adottata	18
3.2 Collaborazione con altri gruppi	18
3.3 Calendario iterazioni	19
3.4 Attività svolte	20

3.5	Iterazione 1	22
3.6	Iterazione 2	23
3.6.1	S1 - studio della struttura iniziale dei volumi T8-T9	23
3.6.2	S2 - individuazione delle criticità relative alla struttura delle directories	25
3.6.3	S3 - scelta progettuale dell'unione o meno dei volumi	25
3.7	Iterazione 3	27
3.7.1	S4 - correzione criticità nel codice del progetto	27
3.7.2	Struttura game repository (T4)	27
3.7.3	Modifiche effettuate sul task T4 (R1)	29
3.7.4	Modifiche effettuate sul task T6 (R1)	33
3.7.5	Modifiche effettuate sul task T5 (R1)	34
3.8	Iterazione 4	43
3.8.1	Modifiche effettuate sul task T8 (R1)	43
3.8.2	Diagramma di attività di misurazione-utente .	46
3.8.3	Ipotesi condivisione volumeT8 con T5	47
3.8.4	Implementazione del requisito R8	49
3.8.5	Modifiche effettuate sul task T4 (R8)	49
3.8.6	Modifiche effettuate sul task T6 (R8)	52
3.8.7	Modifiche effettuate sul task T5 (R8)	54
4	Tools e frameworks utilizzati	57
4.1	DBeaver	57
4.2	Selenium	58

4.3	Postman	60
4.4	Swagger	61
5	Deployment	64
6	Testing	66
6.1	Casi di Test	67
6.1.1	Login	67
6.1.2	Editor	70
7	Sviluppi futuri	77
7.1	Rivedere attributi della tabella games del T4	77
7.2	Rivedere la struttura della api /save-data	78
7.3	Implementare uno scenario di gioco multigiocatore . .	79
7.4	Ridurre il carico dello storing dei dati sul browser . . .	79
7.5	Implementare un servizio separato per la scrittura sul filesystem	80

Introduzione

Il progetto European iNnovative AllianCe for TESTing, ENACTEST, vuole valorizzare l'importanza del testing in quanto lo sviluppo e poi il lancio in web delle Application spesso presenta delle falle non ancora individuate ma che si sarebbero dovute fixare durante la fase di Testing del suddetto progetto. Il Testing però risulta ancora una disciplina poco curata e valorizzata. L'Università di Napoli Federico II, insieme a piccole imprese, vuole coltivare l'uso della disciplina del Testing attraverso un Educational Game dove i Player sfidano un software di generazione automatica di test nella copertura dei casi di test. Questo dovrebbe incentivarne l'apprendimento rendendolo più accattivante.

Capitolo 1

Analisi preliminare del progetto esistente

In questa fase, si vogliono evidenziare e riportare i principali punti che caratterizzano l'architettura base sulla quale si è lavorato.

1.1 Descrizione architettura

L'architettura base è un'architettura a microservizi. Le architetture a microservizi sono un approccio di progettazione software in cui un'applicazione viene decomposta in componenti autonomi chiamati **microservizi**. Ogni microservizio è un'entità indipendente, gestendo specifiche funzionalità dell'applicazione e comunicando con gli altri attraverso interfacce ben definite. Questa suddivisione favorisce la scalabilità e la manutenibilità del sistema. L'utilizzo delle architet-

ture a microservizi offre diversi vantaggi. Innanzitutto, favoriscono la flessibilità, consentendo lo sviluppo, il deployment e la scalabilità indipendenti dei singoli servizi. Ciò facilita anche la gestione e l'aggiornamento continuo, permettendo lo sviluppo parallelo di diverse funzionalità. Inoltre, migliorano la resistenza del sistema alle interruzioni, in quanto il fallimento di un microservizio non influisce sugli altri.

La separazione dei microservizi agevola la diversificazione tecnologica, permettendo l'utilizzo di tecnologie e linguaggi specifici per ciascun servizio. Questo favorisce l'adozione di nuove tecnologie senza influire sull'intero sistema. Inoltre, le architetture a microservizi sono adatte per ambienti cloud, poiché consentono di sfruttare le risorse in modo efficiente attraverso il deploy su container.

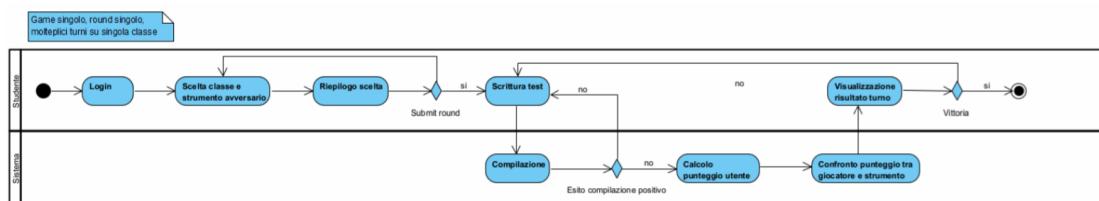
La gestione dei microservizi richiede un'attenzione particolare alla comunicazione e alla sicurezza, ma la flessibilità e la scalabilità che offrono le rendono una scelta preziosa per applicazioni complesse e in continua evoluzione.

1.2 Workflow complessivo

Per descrivere il flusso di funzionamento complessivo dell'applicazione si è deciso di riportare un workflow diagram di alto livello.

Il workflow inizia con il login da parte del giocatore, poi quest'ultimo effettua la scelta della classe da testare e il tool avversario contro

CAPITOLO 1. ANALISI PRELIMINARE DEL PROGETTO ESISTENTE



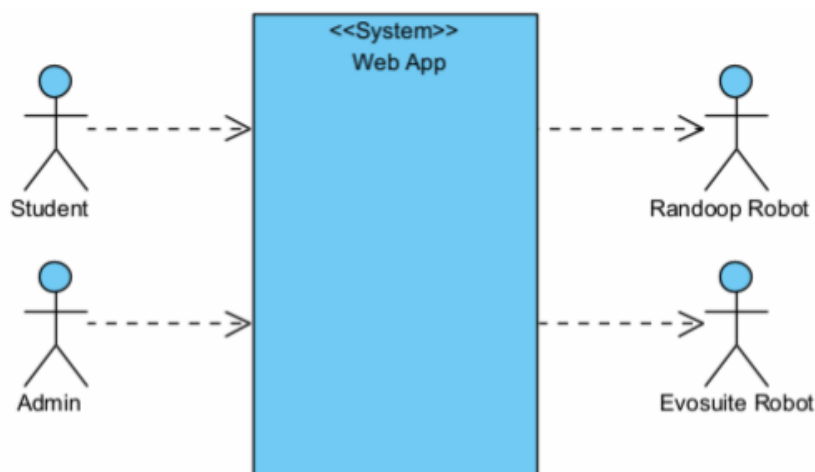
cui gareggiare e, dopo aver confermato la scelta, scrive effettivamente il test case nella schermata dell'editor.

Viene successivamente effettuata la compilazione del test, poi un controllo dell'esito della stessa e calcolato il punteggio utente.

Il punteggio finale viene calcolato sia da Jacoco che da EvoSuite, a seguito dell'integrazione effettuata dal gruppo G41.

Successivamente il sistema fornisce il confronto dei punteggi misurati sia per l'utente che per il robot in quel turno, e poi li mostra a video. In seguito al risultato, in caso di vittoria la partita è conclusa, e al giocatore viene assegnato un punteggio, mentre in caso di sconfitta il giocatore può continuare a giocare fin quando non vince. Dunque l'applicazione non contempla, in questo scenario, il caso di sconfitta in una partita, ma è modellata la sconfitta solo in un determinato turno.

Di seguito inoltre si riporta un diagramma di contesto di alto livello, per evidenziare gli attori coinvolti durante l'interazione col software.



1.3 Analisi dei requisiti

In questa fase di analisi dei requisiti, si riporta esclusivamente l'analisi effettuata per i nuovi requisiti da noi aggiunti. Per una visione complessiva sui requisiti dell'intera applicazione, si rimanda alle documentazioni dei gruppi che hanno sviluppato i singoli task. Di seguito sono riportati i requisiti da aggiungere all'attuale versione dell'applicazione:

- **R1:** rivedere l'organizzazione del File System condiviso che al momento è sparso fra il Volume T8 e T9. Sarebbe preferibile avere un solo volume condiviso che rispetti la struttura riportata nella figura in basso.

Per quanto riguarda i test dello studente, la soluzione indicata permette di salvare sotto la directory *GameId-TurnoXX* i dati del turno XX, comprensivi dei risultati raggiunti dai test di quel turno (*GameData.csv* e *TestReport*) e il codice dei Test di quel turno.

Qualora si valuti che la separazione dei volumi T8 e T9 sia ne-

cessaria per consentire ai due robot di lavorare in maniera indipendente sul File system, valutare l'opzione di non fondere, o di fondere solo in un secondo momento successivo alla generazione, le due strutture dei volumi T8 e T9.

- **R8:** Al termine di una partita, prevedere una funzionalità/servizio per attribuire un punteggio al giocatore.

Si potrà partire da un semplice calcolo del punteggio che attribuisce un valore pari a 1 per ogni partita vinta, indipendentemente dalle metriche di coverage ottenute, -1 per ogni partita persa, 0 in caso di parità.

Come modalità di calcolo aggiuntiva, si potrebbe prevedere un punteggio in caso di vittoria incrementato in base alla percentuale di LOC coverage raggiunta. Ad esempio:

- **Punteggio vittoria** = $1 + \text{LOC\%}$ (dove LOC% potrà variare fra 0 e 1);
- **Punteggio sconfitta** = -1;
- **Punteggio pareggio** = 0.

Occorre scorporare tale funzione di calcolo del punteggio in un servizio a parte, che possa essere anche configurabile ed in futuro estensibile. Tale punteggio partita dovrà essere salvato nel database delle partite (vedi T4) (occorre memorizzare che il giocatore ha ottenuto un certo punteggio al termine di quella partita), sia

attribuito al punteggio globale del giocatore al fine di costruire una classifica dei vari giocatori.

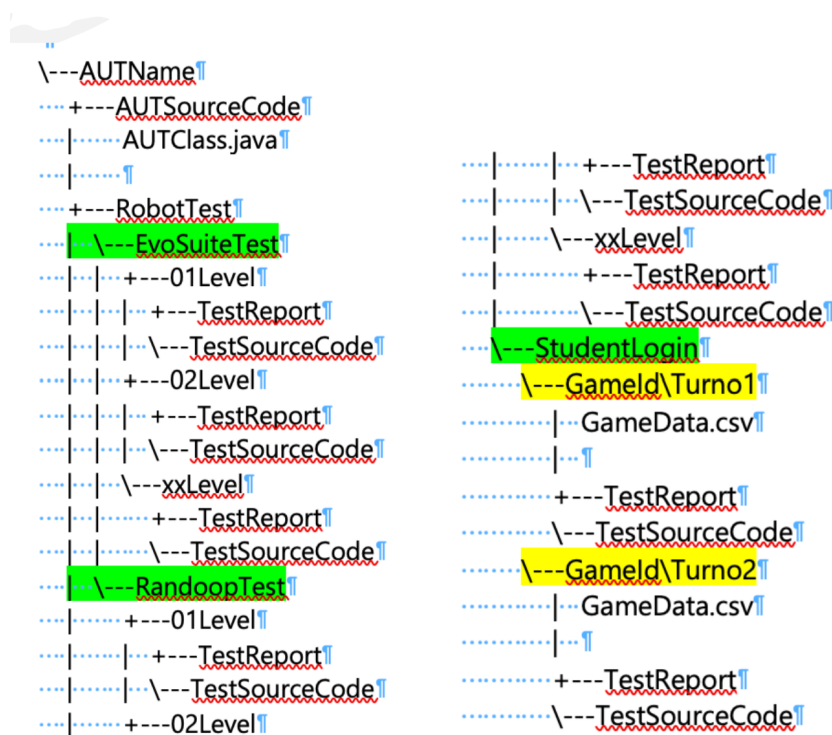


Figura 1.1: Struttura prevista

Priorità dei requisiti da aggiungere

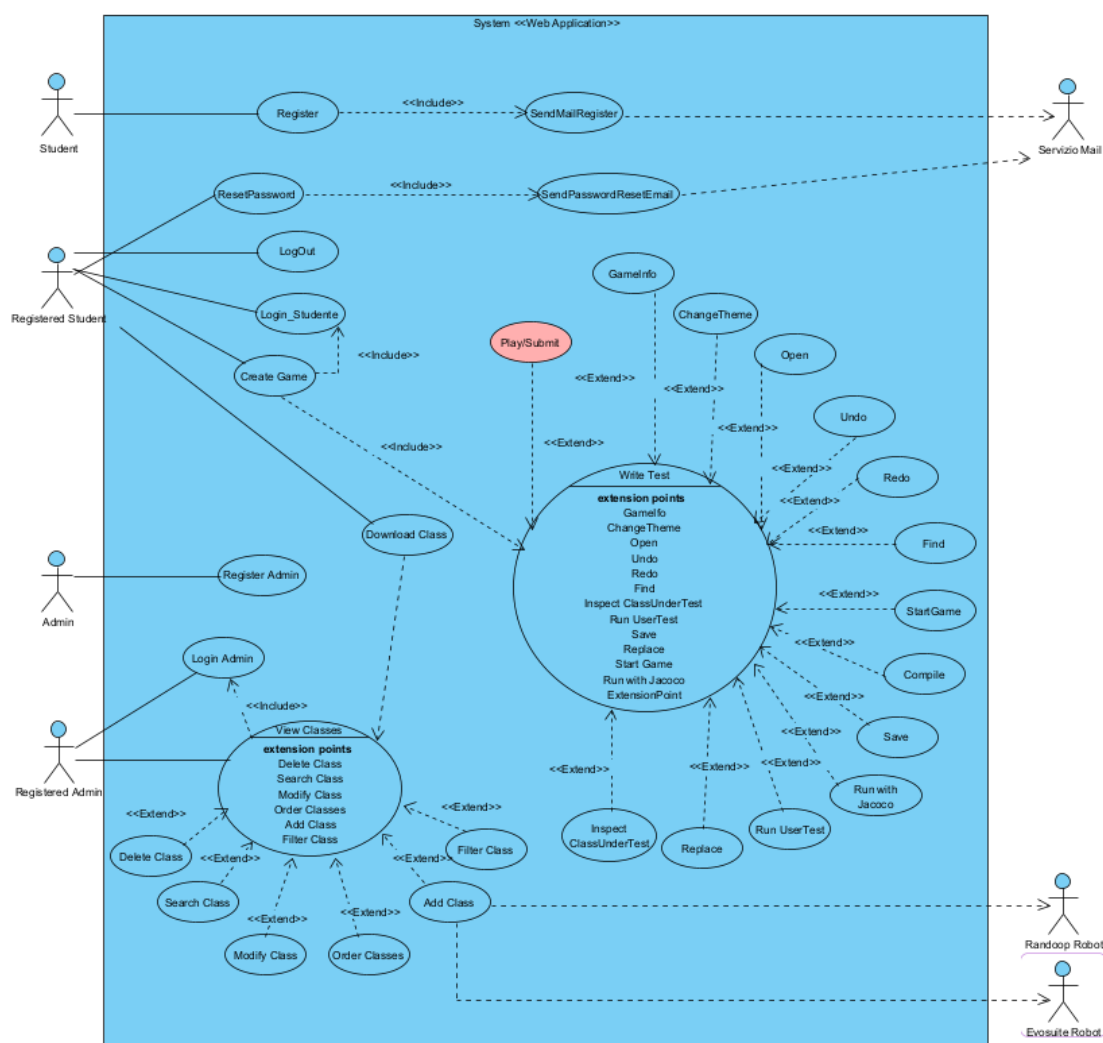
Il requisito R1 è un requisito di tipo manutentivo ad alta priorità, in quanto dovrà essere utilizzata da numerosi altri gruppi, considerando che andrà ad impattare sul db del T4 e sul T5, che è il task centrale all'interno dell'applicazione. Pertanto, verrà implementato per primo, demandando l'implementazione del requisito R8 alle successive iterazioni.

