



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

SOFTWARE ARCHITECTURE DESIGN
MODALITA' SCALATA

Professoressa:
Anna Rita Fasolino

Studenti:

Flavio Filippo Parrotta M63001807, fl.parrotta@studenti.unina.it
Francesco Floriano M63001765, f.floriano@studenti.unina.it
Simone De Lucia M63001720, simon.delucia@studenti.unina.it

URL del repository GitHub:
https://github.com/SimoneDeLucia/A13_R7

Indice

Indice	1
1 - MAN VS AUTOMATED TESTING	3
1.1 PUNTO DI PARTENZA	3
1.2 SERVIZI PER LA SCALATA	7
1.3 PROBLEMATICHE TROVATE	8
1.3.1 Issue Documentate	8
1.3.2 Issue Riscontrate	8
1.4 ANALISI DEI REQUISITI	11
1.5 GLOSSARIO DEI TERMINI	12
1.6 PROCESSO DI SVILUPPO	14
1.7 USER STORIES	16
1.7.1 CRITERI DI ACCETTAZIONE DELLE USER STORIES	16
2 - ANALISI DEI REQUISITI	18
2.1 DIAGRAMMI UML SCALATA	18
2.2 SCENARI SCALATA	19
2.2.1 CreaScalata	19
2.2.2 IniziaPartita	20
2.2.3 Scegli Scalata	21
2.3 SEQUENCE DIAGRAM SCALATA ALTO LIVELLO	21
2.3.1 ScegliScalata	21
2.3.2 CreaScalata	22
2.3.3 IniziaPartita	23
2.4 WORKFLOW DIAGRAM SCALATA	23
3 - ANALISI D'IMPATTO	25
4 - PROGETTAZIONE DELLA SOLUZIONE	26
4.1 - DESCRIZIONE DECISIONI DI PROGETTO	26
4.1.1 - Panoramica	26
4.1.2 – Decisioni in T1	26
4.1.3 – Decisioni in T5	27
4.2 - MODIFICHE AL DATABASE(T4)	29
4.3 - DIAGRAMMI DI DETTAGLIO STATICI	30
4.3.1 Component Diagram	30
4.3.2: Package Diagram T1	31
4.3.3 Package Diagram T5	31
4.4 - DIAGRAMMI DI DETTAGLIO DINAMICI	32
4.4.1: Sequence Diagram di dettaglio – CreaScalata	32

4.4.2: Sequence Diagram di dettaglio – Scegli Scalata(vecchia versione)	34
4.4.4: Sequence Diagram di dettaglio – Scegli Scalata(nuova versione)	35
4.4.5 Sequence Diagram: Inizia Partita (Vecchia versione)	37
4.4.6 Sequence Diagram: Inizia Partita (Nuova versione)	38
4.5 Chiamate API	38
5 – IMPLEMENTAZIONE	39
5.1 - T1 Modifiche e Aggiunte	39
5.2 - T4 Modifiche e Aggiunte	41
5.3 - T5 Modifiche e Aggiunte	42
6 – TESTING	49
6.1 - TESTING MOCK	49
6.2 - TESTING MANUALE	49
6.3 - USER SESSION-BASED TESTING	56
7 – STRUMENTI UTILIZZATI	57
8 - SVILUPPI FUTURI	60
8.1 Fix necessari	61
8.2 Possibili aggiunte	61
Appendice	62
Appendice A: ChangeLog	62
Appendice B: Riferimenti a scalata	72

1 - MAN VS AUTOMATED TESTING

Il progetto a cui abbiamo lavorato utilizza il concetto di gamification, una pratica che consiste nel trasformare attività lavorative o di studio, in giochi, la cui soluzione può rappresentare una possibile soluzione al problema o un modo per imparare.

Da questo nasce il gioco interattivo: “**Man vs Automated Testing Tools challenges**”, che vede *players* competere a colpi di test progettati mediante il framework JUnit, contro dei robot capaci di generare automaticamente tali test con differenti livelli di copertura, che il player dovrà superare per potersi definire il vincitore.

1.1 PUNTO DI PARTENZA

L'applicazione è basata su un'architettura a microservizi, ovvero basata sulla gestione dei vari servizi completamente indipendenti gli uni dagli altri, così da aumentare la scalabilità del sistema, grazie alla possibilità di scalare istanze dei singoli servizi e non istanze multiple dell'intera applicazione, e da semplificare il rilascio e la manutenzione dei servizi.

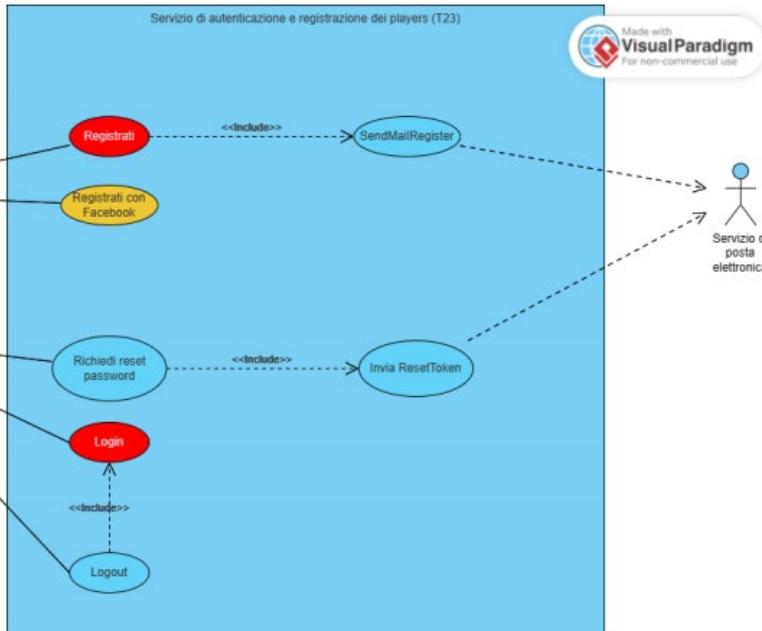
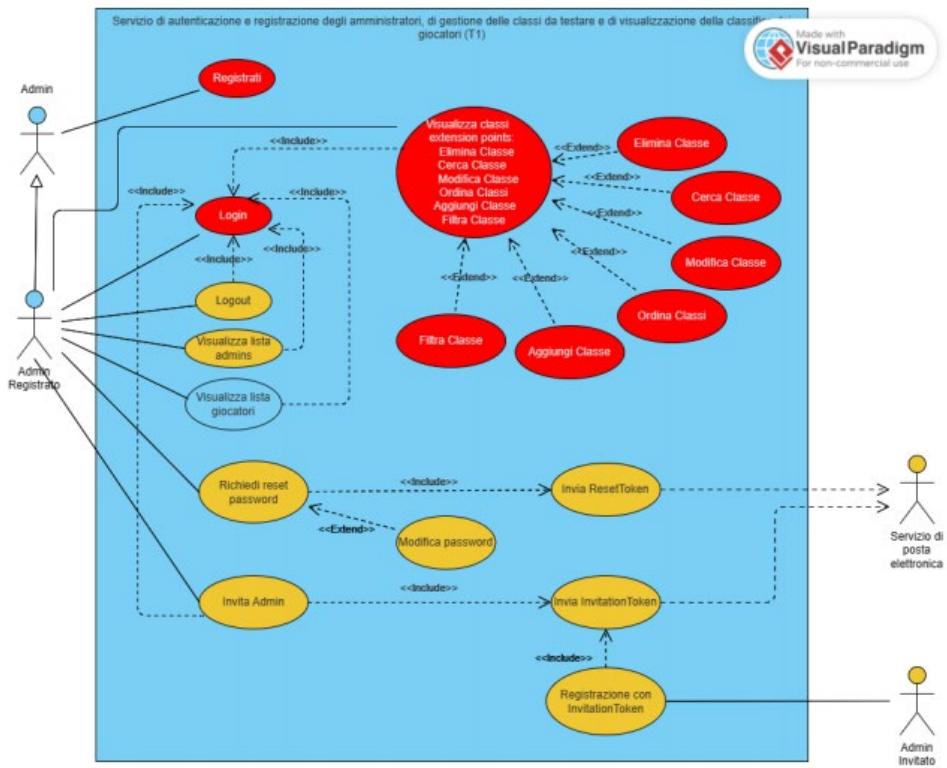
In particolare, nel corso dello sviluppo di questo progetto sono stati implementati nove servizi, che andiamo a ricapitolare brevemente:

- T1 Servizio di gestione delle classi da testare e visualizzazione della classifica dei *players*
- T23 Servizio di autenticazione e registrazione dei *players*
- T4 Servizio di repository dei dati di gioco
- T5 Front-end avvio partita
- T6 Front-end per giocare una partita
- T7 Servizio di compilazione ed esecuzione
- T8 Servizio robot EvoSuite
- T9 Servizio robot Randoop

Il lavoro che andremo ad esporre all'interno di tale elaborato, pone le sue fondamenta sulle ultime documentazioni: A13 e A13-Refactoring (2024): la prima è una versione migliorativa dei requisiti sviluppati in corrispondenza dei servizi T1 e T23, in particolare l'obiettivo era di realizzare un sistema focalizzato su un'autenticazione robusta, una gestione avanzata degli utenti e amministratori, e una user experience migliorata. Elenchiamo a grandi linee le modifiche che sono state fatte.

Requisito	Descrizione
Autenticazione e sicurezza avanzata	A) Fase di registrazione e login migliorata. B) Introduzione di un sistema di token per autenticare gli amministratori e implementazione di misure di sicurezza avanzate per proteggere le password archiviate attraverso algoritmi di hashing.
Gestione degli amministratori	A) Creazione di una sezione riservata per visualizzare la lista degli amministratori registrati e meccanismo per invitare nuovi amministratori. B) Differenziazione tra amministratori con privilegi specifici e semplici giocatori.
Esperienza utente migliorata	A) Feedback visivo per confermare l'avvenuta registrazione, seguito da un reindirizzamento alla pagina di login. B) Introduzione del social login tramite Facebook.

Le modifiche all'interno del sistema a cui i requisiti portano sono ben visualizzate utilizzando un diagramma dei casi d'uso per ognuno dei servizi interessati.



Con l'introduzione del meccanismo di autorizzazione ed autenticazione mediante token JWT lato *amministratore*, è stato fornito un "confine"隐式 di sicurezza in grado di distinguere tra le funzionalità di autenticazione dei *players* rispetto a quelle riservate dagli admin.

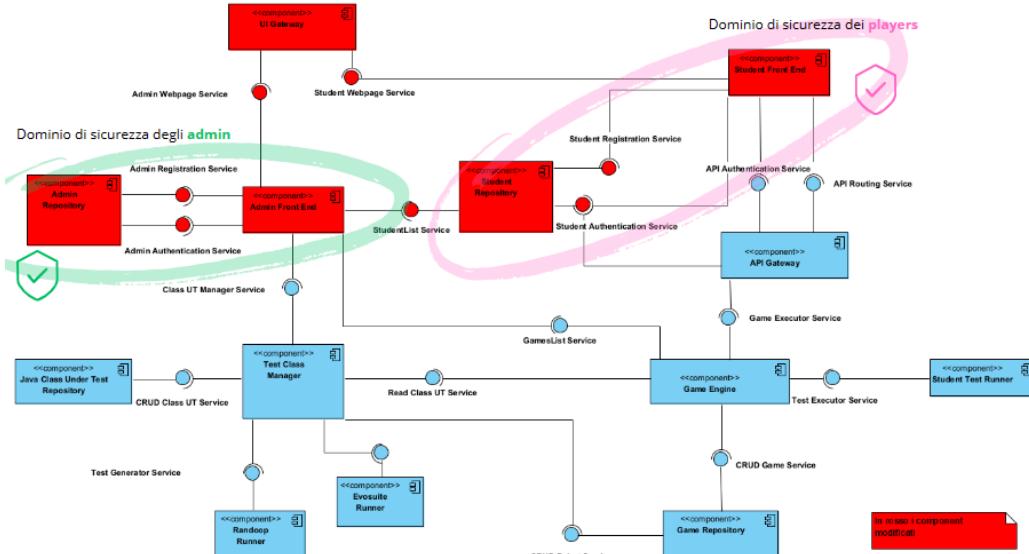


Diagramma delle componenti generale del sistema

Il refactoring riguarda invece lo sviluppo lato front-end del sistema di gioco, specificatamente al Task 5 e Task 6. Il Task 5, come descritto in precedenza, consente ai giocatori autenticati di accedere all'area riservata, selezionare i parametri di gioco (come classi e robot caricati dagli amministratori) e avviare una partita nell'arena di gioco. Il Task 6, invece, offre ai giocatori un editor di test case per scrivere codice Java, richiedere la compilazione, visualizzare i risultati di copertura e confrontare i propri risultati con quelli generati dal robot, decretando così il vincitore. L'applicazione è stata ristrutturata introducendo tre package principali:

Interface:

- Gestisce le interazioni con i servizi REST attraverso il *Service Manager*, un dispatcher centralizzato.
- Facilita l'aggiunta di nuovi task e chiamate REST.
- Permette di combinare e sfruttare i servizi in modo modulare e flessibile.

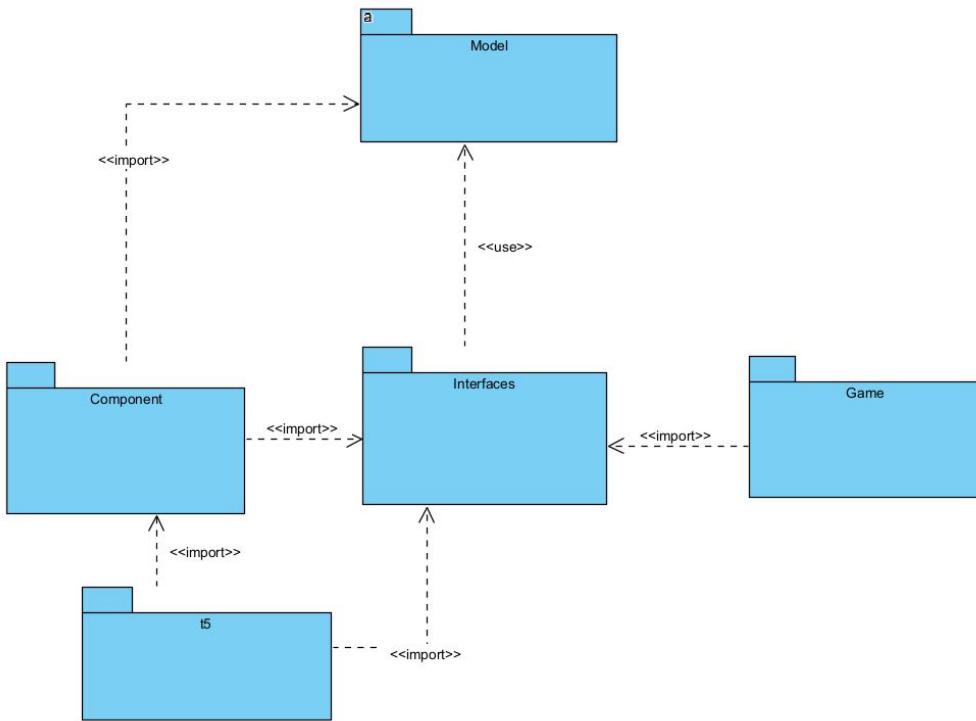
Component:

- Consente la costruzione di pagine web modulari, separando la logica di business dai dati da visualizzare.
- Integra i componenti con il *Service Manager* per migliorare l'interoperabilità.
- Trasforma il precedente *Gui Controller* in un controller dedicato a routing e gestione delle sessioni, migliorando la struttura e l'organizzazione.

