

Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie

dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Documentazione A7

Software Architecture Design

Prof.ssa:

Anna Rita Fasolino

Candidati:

Rinaldi Simone M63/1654

Pezzullo Lorenza M63/1579

Imparato Giuseppe M63/

Ottaiano Giada M63/1640

Anno Accademico 2023–2024

Indice

Introduzione	1
1 Progettazione	2
1.1 Descrizione del Progetto	3
1.2 Descrizione dei Requisiti Assegnati	3
1.2.1 Requisito R1	4
1.2.2 Requisito R8	5
1.3 Stato Iniziale del Progetto	9
1.3.1 Architettura a microservizi	9
1.3.2 Punto di partenza	12
1.4 Architettura complessiva	14
1.4.1 Diagramma dei componenti complessivo	14
1.4.2 Diagramma dei casi d'uso	16
1.4.3 Workflow Diagram	17
2 Sviluppo	18
2.1 Daily Scrum e Sprint Backlog	19
2.1.1 Iterazioni	19
2.2 Strumenti e frameworks utilizzati	20
2.2.1 Microsoft Teams	20

2.2.2	Dropbox	21
2.2.3	OneNote	21
2.2.4	GitHub	21
2.2.5	Visual Paradigm	22
2.2.6	Postman	22
2.2.7	Selenium	22
2.2.8	DBeaver	23
2.3	Ambiente di sviluppo	23
3	Requisito R1	25
3.1	Analisi della Funzionalità di Misurazione Utente . . .	28
3.1.1	Struttura definitiva delle directory	31
3.2	Correzione criticità nel codice	31
3.2.1	Struttura del database T4	32
3.3	Modifiche effettuate sul task T4	33
3.4	Modifiche effettuate sul task T6	35
3.5	Modifiche effettuate sul task T5	36
3.6	Modifiche effettuate sul task T8	42
3.6.1	Refactoring T8	46
4	Requisito R8	47
4.1	Modifiche effettuate sul task T4	47
4.2	Modifiche effettuate sul task T6	49
4.2.1	Calcolo del Punteggio	50
4.3	Modifiche effettuate sul task T5	51
4.4	Aggiornamenti post implementazione dei Requisiti R1 ed R8	52

5	Requisiti Aggiuntivi	55
5.1	Coerenza tra il Database del T4 e il Volume T8	55
5.1.1	Gestione tasto "Run"	56
5.1.2	Gestione tasto "Play/Submit"	57
5.2	Miglioramenti generali	59
5.2.1	Storico	59
5.2.2	Game Info	59
5.3	Nuova modalità e API	60
5.3.1	Aggiunta schermata /gamemode	61
5.3.2	Modalità Allenamento	62
5.3.3	Aggiunta API per il T8	64
5.4	Workflow definitivo	67
6	Deployment	68
6.1	Integrazione con la nuova versione	70
6.1.1	Analisi dei file della nuova versione	70
6.1.2	Corretta gestione degli URL	70
6.1.3	Problematica nella lettura della misurazione dell'utente	71
6.1.4	Aggiornamento classifica studenti	72
7	Testing	73
7.1	Testing delle API con Postman	73
7.2	Verifica della coerenza dei dati tramite DBeaver	75
7.3	Testing automatizzato con Selenium	76
8	Sviluppi futuri	78

8.1	Miglioramento della sicurezza delle pagine web	78
8.2	Miglioramento della robustezza	79
8.3	User Friendliness	79
8.4	Revisione database T4	81
8.5	Diverse modalità di gioco	81
8.6	Consentire al giocatore di riprendere una partita già avviata	82
8.7	Gestire correttamente l'aggiunta e la rimozione delle classi da testare	83
9	Guida all'installazione	84
9.1	Docker e WSL	84
9.2	Installazione	85
9.3	Guida all'utilizzo	87
9.3.1	Registrazione Admin	87
9.3.2	Registrazione utente	87
9.4	Modificare il codice	88
9.5	Problematiche di utilizzo	89
9.5.1	Browser Cache	89
9.5.2	Versione Docker	89
9.5.3	Porte già in uso	90
9.5.4	502 Bad Gateway	91
9.6	Video Funzionamento Web App	92
9.6.1	Funzionamento modalità "Sfida un Robot"	92
9.6.2	Funzionamento modalità "Allenamento"	92
	Glossario	93

Introduzione

Il progetto ENACTE-ST si prefigge di sottolineare l'importanza del testing nell'ambito dello sviluppo e implementazione di applicazioni web. Infatti, esso è cruciale per individuare e correggere vulnerabilità che potrebbero non essere emerse durante le fasi iniziali dello sviluppo. In particolare, l'ENACTE-ST si concentra sulla realizzazione di un Educational Game, un progetto nel quale i partecipanti sono invitati a sfidare un software di generazione automatica di test con l'obiettivo di coprire in modo esaustivo i possibili casi. Esso rende in questo modo l'apprendimento verso questa disciplina più stimolante grazie ad un approccio pratico.

Capitolo 1

Progettazione

Il seguente capitolo fornisce una panoramica dettagliata degli obiettivi, delle dinamiche e della struttura del progetto in esame. Prima di procedere con l'implementazione dei requisiti assegnati, è stato necessario condurre uno studio approfondito sulla richiesta di ciascun requisito e sulle modifiche che tale implementazione avrebbe comportato. Tale approccio è stato adottato al fine di limitare le modifiche esclusivamente a quelle essenziali, al fine di evitare perturbazioni significative nel lavoro iniziale. Pertanto, la fase di implementazione effettiva rappresenta la continuazione di un'analisi accurata e di una progettazione dettagliata per ciascun requisito.

1.1 Descrizione del Progetto

Il gioco si basa sulla sfida degli studenti con strumenti di testing automatizzati, in grado di generare automaticamente casi di test JUnit, come EvoSuite e Randoop. L'obiettivo è raggiungere un determinato livello di copertura, come ad esempio la copertura delle linee di codice. Ogni gruppo è incaricato di implementare requisiti funzionali assegnati, suddivisi in diversi task. È disponibile una documentazione specifica per ciascun requisito, consultabile al seguente indirizzo: [Testing-Game-SAD-2024](#).

Considerate le difficoltà riscontrate da alcuni gruppi nel tentativo di effettuare l'installazione, è stata predisposta una breve guida disponibile nel capitolo dedicato all'installazione in cui vengono riportate anche delle possibili soluzioni a problematiche note.

1.2 Descrizione dei Requisiti Assegnati

Sono stati assegnati due requisiti, uno con alta priorità e uno correlato al primo scenario di gioco con priorità media-alta. Data questa differenza di priorità, si è deciso di implementare prima il requisito R1 e lasciare il requisito R8 alle iterazioni successive.

1.2.1 Requisito R1

Il requisito R1 è un requisito ad alta priorità che impatta sul database del T4 e sul T5. Esso mira a rivedere l'organizzazione del File System condiviso tra i volumi T8 e T9. Idealmente, si propone di consolidare l'intero sistema in un singolo volume con la seguente struttura consigliata.

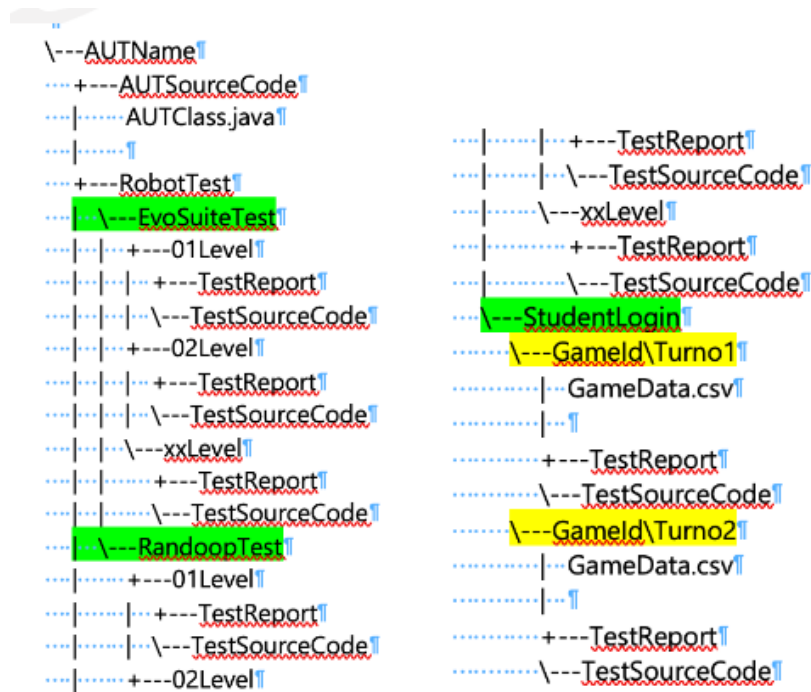


Figura 1.1: Struttura consigliata

La soluzione suggerita prevede di salvare i dati del turno XX, inclusi i risultati dei test (GameData.csv e TestReport) e il codice dei test, sotto la directory **GameId\TurnoXX**. Tuttavia, si consiglia di valutare attentamente la possibilità di non fondere i due volumi o di fonderli in un secondo momento nel caso in cui fosse necessario che i due robot lavorino in maniera indipendente sul File System.

Successivamente, esamineremo in dettaglio la decisione di fondere o meno i due volumi.

1.2.2 Requisito R8

Il requisito R8, di priorità media-alta, riguarda il calcolo del punteggio di una partita. In particolare, al termine di una partita è necessario prevedere un servizio per assegnare un punteggio al giocatore. Inizialmente, si può adottare 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:

- $Punteggio_vittoria = 1 + LOC\%$.
- $Punteggio_sconfitta = -1$.
- $Punteggio_pareggio = 0$.

Questa funzione di calcolo del punteggio dovrebbe essere separata in un servizio distinto, che possa essere anche configurabile ed in futuro estensibile.

Il punteggio della partita dovrebbe essere salvato nel database delle partite e attribuito al punteggio complessivo del giocatore per consentire la creazione di una classifica dei vari giocatori.

User Stories

Di seguito, forniamo la descrizione del requisito R8 attraverso l'utilizzo del paradigma SCRUM, formulato come segue: "**COME UN** [utente], **DESIDERO** [desiderio], **IN MODO CHE** [beneficio]". Inoltre, presentiamo le user stories preesistenti, le quali sono consultabili nella documentazione precedente.

Le User Stories descrivono parti di funzionalità desiderate dal punto di vista dell'attore coinvolto. Seguono il paradigma INVEST:

- **Indipendenti:** Devono essere indipendenti per evitare stime errate e vincoli.
- **Negoziabili:** I dettagli sono negoziati tra il cliente, il Product Owner e il team.
- **Di Valore:** È essenziale interrogarsi sul vero beneficiario della User Story per evitare di dedicare tempo a storie prive di valore.
- **Stimabili:** È possibile avviare un'analisi per stimare le attività coinvolte.

- **Della Giusta Dimensione:** È opportuno garantire che le storie non siano troppo grandi, ma al tempo stesso combinare le storie più piccole.
- **Testabili:** Devono includere criteri di accettazione misurabili attraverso dei test.



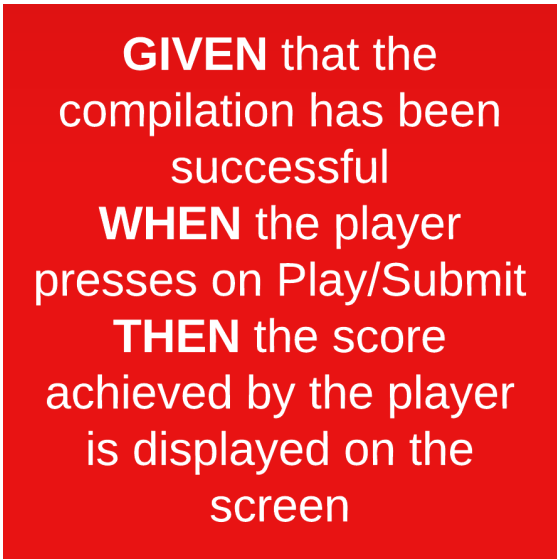
Figura 1.2: User Stories

Esse seguono il paradigma delle 3C:

- **Card:** Ogni User Story viene rappresentata da una carta che descrive in modo conciso la funzionalità o il requisito del sistema dal punto di vista dell'utente.
- **Conversation:** Rappresenta le discussioni e le conversazioni che si svolgono tra i membri del team di sviluppo e gli stakeholder per chiarire i dettagli e i requisiti specifici della user story.
- **Confirmation:** La conferma fornisce una guida chiara per il team di sviluppo su cosa testare e come verificare che la user story soddisfi i requisiti.

Criteri di accettazione

I criteri di accettazione utilizzano il modello "**GIVEN** [stato iniziale], **WHEN** [azione o evento], **THEN** [risultato atteso o comportamento desiderato]". Questo fornisce indicazioni chiare al team di sviluppo su cosa deve essere implementato per soddisfare la user story con successo.



GIVEN that the
compilation has been
successful
WHEN the player
presses on Play/Submit
THEN the score
achieved by the player
is displayed on the
screen

Figura 1.3: Criterio di Accettazione

1.3 Stato Iniziale del Progetto

Si procede con l'illustrare lo stato iniziale del progetto da cui si è partiti, allo scopo di offrire una comprensione dettagliata della struttura iniziale. Il punto di partenza assume un ruolo cruciale nel contesto dell'evoluzione architetturale. Attraverso questa enfattizzazione del punto di partenza, si fornisce al lettore una chiara visione della struttura originaria del progetto. Questa descrizione dettagliata consente di contestualizzare in modo completo le trasformazioni successive, offrendo una prospettiva chiara sulle decisioni prese e sulle modifiche apportate durante lo sviluppo del progetto.

1.3.1 Architettura a microservizi

Il progetto si basa su un'architettura a microservizi. In generale, le architetture a microservizi affrontano problematiche relative la veloce

risposta alle richieste del mercato decomponendo funzionalmente un dominio aziendale in microservizi, ossia servizi che gestiscono una sola responsabilità. Un servizio è una mini applicazione che implementa funzionalità strettamente focalizzate, come la gestione dei clienti e così via. Ciò che importa è che ogni servizio abbia un insieme di responsabilità focalizzate e coese, ma al tempo stesso il servizio deve essere fortemente indipendente da tutti gli altri. Ci sono molteplici **vantaggi** in un'architettura a microservizi:

- permette la consegna e il deployment continuo di applicazioni complesse;
- i servizi sono piccoli e facili da mantenere;
- i servizi sono deployabili in maniera indipendente;
- i servizi sono scalabili in maniera indipendente;
- ogni team sviluppa, testa e distribuisce i propri servizi in modo indipendente;
- permette sperimentazioni e l'adozione di nuove tecnologie;
- ha una migliore fault isolation.

Dalla documentazione del gruppo di integrazione è fornita una rappresentazione dell'architettura a microservizi della web application, che viene riportata di seguito. In particolare, si evidenziano in giallo i task che sono stati modificati per il completamento dei requisiti.

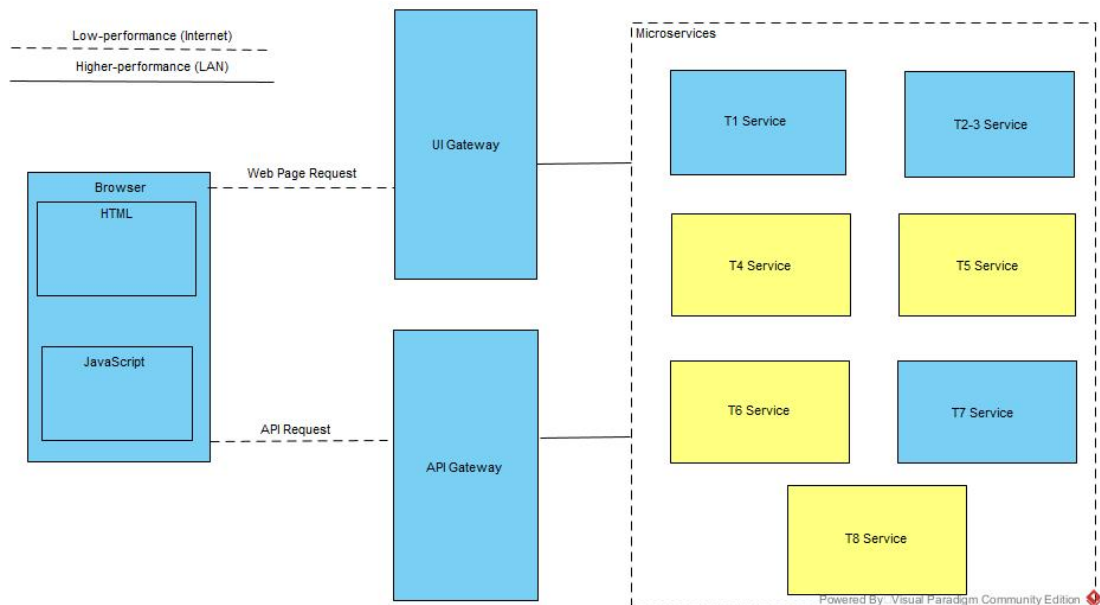


Figura 1.4: Architettura a microservizi

L'**UI Gateway** ha il compito di effettuare il routing delle richieste http relative al frontend. L'**API Gateway** si occupa del routing e della gestione dell'autenticazione e autorizzazione per le richieste relative alle API del sistema.