Il requisito R8 invece prevede l'aggiunta di una nuova funzionalità, le-

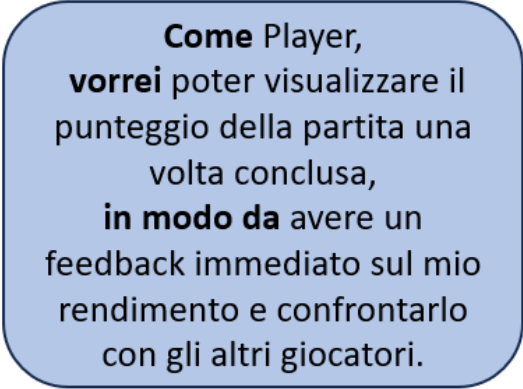
gata all'assegnazione di un punteggio al giocatore alla fine della partita, pertanto, risulta essere meno critico del precedente.

1.3.1 Use case diagram

Di seguito è riportato il diagramma complessivo dei requisiti dell'intera applicazione, con un focus sul requisito da noi modificato, riguardante il messaggio finale alla conclusione della partita con un punteggio assegnato all'utente.



Di seguito è riportata la descrizione del requisito utilizzando il paradigma delle storie SCRUM "AS A [utente], I WANT [desiderio], SO THAT [beneficio]".



Come Player,
vorrei poter visualizzare il
punteggio della partita una
volta conclusa,
in modo da avere un
feedback immediato sul mio
rendimento e confrontarlo
con gli altri giocatori.

E' importante notare come il requisito R1, da un punto di vista utente, sia, al momento, trasparente. Infatti, tale requisito potrà essere utile in futuro per l'implementazione di una classifica/storico globale fra tutti i giocatori.

1.3.2 Scenari

In questa sezione si evidenziano le modifiche che subiscono i casi d'uso in seguito all'implementazione dei task richiesti. In particolare, al metodo Submit è stata aggiunta la richiesta espressa dal requisito R8 di mostrare il punteggio a fine partita.

CAPITOLO 1. ANALISI PRELIMINARE DEL PROGETTO ESISTENTE

CASO D'USO	SUBMIT
ATTORE PRIMARIO	Registered Student
DESCRIZIONE	Si eseguono i test e vengono confrontati i risultati con il robot scelto.
PRE-CONDIZIONI	Classe di test correttamente compilata.
SEQUENZA DI EVENTI PRINCIPALE	Il giocatore preme sul bottone "Submit"
POST-CONDIZIONE	Visualizzazione dei risultati ottenuti da parte dello studente all'interno della console ed il punteggio partita.

1.3.3 Activity diagram

Di seguito l'activity diagram del caso d'uso Submit aggiornato.

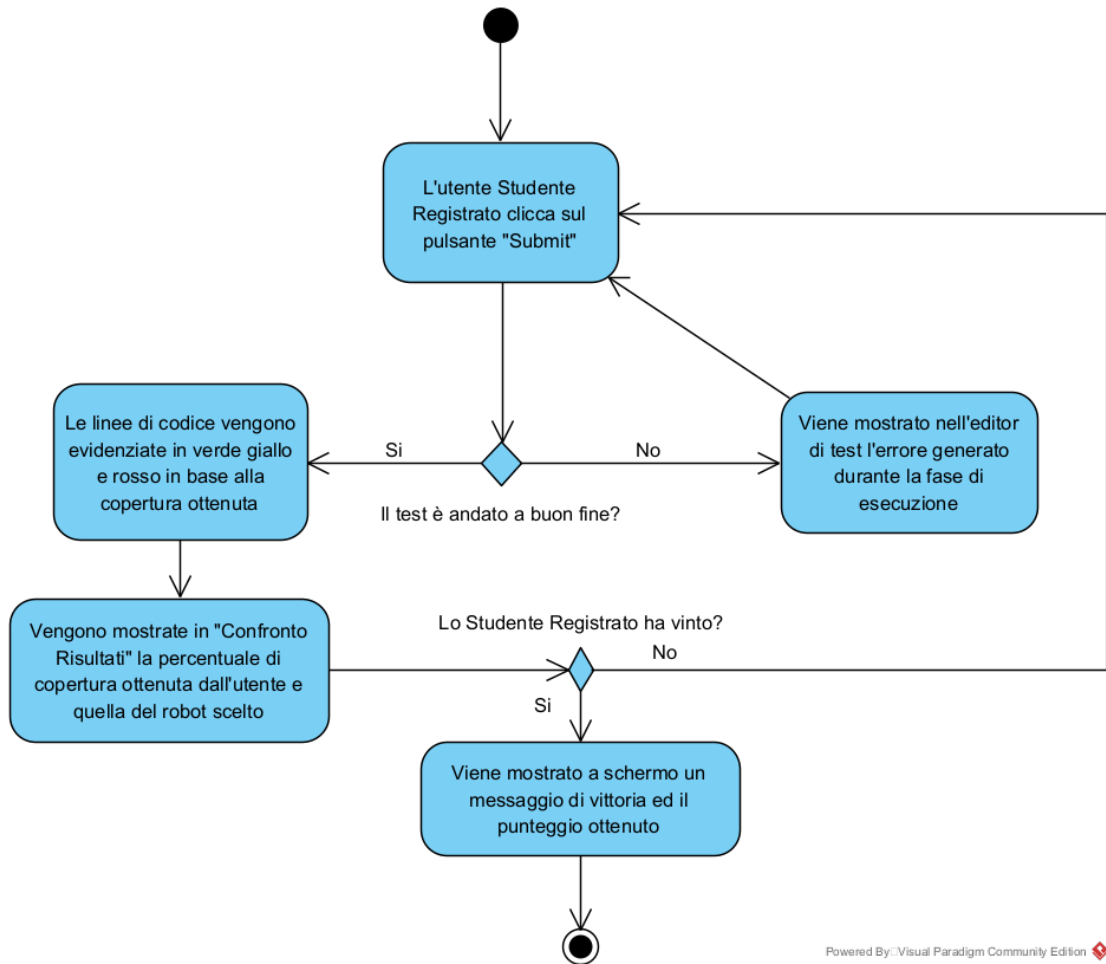


Figura 1.2: Activity diagram di Submit

Capitolo 2

Progettazione

La fase di progettazione dell'applicazione complessiva ha previsto, come specificato nel capitolo precedente, l'impiego di un'architettura a microservizi.

2.1 Architettura a microservizi

In seguito alle modifiche effettuate al software, l'architettura complessiva resta la stessa. In questa sezione si vogliono riportare i punti salienti della stessa per rendere più semplice e comprensibili gli interventi effettuati nella nuova versione del software sviluppata.

Il client può accedere ai servizi forniti dall'applicazione tramite l'interfaccia data dai due gateway mediante la connessione Internet. In particolare abbiamo:

- UI Gateway la quale ha il compito di effettuare il routing delle

richieste http relative al frontend (UI).

- API Gateway la quale si occupa del routing e della gestione dell'autenticazione e autorizzazione per le richieste riguardo le API del sistema.

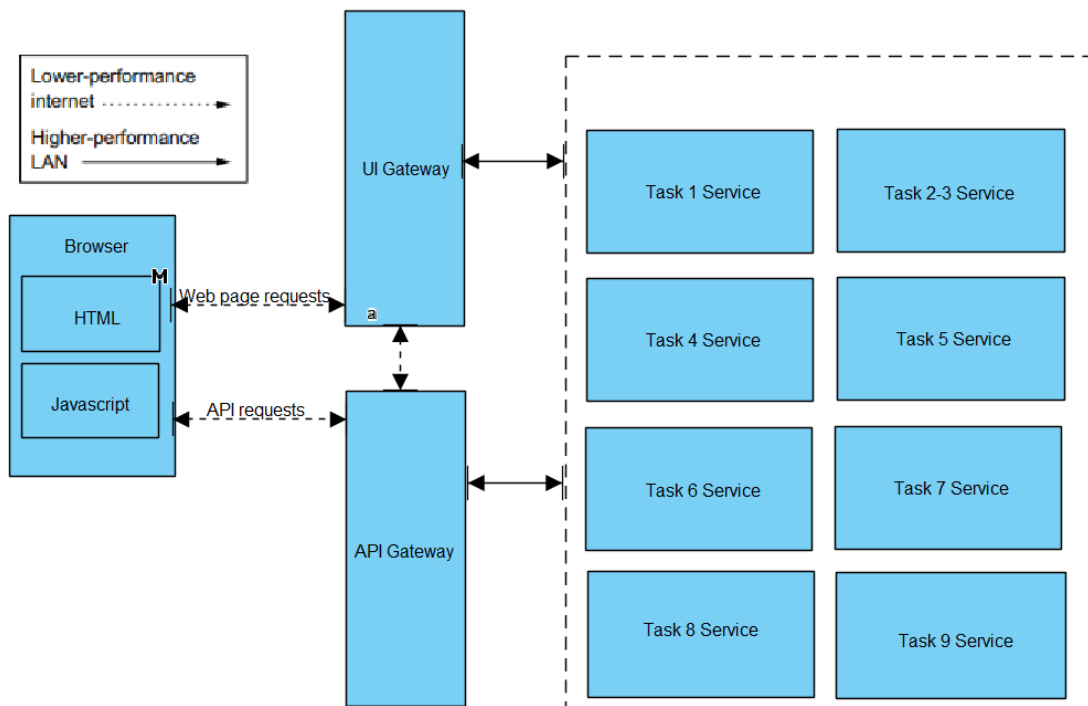


Figura 2.1: Architettura a microservizi

2.2 Diagramma dei componenti

Si riporta il diagramma dei componenti dell'architettura attuale. I componenti da noi modificati sono stati evidenziati con il colore rosa.

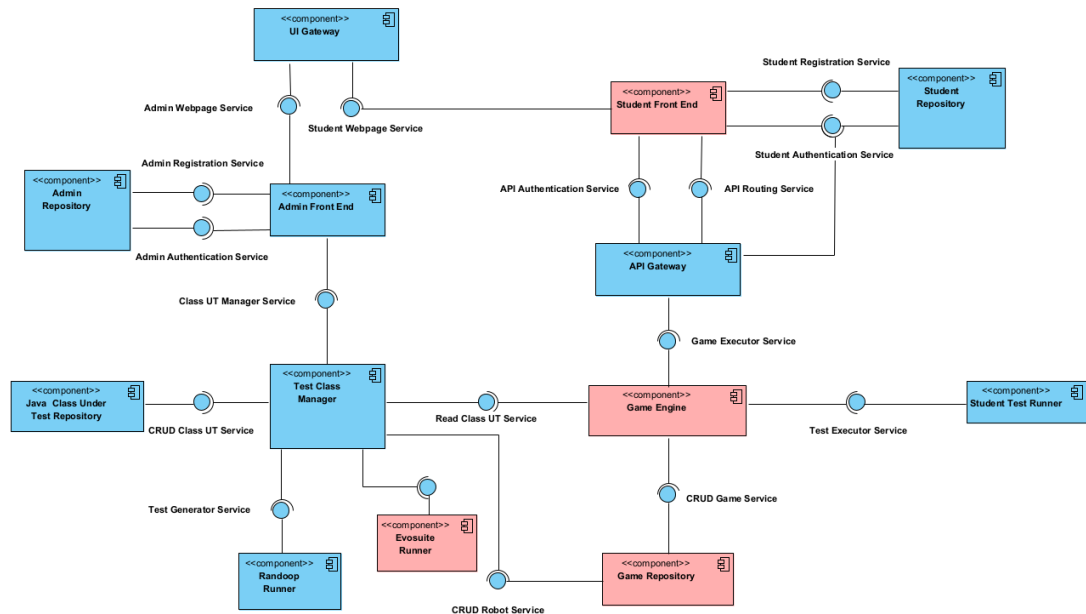


Figura 2.2: Diagramma dei componenti

In seguito, è riportata una breve descrizione per ognuno dei componenti.

Test Class Manager

Il Test Class Manager ha il compito di gestire le classi giocabili, inserite dall'admin. Utilizza i servizi di copertura del codice offerti sia da Randoop e EvoSuite. Ad ogni classe inserita sono generati tutti i livelli possibili dei robot, prima attraverso Randoop e poi attraverso EvoSuite.

Student Front End

Fornisce un'interfaccia all'utente "*Registered Student*", a partire dalla registrazione fino alle schermate di gioco che comprendono la scelta del robot contro il quale giocare e l'interfaccia dell'editor.

EvoSuite Runner

Esso genera i robot EvoSuite e restituisce le statistiche di copertura del codice, presenti nello *statistics.csv*. Comunica con il componente Test Class Manager per generare la copertura tramite EvoSuite, all'atto del caricamento della classe da parte dell'admin. Esso viene usato dallo Student Front End per calcolare la copertura del codice del test, scritto dallo studente.

Game Repository

Ogni Submit viene salvato in un repository interno al T4, dedicato al salvataggio delle partite. Ciò permette di avere uno storico dei risultati delle coperture dei test, ottenute nei diversi turni.

Game Engine

Si occupa della gestione della partita. Usa i servizi offerti dallo Student Test Runner e dal Game Repository per un corretto svolgimento della partita: ottiene i risultati della compilazione e della coverage del test dello studente, i risultati dei robot e salva le informazioni della partita

in corso. Ogni Game offre la possibilità di molteplici turni fin quando non viene superata la percentuale di copertura del test scritto dal robot.

Randoop Runner

Genera test a partire da una classe con il robot Randoop e li salva localmente, per poi calcolarne la copertura con il tool Emma.

2.3 Composite structure diagram

Per mantenere uniformità con il diagramma dei componenti, nel Composite Structure Diagram è stata adottata una notazione che prevede l'utilizzo di un package per ciascuna componente, evidenziando chiaramente quali task sono responsabili per la loro realizzazione e contribuiscono al corretto funzionamento del sistema. Mentre alcuni componenti sono completamente realizzati da singoli task, ce ne sono altri come lo Student Front End e il Game Engine che sono distribuiti su due task. Lo Student Front End è realizzato dal task 2-3 per quanto riguarda la parte dell'accesso al gioco, dunque le schermate di login, registrazione, recupero password. Per la parte che riguarda il gioco vero e proprio, il front end è stato realizzato nel task 5. Il Game Engine, invece, è realizzato nella parte di salvataggio iniziale della partita, dal task 5, e per il resto delle funzionalità dal task 6 (run, compile etc.).

