



Documentazione A16

Anno Accademico 2023/2024

Docente

Anna Rita Fasolino

Studenti

Antonio Bocchetti M63001259

Gianluigi Erra M63001261

Antonio Savino M63001318

Stefano Violante M63001263

Indice

Introduzione	1
1 Progettazione	2
1.1 Descrizione del progetto	2
1.2 Descrizione dei requisiti assegnati	2
1.2.1 Modalità "Sfida tutti i Robot"	3
1.2.2 "Ranking" e "Storico"	4
1.3 Stato iniziale	4
1.3.1 Architettura a microservizi	5
1.3.2 Punto di partenza	7
1.4 Architettura complessiva	7
1.4.1 Diagramma dei componenti complessivo	7
1.5 Diagramma dei casi d'uso	9
1.6 Workflow Diagram	11
1.7 Scenari di utilizzo	12
1.7.1 Scenario 1 - BossRush	12
1.7.2 Scenario 2 - Ranking	14
1.7.3 Scenario 3 - Storico	15
2 Sviluppo	17
2.1 Metodologia AGILE e Kanban	17
2.2 Strumenti utilizzati	19
2.2.1 Discord	19
2.2.2 Miro	19
2.2.3 Docker	20

2.2.4	GitHub	20
2.2.5	Visual Studio Code	20
2.2.6	Maven	21
2.2.7	Postman	21
2.2.8	Selenium	22
2.2.9	DBeaver	23
3	Integrazione	24
3.1	Modalità Boss Rush	25
3.1.1	Modifiche apportate a T5	26
3.1.2	Modifiche apportate a T6	35
3.1.3	Modifiche apportate a T4	39
3.1.4	Modifiche apportate a T8	41
3.2	Storico	43
3.2.1	Modifiche apportate a T5	44
3.2.2	Modifiche apportate a T4	47
3.3	Ranking	55
3.3.1	Modifiche a T5	55
3.3.2	Modifiche a T4	56
3.4	Requisito aggiuntivo: Redirect dopo registrazione . .	56
3.4.1	Modifiche a T23	57
4	Deployment	58
4.1	Integrazione con la nuova versione	59
4.1.1	Analisi dei file della nuova versione	60
4.1.2	Corretta gestione degli URL	60
5	Testing	61
5.1	Testing con Postman	62
5.2	Verifica della coerenza dei dati tramite DBeaver . . .	65
5.3	Testing automatizzato con Selenium	66
5.3.1	Requisiti per il testing automatizzato	70

6	Sviluppi Futuri	71
6.1	Login Admin	71
6.2	Registrazione Admin	72
6.3	User Friendliness	72
6.4	Diverse modalità di gioco	72
6.5	Riprendere una partita già avviata	73
7	Installazione e Problemi Noti	75
7.1	Docker e WSL	75
7.2	Guida all'Installazione	76
7.3	Modificare il Codice	77
7.4	Problematiche Note	78
7.4.1	Versione di Ubuntu	78
7.4.2	Cache Browser	78
7.4.3	Porte in uso	79
7.4.4	502 Bad Gateway	79
	Glossario	81

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

1.1 Descrizione del progetto

Il gioco si basa sul testare in maniera automatizzata classi Java servendosi di JUnit; in particolare, i Robot utilizzati sono *Randoop* ed *EvoSuite*. Lo scopo del gioco è raggiungere un determinato livello di copertura di righe di codice al fine di superare quella dei Robot. Ogni gruppo ha il compito di implementare alcuni requisiti assegnati dalla Docente e produrre una documentazione di quanto realizzato.

1.2 Descrizione dei requisiti assegnati

Il requisito assegnatoci verte sul modificare il lavoro del gruppo A7 in modo che vengano integrate le funzionalità presenti nel progetto del gruppo A6. La versione di A7 prevede tre modalità di gioco:

"Sfida un Robot", "Allenamento" e "Multiplayer" (da implementare). A6, oltre alla classica "Sfida un Robot", presenta la modalità "Sfida tutti i Robot" (detta anche *BossRush*) e le pagine "Ranking" (classifica generale) e "Storico" (classifica utente).

1.2.1 Modalità "Sfida tutti i Robot"

Come già affermato, la versione originale di A7 consente al giocatore di avviare le singole istanze di partita sfidando un solo robot per volta, scegliendone il livello di difficoltà (inteso come livello di copertura). Con l'implementazione della modalità "Sfida tutti i Robot" si porta il gioco ad un nuovo livello di sfida. Inoltre, ad ogni turno giocato, sarà mostrato un *alert* in cui è specificato quanti e quali Robot sono stati battuti con le relative percentuali di copertura.

L'aggiunta di questa modalità è stata possibile studiando approfonditamente il codice e la documentazione del gruppo A6, prestando attenzione particolare al Task T5. La modifica di quest'ultimo, per inglobare le funzionalità del gruppo A6, ha sottoposto il nostro lavoro ad una sfida di consistenza e robustezza del codice. La gestione architetturale del progetto è a microservizi, ovvero ogni servizio viene svolto in maniera indipendente in ambiente controllato e containerizzato; l'esistenza di un network globale sottointende anche una coesistenza delle dipendenze tra diversi servizi, quindi la modifica di

un solo container deve essere sempre doppiamente controllata dalla persistenza del funzionamento di tutti i requisiti nati e sviluppati da altri gruppi.

1.2.2 "Ranking" e "Storico"

Le pagine "Ranking" e "Storico" sono state aggiunte da A6 e rappresentano due tipi di classifiche. La prima riguarda la classifica generale di tutti gli utenti registrati alla piattaforma e mostra il numero di partite giocate e vinte, mentre la seconda si riferisce alle partite del singolo utente autenticato in quel momento e mostra tutte le partite disputate, sia vinte che perse, ed i rispettivi punteggi. In particolare, nel caso della modalità BossRush, verrà mostrata la percentuale di copertura raggiunta ed il numero di Robot sconfitti.

1.3 Stato iniziale

Si procede con l'illustrare lo stato iniziale del progetto da cui si è partiti, allo scopo di offrire una comprensione dettagliata della struttura. Il punto di partenza assume un ruolo cruciale nel contesto dell'evoluzione architettuale, attraverso questa, infatti, 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, in figura 1.1 si evidenziano in giallo i task che sono stati modificati per il completamento dei requisiti.

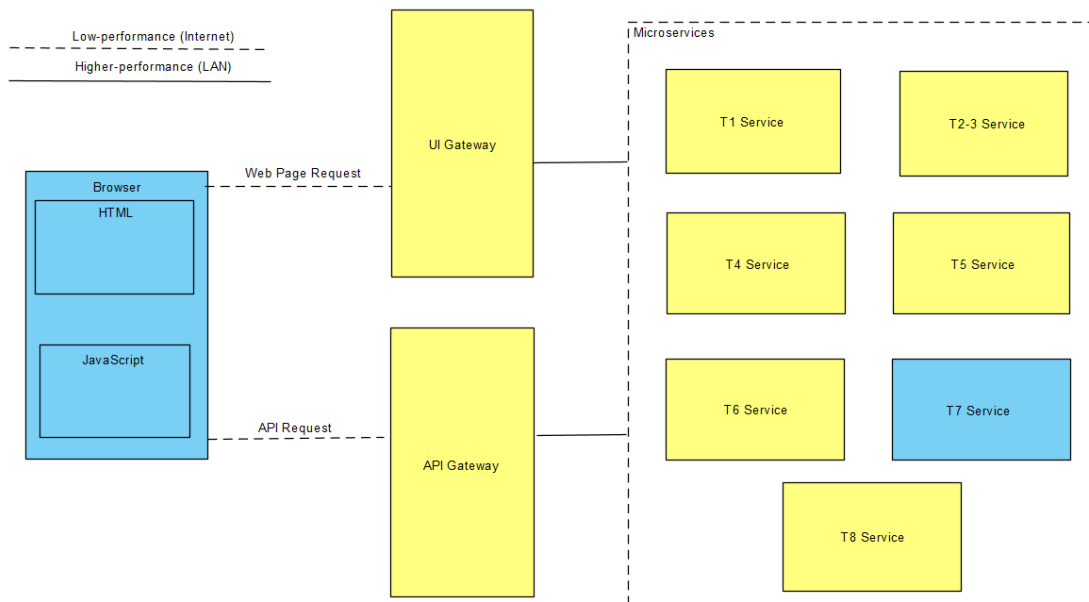


Figura 1.1: Architettura a microservizi.

L'**UI Gateway** ha il compito di effettuare il routing delle richieste http relative al front-end. 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 sono descritte le modifiche svolte sul progetto A7-2024 al fine di integrare le funzionalità di A6-2024. I colleghi di entrambi i gruppi sopracitati hanno lavorato, rispettivamente, sulla riorganizzazione del File System e dei volumi, la memorizzazione del punteggio di una partita giocata dall'utente e l'aggiunta della modalità Allenamento (A7), e sull'inserimento della modalità di gioco *BossRush* per sfidare tutti i robot.

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 in figura 1.2, evidenziando in giallo i componenti che hanno subito delle modifiche. Di seguito è fornita una spiegazione dei componenti modificati:

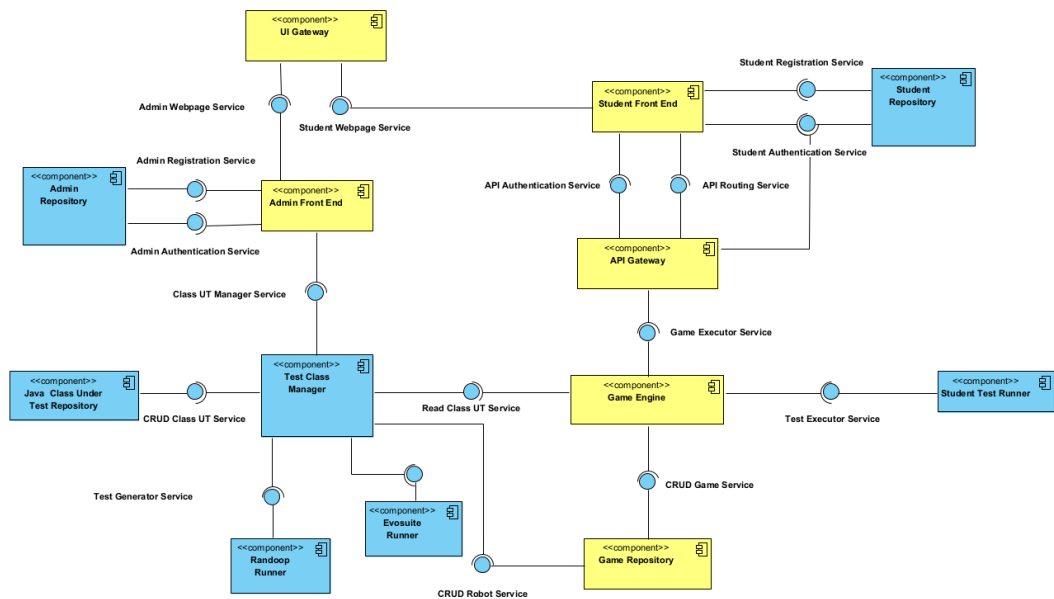


Figura 1.2: Diagramma dei componenti.

Admin Front-End

Questo componente fornisce un'interfaccia all'utente Admin per la visualizzazione e l'aggiunta delle classi da testare e degli utenti registrati.

UI Gateway

L'UI Gateway ha il compito di effettuare il routing delle richieste http relative al front-end.

API Gateway

L'API Gateway si occupa del routing e della gestione dell'autenticazione e autorizzazione per le richieste relative alle API del sistema.

Student Front-End

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

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.

1.5 Diagramma dei casi d'uso

Il diagramma dei casi d'uso visualizza le interazioni tra attori e il sistema, offrendo una panoramica delle funzionalità. Fondamentale per la comprensione delle implementazioni, facilita la comunicazione, supporta l'analisi dei requisiti e fornisce una base per la progetta-

zione dell'architettura del software. In figura 1.3 mostriamo tale diagramma evidenziando in giallo i casi d'uso modificati e, in verde, quelli aggiunti per permettere le varie integrazioni.

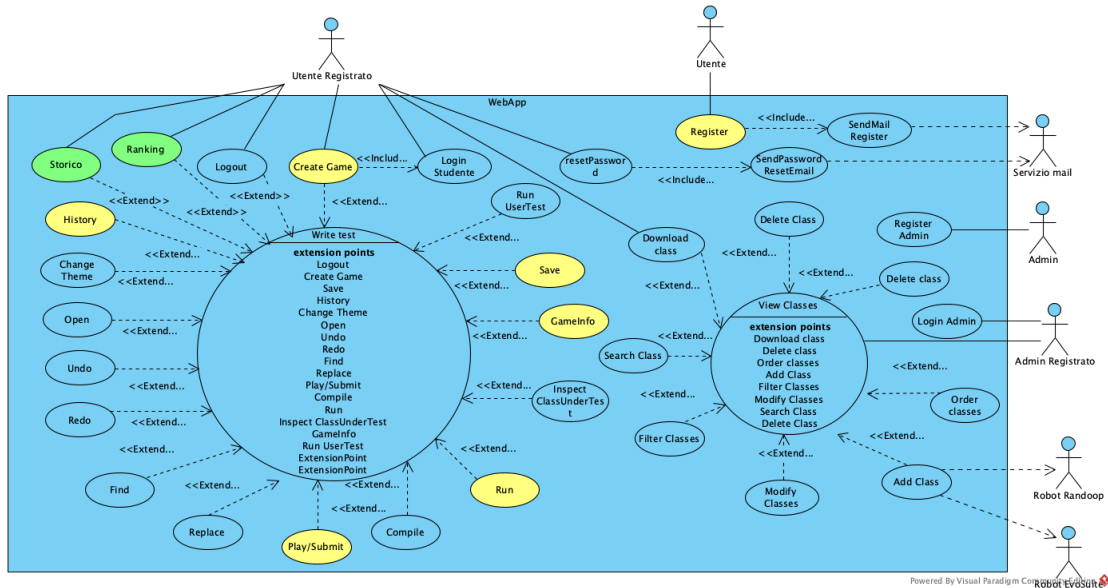


Figura 1.3: Diagramma dei casi d'uso.

Nello specifico, il diagramma dei casi d'uso mostra l'aggiunta delle funzionalità "Storico", in cui l'utente può visualizzare lo storico delle sue partite (nel sistema dovranno essere chiaramente distinte le partite in base alla modalità di gioco), e "Ranking", dove l'utente può visualizzare la classifica generale (ordinata in base al numero di vittorie) dei giocatori registrati alla piattaforma. Inoltre, sono stati modificati i casi d'uso "Play/Submit" e "Run" al fine di permettere il corretto funzionamento nella modalità "Sfida tutti i Robot". E' stato modificato il tasto "GameInfo" che mostra i dati della partita (*ID Utente*, *GameID*, *difficoltà* ecc.), in quanto le informazioni risultava-

no duplicate. Infine, è stata aggiunta una funzione in "Register" per effettuare un redirect alla pagina di login a seguito di registrazione avvenuta con successo.