Game:

- Gestisce la logica di gioco con un controller REST per la concorrenza e le partite attive.
- Introduce la classe *GameLogic* per definire facilmente nuove modalità di gioco.
- Ristruttura il precedente *GameEngine* frammentato, creando un backend scalabile e robusto.
- Rende l'editor di test case neutrale e riutilizzabile per tutte le modalità di gioco.

Il diagramma dei pacchetti fornisce una visione d'insieme dell'organizzazione del sistema, evidenziando la suddivisione in moduli distinti, di cui sono stati forniti anche i diagrammi delle classi.



Le attività di cui ci dovremo occupare riguardano invece una sezione specifica del progetto, ovvero la Modalità Scalata, dove un giocatore può cimentarsi in una partita a più round, a difficoltà crescente e contro diverse classi di test. Alcune informazioni sulla scalata fanno riferimento ad una documentazione risalente all'anno 2023/2024, dove vengono descritte le chiamate api effettuate.

/scalata					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la pagina di configurazione della nuova modalità di gioco denominata: "Scalata"	showGamePageScalata			"302": re-indirizzamento alla pagina di configurazione della modalità "Scalata" (/scalata) "302": re-indirizzamento pagina di login (/loginAdmin) (token JWT invalido)
/configureScalata					
POST	Richiesta di configurazione e conseguente creazione della propria "Scalata"	uploadScalata		required: true content: Scalata	"200": Salvataggio avvenuto correttamente all'interno del DB "500": Attenzione, non sei loggato (token JWT invalido)
/scalate_list					
GET	Recupero di tutte le "Scalate" memorizzate nel sistema	listScalate			"200": recupero delle "Scalate" memorizzate nel sistema avvenuto correttamente
/delete scalata{scalataName}					
DELETE	Rimozione di una specifica "Scalata" memorizzata nel sistema	deleteScalataByName	scalataName (nome "Scalata")	required: true content: Scalata	"200": "Scalata" rimossa con successo "500": Attenzione, non sei loggato (token JWT invalido) "404": "Scalata" specificata non trovata
/retrieve scalata{scalataName}					
GET	Recupero di una specifica "Scalata" memorizzata nel sistema	retrieveScalataByName	scalataName (nome "Scalata")		"200": recupero della "Scalata" memorizzata nel sistema avvenuto correttamente "404": "Scalata" specifica non trovata

Da questa base di partenza, i nostri obiettivi si concentreranno su alcuni aspetti di giocabilità, lì dove la modalità scalata è stata poco toccata dalle documentazioni precedenti, aspetti di personalizzazione e aspetti di performance e robustezza, di cui parleremo più nel dettaglio più avanti.

Task da svolgere R7: (Miglioramento Modalità di gioco della Scalata)

Task	Descrizione	Team
Task R7	<p>Rivedere la modalità di gioco di scalata e renderla più configurabile rispetto ai Robot e alla loro difficoltà</p> <p>Salvare nel database del gioco T4 anche i Robot contro cui hanno giocato</p>	

1.2 SERVIZI PER LA SCALATA

Dopo questa attenta analisi di tutti i servizi presenti, si evince che quelli di nostro interesse sono principalmente tre: T1, T4 e T5. In tabella è possibile visualizzare più dettagliatamente i compiti di ogni task, così da capire meglio cosa dovremo andare a toccare.

ID TASK	Descrizione
T1	<p>Il servizio permette agli amministratori di: registrarsi ed effettuare la procedura di login.</p> <p>Il servizio permette agli amministratori, interagendo con un opportuno cruscotto, di: (1) caricare, modificare, ordinare, filtrare, ricercare, scaricare ed eliminare classi di programmazione da testare all'interno del catalogo, (2) visualizzare una classifica dei <i>players</i>.</p> <p>Il servizio permette ai <i>players</i> correttamente autenticati, di: visualizzare tutte le classi precedentemente caricate dagli <i>amministratori</i>.</p>
T4	Il servizio permette al game engine di: (1) creare una partita memorizzando una serie di informazioni, (2) aggiornarla per recuperare la data e l'ora di inizio e fine, (3) eliminarla, (4) generare, contestualmente la partita appena creata, i round, (5) aggiornali, (6) creare, nell'ambito dei round, diversi turni associati a ciascun partecipante della partita, (7) aggiornarli (8) recuperare, durante ciascun round, i punteggi prodotti dai robot relativi la classe di test in gioco.
T5	Il servizio permette ai players correttamente autenticati di: (1) accedere all'area riservata di selezione dei parametri di gioco dove poter visualizzare le classi ed i robot disponibili, precedentemente caricati dagli <i>amministratori</i> , (2) avviare una partita accedendo all'arena di gioco.

1.3 PROBLEMATICHE TROVATE

Nelle prime fasi di sviluppo del progetto, si è deciso di interagire con la piattaforma per ricostruire, quanto più accuratamente possibile, i vari scenari di uso previsti in precedenza, in particolare quale di questi dessero il maggior numero di problemi. Un elenco di foto e video delle problematiche riscontrate e riprodotte può essere recuperato [qui](#).

1.3.1 Issue Documentate

Tra le issue documentate nell'apposita sezione del Github, si è stato in grado di riprodurre e trovare una risoluzione alle seguenti:

1. #ISSUE 38: ERRATO UTILIZZO DELLE FINESTRE CONSOLE E RESULTS

[Scalata] Errato utilizzo delle finestre Console e Results #38

 Open



reverse-unina opened on Oct 24, 2024

Describe the bug

Durante la scalata viene utilizzata la finestra Console per riportare i risultati di copertura anziché per riportare l'esito della compilazione. È un problema particolarmente grave in caso di errore poiché non c'è possibilità di valutare l'errore.

To Reproduce

Steps to reproduce the behavior:

1. Avvia un partita in modalità scalata
2. Play
3. Aggiungi dei test (con errore di sintassi)
4. L'interfaccia ammette l'esistenza di un errore ma nella finestra Console non viene riportato l'esito della compilazione

Expected behavior

L'esito della compilazione dovrebbe sempre essere nella finestra Console, eventualmente sovrascrivendo i risultati precedenti.

2. #ISSUE 40: MANCATO AGGIORNAMENTO DELLA GUI

[Scalata] Mancato aggiornamento della GUI #40

 Open



reverse-unina opened on Oct 24, 2024

...

Is your feature request related to a problem? Please describe.

La GUI della scalata (localhost/editor_old) non è allineata rispetto ai miglioramenti grafici e tecnici dell'interfaccia principale (localhost/editor)

Describe the solution you'd like

Le due interfacce dovrebbe essere allineate in tutti i particolari tecnici, ad eccezione ovviamente del diverso comportamento del pulsante Submit.



reverse-unina added

[enhancement](#)

[refactoring](#)

on Oct 24, 2024

1.3.2 Issue Riscontrate

T1- Sezione Crea Scalata Poco Robusto

T1 presenta diverse problematiche di robustezza: la creazione parte dall'inserimento degli attributi, tra i quali il nome che, per un vincolo già presente, dovrebbe essere univoco: tale vincolo viene però confrontato alla fine dell'operazione, non garantendo efficienza e

robustezza; un altro esempio è il tasto aggiungi che si dovrebbe disabilitare quando si passa a selezionare le classi.

T1- La cancellazione di una classe non cancella a cascata una scalata che la contiene

All'atto della cancellazione di una classe dal lato admin, non si tiene in considerazione se la stessa classe è presente in una scalata. Sarebbe quindi possibile eliminare una classe di test presente però in una scalata creata in precedenza. Questo può risultare in problemi di inconsistenza nel caso in cui dovesse essere eseguita una scalata senza una classe cancellata al suo interno

T1-T5 Coperture mancanti per alcune classi

All'interno di alcune classi sono assenti delle coperture relative a robot e difficoltà. Può risultare grave nel momento in cui un giocatore decidesse di giocare una partita contro un determinato robot a una determinata difficoltà, dove in realtà questa non è disponibile. Va aggiunto che il problema è stato riscontrato con il materiale delle classi di test forniti all'interno del progetto; tuttavia, l'assenza di controlli all'interno del menù sfida in T5 relativi a tali condizioni risulta a tutti gli effetti una problematica.

T1-T5: Utilizzo di classi in T1 non salvate in maniera persistente

All'atto di avvio di una partita, viene effettuato un download del file della classe da parte di T5 all'interno di una cartella chiamata FilesUtil, archiviata all'interno del T1 in maniera non persistente (come, ad esempio, i volumi T8 e T9): questo porta a errori di visualizzazione della classe nel momento in cui il T1 dovesse subire modifiche e venire ricompilato.

Ad ora, questo issue è rimasto irrisolto in questa versione(si rimanda agli sviluppi futuri)

T4- Mancato salvataggio dei robot durante la partita

In T4, come vediamo dai task che dobbiamo implementare, è richiesto il salvataggio, durante lo svolgersi di una partita in modalità scalata, dei robot contro cui l'utente gioca in ogni round della scalata: infatti allo stato attuale non vengono salvati i robot contro cui gioca in ogni round, evincendo che manca una relazione tra robot e round. Il salvataggio sarebbe invece molto utile, poiché permette di tracciare con precisione il percorso e la progressione dell'utente nella scalata, facilitando analisi sulle performance e sulle difficoltà incontrate. Tuttavia, tale relazione è presente all'interno della documentazione dove, tra l'altro, la relazione tra robot e round riscontrata al suo interno ci risulta errata: secondo noi, un robot può essere associato a più round, ma un round può avere al suo interno un solo robot.

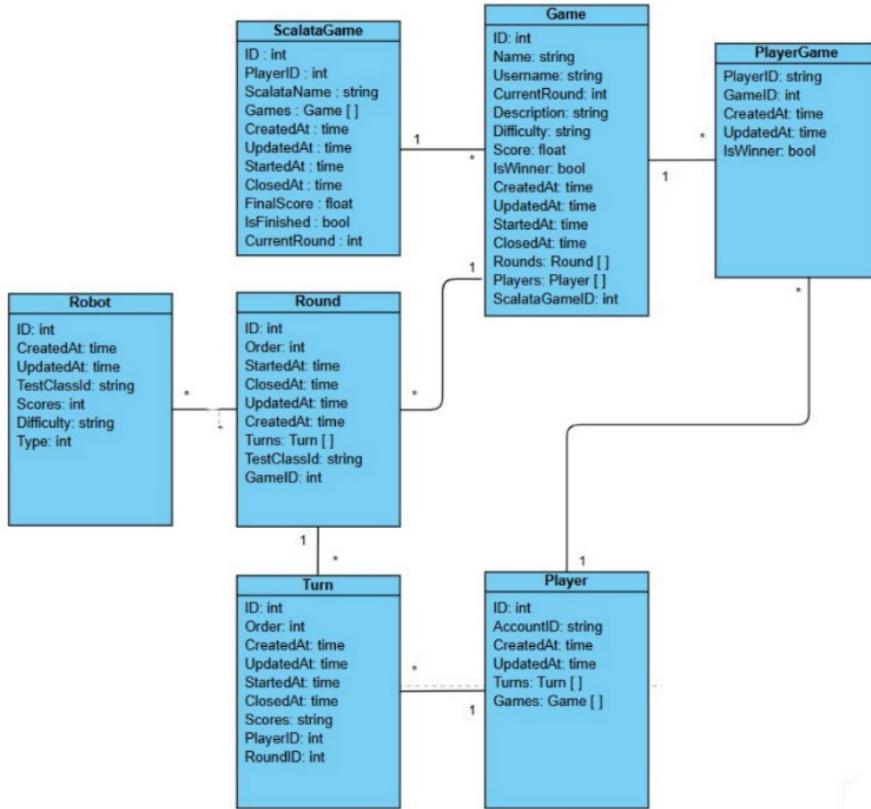


Diagramma delle classi all'interno di T4, ottenuto dalla documentazione fornita.

T5- Passaggio cablato di robot e difficoltà durante la scalata

Sicuramente il problema di maggior interesse è dovuto dalla presenza del passaggio cablato del robot evoSuite di difficoltà 1 all'atto della creazione della partita: ciò verrà riscontrata anche all'atto della creazione di partite successive.

```

localStorage.setItem("scalataId", result.scalataGameId);
localStorage.setItem("difficulty", 1);
localStorage.setItem("robot", "evoSuite");
localStorage.setItem("current_round_scalata", 1);
localStorage.setItem("scalata_name", selectedScalata);

retrieveScalata(selectedScalata)
.then( data => {return createGame("evoSuite", data[0].selectedClasses[0], 1, result.scalataGameId, username, "Scalata")})

```

Impostazione cablata della difficoltà e del robot all'interno della creazione della prima partita della scalata(gamemode_scalata.js riga 161-170)

```

//The player has completed the round, not the Scalata
swal("Complimenti!", `Hai completato il round ${current_round_scalata}/${total_rounds_scalata}!` + `${displayRobotPoints}` , "success").then(
  current_round_scalata++;
  localStorage.setItem("current_round_scalata", current_round_scalata);
  classe = getScalataClasse(current_round_scalata-1, localStorage.getItem("scalata_classes"));
  localStorage.setItem("classe", classe);
  console.log("[editor.js] classes in scalata: "+localStorage.getItem("scalata_classes")+"\n\
    selected class: "+classe);
  incrementScalataRound(localStorage.getItem("scalataId"), current_round_scalata).then((data) => {
    console.log("[editor.js] Creating new game for next round in scalata with parameters: \
      Robot: evosuite\n\
      Classe: "+classe+"\n\
      Difficulty: 1\n\
      ScalataId: "+localStorage.getItem("scalataId")+"\n\
      Username: "+localStorage.getItem("username")+"." );
    createGame("evosuite", classe, 1, localStorage.getItem("scalataId"), localStorage.getItem("username"),localStorage.getItem("modalita"));
    console.log(data);
    window.location.href = "editor_old";
  })
)
Impostazione cablata della difficoltà e del robot all'interno delle partite successive della scalata(editor.js  
riga 391-408)

```

Questa forzatura impedisce la possibilità di personalizzare il robot e la difficoltà all'interno della scalata. Consultando l'infografica per la modalità, ci si è resi conto che è stata una scelta di design, ma che va considerata come problematica in quanto in contrasto con la richiesta dei requisiti. La scelta è stata interpretata come una mancanza di passaggio dei parametri di robot e difficoltà all'interno delle singole classi che compongono la scalata.