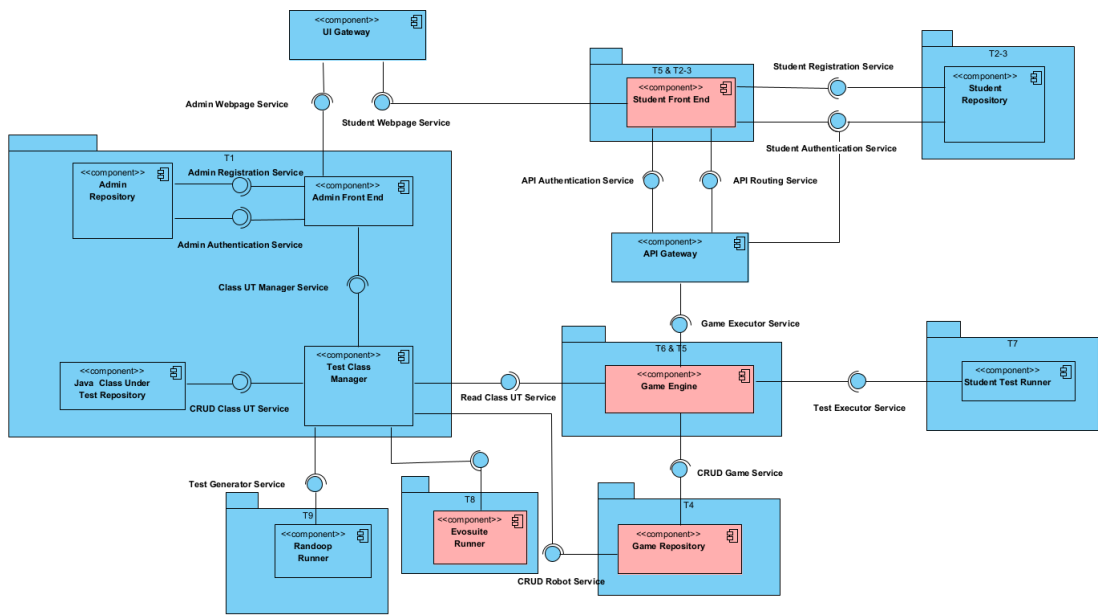


Figura 2.3: Composite Structure Diagram

In rosa sono evidenziati i componenti modificati.

Capitolo 3

Implementazione

3.1 Metodologia adottata

Per la realizzazione del progetto sono stati seguiti i metodi e i principi della metodologia SCRUM. SCRUM è un framework Agile che si concentra sulla gestione dei progetti e lo sviluppo del software. E' basato su un set di principi e valori definiti nel "Manifesto Agile" e offre una struttura per aiutare i team a lavorare in modo collaborativo.

3.2 Collaborazione con altri gruppi

Durante lo sviluppo del progetto, è stato necessario comunicare con diversi altri gruppi per discutere le scelte progettuali, fornire chiarimenti sulle implementazioni effettuate e richiedere spiegazione dai gruppi a cui erano assegnati i singoli task.

In particolare, abbiamo collaborato con i seguenti gruppi:

- **T4-G18**: per richiedere informazioni sulla configurazione e sull'utilizzo del database;
- **T11-G41**: per richiedere informazioni sulle metodologie utilizzate per l'integrazione del task T8;
- **R1_8-A7**: per discutere delle scelte progettuali iniziali legate alla fusione/separazione dei volumi Docker;
- **R9-A9**: per fornire informazioni sulla nuova struttura della tabella games da noi implementata;
- **R10-A11**: per fornire chiarimenti sul nuovo flusso di aggiornamento degli id delle tabelle games, rounds e turns da noi implementata.

3.3 Calendario iterazioni

Di seguito sono riportati i periodi di svolgimento delle varie iterazioni:

- **Iterazione 1**: 31/10/23 – 20/11/23;
- **Iterazione 2**: 21/11/23 - 05/12/23;
- **Iterazione 3**: 09/12/23 - 19/12/23;
- **Iterazione 4**: 27/12/23 - 15/01/24.

3.4 Attività svolte

Di seguito è riportato un sommario delle attività svolte in ognuna delle quattro iterazioni.

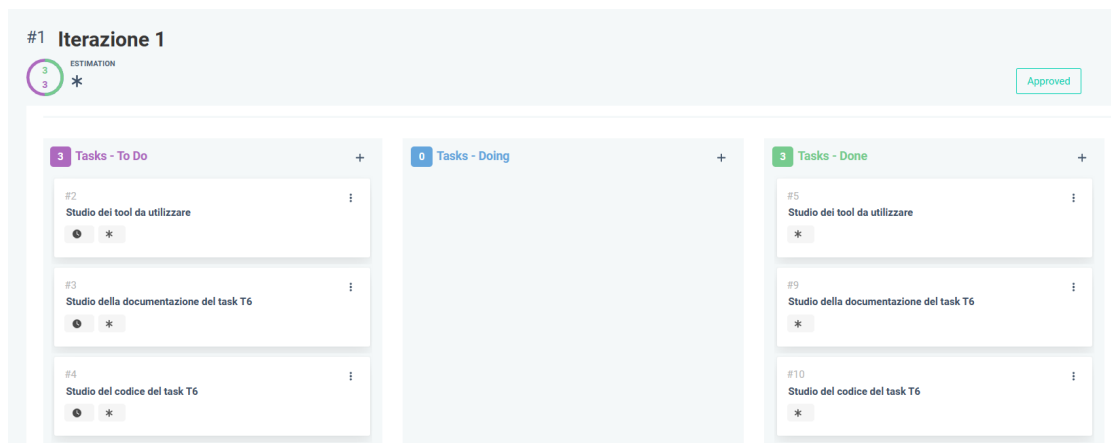


Figura 3.1: Iterazione 1

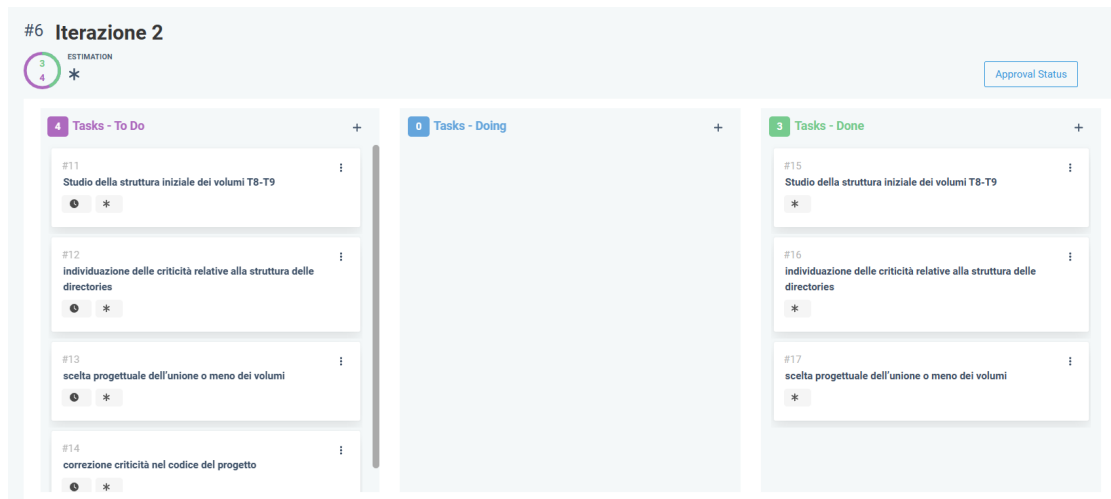


Figura 3.2: Iterazione 2

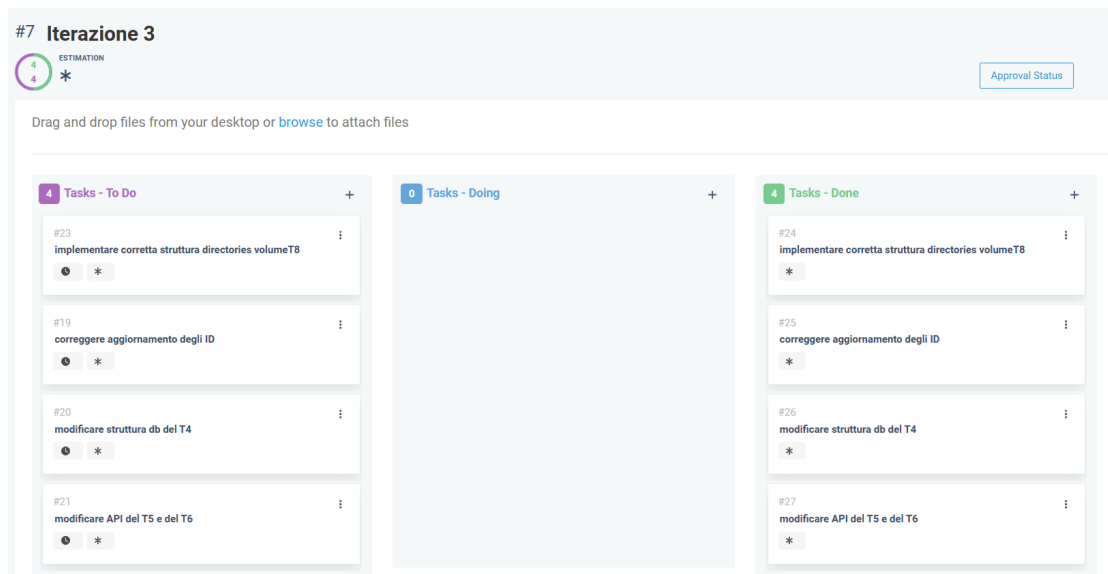


Figura 3.3: Iterazione 3

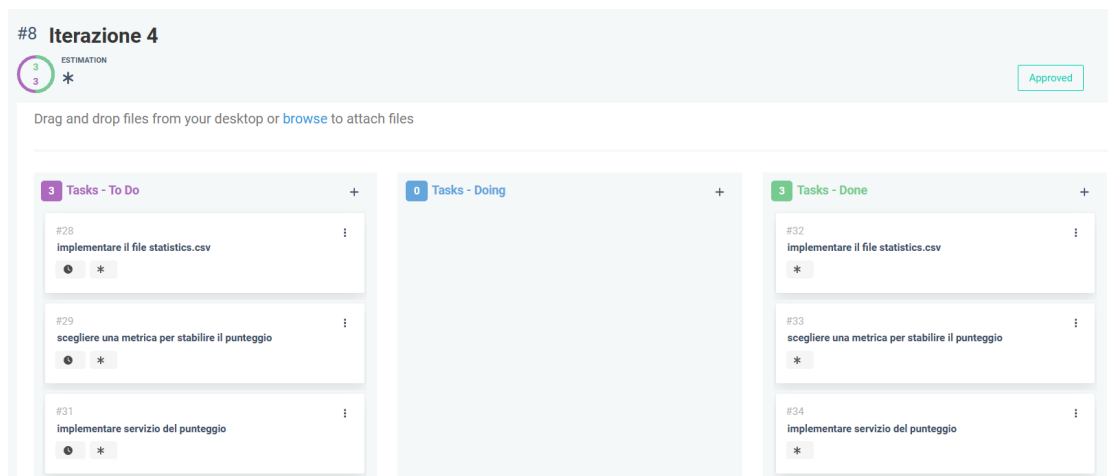


Figura 3.4: Iterazione 4

3.5 Iterazione 1

Durante la prima iterazione si sono analizzati in modo approfondito i task 5 e 6. In particolare, i task in questione hanno lo scopo principale di gestire il front-end dello studente durante la partita e le informazioni della stessa lato back-end. Si è notata una criticità in questi due task, in quanto nell'implementazione finale sono stati, di fatto, logicamente accorpati in un unico task. Questa soluzione si può giustificare con due considerazioni:

- I due task sono uniformi nelle tecnologie utilizzate (entrambi usano Spring MVC e Java). Inoltre, essi insieme formano effettivamente il front-end dello studente, dunque sono anche semanticamente e funzionalmente vicini;
- Unire i due task in uno ha reso più facile il passaggio delle informazioni della partita selezionate dallo studente nella UI del task 5 verso l'editor del task 6.

Questa modifica ha portato ad una incoerenza nei file che compongono il progetto. In particolare, esistono due copie del file *editor.html*, il quale gestisce sostanzialmente tutto il front-end dell'editor, di cui una di questa non è più aggiornata (in particolare quella del task 6). Inoltre, l'unico utilizzo del task 6 è lato back-end, tramite la api */run* (che si attiva alla pressione del tasto *play/submit* dell'editor), la quale ha lo scopo di salvare e concludere la partita e decretare il vincitore.

3.6 Iterazione 2

Durante la seconda iterazione si è proceduto con un'analisi del requisito R1. Si è suddivisa l'implementazione del requisito R1 in quattro sottotask:

- **S1:** studio della struttura iniziale dei volumi T8-T9;
- **S2:** individuazione delle criticità relative alla struttura del filesystem;
- **S3:** scelta progettuale dell'unione o meno dei volumi;
- **S4:** correzione criticità nel codice del progetto.

3.6.1 S1 - studio della struttura iniziale dei volumi T8-T9

Si riporta la struttura iniziale del filesystem condiviso.

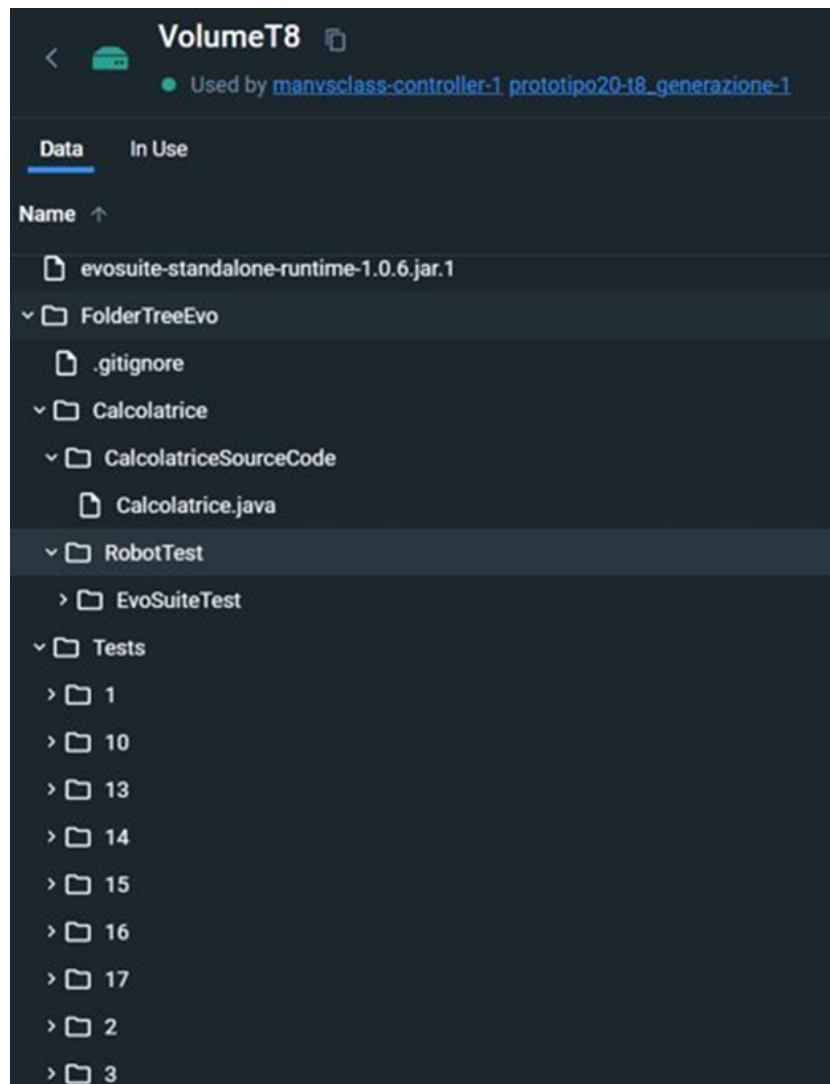


Figura 3.5: Struttura iniziale VolumeT8

Il file system è suddiviso su due volumi, volumeT8, relativo alle informazioni sul robot EvoSuite, e volumeT9, relativo alle informazioni sul robot Randoop. Inoltre, il volumeT8 contiene anche le informazioni sul test case scritto dallo studente.

Il volumeT8 è visibile dal T1 e dal T8, mentre volumeT9 è visibile dal T1 e dal T9.

3.6.2 S2 - individuazione delle criticità relative alla struttura delle directories

La struttura iniziale dunque non rispetta quella stabilita nei requisiti. Inoltre, c'è anche un problema riguardo gli id delle partite e del turno. Infatti, per ogni partita, caratterizzata da un certo gameId, il turno viene incrementato insieme al gameId, il che è sbagliato in quanto il turno dovrebbe tornare al valore 1 all'inizio di ogni partita. In sostanza, gameId, roundID e turnID, in questa fase del progetto, coincidono.