1.3.2 Punto di partenza

In questa documentazione è descritto il lavoro svolto partendo dal progetto **T11-G41** per poi integrare le varie modifiche effettuate nella versione **A10-2024** più recente. I colleghi da cui abbiamo ereditato il progetto avevano lavorato sulla generazione dei livelli e sulla misurazione della copertura dell'utente. Allo stato iniziale, il sistema fornitoci presentava la seguente struttura di componenti:

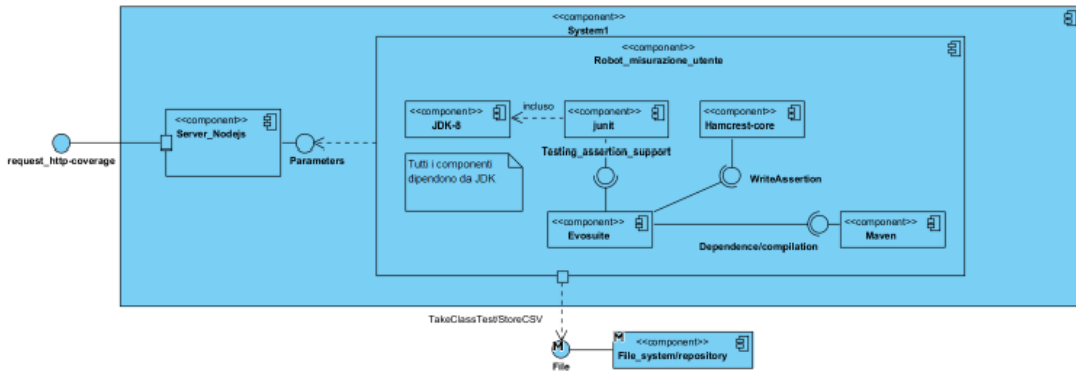


Figura 1.5: Diagramma dei componenti misurazione utente

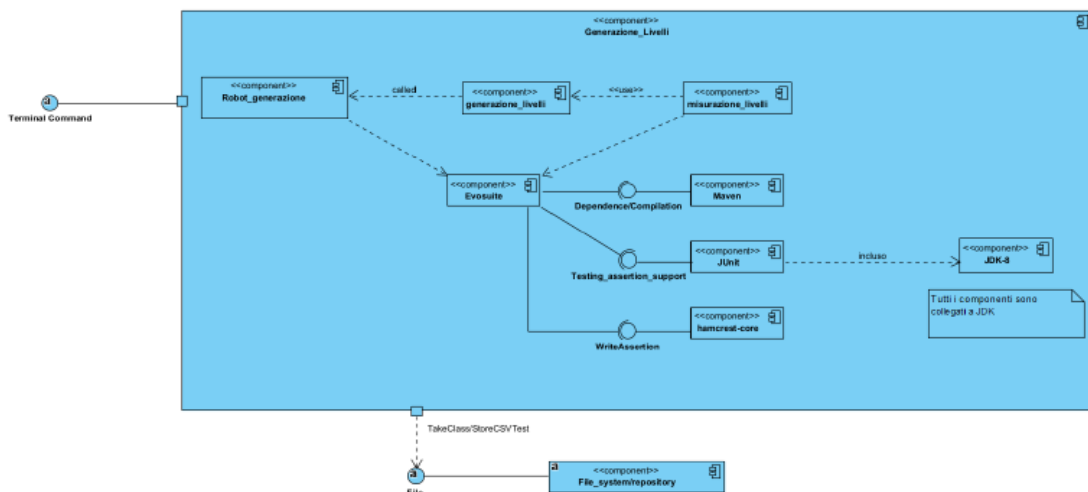


Figura 1.6: Diagramma dei componenti generazione livelli

N.B. Ulteriori dettagli sono consultabili nella documentazione del T8 della repository di partenza **T11-G41**.

Il codice è stato oggetto di un'analisi approfondita e di un processo di refactoring al fine di migliorarne la comprensibilità e la manutenibilità. Esso, infatti, risultava poco chiaro a causa della mancanza di commenti descrittivi, oltre alla presenza di variabili con nomi ambigui e poco esplicativi e alla presenza di una serie di cartelle apparentemente inutilizzate. Sono state apportate le seguenti modifiche:

- **Commenti descrittivi:** sono stati aggiunti commenti descrittivi per spiegare chiaramente ogni sezione di codice, facilitando la comprensione del funzionamento complessivo del programma.
- **Nomi esplicativi per le variabili:** le variabili con nomi ambigui o poco chiari sono state rinominate in modo più descrittivo. Questo rende il codice più leggibile e comprensibile per chiunque ne prenda visione in futuro.

Inoltre, è stato chiarito che le 13 cartelle vuote presenti nel T8 vengono generate e utilizzate temporaneamente durante il processo di creazione dei livelli e rimangono in disuso successivamente.

Queste modifiche hanno lo scopo di rendere il codice più leggibile, comprensibile e manutenibile, facilitando il lavoro sia per chi attualmente lavora sul progetto che per chi dovrà gestirlo in futuro.

È possibile consultare le modifiche nella sezione dedicata.

1.4 Architettura complessiva

Si procede con una descrizione generale dell'architettura nella sua interezza, mediante un diagramma dei casi d'uso e il diagramma dei componenti.

1.4.1 Diagramma dei componenti complessivo

Il diagramma dei componenti dell'architettura attuale è riportato di seguito, evidenziando in giallo i componenti che hanno subito delle modifiche:

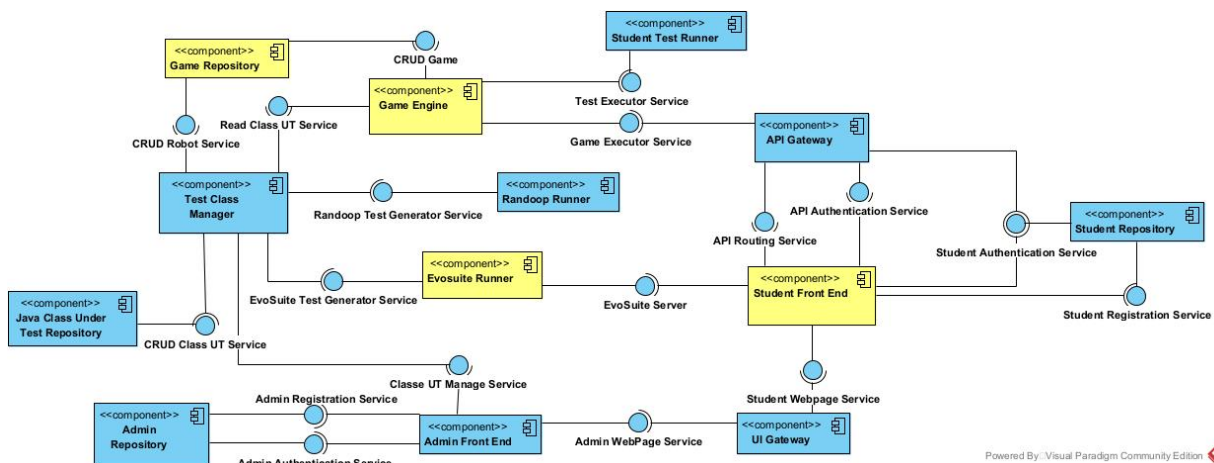


Figura 1.7: Diagramma dei componenti

Di seguito è fornita una spiegazione dei componenti modificati.

Student Front End

Questo componente fornisce un'interfaccia all'utente "Registered Student", dalla registrazione fino alle schermate di gioco.

EvoSuite Runner

Questo componente genera il robot EvoSuite e restituisce le statistiche di copertura del codice. È possibile osservare che comunica con il componente Test Class Manager per generare la copertura tramite Evosuite, mentre è utilizzato dallo Student Front End per calcolare la copertura del codice del test dello studente.

Game Engine

Questo componente gestisce la partita, utilizzando i servizi offerti dallo Student Test Runner e dal Game Repository per un corretto svolgimento della partita. In particolare, ottiene i risultati della compilazione e della copertura del test dello studente, i risultati dei robot e salva le informazioni della partita in corso. Ogni Game offre la possibilità di svolgere molteplici turni fino a quando non viene superata la percentuale di copertura del test scritto dal robot.

Game Repository

Ogni submit viene salvato in un repository interno al T4, dedicato al salvataggio delle partite. Questo consente di mantenere uno storico

dei risultati delle coperture dei test ottenuti nei diversi turni.

1.4.2 Diagramma dei casi d'uso

Il diagramma dei casi d'uso è riportato in seguito:

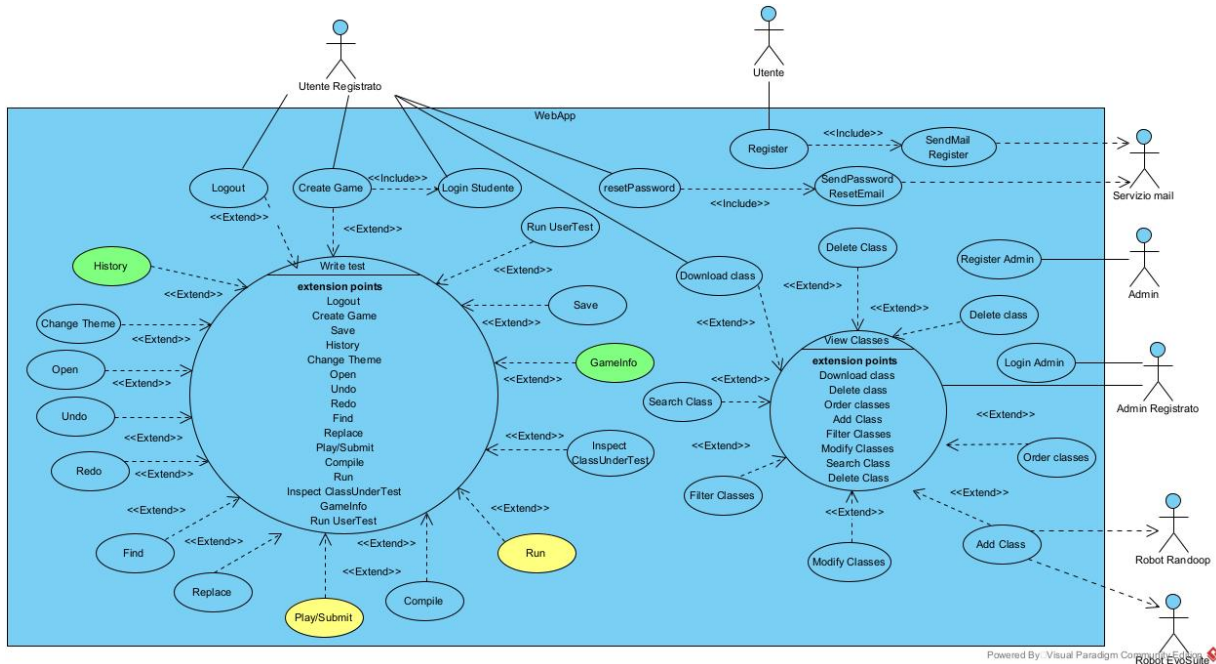


Figura 1.8: Diagramma dei casi d'uso

Si evidenziano in giallo i casi d'uso modificati relativamente ai requisiti R1 ed R8, mentre in verde i casi d'uso modificati relativamente a requisiti aggiuntivi per un miglior funzionamento dell'applicazione.

Le modifiche apportate sono dettagliatamente illustrate nei successivi paragrafi.

1.4.3 Workflow Diagram

Al fine di delineare il contesto iniziale del gioco, viene presentato un diagramma di flusso di alto livello. Inizialmente, il giocatore esegue il login, quindi seleziona la classe da testare e il robot da sfidare. Successivamente, dopo aver confermato la scelta, procede effettivamente alla scrittura del test della classe nell'editor.

Questa modalità di gioco implica la compilazione del test e, in caso di esito positivo della stessa, il calcolo del punteggio utente. Tuttavia, viene concessa all'utente la possibilità di riscrivere il test e tentare di battere il robot anche se subisce una sconfitta nella stessa partita.

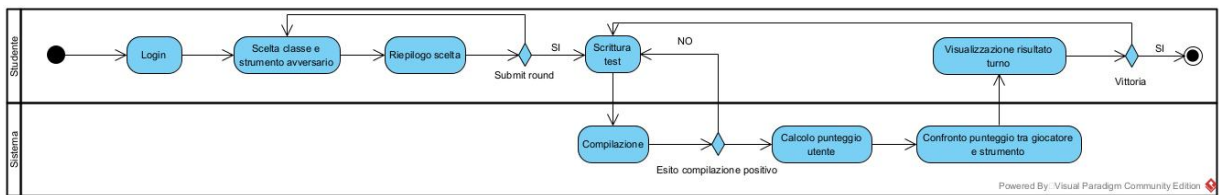


Figura 1.9: Workflow Diagram

Quest'ultimo meccanismo ha subito delle modifiche in questa versione del gioco, in quanto, dal punto di vista della gamification, si è ritenuto più opportuno non concedere al giocatore la possibilità di riscrivere il test in caso di sconfitta. Sarà fornita di seguito una descrizione aggiornata del flusso di gioco.

Capitolo 2

Sviluppo

Si è adottata la **metodologia AGILE** basata su **Sprint Backlog** e **Daily SCRUM**. In particolare, si è proceduto dall'identificazione dello Sprint Backlog. La metodologia di sviluppo AGILE permette di focalizzarsi sul codice anziché sul design e sulla documentazione, adattandosi prontamente ai cambiamenti dei requisiti e rilasciando tempestivamente software funzionante ai clienti. SCRUM rappresenta un framework agile per la gestione del ciclo di sviluppo del software, iterativo e incrementale, all'interno del quale è possibile impiegare molteplici processi e tecniche. In particolare, lo sprint backlog consiste in un piano dettagliato di sviluppo per ogni sprint, ciascuno dei quali presenta un obiettivo da realizzare; mentre il Daily Scrum è un evento della durata di 15 minuti, durante il quale il team di sviluppo sincronizza le proprie attività e pianifica il lavoro delle successive 24 ore.

2.1 Daily Scrum e Sprint Backlog

La principale sfida è stata l'analisi del problema da affrontare. A tal fine, sono state organizzate diverse riunioni, sia in presenza che da remoto, al fine di discutere delle problematiche riscontrate e di pianificare l'intero lavoro necessario.

Le riunioni sono state fondamentali per definire gli Sprint Backlog, nei quali sono stati assegnati compiti ai membri del team e sono state stabilite scadenze da rispettare. In particolare, sebbene una parte del lavoro sia stata eseguita tra novembre e dicembre, la maggior parte del progetto è stata realizzata tra gennaio e febbraio, con scadenze fissate ogni due settimane. Pertanto, lo sforzo è stato distribuito su un periodo di tempo sufficientemente esteso.

2.1.1 Iterazioni

Le attività sono state eseguite durante 6 iterazioni:

- **Iterazione 1:** Si è proceduto con lo studio della documentazione del T8 e dei tool da utilizzare, con il brainstorming dei requisiti del software e la discussione dell'architettura;
- **Iterazione 2:** È stato effettuato lo studio dei volumi T8-T9, l'individuazione delle criticità relative alla struttura delle directories (*ndr.* discrepanza degli ID) e la progettazione della struttura definitiva. Si è presa la decisione sull'unione dei volumi.

- **Iterazione 3:** Si è proceduto con lo studio approfondito della documentazione T4, T5 e T6; sono state apportate modifiche al database del T4 e si è studiato l'utilizzo delle API di T5-T6.
- **Iterazione 4:** Si è proceduto con la correzione delle API di T5-T6 e con l'implementazione della struttura scelta delle directories nel volume T8.
- **Iterazione 5:** Si è proceduto con l'implementazione del file GameData.csv e con l'implementazione di un servizio espandibile per il calcolo del punteggio.
- **Iterazione 6:** Si è proceduto ad effettuare un ultimo refactoring e l'integrazione sulla nuova versione dell'applicazione.

2.2 Strumenti e frameworks utilizzati

2.2.1 Microsoft Teams

Microsoft Teams è stato impiegato per la comunicazione da remoto tra i membri del team, sia per condurre videochiamate al fine di discutere dettagliatamente degli step da eseguire, sia per lo scambio rapido dei messaggi.

2.2.2 Dropbox

Dropbox è una piattaforma che offre un servizio di cloud storage e sincronizzazione automatica dei file, il quale è stato sfruttato per condividere in maniera immediata ed efficiente i file necessari tra tutti i membri del gruppo.

2.2.3 OneNote

OneNote è stato utilizzato durante le videochiamate effettuate per la creazione di lavagne e note facilmente condivisibili, sia come file pdf (nonché altre estensioni), sia tramite la condivisione di un unico blocco note utilizzato dall'intero gruppo. Questo strumento consente di mantenere tutto il lavoro organizzato con la divisione in sezioni e pagine, offrendo anche una funzionalità che permette la visualizzazione delle versioni precedenti del lavoro effettuato.

2.2.4 GitHub

GitHub è una piattaforma di sviluppo collaborativo, che abbiamo utilizzato per tenere traccia delle versioni del codice sorgente e per agevolare la collaborazione tra i membri del gruppo per lavorare contemporaneamente. Esso, infatti, permette l'utilizzo di un repository che contiene file e la cronologia delle revisioni del progetto.

2.2.5 Visual Paradigm

Visual Paradigm è uno strumento di modellazione e creazione di diagrammi per la collaborazione agile di team. Esso è stato impiegato per la creazione dei diagrammi UML, necessari nella documentazione.

2.2.6 Postman

Postman è un'applicazione impiegata per la gestione e il testing delle API. Nello specifico, offre la capacità di inviare richieste HTTP a un server e di ricevere le relative risposte attraverso un'interfaccia utente intuitiva. Gli utenti hanno la possibilità di creare collezioni di richieste e ambienti per automatizzare i processi di sviluppo delle API, oltre a poterle testare attraverso la verifica delle risposte. Inoltre, Postman consente di generare automaticamente una documentazione dettagliata per ciascuna API e di monitorarne le prestazioni nel tempo per individuare eventuali problemi di latenza.

2.2.7 Selenium

Selenium costituisce un tool open source impiegato per la gestione automatizzata dei browser e si configura come un framework di testing. È essenzialmente composto da una suite di strumenti, tra i quali figurano Selenium IDE, Selenium Builder, Selenium Grid e Selenium WebDriver. In linea generale, tale risorsa agevola l'automazione delle interazioni dell'utente con il browser, consentendo l'esecuzione efficien-

te e ripetibile di test funzionali su applicazioni web. Nello specifico, Selenium consente la simulazione di azioni quali il clic su elementi, l'inserimento di testo, l'invio di moduli, eccetera. Ulteriormente, offre un supporto multi-piattaforma in quanto compatibile con diversi browser web, quali Chrome, Firefox, e altri, oltre a essere utilizzabile su vari sistemi operativi. La sua integrazione con diversi framework, come JUnit o NUnit, e il sostegno a numerosi linguaggi di programmazione, inclusi Java e JavaScript, ne ampliano la versatilità. Particolarmente rilevante è la funzionalità di registrazione e riproduzione degli scenari di test, che consente di registrare le azioni dell'utente e di riprodurle automaticamente durante la fase di testing.

2.2.8 DBeaver

DBeaver è un'applicazione software client SQL e uno strumento di amministrazione di database. Si tratta di un software open-source che offre un'interfaccia grafica per l'interazione con i database tramite diverse tecnologie di connessione e include il supporto per l'editor SQL. È stato impiegato per stabilire una connessione al database del T4.

2.3 Ambiente di sviluppo

Per la scrittura e compilazione del codice è stato impiegato **Visual Studio Code**, già utilizzato in altre occasioni, poiché esso è estrema-

mente versatile, permettendo l'aggiunta di varie estensioni, ed è veloce e leggero. In particolare, esso:

- offre un'**integrazione con GitHub**, il che permette di effettuare operazioni di push direttamente dall'editor in modo rapido e facile;
- fornisce un **supporto multi-linguaggio**, il che permette di sfruttare un'ampia gamma di linguaggi di programmazione (nel nostro caso, principalmente JavaScript, Java e Go).

Tra le utilissime estensioni di VSCode, un ruolo particolarmente rilevante è ricoperto dall'estensione Docker: l'estensione Docker semplifica la creazione, la gestione e il rilascio di applicazioni containerizzate direttamente da Visual Studio Code. Fornisce, inoltre, il debugging con un solo clic per Node.js, Python e .NET all'interno di un container, nel caso ne fosse necessario l'utilizzo.

Capitolo 3

Requisito R1

Per il soddisfacimento del requisito R1, si è preliminarmente proceduto con l'analisi della struttura iniziale del volume T8 che, come evidenziato nella figura 3.1, presenta una serie di problematiche, oltre al fatto che non rispecchiava la configurazione ideale richiesta.