1.6 Workflow Diagram

Al fine di descrivere il funzionamento del sistema, in figura 1.4 viene mostrato il diagramma di flusso di alto livello. Dopo aver effettuato il login, l'Utente registrato può effettuare una nuova partita oppure visualizzare la classifica generale (Ranking) o il proprio Storico. Se intende giocare, può farlo nella modalità classica (selezionando poi il robot specifico) oppure BossRush. Confermate le sue scelte, si procede alla scrittura del test della classe nell'editor. Entrambe le modalità prevedono la compilazione della classe di test

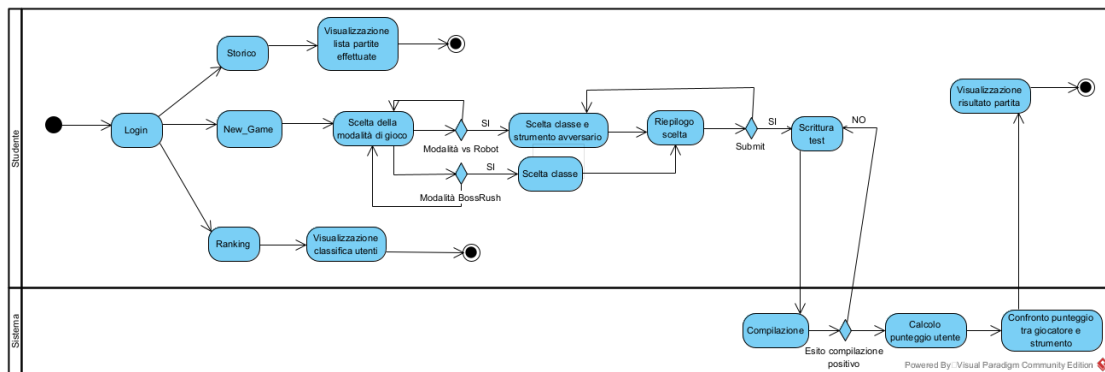


Figura 1.4: Diagramma di flusso.

e, in caso di esito positivo, si passa al calcolo del punteggio utente. Successivamente il sistema confronterà la percentuale di copertura

raggiunta dal nostro test e dal Robot e sarà mostrato il risultato della partita.

1.7 Scenari di utilizzo

Gli scenari di utilizzo descrivono interazioni specifiche tra utenti e sistema, cruciali durante l'analisi dei requisiti e la progettazione del software. Apportano vantaggi come l'identificazione dei requisiti funzionali, la validazione e la guida alla progettazione dell'interfaccia utente, essenziali per comprendere le esigenze degli utenti e orientare lo sviluppo del sistema. Si illustrano, nelle successive sezioni, gli scenari relativi alla modalità BossRush, alla visualizzazione del Ranking ed alla visualizzazione dello Storico. Si è preferito evitare ridondanze in questo paragrafo, dato che la documentazione dettagliata fornita dal gruppo A7 sugli scenari di utilizzo è già esaustiva. Tali scenari sono spiegati in maniera completa nel capitolo 4 della loro documentazione ufficiale.

1.7.1 Scenario 1 - BossRush

Attore principale: Utente Registrato

Trigger: L'utente Registrato sta svolgendo una partita nella modalità "BossRush" e preme il pulsante "Play/Submit"

Precondizioni: L'Utente ha eseguito l'accesso al sistema, diventan-

do Utente Registrato, ha scelto di effettuare una nuova partita e ha selezionato la modalità "BossRush" e la classe da testare

Flusso principale:

1. L'Utente Registrato preme il bottone "Compile" per eseguire la compilazione del codice da lui scritto
 - **scenario alternativo:** il codice somministrato presenta errori di compilazione
 - Il sistema mostra un errore di compilazione e la partita non può essere continuata
2. L'Utente Registrato preme il bottone "*Play/Submit*"
3. Il sistema elabora l'azione, calcola le percentuali di copertura del codice ottenute dal giocatore e recupera quelle dei robot
4. Il sistema mostra un messaggio con i risultati della sfida, indicando la vittoria parziale, totale o sconfitta del giocatore

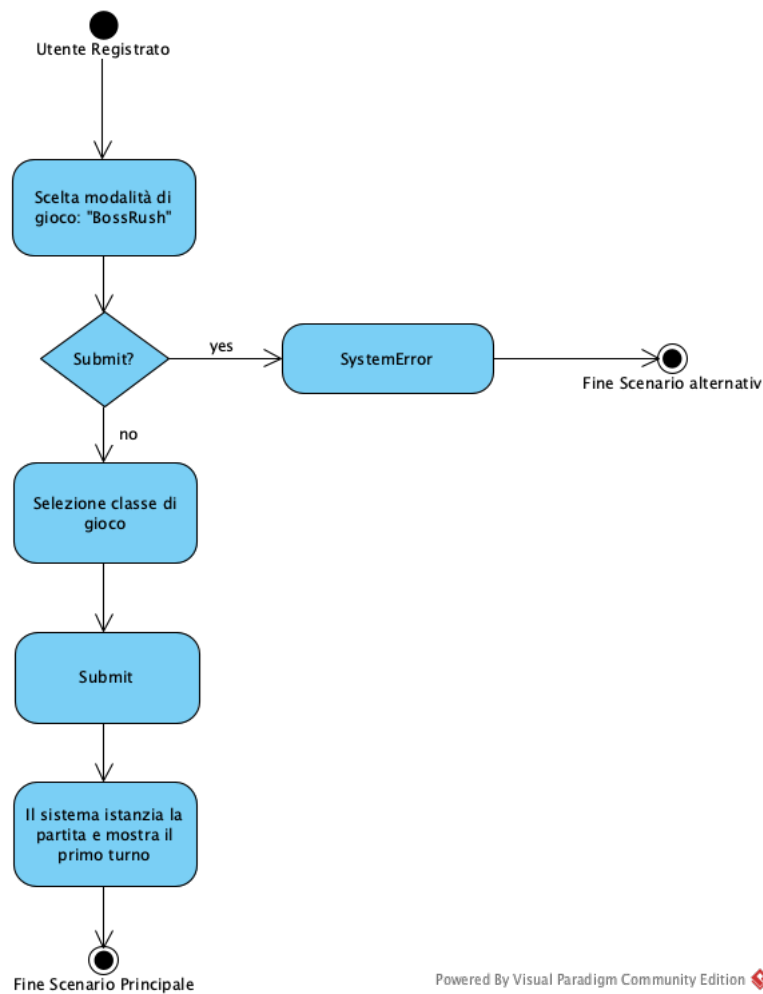


Figura 1.5: Activity Diagram per la modalità BossRush.

1.7.2 Scenario 2 - Ranking

Attore principale: Utente Registrato

Trigger: L'Utente Registrato è nella schermata principale del sistema (*main.html*) e preme il pulsante "*Ranking*"

Precondizioni: L'Utente ha eseguito l'accesso al sistema, diventando Utente Registrato

Flusso principale:

1. L'Utente Registrato preme il pulsante "*Ranking*"

2. Il sistema mostra la classifica degli utenti registrati
3. Il Ranking mostra la classifica ordinata in base al numero di vittorie



Figura 1.6: Activity Diagram per funzionalità Ranking.

1.7.3 Scenario 3 - Storico

Attore principale: Utente Registrato

Trigger: L'Utente Registrato è nella schermata principale del sistema (*main.html*) e preme il pulsante "*Storico*"

Precondizioni: L'Utente ha eseguito l'accesso al sistema, diventando Utente Registrato

Flusso principale:

1. L'Utente Registrato preme il pulsante "*Storico*"
2. Il sistema mostra una lista, eventualmente vuota, con lo storico delle partite giocate dall'utente
3. Lo storico rende evidente la distinzione tra i dati ottenuti dalla modalità classica e la modalità contro tutti i robot



Figura 1.7: Activity Diagram per funzionalità Storico.

Capitolo 2

Sviluppo

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

2.1 Metodologia AGILE e Kanban

Apprese le differenze iniziali dei due progetti, si è scelto di utilizzare, per lo sviluppo del Task assegnatoci, un approccio **AGILE** basato su **Kanban**. La "metodologia agile" è un approccio allo sviluppo del software basato sulla distribuzione continua di software efficienti creati in modo rapido e iterativo; non prevede regole da rispettare tassativamente nell'ambito dello sviluppo del software ma si tratta di un tipo di approccio alla collaborazione e ai flussi di la-

voro fondato su una serie di valori in grado di guidare il modo di procedere. La metodologia di sviluppo AGILE permette, quindi, di focalizzarsi sul codice anziché sul design e sulla documentazione.

In particolare, Kanban è un framework che aiuta a visualizzare il lavoro, massimizzare l'efficienza e migliorare continuamente. Il lavoro viene rappresentato su Kanban Board, consentendo di ottimizzare la consegna dei task tra più team, gestendo anche i progetti più complessi in un unico ambiente. Nel nostro caso è stato utilizzato per tener traccia del lavoro da svolgere, in sviluppo e terminato, come mostrato nella Figura 2.1.

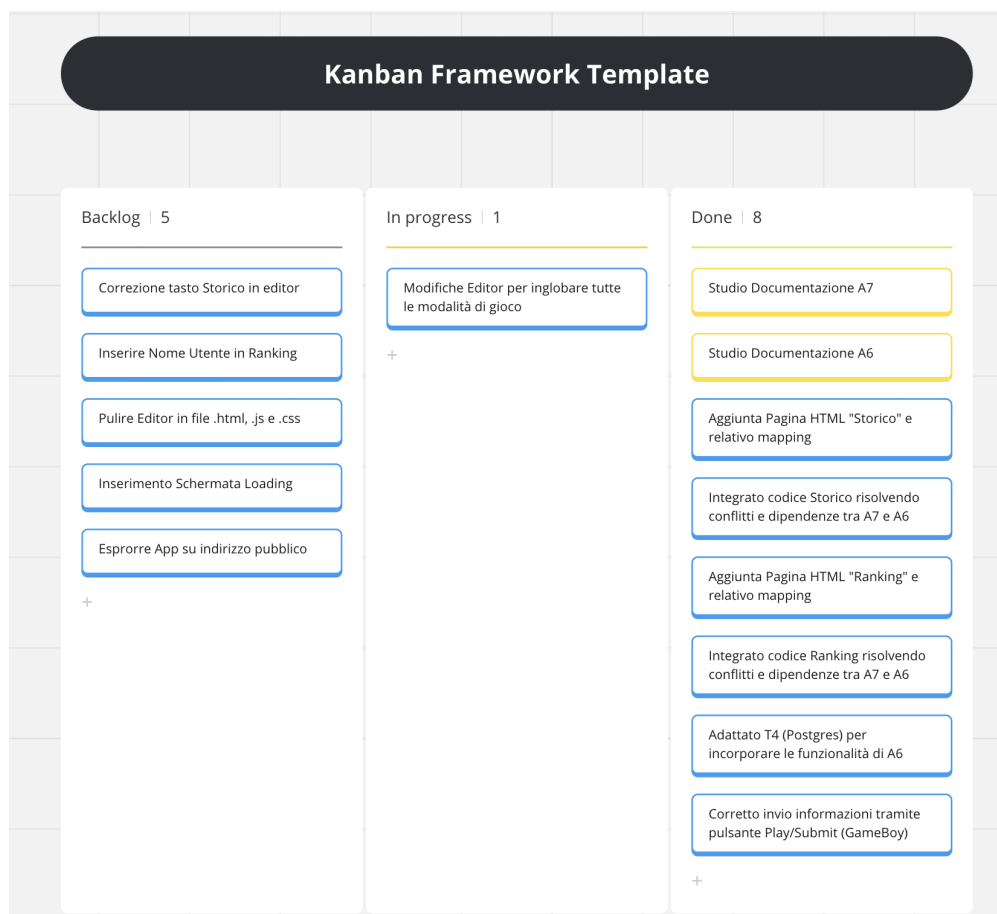


Figura 2.1: Lavagna Kanban di A16 durante il lavoro.

2.2 Strumenti utilizzati

Al fine di realizzare quanto richiesto, ci siamo serviti di alcune piattaforme illustrate nei successivi paragrafi.

2.2.1 Discord

Discord è un servizio di comunicazione vocale, video e testuale che è stato usato principalmente per la sua funzione di condivisione schermo, consentendoci di lavorare insieme durante i momenti più ostici.

2.2.2 Miro

Miro è una piattaforma online innovativa che offre a team e individui uno spazio di lavoro flessibile e intuitivo per la pianificazione collaborativa e la visualizzazione delle idee. Combina la comodità e l'accessibilità di una piattaforma online con la versatilità e la potenza di una lavagna tradizionale. Con Miro, gli utenti possono creare un numero illimitato di lavagne per catturare e organizzare idee, pianificare progetti e ottimizzare i flussi di lavoro. È stato applicato, inoltre, un framework chiamato *Kanban* utilizzato per implementare lo sviluppo di software Agile.

2.2.3 Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker 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. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito.

2.2.4 GitHub

GitHub è un servizio web e cloud-based che, tramite la funzione del controllo delle versioni, aiuta gli sviluppatori ad archiviare e gestire il loro codice e a tracciare e controllare le modifiche.

2.2.5 Visual Studio Code

Visual Studio Code è un editor di codice sorgente leggero ma potente che viene eseguito sul desktop ed è disponibile per Windows, macOS e Linux. Include il supporto predefinito per *JavaScript*, *TypeScript* e *Node.js* e offre un ecosistema ricco di estensioni per altri linguaggi e runtime.

2.2.6 Maven

Apache Maven è uno strumento di automazione build usato principalmente per i progetti Java, sviluppato e gestito dalla Apache Software Foundation. Rende più facile gestire e mantenere grandi progetti fornendo una struttura coerente e una serie di convenzioni su come organizzare il progetto, aiutando gli sviluppatori ad automatizzare il processo di build, test e distribuzione del software. Una delle caratteristiche principali di Maven è la sua capacità di gestire le dipendenze. Maven tiene traccia di tutte le librerie e di altre dipendenze di cui un progetto ha bisogno e le scarica automaticamente quando sono necessari. Ciò rende facile per gli sviluppatori utilizzare librerie esterne nei loro progetti senza doverle scaricare e gestirle manualmente.

2.2.7 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.

2.2.8 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 efficiente e ripetibile di test funzionali su applicazioni web. Nello specifico, Selenium consente la simulazione di azioni quali il clic su elementi, inserimento di testo, invio di moduli, ecc. Inoltre, offre un supporto multi-piattaforma in quanto compatibile con diversi browser web e 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.

2.2.9 DBeaver

DBeaver è un'applicazione software client SQL ed 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.

Capitolo 3

Integrazione

Il task assegnatoci prevedeva l'integrazione tra le features introdotte dal gruppo A6 e la struttura presentata dal gruppo A7. Il lavoro di A6 presenta tre funzionalità, in particolare:

- Modalità BossRush
- Pagina Storico
- Pagina Ranking

La Modalità Boss Rush prevede la realizzazione di una nuova modalità di gioco nella quale il giocatore si scontrerà con tutti i robot appartenenti ad una determinata classe (Calcolatrice o VCardBean). La pagina Storico prevede la creazione di una pagina che elenchi tutte le partite, con i relativi esiti, associate al giocatore che ha attualmente effettuato il log-in. La pagina Ranking prevede l'implementazione di una pagina che permetta di mostrare la classifica di tutti i gioca-

tori elencati per partite vinte.