3.6.3 S3 - scelta progettuale dell'unione o meno dei volumi

Per l'implementazione del requisito si sono sondate varie ipotesi:

- Unire i due volumi in un unico volume;
- Tenere i volumi separati;
- Separare in 3 volumi diversi.

Tenere i *volumi separati* porterebbe i seguenti vantaggi:

- **Prestazioni del disco:** Unificare i volumi potrebbe ridurre l'overhead di lettura/scrittura su più volumi separati. Tuttavia, se i volumi sono su dispositivi di archiviazione diversi o hanno requi-

siti di I/O significativamente diversi, potrebbe essere vantaggioso mantenerli separati;

- **Facilità di backup e ripristino:** Avere volumi separati semplifica la gestione dei backup e dei ripristini. Puoi eseguire backup specifici per ciascun volume e ripristinarli in modo indipendente, semplificando la manutenzione e il ripristino in caso di problemi;
- **Scalabilità e isolamento:** Se i due servizi (Evosuite e Randoop) possono essere scalati in modo indipendente e hanno requisiti diversi, potrebbe avere senso mantenerli separati per garantire un migliore isolamento;
- **Facilità di manutenzione e gestione delle risorse:** Unificare i volumi potrebbe semplificare la gestione delle risorse, ma se ci sono diverse persone o team responsabili per Evosuite e Randoop, la separazione potrebbe facilitare la manutenzione indipendente;
- **Requisiti di sicurezza:** Se ci sono requisiti di sicurezza che richiedono un isolamento più stretto tra i dati generati da Evosuite e Randoop, mantenere volumi separati potrebbe essere preferibile.

Lo svantaggio di tenere i volumi separati è legato al requisito R10, il quale permette di estendere la tipologia di scenari di gioco attuale, abilitando l'utente a giocare contemporaneamente con più robot. Per

tale requisito dunque sarebbe comodo avere un unico filesystem con tutti i test case in gioco (quello scritto dallo studente, quello generato da EvoSuite e quello generato da Randoop).

Tuttavia, pesando i vantaggi e gli svantaggi, si è deciso di tenere i due volumi separati. E' importante notare che i test scritti dallo studente vengono mantenuti nel volumeT8 perché EvoSuite fornisce allo studente delle metriche aggiuntive per i test da lui scritti.

Il sottotask S4 è stato rimandato alla terza iterazione.

3.7 Iterazione 3

3.7.1 S4 - correzione criticità nel codice del progetto

Per implementare il requisito, si è dovuta prima risolvere la grande criticità relativa agli aggiornamenti sbagliati di gameID, roundID e turnID. Prima di tutto, si è analizzato approfonditamente il database fornito dal task 4.

3.7.2 Struttura game repository (T4)

Prima di procedere con le modifiche effettuate sul db del T4 si illustrano diversi diagrammi per descrivere le entità e i package di cui esso è composto, con un focus sugli elementi modificati in questa versione dell'architettura.

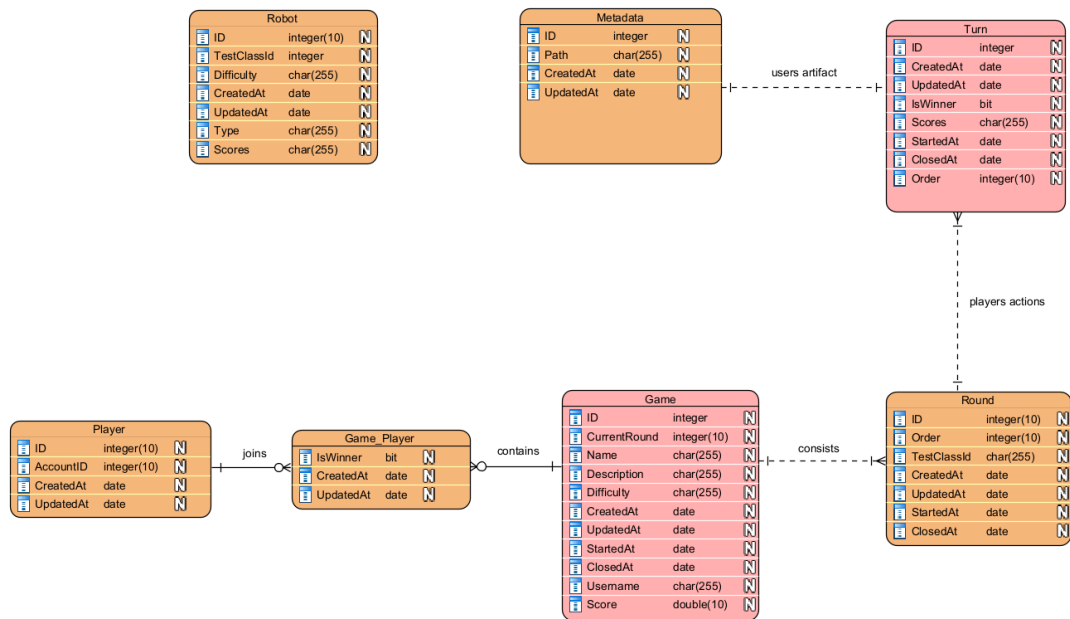


Figura 3.6: Diagramma ER

In rosa sono evidenziate le tabelle che sono state arricchite con nuovi attributi.

Di seguito è mostrato un system domain model, in cui sono evidenziati i package e i file modificati.


```
type Turn struct {
    ID          int64      `gorm:"primaryKey;autoIncrement"`
    Order       int       `gorm:"not null;default:1"`
    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;constraint:OnDelete:SET NULL;"`
    Scores      string     `gorm:"default:null"`
    IsWinner    bool      `gorm:"default:false"`
    PlayerID    int64      `gorm:"index:idx_playerturn;not null"`
    RoundID     int64      `gorm:"index:idx_playerturn;not null"`
}
```

Figura 3.8: Rimozione del vincolo *UNIQUE* dall'attributo `roundId` volumeT8.

Per aggiornare correttamente il campo *order* appena aggiunto è stato necessario modificare anche la api `/turns`, utilizzata per aggiornare il contenuto della tabella *turns* del database del T4.

Per fare ciò è stata:

- Modificata la struct *Turn* per contenere l'attributo *order* in *turn.go*. Tale struct è progettata per essere facilmente convertita da e verso il formato JSON.
- Modificata la struct *CreateRequest* per contenere l'attributo *order* all'interno del file *turn.go*. Tale struct viene utilizzata come parte di una richiesta API per creare o aggiornare un turno.
- Modificata la funzione *fromModel* per contenere l'attributo *order* in *turn.go*. Tale funzione converte un oggetto di tipo *model.Turn* (rappresentante il modello del database) in un oggetto di tipo *Turn* (progettato per l'interazione con l'API).

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

Figura 3.9: struct CreateRequest

- Aggiunto attributo order alla funzione CreateBulk in service.go. Tale funzione è pensata per creare un nuovo turno per ogni playerId passato nel vettore PlayerId. Tale funzione dunque supporta anche lo scenario multigiocatore che verrà sviluppato nei prossimi prototipi dell'applicazione.

```
for i, id := range ids {  
    turns[i] = model.Turn{  
        PlayerID: id,  
        Order:    r.Order,  
        RoundID:  r.RoundId,  
        StartedAt: r.StartedAt,  
        ClosedAt: r.ClosedAt,  
    }  
}
```

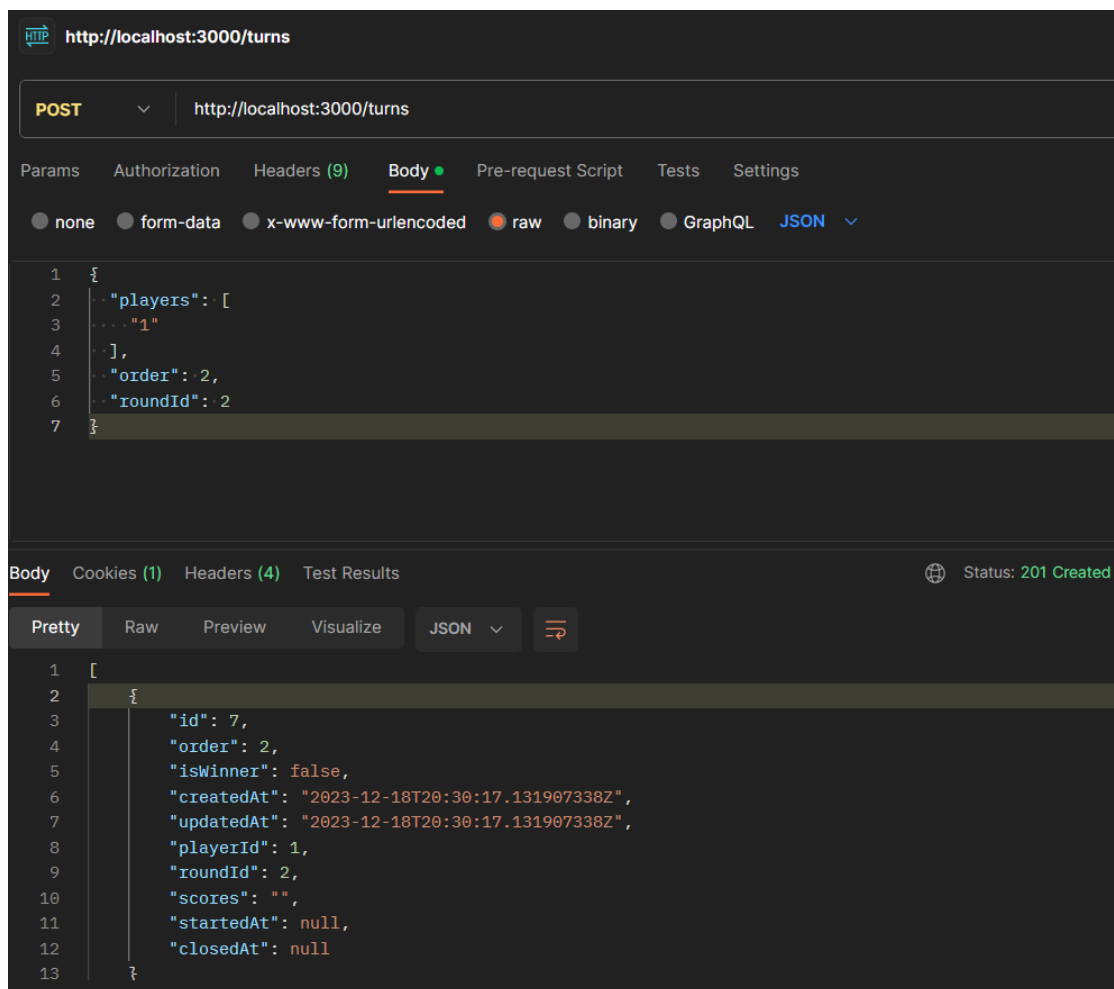
Figura 3.10: func createBulk

Di seguito è riportata la struttura finale della tabella *turns* del database del T4.

id	order	created_at	updated_at	started_at	closed_at	scores	is_winner	player_id	round_id
4	1	2023-12-18 20:24:36.182312+00	2023-12-18 20:24:36.182312+00	2023-12-18 20:24:36.110044+00	2023-12-18 20:24:54.89605+00	0	f	1	2
5	2	2023-12-18 20:24:56.218054+00	2023-12-18 20:24:56.218054+00	2023-12-18 20:24:56.192+00	2023-12-18 20:26:36.627762+00	0	f	1	2
6	3	2023-12-18 20:26:38.016878+00	2023-12-18 20:26:38.016878+00	2023-12-18 20:26:38.002+00	2023-12-18 20:36:00.767962+00	100	t	1	2

Figura 3.11: tabella *turns* dopo le modifiche

Inoltre è riportato anche il test della api `/turns` con Postman dopo le modifiche effettuate.

Figura 3.12: Test api `/turns` con Postman

Infine, in modo analogo all'aggiunta dell'attributo `order` in *turns*, si

è anche aggiunto l'attributo `username` all'interno della tabella *games*. Tale attributo è fondamentale per creare l'associazione tra una partita e il player che l'ha giocata (essendo lo `username` univoco), e verrà utilizzato successivamente per implementare uno storico globale fra tutti i giocatori.

	123 id	abc name	abc username
1	3	nome	prova@gmail.com
2	4	nome	ciao@ciao.it

Figura 3.13: Tabella *games*

3.7.4 Modifiche effettuate sul task T6 (R1)

Per salvare correttamente l'attributo `username` all'interno della tabella *games*, è stato necessario aggiungere un parametro in più all'api `/run` del task T6.

```

1 formData.append("turnId", localStorage.getItem("turnId"));
2 formData.append("roundId", localStorage.getItem("roundId"));
3 formData.append("gameId", localStorage.getItem("gameId"));
4 formData.append("testClassId", localStorage.getItem("classe"));
5 formData.append("difficulty", localStorage.getItem("difficulty"));
6 formData.append("type", localStorage.getItem("robot")); // modificato
7 formData.append("order", localStorage.getItem("orderTurno"));
8 formData.append("username", localStorage.getItem("username"));

```

Inoltre, nel file *myController.java*, è stato aggiunto un parametro in più alla PUT verso la tabella *games* all'atto della conclusione della partita.

```
1 httpPut = new HttpPut("http://t4-g18-app-1:3000/games/" +  
    ↪ String.valueOf(request.getParameter("gameId")));  
2  
3 obj = new JSONObject();  
4 obj.put("closedAt", time);  
5 obj.put("username", request.getParameter("username"));  
6 jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);  
7  
8 httpPut.setEntity(jsonEntity);  
9 response = httpClient.execute(httpPut);  
10 httpPut.releaseConnection();
```

3.7.5 Modifiche effettuate sul task T5 (R1)

La principale modifica effettuata sul T5 è stata quella della logica di aggiornamento di `gameID`, `roundID` e `turnID`.