Criticità relativa agli ID

Sono state immediatamente rilevate delle criticità relative agli ID delle partite e del turno. Infatti, le directories sono strutturate in maniera tale da memorizzare soltanto l'ID del turno giocato che coincide con quello della partita. Più nello specifico, in questa fase iniziale i valori di gameID, roundID e turnID coincidono, il che è errato.

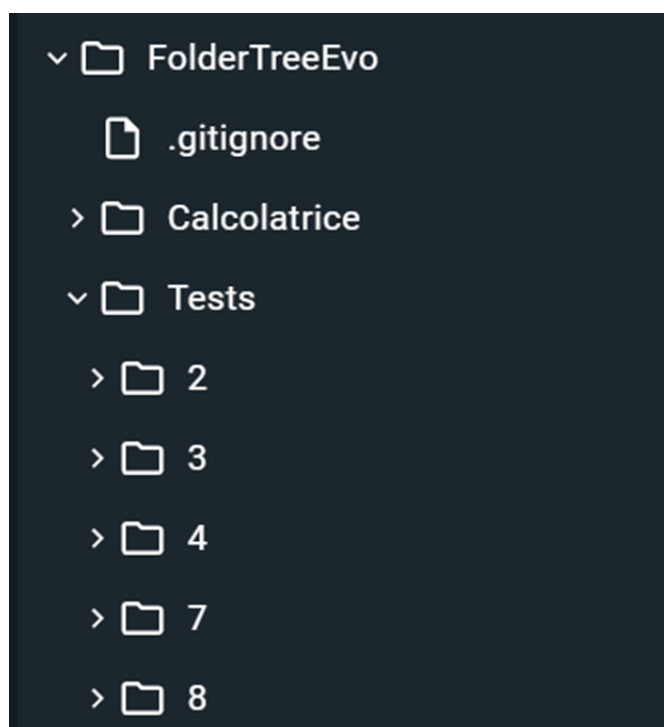


Figura 3.1: Struttura iniziale del volume T8

Unificazione dei volumi T8 e T9

Successivamente, l'attenzione è stata spostata sulla necessità di unificare i volumi T8 e T9. La decisione relativa a questa unificazione è stata oggetto della prima revisione, durante la quale sono state presentate due ragioni per le quali, allo stato attuale, non è utile né necessario combinare i suddetti volumi. Questi motivi sono i seguenti:

- L'unico requisito che potrebbe richiedere l'accesso simultaneo ai due volumi non è ancora contemplato nel contesto attuale del progetto ed è legato alla possibilità di consentire al giocatore di sfidare entrambi i robot, Evosuite e Randoop, durante la stessa partita;

- Da un punto di vista architetturale orientato ai microservizi, risulta più coeso mantenere separati i due volumi al momento, poiché ciò comporta vantaggi in termini di modularità e indipendenza dei moduli, senza apportare effettivi svantaggi;
- Dal punto di vista della sicurezza, mantenere i volumi separati sarebbe preferibile per conservare l'isolamento dei dati generati da Evosuite e Randoop;
- Infine, mantenere i volumi separati garantirebbe una maggiore facilità di manutenzione e una maggiore libertà per effettuare l'eventuale backup dei singoli volumi.

Si auspica che la presente decisione possa facilitare una maggiore modularità del sistema, consentendo una gestione più agevole delle evoluzioni future del progetto. Va notato che la separazione o unificazione dei volumi non impatta significativamente sulla velocità di esecuzione del programma, pertanto, tale decisione non rappresenterebbe una possibilità di ottimizzazione in termini di performance. Questa considerazione è stata inclusa nella valutazione complessiva della decisione, mirando a garantire che le scelte di progettazione siano orientate alla miglior struttura del sistema senza comprometterne l'efficienza operativa.

3.1 Analisi della Funzionalità di Misurazione Utente

Abbiamo avviato l'analisi a partire dal seguente diagramma di sequenza che descrive la funzionalità *misurazione utente*, come presentato nella documentazione redatta dai nostri predecessori, mostrato nella figura 3.2.

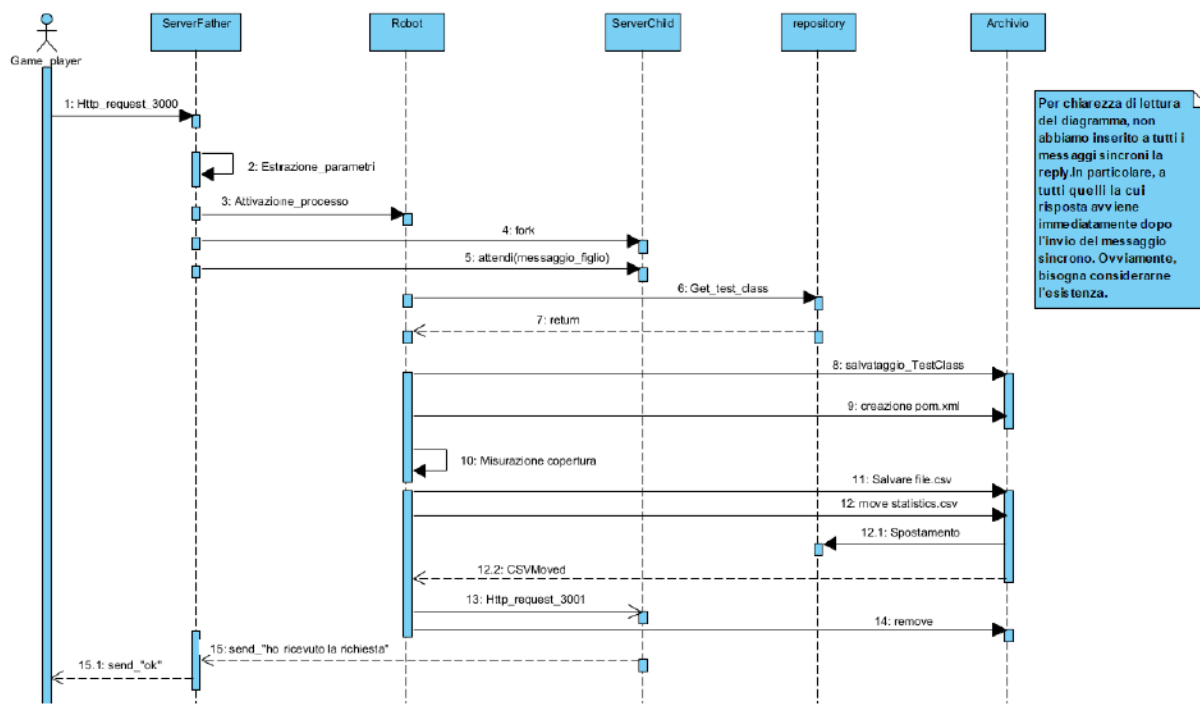


Figura 3.2: Diagramma di sequenza misurazione utente

Il diagramma evidenzia chiaramente le interazioni tra gli attori coinvolti nella misurazione dell'utente e le operazioni eseguite. L'analisi di questo diagramma ha fornito una comprensione dettagliata del flusso di lavoro e delle dipendenze tra i vari componenti.

Un server Node.js è in ascolto sulla porta 3000, pronto a ricevere richieste. L'API coinvolta in questo processo è `/api/`, la quale richiede il passaggio di due percorsi e il file `.java` contenente il test scritto dall'utente. I percorsi necessari sono il percorso della classe sulla quale effettuare la misurazione e il percorso di salvataggio del test scritto dall'utente. Una volta ricevuta la richiesta, viene attivato un server figlio in ascolto sulla porta 3001, il quale si occupa esclusivamente di gestire il termine delle varie operazioni.

Il server padre gestisce i parametri della richiesta e avvia uno script "*robot misurazione utente*", il quale esegue i vari test tramite EvoSuite e genera un file `.csv` contenente tutte le informazioni relative alla copertura del test scritto dall'utente. Questo file viene successivamente salvato in una directory apposita.

In seguito, abbiamo esaminato il diagramma di deployment fornito nella precedente documentazione mostrato nella figura 3.3, evidenziando in rosso i componenti su cui abbiamo concentrato il nostro lavoro per soddisfare il requisito R1.

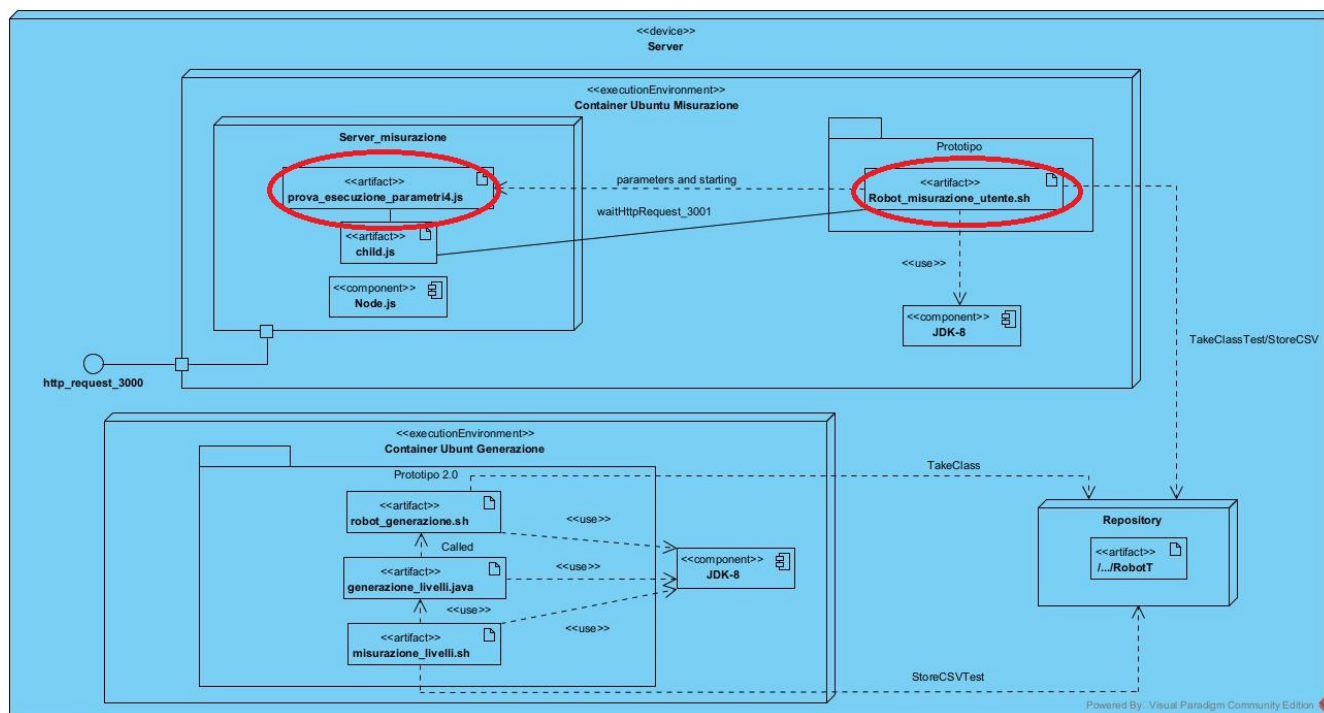


Figura 3.3: Diagramma di deployment

La visualizzazione della distribuzione fisica dei componenti del sistema attraverso il diagramma di deployment fornisce un'organizzazione e collocazione chiara delle diverse parti del sistema su hardware e infrastrutture specifiche. Questa rappresentazione offre una visione dettagliata delle relazioni tra i componenti hardware e software, consentendo una comprensione approfondita della configurazione fisica e della connessione tra le diverse entità del sistema.

Il processo di analisi ha fornito una visione chiara della funzionalità di misurazione utente, garantendo la base necessaria per l'implementazione e l'integrazione dei cambiamenti richiesti.

3.1.1 Struttura definitiva delle directory

Dopo un'attenta valutazione, basata sull'idea di lasciare quanto più spazio di manovra sufficiente per modifiche future senza alterare eccessivamente la struttura già presente, la soluzione finale segue la seguente struttura:



Figura 3.4: Struttura directory definitiva volume T8

3.2 Correzione criticità nel codice

Innanzitutto, per la corretta implementazione del requisito R1 è stato necessario risolvere la criticità relativa ai valori degli ID. Per questa ragione si è proceduto con la seguente analisi del database di T4.

3.2.1 Struttura del database T4

Di seguito viene presentato il diagramma ER per descrivere le entità consultabile nella documentazione di T4. Si mettono in evidenza le tabelle modificate, specificando che ciascuna entità contiene due campi per registrare la data di creazione e modifica (**CreatedAt** e **UpdatedAt**). Inoltre, si sottolinea che le entità che richiedono il tracciamento dello stato di apertura e chiusura presentano due attributi distinti (**StartedAt** e **ClosedAt**).

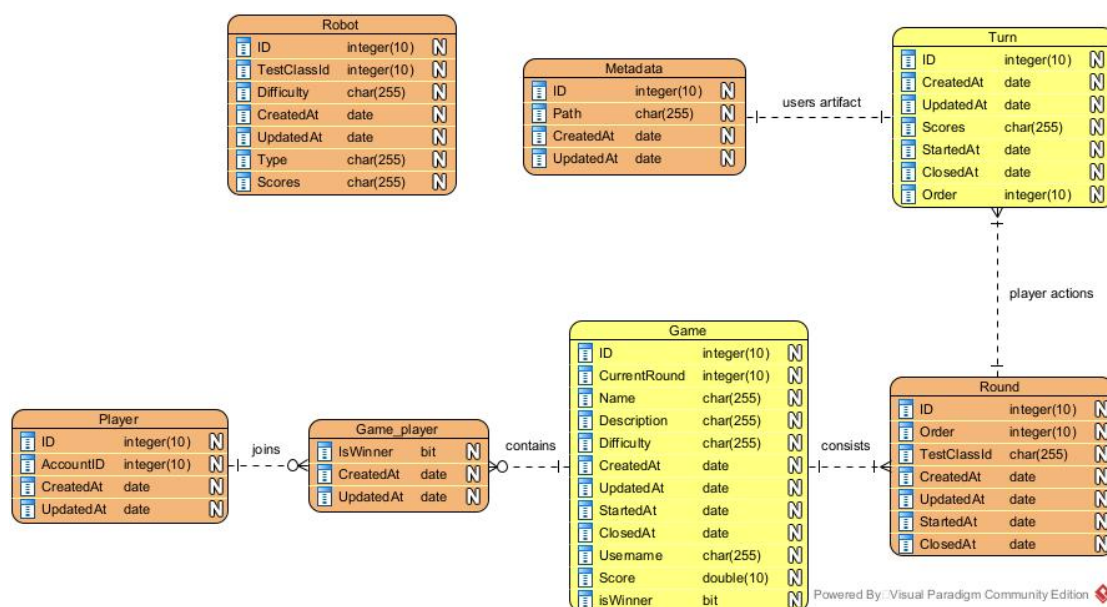


Figura 3.5: Diagramma ER

Gli attributi aggiunti sono i seguenti: *Order* come integer(10) nella tabella **Turn** e *Username*, *Score* come char(255) e double(10) rispettivamente nella tabella **Game**.

In particolare:

- l'attributo **Order** è stato aggiunto al fine di mantenere traccia dell'ordine dei turni all'interno di ciascuna partita.
- l'attributo **Username** è stato introdotto allo scopo di ottenere un'associazione tra la partita e il giocatore che l'ha disputata.
- l'attributo **Score** è stato introdotto per tenere traccia del punteggio ottenuto dall'utente al termine della partita.

3.3 Modifiche effettuate sul task T4

La prima esigenza consisteva nell'associare più turni allo stesso round, pertanto è stata analizzata la tabella contenente i dati dei turni. La colonna RoundID presentava il vincolo UNIQUE, rendendo necessaria la sua rimozione. Inoltre, è stato necessario rimuovere l'indice *idx_playerturn* sempre in corrispondenza della colonna RoundID poiché impediva la creazione di più turni con gli stessi valori della coppia di *roundID* e *playerID*.

Il campo *Order*, aggiunto in questa fase, viene aggiornato tramite l'API */turns*, la quale è stata opportunamente modificata. In particolare, sono state apportate diverse modifiche a seguito della decisione di aggiungere l'attributo *Order*:

- è stata modificata la struct **Turn** nel file *turn.go* per includere l'attributo *Order*;

- analogamente, è stata apportata la stessa modifica alla struct **CreateRequest**, anch'essa contenuta in *turn.go*;

Entrambe le modifiche sono illustrate in figura 3.6.

```
type Turn struct {
    ID        int64    `json:"id"`
    Order     int      `json:"order"`
    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"`
}

Codeium: Explain
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 3.6: Modifiche struct *Turn* e *CreateRequest*

Successivamente, è stata adattata la funzione **fromModel** in *turn.go*, la quale converte un oggetto *model.Turn* in un oggetto **Turn**, il primo destinato al database e il secondo per l'interazione con l'API. Infine, è stato aggiunto l'attributo *Order* anche alla funzione **CreateBulk** presente in *service.go*. Questa funzione è responsabile della creazione di nuovi turni per ciascun *PlayerID*.

Infine, seguendo una logica simile all'introduzione dell'attributo *Order* nella struct **Turn**, è stato implementato l'attributo *Username* all'in-

terno della tabella **Game**. Questo attributo riveste un ruolo fondamentale nell'associare una partita a un giocatore specifico, considerando l'unicità dello username. Inoltre, la presenza di tale attributo apre la possibilità per future implementazioni, come la creazione di un registro storico globale che coinvolga tutti i giocatori.

In questa fase dell'attuazione, la situazione dal punto di vista organizzativo è la seguente:

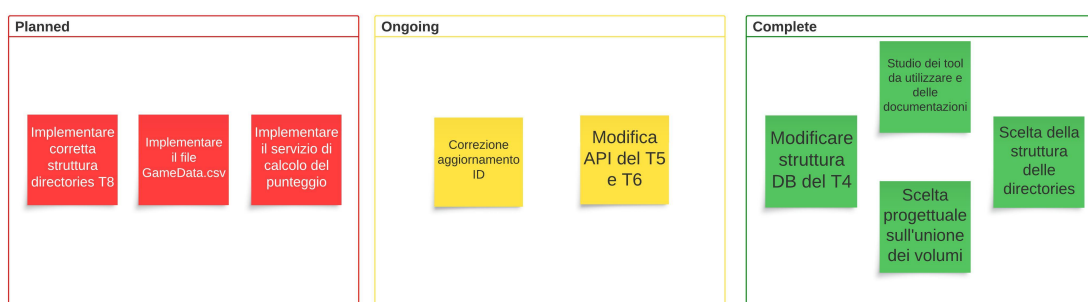


Figura 3.7: Progressi iterazioni

3.4 Modifiche effettuate sul task T6

In seguito all'introduzione dell'attributo *Username*, è stato indispensabile valutare la corretta memorizzazione all'interno della tabella **Game**, richiedendo così una modifica all'API */run* in T6. In particolare è stato necessario apportare una modifica alla richiesta HTTP PUT indirizzata alla tabella nel file *myController.java* in modo tale da tener conto anche del nuovo attributo. Le modifiche sono illustrate di seguito.