T5- Assenza di robustezza e di semplicità in editor_old

Analizzando la scalata attualmente implementata lato frontend, notiamo un problema di design della pagina ‘modalitàScalata’: per tornare indietro alla schermata con le varie modalità, abbiamo invece un tasto “Logout”; abbiamo poi difetti sulla situazione ‘FinePartita’: se vinciamo rimaniamo bloccati in quello stato, mentre se perdiamo ritorniamo direttamente alla schermata iniziale, quando avrebbe molto più senso ritornare alla modalità scalata, così da poter ritentare; dopo aver vinto un round bisogna cliccare più volte il tasto submit per poter proseguire nella scalata.

T5- Le sfide non vengono salvate in T4

Le partite svolte in modalità scalata venivano correttamente memorizzate in database (seppure non riportando il robot e la difficoltà all'interno del round stesso), tuttavia ciò non accadeva per la sfida classica.

1.4 ANALISI DEI REQUISITI

Oltre ai difetti discussi al paragrafo precedente, si è pensato di aggiungere una maggior personalizzazione alla modalità stessa della scalata, non solo in base alla difficoltà della classe intrinseca inserita nel round, ma anche dal robot da affrontare e la difficoltà del robot da affrontare. Da tutte le criticità emerse dalla situazione di partenza, sono stati proposti i seguenti requisiti funzionali per migliorare la modalità scalata:

1. Il sistema deve fornire all’utente la possibilità di scegliere tra più scalate inserite dall’amministratore con annesse informazioni sul tipo di scalata. (T56) [Priorità 2]
2. Il sistema deve poter fornire all’amministratore, all’atto della creazione di una scalata, oltre alla difficoltà delle classi, anche le coverage dei robot. (T1) [Priorità 3]

3. Il sistema deve fornire all'amministratore la possibilità di recuperare le coverage delle varie classi risolte dai robot. (T1<->T4) [Priorità 5]
4. Il sistema deve fornire all'amministratore la possibilità di creare una SCALATA PERSONALIZZABILE, con un ordine delle classi ad ogni round scelto da lui. (T1) [Priorità 5]
5. Il sistema deve permettere all'amministratore di personalizzare livello e robot di ciascuna classe all'interno della scalata che sta creando. (T1) [Priorità 4]
6. Il sistema deve permettere all'amministratore di poter scegliere una SCALATA PREDEFINITA, scegliendo un unico tipo di robot e definendo le difficoltà associate. (T1) [Priorità 2]
7. Il sistema deve variare il robot utilizzato, in modo da incrementare la difficoltà della scalata. (T56) [Priorità 5]
8. Il sistema deve poter memorizzare al suo interno i robot contro cui l'utente ha giocato durante la scalata(T4) [Priorità 5]
9. Il sistema deve dare la possibilità all'utente di ritornare al menù scalata nel caso in cui l'utente perda una partita (T56) [Priorità 1]

Per la memorizzazione della scalata, nelle fasi di creazione e di registrazione di statistiche, sono stati pensati i seguenti requisiti sui dati

1. All'atto della creazione, la scalata sarà formata da un ID, un nome, una descrizione, un numero di round, e dalle classi selezionate. Per ciascuna classe selezionata, ne va specificato il robot e la difficoltà.
2. Durante una partita, la scalata deve avere un identificativo, un id del giocatore, un nome, una data di creazione, di aggiornamento e di chiusura, un punteggio, un booleano per identificarne la fine, e un contatore che tiene traccia del numero di round attuale. Il game, se si tratta di una scalata, deve avere al suo interno un identificativo della scalata. Ogni round al suo interno deve tener traccia di un identificativo del robot che si va ad affrontare. Un robot è caratterizzato da un identificativo, da una data di creazione, da un id della classe, da un punteggio, da una difficoltà e da un tipo (0 per randoop, 1 per evosuite)

1.5 GLOSSARIO DEI TERMINI

Il glossario è solo ai fini di presentare ciò che è parso di capire a noi dell'applicazione nella sua interezza: si è scelti di andare a coprire aspetti ben definiti, per motivi di chiarezza, ma soprattutto di aggiungere altri termini ed approfondire quelli esistenti, per evitare il più possibile disambiguazioni.

Per introdurre i requisiti sopra definiti, è stato necessario porre una maggior attenzione su determinati termini.

Termino	Descrizione	Sinonimi
Utente	Persona che ha l'accesso al sito e può giocare alle modalità proposte dal sito.	Utente registrato
Amministratore	Persona che ha l'accesso al sito in modalità privilegiata. Tra le altre cose, esso può aggiungere classi e creare scalate.	Admin, Amministratore registrato

Classe	<p>Ci si riferisce a una classe scritta nel linguaggio Java, con i suoi metodi all'interno che sono oggetto di testing.</p> <p>La dimensione di una classe è definita in righe di codice</p> <p>All'interno del sistema, una classe fornisce il suo nome, una data di caricamento, una descrizione, tre categorie, il suo codice sorgente in Java, e i testing effettuati da Randoop ed Evosuite</p>	
Robot	<p>Generatore di test delle classi usato per confrontare il suo risultato con quello dell'utente, per decretare il vincitore della partita.</p> <p>Un robot può avere dei livelli di difficoltà (Facile, medio, difficile).</p> <p>Nel sistema attualmente sono presenti due robot: Randoop ed EvoSuite.</p> <p>È stato specificato che, solitamente, Evosuite sia più impegnativo da affrontare rispetto a Randoop.</p>	
Testing	<p>Ci si riferisce in particolare al testing di unità sulla classe proposta, ovvero verificare che il comportamento dei metodi della classe in esame corrisponda a un comportamento atteso.</p> <p>In questo caso, del testing di unità viene valutata la copertura delle righe di codice.</p>	Test, testare
Copertura delle righe di codice	<p>Misura utilizzata per verificare quante linee di codice della classe sono state eseguite (o coperte) dai test scritti dall'utente, sulla totalità delle linee di codice della stessa che è possibile eseguire.</p> <p>Viene fornita in percentuale</p>	Coverage
Difficoltà	<p>Indicatore utilizzato per quantificare l'impegno richiesto dall'utente per poter superare con esito positivo un round.</p> <p>La difficoltà si scomponete in due fattori: Difficoltà della classe e dei robot. Anche la scelta dei robot può rappresentare un criterio di difficoltà, siccome la qualità dei test di Randoop è solitamente inferiore rispetto a quella di Evosuite</p>	Difficoltà complessiva
Difficoltà della classe	<p>Quanto impegnativa risulta essere una classe, in base al numero di metodi al suo interno e la loro complessità. Allo stato attuale, la difficoltà viene indicata da un intervallo di valori (da 1 a 3) e viene assegnata esclusivamente dall'amministratore.</p>	
Difficoltà del robot	<p>Quanto impegnativo risulta il robot da battere, in base alla copertura delle linee di codice che riesce a raggiungere.</p>	
Round	<p>Una sfida tra utente e robot sul testare una classe presente nel sistema.</p> <p>Il vincitore viene decretato chi, tra il robot e l'utente, copre il maggior numero di righe di codice con i propri test.</p>	
Turno	<p>Un tentativo di compilazione per un confronto tra utente e robot durante uno specifico round.</p>	
Partita	<p>Modalità di gioco composta da un solo round, in cui l'utente può sfidare uno dei due robot in una delle difficoltà proposte.</p>	Partita Singola, sfida

	In una partita l'utente può selezionare il robot e la sua difficoltà.	
Scalata	<p>Modalità di gioco a round multipli in cui l'utente può sfidare il robot.</p> <p>In ciascun round, l'utente e il robot si sfidano con una classe differente man mano che la scalata progredisce.</p> <p>Una scalata è composta da almeno due classi, un nome e una descrizione.</p> <p>Si vuole specializzare la modalità scalata già presente in scalata personalizzabile e scalata predefinita.</p>	
Scalata predefinita	Una scalata in cui si affronta un solo robot impostato all'atto della creazione della scalata, con difficoltà impostate <u>dall'amministratore</u> .	Scalata singola
Scalata Personalizzabile	<p>Una scalata in cui è possibile affrontare entrambi i robot presenti con difficoltà a discrezione dell'amministratore.</p> <p>All'atto della creazione di una scalata, per ciascuna classe al suo interno vanno definiti robot e difficoltà del robot</p>	Scalata Progressiva Scalata personalizzata

1.6 PROCESSO DI SVILUPPO

Il nostro processo di sviluppo si basa sui metodi agili, un approccio moderno ed efficace che si distingue per la sua flessibilità e capacità di adattamento ai cambiamenti. La caratteristica principale di questi metodi è la possibilità di concentrarsi maggiormente sul codice e sulla funzionalità del software, piuttosto che su aspetti formali come design e documentazione.

Tutti i metodi agili si basano su uno sviluppo iterativo ed evolutivo, un principio fondamentale che garantisce la continua crescita e miglioramento del progetto. Questo approccio è alla base anche del processo di sviluppo UP (Unified Process), il quale prevede la realizzazione del sistema attraverso incrementi successivi: si sviluppa il sistema in incrementi e si valuta ogni incremento prima di procedere allo sviluppo del successivo. Questi metodi sono adatti a sviluppare applicazioni i cui requisiti cambiano rapidamente durante lo sviluppo. Abbiamo utilizzato questa strategia per una miglior organizzazione lato team e per evitare problematiche dovute a possibili cambiamenti dei requisiti; infatti, a differenza di altri modelli di sviluppo, come il tradizionale modello a cascata, i metodi agili prevedono e gestiscono attivamente le modifiche durante l'intero ciclo di vita del progetto. Nel modello a cascata, invece, i requisiti sono stabiliti fin dall'inizio e qualsiasi cambiamento successivo risulta difficoltoso e costoso da implementare. Al contrario, con i metodi agili è possibile aggiornare piani, specifiche e modelli in base ai feedback ricevuti, garantendo così una maggiore reattività e un prodotto finale più aderente alle necessità del cliente.

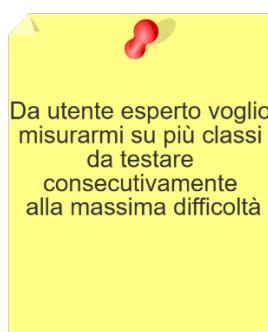
Tra i principali metodi agili, due dei più diffusi e apprezzati sono Extreme Programming (XP) e SCRUM. Extreme Programming (XP) si distingue per il suo approccio rigoroso e metodico, che prevede l'esecuzione di tutti i test a ogni build del sistema. Una build viene accettata solo se tutti i test vengono superati con successo, garantendo così una qualità del software elevata e priva di errori critici. XP adotta però numerose altre pratiche di sviluppo, molte delle quali risultano particolarmente utili per il nostro team: in XP, ad esempio, i requisiti sono espressi come storie utente ed è il team di sviluppo che le suddivide in task da

implementare: tali task sono la base per la stima dei costi e la schedulazione del lavoro, basata sulla priorità dei task e sulla relazione tra essi: infatti ai requisiti utente è assegnata una priorità e i requisiti di priorità più alta sono inclusi in incrementi iniziali. Per visualizzare più facilmente come ci siamo strutturati e da cosa partiamo, facciamo riferimento al Product Backlog, che in Scrum viene usato per definire un elenco di voci, ordinato per priorità: all'inizio di ciascuno Sprint, che rappresenta un ciclo di sviluppo a durata fissa, selezioneremo dal Backlog un insieme di voci da sviluppare in quella iterazione (lo Sprint Goal).

Order	ID	Item	Type	Status	Estimate
1	01	Il sistema deve fornire all'amministratore la possibilità di recuperare le coverage delle varie classi risolte dai robot	Requirement	Not started	7
2	02	Il sistema deve fornire all'amministratore la possibilità di creare una scalata personalizzabile, con un ordine delle classi ad ogni round scelto da lui	Improvement	Not started	3
3	03	Il sistema deve variare il robot utilizzato, in modo da incrementare la difficoltà della scalata.	Improvement	Not started	5
4	04	Il sistema deve poter memorizzare al suo interno i robot contro cui l'utente ha giocato durante la scalata	Requirement	Not started	4
5	05	Il sistema deve permettere all'amministratore di personalizzare livello e robot di ciascuna classe all'interno della scalata che sta creando	Improvement	Not started	6
6	06	Il sistema deve poter fornire all'amministratore, all'atto della creazione di una scalata, oltre alla difficoltà delle classi, anche le coverage dei robot.	Improvement	Not started	7
7	07	All'atto della creazione, la scalata sarà formata da un ID, un nome, una descrizione, un numero di round, e dalle classi selezionate. Per ciascuna classe ne va specificato il robot e la difficoltà.	Improvement	Not started	2
8	08	Il sistema deve fornire all'utente la possibilità di scegliere tra più scalate inserite dall'amministratore con annessa informazioni sul tipo di scalata.	Improvement	Not started	1
9	09	Il sistema deve permettere all'amministratore di poter scegliere una scalata predefinita, scegliendo un unico tipo di robot e definendo le difficoltà associate	Requirement	Not started	4
10	010	Il sistema deve dare la possibilità all'utente di ritornare al menù scalata nel caso in cui l'utente perda una partita.	Overhead	Not started	1

1.7 USER STORIES

Una user story rappresenta una descrizione semplice e chiara di una funzionalità o di un requisito che è rilevante per un utente del sistema software, spiegando cosa vuole ottenere e perché è importante. Ci siamo immaginati delle storie utente rispetto alle funzionalità offerte dalla scalata.