In precedenza, ad ogni pressione del tasto *play/submit* dell'editor, veniva utilizzata la API */save-data*, la quale effettua sostanzialmente 3 richieste POST verso le tabelle *turns*, *round* e *games*.

```
1 public JSONObject saveGame(Game game) {  
2     try {  
3         String time =  
            ↪ ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);  
4         JSONObject obj = new JSONObject();  
5  
6         obj.put("difficulty", game.getDifficulty());  
7         obj.put("name", game.getName());  
8         obj.put("description", game.getDescription());  
9         obj.put("startedAt", time);  
10  
11         JSONArray playersArray = new JSONArray();  
12         playersArray.put(String.valueOf(game.getPlayerId()));  
13     }  
}
```

```
14         obj.put("players", playersArray);
15
16         HttpPost httpPost = new HttpPost("http://t4-g18-app-1:3000/games");
17         StringEntity jsonEntity = new StringEntity(obj.toString(),
18             ↪ ContentType.APPLICATION_JSON);
19
20         httpPost.setEntity(jsonEntity);
21
22         HttpResponse httpResponse = httpClient.execute(httpPost);
23         int statusCode = httpResponse.getStatusLine().getStatusCode();
24
25         if(statusCode > 299) {
26             System.err.println(EntityUtils.toString(httpResponse.getEntity()));
27             return null;
28         }
29
30         HttpEntity responseEntity = httpResponse.getEntity();
31         String responseBody = EntityUtils.toString(responseEntity);
32         JSONObject responseObj = new JSONObject(responseBody);
33
34         Integer game_id = responseObj.getInt("id"); // salvo il game id che l'Api
35             ↪ mi restituisce
36
37         JSONObject round = new JSONObject();
38         round.put("gameId", game_id);
39         round.put("testClassId", game.getClasse());
40         round.put("startedAt", time);
41
42         httpPost = new HttpPost("http://t4-g18-app-1:3000/rounds");
43         jsonEntity = new StringEntity(round.toString(),
44             ↪ ContentType.APPLICATION_JSON);
45
46         httpPost.setEntity(jsonEntity);
47
48         httpResponse = httpClient.execute(httpPost);
49         statusCode = httpResponse.getStatusLine().getStatusCode();
50
51         if(statusCode > 299) {
52             System.err.println(EntityUtils.toString(httpResponse.getEntity()));
53             return null;
54         }
```

```
53     responseEntity = httpResponse.getEntity();
54     responseBody = EntityUtils.toString(responseEntity);
55     responseObject = new JSONObject(responseBody);
56
57     Integer round_id = responseObject.getInt("id"); // salvo il round id che
           ↳ l'Api mi restituisce
58
59     JSONObject turn = new JSONObject();
60
61     turn.put("players", playersArray);
62     turn.put("roundId", round_id);
63     turn.put("startedAt", time);
64
65     httpPost = new HttpPost("http://t4-g18-app-1:3000/turns");
66     jsonEntity = new StringEntity(turn.toString(),
           ↳ ContentType.APPLICATION_JSON);
67
68     httpPost.setEntity(jsonEntity);
69
70     httpResponse = httpClient.execute(httpPost);
71     statusCode = httpResponse.getStatusLine().getStatusCode();
72
73     if(statusCode > 299) {
74         System.err.println(EntityUtils.toString(httpResponse.getEntity()));
75         return null;
76     }
77
78     responseEntity = httpResponse.getEntity();
79     responseBody = EntityUtils.toString(responseEntity);
80
81     JSONArray responseArrayObj = new JSONArray(responseBody);
82     Integer turn_id = responseArrayObj.getJSONObject(0).getInt("id"); //
           ↳ salvo il turn id che l'Api mi restituisce
83
84     JSONObject resp = new JSONObject();
85     resp.put("game_id", game_id);
86     resp.put("round_id", round_id);
87     resp.put("turn_id", turn_id);
88
89     return resp;
90 } catch (IOException e) {
           // Gestisci l'eccezione o restituisci un errore appropriato
91
```



```
92         System.err.println(e);  
93         return null;  
94     }
```

Ora, questa API viene utilizzata solo all'atto dell'inizio della partita, alla pressione del tasto submit della schermata /report del T5. In questo modo vengono correttamente create le rispettive entry nelle tabelle *games*, *rounds* e *turns*.

Resta ora di modificare l'aggiornamento del *turnId*. In particolare, alla pressione del tasto play/submit del T6, viene chiamata la api /run, che si occupa di fare delle PUT verso le tabelle *games*, *rounds* e *turns*. In particolare, la PUT verso *turns* ha lo scopo di aggiornare lo score e il vincitore di quel determinato turno. Inoltre, la PUT verso *games* e *rounds* viene effettuata solo se si è vinto quel determinato turno, in modo da aggiornare il campo *ClosedAt*, che indica il timestamp della chiusura della partita e del round.

Per maggiore chiarezza si riporta un diagramma di flusso che descrive il processo complessivo dell'aggiornamento degli ID.

Seguono un diagramma di attività e un diagramma sequenza per formalizzare il flow chart precedente.

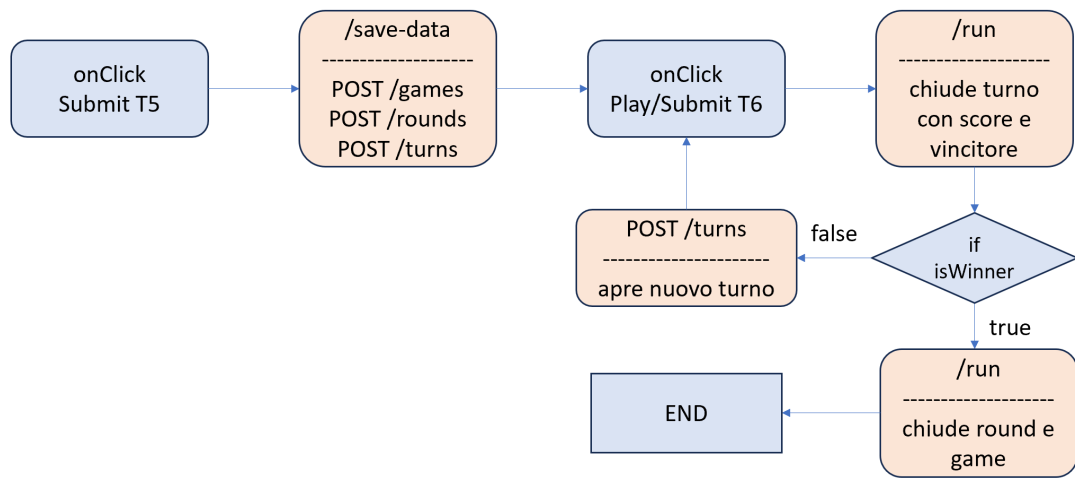
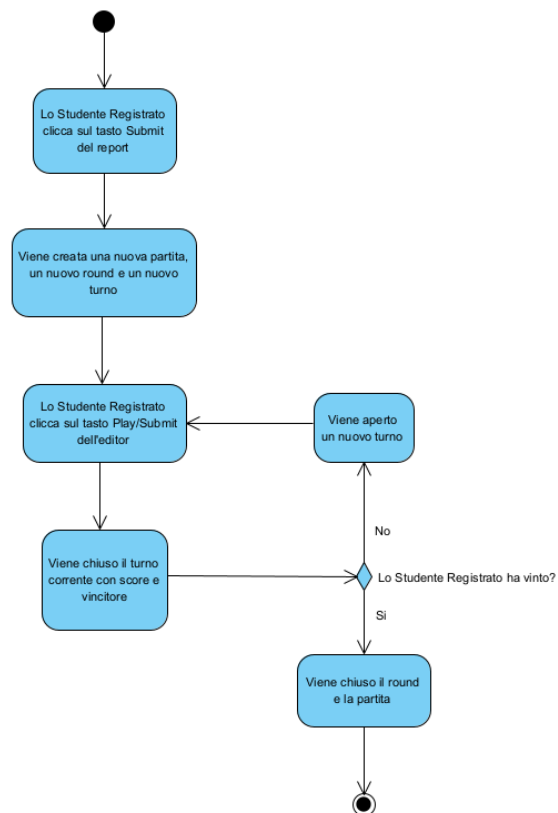
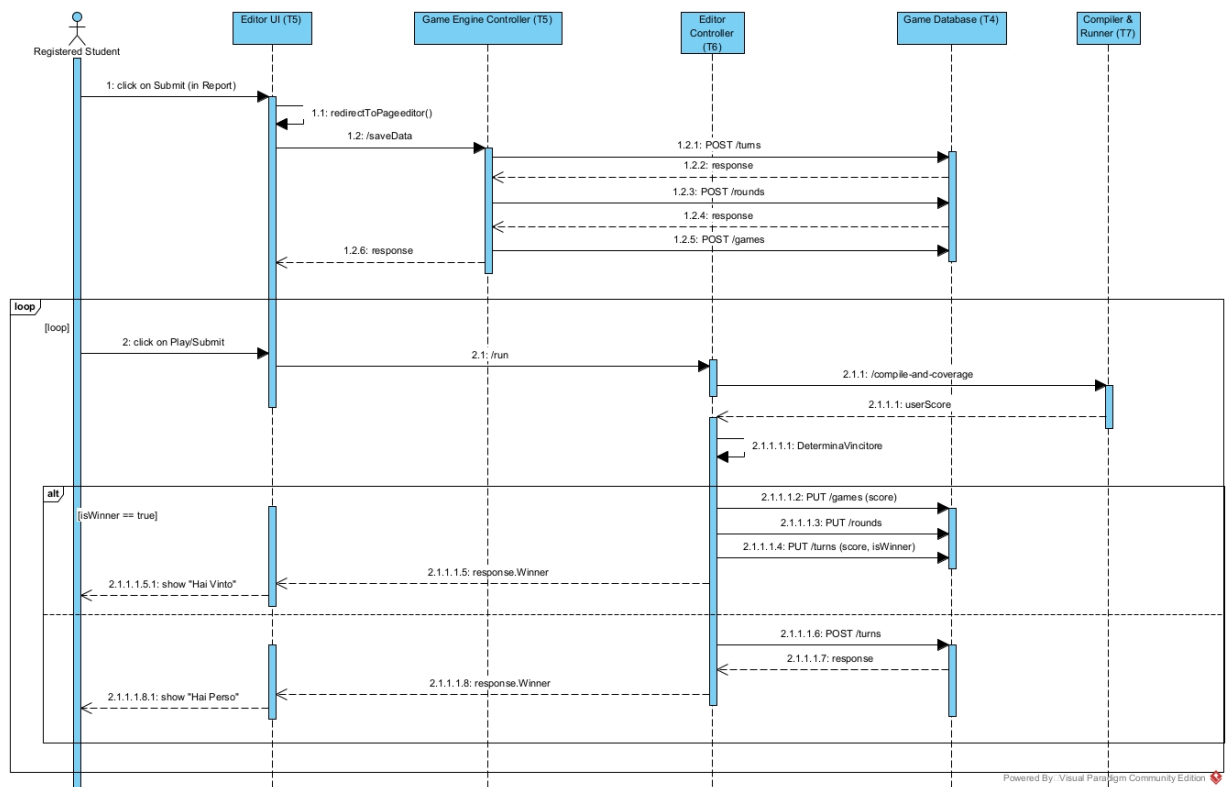


Figura 3.14: Processo di aggiornamento degli ID



CAPITOLO 3. IMPLEMENTAZIONE



Si riporta inoltre la parte di codice di editor.html coinvolta nelle modifiche.

```
1 $.ajax({
2     url: "http://localhost/api/run", // con questa verso il task 6, si salva e
3     type: "POST",
4     data: formData,
5     processData: false,
6     contentType: false,
7     dataType: "json",
8     success: function (response) {
9         console.log(response);
10        risp = response;
11        perc_robot = response.robotScore.toString();
12        consoleArea.setValue(response.outCompile);
13        highlightCodeCoverage($.parseXML(response.coverage));
14        alert(response.win == true ? "Hai vinto!" : "Hai perso!");
15
16        if (response.win == true) {
```

```
17     localStorage.setItem("gameId", null);
18   }else {
19
20     var now = new Date();
21     var startedAt = now.toISOString();
22
23     $.ajax({
24       url: "http://localhost:3000/turns",
25       type: 'POST',
26       contentType: "application/json",
27       data: JSON.stringify({
28         "players": [String(parseJwt(getCookie("jwt")).userId)],
29         "order": parseInt(localStorage.getItem("orderTurno"))+1,
30         "roundId": parseInt(localStorage.getItem("roundId")),
31         "startedAt": startedAt
32       }),
33       dataType: "json",
34       success: function (response) {
35         console.log("Success:", response);
36         localStorage.setItem("turnId", response[0].id.toString());
37         localStorage.setItem("orderTurno", response[0].order.toString())
38       },
39       error: function (error) {
40         console.error('Error:', error);
41         alert("Dati non inviati con successo");
42       }
43     });
44   },
45   },
46   error: function () {
47     console.log("Errore durante l'invio della partita.");
48   }
49 });
```

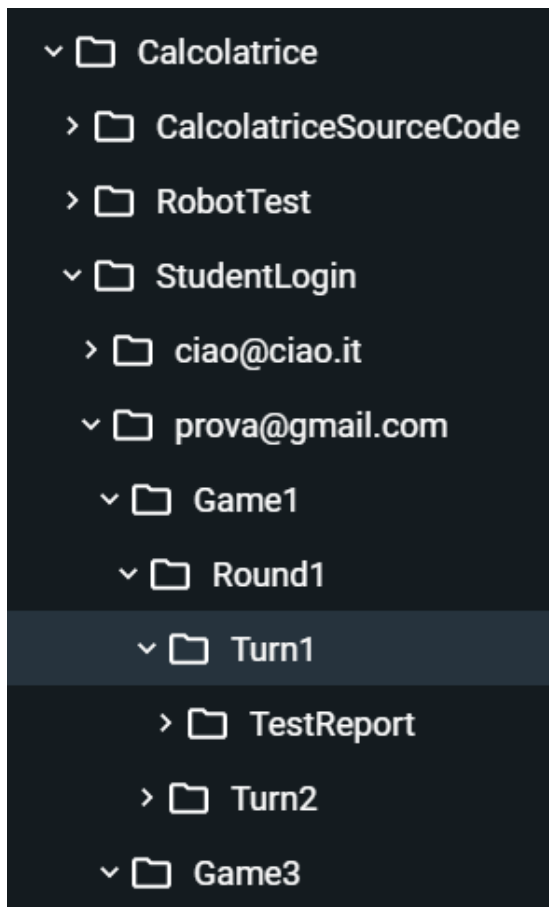
Come ultima modifica, è stato necessario salvare nel `localStorage` lo username dell'utente nel file *report.html*, in modo da tenere conto di chi ha giocato un determinato game. Lo username è stato estratto direttamente dalla finestra `html`.

```
1 var username = parseJwt(getCookie("jwt")).sub;
2 username = username.toString();
3 localStorage.setItem("username", username);
4 console.log("username :", username);
```

Dopo aver effettuato le modifiche, è stato finalmente possibile implementare la struttura delle directory finale nel volumeT8. Per tale scopo, è stato modificato il percorso di destinazione che viene passato alla API fornita dal T8, includendo anche le informazioni sulle cartelle del Game, del Round e del Turn, oltre lo username dell'utente che ha giocato la partita.

```
1 var test = 'VolumeT8/FolderTreeEvo/' + localStorage.getItem("classe") +
    ↳ '/StudentLogin/' + localStorage.getItem("username") + Game' +
    ↳ localStorage.getItem("gameId") + '/Round1/Turn' +
    ↳ localStorage.getItem("orderTurno") + '/TestReport';
```

Si riporta infine la struttura del volumeT8 a seguito delle modifiche.



3.8 Iterazione 4

In questa iterazione si è provveduto a concludere il requisito R1, implementando il salvataggio del file contenente le statistiche dello studente (`statistics.csv`), nella directory corretta, ossia quella del turno corrente nel volume T8.

Inoltre, è stato implementato il requisito R8, applicando la formula per il calcolo del punteggio finale della partita discussa nell'iterazione precedente.

Nelle sezioni successive vengono trattate in dettaglio le modifiche effettuate sui file per quanto riguarda l'implementazione del requisito R1.

3.8.1 Modifiche effettuate sul task T8 (R1)

Per salvare nel volume T8 il file `statistics.csv`, è stato necessario aggiungere un comando shell nello script `robot_misurazione_utente.sh` che consiste nel copiare il file dal filesystem del container a quello del volume.

Il file viene generato automaticamente dal robot EvoSuite. Tramite l'api fornita dal T8, e utilizzata nel T5, viene inviato al robot il codice del test scritto dallo studente e tre percorsi:

- Il percorso P1 della classe da testare;

- Il percorso P2 dove salvare il test scritto dallo studente (percorso che cambia in funzione del game, round e turno in cui il player gioca);
- Il percorso P3 dove salvare il file `statistics.csv` (nel nostro caso `/app`). Si riporta per completezza un estratto della documentazione del T8 originale che descrive la struttura della API.