Il lavoro di A6, inoltre, presenta una sensibile diversità di codice rispetto a quello di A7, che gode di un'ottima struttura e leggibilità. Questo ha reso più difficile l'integrazione in quanto si è dovuto mediare tra le due strutture mantenendo l'ordine di A7, integrato, però, con le funzionalità sopracitate. Elenchiamo, dunque, le singole modifiche effettuate, specificando per ciascuna di esse quali task sono stati modificati.

3.1 Modalità Boss Rush

La modalità Boss Rush è il principale requisito di implementazione del gruppo A6 ed è la principale fonte di differenza di codice tra i due gruppi, quindi oggetto del nostro task. Abbiamo dedicato quindi all'integrazione di questa modalità particolare attenzione, soprattutto considerando che le modifiche associate a questa integrazione hanno un effetto anche sulle successive integrazioni (Ranking e Storico). Iniziamo, quindi, col vedere quali task sono stati oggetto di modifica per permettere l'integrazione di questo requisito, ossia: T5, T6, T4 e T8.

3.1.1 Modifiche apportate a T5

Ci apprestiamo ad elencare le modifiche apportate al task T5 e la modalità di integrazione impiegata.

3.1.1.1 `new_game`

Iniziamo dalla pagina **`new_game.html`**, pagina alla quale si viene reindirizzati laddove si decida di iniziare una partita dalla pagina **`main.html`**.

Nell'andare ad effettuare l'integrazione abbiamo deciso di adottare l'interfaccia di A7 anziché quella di A6 poiché l'abbiamo ritenuta più gradevole. Il risultato delle nostre modifiche ha portato, quindi, l'interfaccia della pagina `main.html` di A7 con il rimando alla modalità `BossRush` proposta dal lavoro di A6. Per effettuare l'integrazione non abbiamo fatto altro che aggiungere, nel file **`main.js`**, la funzionalità "Sfida tutti i Robot" alla funzione "selectGameMode"; abbiamo poi dovuto aggiungere il mapping `"/all_robots"` al file **`GuiController`**, definito in T6, che permette di mappare la pagina `html` correttamente.

La pagina **`new_game.html`** si presenta come in figura 3.1

Si nota l'introduzione della sezione "Sfida tutti i Robot" che denota la modalità `BossRush`, precedentemente non presente in A7.

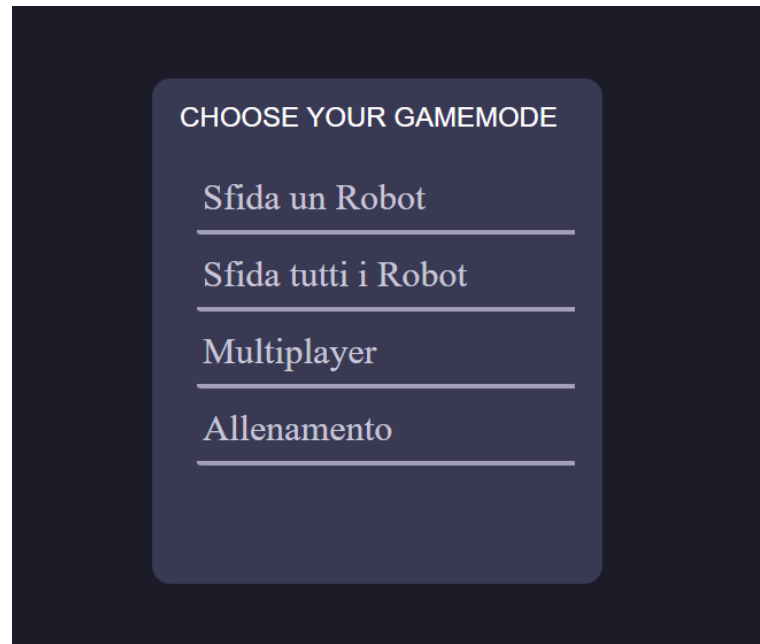


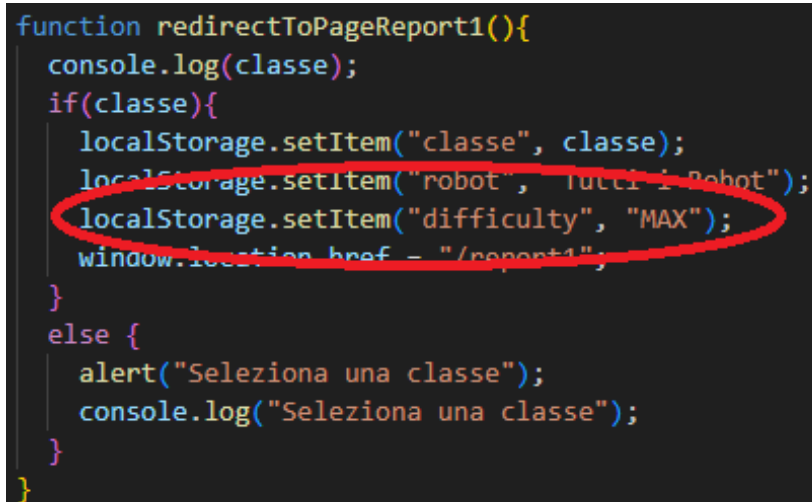
Figura 3.1: new_game.html

3.1.1.2 Modalità Sfida tutti i Robot

Il primo vero passo dell'integrazione della modalità BossRush corrisponde nello specificare la pagina html di riferimento. L'integrazione consiste nella semplice introduzione della pagina html definita dal gruppo A6 "**all_robots.html**". Questa pagina permette di selezionare solo la classe con la quale vogliamo scontrarci: mentre nel caso della modalità 1VS1 possiamo selezionare il Robot con la relativa difficoltà, in questa modalità dobbiamo obbligatoriamente scontrarci contro tutti.

L'integrazione effettuata non si è limitata solo all'inserimento della pagina html sopracitata, ma abbiamo dovuto anche risolvere una problematica relativa al codice di A6: nella loro implementazione, la difficoltà della partita non veniva mai calcolata poiché non viene

mai chiamata la funzione "HandleButtonRobot" definita all'interno di **main.js**. Questo comportava una non corretta visualizzazione della difficoltà all'interno della pagina Storico. Abbiamo quindi scelto che ogni volta che viene selezionata la modalità sopracitata, la difficoltà sarà settata sempre su "MAX", a rappresentare il fatto che stiamo combattendo contro tutti i robot disponibili per quella specifica classe. Questo viene fatto dopo aver cliccato su "Play against all Robots", mediante un redirect che viene effettuato su un'ulteriore nuova pagina html: **report1.html**. Ref: 3.2.



```
function redirectToPageReport1(){
  console.log(classe);
  if(classe){
    localStorage.setItem("classe", classe);
    localStorage.setItem("robot", "tutti i Robot");
    localStorage.setItem("difficulty", "MAX");
    window.location.href = "/report1";
  }
  else {
    alert("Seleziona una classe");
    console.log("Seleziona una classe");
  }
}
```

Figura 3.2: Estratto di codice main.js

Quest'ultima pagina non fa altro che fornirci un riepilogo della modalità e della classe scelti per poi permetterci di giocare premendo "Submit". La pagina in questione è stata integrata direttamente dal codice fornito dal gruppo A6.

3.1.1.3 editor

L'editor rappresenta la fonte più cospicua di modifiche apportate al progetto di A7 relativo al task T5. Il gruppo A6 impiega nel proprio progetto un editor contenuto tutto all'interno del file **editor.html**. Qui dentro inseriscono tanto il codice html, quanto quello JS e CSS associati all'editor; in maniera opposta l'editor di A7 è caratterizzato da una migliore "pulizia" di codice. Ci siamo posti di fronte alla scelta di andare a modificare l'editor di A7 integrando la modalità BossRush oppure utilizzare l'editor di A6 e "pulirlo" nel codice, rimuovendo anche alcuni bug; abbiamo optato per la seconda scelta. Siamo quindi partiti dal file **editor.html** del gruppo A6 e l'abbiamo scompattato negli attuali file **editor.html**, **editor.js** e **editor.css** presenti nel progetto. Questa modifica comportava la perdita del loading screen inserito dagli A7 nel contesto del loro progetto. Si è, quindi, provveduto all'inserimento di questo anche nei file nuovamente realizzati. Una raffigurazione grafica del flusso di operazioni che vengono svolte front-end nell'editor può essere ottenuta mediante l'activity diagram presentato in figura 3.3.

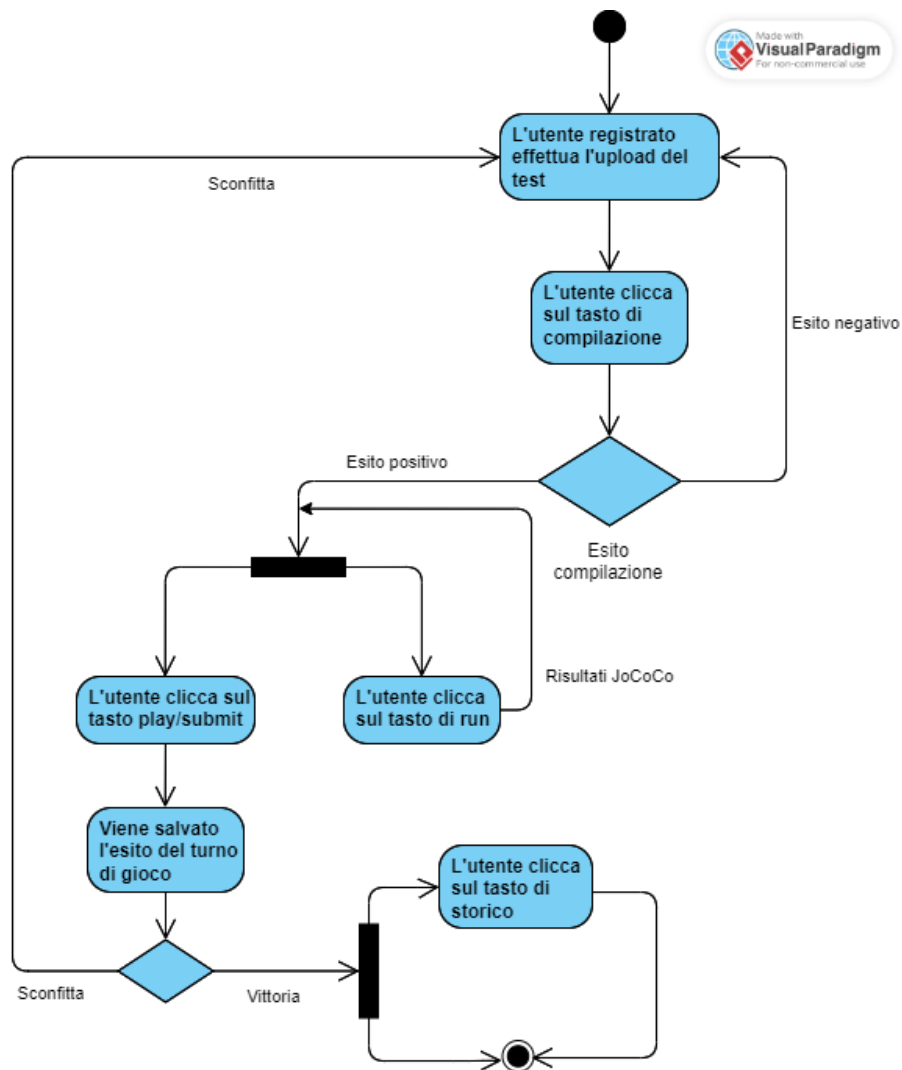


Figura 3.3: Activity Diagram editor

Abbiamo provveduto inoltre a risolvere alcuni bug tralasciati precedentemente dai nostri colleghi:

- Esito partita: Precedentemente l'esito nel caso di una partita 1VS1 non veniva mostrato correttamente, mostrando sempre la sconfitta del giocatore. In figura 3.4 mostriamo lo snippet di codice che risolve tale bug.

```

else {
    perc_robot = response.robotScore.toString();

    alert(response.win == "true" ? "Hai vinto!" : "Hai perso!"); //A16 prima era true senza apici
}
if(response.win == "true"){ //A16 prima era true senza apici
    turno++; // incremento il numero di turno giocati fino ad ora

    localStorage.setItem("gameId",null); // setto gameId null invece che tutto
}

```

Figura 3.4: Soluzione al bug esito partita

- API Run: abbiamo inserito all'interno del JSON anche lo username, richiesto per l'API Run del T6 e precedentemente assente. Ref: 3.5.

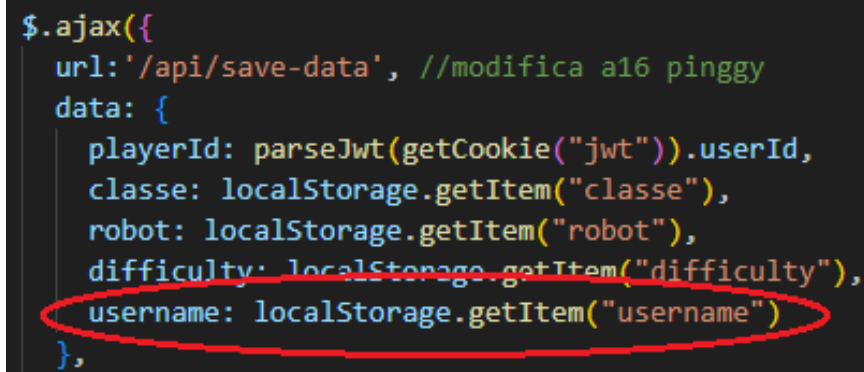
```

var formData = new FormData();
formData.append("testingClassName", "Test" + localStorage.getItem("classe") + ".java");
formData.append("testingClassCode", editor.getValue());
formData.append("underTestClassName", localStorage.getItem("classe") + ".java");
formData.append("underTestClassCode", sidebarEditor.getValue());
formData.append("turnId", localStorage.getItem("turnId"));
formData.append("roundId", localStorage.getItem("roundId"));
formData.append("gameId", localStorage.getItem("gameId"));
formData.append("testClassId", localStorage.getItem("classe"));
formData.append("difficulty", localStorage.getItem("difficulty"));
formData.append("type", localStorage.getItem("robot")); // modificato
formData.append("username", localStorage.getItem("username"));
$.ajax({
    url: "/api/run", // con questa verso il task 6, si salva e conclude la partita e si dec
    type: "POST",
    data: formData,
    processData: false,
    contentType: false,
    dataType: "json",

```

Figura 3.5: estratto api "/run" editor.js

- API SaveData: abbiamo inserito all'interno del JSON anche lo username, richiesto per l'API SaveData del T6 e precedentemente assente. Ref: 3.6.