1.7.1 criteri di accettazione delle user stories

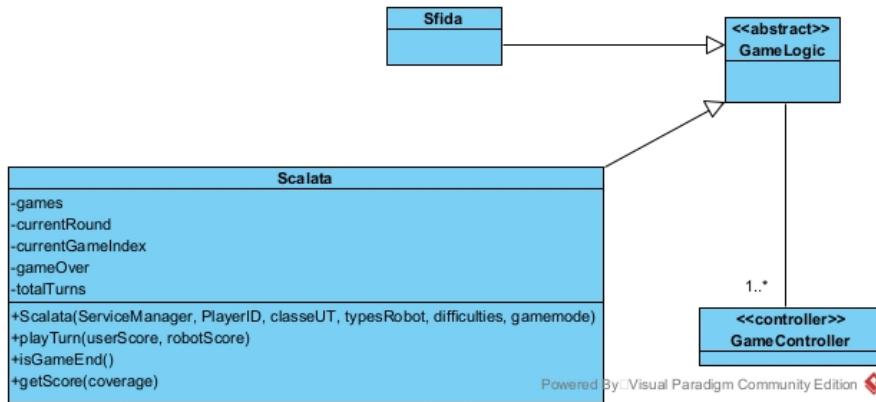
1. Da utente voglio poter giocare contro i robot finché non perdo
 - a. Supponendo che l'utente abbia effettuato l'accesso che nel sistema ci sia almeno una scalata che l'utente abbia cominciato una partita nella modalità scalata Quando l'utente perde un round nella scalata Allora assicurati che venga visualizzato un messaggio di sconfitta e assicurati che l'utente ritorna nel menù della modalità scalata

- b. Supponendo che l'utente abbia effettuato l'accesso
che nel sistema ci sia almeno una scalata
che l'utente abbia cominciato una partita nella modalità scalata
Quando un utente vince tutti i round nella scalata
Allora assicurati che venga mostrato un messaggio finale di vittoria
e assicurati che l'utente venga reindirizzato alla leaderboard
- 2. Da amministratore, vorrei creare una scalata personalizzabile con al suo interno classi con difficoltà differenti, in modo da avere una progressività
 - a. Supponendo che l'amministratore abbia effettuato l'accesso
che l'amministratore sia nel menù di creazione della scalata
che nel sistema siano caricate almeno due classi
Quando l'amministratore crea una scalata
Allora assicurati che possa scegliere di poter rendere la scalata personalizzabile
e assicurati che sia possibile scegliere sia il robot che la difficoltà del robot per ciascuna classe.
- 3. Da utente voglio poter sfidare robot diversi nei vari round
 - a. Supponendo che l'utente abbia effettuato l'accesso
che l'utente sia nel menù della modalità scalata
che ci sia almeno una scalata personalizzabile nel sistema
Quando l'utente seleziona una scalata personalizzabile
Allora assicurati che l'utente possa sfidare sia il robot Evosuite sia il robot Randoop
in diversi round della scalata
e assicurati che la difficoltà e il robot in un round della scalata personalizzabile coincidano con quelli della scalata all'atto di creazione
- 4. Da utente esperto, voglio misurarmi su più classi da testare consecutivamente alla massima difficoltà
 - a. Supponendo che l'utente abbia effettuato l'accesso
che l'utente sia nel menù della modalità scalata
che nel sistema ci sia almeno una scalata personalizzabile che contenga le classi a massima difficoltà
Quando l'utente sceglie una scalata
Allora assicurati che all'utente vengano correttamente mostrate, durante la partita, il robot e la difficoltà della classe scelta

2 - ANALISI DEI REQUISITI

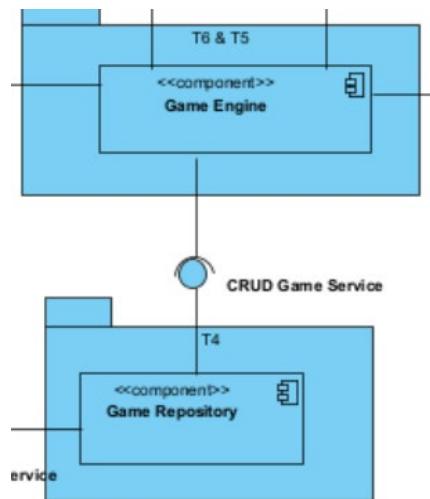
2.1 DIAGRAMMI UML SCALATA

Dalla documentazione fornita notiamo che la modalità scalata non è stata descritta dai diagrammi delle classi che dovrebbero trattarla, ovvero nel package Game. Procediamo quindi a costruire un'entità Scalata, che, come l'entità Sfida, dipenderà dalla classe GameLogic definita all'interno di T5 e la implementerà.



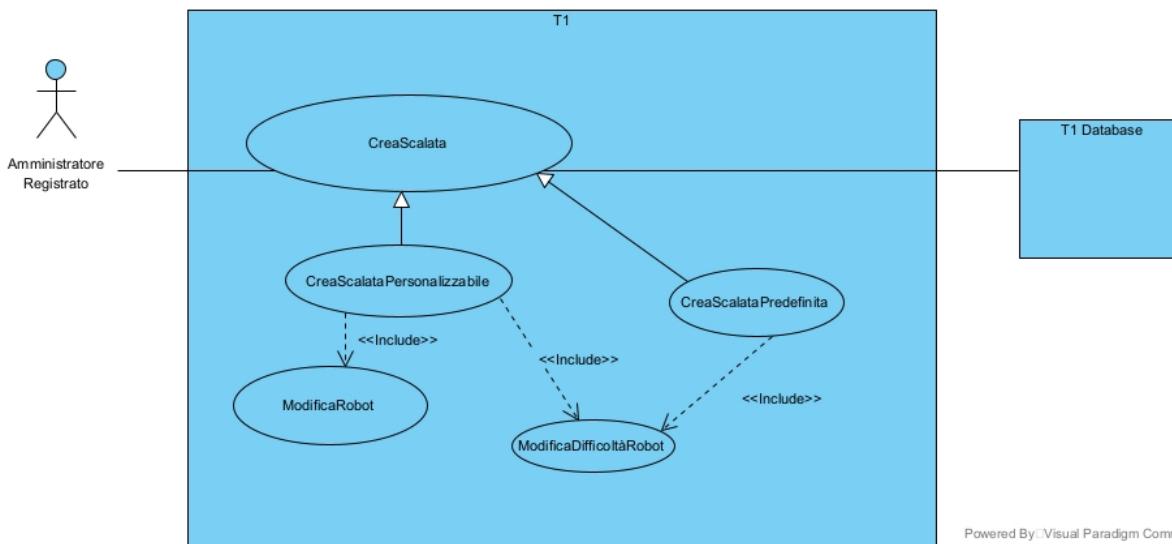
Primo Class Diagram che rappresenta lo stato iniziale della scalata all'interno del T5

Per capire meglio cosa dovrà essere modificato nel codice, costruiamo i diagrammi dei casi d'uso per ogni task che andiamo a toccare (T1, T4, T56), così da farci un'idea delle interazioni tra attori e sistema e delle interazioni tra le azioni stesse del sistema. Nello specifico le interazioni tra T4 e T56 saranno date dal seguente schema.

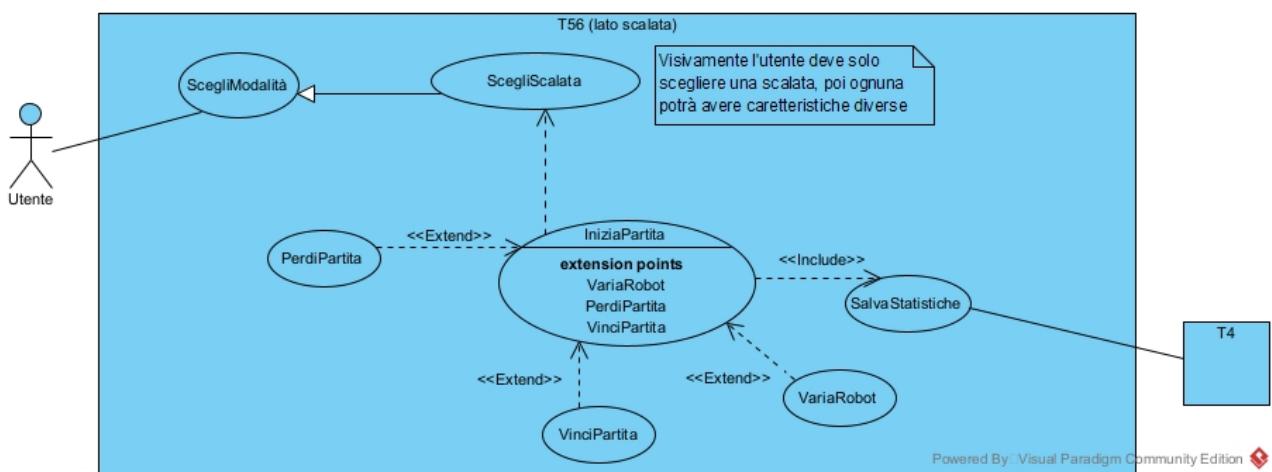


Interfacciamento tra T5 e T6 rappresentato tramite component Diagram

Analizziamo le nostre modifiche effettuate per la modalità scalata in T1 e T56. Per modellare le interazioni tra utenti e sistema, utilizzeremo i Diagrammi dei Casi d'Uso. Essi ci consentono di trasformare le descrizioni informali dei requisiti funzionali in una notazione più formale e strutturata, facilitando la comprensione delle esigenze degli utenti e delle interazioni con il sistema. Grazie ai Diagrammi dei Casi d'Uso, saremo in grado di identificare le azioni che gli utenti possono eseguire, le risposte che il sistema fornisce e i diversi scenari d'uso, sia positivi che negativi.



Caso d'uso dello scenario di creazione di una scalata



Caso d'uso dell'avvio di una partita

2.2 SCENARI SCALATA

Riportiamo gli scenari dei casi d'uso.

2.2.1 CreaScalata

CreaScalata	
Attore primario	Amministratore Registrato
Attore secondario	-
Descrizione	Per poter far giocare l'utente alla scalata, l'amministratore deve caricare le classi che insieme formano la scalata.
Pre-Condizioni	Login come Amministratore Classi devono esser state caricate
Sequenza di eventi principale	1. Il caso d'uso inizia quando l'amministratore vuole creare una scalata

	<ol style="list-style-type: none"> 2. L'amministratore deve fornire nome, descrizione e numero di rounds 3. L'amministratore deve scegliere la tipologia di scalata (casi d'uso correlati) 4. L'amministratore deve selezionare un numero di classi pari al numero di round scelto, tra le classi caricate in precedenza 5. Dopo la creazione della scalata, verrà memorizzata all'interno del database (T4)
Post-Condizioni	Gli utenti troveranno varie tipologie di scalata tra cui scegliere.
Casi d'uso correlati	CreaScalataPredefinita (figlio); CreaScalataPersonalizzabile (figlio);
Sequenza di eventi alternativi	<ol style="list-style-type: none"> 1. Se l'amministratore vuole scegliere di usare lo stesso robot per le classi caricate: Punto di specializzazione: CreaScalataPredefinita. Il sistema cambia automaticamente il robot a seconda del tipo di CreaScalataPredefinita selezionato. Il sistema poi procede con la selezione delle difficoltà (caso d'uso d'inclusione ModificaDifficoltàRobot). 2. Se l'amministratore vuole caricare classi con robot diversi: Punto di specializzazione: CreaScalataPersonalizzabile. Il sistema chiede quale robot e con che difficoltà (casi d'uso d'inclusione ModificaRobot e ModificaDifficoltàRobot).

2.2.2 IniziaPartita

IniziaPartita	
Attore primario	Utente
Attore secondario	-
Descrizione	L'utente deve poter giocare alla scalata scelta
Pre-Condizioni	Login come Utente L'utente deve aver scelto una scalata tra quelle presenti
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente parte dal primo round con la prima classe presente nella scalata 2. Per ogni round, l'utente, una volta scritti i test, può confrontare i risultati dei suoi test con quelli del robot (punti di estensione: VariaRound, VinciPartita, PerdiPartita) 3. Include (SalvaStatistiche): Al termine della partita, verranno salvate le statistiche all'interno del database (T4)
Post-Condizioni	Gli utenti avranno vinto o perso.
Casi d'uso correlati	VariaRound (estensione); PerdiPartita (estensione); VinciPartita (estensione); SalvaStatistiche (inclusione)
Sequenza di eventi alternativi	<ol style="list-style-type: none"> 1. Alla fine di un round superato, inizierà un nuovo round; Punto di estensione: VariaRound 2. La partita termina con la fine dell'ultimo round. Punto di estensione: VinciPartita 3. La partita termina se si ottiene una percentuale minore del robot in un round; Punto di estensione: PerdiPartita

2.2.3 Scegli Scalata

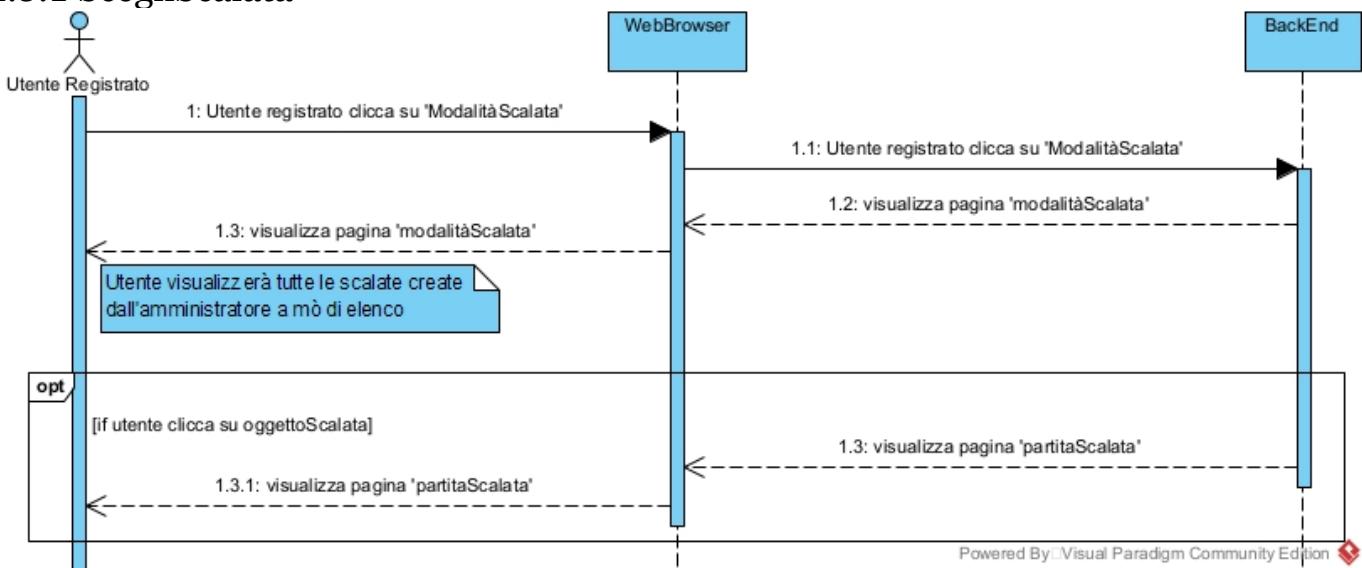
Scegli	
Attore primario	Utente
Attore secondario	-
Descrizione	L'utente può scegliere a quale scalata giocare tra quelle già create nel sistema.
Pre-Condizioni	Login come Utente L'utente si trova nel menù Scalata Classi devono esser state caricate
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il caso d'uso inizia quando l'utente vuole scegliere una scalata 2. L'utente può scegliere di giocare ad una scalata personalizzata oppure ad una predefinita
Post-Condizioni	Gli utenti avvieranno la partita sulla scalata scelta.
Casi d'uso correlati	-
Sequenza di eventi alternativi	-