```
httpPut = new HttpPut("http://t4-g18-app-1:3000/games/" + String.valueOf(request.getParameter("gameId")));  
  
obj = new JSONObject();  
obj.put(name:"closedAt", time);  
obj.put(name:"username", request.getParameter("username"));
```

Figura 3.8: Modifiche effettuate nell'API */run*

3.5 Modifiche effettuate sul task T5

Per eseguire l'aggiornamento corretto degli ID delle partite, dei round e dei turni, è stato necessario esaminare l'API */save-data* ed apportare una modifica operativa. Questa API ha il compito di effettuare tre richieste POST verso le tabelle **Turn**, **Round** e **Game**. Tuttavia, è stata apportata una modifica sostanziale nel suo utilizzo: ora viene impiegata solo all'*inizio* della partita, creando così le rispettive voci nelle tabelle sopramenzionate, anziché essere utilizzata ad ogni pressione del tasto "Play/Submit".

Successivamente, al momento in cui si preme il tasto "Play/Submit", viene invocata l'API */run*, la quale esegue delle richieste PUT alle tabelle **Turn**, **Round** e **Game**. In dettaglio, la PUT sulla tabella **Turn** si occupa di aggiornare lo score e il vincitore del turno, mentre quella relativa a **Round** e **Game** viene eseguita solo nel caso di vittoria in quel particolare turno aggiornando il campo *ClosedAt*.

Sono state apportate modifiche al fine di incrementare il valore del turno al momento della pressione del tasto "Run". Tale azione avvia la misurazione della copertura del codice redatto dall'utente mediante

JaCoCo. L'obiettivo è consentire al giocatore di effettuare diverse misurazioni del proprio codice tramite JaCoCo, le quali sono registrate come turni giocati. Questo avviene prima di effettuare il confronto effettivo con il robot mediante la pressione del tasto "Play/Submit", il quale conclude la partita e determina il vincitore.

In particolare, prima delle modifiche, se l'utente perdeva contro il robot, poteva continuare a giocare fino a quando non riusciva a vincere. Con le attuali modifiche, una volta dichiarato il vincitore, la partita termina e i dati vengono salvati nel database e nel VolumeT8. Pertanto, anche in caso di sconfitta, il giocatore è tenuto a iniziare una nuova partita e non ha la possibilità di continuare a giocare. In ogni partita, comunque, il giocatore può effettuare diverse misurazioni tramite JaCoCo, le quali vengono conteggiate come turni, prima di confrontarsi con il robot. Al fine di illustrare in modo più approfondito come avvengono le modifiche nel volume T8 dopo le modifiche apportate, si presentano i seguenti diagrammi:

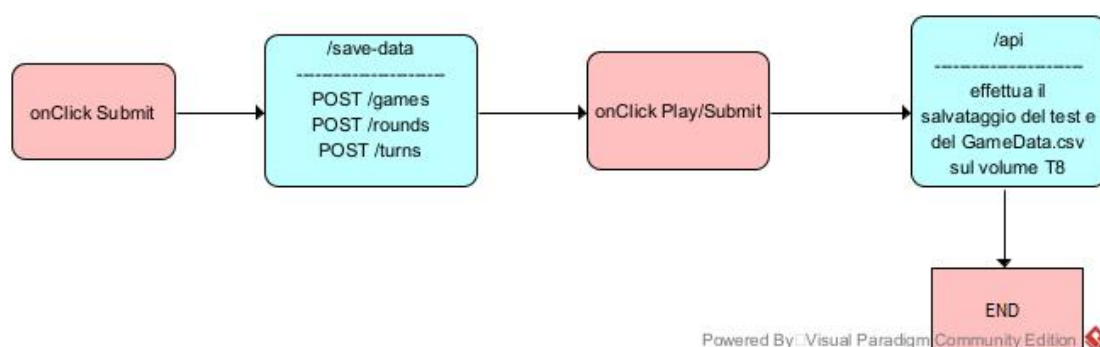


Figura 3.9: Diagramma di flusso per "Play/Submit"

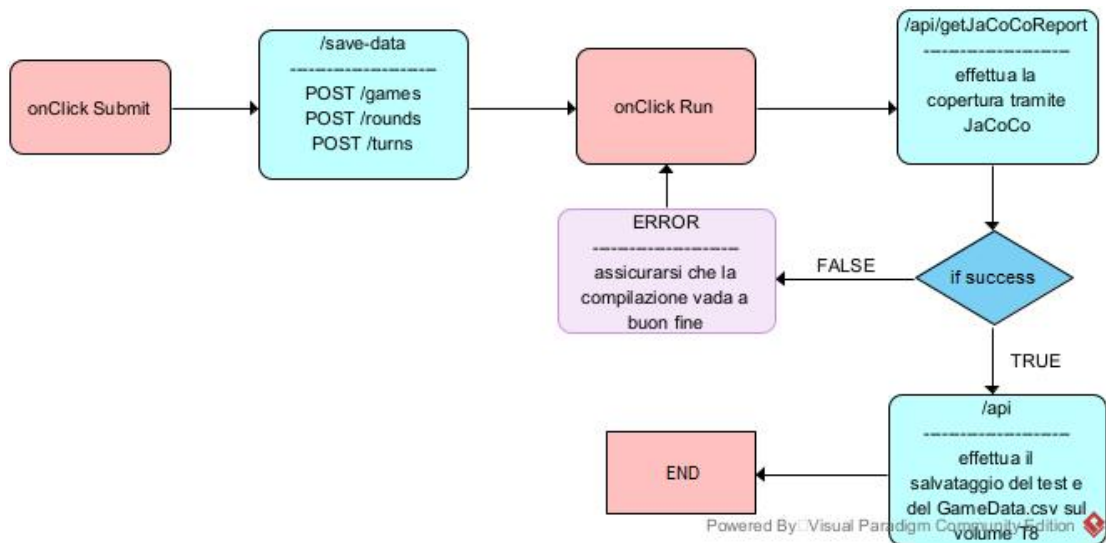


Figura 3.10: Diagramma di flusso per "Run"

Inoltre, anche il confronto finale con il robot viene considerato un turno aggiuntivo della partita e il motivo per il quale avviene ciò è correlato al fatto che un giocatore potrebbe scegliere di non eseguire misurazioni tramite JaCoCo durante una partita e potrebbe decidere di premere direttamente il tasto "Play/Submit" per determinare il vincitore. In questa eventualità, se non considerassimo anche questo passaggio come un turno aggiuntivo, non ci sarebbe alcun dato salvato riguardante la partita giocata dall'utente, poiché nessun turno sarebbe stato effettivamente giocato.

Inoltre, la decisione di limitare il confronto dell'utente con il robot a una sola volta è motivata dal fatto che, una volta premuto il tasto "Play/Submit", viene visualizzata nella console la percentuale di copertura del robot con cui si sta giocando. Questo implica che il

giocatore potrebbe continuare a eseguire misurazioni tramite JaCoCo fino a raggiungere una percentuale di copertura sufficiente per battere il robot e vincere la partita. Tuttavia, tale comportamento potrebbe essere considerato poco corretto in quanto vanificherebbe lo scopo del confronto iniziale e potrebbe compromettere l'integrità del gioco.

Di seguito forniamo un diagramma di attività per formalizzare la sequenza di operazioni compiute.

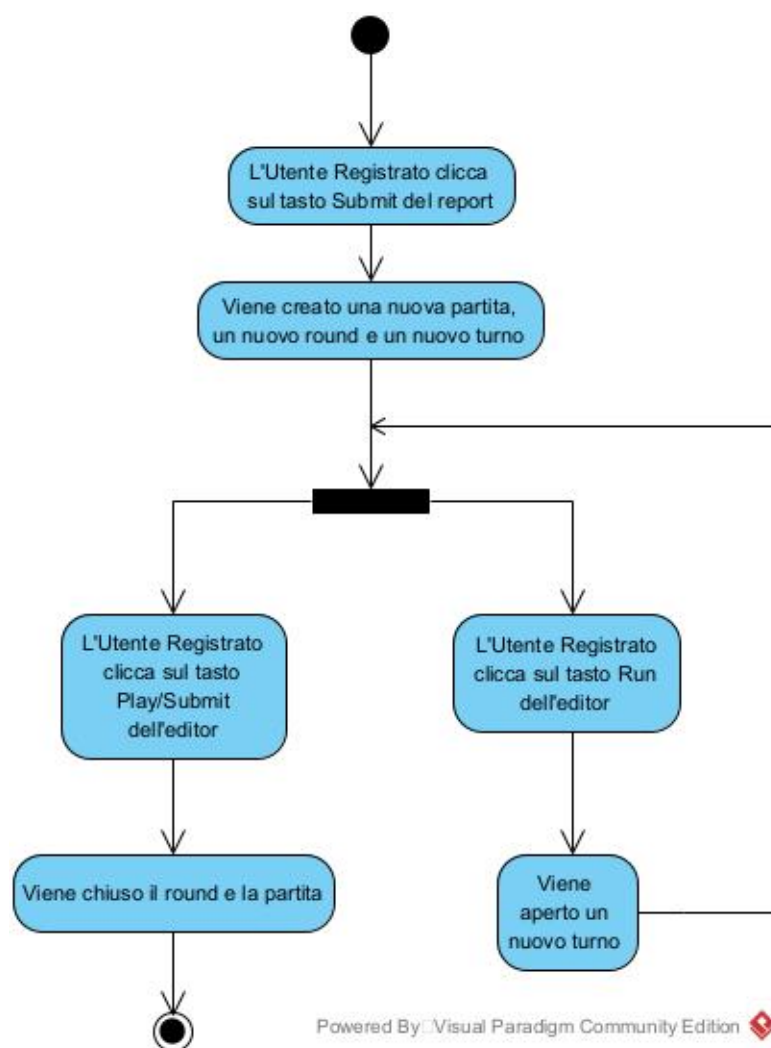


Figura 3.11: Diagramma di attività per "Run" e "Play/Submit"

Infine, per tenere traccia di chi ha partecipato a un determinato gioco, è stato indispensabile memorizzare lo username dell'utente nel local-Storage e recuperarlo direttamente dal cookie. Riportiamo le modifiche effettuate al codice del file *report.html*.

```
var username = parseJwt(getCookie("jwt")).sub;  
username = username.toString();  
localStorage.setItem("username", username);  
var nameCUT= localStorage.getItem("classe");  
var robotScelto = localStorage.getItem("robot");  
var difficulty = localStorage.getItem("difficulty");
```

Figura 3.12: Modifiche *report.html*

Per garantire che la struttura delle directory del volume T8 rispecchi quella prefissata, abbiamo modificato l'URL delle richieste effettuate nell'*editor.html* in modo da inviare i parametri corretti. In particolare, l'API per il salvataggio dei file pertinenti nel volume T8, fornita dalla precedente documentazione, presenta il seguente formato:

http://ip:porta/api/percorso_classe+percorso_test+percorso_salvataggio

- **ip:porta:** Indirizzo IP e numero di porta su cui il server è connesso e comunica. La porta scelta è la 3080;
- **percorso_classe:** Percorso della classe da testare. Deve includere sia il nome della classe che il nome del package in cui è contenuta;
- **percorso_test:** Percorso della cartella in cui salvare il test scritto dall'utente;

- **percorso_salvataggio**: Percorso della cartella in cui salvare i risultati ottenuti. Il nome del file salvato sarà "GameData.csv".

Le modifiche principali sono state apportate al parametro **percorso_test** della richiesta e nello specifico prevedono l'aggiunta delle directory *StudentLogin* e *TestReport*, nonché di tutte le altre directory legate agli ID della partita, del round e del turno.

```
// Percorso Classe
var classe = 'VolumeT8/FolderTreeEvo/' + localStorage.getItem("classe") + '/'
+ localStorage.getItem("classe") + 'SourceCode/' + localStorage.getItem("classe") + '.java';

// Percorso Test
var test = '/VolumeT8/FolderTreeEvo/' + localStorage.getItem("classe")
+ '/StudentLogin/Game' + localStorage.getItem("gameId") + '/Round' + localStorage.getItem("roundId")
+ '/Turn' + orderTurno + '/TestReport';

// Percorso Api
var apiBaseUrl = '/api/';

// Url Finale
var url = apiBaseUrl + classe + '+' + test + '+/app';
```

Figura 3.13: Modifiche parametro **percorso_test**

Come possiamo osservare dalla figura precedente, l'ID del turno da inserire nella richiesta per la creazione della relativa cartella nel volume T8 non è prelevato dal **localStorage**, ma piuttosto attraverso la variabile *orderTurno*. Quest'ultima è stata appositamente introdotta per tenere traccia dei vari turni all'interno di una singola partita.

3.6 Modifiche effettuate sul task T8

In seguito a questa serie di modifiche orientate alla correzione della gestione degli ID e all'introduzione di attributi necessari, è stato possibile procedere alla modifica del T8 al fine di consentire la corretta gestione delle richieste con i nuovi parametri. Innanzitutto, è stata modificata l'API `/tests/`, la quale ha lo scopo di consentire l'accesso ai test scritti dall'utente per mostrare eventuali storici. Tale modifica è stata necessaria in quanto è stato modificato il percorso di salvataggio del test, che deve essere ora recuperato. La figura 3.14 illustra le modifiche effettuate sul file `prova_esecuzione_parametri4.js`.

```
else if (req.url.startsWith('/tests/')) {
  res.setHeader('Access-Control-Allow-Origin', '*');

  const path = req.url.split('/').slice(2); // Rimuovi il primo elemento vuoto e "/tests/"

  if (path.length >= 2) {
    const game = path[0];
    const round = path[1];
    const turn = path[2];
    const classe = path[3];

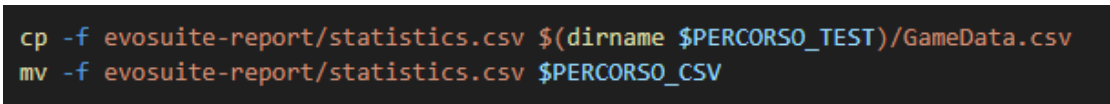
    console.log(`Valore di "gameId": ${game.replace('Game', '')}, Valore di "roundId":
    ${round.replace('Round', '')}, Valore di "turnId": ${turn.replace('Turn', '')},
    Valore di "nomeClasse": ${classe}`);

    const testPath = '/VolumeT8/FolderTreeEvo/' + classe + '/StudentLogin/' + game
    + '/' + round + '/' + turn + '/TestReport/Test' + classe + '.java';

    fs.readFile(testPath, (err, data) => {
      if (err) {
        res.statusCode = 500;
        res.end('Errore nel leggere il file Java');
      } else {
        res.setHeader('Content-Disposition', 'attachment; filename=yourfile.java');
        res.setHeader('Content-Type', 'text/plain');
        res.end(data);
      }
    });
  } else {
    console.log('Percorso della richiesta non contiene numero e nome');
  }
}
```

Figura 3.14: Modifiche legate all'API `/tests/`

In aggiunta, per la conclusione del requisito R1 sono state apportate ulteriori modifiche allo script *robot_misurazione_utente.sh*. Queste modifiche sono state necessarie poiché il percorso di salvataggio del file *GameData.csv*, che dovrebbe contenere la copertura del codice scritto dall'utente, era completamente errato e non vi era modo di accedervi tramite il volume T8. Per risolvere questo problema, è stato eseguito un processo di copia del file in questione nella cartella apposita del volume T8.