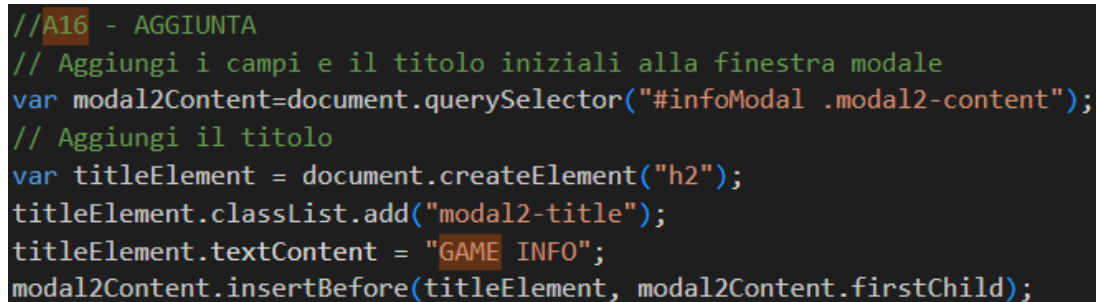


```
$.ajax({
  url: '/api/save-data', //modifica a16 pinggy
  data: {
    playerId: parseJwt(getCookie("jwt")).userId,
    classe: localStorage.getItem("classe"),
    robot: localStorage.getItem("robot"),
    difficulty: localStorage.getItem("difficulty"),
    username: localStorage.getItem("username")
  },

```

Figura 3.6: estratto api `/save-data` editor.js

- GameInfo: il bottone del gruppo A6 duplicava le informazioni ed impiegando il loro codice lo avevamo ereditato. Queste ultime erano presenti sia alla fine del file `editor.js` che all'interno della chiamata all'API `receiveClassUnderTest`, per cui le abbiamo lasciate solo in quest'ultima, come mostrato in figura 3.7.



```
//A16 - AGGIUNTA
// Aggiungi i campi e il titolo iniziali alla finestra modale
var modal2Content=document.querySelector("#infoModal .modal2-content");
// Aggiungi il titolo
var titleElement = document.createElement("h2");
titleElement.classList.add("modal2-title");
titleElement.textContent = "GAME INFO";
modal2Content.insertBefore(titleElement, modal2Content.firstChild);
```

Figura 3.7: Soluzione al bug di GameInfo

3.1.1.4 Creazione e salvataggio del Game

I file **Game.java** e **GameDataWriter.java** sono i file che si prepongono alla creazione e corretta popolazione della classe `"Game"` che viene poi impiegata da **GuiController.java** nella api `/save-data`

per effettuare il salvataggio della sessione di gioco. Abbiamo deciso di utilizzare le relative versioni di A7 integrandovi all'interno le funzionalità necessarie al corretto salvataggio delle informazioni sulla base delle integrazioni precedentemente fatte da A6.

Incominciamo dalla classe java **Game**: gli A6 utilizzano un oggetto lievemente diverso rispetto a quello impiegato dagli A7, in particolare modo aggiungono una variabile **playerName**. Come suggerisce il nome della variabile questa viene utilizzata per salvare proprio il nome del giocatore associato alla specifica partita alla quale si fa riferimento. Nell'integrare abbiamo quindi anche noi aggiunto questa variabile e anche un metodo che ci permetta di reperirlo a partire dalla classe.

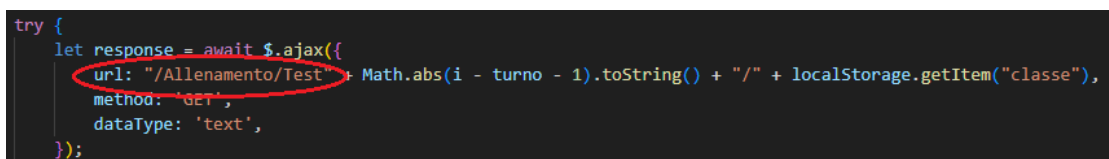
Soffermiamoci ora su **GuiController**, precedentemente avevamo visto come questo ci fosse servito per effettuare i vari mapping delle pagine html di riferimento; adesso ci serve in particolare il PostMapping che viene fatto della Api **/save-data**. In questa Api che viene utilizzata per il salvataggio della partita attuale, viene inizializzato un oggetto Game, in concomitanza con la modifica effettuata precedentemente allora abbiamo modificato anche il costruttore di Game, passandogli il nome "nameAuth" che corrisponde al nome del giocatore che ha correntemente effettuato il log-in.

Infine arriviamo al **GameDataWriter.java**, questo ha il compito di effettuare il salvataggio vero e proprio; GuiController infatti istanzia

un oggetto `GameDataWriter`, il quale costruttore necessita il `Game` precedentemente creato. Andiamo a prelevare all'interno del `GameDataWriter` il nome del giocatore dal `Game` precedentemente creato, questo ci permette quindi di salvare correttamente il nome associato ad un giocatore nel contesto della singola partita.

3.1.1.5 Modifiche ad editorAllenamento

Abbiamo provveduto a modificare anche l'editorAllenamento realizzato dal gruppo A7. Questa modifica è diventata necessaria in quanto siamo andati a modificare il file **prova_esecuzione_parametri4.js**; questo file permetteva agli A7 di definire un'API chiamata dal Volume 8 del task T8. Le modifiche associate a questo file vengono trattate in maniera più approfondita nella sotto-sezione 3.1.4. Soffermiamoci sulle modifiche ad editorAllenamento: abbiamo semplicemente modificato il nome della API stessa; precedentemente l'API era `/tests/` ora è diventata `/Allenamento/`.



```
try {  
  let response = await $.ajax({  
    url: "/Allenamento/Test" + Math.abs(i - turno - 1).toString() + "/" + localStorage.getItem("classe"),  
    method: 'GET',  
    dataType: 'text',  
  });  
}
```

Figura 3.8: estratto file editorAllenamento.js

Il motivo di questa modifica è legato all'introduzione dell'editor di A6: A6 gestisce il salvataggio dei test di una partita allo stesso mo-

do di A7, avevamo quindi una collisione tra la gestione dell'editor e dell'editorAllenamento per mezzo di queste API. Abbiamo quindi dovuto scegliere di modificarne uno o l'altro, abbiamo optato per modificare quello di A7 nell'integrazione.

3.1.2 Modifiche apportate a T6

Le modifiche apportate al task T6 sono sostanziali. Nell'integrare A6 dentro A7 abbiamo notato come l'API `"/run"` del gruppo A7, che viene mappata nel file **MyController.java**, sia molto lunga e "poco pulita"; al contrario l'implementazione che ha fatto il gruppo A6 della stessa API è molto pulita e di facile comprensione. Questo è dovuto al fatto che A6 hanno suddiviso l'API `"/run"` in due mediante l'impiego di una classe creata ad-hoc: **RunnerHelper.java**. Questa infatti permette non solo di discriminare tra le due modalità viste precedentemente in maniera molto elegante e pulita, ma anche di calcolare il punteggio della partite ed effettuare il salvataggio delle informazioni tutto mediante delle funzioni create appositamente. Nell'effettuare l'integrazione del codice abbiamo quindi optato per utilizzare la versione di A6 della gestione della `"/run"` utilizzando anche noi quindi i file sopra citati. L'impiego del **MyController.java** così realizzato ha però comportato la perdita dell'API `"/allenamento"`, abbiamo quindi provveduto ad inserirla nuovamente.

Le modifiche apportate al controller di T6 ne hanno modificato molto

la struttura; possiamo quindi illustrare nella figura 3.9 le modifiche apportate mediante un diagramma delle classi.

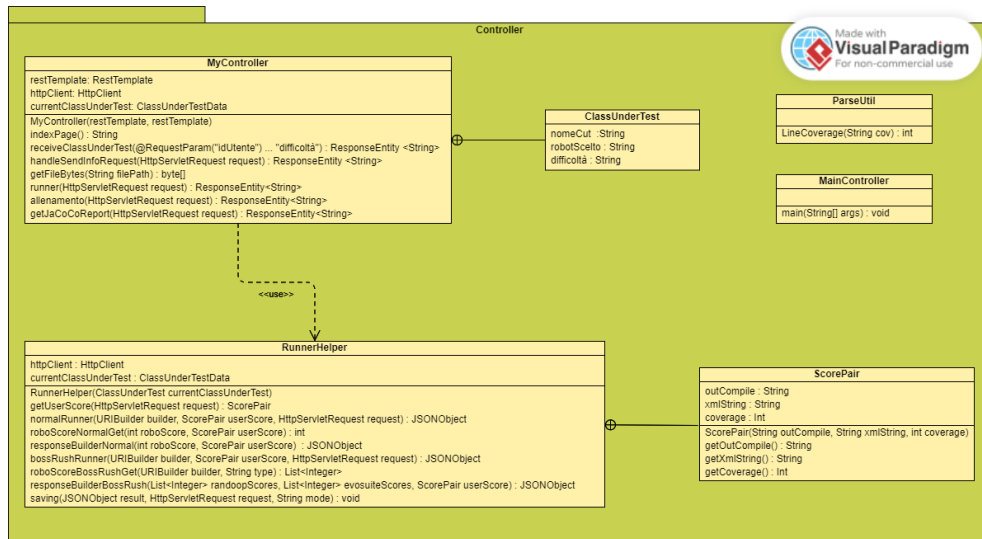


Figura 3.9: Diagramma Classi T6

Possiamo invece mostrare il funzionamento nel dettaglio e ad alto livello del container T6 rispettivamente mediante il sequence diagram (Refs: 3.10 e 3.11) e il diagramma d'attività (Ref: 3.12) nelle seguenti figure.

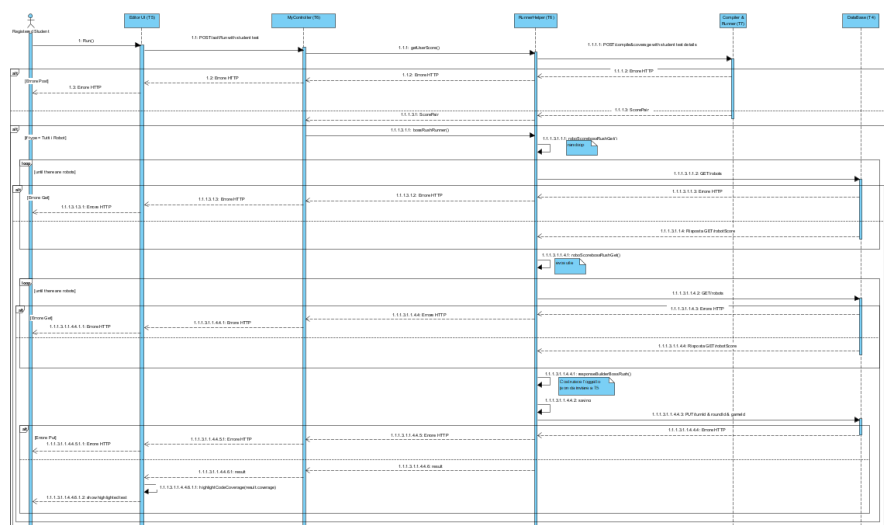


Figura 3.10: Sequence Diagram Pt.1

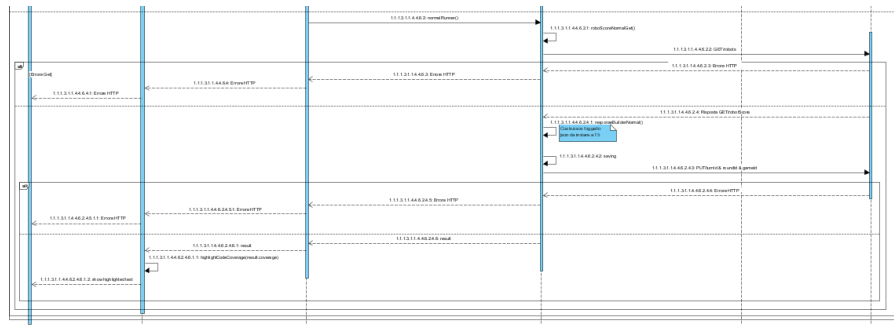


Figura 3.11: Sequence Diagram Pt.2

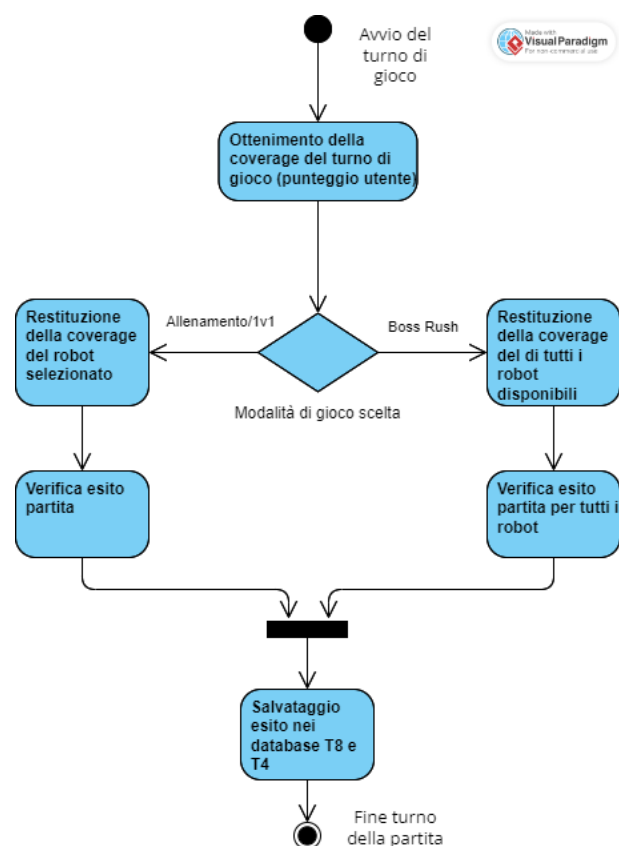


Figura 3.12: Diagramma Attività T6

Le ultime operazioni che ci sono rimaste sono quindi quelle di effettuare bug-fixing:

- HttpPut di rounds: non veniva salvato correttamente il round nella relativa tabella di T4, abbiamo quindi provveduto a pas-

sare correttamente tutti i parametri della partita per un corretto salvataggio. Ref: 3.13.

```
// chiusura round
httpPut = new HttpPut("http://t4-g18-app-1:3000/rounds/" + String.valueOf(request.getParameter("roundId")));

obj = new JSONObject();

obj.put("closedAt", time);
obj.put("testClassId", nomeCUT); //A16 aggiunta A16

jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);
httpPut.setEntity(jsonEntity);
```

Figura 3.13: estratto api "/rounds" RunnerHelper.java

- HttpPut di games: discorso analogo a quanto detto sopra. Non venivano salvate correttamente le informazioni della specifica partita nella relativa tabella di T4, abbiamo quindi provveduto a passare correttamente tutti i parametri della partita per un corretto salvataggio. Abbiamo inoltre aggiunto il salvataggio delle informazioni relative al punteggio Score della tabella Game del task T4; il gruppo A6 non prevede tale parametro nel proprio codice quindi nell'integrare abbiamo dovuto provvedere al corretto salvataggio del punteggio. Questo discorso verrà approfondito meglio nella prossima sotto-sezione. Ref: 3.14.

```
// chiusura gioco
httpPut = new HttpPut("http://t4-g18-app-1:3000/games/" + String.valueOf(request.getParameter("gameId")));

obj = new JSONObject();
obj.put("closedAt", time);
obj.put("username", request.getParameter("username")); //A16 aggiunta A16
obj.put("difficulty", difficoltà); //A16 aggiunta A16
obj.put("isWinner", Boolean.parseBoolean(win)); //A16 aggiunta A16
//A16 qui sotto aggiunta A16
if(mode.equals("bossRush")) {
    int numberOfBeaten = Integer.parseInt(result.getString("numberOfBeaten"));
    int numberOfUnbeaten = Integer.parseInt(result.getString("numberOfUnbeaten"));
    String s = result.getString("score") + "(" + String.valueOf(numberOfBeaten) + "/" + String.valueOf(numberOfBeaten + numberOfUnbeaten) + ")";
    obj.put("score", s);
}
else
    obj.put("score", result.getString("score"));

jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);
httpPut.setEntity(jsonEntity);
```

Figura 3.14: estratto api "/games" RunnerHelper.java

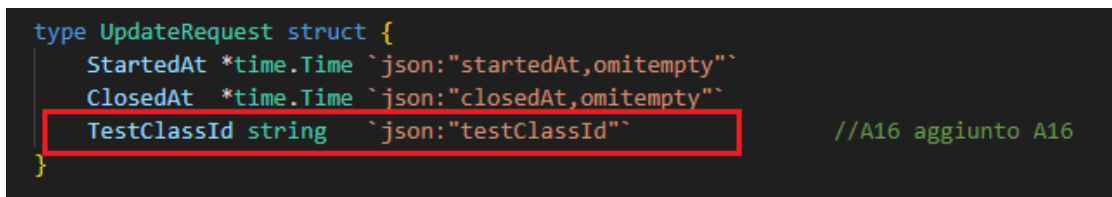
Al termine di queste operazioni tutte le tabelle di T4 vengono salvate correttamente per potenziali utilizzi delle pagine di Storico e Ranking che vedremo dopo.