<http://ip:porta/api/percorso classe+percorso test+percorso salvataggio>

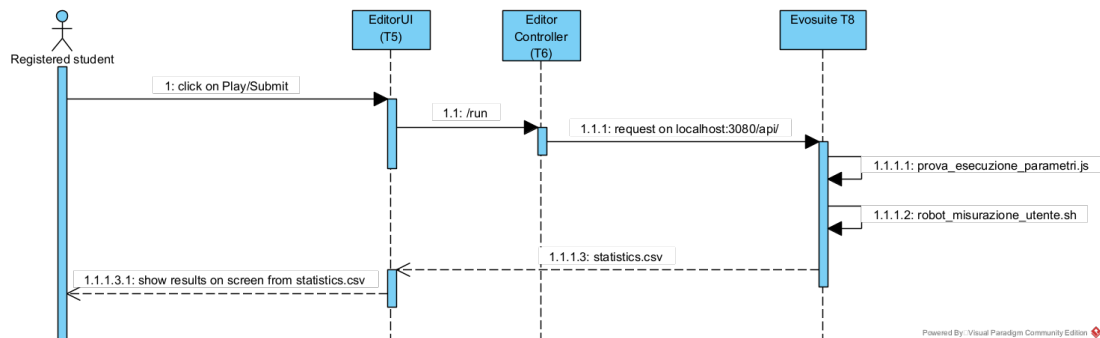
Il file viene generato nella cartella di lavoro del robot `/evosuite-report` e successivamente viene rimosso. Nel codice era presente un comando per copiare questo file prima di essere eliminato, ma il percorso era totalmente sbagliato, causando di fatto la perdita di questo file. Dopo un'attenta analisi degli script utilizzati nel T8, è stato necessario modificare il comando di copia con i percorsi corretti.

```
96  cd $PERCORSO
97
98
99  mv -f evosuite-report/statistics.csv $PERCORSO_CSV
100
101  cp -r -f $PERCORSO_CSV/statistics.csv $PERCORSO_TEST
102
103  rm -r $PERCORSO/evosuite-report
```

`$PERCORSO_TEST` corrisponde al P3 della api precedente. Infatti, in seguito all'implementazione della prima parte del requisito R1, il file `statistics.csv` viene salvato nella stessa cartella del codice del test scritto dallo studente. Per cui il comando `cp -r -f` permette di salvare

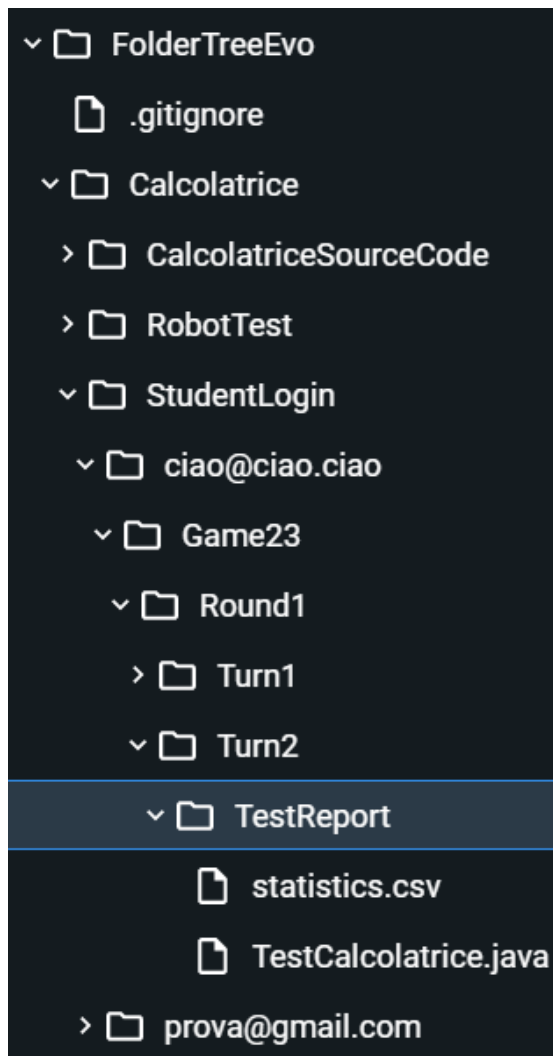
correttamente il file rispettando la struttura delle directory del volumeT8, concludendo, di fatto, l'implementazione del requisito R1.

Dopo le modifiche effettuate sul task T8, si riporta un sequence diagram che illustra il processo di creazione e salvataggio di *statistics.csv* e della restituzione dei risultati di copertura nella finestra dell'editor:



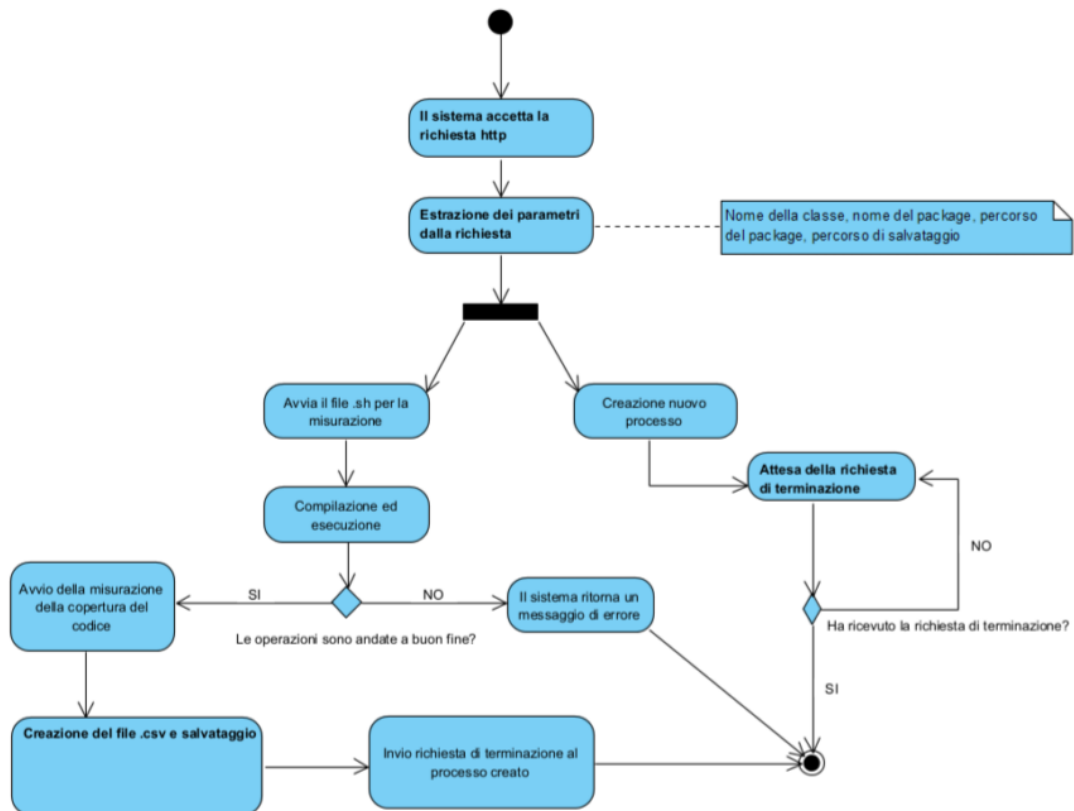
- Lo step 1.1.1 consiste nell'inviare una http request all'endpoint `/api` di T8, il cui body contiene: il codice test; il percorso classe UT; il percorso del volume su cui scrivere il test scritto dallo studente; il percorso di salvataggio del file *statistics.csv*;
- Lo step 1.1.1.1 consiste nell'attivazione della funzione di parsing contenuta in *prova_esecuzione_parametri.js*, la quale spaccetta il messaggio ricevuto estraendo i percorsi necessari all'esecuzione del programma Evosuite;
- Nello step 1.1.1.2, tali percorsi vengono memorizzati come variabili in *robot_misurazione_utente.sh*, permettendo l'esecuzione degli script Evosuite che producono il file *statistics.csv*, contenente le statistiche della partita giocata dallo studente.

Infine, viene mostrata la struttura finale del filesystem, comprensiva del file *statistics.csv*.



3.8.2 Diagramma di attività di misurazione-utente

In seguito, per chiarire il processo di misurazione del test case scritto dallo studente, si riporta un diagramma di attività:



3.8.3 Ipotesi condivisione volumeT8 con T5

Tra le ipotesi preliminari, per agevolare il lavoro di sviluppo, si era sondata l'ipotesi di condividere il volumeT8 anche con il container di T5, risparmiando concettualmente una chiamata alla API fornita dal T8 per salvare il test scritto dallo studente nel volumeT8. Tuttavia, questa ipotesi non teneva in considerazione del fatto che il test scritto dallo studente deve comunque essere inviato al T8 per il calcolo delle varie metriche di copertura (generando il file statistics.csv).

Dunque è ragionevole continuare ad utilizzare la API fornita dal T8, diminuendo la responsabilità singola del T5 ed enfatizzando la comunicazione tra il T5 e il T8, rispettando la struttura a microservizi

dell'applicazione complessiva.

3.8.4 Implementazione del requisito R8

Per implementare il R8 sono stati modificati diversi file, in quanto si è dovuto aggiornare la tabella *games* per includere l'attributo *scores*, che di default viene messo a 0. Il calcolo dello score viene effettuato nella api */run*, attraverso la funzione *calculateScore*, nel caso in cui il turno giocato risulta quello vincente.

3.8.5 Modifiche effettuate sul task T4 (R8)

Così come fatto per l'attributo *order* menzionato nell'iterazione precedente, è stato prima di tutto modificato lo struct *Game* in *model.go* per includere l'attributo score. Da notare che è stato modificato anche l'ordine delle colonne della tabella in modo da essere più comprensibile ed in linea con le altre tabelle.

```

1 type Game struct {
2     ID          int64      `gorm:"primaryKey;autoIncrement"`
3     Name         string     `gorm:"default:null"`
4     Username     string     `gorm:"default:null"`
5     CurrentRound int        `gorm:"default:1"`
6     Description   sql.NullString `gorm:"default:null"`
7     Difficulty    string     `gorm:"default:null"`
8     Score         float64    `gorm:"default:0"`
9     CreatedAt     time.Time  `gorm:"autoCreateTime"`
10    UpdatedAt     time.Time  `gorm:"autoUpdateTime"`
11    StartedAt     *time.Time `gorm:"default:null"`
12    ClosedAt      *time.Time `gorm:"default:null"`
13    Rounds        []Round    `gorm:"foreignKey:GameID;constraint:OnDelete:CASCADE;"`
14    Players       []Player
15    ↪ `gorm:"many2many:player_games;foreignKey:ID;joinForeignKey:GameID;
      References:AccountID;joinReferences:PlayerID"`

```

```
16 }
```

Inoltre, è stato modificata la struct *Game* di *game.go* per contenere l'attributo *score*. Tale struct è progettata per essere facilmente convertita da e verso il formato JSON.

```
1 type Game struct {
2     ID          int64      `json:"id"`
3     CurrentRound int        `json:"currentRound"`
4     Username     string     `json:"username"`
5     Description  string     `json:"description"`
6     Difficulty   string     `json:"difficulty"`
7     Score        float64    `json:"score"`
8     CreatedAt    time.Time  `json:"createdAt"`
9     UpdatedAt    time.Time  `json:"updatedAt"`
10    StartedAt     *time.Time `json:"startedAt"`
11    ClosedAt      *time.Time `json:"closedAt"`
12    Name          string     `json:"name"`
13    Players       []Player   `json:"players,omitempty"`
14 }
```

E' stata quindi modificata la struct per la gestione del JSON sulle richieste di Update.

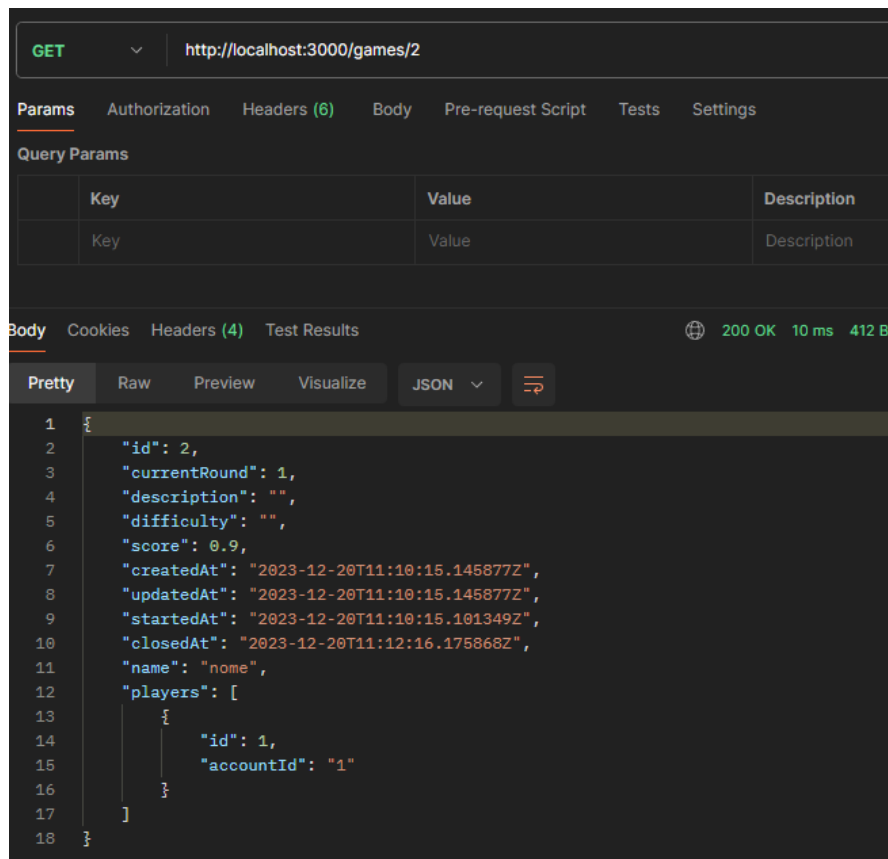
```
1 type UpdateRequest struct {
2     CurrentRound int        `json:"currentRound"`
3     Name          string     `json:"name"`
4     Username      string     `json:"username"`
5     Description   string     `json:"description"`
6     Score         float64    `json:"score"`
7     StartedAt     *time.Time `json:"startedAt,omitempty"`
8     ClosedAt      *time.Time `json:"closedAt,omitempty"`
9 }
```

Infine è stata modificata la funzione di conversione da un oggetto di ti-

po *model.Game* (rappresentante il modello del database) in un oggetto di tipo *Game* (progettato per l'interazione con l'API).

```
1 func fromModel(g *model.Game) Game {  
2     return Game{  
3         ID:          g.ID,  
4         CurrentRound: g.CurrentRound,  
5         Username:    g.Username,  
6         Difficulty:   g.Difficulty,  
7         Description:  g.Description.String,  
8         Score:        g.Score,  
9         CreatedAt:    g.CreatedAt,  
10        UpdatedAt:    g.UpdatedAt,  
11        Name:          g.Name,  
12        StartedAt:    g.StartedAt,  
13        ClosedAt:     g.ClosedAt,  
14        Players:      parsePlayers(g.Players),  
15    }  
16 }
```

A seguito delle modifiche effettuate, sono state testate con Postman le api modificate:



Inoltre, si riporta, per completezza, la tabella *games* complessiva, evidenziando l'aggiunta dell'attributo **score**.

	id	name	username	current_round	description	difficul	score	created_at	updated_at
1	3	nome	prova@gmail.com	1	[NULL]	[NULL]	1	2024-01-19 12:14:40.058 +0100	2024-01-19 12:14:40.058 +0100
2	4	nome	ciao@ciao.it	1	[NULL]	[NULL]	1	2024-01-19 12:36:33.421 +0100	2024-01-19 12:36:33.421 +0100

Figura 3.15: Struttura definita tabella *Games*

3.8.6 Modifiche effettuate sul task T6 (R8)

Per introdurre un criterio efficace per il calcolo del punteggio, adatto allo scenario di gioco attualmente implementato, che prevede la fine della partita dopo la vittoria di un turno, si è deciso di stabilirlo in funzione del numero di turni che il giocatore impiega per vincere. Infatti,

poiché per terminare la partita il giocatore deve obbligatoriamente vincere, viene premiato con un punteggio più alto il giocatore che vince in un numero di turni inferiore, con una LOC più alta. Infatti, il termine di LOC viene aggiunto per tenere conto della difficoltà di quella partita: vincere con il 100% di LOC è diverso dal vincere con il 10% (in questo secondo caso la partita risulta abbastanza semplice).