```
cp -f evosuite-report/statistics.csv $(dirname $PERCORSO_TEST)/GameData.csv
mv -f evosuite-report/statistics.csv $PERCORSO_CSV
```

Figura 3.15: Modifiche legate allo script *robot_misurazione_utente.sh*

In questo modo, il file *GameData.csv* prima di essere eliminato dalla cartella */app* utilizzata in locale dal *robot_misurazione_utente.sh* viene copiato nella apposita directory del volume T8.

Allo scopo di riassumere il processo di misurazione dell'utente riportiamo il seguente diagramma di attività illustrato nella figura 3.16.

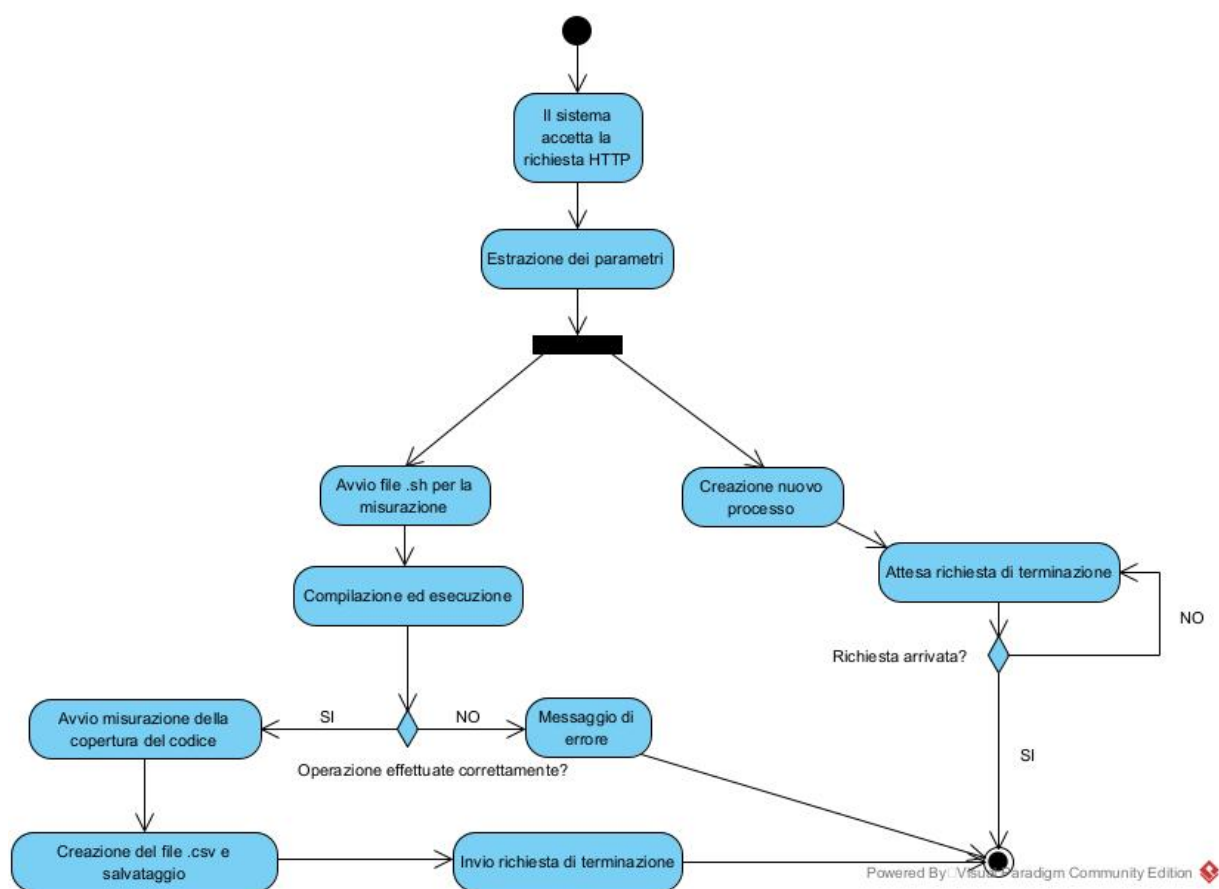


Figura 3.16: Diagramma di attività misurazione utente

Nello specifico, l'estrazione dei parametri, che è la parte che ha principalmente subito modifiche, consiste nell'estrazione del nome della classe, del nome e del percorso del package, del percorso di salvataggio del test scritto dall'utente e del percorso di salvataggio del file *GameData.csv*.

In definitiva, la struttura finale delle directory del volume T8 risulta essere la seguente:

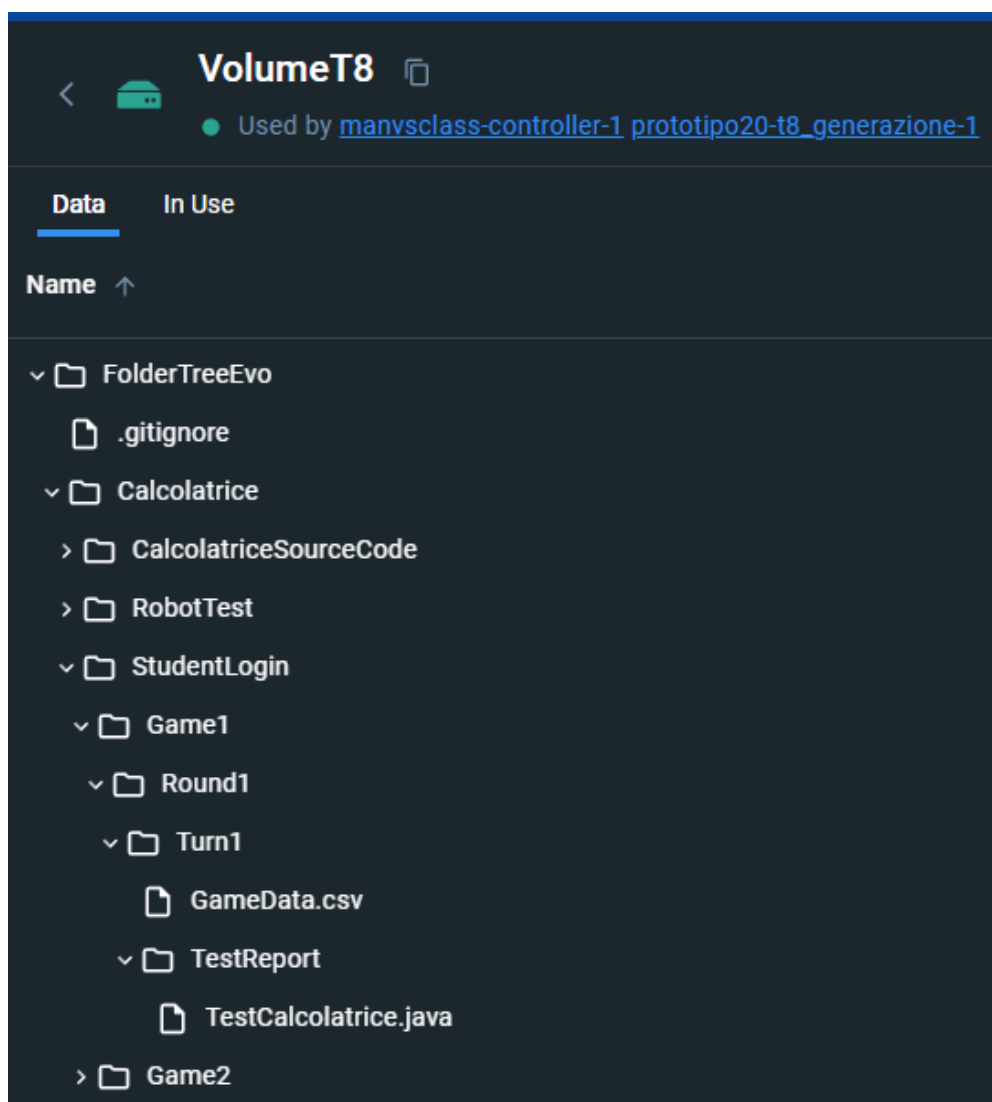


Figura 3.17: Struttura definitiva volume T8

Con questo si conclude l'implementazione del requisito R1, il quale ha portato a modifiche piuttosto evidenti nella struttura del volume T8, nonostante il numero di righe non sia aumentato particolarmente.

3.6.1 Refactoring T8

Durante la fase di refactoring, è stato necessario rinominare le variabili utilizzate con nomi più indicativi e chiari, al fine di rendere il codice più leggibile e comprensibile per coloro che non hanno lavorato su di esso fin dall'inizio. In particolare, ci si è concentrati sul file *prova_esecuzione_parametri4.js* per effettuare un refactoring dell'estrazione dei parametri dall'URL. Sono stati utilizzati nomi più esplicativi che chiariscono la porzione esatta dell'URL, come ad esempio *classPath* o *className*. Infine, si è proceduto a commentare brevemente alcune righe, rimuovendo invece commenti non pertinenti.

```
// Rimuove '/api/' dalla stringa dell'URL e sostituisce '+' con spazi.  
// Converte la stringa in un array di stringhe  
var parametri = req.url.slice(5).replace(/\+/g, ' ').split(' ');  
  
// Percorso della classe da testare  
var classPath = parametri[0];  
  
// Nome della classe da testare  
var className = classPath.substring(classPath.lastIndexOf("/") + 1,  
classPath.lastIndexOf("."));  
  
// Percorso del package della classe  
var packagePath = classPath.substring(0, classPath.lastIndexOf("/"));  
  
// Nome del package della classe  
var packageName = packagePath.substring(packagePath.lastIndexOf("/") + 1);  
  
// Elimina la prima stringa dall'array, così da poterla aggiornare  
// con i parametri sopra descritti  
parametri.shift();  
  
// Concatena i parametri in una stringa che viene inviata  
// al file robot_misurazione_utente.sh  
parametri = "/" + packagePath + " " + packageName + " " + className + " "  
+ parametri[0] + " " + parametri[1];
```

Figura 3.18: Codice sottoposto a refactoring

Capitolo 4

Requisito R8

Per l'implementazione del requisito R8, è stato necessario apportare delle modifiche alla tabella **Game** mediante l'aggiunta dell'attributo *Score*. Questo attributo consente la memorizzazione del punteggio di una partita giocata dall'utente, rendendo possibile anche la creazione di una classifica generale dei migliori giocatori. Inoltre, è stato necessario introdurre un nuovo servizio per il calcolo dello score dell'utente.

4.1 Modifiche effettuate sul task T4

Analogamente a quanto eseguito nel requisito R1 per l'attributo *Order*, anche in questo caso sono state apportate delle modifiche alle struct *Game* presenti nei file *model.go* e *game.go*. Precisamente, entrambe sono state adattate per includere l'attributo *Score*. Successivamente,

è stata effettuata una modifica alla struct *UpdateRequest* per gestire il formato JSON dato il nuovo parametro, e la funzione *fromModel* è stata aggiornata per la conversione da un oggetto *model.Game* a un oggetto *Game*, il primo destinato al database e il secondo per l'interazione con l'API.

A seguire sono riportate le immagini che illustrano alcune modifiche.

```
type Game struct {
    ID          int64      `json:"id"`
    CurrentRound int       `json:"currentRound"`
    Username    string    `json:"username"`
    Description  string    `json:"description"`
    Difficulty   string    `json:"difficulty"`
    Score       float64   `json:"score"`
    CreatedAt   time.Time `json:"createdAt"`
    UpdatedAt   time.Time `json:"updatedAt"`
    StartedAt   *time.Time `json:"startedAt"`
    ClosedAt    *time.Time `json:"closedAt"`
    Name        string    `json:"name"`
    Players     []Player  `json:"players,omitempty"`
}
```

Figura 4.1: Modifiche Struct *Game* in *game.go*

```
type UpdateRequest struct {
    CurrentRound int    `json:"currentRound"`
    Name         string `json:"name"`
    Username     string `json:"username"`
    Description  string `json:"description"`
    Score       float64 `json:"score"`
    StartedAt   *time.Time `json:"startedAt,omitempty"`
    ClosedAt    *time.Time `json:"closedAt,omitempty"`
}
```

Figura 4.2: Modifiche Struct *UpdateRequest*

```
func fromModel(g *model.Game) Game {  
    return Game{  
        ID:            g.ID,  
        CurrentRound:  g.CurrentRound,  
        Username:      g.Username,  
        Difficulty:    g.Difficulty,  
        Description:    g.Description.String,  
        Score:          g.Score,  
        CreatedAt:      g.CreatedAt,  
        UpdatedAt:      g.UpdatedAt,  
        Name:          g.Name,  
        StartedAt:      g.StartedAt,  
        ClosedAt:       g.ClosedAt,  
        Players:        parsePlayers(g.Players),  
    }  
}
```

Figura 4.3: Modifiche funzione *FromModel* in *game.go*

Per visualizzare l'intera struttura del database del T4 a seguito delle modifiche si rimanda alla figura 3.5

4.2 Modifiche effettuate sul task T6

Inizialmente è stata presa la decisione di implementare il calcolo del punteggio seguendo il medesimo approccio proposto dal requisito assegnato, secondo il quale:

- $Punteggio_vittoria = 1 + LOC\%$.
- $Punteggio_sconfitta = -1$.

In particolare, abbiamo pensato di escludere il caso di pareggio, poiché potrebbe verificarsi frequentemente una percentuale di copertura del 100% da parte dei robot di livello superiore e non solo, il che porterebbe a un numero elevato di partite concluse in parità.

Tuttavia, questo approccio si è rivelato poco efficace poiché non considera il numero di misurazioni effettuate dal giocatore prima del confronto diretto con il robot, ossia il numero di turni. Pertanto, si è deciso di determinare il punteggio con un criterio differente.

4.2.1 Calcolo del Punteggio

La scelta definitiva è stata quella di determinare il punteggio in base al numero di turni impiegati dal giocatore per prepararsi al confronto. In particolare, viene premiato col punteggio maggiore il giocatore che vince in un numero di turni inferiore e con una LOC maggiore.

La funzione *calculateScore*, che viene chiamata nell'API */run*, è stata inserita nella classe *ParseUtil*, per soddisfare il criterio per cui la funzione di calcolo sia estensibile in futuro. Di seguito ne riportiamo l'implementazione:

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

Figura 4.4: Funzione *CalculateScore*

È essenziale evidenziare che questa funzione calcola il punteggio della partita indipendentemente dal suo esito. Tuttavia, è stato aggiunto un bonus supplementare di **50** punti al punteggio restituito dalla funzione *CalculateScore*, precedentemente implementata, nel caso in cui l'utente vinca la partita. Di seguito sono riportate le modifiche apportate al codice.