3.1.3 Modifiche apportate a T4

Le modifiche effettuate a T4 sono numerose e meglio approfondite nel paragrafo 3.2.2, in questo paragrafo ci soffermiamo solo sulle

modifiche effettuate in concomitanza con quelle affrontate nei paragrafi precedenti e riguardano maggiormente il "bug-fixing". Siamo intervenuti su alcune delle tabelle del task T4 in quanto i dati non venivano salvati correttamente, in particolare:

- UpdateRequest di Round: abbiamo aggiunto il parametro TestClassID alla relativa tabella nella funzione di update. Non venivano salvate correttamente nella tabella Round le informazioni sulla classe testata. Ref: 3.15.



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

//A16 aggiunto A16

Figura 3.15: UpdateRequest tabella "Round"

- parametro Score di Game: abbiamo modificato il parametro Score da float64 in string. Questo è stato fatto a seguito delle modifiche apportate nel file RunnerHelper nel salvataggio dello score. Lo score veniva salvato come stringa e quindi siamo andati a modificare il relativo tipo nella tabella Game. Ref: 3.16.

Questo ha comportato delle modifiche al file player.html del task T1, con l'introduzione di un parseFloat sullo score.

```

type Game struct {
    ID          int64      `json:"id"`
    CurrentRound int       `json:"currentRound"`
    Username    string     `json:"username"`
    Description  string     `json:"description"`
    Difficulty   string     `json:"difficulty"`
    Score        string     `json:"score"`           //A16 modificato da float a string
    CreatedAt    time.Time  `json:"createdAt"`
    UpdatedAt    time.Time  `json:"updatedAt"`
    StartedAt    *time.Time `json:"startedAt"`
    ClosedAt     *time.Time `json:"closedAt"`
    IsWinner     bool       `json:"isWinner"`
    Name         string     `json:"name"`
    Players      []Player   `json:"players,omitempty"`
}

type UpdateRequest struct {
    CurrentRound int       `json:"currentRound"`
    Name         string     `json:"name"`
    Username     string     `json:"username"`
    Description   string     `json:"description"`
    Score         string     `json:"score"`           //A16 modificato da float a string
    IsWinner      bool       `json:"isWinner"`
    StartedAt     *time.Time `json:"startedAt,omitempty"`
    ClosedAt      *time.Time `json:"closedAt,omitempty"`
    Difficulty     string     `json:"difficulty"`       //aggiunto da A6 (A16)
}

```

Figura 3.16: Modifiche a tabella "Games" e UpdateRequest di "Games"

3.1.4 Modifiche apportate a T8

Il task T8 corrisponde al principale database dell'applicazione. Qui dentro vengono infatti salvate informazioni sulle classi testate e, importanti nel contesto della nostra integrazione, i Test che vengono effettuati in seguito alle esecuzioni negli editor. Questi sono di particolare interesse ai fini dell'utilizzo del tasto "Storico" all'interno dell'editor stesso. Tanto il gruppo A6 quanto il gruppo A7 facevano utilizzo della API `"/tests/"` nei propri editor: `editor.js` nel caso di A6 e `editor.js/editorAllenamento.js` nel caso di A7. Questa ed altre API sono definite all'interno del file **prova_esecuzione_parametri4.js**. Il problema sorge di fronte all'aver sostituito l'editor di A7 con quello di A6 ereditando l'utilizzo dell'API `"/tests/"` così come implementata dal secondo gruppo. Il gruppo A6 non si ritrovava a dover gestire l'e-

sistenza della modalità di gioco "Allenamento" e quindi non ha mai avuto la necessità di definire un utilizzo duplice per l'API stessa, come invece fatto dal gruppo A7. Questi infatti hanno provveduto a realizzare l'API appena definita per discriminare se la chiamata venisse effettuata nell'editor o nell'editorAllenamento. Nell'andare ad utilizzare l'API del gruppo A6 avremmo potuto effettuare un'integrazione in "stile A7" con una sola API ("/tests/") che gestisse entrambe le chiamate; abbiamo preferito utilizzare un approccio diverso: nel file precedentemente definito abbiamo incluso entrambe le versioni dell'API "/tests/"; quella relativa all'editor lasciata con nome immutato, quella relativa ad editorAllenamento rinominata come "/Allenamento/". Nel fare questo abbiamo dovuto mappare la nuova API nel file **default.conf** dello UiGateway. Abbiamo deciso di adottare questa soluzione in quanto risulta più leggibile e chiaro un riferimento ad una specifica API in base allo specifico editor che viene impiegato; inoltre così facendo abbiamo potuto lasciare invariata la struttura delle API così come definite da ciascun gruppo.

Le richieste finali che quindi verranno effettuate da una e l'altra API prendono il seguente formato:

`http://ip:porta/Allenamento/TestX/Classe`
`http://ip:porta/tests/NumeroDelTest/Test+"Classe"`

Un rimando al codice associato al file `prova_esecuzione_parametri4.js` lo si può trovare in figura 3.17.

```

//18 modifica per editorAllenamento
else if (req.url.startsWith('/Allenamento/')) { //Al6 al posto di Allenamento c'era "tests"
  res.setHeader('Access-Control-Allow-Origin', '*');

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

  let testPath;
  console.log(path);

  if (path.length <= 2) {
    const test = path[0];
    const classe = path[1];
    testPath = '/Volume18/FolderTreeEvo/Allenamento/' + test + '/TestReport/Test' + classe + '.java';
  } else {
    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}');

    testPath = '/Volume18/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);
    }
  });
}

//Al6 aggiunta per editor lvi e lvali
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 numero = path[0];
    const nome = path[1];
    console.log('Valore di "numero": ${numero}, Valore di "nome": ${nome}');

    const testPath = '/Volume18/FolderTreeEvo/Tests/' + numero + '/' + nome + '.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 3.17: estratto file prova_esecuzione_parametri4.js

3.2 Storico

La pagina *storico.html* mostra lo storico delle partite giocate dall'utente che ha effettuato il log-in. E' uno dei requisiti introdotti dal gruppo A6 ed assente nel progetto del gruppo A7 ed è stato quindi di particolare interesse nel contesto della nostra integrazione.

Per implementare la funzionalità dello storico abbiamo dovuto apportare modifiche a tre task: T4, T5 e T6. Le modifiche ai primi due task citati vengono approfondite in questo capitolo, le modifiche al task T6 invece vengono approfondite nel paragrafo 3.1.2.

3.2.1 Modifiche apportate a T5

Ci apprestiamo quindi ad elencare brevemente le modifiche apportate al task T5 e la modalità di integrazione impiegata.

3.2.1.1 main

Abbiamo dovuto iniziare dal modificare la pagina iniziale. Il gruppo A7 successivamente al log-in portava il giocatore a scegliere direttamente la modalità di gioco, questo poichè non prevedeva alcuna funzionalità extra per il giocatore al di fuori della possibilità di giocare al gioco stesso; il gruppo A6, d'altro canto, per permettere al giocatore di accedere ad uno storico o al ranking ha dovuto realizzare una pagina d'apertura che permettesse al player di selezionare a quale delle modalità precedentemente descritte volesse accedere. Abbiamo quindi modificato la pagina **main.html** per permettere questi cambiamenti; in particolare abbiamo adottato la formattazione introdotta dal gruppo A6 nel fare questo. Ovviamente i cambiamenti al template offerto dalla pagina **main.html** non sono sufficienti da soli, abbiamo dovuto modificare anche il file **main.js** per permettere di effettuare un redirect verso la pagina html stessa.

Una vista di come si presenta l'interfaccia dell'applicativo per gli utenti, una volta apportate le modifiche, la si può trovare nella figura 3.18.

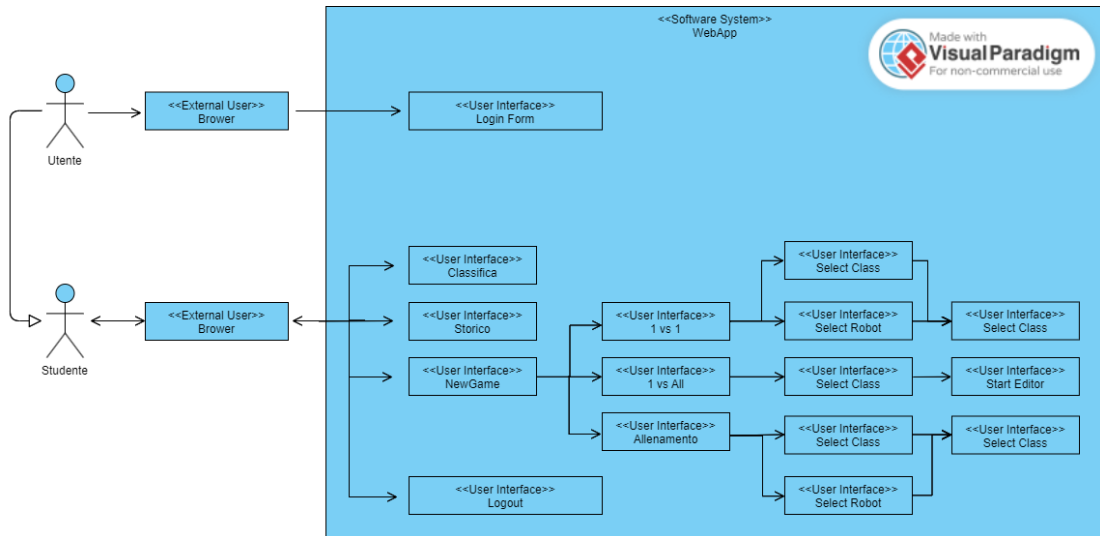


Figura 3.18: Diagramma Contesto

3.2.1.2 storico.html

Soffermiamoci ora sulla pagina **storico.html**, la pagina di storico vera e propria. Dal momento che questa è una funzionalità introdotta da A6 e precedentemente non presente in A7 abbiamo provveduto semplicemente ad inserirla nel contesto del task T5. L'introduzione della pagina **storico.html** era caratterizzata però da delle incongruenze, affrontiamo il discorso in maniera più specifica. Come visibile in figura 3.19, le informazioni alle quali abbiamo accesso sono: "*ID Partita*", Orario di Inizio e Fine Partita, "*Classe Testata*", "*Robot Sfidato*", "*Risultato Partita*", "*Difficoltà*" e "*Scores*".

Il parametro "*Difficoltà*" è di particolare interesse, nel caso di una partita nella modalità BossRush questo parametro non viene mai prelevato dalla classe selezionata e quindi mai salvato correttamente.



ID Partita	Inizio Partita	Fine Partita	Classe Testata	Robot Sfidato	Risultato Partita	Difficoltà	Scores
4	2024-05-06T08:40:05.374477Z	2024-05-06T08:40:56.344012Z	Calcolatrice	evosuite	Vittoria	1	100
5	2024-05-06T08:41:41.2063Z	2024-05-06T08:42:04.433894Z	VCardBean	Tutti I Robot	Sconfitta	MAX	9(0/6)
7	2024-05-06T08:50:47.630949Z	2024-05-06T08:51:04.311614Z	Calcolatrice	Tutti I Robot	Sconfitta	MAX	75(0/2)
8	2024-05-06T08:51:08.704654Z	2024-05-06T08:51:51.196597Z	Calcolatrice	Tutti I Robot	Vittoria	MAX	100(2/2)

Figura 3.19: storico.html

te. Questo è dovuto ad un errore di sviluppo relativo al gruppo A6; abbiamo quindi dovuto decidere autonomamente come salvare la difficoltà associata ad una partita in questa modalità, optando per impostare la difficoltà come "MAX". Il discorso viene approfondito nel contesto del paragrafo 3.1.1.2.

Le modifiche adoperate per ottenere il parametro "*Difficoltà*" non sono però le uniche apportate al progetto per permettere il funzionamento dello storico e del ranking. Il gruppo A6 ha infatti apportato importanti modifiche al task T4 per permettere il salvataggio di alcuni dei parametri che vediamo sopra. Queste modifiche vengono approfondite nel paragrafo 3.2.2.

3.2.1.3 GuiController.java

Per permettere che il tutto funzioni correttamente abbiamo dovuto "mappare" la pagina di storico nel GuiController come si può

vedere in figura 3.20, nel fare questo abbiamo integrato direttamente da A6.