2.3 SEQUENCE DIAGRAM SCALATA ALTO LIVELLO

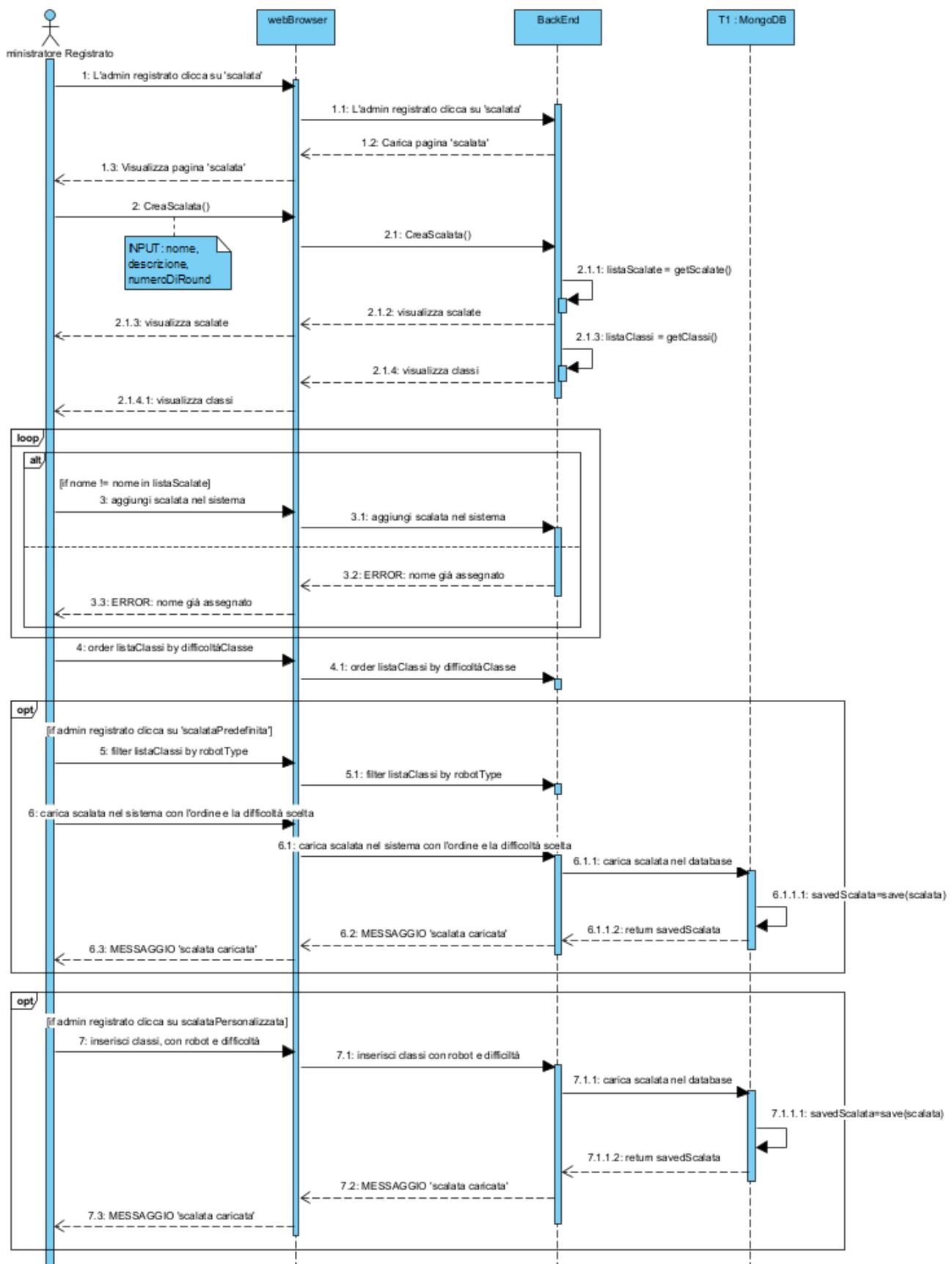
Per comprendere appieno le dinamiche di interazione tra i componenti del nostro sistema, ci avvarremo dei sequence diagram.

Questi primi diagrammi ci consentono di avere un'idea di massima sulle iterazioni che avvengono tra le varie componenti e non rappresentano ancora in dettaglio quali funzioni vengono invocate da una parte e dall'altra.

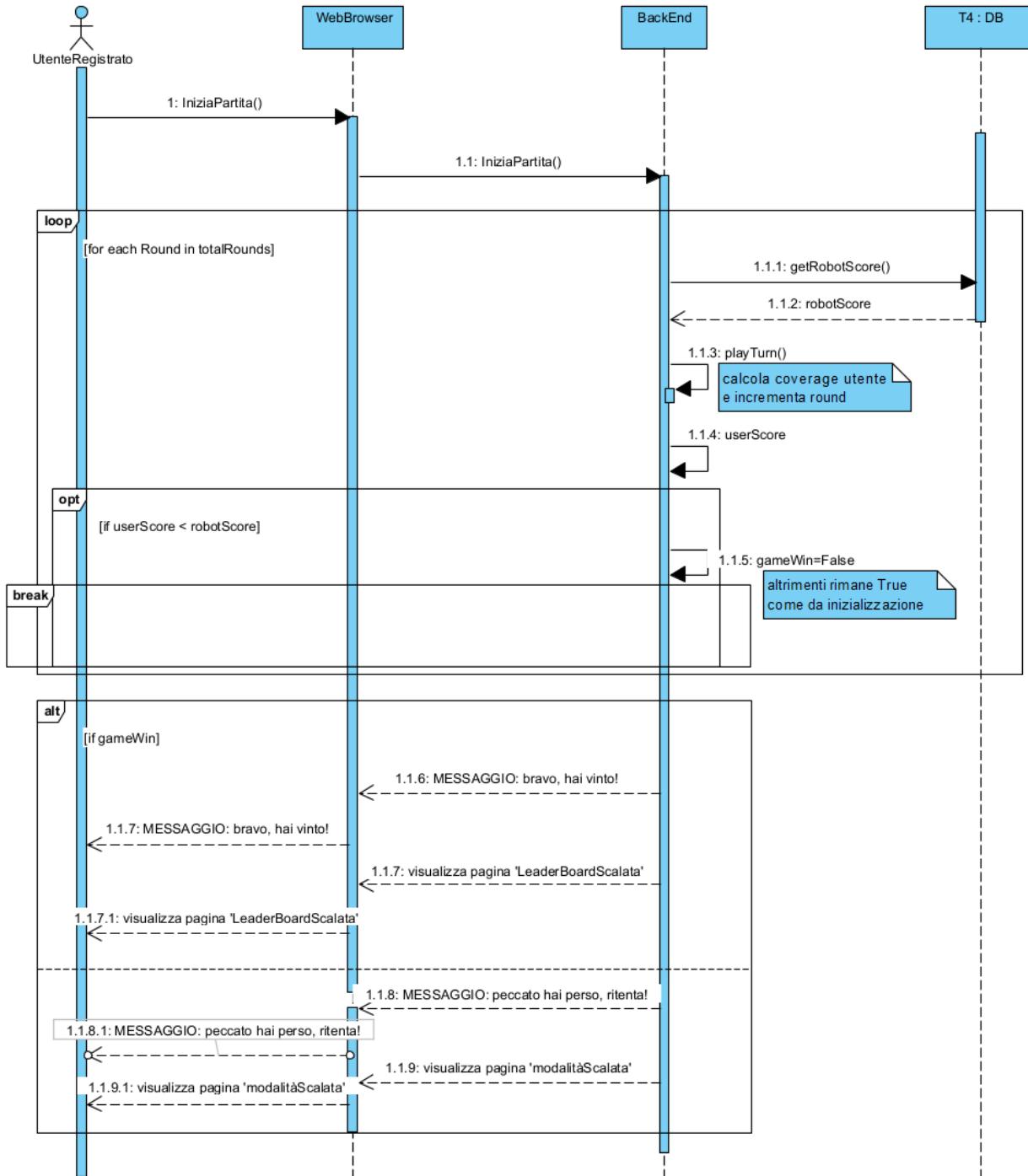
2.3.1 ScegliScalata



2.3.2 CreaScalata

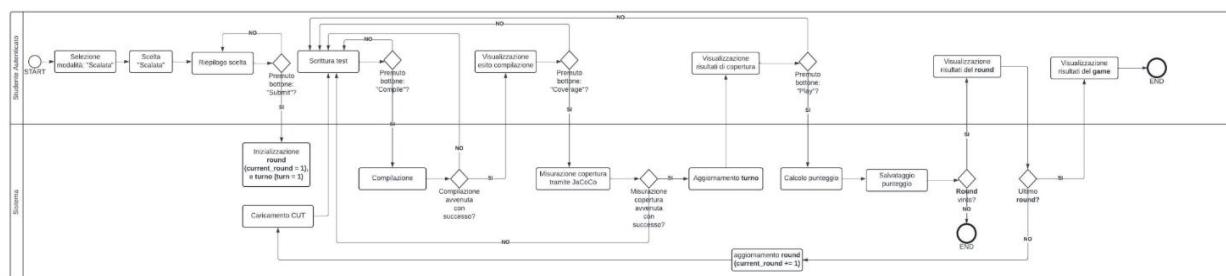


2.3.3 IniziaPartita



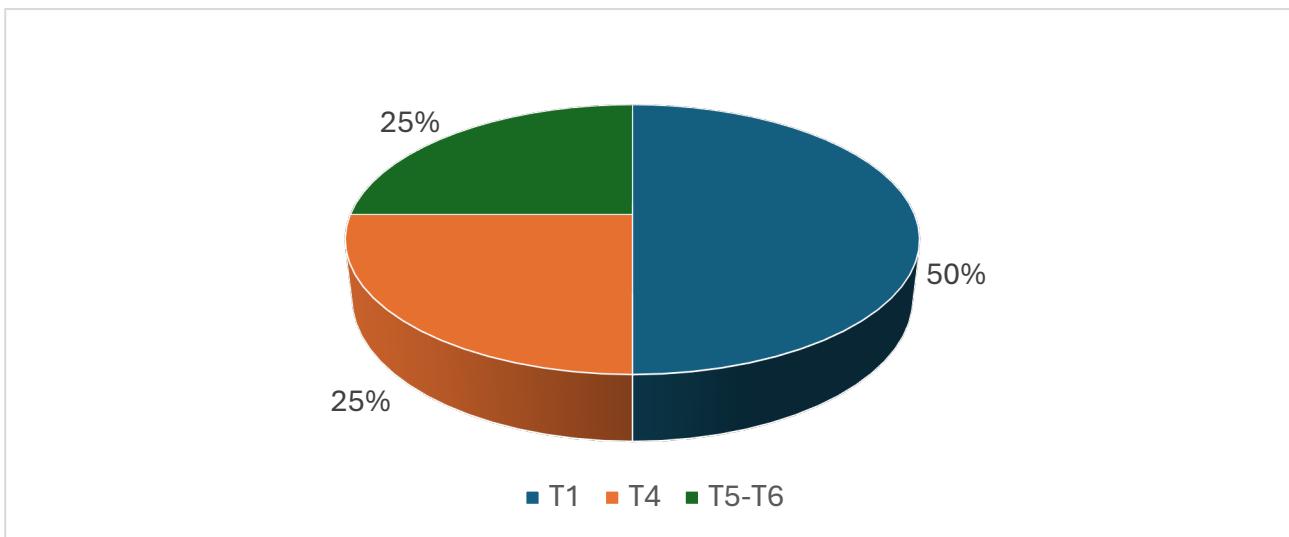
2.4 WORKFLOW DIAGRAM SCALATA

Di seguito, viene riportato un workflow diagram relativo all'avvio di una partita e il "gameplay loop" in modalità scalata, estratto da documentazioni precedenti. Dal punto di vista concettuale, non sono state apportate modifiche.

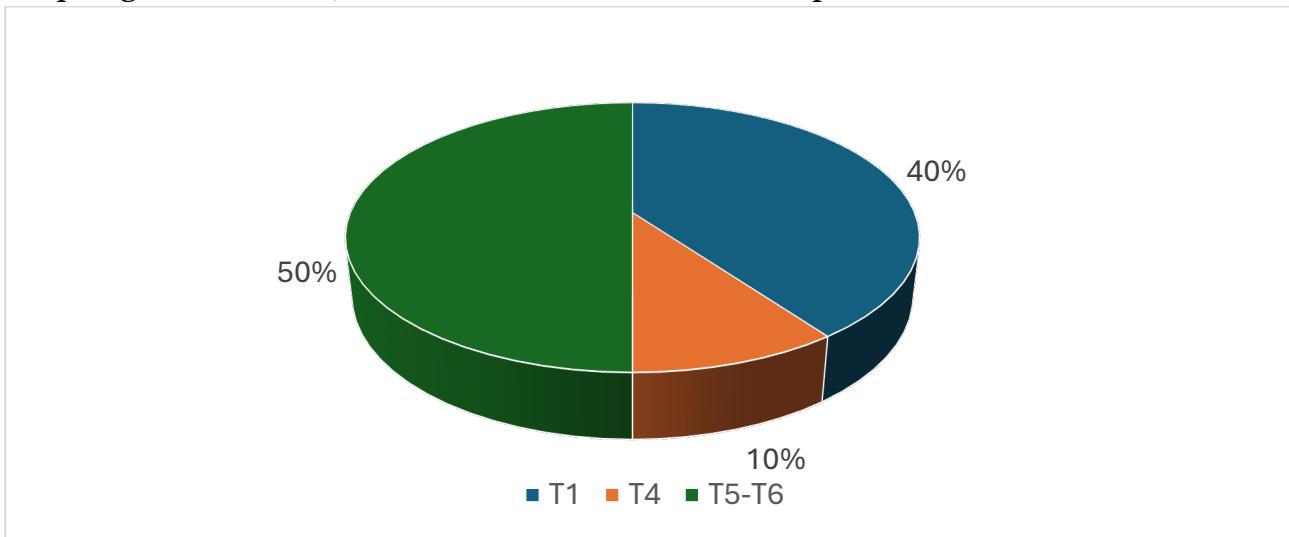


3 - ANALISI D'IMPATTO

L'Analisi d'Impatto è una fase fondamentale nella gestione delle modifiche a un sistema software o a un'architettura esistente. Il suo obiettivo è identificare quali componenti del sistema verranno influenzati da una modifica. Modifiche al **T1** incidono sulla creazione e personalizzazione delle scalate: lo scopo è di offrire all'amministratore un maggiore controllo sulla difficoltà e sulle coverage dei robot, permettendo così di bilanciare meglio le scalate: ciò non necessita di un elevato sforzo. Le modifiche **T4** devono introdurre solo la memorizzazione dei robot affrontati dall'utente, consentendo analisi più dettagliate sulle partite giocate e migliorando il tracciamento del progresso dell'utente. In **T5-T6**, invece, va migliorata l'esperienza utente, aumentando la varietà dei robot utilizzati nei vari round, ma vanno sistematiche anche le varie problematiche affrontate in precedenza, che richiedono quindi uno sforzo maggiore.



L'integrazione del nuovo editor ha modificato l'analisi d'impatto prevista, poiché ha coinvolto più componenti del sistema di quanto inizialmente stimato, come noteremo anche nei paragrafi successivi, richiedendo ulteriori modifiche specialmente del T5.