```
if (roboScore > userScore) {  
    gameScore = Math.round(ParseUtil.calculateScore(userScore, numTurnsPlayed + 1));  
} else {  
    gameScore = Math.round(ParseUtil.calculateScore(userScore, numTurnsPlayed + 1) + 50);  
    isWinner = true;  
}
```

Figura 4.5: Modifiche *MyController.java* requisito R8

Di conseguenza, il punteggio massimo auspicabile da un utente al termine di una partita è pari a **150**, il che avviene solo quando l'utente confronta direttamente il suo codice con quello del robot senza effettuare misurazioni preliminari e vince la partita.

4.3 Modifiche effettuate sul task T5

È stata effettuata la modifica del file *editor.html* con l'aggiunta dei parametri *orderTurno* e *gameScore*, dove il primo è passato all'API `/run` e il secondo è mostrato nel resoconto della partita.

4.4 Aggiornamenti post implementazione dei Requisiti R1 ed R8

Il soddisfacimento dei requisiti R1 e R8 ha generato delle variazioni nello scenario dei casi d'uso *Submit* e *Run* derivanti sia dalla visualizzazione del punteggio al termine della partita sia dalle modifiche effettuate al funzionamento intrinseco dell'applicazione. Di seguito riportiamo il diagramma di attività per il caso d'uso *Submit* e gli scenari aggiornati per i casi d'uso precedentemente menzionati.

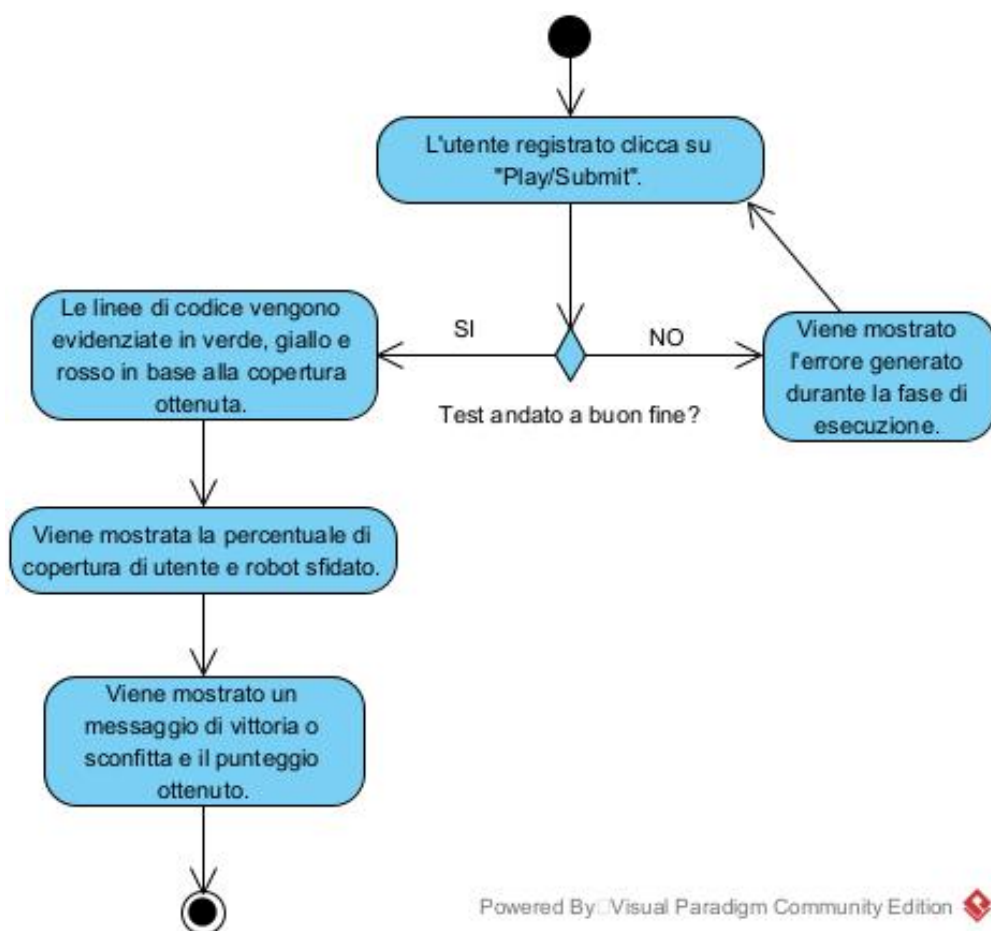


Figura 4.6: Diagramma di attività Submit

CASO D'USO	SUBMIT
ATTORE PRIMARIO	Utente registrato
DESCRIZIONE	I risultati del test scritto dall'utente vengono confrontati con quelli del robot e la partita viene conclusa.
PRE-CONDIZIONI	L'utente ha avviato la partita correttamente e ha prodotto una classe di test.
SEQUENZA DI EVENTI PRINCIPALE	<ol style="list-style-type: none"> 1. Il giocatore inserisce la soluzione del livello. 2. Il giocatore preme il pulsante "Play/Submit". 3. Viene mostrato a schermo il vincitore della partita e vengono mostrati sulla console i risultati del confronto con il robot.
POST-CONDIZIONE	Aggiornamento e salvataggio delle informazioni della partita sia all'interno del volume T8 che del database.
SEQUENZA DI EVENTI ALTERNATIVA	<ul style="list-style-type: none"> • Se il test scritto dall'utente non supera la fase di compilazione viene mostrato un messaggio di errore.

Tabella 4.1: Scenario aggiornato caso d'uso "SUBMIT"

Come possiamo notare dalle tabelle 4.1 e 4.2, le modifiche effettuate riguardano principalmente la sequenza di eventi principali e le post-condizioni dello scenario per entrambi i casi d'uso.

CASO D'USO	RUN
ATTORE PRIMARIO	Utente registrato
DESCRIZIONE	Il test scritto dall'utente subisce un processo di misurazione tramite JaCo-Co e viene giocato un turno.
PRE-CONDIZIONI	L'utente ha avviato la partita correttamente e ha prodotto una classe di test.
SEQUENZA DI EVENTI PRINCIPALE	<ol style="list-style-type: none"> 1. Il giocatore inserisce la soluzione del livello. 2. Il giocatore preme il pulsante "Run". 3. Vengono mostrati sulla console i risultati della misurazione del test scritto dall'utente.
POST-CONDIZIONE	Aggiornamento e salvataggio delle informazioni del turno sia all'interno del volume T8 che del database
SEQUENZA DI EVENTI ALTERNATIVA	<ul style="list-style-type: none"> • Se il test scritto dall'utente non supera la fase di compilazione viene mostrato un messaggio di errore.

Tabella 4.2: Scenario aggiornato caso d'uso "RUN"

Si conclude così anche l'implementazione del requisito R8, il quale ha richiesto meno modifiche rispetto al requisito R1. L'unica aggiunta evidente al codice è stata la funzione *CalculateScore*, la quale è stata progettata per garantire la massima flessibilità possibile.

Capitolo 5

Requisiti Aggiuntivi

Durante l'implementazione dei requisiti R1 e R8, è emersa la necessità di apportare modifiche all'applicazione al fine di migliorare la sua qualità complessiva. Di seguito sono elencati e descritti ulteriori requisiti aggiuntivi che sono stati trattati.

5.1 Coerenza tra il Database del T4 e il Volume T8

Il problema principale riscontrato riguarda la discrepanza tra gli ID dei turni, dei round e delle partite salvate nel volume T8 rispetto a quelli presenti nel database. Questa discrepanza è causata da problemi di gestione dei salvataggi all'interno del database stesso. È importante notare che questa problematica è stata precedentemente riportata nel requisito R2, anche se non era stato assegnato al nostro team. Tut-

tavia, in seguito alle modifiche sostanziali apportate al funzionamento delle partite, abbiamo deciso di affrontare anche questo problema aggiuntivo.

5.1.1 Gestione tasto "Run"

Con le nuove modifiche effettuate per il soddisfacimento dei requisiti R1 e R8, al momento dell'avvio di una nuova partita da parte dell'utente, vengono create nel database varie voci relative alla nuova partita attraverso una richiesta all'API */save-data*. In particolare, vengono generati **3** record distinti nelle tabelle **Game**, **Round** e **Turn**. Tuttavia, quando l'utente preme il pulsante "Run" per giocare il turno successivo, viene salvato correttamente il turno nel volume T8, ma non viene effettuata alcuna richiesta per aggiornare il turno nel database.

A tal fine, sono state introdotte due richieste condizionate al momento della pressione del tasto "Run". In particolare, se si verifica la condizione in cui *Order* == 1, indicando il primo turno, verrà eseguito un aggiornamento tramite una richiesta PUT nel record relativo al turno già creato nel database all'avvio della partita. Se tale condizione invece non si verifica significa che il primo turno è già stato aggiornato correttamente con la PUT, per cui sarà necessario inserirne uno nuovo. In questo caso, verrà effettuata una richiesta POST per inserire un nuovo record nella tabella **Turn** del database.

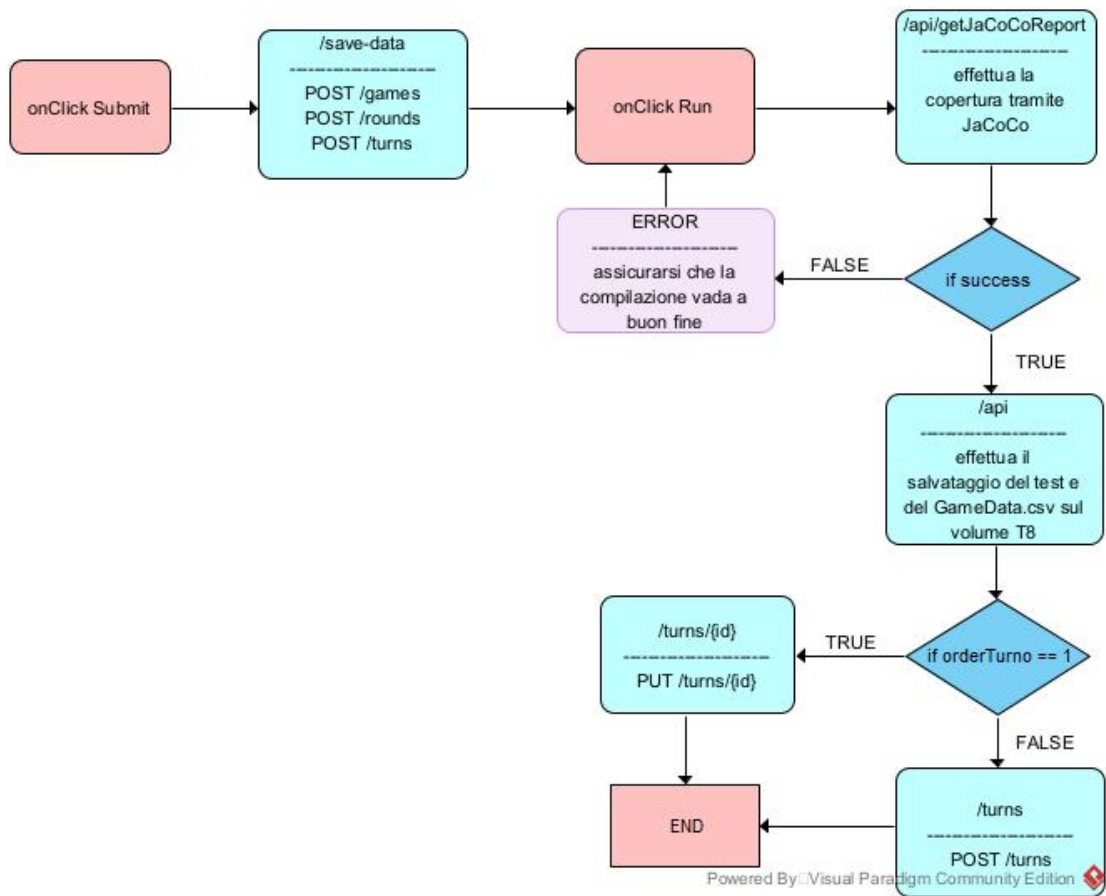


Figura 5.1: Diagramma di flusso "Run" aggiornato

5.1.2 Gestione tasto "Play/Submit"

Analogamente, anche alla pressione del pulsante "Play/Submit", che consente di chiudere e salvare la partita tramite una chiamata all'API */run*, si è verificato che il salvataggio nel volume T8 avveniva correttamente, mentre era necessario apportare delle modifiche per mantenere la coerenza con il database. Infatti, l'API */run* si limitava a effettuare aggiornamenti all'interno del database, presumendo che ci fossero già voci da modificare. Tuttavia, se il giocatore aveva già giocato dei turni, l'entry iniziale del turno inserita durante l'avvio della partita

sarebbe stata già aggiornata. Per questo motivo, è stata introdotta una clausola condizionale e una nuova richiesta anche in questo caso. Quando si verifica che $Order == 1$, viene eseguita una PUT nella voce già creata, mentre in caso contrario viene effettuata una POST.

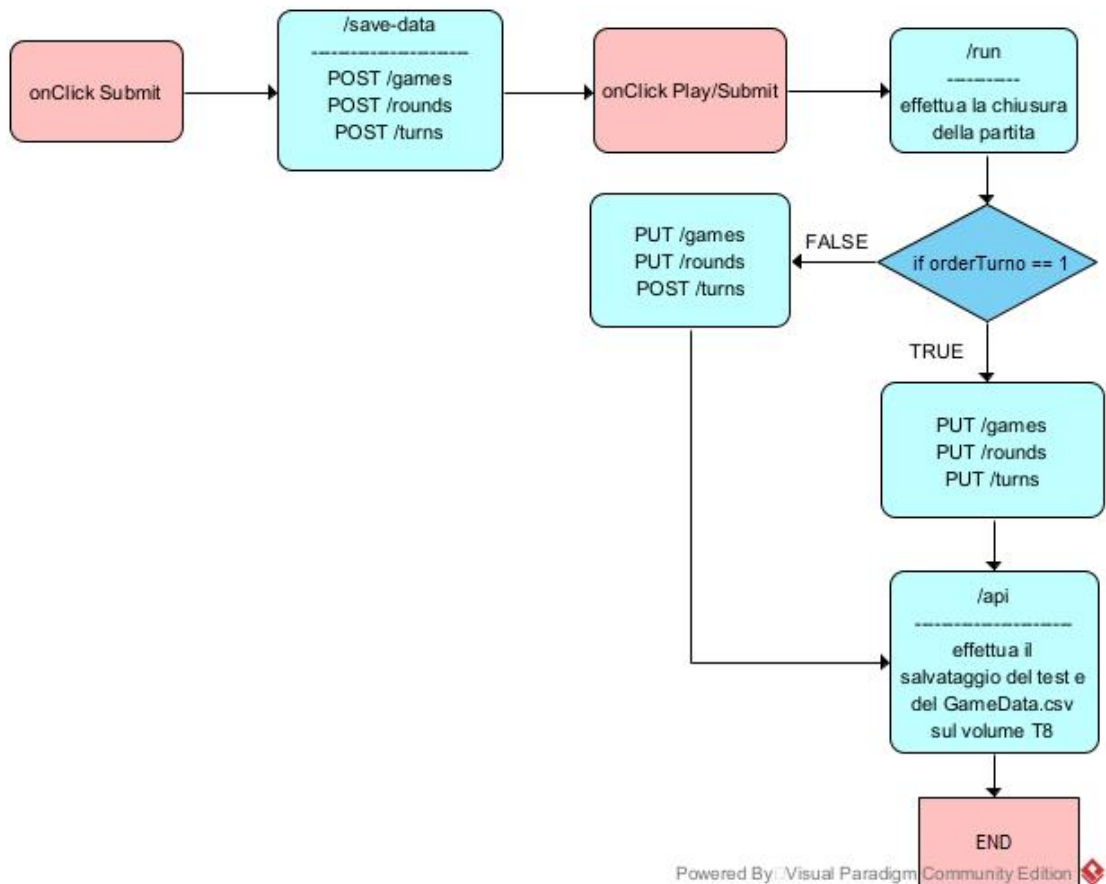


Figura 5.2: Diagramma di flusso "Play/Submit" aggiornato

Nelle figure 5.1, 5.2 è possibile consultare rispettivamente i diagrammi di flusso aggiornati per il tasto "Run" e per il tasto "Play/Submit".

5.2 Miglioramenti generali

Tra le problematiche aggiuntive emerse durante il testing manuale dell'applicazione web, si è riscontrato il malfunzionamento dello storico dei turni giocati e una visualizzazione errata delle informazioni della partita correntemente giocata.

5.2.1 Storico

In particolare, lo storico non riusciva a ottenere i test scritti dall'utente nei turni giocati a causa della modifica del percorso di salvataggio di questi ultimi nel requisito R1. Per risolvere questo problema, è stato necessario modificare l'URL della richiesta effettuata nel file *editor.html* del T5 e risolvere varie problematiche di visualizzazione del contenuto nella console area. È da notare che la richiesta con il nuovo URL è stata gestita correttamente dal T8 in quanto era stata precedentemente adattata (vedi figura 3.13).

5.2.2 Game Info

In merito alla visualizzazione delle informazioni della partita, con il tasto "GameInfo" si otteneva la schermata illustrata in figura 5.3.

Di conseguenza, è immediatamente evidente la ripetizione di alcuni parametri e l'errata visualizzazione degli ID del turno corrente. Inoltre,



Figura 5.3: Schermata Game Info

dopo aver giocato vari turni, tale schermata non veniva mai aggiornata e mostrava sempre gli stessi valori.

Per risolvere questo problema, è stata modificata la funzione *OpenInfoModal* all'interno del file *editor.html* nel T5 in modo da aggiornare correttamente i parametri ogni volta che viene premuto il tasto "GameInfo".

5.3 Nuova modalità e API

La modifica del funzionamento del gioco in sé è motivata sotto vari aspetti. Ad esempio, sarebbe incongruo introdurre una classifica dei giocatori migliori con i relativi punteggi se fosse permesso continuare a giocare dopo una sconfitta fino al raggiungimento della vittoria. Queste modifiche potrebbero, tuttavia, sollevare la questione dell'u-

tente che non potrebbe più esercitarsi nella scrittura dei test una volta confrontatosi con il robot, poiché la partita terminerebbe indipendentemente. In realtà, a seguito delle modifiche, l'utente potrebbe continuare ad esercitarsi effettuando misurazioni tramite JaCoCo durante i vari turni di gioco. Tuttavia, si è scelto di offrire al giocatore la possibilità di allenarsi in tranquillità con il robot selezionato senza essere valutato.

5.3.1 Aggiunta schermata /gamemode

A tal scopo, è stata inizialmente implementata la schermata sottostante, con l'idea di renderla disponibile ai colleghi per future implementazioni.



Figura 5.4: Schermata per la scelta della modalità di gioco

Per integrare questa schermata nel flusso dell'applicazione, è stato creato un nuovo file denominato *gamemode.html*, nel quale è stato trasferito l'intero funzionamento precedente della schermata principale, che consentiva di selezionare la classe e il robot per giocare. Successivamente, la nuova schermata è stata inserita nel file *main.html*. In questo modo, dopo il login, la prima schermata visualizzata dall'utente sarà quella relativa alla scelta della modalità di gioco. Questo approccio ha comportato una notevole riduzione delle modifiche necessarie al codice per implementare tale schermata. Inoltre, sono stati utilizzati gli stili già presenti nei file CSS e il file *main.js* per gestire adeguatamente la selezione della modalità al momento della pressione di ciascun tasto.

È importante sottolineare che al momento sono disponibili solo le modalità "Sfida un Robot" e "Allenamento" (di cui parleremo nella sezione successiva), mentre per quanto riguarda la modalità "Multiplayer" viene mostrato un messaggio a schermo che avverte della sua indisponibilità.

5.3.2 Modalità Allenamento

Dopo l'introduzione della schermata per la selezione delle modalità di gioco, è stata completamente implementata la modalità "Allenamento" con l'obiettivo di offrire ai giocatori la possibilità di migliorare le proprie abilità nella scrittura di testi senza che i punteggi ottenuti in-

fluenzino le statistiche dell'account. Questa modalità presenta le stesse schermate della modalità di sfida, con alcune modifiche riguardanti l'editor. In particolare sono stati creati due nuovi file *editorAllenamento.html* ed *editorAllenamento.js* per la separazione del codice HTML e JavaScript, mentre per quanto concerne il CSS è stato riutilizzato quello già disponibile. Poiché la modalità è finalizzata all'allenamento, non richiede il salvataggio dei dati. Di conseguenza, sono state apportate le seguenti modifiche:

- È stato eliminato il tasto "Game Info", poiché non è necessario senza la presenza di partite o turni registrati;
- I tasti "Play/Submit" e "Run" stati racchiusi in un unico tasto "RunAllenamento" caratterizzato dalla stessa grafica del tasto "Run" e che gestisce sia la misurazione dell'utente che la visualizzazione dei risultati.

Per quanto riguarda lo storico dei test, si è deciso di mantenerlo in quanto per sessioni di allenamento prolungate l'utente potrebbe voler dare un'occhiata ai test scritti precedentemente per valutarne le differenze. A tal scopo, è stata prevista una funzionalità di salvataggio temporaneo dei test scritti durante l'allenamento nel volume T8 sfruttando l'API già esistente utilizzata per la creazione del percorso in cui salvare i test e il *GameData.csv* dei vari turni giocati nella modalità "Sfida un Robot". La chiamata a questa API è stata integrata nella

funzione associata al tasto "RunAllenamento", e il percorso specificato nell'URL è mostrato in figura 5.5.

```
// Percorso salvataggio test allenamento  
var test = '/VolumeT8/FolderTreeEvo/Allenamento/Test' + testCounter + '/TestReport';
```

Figura 5.5: Nuovo percorso Allenamento

In particolare, la variabile *testCounter* tiene traccia del numero di test scritti dall'utente durante l'allenamento. In questo modo, viene creata nel volume T8 una cartella **Allenamento** momentanea che memorizzerà tutti i file *Java* delle classi di test prodotte dall'utente. Tale cartella viene sfruttata insieme all'indice *testCounter* per conservare la funzionalità dello storico e stampare sulla console tutti i test che sono stati misurati durante una sessione di allenamento.

Nel momento in cui l'utente decide di effettuare il logout o avviare una nuova sessione di allenamento, questa cartella viene eliminata completamente e ricreata per l'inserimento dei test relativi alla nuova sessione.

5.3.3 Aggiunta API per il T8

Per supportare queste funzionalità, è stata introdotta nel file *prova_esecuzione_parametri4.js* del T8 una nuova API, accessibile all'indirizzo **/remove-allenamento**. Questa API ha il semplice compito di eliminare la cartella "Allenamento" nel volume T8, se presente, e for-

nire un feedback sull'operazione. La sua implementazione è mostrata in figura 5.6.

```
else if (req.url.startsWith('/remove-allenamento')) {
  // Rimozione delle cartelle
  const pathToDelete = '/VolumeT8/FolderTreeEvo/Allenamento';
  try {
    // Verifica se la cartella esiste
    if (fs.existsSync(pathToDelete)) {
      // Rimuovi la cartella
      fs.rmdirSync(pathToDelete, { recursive: true });
      console.log('Cartella rimossa con successo:', pathToDelete);
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end('Cartella rimossa con successo');
    } else {
      console.log('La cartella non esiste:', pathToDelete);
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end('La cartella non esiste');
    }
  } catch (err) {
    console.error('Errore durante la rimozione della cartella:', err);
    res.writeHead(500, { 'Content-Type': 'text/plain' });
    res.end('Errore durante la rimozione della cartella');
  }
} else {
  res.end('Richiesta http per test non andata a buon fine');
}
```

Figura 5.6: API */remove-allenamento*

È stata poi introdotta una funzione *removeAllenamentoFolders* nel file *editorAllenamento.js* che si occupa di effettuare una chiamata all'API in questione. Tale funzione è stata integrata sia nella funzione di logout che in un *eventListener* attivato quando la pagina è pronta per essere scaricata, garantendo così la pulizia dei file delle sessioni precedenti anche in caso di chiusura diretta del browser.

In figura 5.7 sono illustrate le precedenti due aggiunte.

```
// Funzione per rimuovere la cartella Allenamento
Codeium: Refactor | Explain | X
function removeAllenamentoFolders() {
    var url = "/remove-allenamento";

    $.ajax({
        url: url,
        type: 'GET',
        timeout: 30000,
        Codeium: Refactor | Explain | Generate JSDoc | X
        success: function (data, textStatus, xhr) {
            console.log("Cartelle di allenamento rimosse con successo");
        },
        Codeium: Refactor | Explain | Generate JSDoc | X
        error: function (xhr, textStatus, errorThrown) {
            console.error("Errore durante la rimozione delle cartelle di allenamento:", errorThrown);
        }
    });
}