```
@GetMapping("/storico")
public String storicoPage(Model model, @CookieValue(required = false) String jwt){
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
    formData.add("jwt", jwt);

    Boolean isAuthenticated = restTemplate.postForObject("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class);

    if(isAuthenticated == null || !isAuthenticated) return "redirect:/login";

    Integer IdTemp = restTemplate.postForObject("http://t23-g1-app-1:8080/IdToken", formData, Integer.class);
    if(IdTemp != null){
        IdAuth = IdTemp.toString();
        String url = "http://t4-g18-app-1:3000/turns/account/" + IdAuth;
        List<Map<String, Object>> storico = restTemplate.getForObject(url, List.class);
        if(storico!=null){
            List<Map<String, Object>> filteredStorico = storico.stream()
                .filter(turn -> turn.get("closedAt") != null) //filtra per escludere "closedAt" == null
                .sorted(Comparator.comparingInt((Map<String, Object> turn) -> (Integer) turn.get("id"))) //ordina per "id"
                .collect(Collectors.toList());
            model.addAttribute("storico", filteredStorico);
        } else{
            System.out.println("Nessun storico ricevuto o errore nella richiesta");
        }
        // storico.sort(Comparator.comparingInt((Map<String, Object> turn) -> (Integer) turn.get("id")));
        //System.out.println("ID utente: " + IdAuth);
    }else{
        System.out.println("ID utente non ricevuto o errore nella richiesta");
    }

    model.addAttribute("IdAuth", IdAuth);

    return "storico";
}
```

Figura 3.20: Mapping di storico nel GuiController

3.2.2 Modifiche apportate a T4

Le modifiche apportate al task T4 sono tanto funzionali quanto strutturali, esse sono fondamentali in quanto ci permettono di memorizzare correttamente l'esito delle partite e le associazioni partite-giocate/giocatori.

3.2.2.1 Tabella Player

Nell'andare ad integrare la modalità BossRush, precedentemente, avevamo già visto come l'API `"/save-data"` faccia riferimento

ad un `GameDataWriter` ed in particolare alla funzione `"saveGame"` (3.1.1.4). Qui dentro si poteva infatti notare un primo riferimento alla tabella `"players"`; il gruppo A6 ha infatti creato una nuova tabella nota come **Player** alla quale ci si riferenzia come `"players"`.

Questa tabella ha lo scopo di memorizzare informazioni in modo analogo al database MySQL `"STUDENTSREPO"` che viene impiegato nel task T23; la differenza è che questo database e in particolar modo questa tabella tiene traccia delle vittorie dei player e dei turni giocati.

```
type Player struct {  
    ID          int64      `json:"id"`  
    AccountID   string     `json:"accountId"`  
    Name        string     `json:"name"`  
    CreatedAt   time.Time   `json:"createdAt"`  
    UpdatedAt   time.Time   `json:"updatedAt"`  
    Wins        int64      `json:"wins"`  
    TurnsPlayed int64      `json:"turnsPlayed"` //aggiunto  
}
```

Figura 3.21: estratto tabella Player

Questa tabella risulta quindi più utile, rispetto al precedentemente citato `STUDENTSREPO`, ai fini della realizzazione dello storico e del ranking. Nell'affrontare la fase di integrazione abbiamo quindi deciso di integrare anche questa nuova tabella per realizzare un lavoro più coerente con le fasi successive.

3.2.2.2 Tabella Turn

La pagina `storico.html` tenta di accedere, mediante il mapping effettuato dal `GuiController`, alle righe della tabella **Turn** che viene definita nel task T4. In particolare lo storico cerca di accedere a tre risorse precedentemente non definite da A7:

- Difficulty
- Robot
- TestClass

Abbiamo quindi provveduto a modificare la tabella **Turn** aggiungendo queste nuove entry, integrando ciò che era stato fatto precedentemente da A6, modificando anche la funzione di Update che viene effettuata sulla tabella stessa.

```
type Turn struct {
    ID          int64      `json:"id"`
    IsWinner    bool       `json:"isWinner"`
    Order       int        `json:"order"`
    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"`
    TestClass   string     `json:"testClass"`
    Robot       string     `json:"robot"`
    Difficulty   string     `json:"difficulty"`
}
```

Figura 3.22: estratto tabella Turn

Sempre associata alla tabella Turn, abbiamo aggiunto una Update sul parametro IsWinner, responsabile del conservare informazioni sull'esito del turno di gioco. Precedentemente gli A6 non salvavano correttamente questa informazione, abbiamo provveduto quindi a risolvere questo problema.

Il gruppo A6 ha apportato poi altre modifiche funzionali alla tabella Turn, implementando quattro funzioni:

- ListByAccount: definita nel **controller.go** della tabella Turn, serve allo storico per poter accedere allo specifico AccountID. Viene infatti utilizzata da storico.html e GuiController.java (sempre nel contesto dello storico). Un riferimento alla funzione lo si trova in figura 3.23.

```
func (tc *Controller) ListByAccount(w http.ResponseWriter, r *http.Request) error{
    var accountID KeyType
    accountID, err := api.FromUrlParams[KeyType](r, "accountID")
    if err != nil{
        return err
    }
    accountIDString := strconv.FormatInt(int64(accountID),10)
    turns, err := tc.service.GetTurnsByAccountID(accountIDString)
    if err != nil {
        return api.MakeHttpError(err)
    }

    return api.WriteJson(w, http.StatusOK, turns)
}
```

Figura 3.23: funzione ListByAccount

- `GetTurnsByAccountID`: definita nel **service.go** della tabella `Turn`. Questa funzione, che viene chiamata dalla precedente, recupera tutti i turni associati a un determinato account dal database e li restituisce come slice di strutture `Turn`. Un riferimento alla funzione lo si trova in figura 3.24.

```
func (tr *Repository) GetTurnsByAccountID(accountID string) ([]Turn, error) {
    var turns []model.Turn

    err := tr.db.
        Model(&model.Turn{}).
        Joins("Join players ON turns.player_id = players.id").
        Where("players.account_id = ?", accountID).
        Find(&turns).
        Error

    if err != nil {
        return nil, err
    }

    resp := make([]Turn, len(turns))
    for i, turn := range turns {
        resp[i] = fromModel(&turn)
    }
    return resp, nil
}
```

Figura 3.24: funzione `GetTurnsByAccountID`

- `AfterSave`: definita nel file **model.go** esterno alla cartella `Turn`; questa funzione viene richiamata ogni volta che viene salvato un record nella tabella `Turn`. Il suo scopo è quello di associare il turno di gioco ad uno specifico player e provvedere ad aggiornare non solo il campo `TurnsPlayed` dal giocatore ma anche il numero di vittorie sulla base del parametro `IsWinner`; tutte queste informazioni sono utili soprattutto al ranking che vedremo dopo. Un riferimento alla funzione lo si trova in figura 3.25.

```

//Hook che si attiva ogni volta che si salva un record nella tabella Turns
func(pg *Turn) AfterSave(tx *gorm.DB) (err error){
    playerID := pg.PlayerID
    var turnsPlayed int64
    result := tx.Model(&Turn{}).Where("player_id = ? AND closed_at IS NOT NULL", playerID).Count(&turnsPlayed)
    if result.Error != nil {
        return result.Error
    }

    //Aggiorna il campo TurnsPlayed nella tabella player
    result = tx.Model(&Player{}).Where("id = ?", playerID).Update("turns_played", turnsPlayed)
    if result.Error != nil {
        return result.Error
    }

    // Ottieni il valore IsWinner dalla tabella Turn per il giocatore specifico
    result = tx.Model(&Turn{}).Where("player_id = ? AND is_winner = ?", playerID, true).Select("is_winner").Scan(&isWinner)
    if result.Error != nil {
        return result.Error
    }

    // Se isWinner è true, aggiorna il campo Wins nella tabella Player
    if isWinner {
        var winsCount int64

        // Calcola il numero di vittorie per il giocatore specifico dalla tabella Turn
        result = tx.Model(&Turn{}).Where("player_id = ? AND is_winner = ?", playerID, true).Count(&winsCount)
        if result.Error != nil {
            return result.Error
        }

        // Aggiorna il campo Wins nella tabella Player con il numero di vittorie ottenuto
        result = tx.Model(&Player{}).Where("id = ?", playerID).Update("wins", winsCount)
        if result.Error != nil {
            return result.Error
        }
    }

    return nil
}

```

Figura 3.25: funzione AfterSave

- UpdatePlayerWins: sempre definita nel file **model.go**, questa funzione viene chiamata dalla funzione di Update della tabella Turn, e quindi definita nel file **service.go**. La Update di Turn è stata infatti modificata come mostrato in figura 3.26.

```

func (tr *Repository) Update(id int64, r *UpdateRequest) (Turn, error) {
    //A16: la versione commentata è quella originale di A7, modificata poi da A9 con la versione non commentata scritta sotto
    /*
    var {
        turn model.Turn = model.Turn{ID: id}
        err error
    }

    err = tr.db.Model(&turn).Updates(r).Error
    return fromModel(&turn), api.MakeServiceError(err)
    */

    //A16: questa è la versione usata da A9
    var turn model.Turn
    err := tr.db.First(&turn, id).Error
    if err != nil {
        return Turn{}, api.MakeServiceError(err)
    }

    playerID := turn.PlayerID
    log.Printf("Updating wins for playerID: %d", playerID)

    err = tr.db.Model(&turn).Updates(r).Error
    if err != nil {
        return Turn{}, api.MakeServiceError(err)
    }

    if err := model.UpdatePlayerWins(tr.db, playerID); err != nil {
        log.Printf("Error updating player wins: %v", err)
        return Turn{}, api.MakeServiceError(err)
    }

    return fromModel(&turn), nil
}

```

Figura 3.26: modifiche ad Update di turn

Le modifiche effettuate alla Update si concretizzano quindi con la chiamata della funzione UpdatePlayerWins, che ha il solo scopo di calcolare ed aggiornare il numero di vittorie della tabella Player. Un estratto della funzione lo si trova in figura 3.27.

```
// Funzione per aggiornare il campo Wins per un giocatore specifico
func UpdatePlayerWins(db *gorm.DB, PlayerID int64) error {
    var winsCount, turnsPlayed int64

    // Calcola il numero di vittorie per il giocatore specifico
    result := db.Model(&Turn{}).Where("player_id = ? AND is_winner = ?", PlayerID, true).Count(&winsCount)
    if result.Error != nil {
        log.Printf("Error counting wins for player %d: %v", PlayerID, result.Error)
        return result.Error
    }
    log.Printf("Player %d has %d wins", PlayerID, winsCount)

    // Calcola il numero totale di partite giocate e terminate correttamente per il giocatore specificato
    result = db.Model(&Turn{}).Where("player_id = ? AND closed_at IS NOT NULL", PlayerID).Count(&turnsPlayed)
    if result.Error != nil {
        log.Printf("Error counting turns played for player %d: %v", PlayerID, result.Error)
        return result.Error
    }
    log.Printf("Player %d has played %d turns", PlayerID, turnsPlayed)

    // Aggiorna il campo Wins e TurnsPlayed nella tabella Player
    result = db.Model(&Player{}).Where("id = ?", PlayerID).Updates(map[string]interface{}{
        "wins":      winsCount,
        "turns_played": turnsPlayed,
    })
    if result.Error != nil {
        log.Printf("Error updating player %d: %v", PlayerID, result.Error)
        return result.Error
    }
    log.Printf("Updated player %d: %d wins, %d turns played", PlayerID, winsCount, turnsPlayed)

    /*
    // Aggiorna il campo Wins nella tabella Player con il numero di vittorie ottenuto
    result = db.Model(&Player{}).Where("id = ?", PlayerID).Update("wins", winsCount)
    if result.Error != nil {
        return result.Error
    }
    */

    return nil
}
```

Figura 3.27: funzione UpdatePlayerWins

Queste funzioni sono fondamentali per effettuare l'associazione turni giocati/giocatori e quindi fare riferimento ai campi della tabella Player definita al paragrafo precedente.

La struttura del database del task T4 in seguito alle modifiche apportate è rappresentata in figura 3.28 mediante diagramma ER.

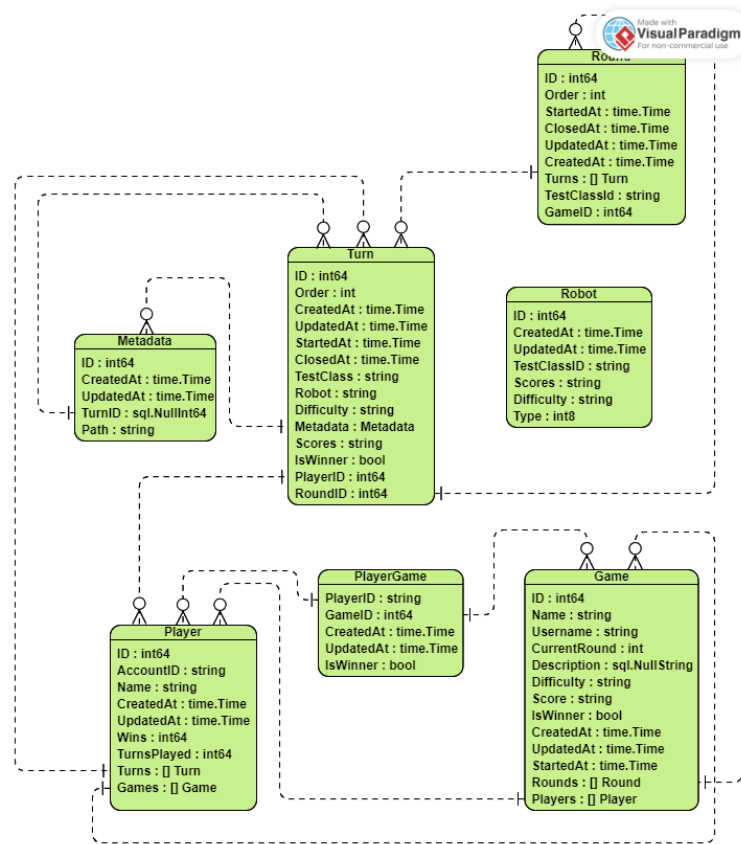


Figura 3.28: Diagramma ER T4

3.3 Ranking

Il ranking è l'ultimo requisito aggiunto da A6 e da noi integrato nel contesto dell'applicazione. La funzionalità esposta dal ranking è quella di elencare tutti i giocatori presenti nel database dell'applicazione in ordine decrescente di vittorie. Ref: 3.29 .



ID Giocatore	Nome Giocatore	Partite Vinte	Partite Effettuate
1	elio	9	9
5	prova	8	8
4	ANTONIO	5	7
2	test	1	1
3	testblue	1	1

Figura 3.29: Pagina Ranking

Le modifiche che siamo andati ad effettuare sono condensate nelle considerazioni effettuate nei capitoli precedenti; l'integrazione di queste funzionalità ha infatti richiesto modifiche ai task: T4, T5 e T6. Affronteremo quindi molto brevemente le modifiche effettuate per non essere ridondanti con i paragrafi precedenti, effettuando un rimando a quelli associati alle modifiche necessarie per il funzionamento della funzionalità "Ranking".

3.3.1 Modifiche a T5

Le modifiche effettuate a T5 sono relative all'integrazione della pagina **ranking.html**. Questa viene totalmente integrata da A6,

così come il suo mapping che viene effettuato nel file **GuiController.java**. Il funzionamento del ranking è definito interamente nel **GuiController**, viene infatti effettuato un ordinamento inverso basato sul parametro "*wins*" dei player contenuti nella tabella "*players*" come vediamo nella figura 3.30. .