4 - PROGETTAZIONE DELLA SOLUZIONE

4.1 - DESCRIZIONE DECISIONI DI PROGETTO

4.1.1 - Panoramica

Il cuore della soluzione che vogliamo raggiungere può essere riassunto in tre punti

1. Permettere all'amministratore di scegliere robot e difficoltà di ogni singola classe nella scalata
2. Trasmettere informazioni complete su una scalata dal T1 al T5.
3. Salvare correttamente le partite giocate all'interno di T4, includendo informazioni sul robot e difficoltà.

Dopodiché, si è deciso di approcciare altre tematiche importanti per il funzionamento della nostra soluzione in maniera ottimale, cercando di risolvere problematiche critiche del sito riscontrate durante la fase del progetto, all'interno di aree di nostro interesse. Tra queste, la decisione di irrobustire il lato admin sulla creazione della scalata.

Infine, la soluzione si basa anche sul cambiamento dell'editor, allineando l'editor della modalità scalata con quello presente in modalità sfida. Questa transizione, inizialmente posta come scelta di refactoring, si è rivelata essere importante per il funzionamento del sistema nella sua interezza, siccome il salvataggio delle partite all'interno della modalità sfida veniva effettuato diversamente.

4.1.2 – Decisioni in T1

La prima modifica da fare all'interno di T1 è dare la possibilità di memorizzare, per ogni classe all'interno di una scalata, il robot e la difficoltà selezionati. Questo servirà poi per comunicare al front-end in T5 più in dettaglio i connotati della scalata.

La nostra soluzione propone di risolvere il problema in due punti principali:

1. Innanzitutto, si decide di dare all'amministratore la possibilità di visualizzare le coperture per ogni classe. Questo non solo migliora la personalizzazione della scelta, fornendo all'amministratore un feedback visivo immediato, ma evidenzia anche l'assenza di alcune coperture per determinate combinazioni di robot e difficoltà, per le quali non viene data la possibilità di selezione; in questo modo, si abbassa il rischio di errori in caso di coperture assenti. Tale scelta si può suddividere in tre ulteriori step:
 - a. Memorizzare le combinazioni di robot, difficoltà e copertura per ogni singola classe
 - b. Ottenere le informazioni di copertura all'interno del T1
 - c. Poder visualizzare tutte le informazioni, così come la scelta effettuata, all'interno della scheda sulla classe, presente nel menù di creazione della scalata
2. Memorizzare la scelta di classi, robot e difficoltà all'interno del repository della scalata.

È stato scelto di raccogliere i dati su robot, difficoltà e coperture all'interno di RobotUtils siccome è il primo metodo a gestire tali dati, prima di inviarli direttamente al T4 per il salvataggio al suo interno.

Da un punto di vista concettuale, ci è sembrato un overhead inutile chiedere informazioni sui robot al T4 per conservarle all'interno di T1, se queste informazioni vengono processate prima da quest'ultimo.

La visualizzazione di tali informazioni ha portato, infine, alla modifica delle impostazioni grafiche delle card, in modo tale da poter condensare al suo interno le informazioni; inoltre dare la possibilità all'amministratore di scegliere robot e difficoltà di ogni classe di test che forma la scalata limita i passaggi che deve fare l'amministratore ed evita problematiche che può portare un inserimento errato, a differenza di opzioni selezionabili già prestabilite.

All'atto del salvataggio della scalata, vanno salvate solo alcune generalità della classe selezionate, e non tutti i parametri caratterizzanti la stessa. In precedenza, la classe scelta veniva salvata come stringa. Questo, più in avanti, si è rivelato essere una limitazione per la nostra soluzione.

Dopodiché, si è scelto un approccio che da importanza all'esperienza dell'amministratore. I cambiamenti, quindi, devono rendere i passaggi per la creazione di una scalata, veloci ed intuitivi: inoltre, ogni decisione dà molta importanza anche all'aspetto di robustezza, spesso tenuto in considerazione anche nei paragrafi precedenti, tra cui un miglior controllo sui parametri d'ingresso della scalata e la possibilità di cancellare una scalata, nel caso in cui questa contiene una classe cancellata al suo interno.

4.1.3 – Decisioni in T5

All'interno del front-end in T5 del vecchio editor si è cercato solamente di adattarne il funzionamento ai nuovi tipi di dato che venivano passati da T1, siccome le chiamate API già in essere funzionavano a fronte delle modifiche in T1.

Per una fase iniziale si è sfruttato quindi tutto ciò che funzionava del vecchio codice, adattandolo alle nostre esigenze.

Per i parametri aggiuntivi di robot e difficoltà di ogni classe, si è scelti di memorizzarli all'interno del browser, per un rapido recupero di tutti i dettagli necessari per la sessione di gioco.

Dopo aver testato che il comportamento dell'editor, a fronte delle nuove modifiche, si allineasse a quello atteso, si è deciso di abbandonarlo, siccome si è stimato che le modifiche grafiche richiedevano tempo per la totale comprensione, così come la risoluzione di problemi di tipo architettonico. Di seguito le problematiche riscontrate:

- I pulsanti presenti all'interfaccia presentavano vari problemi relativi al feedback: il pulsante “play” dava un alert di errore, dopo qualche istante però avveniva con successo la compilazione. Il pulsante submit poteva essere cliccato più volte, e per ogni volta che veniva cliccato, esso dava un punteggio diverso (questo verrà spiegato più in avanti nel sequence diagram)
- Ridondanza tra il pulsante play e il pulsante compile: entrambi svolgevano la stessa azione.
- Alla pressione del pulsante submit, appariva un primo alert in cui diceva l'esito della partita, prima ancora che terminasse l'elaborazione del report dei risultati e la stampa in console.
- Al suo interno, il vecchio editor possedeva chiamate dirette verso il T4 per aggiornare il database, relativamente allo stato della scalata: questo viola il pattern MVC, in quanto è la view ad aggiornare il model, e non il controller.
- Venivano effettuate chiamate API che si interfacciavano con classi e metodi in disuso. Un esempio è il salvataggio delle partite all'interno del T4: veniva effettuata una chiamata a /save-data all'interno del GuiEditor in T5, la quale invocava un metodo all'interno di GameDataWriter. Questo metodo, però, è stato etichettato come legacy.

Altro esempio è la migrazione della chiamata /run da T6 a T5 per quanto riguarda le sfide normali, utilizzando differenti parametri in ricezione per la chiamata. Questi esempi verranno discussi più approfonditamente durante la discussione dei sequence diagram.

Oltre alle migliorie grafiche, il nuovo editor aveva un insieme di funzionalità interessanti, come la possibilità di recuperare una partita già in corso, che può risultare un'ottima scelta per una modalità di gioco dispendiosa dal punto di vista del tempo.

Il passaggio da un editor all'altro presentava tre problemi principali:

1. Non permetteva il salvataggio dei dati relativi alla scalata in T4, così come i game, round e turni al suo interno; cosa che con le vecchie chiamate API all'interno del T5 e del T6 avveniva.
2. Il nuovo editor è stato incentrato su derivazioni della sfida, e la relazione tra una sfida e una scalata è di tipo aggregativa.
3. La presenza pregressa di codice relativo alla scalata non implicava direttamente il suo funzionamento per i nostri scopi. Andava compreso cosa aggiungere e cosa rimuovere alla logica precedentemente sviluppata.

Front-End editor nuovo T5

Il front-end del nuovo editor non solo dovrà essere dotato, come il vecchio editor, della possibilità di poter memorizzare i dati della scalata in locale, ma anche di saper discernere quali dati prelevare a seconda della modalità di gioco, siccome la stessa piattaforma verrà utilizzata anche da altre modalità.

Infatti, la scalata necessita di una gestione a sé sia dal punto di vista dell'avvio di un nuovo round, sia nel passaggio da un round all'altro. Il nostro intervento ha suddiviso queste logiche in tre punti differenti

- Recupero dei dati all'inizio della partita, all'atto della creazione del documento
- Gestione del round, al termine della sua fine. Qui viene deciso se la scalata è conclusa, oppure si passa al round successivo
- Prevedere alla fine di ogni round della scalata un'interazione per scenario, in caso di vittoria, di passaggio del turno e di sconfitta.

Tutto questo, precedentemente, veniva effettuato all'interno di un singolo metodo che non veniva mai invocato. Questa strada è stata adottata per metterci in linea con la leggibilità e l'atomicità di ogni singola nuova funzione scritta all'interno del nuovo editor, evitando quanto più possibile l'aggregamento di logiche complesse in più parti.

L'aspetto di memorizzazione nel browser locale è rimasto nonostante il salvataggio della sessione di gioco in T5 sviluppato più tardi, per una questione di integrità e coerenza dei dati in più parti del codice.

T5 GameController-ScalataGame

Dal punto di vista progettuale, si è sfruttato il polimorfismo di GameLogic, con l'obiettivo di riutilizzare quanto più codice possibile, senza alterare l'esecuzione delle altre modalità.

Tuttavia, l'estensione di GameLogic esistente relativa alla scalata, ovvero scalataGame si è rivelata essere carente e poco coerente con l'esecuzione effettiva di una scalata. Serve una refattorizzazione che permetta in particolare di migliorare la riscrittura di metodi come PlayTurn e getScore ed adattarle in base al contesto richiesto.

Il problema principale relativo all'uso di GameLogic però si presenta nella gestione della partita, più specificatamente non è stata prevista, dal punto di vista strutturale, una modalità

di gioco che, al termine del round, ne venga caricato uno nuovo. Si è trovati una soluzione in due punti

1. Dotare la partita scalata in T5 di uno stato, a seconda della quale terminare o aggiornare la partita in corso
2. In caso di passaggio al round successivo, il gameController deve distinguere tra partita ripresa successivamente oppure quando viene caricato un round nuovo

Infine la comunicazione tra front-end e back-end necessitava di supportare il passaggio e l'eventuale manipolazione dei dati relativi alla scalata, tra cui l'aggiunta degli appositi parametri nella richiesta POST di /StartGame, e di dotare della risposta sia di StartGame sia di Run dei parametri relativi alla scalata.

Memorizzazione in T4

Per concludere, tutti i dati relativi al T4 devono essere inviati tramite delle specifiche chiamate API implementate tramite il serviceManager.

Le chiamate verso il T4 sono state aggiunte all'interno della gestione dello stato della scalata, dando la responsabilità alla gestione interna della stessa di aggiornare il database sul suo stato. Alcuni di questi metodi per contattare il T4 sono stati aggiunti, e quelli presenti sono stati modificati per integrare al meglio le scelte di progetto effettuate.

Sono poi stati aggiunti tutti i service relativi al T4, che consentono di migliorare la gestione e il monitoraggio della scalata, assicurando che ogni fase del gioco sia registrata in maniera coerente e affidabile. La possibilità di creare, aggiornare e chiudere una scalata in modo strutturato garantisce una maggiore tracciabilità del processo, permettendo di analizzare le partite a posteriori e di identificare eventuali anomalie o criticità. L'aggiornamento costante della scalata facilita inoltre l'integrazione con altri sistemi, migliorando la sincronizzazione tra le diverse componenti del gioco e rendendo il sistema più robusto.

4.2 - MODIFICHE AL DATABASE (T4)

In T4, per implementare la relazione tra robot e round che comporta il salvataggio del robot contro cui l'utente ha giocato nello svolgersi di ogni round, è stato necessario rivisitare la struttura del round in round.go, che rappresenta proprio come viene salvata l'entità nel database.

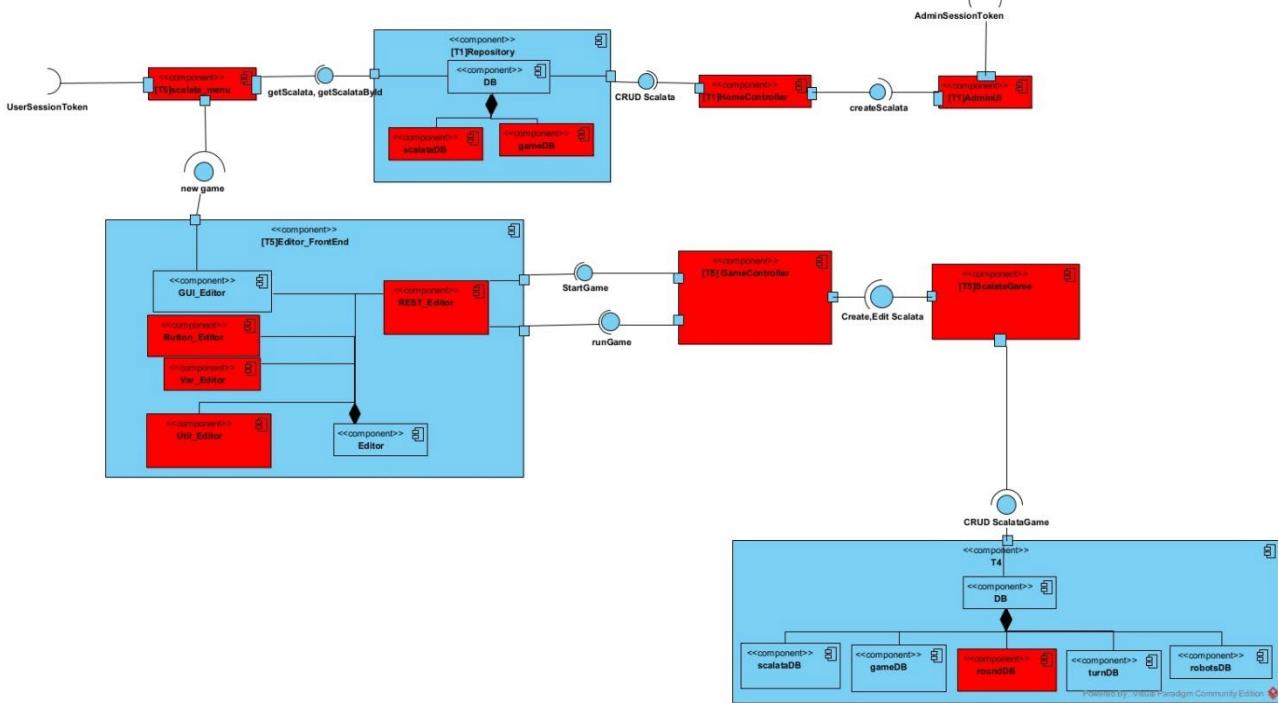
Inoltre, salvando il robot contro cui si gioca ad ogni round, servirà recuperare l'id di tale robot per utilizzare la nuova associazione tra round e robot: ciò può essere fatto direttamente in GameLogic, perché definisce attributi comuni sia a partita sia a scalata.

Nello specifico abbiamo aggiunto due attributi: RobotID, un intero, e Robot, che contiene la struttura completa di quel robot. Nello specifico, RobotID rappresenta l'identificatore univoco del robot associato al round e funge da chiave esterna, utilizzata per collegare il round con un record specifico nella tabella dei robot. Robot rappresenta l'intero oggetto robot associato al round, con tutti i suoi dettagli. Per il nostro semplice requisito quest'ultimo non sarebbe necessario, ma potrebbe essere comunque utile fornire la possibilità di accedere ai dati completi del robot: l'oggetto Robot fornisce tutte le informazioni, piuttosto che dover cercare il robot ogni volta tramite l'ID (con una query al database o una chiamata API).