La metrica di punteggio è stata implementata nella funzione *calculateScore*, la quale a sua volta è stata inserita nella classe *ParseUtil* in modo da essere modificabile ed estensibile.

```
1 public static double calculateScore(int loc, int numTurnsPlayed) {  
2     double locAsDoublePerc = ((double) loc)/100;  
3     return (locAsDoublePerc - (0.1 * (numTurnsPlayed - 1)));  
4 }
```

Tale funzione viene chiamata nell'api */run*. Di seguito viene mostrata la porzione di codice in cui viene prelevato lo *userScore* (che sarebbe la LOC) ed il valore del turno corrente, entrambi utilizzati nella formula di calcolo. Ciò viene fatto solo quando il turno corrente risulta quello vincente.

```
1 double gameScore = 0;  
2 if (roboScore > userScore) {  
3     obj.put("isWinner", false);  
4 } else {  
5     obj.put("isWinner", true);  
6     int numTurnsPlayed = Integer.parseInt(request.getParameter("order"));  
7     gameScore = ParseUtil.calculateScore(userScore, numTurnsPlayed);  
8 }
```

Infine, *gameScore* viene passato nel JSON di response:

```

1      JSONObject result = new JSONObject();
2      result.put("outCompile", outCompile);
3      result.put("coverage", xml_string);
4      result.put("win", userScore >= roboScore);
5      result.put("robotScore", roboScore);
6      result.put("score", userScore);
7      result.put("gameScore", gameScore);

```

3.8.7 Modifiche effettuate sul task T5 (R8)

E' stato modificato il file *editor.html* per aggiungere il parametro *orderTurno* da dare nella request verso */run*, che come detto prima servirà per il calcolo del punteggio. Inoltre è stata aggiunta la variabile *gameScore* la quale viene mostrata nell>alert di resoconto della partita nel caso di vittoria.

```

1 var formData = new FormData();
2 formData.append("testingClassName", "Test" + localStorage.getItem("classe") +
   ↪ ".java");
3 formData.append("testingClassCode", editor.getValue());
4 formData.append("underTestClassCode", localStorage.getItem("classe")+".java");
5 formData.append("underTestClassCode", sidebarEditor.getValue());
6
7 formData.append("turnId", localStorage.getItem("turnId"));
8 formData.append("roundId", localStorage.getItem("roundId"));
9 formData.append("gameId", localStorage.getItem("gameId"));
10 formData.append("testClassId", localStorage.getItem("classe"));
11 formData.append("difficulty", localStorage.getItem("difficulty"));
12 formData.append("type", localStorage.getItem("robot")); // modificato
13 formData.append("order", localStorage.getItem("orderTurno"));
14 formData.append("username", localStorage.getItem("username"));
15 $.ajax({

```

```
16 url: "http://localhost/api/run", // con questa verso il task 6, si salva e
    ↳ conclude la partita e si decreta il vincitore
17 type: "POST",
18 data: formData,
19 processData: false,
20 contentType: false,
21 dataType: "json",
22 success: function (response) {
23     console.log(response);
24     risp = response;
25     perc_robot = response.robotScore.toString();
26     gameScore = response.gameScore.toString();
27     consoleArea.setValue(response.outCompile);
28     highlightCodeCoverage($.parseXML(response.coverage));
29     alert(response.win == true ? "Hai vinto! Punteggio: " + gameScore : "Hai
    ↳ perso!");
30     if (response.win == true) {
31         turno++; // incremento il numero di turno giocati fino ad ora
32         localStorage.setItem("gameId", null); // setto gameId null invece che
    ↳ tutto
33     }else {
34         console.log("Player: " + [parseJwt(getCookie("jwt")).userId]);
35         console.log("turnId: " + localStorage.getItem("turnId"));
36         console.log("roundId: " + localStorage.getItem("roundId"));
37         $.ajax({
38             url: "http://localhost:3000/turns",
39             type: 'POST',
40             contentType: "application/json",
41             data: JSON.stringify({
42                 "players": [String(parseJwt(getCookie("jwt")).userId)],
43                 "order": parseInt(localStorage.getItem("orderTurno"))+1,
44                 "roundId": parseInt(localStorage.getItem("roundId")),
45                 "startedAt": new Date().toISOString() // Get current date and
    ↳ time in ISO 8601 format
46             })),
47             dataType: "json",
48             success: function (response) {
49                 console.log("Success:", response);
50                 localStorage.setItem("turnId", response[0].id.toString());
51                 localStorage.setItem("orderTurno", response[0].order.toString());
52                 turno++; // incremento il numero di turno giocati fino ad ora
53             },
```

```
54         error: function (error) {  
55             console.error('Error:', error);  
56             alert("Dati non inviati con successo");  
57         }  
58     });  
59 }  
60 },  
61 error: function () {  
62     // Gestisci l'errore, ad esempio mostra un messaggio di errore  
63     console.log("Errore durante l'invio della partita.");  
64 }  
65 });
```

Infine, al click del pulsante *play/submit*, nella finestra dell'esito della partita, è stato aggiunto anche il punteggio con cui si è vinta la partita.



Capitolo 4

Tools e frameworks utilizzati

Nel corso dello sviluppo del progetto abbiamo utilizzato diversi tool. Di seguito è riportata una descrizione per ognuno dei tool.

4.1 DBeaver

DBeaver è un client di database universale e gratuito che supporta molti tipi di database popolari. Si tratta di un software open-source che fornisce un'interfaccia grafica per interagire con database attraverso diverse tecnologie di connessione ed il supporto dell'editor SQL. Di seguito la configurazione necessaria al collegamento col database del T4. La password utilizzata è "postgres".

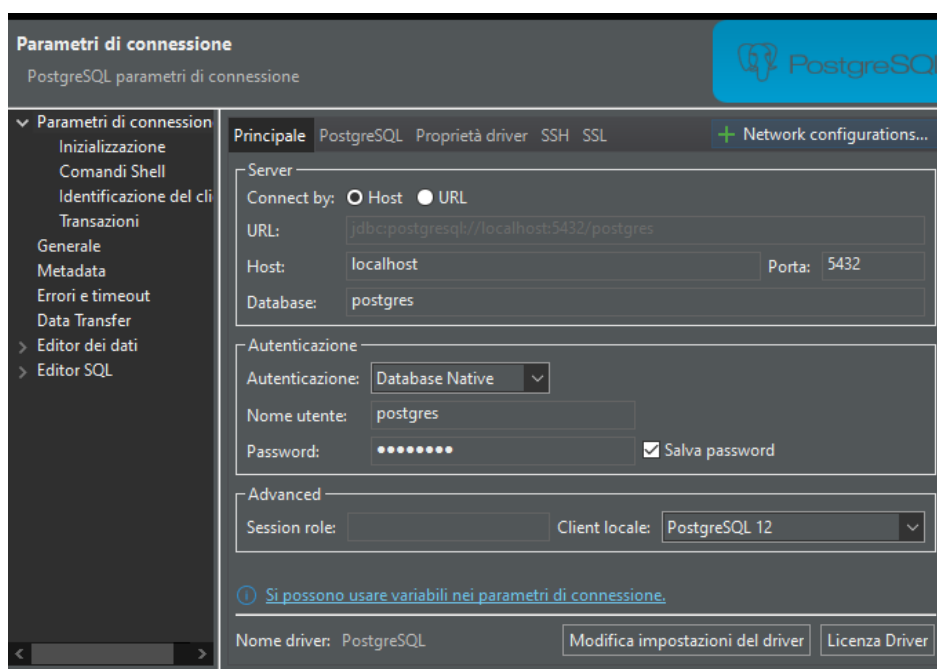


Figura 4.1: Configurazione connessione database T4

4.2 Selenium

Selenium è un framework di automazione dei test utilizzato per testare le applicazioni web attraverso una serie di script. Il suo principale obiettivo è automatizzare le azioni dell'utente su un browser web, consentendo agli sviluppatori e agli operatori dei test di eseguire in modo efficiente e ripetibile test funzionali su applicazioni web. Di seguito sono riportati alcuni aspetti chiave:

- **Automazione del browser:** Selenium consente di automatizzare le interazioni dell'utente con un browser web. Può simulare azioni come fare clic su elementi, inserire testo, inviare moduli, navigare tra le pagine e molto altro;
- **Supporto multiplatforma:** Selenium è compatibile con di-

versi browser web come Chrome, Firefox, Safari, Edge, e altri. Inoltre, è multiplatforma e può essere utilizzato su sistemi operativi come Windows, macOS e Linux;

- **Linguaggi di programmazione supportati:** Selenium supporta diversi linguaggi di programmazione, tra cui Java, Python, C, Ruby, JavaScript e molti altri. Gli sviluppatori possono scegliere il linguaggio che preferiscono per scrivere i loro script di automazione;
- **Integrazione con strumenti di test:** Selenium può essere integrato con vari strumenti e framework di test, come TestNG, JUnit e NUnit. Questa integrazione facilita la gestione dei test, l'esecuzione in parallelo e la generazione di report dettagliati;
- **Testing Cross-browser e cross-platform:** Selenium è ampiamente utilizzato per eseguire test su diverse configurazioni di browser e sistemi operativi, garantendo che un'applicazione web funzioni correttamente su diverse combinazioni di browser e piattaforme;
- **Registrazione e riproduzione:** Selenium permette la registrazione e la riproduzione di scenari di test, consentendo agli sviluppatori di registrare le azioni dell'utente e riprodurle in modo automatico durante i test successivi.

In sintesi, Selenium è uno strumento potente per l'automazione dei test di applicazioni web. È ampiamente utilizzato per garantire la qualità del software, ridurre il tempo necessario per eseguire i test e facilitare l'integrazione continua nello sviluppo del software.

4.3 Postman

Postman è un'applicazione utilizzata principalmente per semplificare il processo di sviluppo, test e gestione delle API (Interfacce di Programmazione delle Applicazioni). È uno strumento molto popolare tra gli sviluppatori di software e gli ingegneri API, poiché offre un'interfaccia utente intuitiva che consente di inviare richieste HTTP a un server e ricevere le risposte.

Di seguito riportiamo alcune delle funzionalità principali di Postman:

- **Creazione e Invio di Richieste API:** permette agli sviluppatori di creare, inviare e ricevere richieste HTTP o HTTPS a API RESTful o endpoint;
- **Automatizzazione delle Richieste:** gli utenti possono creare collezioni di richieste e ambienti per automatizzare i processi di test e di sviluppo delle API;
- **Test delle API:** offre un sistema di test integrato che consente agli sviluppatori di verificare le risposte delle API, validare i dati di risposta e monitorare le prestazioni;

- **Documentazione API:** Postman consente di generare automaticamente documentazione chiara e dettagliata per le API, rendendo più facile la comprensione delle funzionalità e dei parametri disponibili;
- **Collaborazione:** permette agli sviluppatori di lavorare insieme su progetti di API, condividendo facilmente collezioni di richieste, ambienti e documentazione;
- **Simulazione di Ambienti:** gli sviluppatori possono simulare diversi ambienti (ad esempio, sviluppo, test, produzione) per testare le API in diversi contesti;
- **Monitoraggio delle Prestazioni:** consente di monitorare le prestazioni delle API nel tempo, individuando eventuali problemi di latenza o altri problemi.

In sintesi, Postman semplifica il processo di sviluppo e gestione delle API fornendo uno strumento completo per testare, documentare e collaborare su progetti di API.

4.4 Swagger

Swagger è un insieme di strumenti open-source progettati per aiutare gli sviluppatori a progettare, documentare e consumare servizi web RESTful. L'obiettivo principale di Swagger è semplificare la creazione

e la gestione di documentazione chiara e leggibile per le API REST, rendendo più facile la comprensione delle funzionalità e dei parametri di un servizio web.

Di seguito si riportano alcuni aspetti chiavi di Swagger:

- **Specifiche OpenAPI:** Swagger utilizza le specifiche OpenAPI (in precedenza conosciute come Swagger Specification) per definire l'interfaccia di programmazione di un servizio web RESTful. Le specifiche OpenAPI sono scritte in formato JSON o YAML e descrivono in modo dettagliato tutte le risorse, gli endpoint, i metodi HTTP supportati, i parametri richiesti e le risposte possibili.
- **Generazione automatica della documentazione:** utilizzando le specifiche OpenAPI, Swagger può generare automaticamente una documentazione interattiva per le API. Questa documentazione include informazioni dettagliate su come utilizzare ogni endpoint, quali parametri fornire e quali risposte aspettarsi.
- **Toolchain completa:** Swagger offre una vasta gamma di strumenti, inclusi Swagger Editor (per creare o modificare le specifiche OpenAPI), Swagger UI (per visualizzare la documentazione generata in modo interattivo), e Swagger Codegen (per generare client API e server in diversi linguaggi di programmazione basati sulle specifiche OpenAPI);

- **Test automatici delle API:** Swagger consente agli sviluppatori di testare le API direttamente dalla documentazione generata, fornendo un modo rapido ed efficiente per verificare che le API rispondano correttamente alle richieste;
- **Collaborazione e conformità standard:** l'utilizzo di specifiche OpenAPI permette agli sviluppatori di collaborare facilmente e di garantire la conformità alle specifiche API stabilite. Le specifiche OpenAPI sono supportate da un'ampia comunità e sono diventate uno standard nel settore.

In sintesi, Swagger è una suite di strumenti che facilita la progettazione, la documentazione e la gestione delle API REST, contribuendo a migliorare la comunicazione tra gli sviluppatori, a semplificare l'integrazione di servizi web e a garantire una documentazione chiara e accurata.

Capitolo 5

Deployment

Nel diagramma sottostante, viene messa in evidenza la disposizione fisica dei componenti di un sistema software e la loro interazione all'interno di un ambiente di esecuzione.

L'applicativo è ospitato su un server che utilizza il sistema operativo Windows 11. In particolare, il software è in esecuzione in ambiente Docker ed è distribuito su vari container. Da evidenziare i colore dei vari container:

- **Rosa:** modificati;
- **Verdi:** inalterati.