```
//A10 - Integrazione di Mapping per la pagina Classifica
@GetMapping("/ranking")
public String rankingPage(Model model, @CookieValue(name = "jwt", required = false) String jwt){
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
    formData.add("jwt", jwt);

    Boolean isAuthenticated = restTemplate.postForObject("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class);
    if(!isAuthenticated == null || !isAuthenticated) return "redirect:/login";

    List<Map<String, Object>> classifica = restTemplate.getForObject("http://t4-g18-app-1:3000/players", List.class);
    classifica.sort(Comparator.comparingInt((Map<String, Object> giocatore) -> (Integer) giocatore.get("wins")).reversed());
    model.addAttribute("classifica", classifica);

    return "ranking";
}
```

Figura 3.30: API GetMapping ranking

3.3.2 Modifiche a T4

Così come il paragrafo precedente, anche questo sarà molto riassuntivo. Risulta evidente infatti dalle considerazioni effettuate precedentemente come si dipenda in maniera diretta dalla tabella Player, la quale struttura è stata definita precedentemente nel paragrafo 3.2.2.1.

3.4 Requisito aggiuntivo: Redirect dopo registrazione

In riferimento alla sezione "Note e Sviluppi Futuri" di A10, in cui consigliano di ritornare alla pagina di "login" dopo l'avvenuta

registrazione di un nuovo utente, abbiamo deciso di apportare questa modifica per migliorare l'esperienza utente.

3.4.1 Modifiche a T23

Abbiamo modificato il file "register.html" nel Task T23 al fine di ritornare automaticamente alla pagina di login a seguito di un messaggio di successo di avvenuta registrazione. In particolare, se la POST va a buon fine, verrà mostrato un alert con il messaggio "Registration completed successfully!" e dopo aver cliccato su "OK", passato un secondo, verremo reindirizzati alla pagina di "login". Se, al contrario, abbiamo errori sul controllo password o sull'email, avremo l'alert corrispondente e resteremo sulla pagina "register" per modificare i parametri sbagliati. Di seguito, il codice aggiunto per abilitare questa funzionalità.

```
<div class="container">
  <!-- A16 Aggiunta function per effettuare redirect a pagina di login dopo registrazione, se avvenuta con successo-->
  <script type="text/javascript">
    function register() {
      const form = document.getElementById('registerForm');
      const formData = new FormData(form);
      const searchParams = new URLSearchParams();
      for (const pair of formData) {
        searchParams.append(pair[0], pair[1]);
      }

      fetch(form.action, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        },
        body: searchParams.toString()
      }).then(response => {
        if (response.ok) {
          return response.text().then(data => {
            alert(data);
            setTimeout(() => {
              window.location.href = '/login';
            }, 1000); // Redirect dopo 1 secondo
          });
        } else {
          return response.text().then(data => {
            alert("Errore: " + data);
          });
        }
      })
      .catch(error => console.error('Error:', error));
    }
  </script>

```

Figura 3.31: Funzione register

Capitolo 4

Deployment

Segue, in figura 4.1, il diagramma di deployment che è stato consultato durante la fase di progettazione. I container invariati sono evidenziati in verde, mentre quelli soggetti a modifiche sono segnalati in giallo. L'applicativo è ospitato su un server che utilizza il sistema operativo Ubuntu 20.04. 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 di A6 precedente rispetto alla situazione attuale in cui, in A7, c'è l'esposizione su rete pubblica (presa dal gruppo A10). Tuttavia, rappresenta accuratamente le modifiche apportate per l'implementazione dell'integrazione. Per la soluzione più recente, si rimanda alla documentazione aggiornata di A7 che include l'esposizione su rete pubblica introdotta da A10. Dall'analisi del diagramma, emerge che il volume T8

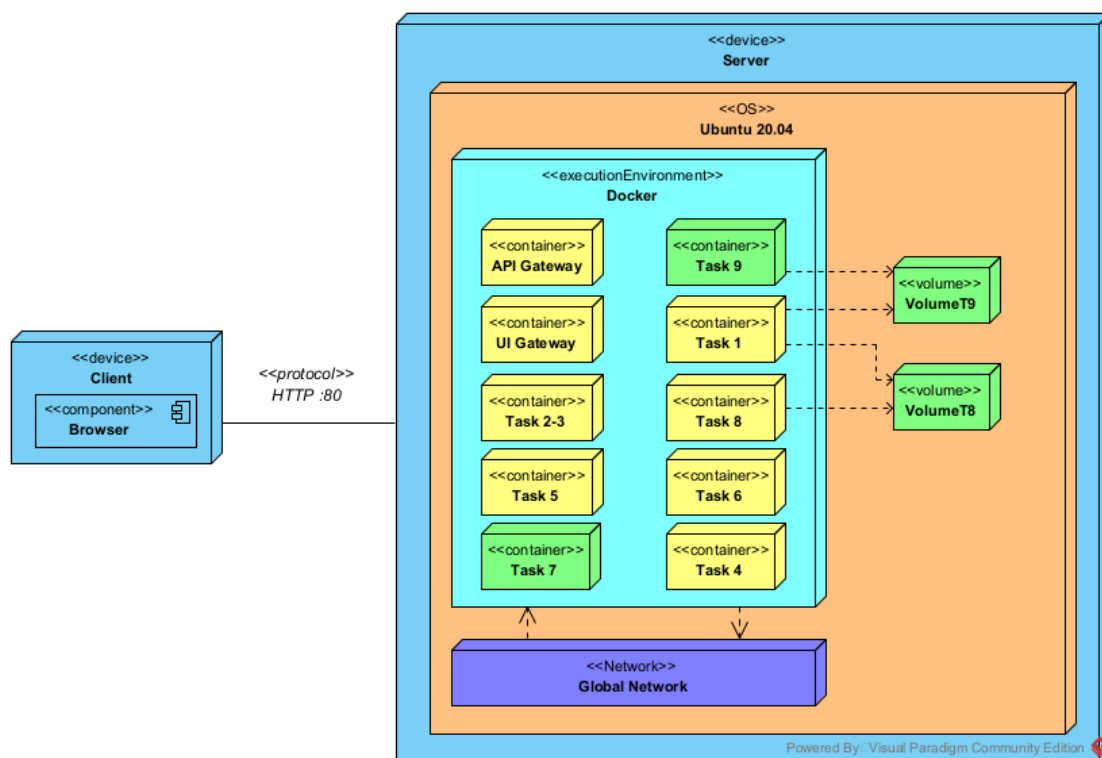


Figura 4.1: Diagramma di Deployment.

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.

4.1 Integrazione con la nuova versione

Tutte le modifiche menzionate nei capitoli precedenti sono state effettuate sulla versione dell'applicazione di A7 la cui configurazione era settata in modo da garantire anche l'esposizione su un dominio

pubblico, mentre A6 non lo prevedeva. Per cui, di seguito, mostriamo le modifiche effettuate affinché le funzionalità di A6 fossero raggiungibili da un indirizzo pubblico.

4.1.1 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. Come descritto nel capitolo precedente, un esempio è il file `editor.html`, che inizialmente conteneva sia il codice *HTML* che il *CSS* ed 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.

4.1.2 Corretta gestione degli URL

Una delle problematiche è stata la modifica degli URL delle richieste che erano state aggiunte per la corretta gestione dei salvataggi nel database del T4. Inoltre, l'applicazione consente l'esposizione su rete pubblica, per questo motivo è stato necessario rimuovere tutti i riferimenti a *localhost* ma, per un maggiore approfondimento, rimandiamo alla documentazione del team A10.

Capitolo 5

Testing

Nell'industria del software, la pratica del testing assume un ruolo cruciale per garantire la qualità e l'affidabilità dei prodotti sviluppati. Il testing, come processo di valutazione e verifica del software, riveste fondamentale importanza nell'intero ciclo di vita di un'applicazione. Questo capitolo esplorerà l'importanza del testing, evidenziando i suoi benefici e il suo impatto sulla corretta riuscita del prodotto richiesto, sia da un punto di vista funzionale sia realizzativo.

Il testing rappresenta un pilastro fondamentale per garantire che un'applicazione soddisfi gli standard di qualità prefissati e richiesti. Attraverso il processo di individuazione e correzione di difetti, errori o comportamenti indesiderati, il testing assicura che il software sia pronto per essere rilasciato al pubblico, o come nel nostro caso, ci permette di presentare un progetto stabile.

Nel contesto delle metodologie di sviluppo AGILE, il testing assume un ruolo ancora più cruciale. Nelle pratiche AGILE, il testing viene eseguito costantemente durante ogni iterazione di sviluppo, consentendo un rapido feedback e una rapida risoluzione dei problemi. Nel nostro specifico caso, l'attività richiesta esprimeva la necessità di integrazione di funzionalità provenienti da due progetti diversi. Pertanto, il testing ha giocato un ruolo principale nella nascita e nella mitigazione di ulteriori bug date le fondamenta architetturali completamente diverse. Il testing quindi, non solo contribuisce a garantire che l'applicazione funzioni come previsto, ma migliora anche la stabilità, la sicurezza e le prestazioni complessive del prodotto.

5.1 Testing con Postman

Per garantire la conformità lato database PostgreSQL e alle sue tabelle fondamentali abbiamo adottato un approccio di testing ispirato al metodo utilizzato dal gruppo A7 nel loro progetto. Abbiamo iniziato con un'attenta analisi del funzionamento delle API coinvolte, utilizzando lo strumento Postman (Sez. 2.2.7), come descritto nella loro metodologia. Attraverso richieste GET mirate agli indirizzi appropriati, abbiamo verificato con precisione l'inserimento corretto dei parametri chiave nelle tabelle di Games (Fig.5.1, Fig.5.2) e di Turns (Fig.5.3, Fig.5.4). Ricordando che il gruppo A7 prevede la sola sfida contro il singolo robot, il nostro scopo è riuscito a fornir-

re le stesse informazioni inserendo anche la modalità BossRush del gruppo A6.

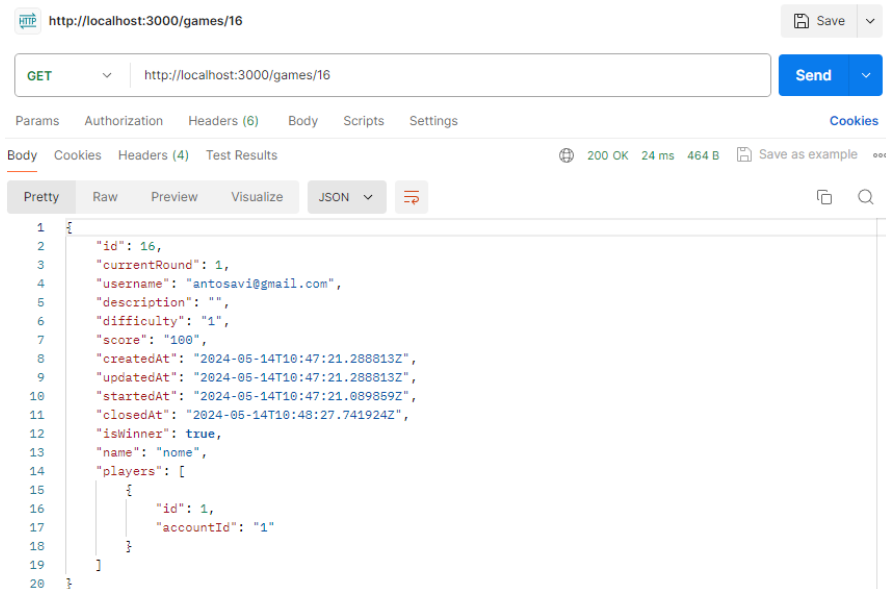


Figura 5.1: Risultato test */games* - 1VS1

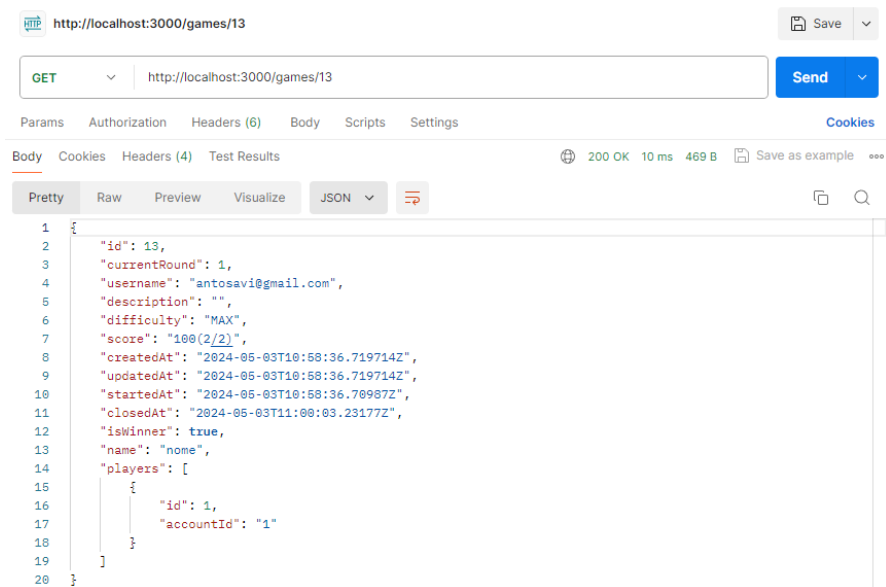


Figura 5.2: Risultato test */games* - BossRush

Grazie a questa analisi, siamo stati in grado di confermare che i dati necessari erano stati inviati correttamente nelle rispettive ta-

belle del database. Questo approccio ci ha consentito di verificare in modo esaustivo la conformità delle nostre funzionalità integrate, assicurandoci che il nostro progetto mantenesse la compatibilità con il database PostgreSQL e le sue tabelle cruciali per il funzionamento delle API nell'applicazione.

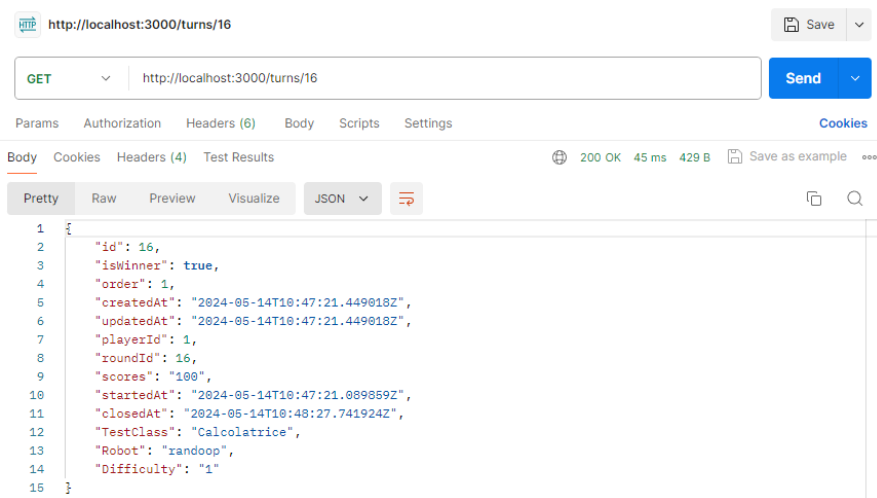


Figura 5.3: Risultato test */turns* - 1VS1

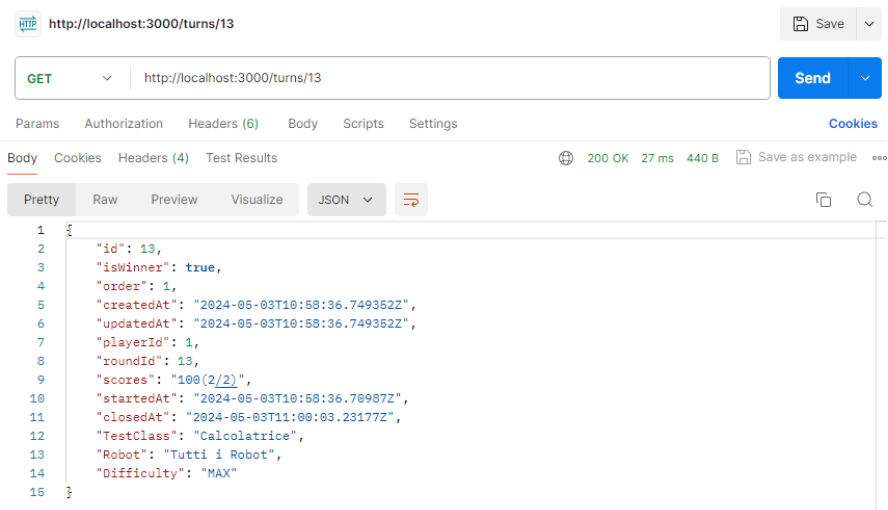


Figura 5.4: Risultato test */turns* - BossRush

5.2 Verifica della coerenza dei dati tramite DBeaver

Successivamente, abbiamo proseguito con l'utilizzo del programma DBeaver (Sez. 2.2.9) 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.

id	order	created_at	updated_at	started_at	closed_at	test_class	robot	difficulty	score	is_winner	player_id	round_id
2	1	02 12:25:37.11	02 12:25:37.1	02 12:25:37.002	2 12:26:20.284	Calcolatrice	randoop	1	100	[v]	1	2
10	1	03 12:41:52.05	03 12:41:52.0	03 12:41:51.571	3 12:43:15.806	Calcolatrice	randoop	1	100	[v]	1	10
11	1	03 12:48:30.63	03 12:48:30.6	03 12:48:30.454	3 12:49:32.384	Calcolatrice	randoop	1	100	[v]	1	11
12	1	03 12:53:32.57	03 12:53:32.5	03 12:53:32.104	3 12:54:35.414	Calcolatrice	randoop	1	100	[v]	1	12
13	1	03 12:58:36.74	03 12:58:36.7	03 12:58:36.709	3 13:00:03.231	Calcolatrice	Tutti i Robot	MAX	100(2/2)	[v]	1	13
14	1	08 15:11:36.75	08 15:11:36.7	08 15:11:36.613	3 15:12:22.910	Calcolatrice	randoop	1	100	[v]	1	14
15	1	08 15:13:58.72	08 15:13:58.7	08 15:13:58.706	3 15:14:31.726	Calcolatrice	Tutti i Robot	MAX	100(2/2)	[v]	1	15
16	1	14 12:47:21.44	14 12:47:21.4	4 12:47:21.089	4 12:48:27.741	Calcolatrice	randoop	1	100	[v]	1	16
17	1	14 18:11:53.71	14 18:11:53.7	4 18:11:53.538	4 18:12:50.141	Calcolatrice	randoop	1	100	[v]	17	17
18	1	14 18:14:28.86	14 18:14:28.8	4 18:14:28.845	4 18:14:53.626	Calcolatrice	Tutti i Robot	MAX	100(2/2)	[v]	17	18
19	1	14 18:18:50.44	14 18:18:50.4	4 18:18:50.416	4 18:19:21.899	Calcolatrice	Tutti i Robot	MAX	75(0/2)	[]	17	19
21	1	14 18:20:05.78	14 18:20:05.7	4 18:20:05.768	4 18:20:33.531	Calcolatrice	Tutti i Robot	MAX	75(0/2)	[]	17	21
23	1	14 18:49:40.65	14 18:49:40.6	4 18:49:40.566	4 18:50:59.868	Calcolatrice	randoop	1	100	[v]	17	23
24	1	14 19:01:43.16	14 19:01:43.1	4 19:01:43.136	4 19:02:31.055	Calcolatrice	Tutti i Robot	MAX	100(2/2)	[v]	17	24
28	1	15 12:13:32.16	15 12:13:32.1	5 12:13:32.130	5 12:14:23.276	Calcolatrice	randoop	1	100	[v]	17	28

Figura 5.5: Risultato tabella */turns* Postgres

Durante la fase di esame del codice iniziale e di indagine delle prime settimane, abbiamo adottato una strategia mirata a comprendere il funzionamento del database gestito dietro al container Docker di Postgres. Abbiamo utilizzato il comando *docker exec -it* per accedere al container e abbiamo interagito direttamente con il database tramite il terminale. Questa fase è stata cruciale per comprendere

id	name	username	current_round	difficulty	score	is_winner	created_at	updated_at	started_at	closed_at
2	anto	antosavi@gmail.com	1	[N] 1	100	[v]	12:25:37.041 +020	12:25:37.041 +020	12:25:37.002 +020	12:26:20.284 +020
10	anto	antosavi@gmail.com	1	[N] 1	100	[v]	12:41:51.906 +020	12:41:51.906 +020	12:41:51.571 +020	12:43:15.806 +020
11	anto	antosavi@gmail.com	1	[N] 1	100	[v]	12:48:30.566 +020	12:48:30.566 +020	12:48:30.454 +020	12:49:32.384 +020
12	anto	antosavi@gmail.com	1	[N] 1	100	[v]	12:53:32.436 +020	12:53:32.436 +020	12:53:32.104 +020	12:54:35.414 +020
13	anto	antosavi@gmail.com	1	[N] MAX	100(2/2)	[v]	12:58:36.719 +020	12:58:36.719 +020	12:58:36.709 +020	13:00:03.231 +020
14	anto	antosavi@gmail.com	1	[N] 1	100	[v]	15:11:36.701 +020	15:11:36.701 +020	15:11:36.613 +020	15:12:22.910 +020
15	anto	antosavi@gmail.com	1	[N] MAX	100(2/2)	[v]	15:13:58.708 +020	15:13:58.708 +020	15:13:58.706 +020	15:14:31.726 +020
16	anto	antosavi@gmail.com	1	[N] 1	100	[v]	12:47:21.288 +020	12:47:21.288 +020	12:47:21.089 +020	12:48:27.741 +020
17	ANTONIO	antosavinoprova@gmail.com	1	[N] 1	100	[v]	18:11:53.619 +020	18:11:53.619 +020	18:11:53.538 +020	18:12:50.141 +020
18	ANTONIO	antosavinoprova@gmail.com	1	[N] MAX	100(2/2)	[v]	18:14:28.848 +020	18:14:28.848 +020	18:14:28.845 +020	18:14:53.626 +020
19	ANTONIO	antosavinoprova@gmail.com	1	[N] MAX	75(0/2)	[]	18:18:50.420 +020	18:18:50.420 +020	18:18:50.416 +020	18:19:21.899 +020
21	ANTONIO	antosavinoprova@gmail.com	1	[N] MAX	75(0/2)	[]	18:20:05.770 +020	18:20:05.770 +020	18:20:05.768 +020	18:20:33.531 +020
23	ANTONIO	antosavinoprova@gmail.com	1	[N] 1	100	[v]	18:49:40.604 +020	18:49:40.604 +020	18:49:40.566 +020	18:50:59.868 +020
24	ANTONIO	antosavinoprova@gmail.com	1	[N] MAX	100(2/2)	[v]	19:01:43.141 +020	19:01:43.141 +020	19:01:43.136 +020	19:02:31.055 +020
28	ANTONIO	antosavinoprova@gmail.com	1	[N] 1	100	[v]	12:13:32.141 +020	12:13:32.141 +020	12:13:32.130 +020	12:14:23.276 +020

Figura 5.6: Risultato tabella /games Postgres

le intenzioni dei gruppi che ci hanno preceduto. È importante evidenziare che, durante lo sviluppo e i test iniziali, abbiamo controllato manualmente le tuple inserite nel database al fine di garantire un'adeguata integrazione con le API previste nel codice del gruppo A7.

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".

5.3 Testing automatizzato con Selenium

Abbiamo impiegato Selenium (Sez. 2.2.8), un framework di automazione dei test per applicazioni web, al fine di eseguire una serie di test che dimostrassero l'integrazione riuscita delle funzionalità Bos-

srush di A6 in A7. Selenium è uno strumento potente che consente di simulare l'interazione dell'utente con un'applicazione web, consentendo di automatizzare il processo di testing e di eseguire facilmente test ripetibili su diverse configurazioni e ambienti.

Il nostro approccio di testing ha compreso due test principali. Il primo test (Fig. 5.7) ha verificato non solo le funzioni dell'editor 1VS1, ma ha anche testato con successo il corretto funzionamento del processo di login. Questo è stato cruciale per garantire che gli utenti possano accedere e utilizzare le funzionalità integrate senza intoppi.