4.3 - DIAGRAMMI DI DETTAGLIO STATICI

4.3.1 Component Diagram

Partendo dal [component diagram di partenza illustrato a pagina 6](#), si è quindi toccati il T1, il T4 e il T5. viene di seguito proposto una versione zoomata delle componenti relativa alla sola scalata, dove in rosso sono state evidenziate quelle modificate per l'adempimento dei nostri scopi.



A partire in alto a destra, è presente un punto d'accesso per l'admin, il quale interagisce con le componenti all'interno di T1 per la creazione della scalata, e la conservazione all'interno della stessa all'interno del repository dello stesso microservizio.

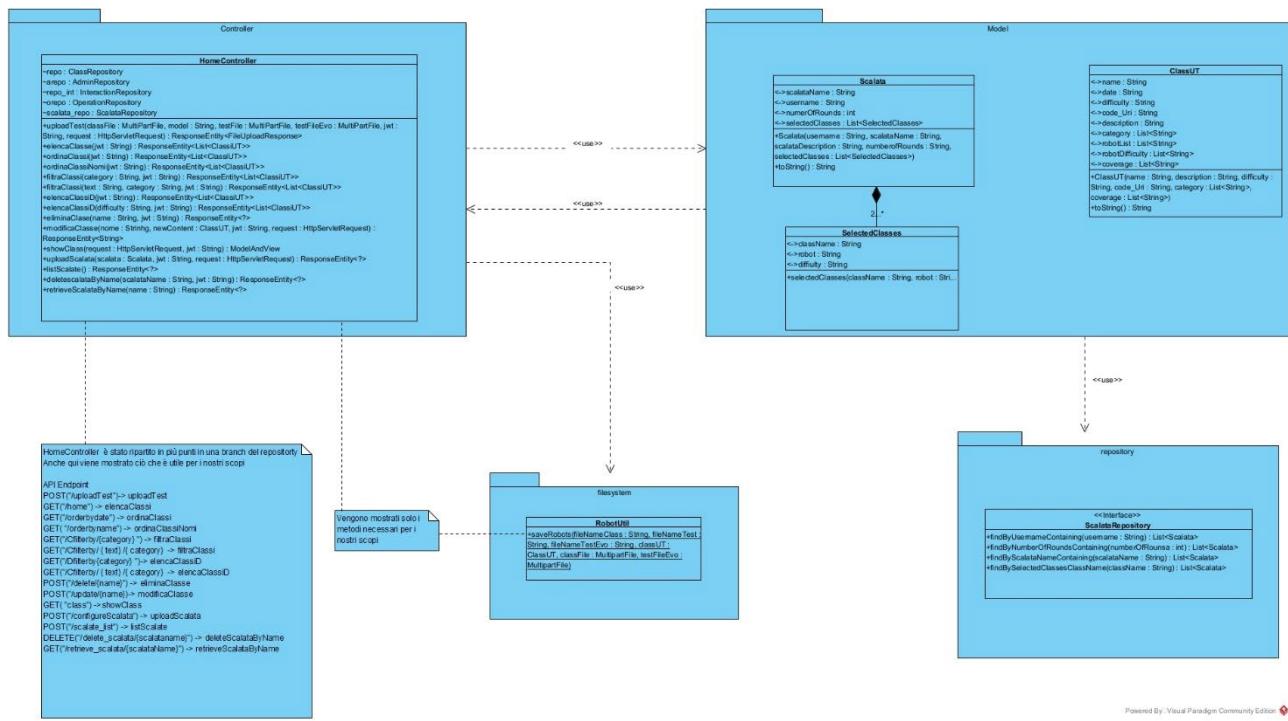
Il punto d'ingresso dell'utente registrato è situato in Scalata_menu, situato in T5, per avviare una nuova partita; qui avviene l'interazione con il repository in T1 dapprima per raccogliere tutti le scalate presenti nel sistema, dopodiché, non appena si avvia una partita, vengono prelevati i dettagli della scalata selezionata sempre in T1 e passati al GameController, che gestisce la logica principale di gioco. Al suo interno viene creato un'istanza di ScalataGame, che rappresenta la logica della partita in corso, che a sua volta contiene un ServiceManager, responsabile delle operazioni CRUD e dell'interazione con il database. Durante una partita, è compito del GameController orchestrare a più ampio livello le azioni da effettuare, a seconda delle azioni ricevute dall'editor e dello stato comunicato da parte di ScalataGame. L'editor (Editor_FrontEnd) permette poi al giocatore di giocare una partita. Al suo interno ci sono varie componenti che ne permettono il suo funzionamento: un modulo per la logica dei buttoni, uno per le chiamate REST da effettuare all'interno di T5, uno dove sono conservate le variabili di gioco e infine un modulo contenente funzioni di utilità per la gestione delle varie fasi di gioco.

4.3.2: Package Diagram T1

Per la visione in alta risoluzione si consiglia di cliccare su ... e poi su "view original"

Di seguito un package diagram che coinvolge tutte le unità toccate e le modifiche effettuate alle classi

Si fa presente che, per quanto riguarda l'homeController, non sono stati riportati tutti i metodi ma solo quelli d'interesse per la modalità scalata, e fanno riferimento al commit 604d2b6 del repository.



Le modifiche principali sul T1 sono state effettuate lato front-end, tuttavia il back-end, così come il model, doveva essere riadattato al fine di poter supportare il nuovo modo di memorizzare i parametri della scalata. In particolare, dell'HomeController sono stati modificati i metodi relativi alla chiamata API di uploadTest e updateClass.

4.3.3 Package Diagram T5

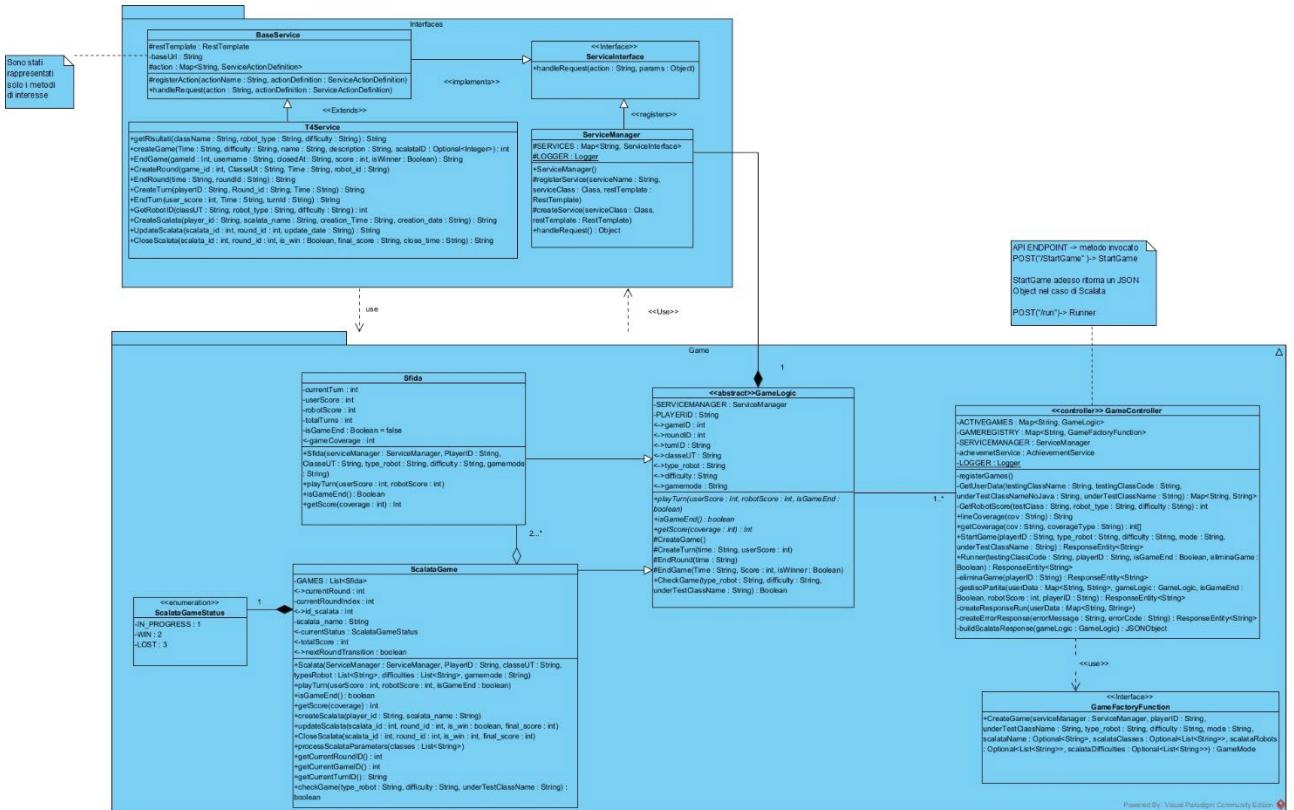
Per la visione in alta risoluzione si consiglia di cliccare su ... e poi su "view original"

Di seguito, una vista generale sul package diagram in T5. Sono stati evidenziati solamente i metodi utilizzati a più diretto contatto e quelli modificati

Il package diagram si suddivide in due distinzioni principali:

1) Interfaces in cui ServiceManager funge da mediatore principale per la gestione delle azioni. Contiene: un registro di servizi (RESERVES), funzioni per l'invocazione di servizi, e invoca specifiche azioni API.

2) Game, che si interfaccia tramite GameLogic, che viene esteso poi alle due modalità principali del package: Sfida e ScalataGame. Infine, gameController gestisce le richieste REST e funge da punto d'ingresso per il sistema.



4.4 - DIAGRAMMI DI DETTAGLIO DINAMICI

Si è scelto di rappresentare tramite sequence diagram l'interazione tra le varie componenti all'interno del sistema. Questo è stato fondamentale per comprendere in quale file è stato necessario apportare delle modifiche. Il livello di dettaglio, per quanto possa essere elevato, non comprende appieno tutti gli scenari relativi alle chiamate senza successo. Notare che il nome di alcuni metodi può non corrispondere del tutto ai metodi effettivamente chiamati, ma cercano di descriverne il loro comportamento al meglio delle possibilità.

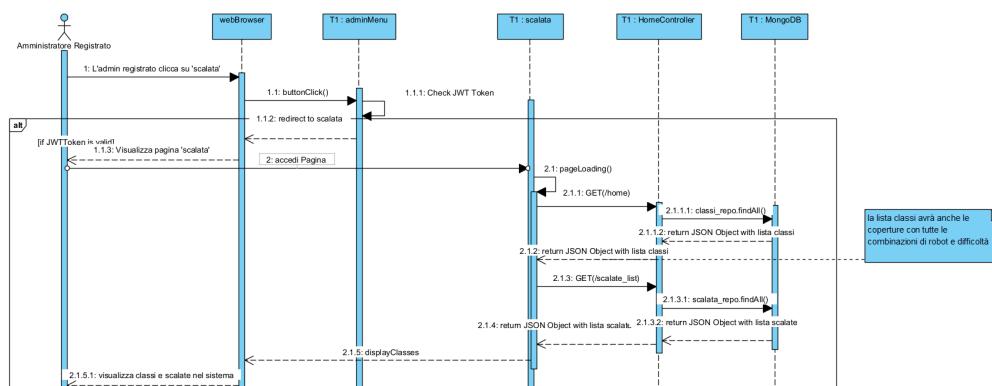
4.4.1: Sequence Diagram di dettaglio – CreaScalata

Per la visione in alta risoluzione si consiglia di cliccare su ... e poi su "view original"

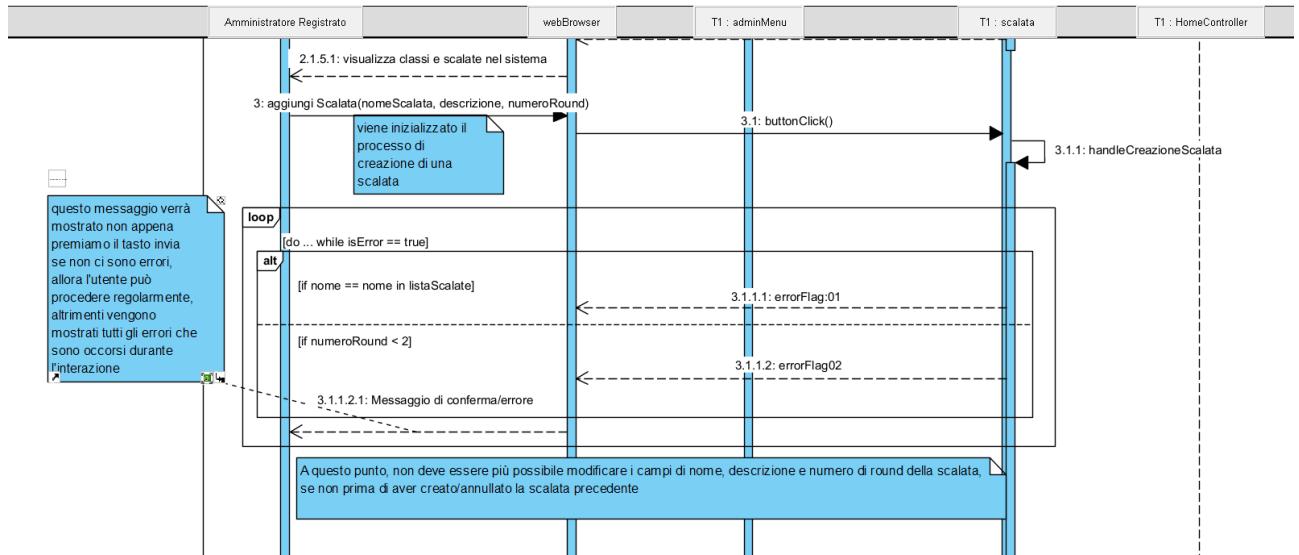
Il primo diagramma che si rappresenta è relativo alla creazione della scalata. Notare che in una versione attuale, si può ordinare le classi per difficoltà e filtrare il tipo di scalata tra personalizzata, solo Randoop oppure solo Evosuite.