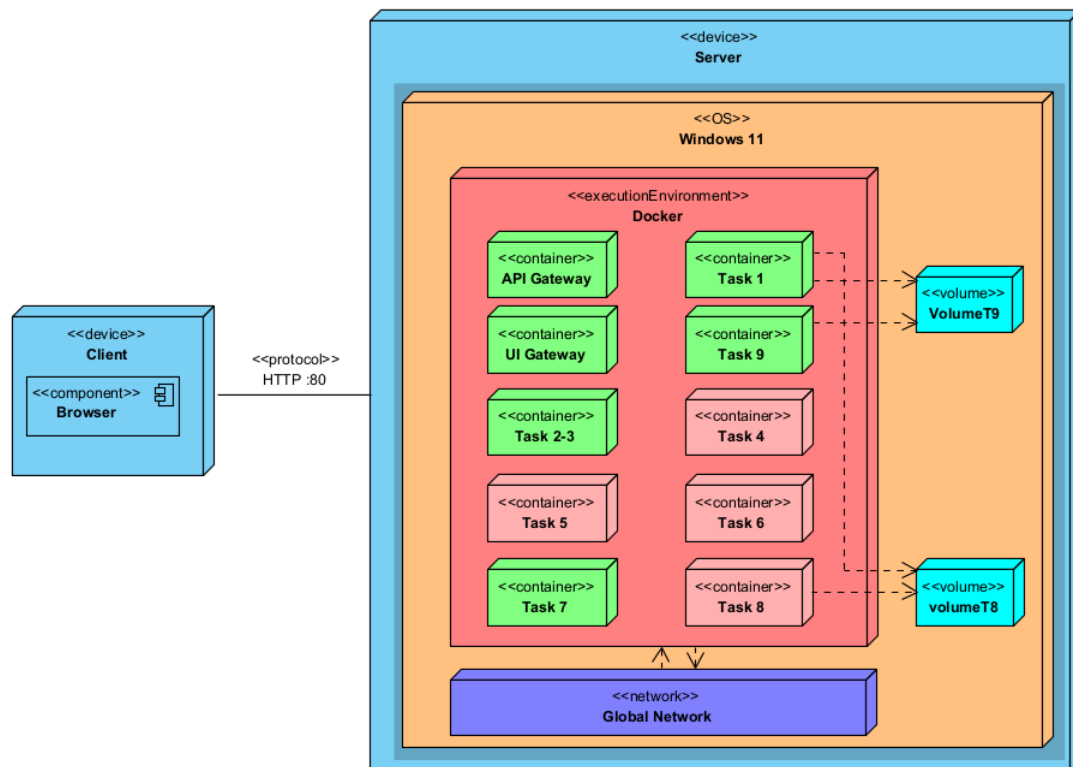
Nel diagramma sono presenti degli elementi a cui prestare attenzione:

- La rete è condivisa dai container di tutti i task, ciò permette di isolare i container dalla rete dell'host e al tempo stesso permet-

tere la comunicazione tra di essi (si ottiene una rete virtuale): l'unico container che è esposto verso l'esterno è il gateway che si occupa di gestire le richieste;

- Il volumeT8 è condiviso dai container dei task 1 e 8, la parte del filesystem in comune è quella relativa alle classi e alla copertura generata da EvoSuite;
- Il volumeT9 è condiviso dai container dei task 1 e 9, questo perchè entrambi devono accedere alla stessa porzione di filesystem: quella relativa alle classi e alla copertura generata da Randoop e Emma.

Si riporta dunque il diagramma di deployment.



Capitolo 6

Testing

Per la fase di testing, oltre ai test delle API già descritti precedentemente nella sezione di implementazione, si è fatto uso di un test di tipo end-to-end, tramite l'utilizzo del tool Selenium.

Il nostro obiettivo è l'integrazione di tutti i componenti per ottenere un'applicazione funzionante. A tale scopo, è stato condotto il "Test End-to-End (E2E)", noto anche come "Test di Sistema". Si tratta di un metodo di testing che valuta l'intero flusso dell'applicazione, garantendo che tutti i componenti, combinati tra loro, operino come previsto in scenari reali. Infatti, il software viene testato dal punto di vista dell'utente finale, simulando uno scenario realistico, inclusa l'interfaccia utente, i servizi di backend, i database e la comunicazione di rete.

Tra gli scopi c'è quello di convalidare il comportamento complessivo dell'applicazione, includendo funzionalità, affidabilità, prestazioni

e sicurezza. L'obiettivo finale del test E2E è individuare difetti o problemi che potrebbero sorgere quando le diverse parti dell'applicazione interagiscono tra loro. Solitamente, il test E2E viene eseguito dopo il test di integrazione, che verifica i singoli moduli, e prima del test di accettazione dell'utente, che assicura che l'applicazione soddisfi i requisiti dell'utente.

Come citato nella sezione *Tools e Frameworks utilizzati*, in questa fase è stato fatto uso di Selenium.

6.1 Casi di Test

Di seguito sono riportati i vari test effettuati. I flussi dell'applicazione testati sono quelli relativi all'accesso (Login Test) e all'utilizzo dell'editor (Editor Test). In entrambi si possono individuare due fasi importanti: una relativa alla configurazione dell'ambiente e un'altra relativa ai test veri e propri.

6.1.1 Login

Setup : La fase di configurazione consiste nel definire quali operazioni devono essere effettuate in via preliminare, prima che i test comincino. Entrando più nello specifico, tali operazioni riguardano:

- l'impostazione del driver di Selenium che gli permette di utilizzare il browser;

- l'apertura del browser all'avvio di un test;
- la chiusura del browser al termine di un test.

```
...
private static ChromeDriver driver;
private static int timeout = 10;

@BeforeClass
public static void setDriver() {
    System.setProperty("webdriver.chrome.driver", "/path/to/driver");
}

@Before
public void openBrowser(){
    driver = new ChromeDriver();
    driver.manage().timeouts().implicitlyWait(timeout, TimeUnit.SECONDS);
}

@After
public void closeBrowser(){
    driver.close();
}
...
```

Figura 6.1: Login Test – Setup

Credenziali valide : Tale test verifica che l'accesso avvenga correttamente utilizzando credenziali valide (ovvero di un studente già registrato). Per fare ciò, il robot effettua le seguenti operazioni:

1. si collega alla pagina di accesso (/login);
2. inserisce le credenziali (email e password);
3. clicca il pulsante di accesso;
4. verifica che sia stato effettuato il redirect alla prima pagina dell'editor (/main).


```
...
@Test
public void validCredentials(){
    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("mariorossi@mail.com");
    driver.findElement(By.id("password")).sendKeys("Mariorossi0");
    driver.findElement(By.cssSelector("input[type=submit]")).click();

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    String urlPaginaDiRedirezione = "http://localhost/main";
    try {
        wait.until(ExpectedConditions.urlToBe(urlPaginaDiRedirezione));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    Assert.assertEquals("Test fallito! Il login non è avvenuto correttamente.",
        driver.getCurrentUrl(), urlPaginaDiRedirezione);
}
...
```

Figura 6.2: Login Test – Credenziali valide

Credenziali non valide : Tale test verifica che l'accesso venga negato utilizzando credenziali non valide. Per fare ciò, il robot effettua le seguenti operazioni:

1. si collega alla pagina di accesso (/login);
2. inserisce le credenziali (email e password);
3. clicca il pulsante di accesso;
4. verifica che sia stato effettuato il redirect alla prima pagina dell'editor (/main).

```
...
@Test
public void invalidCredentials(){
    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("pippobaudo@gmail.com");
    driver.findElement(By.id("password")).sendKeys("password");
    driver.findElement(By.cssSelector("input[type=submit]")).click();

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.textToBe(By.tagName("body"), "Incorrect
password"));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    Assert.assertEquals("Test fallito! Il login è avvenuto correttamente.",
driver.findElement(By.tagName("body")).getText(), "Incorrect password");
}
...
```

Figura 6.3: Login Test – Credenziali non valide

6.1.2 Editor

Setup : La fase di configurazione consiste nel definire quali operazioni devono essere effettuare in via preliminare, prima che i test comincino. Entrando più nello specifico, tali operazioni riguardano:

- l'impostazione del driver di Selenium che gli permette di utilizzare il browser;
- l'impostazione del percorso dei download;
- l'apertura del browser e l'autenticazione dell'utente all'avvio di un test;
- la chiusura del browser al termine di un test.

```

...
private static ChromeDriver driver;
private static int timeout = 60;

@BeforeClass
public static void setDriver() {
    System.setProperty("webdriver.chrome.driver", "/path/to/driver");
}

@Before
public void openBrowser(){
    ChromeOptions options = new ChromeOptions();
    options.setCapability(CapabilityType.UNEXPECTED_ALERT_BEHAVIOUR,
UnexpectedAlertBehaviour.ACCEPT);
    HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
    chromePrefs.put("profile.default_content_settings.popups", 0);
    chromePrefs.put("download.default_directory", "/path/to/download");
    options.setExperimentalOption("prefs", chromePrefs);

    driver = new ChromeDriver(options);
    driver.manage().timeouts().implicitlyWait(timeout, TimeUnit.SECONDS);

    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("mariorossi@mail.com");
    driver.findElement(By.id("password")).sendKeys("Mariorossi0");
    driver.findElement(By.cssSelector("input[type=submit]")).click();

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    String urlPaginaDiRedirezione = "http://localhost/main";
    try {
        wait.until(ExpectedConditions.urlToBe(urlPaginaDiRedirezione));
    } catch(TimeoutException e) {
        Assert.fail();
    }
}

@After
public void closeBrowser(){
    driver.close();
}
...

```

Figura 6.4: Editor Test – Setup

Download classe : Tale test verifica che il download di una classe avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. clicca la prima classe disponibile;

2. clicca il pulsante di download;
3. verifica che il file scaricato sia presente all'interno dei download.

```
...
@Test
public void download() throws InterruptedException {
    driver.findElement(By.id("0")).click();

    driver.findElement(By.id("downloadButton")).click();

    File f = new File("/path/to/download/class.java");

    Thread.sleep(5000);

    Assert.assertTrue(f.exists());
}
...
```

Figura 6.5: Editor Test – Download classe

Scelta classi e robot : Tale test verifica che la scelta di una classe e di un robot avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. seleziona la classe e il robot;
2. clicca il pulsante di conferma;
3. verifica che sia stato effettuato il redirect alla pagina di conferma (/report).

```
...
@Test
public void selection() {
    String urlPaginaDiRedirezione = "http://localhost/report";

    moveToReport(urlPaginaDiRedirezione);

    Assert.assertEquals("Test fallito! La selezione non è avvenuta correttamente.",
        driver.getCurrentUrl(), urlPaginaDiRedirezione);
}
...
```

Figura 6.6: Editor Test – Riepilogo scelta

Avvio partita Tale test verifica che l’avvio di una partita avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. clicca il pulsante di conferma delle scelte effettuate;
2. verifica che sia stato effettuato il redirect all’editor (/editor).

```
...
@Test
public void startGame() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    Assert.assertEquals("Test fallito! L'avvio della partita non è avvenuto
        correttamente.", driver.getCurrentUrl(), urlPaginaDiRedirezione);
}
...
```

Figura 6.7: Editor Test – Avvio partita

Compilazione classe Tale test verifica che la compilazione della classe di test scritta dall’utente avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata;

2. clicca il pulsante di compilazione;
3. verifica che sia visibile il risultato della compilazione.

```
...
@Test
public void compile() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#sidebar-
textarea + div > * div.CodeMirror-code > *"), 1));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    driver.findElement(By.id("compileButton")).click();

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#console-
textarea + div > * div.CodeMirror-code > *"), 1));
    } catch(TimeoutException e) {
        Assert.fail();
    }
}
...
```

Figura 6.8: Editor test – Compilazione classe utente

Copertura classe Questo test mira a verificare che siano visibili i risultati di copertura, seguendo queste operazioni:

1. attende che la classe da testare sia caricata;
2. clicca il pulsante di copertura;
3. verifica che siano visibili i risultati della copertura.

```
...
@Test
public void coverage() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#sidebar-
textarea + div > * div.CodeMirror-code > *"), 1));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    driver.findElement(By.id("coverageButton")).click();

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#sidebar-
textarea + div > * div.CodeMirror-code > * .uncovered-line"), 0));
    } catch(TimeoutException e) {
        Assert.fail();
    }
}
...
```

Figura 6.9: Editor Test – Copertura classe utente

Submit della partita Tale test verifica che il tentativo dell'utente venga consegnato e che la partita venga processata correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata;
2. clicca il pulsante di submit;
3. verifica che sia visibile l'esito della partita.

```
...
@Test
public void run() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#sidebar-
textarea + div > * div.CodeMirror-code > *"), 1));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    driver.findElement(By.id("runButton")).click();

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#console-
textarea2 + div > * div.CodeMirror-code > *"), 1));
    } catch(TimeoutException e) {
        Assert.fail();
    }
}
...

```

Figura 6.10: Editor Test – Submit partita

Di seguito sono riportati i risultati dei test effettuati, 9 per quanto riguarda l’editor e 3 per il login.

```
[INFO] Results:
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 03:45 min

```


Capitolo 7

Sviluppi futuri

Qui vengono proposte idee ed osservazioni effettuate durante lo sviluppo del software, applicabili su implementazioni future dell'applicazione.

7.1 Rivedere attributi della tabella games del T4

Durante l'analisi del database fornito dal task T4 si è notata la presenza di alcuni attributi apparentemente senza senso per la tabella games, come i campi Descrizione e Nome. Sarebbe opportuno rivedere il significato di questi due attributi per legarli con il contesto attuale dell'applicazione. Tale criticità infatti è dovuta al fatto che il task T4 è tra i task originali, cioè quelli sviluppati in modo indipendente, senza tenere conto effettivamente del funzionamento generale

dell'applicazione e al workflow complessivo della partita.

7.2 Rivedere la struttura della api */save-data*

Come già detto, *save-data* è una api che si occupa di salvare i dati ricevuti inerenti alla partita. Tuttavia, la creazione dell'oggetto *Game* presente nel corpo della funzione, possiede due variabili che vengono impostate come placeholder e mai più trattate, nello specifico la variabile *description* ha "descrizione" e *name* ha "nome". Inoltre, è da chiarire il loro ruolo all'interno della tabella games nel contesto della partita.

Inoltre, è doveroso aprire una parentesi su come *save-data* possieda troppe responsabilità, e **non rispetta** quindi **un'appropriata separazione degli interessi**. Essa infatti gestisce sia la logica del controller (gestendo la richiesta HTTP) che la logica di business (gestendo la creazione e l'invio dei dati del gioco). Separare queste responsabilità renderebbe il codice più modulare, comprensibile e facilmente manutenibile. Un'idea di implementazione sarebbe spostare la logica di business espressa nella creazione dell'oggetto *Game* ed il suo salvataggio attraverso *gameDataWriter.saveGame* in una classe apposita chiamata *GameDataService*, in modo da poter sfruttare la **dependency injection** per gestire eventuali dipendenze necessarie per il corretto

funzionamento del service.

7.3 Implementare uno scenario di gioco multigiocatore

Introdurre un nuovo scenario di gioco all'applicazione giustificherebbe l'esistenza del concetto di round. Fino ad ora, il concetto di round è stato sempre equivalente a quello del game. Il requisito R8 introdotto da noi è già predisposto a questa modifica (basta semplicemente modificare il criterio del punteggio nella funzione `calcolaPunteggio`), mentre per quanto riguarda R1 la gestione degli ID del round e dei rispettivi turni associati ad un determinato round è robusta all'introduzione di più round in una singola partita.

7.4 Ridurre il carico dello storing dei dati sul browser

Durante lo sviluppo dell'applicazione, si è notato un abuso del `localStorage` all'interno di `editor.html`, memorizzando molte informazioni all'interno del browser durante il gioco. Sarebbe opportuno rivalutare quali variabili necessitino effettivamente di essere memorizzate nel `localStorage` e quali siano più opportune lasciare al model del backend.

7.5 Implementare un servizio separato per la scrittura sul filesystem

Una delle criticità emerse durante l'implementazione del requisito R1 è relativa al fatto che il filesystem non è esposto all'esterno attraverso un servizio di scrittura ad hoc per scriverci sopra (magari tramite API). Infatti, la scrittura avviene in maniera grezza: il microservizio che necessita di scrivere in questo volume lo fa direttamente lui (come T1 e T8 nel caso del volumeT8). Quindi, se nelle versioni successive dell'applicazione la struttura del filesystem venisse cambiata, tutti i componenti che attualmente scrivono sul filesystem dovrebbe essere modificati. Per risolvere questa criticità, dovrebbe esistere un servizio aggiuntivo che espone delle interfacce per operare sul filesystem, in modo tale da poter interagire con quest'ultimo senza dover necessariamente conoscere tutti i dettagli implementativi.

Questa proposta di modifica dimostra come la soluzione attuale possa creare delle dipendenze tra i vari task. Ma in realtà in un'architettura a microservizi si dovrebbe enfatizzare l'autonomia del microservizio, la sua elevata coesione interna, senza che questo debba aver bisogno di altri microservizi per poter funzionare correttamente.