```
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 270.0 s -- in com.example.EditorTest
[INFO] Results:
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 04:50 min
[INFO] Finished at: 2024-05-16T13:55:58+02:00
[INFO] -----
```

Figura 5.7: Risultato Test 1vs1

Il secondo test (Fig. 5.8) si è concentrato sulla modalità Bos-sRush, verificando che l'integrazione abbia mantenuto l'esperienza fluida e le performance ottimali anche in questa modalità di gioco più complessa. Mediante l'uso di Selenium, siamo stati in grado di eseguire questi test in modo automatizzato e sistematico, garantendo che l'integrazione fosse completa e senza difetti.

```
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 301.4 s -- in com.example.EditorTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO]
[INFO] Total time: 05:04 min
[INFO] Finished at: 2024-05-16T18:22:31+02:00
[INFO] -----
```

Figura 5.8: Risultato Test BossRush

Di seguito (Fig. 5.9) viene presentato un unico report che elenca i casi di test. Poiché l'obiettivo principale è l'integrazione, questo documento include sia i test per le funzionalità 1VS1 sia quelli per BossRush.

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 5.9: Casi di Test

5.3.1 Requisiti per il testing automatizzato

- Scaricare Google Chrome in wsl:
 - ‘`wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb`’
 - ‘`sudo apt -y install ./google-chrome-stable_current_amd64.deb`’
 - ‘`google-chrome --version`’
- Scaricare Chromedriver per linux64:
 - ‘`wget https://storage.googleapis.com/chrome-for-testing-public/<num_version>/linux64/chromedriver-linux64.zip`’
 - ‘`unzip chromedriver_linux64.zip`’
 - ‘`sudo mv chromedriver /usr/local/bin/`’

Capitolo 6

Sviluppi Futuri

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

6.1 Login Admin

Il login dell'amministratore al momento è presente, ma è possibile accedere alla pagina della homepage dell'admin senza effettuare il login. Questa situazione potrebbe consentire l'accesso a dati sensibili da parte di utenti non autorizzati. Una soluzione è replicare il meccanismo del token, già implementato per la parte studente al fine di avere una reale protezione anche lato admin.

6.2 Registrazione Admin

Al momento la registrazione dell'Admin è prevista per qualsiasi utente. In futuro si potrebbe implementare la conferma di registrazione tramite email e l'inserimento di un token specifico per la registrazione, in modo che solo gli utenti in possesso del token possano registrarsi come Admin.

6.3 User Friendliness

Per migliorare l'usabilità dell'applicazione, 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.

6.4 Diverse modalità di gioco

Potrebbe essere utile considerare l'implementazione di una modalità "Multiplayer" che consenta a più giocatori di partecipare con-

temporaneamente 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 di selezione della modalità di gioco dell'applicazione dal gruppo A7, per cui è necessario effettuare solo l'implementazione vera e propria della modalità.

6.5 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 uten-

te 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.

Capitolo 7

Installazione e Problemi Noti

7.1 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 WSL (Windows Subsystem for Linux), il 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.

7.2 Guida all'Installazione

Ci sono 3 passi da seguire per l'installazione:

1. Effettuare il download e l'installazione di Docker Desktop. Nel caso non sia la prima installazione dell'app è necessario effettuare la disinstallazione utilizzando *uninstaller.bat* mentre si ha in esecuzione 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, 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 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*".

Alla fine dell'installazione si avrà:

- la creazione della rete *global-network* comune a tutti i container,
- la creazione dei volumi *VolumeT8* e *VolumeT9*,
- la creazione dei singoli container in Docker Desktop,
- la configurazione del container *manvsclass-mongo db-1*.

3. 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.

N.B. È normale che il container *progetto-sad-g19-master* resti sempre in stato di *exited*.

7.3 Modificare il Codice

Al fine di rendere più agevoli eventuali cambiamenti, mostriamo una serie di passi da seguire per modificare il codice:

1. in Docker Desktop, recarsi nella sezione relativa ai containers;
2. selezionare i container relativi alle porzioni di codice modificate ed eliminarli;
3. recarsi nella sezione relativa alle immagini;
4. selezionare le immagini relative ai containers precedentemente eliminati ed effettuarne la cancellazione;
5. sull'IDE utilizzato, aprire il terminale integrato nella cartella ove è presente il file "pom.xml" e digitare "*mvn clean package*";
6. fare clic sul tasto destro sul file "docker-compose.yml" e selezionare "*compose up*";

7. riavviare i containers.

N.B. Eseguire il passo 5 solo nel caso in cui il file "pom.xml" sia presente.

7.4 Problematiche Note

Durante l'utilizzo dell'applicazione o di una modifica potrebbero presentarsi non pochi problemi. I più comuni e facilmente risolvibili sono illustrati nei seguenti paragrafi.

7.4.1 Versione di Ubuntu

Durante le varie installazioni, potrebbe capitare che Ubuntu aggiorni le proprie versioni facendo sì che le installazioni non vadano a buon fine. Questo problema si è verificato, in particolare, nel task **T1**, dove nel *Dockerfile* avevamo "FROM ubuntu:latest", che installa l'ultima versione disponibile di Ubuntu. Per risolvere tale bug abbiamo specificato la versione da utilizzare, correggendo in "FROM ubuntu:18.04".

7.4.2 Cache Browser

Pulendo 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 pulirla 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. In *Safari* è possibile svuotare la Cache in Sviluppo ->Svuota la Cache, oppure, in altri Browser (come Firefox), è possibile far in modo che questa venga eliminata all'uscita.

7.4.3 Porte in uso

Durante l'avvio dell'applicazione, potrebbe verificarsi un problema per 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 per individuare il processo che sta attualmente utilizzando la porta in questione e terminarlo. In alternativa, è possibile risolvere questo problema modificando la porta del container in questione per evitare conflitti.

7.4.4 502 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 que-

sto tipo di errore potrebbe essere correlata alla rapidità con cui si tenta di accedere all'applicazione 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. Attendere alcuni istanti prima di tentare di accedere può spesso risolvere il problema. Tuttavia, se l'errore persiste, è consigliabile far ripartire tutti i container (dopo averli stoppati) oppure riavviare Docker.

Glossario

Admin

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

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.

Robot

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

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.

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.