// Gestore per l'evento di chiusura della pagina
window.addEventListener('beforeunload', function (e) {
    removeAllenamentoFolders();
});
```

Figura 5.7: Funzione *removeAllenamentoFolders* ed *eventListener*

Tuttavia, poiché questo approccio non funziona sempre con tutte le tipologie di browser è stata introdotta una chiamata aggiuntiva a tale API anche in fase di avvio dell'allenamento per assicurare la corretta pulizia dei file delle sessioni precedenti.

Infine, per ottimizzare l'efficienza e la manutenibilità del sistema, data la similitudine tra gli editor utilizzati nelle due modalità di gioco, è stato introdotto un nuovo file denominato *commonEditor.js*, integrato nel T5. Al suo interno sono state raggruppate tutte le funzioni che sono condivise tra le due modalità di gioco, ottenendo così una significativa riduzione della complessità del codice, evitando inutili ridondanze e migliorando la coerenza e la coesione del sistema nel suo complesso.

5.4 Workflow definitivo

Nelle sezioni precedenti, sono state delineate tutte le modifiche apportate durante le iterazioni. Per riepilogare le modifiche al funzionamento del gioco, presentiamo i seguenti diagrammi di flusso, da considerare come versioni definitive. Il primo diagramma riguarda la pressione del tasto "Run", mentre il secondo riguarda la pressione del tasto "Play/Submit".

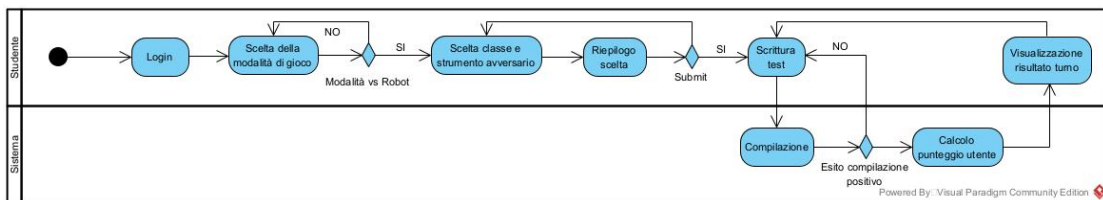


Figura 5.8: Workflow relativo a "Run"

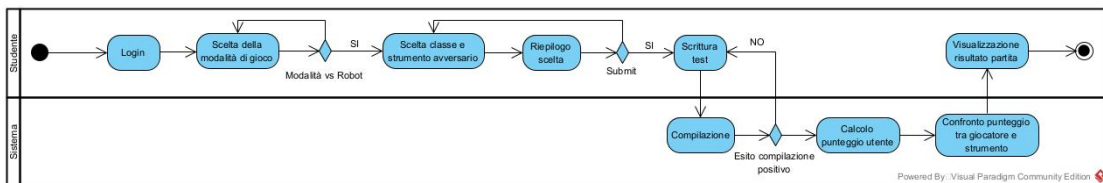


Figura 5.9: Workflow relativo a "Play/Submit"

Il flusso della modalità allenamento è da considerarsi simile a quello legato al tasto "Run". Con questo si può considerare conclusa l'implementazione dei requisiti aggiuntivi e, in generale, di tutti i miglioramenti apportati all'applicazione durante le molteplici iterazioni.

Capitolo 6

Deployment

Segue il diagramma di deployment in figura 6.1, che è stato consultato durante la fase di progettazione. I container invariati sono evidenziati in verde, mentre quelli soggetti a modifiche sono segnalati in rosso se riguardanti i requisiti R1 ed R8, in giallo se riguardanti i requisiti aggiuntivi.

L'applicativo è ospitato su un server che utilizza il sistema operativo Windows 11. La rete è condivisa da tutti i task, garantendo l'isolamento dei container dalla rete dell'host e consentendo al contempo la comunicazione tra di essi tramite una rete virtuale.

Il seguente diagramma riflette la configurazione precedente rispetto alla situazione attuale, tuttavia, rappresenta accuratamente le modifiche apportate per l'implementazione dei requisiti R1 ed R8. Per la soluzione più recente, si rimanda alla documentazione aggiornata che

include l'esposizione su rete pubblica.

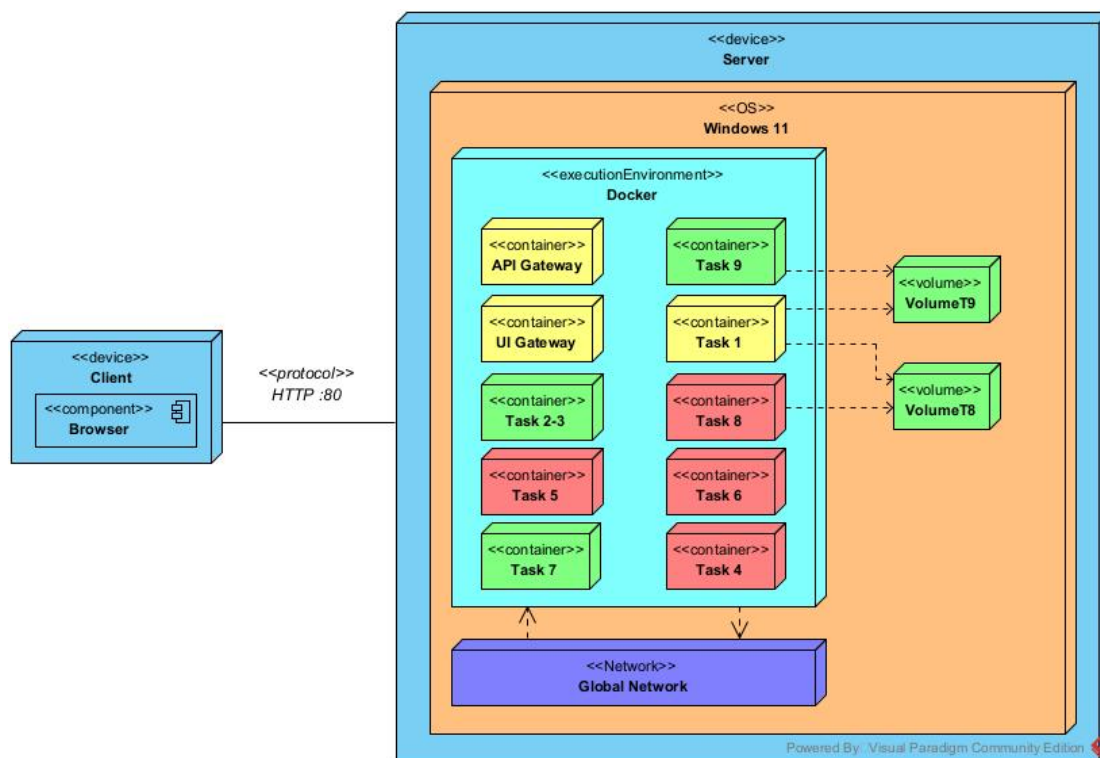


Figura 6.1: Diagramma di Deployment

Dall'analisi del diagramma, emerge che il volume T8 viene condiviso tra i container relativi ai task 1 e 8. Nello specifico, la porzione del File System condivisa riguarda le classi e la copertura di EvoSuite. Analogamente, il volume T9 è condiviso tra i container dei task 1 e 9, dove la sezione condivisa del File System riguarda le classi e la copertura di Randoop ed Emma.

6.1 Integrazione con la nuova versione

Tutte le modifiche menzionate nei capitoli precedenti sono state effettuate sulla versione dell'applicazione che non poteva ancora essere esposta su un dominio pubblico. Queste modifiche sono state quindi integrate nella nuova versione fornita dal team A10.

6.1.1 Analisi dei file della nuova versione

Il principale problema durante l'integrazione è stato l'analisi dei file che richiedevano modifiche, poiché non presentavano più le stesse strutture. Un esempio è il file *editor.html*, che inizialmente conteneva sia il codice HTML che il CSS e il JavaScript. Nella nuova versione, le responsabilità sono state separate ed è stato creato un nuovo file *editor.js*, che principalmente contiene il codice JavaScript. Di conseguenza, la maggior parte delle modifiche che erano state inizialmente implementate nell'*editor.html* sono ora da considerare effettuate nel file *editor.js*.

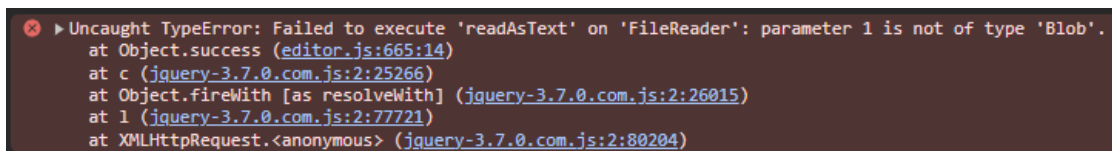
6.1.2 Corretta gestione degli URL

Un'altra problematica è stata la modifica degli URL delle richieste che erano state da noi aggiunte per la corretta gestione dei salvataggi nel database del T4. Poiché la nuova versione dell'applicazione consente l'esposizione su una rete pubblica, tutti gli URL non facevano più ri-

ferimento al prefisso *localhost*, ma venivano gestiti tramite un server proxy. Di conseguenza, è stato necessario consultare la documentazione fornita dal team per correggere accuratamente tutti gli indirizzi delle richieste in modo che fossero conformi alla nuova configurazione del server.

6.1.3 Problematica nella lettura della misurazione dell'utente

Nella nuova versione dell'applicazione, è stato riscontrato un problema che impediva alla console di visualizzare correttamente i risultati della misurazione dell'utente quando si premeva il pulsante "Run". Durante l'analisi dell'applicazione tramite gli strumenti web forniti da *Chrome*, è stato individuato l'errore illustrato in figura 6.2.



```
✖ ▶ Uncaught TypeError: Failed to execute 'readAsText' on 'FileReader': parameter 1 is not of type 'Blob'.
    at Object.success (editor.js:665:14)
    at c (jquery-3.7.0.com.js:2:25266)
    at Object.fireWith [as resolveWith] (jquery-3.7.0.com.js:2:26015)
    at l (jquery-3.7.0.com.js:2:77721)
    at XMLHttpRequest.<anonymous> (jquery-3.7.0.com.js:2:80204)
```

Figura 6.2: Errore visualizzazione console

Per risolvere questo problema, sono state apportate delle modifiche alle righe di codice coinvolte, in particolare quelle riportate nel messaggio di errore. Si è scoperto che la gestione della lettura del file *.csv* contenente le statistiche di misurazione del test scritto dall'utente era erranea ed è stata corretta di conseguenza.

6.1.4 Aggiornamento classifica studenti

Per completare questa fase di integrazione, è stata presa la decisione di adattare il funzionamento della classifica degli studenti, accessibile attraverso il cruscotto dell'amministratore. Questa classifica è progettata per fornire una visione dei punteggi ottenuti dagli utenti che hanno partecipato almeno a una partita, insieme al tempo di gioco totale e altre informazioni rilevanti. Tuttavia, non era presente alcuna gestione del punteggio ottenuto dall'utente in quanto nella versione A10 mancava la vera e propria logica per il calcolo e il salvataggio di quest'ultimo. Di conseguenza, avendo introdotto in precedenza l'attributo *score* è stato sufficiente effettuare alcune modifiche al file *player.html* del T1 che gestisce il funzionamento della classifica.

Posizione	Punti	Name	Surname	Email	Studies	Partite Giocate	Tempo Totale (min)
1	127	Simone	Rinaldi	ciao@gmail.com	BSc	2	1.74
2	100	Chiara	Rinaldi	prova@gmail.com	BSc	2	0.51

Figura 6.3: Classifica studenti con punteggio

Infatti, la soluzione prevede il recupero dell'attributo *score* dalla tabella **Game**, e tramite l'identificatore del giocatore corrispondente, l'implementazione di una logica per sommare tutti i punteggi ottenuti da quel particolare utente in tutte le partite giocate. Il risultato di questa operazione è il punteggio totale, il quale viene poi visualizzato nella tabella, come illustrato nella figura 6.3.

Capitolo 7

Testing

Nel contesto dell'industria del software, il testing rappresenta un pilastro fondamentale per garantire la qualità e l'affidabilità dei prodotti che vengono sviluppati. Il testing, inteso come processo di valutazione e verifica del software, ha il compito di individuare difetti, errori o comportamenti indesiderati all'interno del sistema. Questo processo non solo aiuta a identificare e correggere i difetti prima che il software venga rilasciato, ma contribuisce anche a migliorare la stabilità, la sicurezza e le prestazioni complessive del prodotto.

7.1 Testing delle API con Postman

Nel nostro caso, abbiamo iniziato a testare attentamente il funzionamento corretto delle API. Per tale scopo, abbiamo utilizzato Postman, un software il cui funzionamento è descritto nel paragrafo dedicato. In

particolare, si è analizzato con maggiore attenzione le seguenti API:

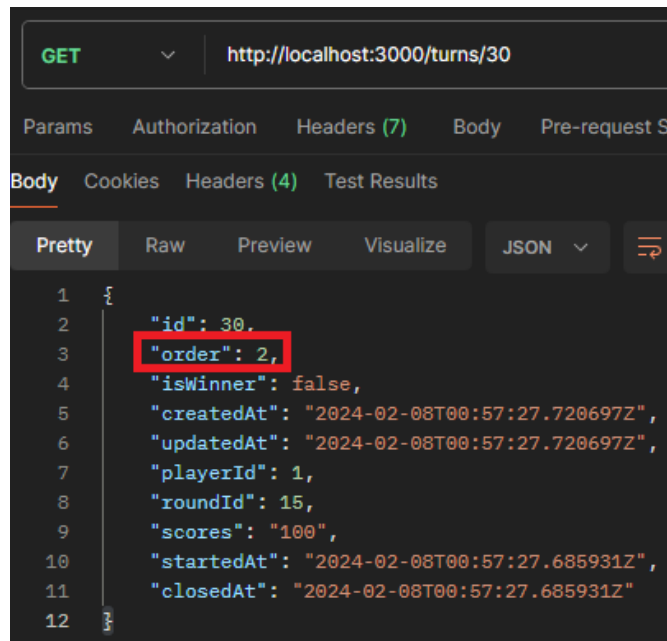


Figura 7.1: Risultati test */turns*

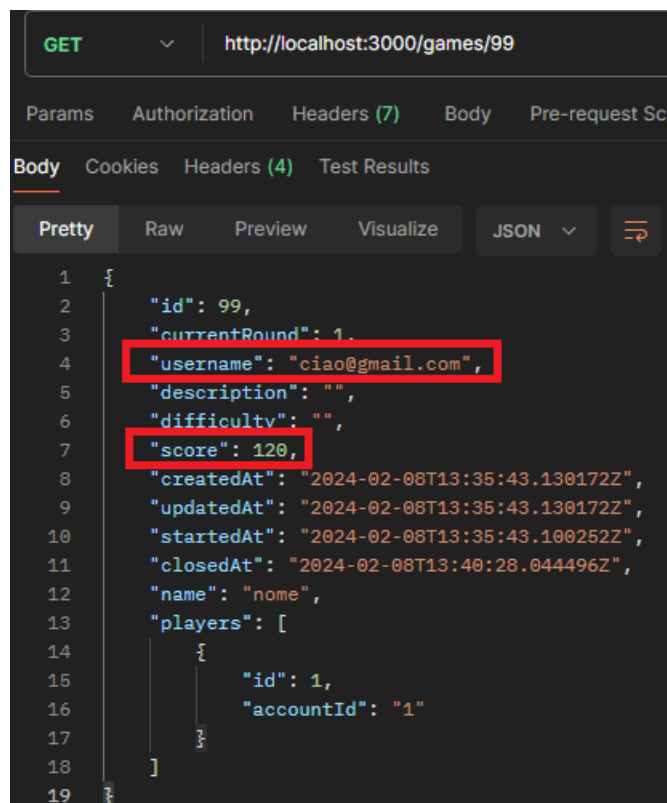


Figura 7.2: Risultati test */games*

Come osservabile dalle figure 7.1 e 7.2, i parametri di interesse, ovvero *Order* nella tabella **Turn**, e *Username* e *Score* nella tabella **Game**, sono stati inseriti con successo nelle rispettive tabelle. Infatti, tramite una richiesta GET all'indirizzo appropriato, siamo in grado di visualizzarli. Tramite Postman siamo stati in grado di testare anche la creazione e l'aggiornamento di record all'interno del database tramite la simulazione di richieste POST e PUT.

7.2 Verifica della coerenza dei dati tramite DBeaver

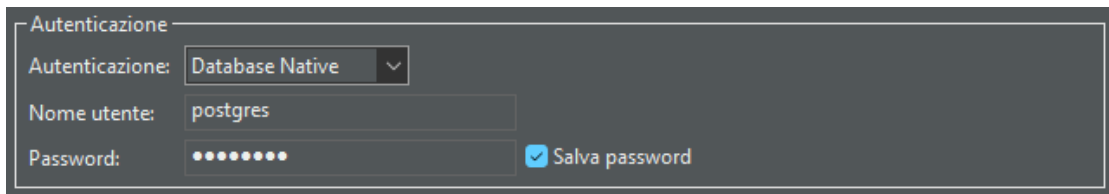
Successivamente, abbiamo proseguito con l'utilizzo del programma DBeaver per l'accesso al database del T4 al fine di eseguire una serie di verifiche supplementari. DBeaver si è rivelato un'importante risorsa in quanto ci ha fornito un'ampia panoramica di tutti i dati presenti nel database, consentendoci di esaminare in modo completo e approfondito la struttura e i contenuti di quest'ultimo.

L'utilizzo di questo strumento ci ha infatti consentito di identificare il motivo per cui non era possibile inserire più turni all'interno di un singolo round di una partita. Abbiamo notato che, attraverso l'esecuzione di uno script SQL per l'inserimento di più turni associati allo stesso *roundID*, il sistema restituiva messaggi di errore.

Questi messaggi di errore, forniti in modo chiaro e preciso, hanno

giocato un ruolo fondamentale nel processo di risoluzione del problema. Grazie a essi, infatti, siamo stati in grado di individuare rapidamente la causa sottostante del malfunzionamento, permettendoci di procedere con le necessarie correzioni e ottimizzazioni del sistema.

N.B. Per quanto riguarda l'accesso al database del T4, è importante notare che il tipo di database utilizzato è *PostgreSQL*. Nel caso si desideri utilizzare un software come DBeaver o qualsiasi altro strumento per l'accesso ai dati, è necessario tenere presente questa informazione. Inoltre, la password da utilizzare per accedere al database è "postgres".

The image shows a dark-themed authentication dialog box titled "Autenticazione". It contains three input fields: "Autenticazione:" with a dropdown menu set to "Database Native", "Nome utente:" with the text "postgres", and "Password:" with masked characters (dots). To the right of the password field is a checked checkbox labeled "Salva password".

7.3 Testing automatizzato con Selenium

Infine, abbiamo impiegato Selenium per automatizzare il testing riguardante l'uso dell'editor. Di conseguenza, abbiamo configurato l'ambiente per l'utilizzo di Selenium e abbiamo preparato i vari casi di test necessari. In seguito riportiamo una tabella contenente le casistiche analizzate con le relative post condizioni e i risultati dei test.

Test ID	Descrizione	Pre-Condizioni	Input	Output	Post-Condizioni	Esito
1	Verifica che la scelta di una classe e di un Robot avvenga correttamente.	L'utente ha effettuato l'accesso correttamente.	Calcolatrice, EvoSuite.	Caricamento della schermata successiva /report.	La classe e il Robot sono stati scelti correttamente.	PASS
2	Verifica che l'avvio di una partita avvenga correttamente.	L'utente ha selezionato la classe e il Robot con cui giocare.	Submit.	Caricamento della schermata successiva /editor.	La partita è stata inizializzata correttamente nel database.	PASS
3	Verifica che la compilazione del test scritto dall'utente avvenga correttamente.	L'utente ha avviato la partita ed ha scritto la sua soluzione.	Compila.	Visualizzazione sulla console dei risultati della compilazione.	Il test è stato compilato correttamente.	PASS
4	Verifica che la chiusura di un turno avvenga correttamente.	La compilazione del test scritto dall'utente ha avuto successo.	Run.	Visualizzazione sulla console dei risultati della misurazione ottenuta.	Il turno è stato memorizzato correttamente nel database e nel volume T8.	PASS
5	Verifica che la chiusura di una partita avvenga correttamente.	La compilazione del test scritto dall'utente ha avuto successo.	Play/Submit.	Visualizzazione del punteggio ottenuto con un messaggio di alert e sulla console dei confronti tra i test dell'utente e del Robot.	La partita è stata memorizzata correttamente nel database e nel volume T8.	PASS
6	Verifica che la visualizzazione dello storico avvenga correttamente.	L'utente ha giocato dei turni.	Storico.	Visualizzazione dei risultati e dei codici dei turni precedenti nella console.	Lo storico è stato visualizzato correttamente.	PASS
7	Verifica che la visualizzazione delle informazioni della partita avvenga correttamente.	L'utente ha avviato la partita con successo.	GameInfo.	Visualizzazione delle informazioni della partita corrente.	Le informazioni della partita sono state visualizzate correttamente.	PASS

Figura 7.3: Casi di test e relativi risultati

Capitolo 8

Sviluppi futuri

Ecco di seguito le nostre proposte per migliorare l'applicazione in diversi aspetti.

8.1 Miglioramento della sicurezza delle pagine web

Potrebbe essere necessario migliorare la sicurezza delle pagine web, poiché attualmente i sorgenti JavaScript richiamati sono visibili in chiaro. È consigliabile adottare pratiche di sicurezza per proteggere i sorgenti dell'applicazione. A tal fine, si raccomanda l'utilizzo del protocollo HTTPS anziché HTTP. Il protocollo HTTPS offre una maggiore sicurezza crittografica durante la trasmissione dei dati attraverso la rete, garantendo una comunicazione più sicura e protetta dagli attacchi informatici.

8.2 Miglioramento della robustezza

È importante affrontare il problema della robustezza dell'applicazione, specialmente in situazioni di bassa connettività. Attualmente, la pressione ripetuta di alcuni tasti (ad esempio, il pulsante di Submit per avviare una partita) può causare la creazione di più voci contemporanee nel sistema. Si consiglia di implementare controlli sul lato server per gestire correttamente queste situazioni e prevenire la creazione di voci duplicate o indesiderate nel sistema.

Inoltre, se l'utente decide di chiudere la partita o di giocare un turno senza aver prodotto un test che compila correttamente vengono comunque generate delle voci vuote nel database, generando così errori non gestiti. Si consiglia di implementare un meccanismo che gestisca adeguatamente questa situazione, evitando la creazione di voci vuote nel database quando la compilazione del test non è stata completata con successo.

8.3 User Friendliness

Per migliorare l'usabilità dell'applicazione per gli utenti, soprattutto nella sezione dei menù di gioco, potrebbe essere utile aggiungere dinamiche che semplifichino la navigazione e l'interazione. Per esempio, sarebbe vantaggioso includere un'opzione che consenta agli utenti di ritornare alla schermata di selezione della classe da testare e del robot

direttamente dall'interfaccia di gioco. Attualmente, la sola opzione disponibile è quella per effettuare il logout, il che comporta la necessità di eseguire nuovamente l'accesso dopo la conclusione di una partita al fine di continuare a giocare.

Potrebbe essere utile considerare lo sviluppo di una schermata home più compatta anziché dividere le funzionalità in schermate separate. Questo approccio potrebbe rendere l'applicazione più intuitiva ed esteticamente apprezzabile per gli utenti, riducendo il tempo necessario per navigare tra le varie sezioni e semplificando l'esperienza complessiva dell'utente. Una schermata home più compatta potrebbe includere un layout ben strutturato che fornisca un accesso rapido e diretto alle principali funzionalità dell'applicazione, migliorando così l'usabilità complessiva e l'impatto visivo dell'applicazione.

Infine, potrebbe risultare utile durante lo svolgimento della partita introdurre degli indicatori di caricamento nelle fasi di compilazione e di invio, al fine di fornire all'utente un chiaro feedback sullo stato attuale dell'applicazione. Questo permetterebbe di ridurre l'incertezza dell'utente riguardo al progresso delle operazioni in corso e migliorerebbe l'esperienza complessiva dell'utilizzo dell'applicazione.

8.4 Revisione database T4

Nel database del T4, si riscontra la presenza di alcuni attributi privi di significato nella tabella **Game**, specificamente i campi *Descrizione* e *Nome*. Si raccomanda una revisione del loro significato al fine di renderli più coerenti con il contesto attuale dell'applicazione.

8.5 Diverse modalità di gioco

Potrebbe essere utile considerare l'implementazione di una modalità "Multiplayer" che consenta a più giocatori di partecipare contemporaneamente e di sfidarsi tra loro, oltre che di competere contro il robot. Questa implementazione richiederebbe lo sviluppo di un meccanismo che permetta agli utenti di visualizzare tutti gli altri giocatori online e di sfidarli direttamente attraverso l'applicazione.

Un'opzione dedicata a questa modalità è già stata predisposta nel menu dell'applicazione, come descritto nel paragrafo dedicato, consentendo agli utenti di selezionare la modalità di gioco desiderata all'avvio, per cui è necessario effettuare solo l'implementazione vera e propria della modalità.

8.6 Consentire al giocatore di riprendere una partita già avviata

Nella modalità "Sfida un Robot", è possibile che gli utenti selezionino una classe e un robot da sfidare, confermino la selezione, avviino la partita e poi chiudano l'applicazione, magari dopo aver giocato alcuni turni o senza giocare affatto. Tale chiusura improvvisa potrebbe verificarsi anche involontariamente. Indipendentemente dalla causa, è importante notare che la partita viene comunque avviata e le relative informazioni vengono memorizzate nel database.

Per migliorare l'esperienza dell'utente e gestire efficacemente queste situazioni, potrebbe essere utile consentire agli utenti di riprendere una partita precedentemente avviata quando effettuano nuovamente il login. A tale scopo, potrebbe essere introdotto un nuovo pulsante che appare solo in queste circostanze, consentendo agli utenti di riprendere la partita interrotta.

Tuttavia, nel caso in cui un utente decida di non riprendere la partita, è possibile implementare un meccanismo che elimina tutte le informazioni relative a quella specifica partita sia dal volume T8 che dal database. Questo aiuterebbe a evitare il salvataggio di informazioni inutili e a ottimizzare l'uso delle risorse del sistema.

8.7 Gestire correttamente l'aggiunta e la rimozione delle classi da testare

Durante le fasi di testing dell'applicazione, è stato individuato un problema significativo correlato all'operazione di eliminazione di una classe tramite il tasto "delete" rosso situato nella schermata delle classi dell'amministratore. Dopo aver rimosso una classe mediante questa funzionalità, è stato osservato che il tentativo di aggiungere nuovamente una classe con lo stesso identico nome ha portato a un errore interno nell'applicazione.

Questa situazione ha creato una condizione in cui il ripristino della classe precedentemente eliminata con il medesimo nome è risultato impossibile compromettendo la funzionalità di gestione delle classi nell'ambiente dell'amministratore. Pertanto, è essenziale affrontare e risolvere questo problema al fine di garantire un'esperienza ottimale anche per l'admin.

Capitolo 9

Guida all'installazione

9.1 Docker e WSL

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. In particolare, raccoglie il software in unità standardizzate chiamate container che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime.

Per effettuare l'esecuzione di Docker in ambiente Windows è necessario utilizzare la WSL (Windows Subsystem for Linux), la quale permette di eseguire un ambiente Linux all'interno di Windows, consentendo agli utenti di utilizzare strumenti e applicazioni Linux direttamente sul sistema operativo Windows. Docker favorisce la modularità, la scalabilità e semplifica la distribuzione delle applicazioni, gestendo efficientemente le risorse. Docker permette di creare immagini e container

che operano in ambienti isolati, eseguendo i servizi separatamente e consentendo la comunicazione tra di essi tramite dei porti specifici.

9.2 Installazione

Ci sono 3 passi da seguire per l'installazione:

1. Scaricare Docker Desktop a questo indirizzo. Nel caso non sia la prima installazione, è necessario effettuare la disinstallazione, quindi utilizzare `uninstaller.bat` mentre si ha in esecuzione Docker: in questo modo si elimina qualunque file presente su Docker. In caso di errore *136 Docker desktop - unexpected wsl error* sarà necessario eseguire il comando `"wsl --shutdown"` nel terminale ed eseguire il riavvio. Se l'errore persiste, allora è consigliabile installare o aggiornare WSL all'ultima versione con il comando `"wsl --install"` (o con `"wsl --update"`).
2. Una volta scaricata la cartella del progetto, avviare lo script `installer.bat` su Windows. Su MacOS è necessario eseguire il comando `"chmod +x installermac.sh"` nella cartella in cui è presente il file `installermac.sh` e poi eseguire `"./installermac.sh"` per eseguirlo. Alla fine dell'installazione si avrà:
 - la creazione della rete `global-network` comune a tutti i container;

- la creazione dei volumi VolumeT8 e VolumeT9 per i task 1-8 e 1-9, rispettivamente;
- la creazione dei singoli container nell'applicazione Docker desktop.
- la configurazione del container *manvsclass-mongo db-1*; in particolare, lo script esegue in automatico le seguenti operazioni:

```
- 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)
```

Per l'utilizzo, è necessario avviare tutti i container ad eccezione di *ui gateway*, che dovrà essere avviato per ultimo. L'applicazione è raggiungibile sulla porta ":80".

Per esporre l'applicazione su un indirizzo pubblico si rimanda alla documentazione dei colleghi del gruppo A10.

9.3 Guida all'utilizzo

Dopo l'avvio dei container è necessario registrarsi tramite la schermata di accesso. Prima di iniziare a giocare è necessario inserire una classe di test da sottoporre a verifica e ciò può essere fatto solo dall'utente amministratore. Quindi, l'amministratore deve registrarsi al sistema in modo da poter caricare la classe di test. È fornita una spiegazione più dettagliata di seguito.

9.3.1 Registrazione Admin

L' amministratore, se è il primo accesso alla applicazione, si deve registrare nella schermata di registrazione ottenuta inserendo nella barra degli indirizzi questo indirizzo.

9.3.2 Registrazione utente

Se l'utente è al primo utilizzo dell' applicazione, deve digitare l'url `http://localhost/login` per accedere alla schermata di login e cliccare sulla dicitura "*Non sei ancora registrato? Registrati*". A questo punto è possibile effettuare la registrazione inserendo nome, cognome, email e password, la quale deve contenere un carattere speciale, una lettera maiuscola, una minuscola ed essere lunga almeno 8 caratteri.

N.B. Sono presenti dei video tutorial per le due registrazioni in caso di ulteriori problemi messe a disposizione da altri gruppi.

9.4 Modificare il codice

Si noti che nel caso fosse necessario visualizzare in maniera pratica eventuali modifiche effettuate sul codice, sarà necessario seguire 11 passaggi:

1. recarsi nell'applicazione Docker;
2. aprire la sezione relativa ai containers;
3. selezionare tutti i containers relativi ai file modificati;
4. effettuare la delete di tali containers;
5. aprire la sezione images;
6. selezionare le immagini relative ai file modificati;
7. effettuare l'eliminazione di tali immagini;
8. recarsi sull'IDE utilizzato ed aprire il terminale integrato della cartella in cui è presente il file "pom.xml" relativo ai file modificati;
9. eseguire il comando "mvn clean package";
10. fare clic sul tasto destro sul file "docker-compose.yml" e selezionare "compose up";
11. riavviare i container.

9.5 Problematiche di utilizzo

In seguito si elencano alcune delle problematiche riscontrate durante l'utilizzo dell'applicazione al fine di agevolarne l'utilizzo per i gruppi successivi.

9.5.1 Browser Cache

Disattivando la cache del browser, è possibile garantire che ogni modifica apportata al codice sorgente o alle risorse dell'applicazione sia immediatamente visualizzata nel browser. La cache del browser può causare problemi di compatibilità e disattivarla semplifica il processo di debug, consentendo agli sviluppatori di identificare e risolvere rapidamente i bug senza dover preoccuparsi di eventuali caching persistenti che potrebbero mascherare il problema. È possibile disattivare la cache aprendo la schermata *ispezione elemento*, andando nella sezione *Rete* e selezionando la casella *Disattiva Cache*.

9.5.2 Versione Docker

Durante lo sviluppo dell'applicazione, è emersa una problematica legata alla versione di Docker. È stato osservato che nelle versioni più recenti di Docker, dopo una serie di disinstallazioni e reinstallazioni dell'applicazione, potrebbero verificarsi problemi durante il download quando si avvia il file *installer.bat*. Nel caso in cui si riscontrino errori

come quello mostrato nella figura 9.1, potrebbe essere necessario disinstallare Docker e utilizzare una versione non successiva alla **4.25.1** evitando così errori nella costruzione dei container.

```
> [app internal] load build context:
failed to solve: changes out of order: "target/t5-0.0.1-SNAPSHOT.jar" ""
"Errore nell'installazione del Task "/T5-G2/t5""
"Installazione in corso in "/T6-G12/T6""
2024/02/03 01:10:07 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 1.2s (6/7)                                docker:default
=> [app internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 395B 0.0s
=> [app internal] load metadata for docker.io/library/openjdk:17-jdk-alpine 1.0s
=> [app internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [app internal] load build context 0.0s
=> => transferring context: 2B 0.0s
=> CANCELED [app 1/3] FROM docker.io/library/openjdk:17-jdk-alpine@sha256:4b6abae565492dbe9e7a894137c966a7485154 0.1s
=> => resolve docker.io/library/openjdk:17-jdk-alpine@sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9 0.0s
=> => sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9dbd9784383d81 319B / 319B 0.0s
=> => sha256:a996cdcc940704ec6badaf5fecf1e144c096e00231a29188596c784bcf858d05 951B / 951B 0.0s
=> => sha256:264c9bdce361556ba6e685e401662648358980c01151c3d977f0fd777f7c26ab 3.48kB / 3.48kB 0.0s
=> ERROR [app 2/3] COPY T6/target/T6-0.0.1-SNAPSHOT.jar /app/T6-0.0.1-SNAPSHOT.jar 0.0s
> [app 2/3] COPY T6/target/T6-0.0.1-SNAPSHOT.jar /app/T6-0.0.1-SNAPSHOT.jar:
failed to solve: failed to compute cache key: failed to calculate checksum of ref cf5b3da7-61d5-491e-a26f-69068da5a976::
aidot7yo8g23e0ppjeyciuarh: failed to walk /var/lib/docker/tmp/buildkit-mount3641415463/T6/target: lstat /var/lib/docker/
tmp/buildkit-mount3641415463/T6/target: no such file or directory
```

Figura 9.1: Errore *installer.bat*

9.5.3 Porte già in uso

Durante l'avvio dell'applicazione, potrebbe verificarsi un problema in cui alcuni container non si avviano correttamente, mostrando un messaggio di errore relativo alla porta già in uso. Un esempio comune potrebbe riguardare il container T4, poiché la porta 3000 è comunemente utilizzata dal processo *MySQL80*. In questo caso, è necessario aprire la schermata dei servizi attivi su Windows per individuare e terminando il processo che sta attualmente utilizzando la porta in questione. In alternativa, è possibile risolvere questo problema modificando la porta del container in questione per evitare conflitti.

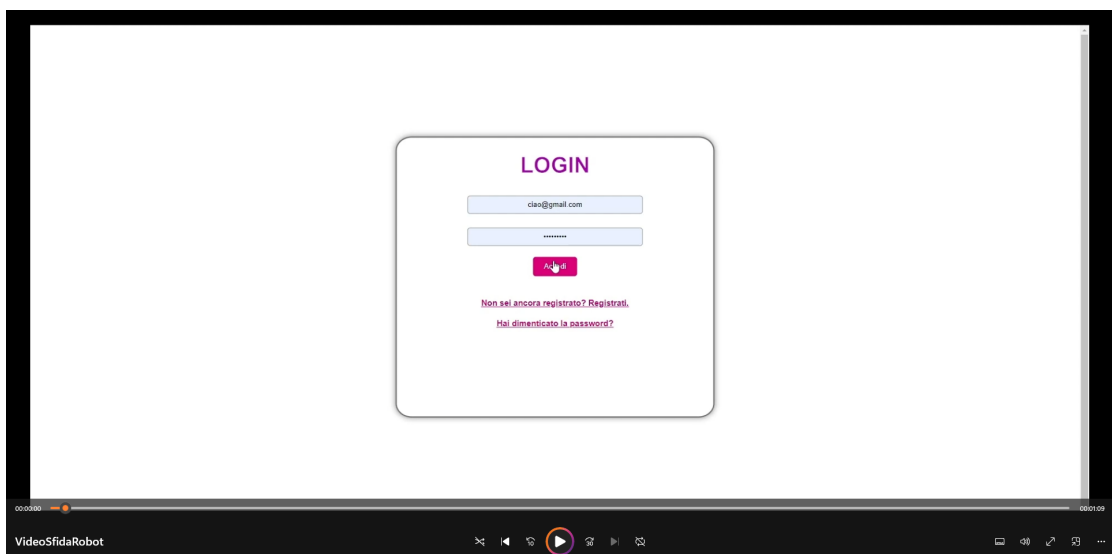
9.5.4 502 Bad Gateway

Occasionalmente, anche dopo che tutti i container sono stati avviati con successo, la pagina web potrebbe visualizzare un messaggio di errore del tipo *502 Bad Gateway*. Una possibile ragione dietro questo tipo di errore potrebbe essere correlata alla rapidità con cui si tenta di accedere all'applicazione web subito dopo il riavvio dei container. In questi casi, la piattaforma potrebbe richiedere del tempo per completare il processo di avvio e stabilire la connessione corretta tra i vari componenti dell'applicazione. Attendere alcuni istanti prima di tentare di accedere all'applicazione può spesso risolvere il problema. Tuttavia, se l'errore persiste nonostante l'attesa, è consigliabile effettuare il riavvio di Docker.

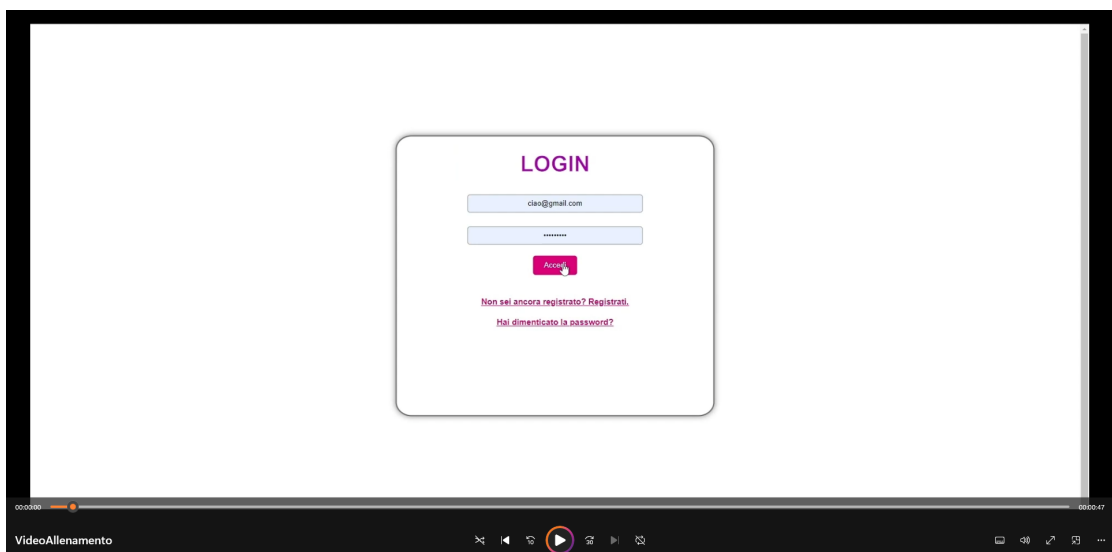
9.6 Video Funzionamento Web App

Di seguito riportiamo i link ai video dimostrativi delle due modalità di gioco attualmente disponibili. **N.B.** Nel caso in cui i link non funzionassero, i video sono disponibili nella repository del progetto.

9.6.1 Funzionamento modalità "Sfida un Robot"



9.6.2 Funzionamento modalità "Allenamento"



Glossario

Robot

Si riferisce a strumenti di generazione automatica di test (Randoop ed EvoSuite). Rappresentano gli avversari dei giocatori.

Turno

È l'unità indivisibile di un round e rappresenta le azioni di un giocatore durante il round. In particolare, il giocatore avvia una partita selezionando il suo sfidante e la classe di test.

Round

È l'entità che contiene le informazioni di gioco principali. Ogni game è composto da un numero finito di round durante il quale ciascun giocatore effettua la propria giocata.

Game

È l'entità principale del sistema. Un game è composto da più round.

Giocatore

È l'utente registrato che sfida il Robot eseguendo un turno nei round di un game.

Admin

Un utente registrato come amministratore di sistema; partecipa all'applicazione effettuando il caricamento di Classi e Test.