Vengono evidenziati, di seguito, le interazioni chiave:

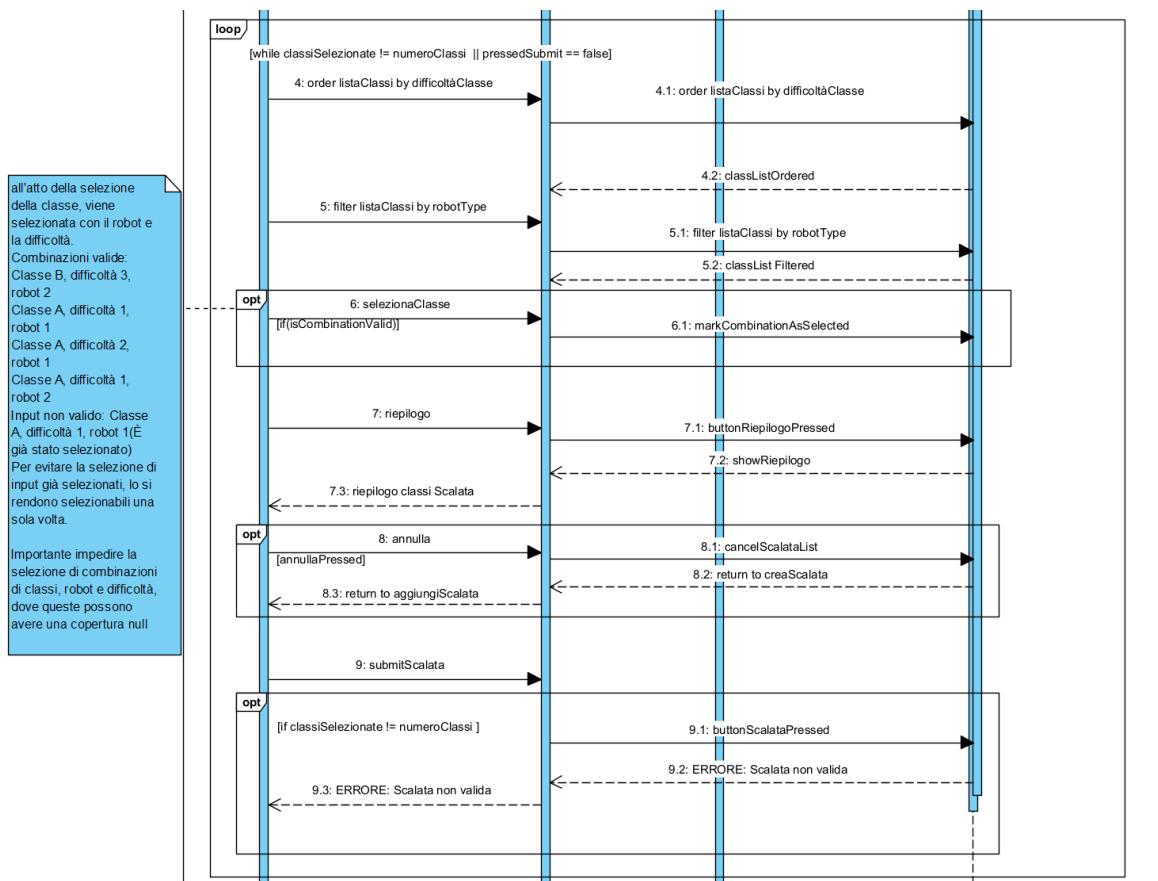
- Accesso alla pagina



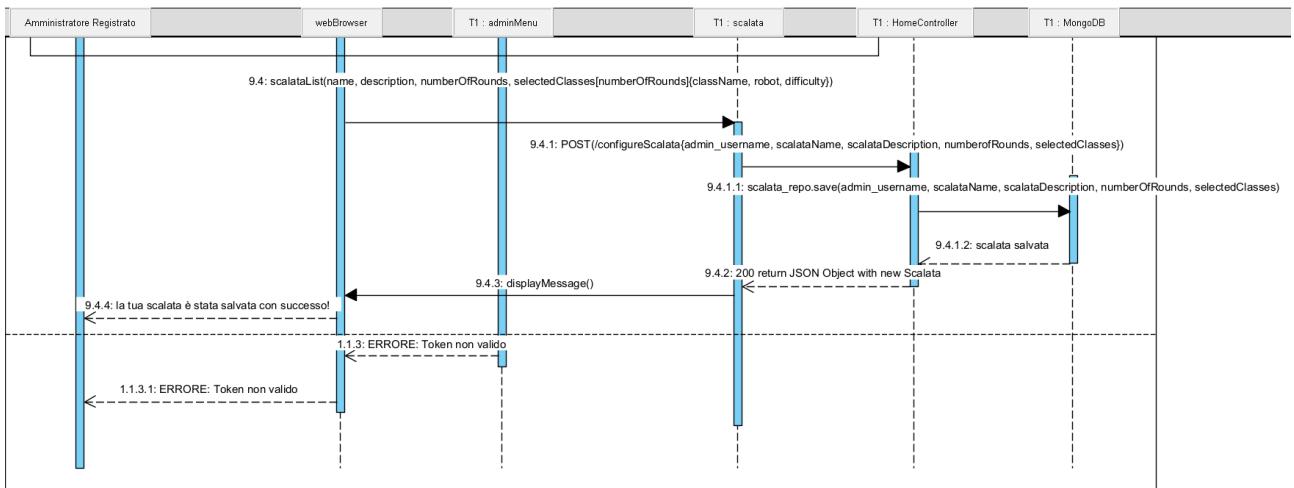
- Avvio processo di creazione della scalata



- Scelta classi per la scalata



- Salvataggio scalata – Errore Token JWT non valido



4.4.2: Sequence Diagram di dettaglio – Scegli Scalata(vecchia versione)

Per la visione in alta risoluzione si consiglia di cliccare su ... e poi su "view original"

La documentazione iniziale fornita ci sulla scalata conteneva un'overview generale sulle azioni che venivano svolte: tuttavia il livello di dettaglio non era sufficiente per comprendere in quali file era necessario apportare delle modifiche; in particolare, era necessario comprendere

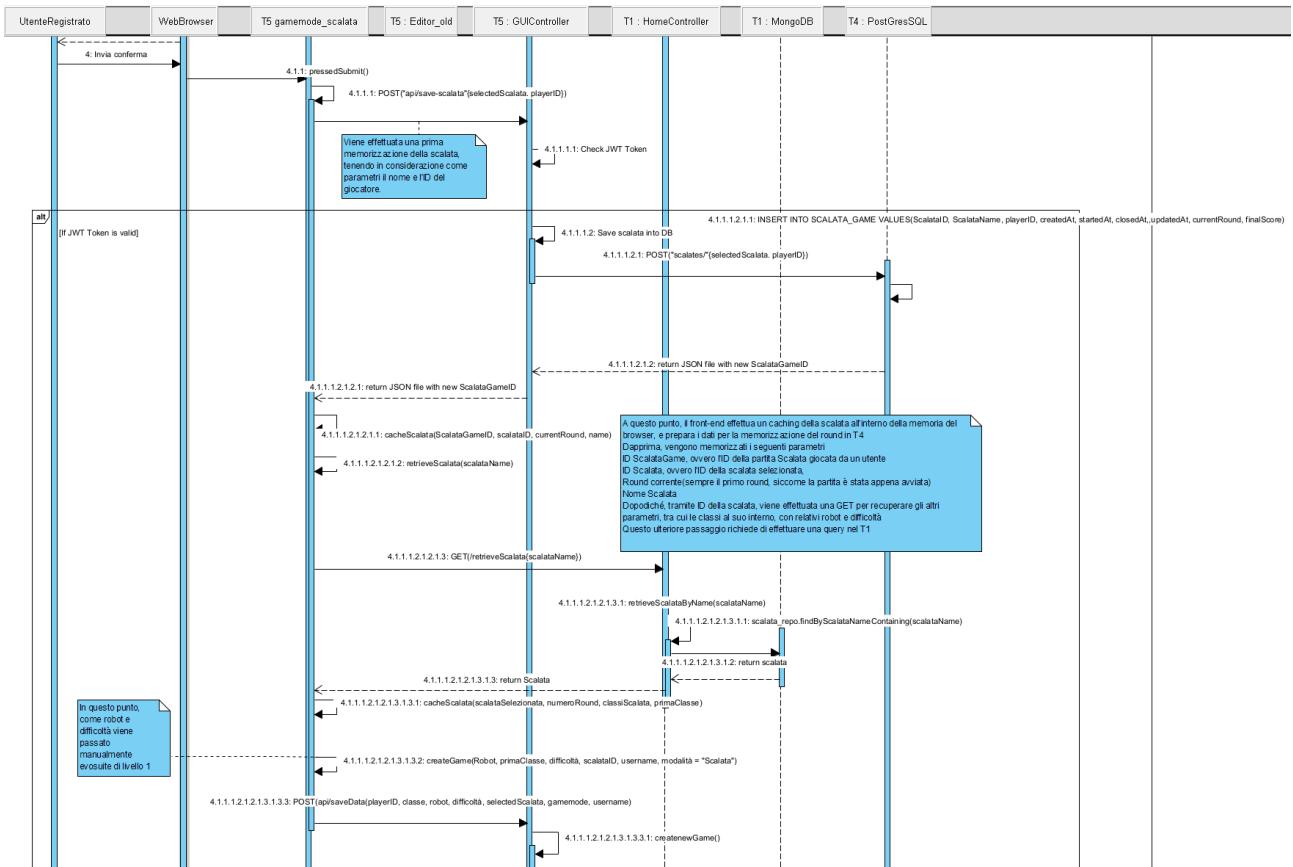
- Le chiamate API che venivano effettuate
- Il passaggio dei parametri da una parte all'altra (tra un microservizio e l'altro, oppure tra il front-end e il back-end)
- Quali file, classi e metodi andare a guardare

La presenza di due editor, così come la presenza di metodi relativi alla scalata in entrambi, ci ha portati a chiedere semplicemente quali metodi venissero invocati o meno.

Ciò ha portato alla creazione di questi sequence diagram a partire dal codice presente. Viene dedicata questa sezione, e una sezione successiva, relativa a questo processo di reverse-engineering, in particolare ai frutti che tale processo ha portato.

Il diagramma ottenuto da tale processo non è in alcun modo da prendere come riferimento assoluto per le interazioni tra le parti, siccome sono stati modificati col passare del tempo, ma è stato una mappa utile per capire dove intervenire, e soprattutto per dare una comprensione generale su quali pezzi di codice focalizzare la nostra attenzione.

In particolare, si fa presente questa parte di sequence Diagram



Questa sequenza di azioni è molto importante, siccome da qui avviene il passaggio cablato di evosuite al livello 1, una delle problematiche del nostro progetto: all'atto dell'invio della conferma di una determinata scalata, viene eseguita una funzione pressedSubmit che, dopo aver creato una partita Scalata tramite chiamata API, genera la partita Scalata all'interno di T4, ritornando un'ID.

- Viene poi usato il nome della scalata per ricavare le sue informazioni
- Vengono ritornate dal T1 queste informazioni
- Vengono memorizzate all'interno della cache del browser
- Viene creata una partita, facendo poi partire le chiamate verso la creazione in DB di game, round e turni.

Ciò ha evidenziato due delle azioni da intraprendere per quanto riguarda il nostro gruppo

1. Fare in modo che la scalata in T1 conservi al suo interno Classe, Robot e Difficoltà
2. Fare in modo che questi parametri vengano trasferiti correttamente all'atto della creazione di una partita.

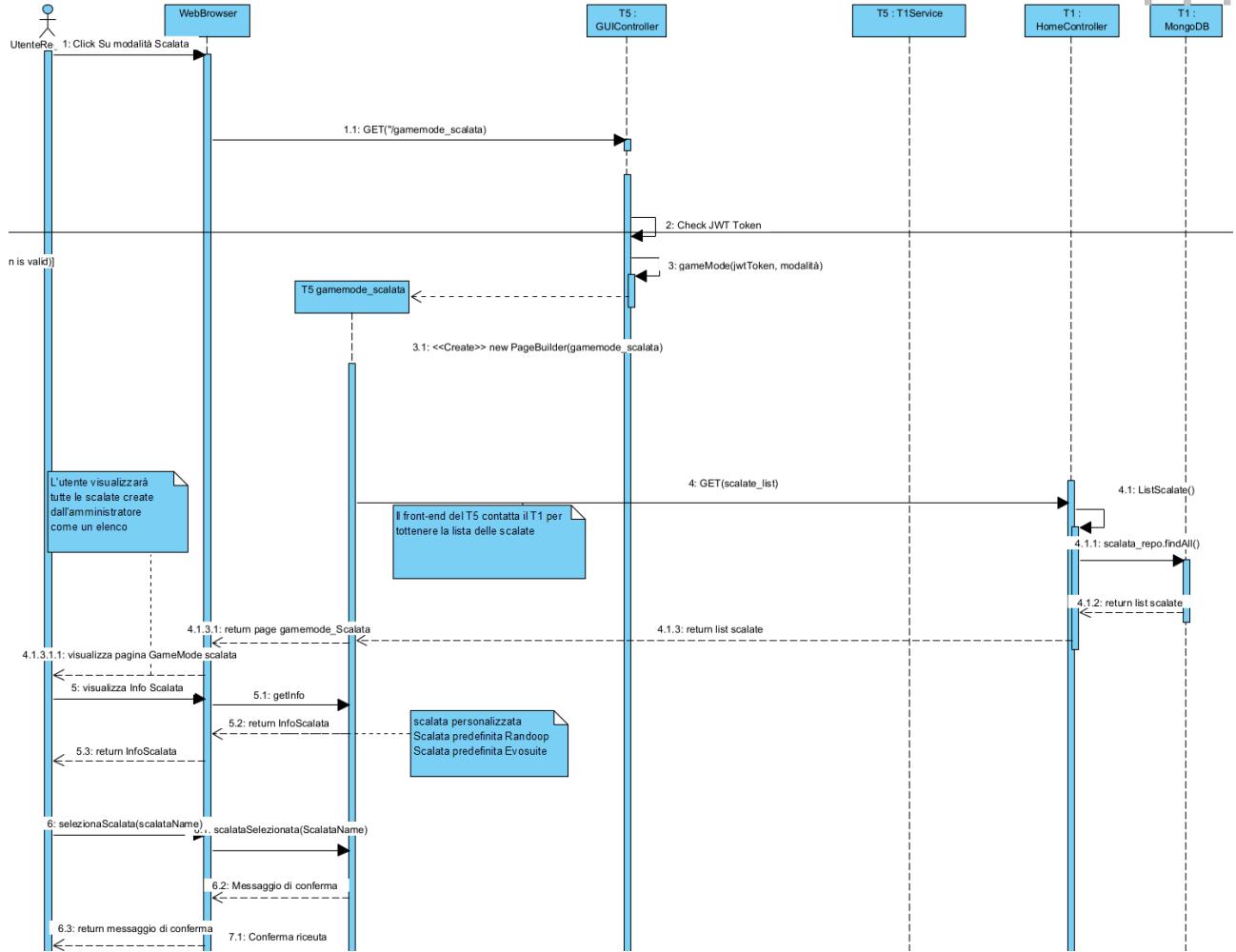
Oltre ciò, tale sequence diagram presenta una chiamata API all'apertura del documento verso T1 in cui si richiede la lista delle scalate in sistema, in modo da essere selezionabili per l'utente, e la creazione di una partita, dove viene effettuata una serie di chiamate per il salvataggio di una scalata (come già visto) e al reindirizzo all'editor.

4.4.3: Sequence Diagram di dettaglio – Scegli Scalata(nuova versione)

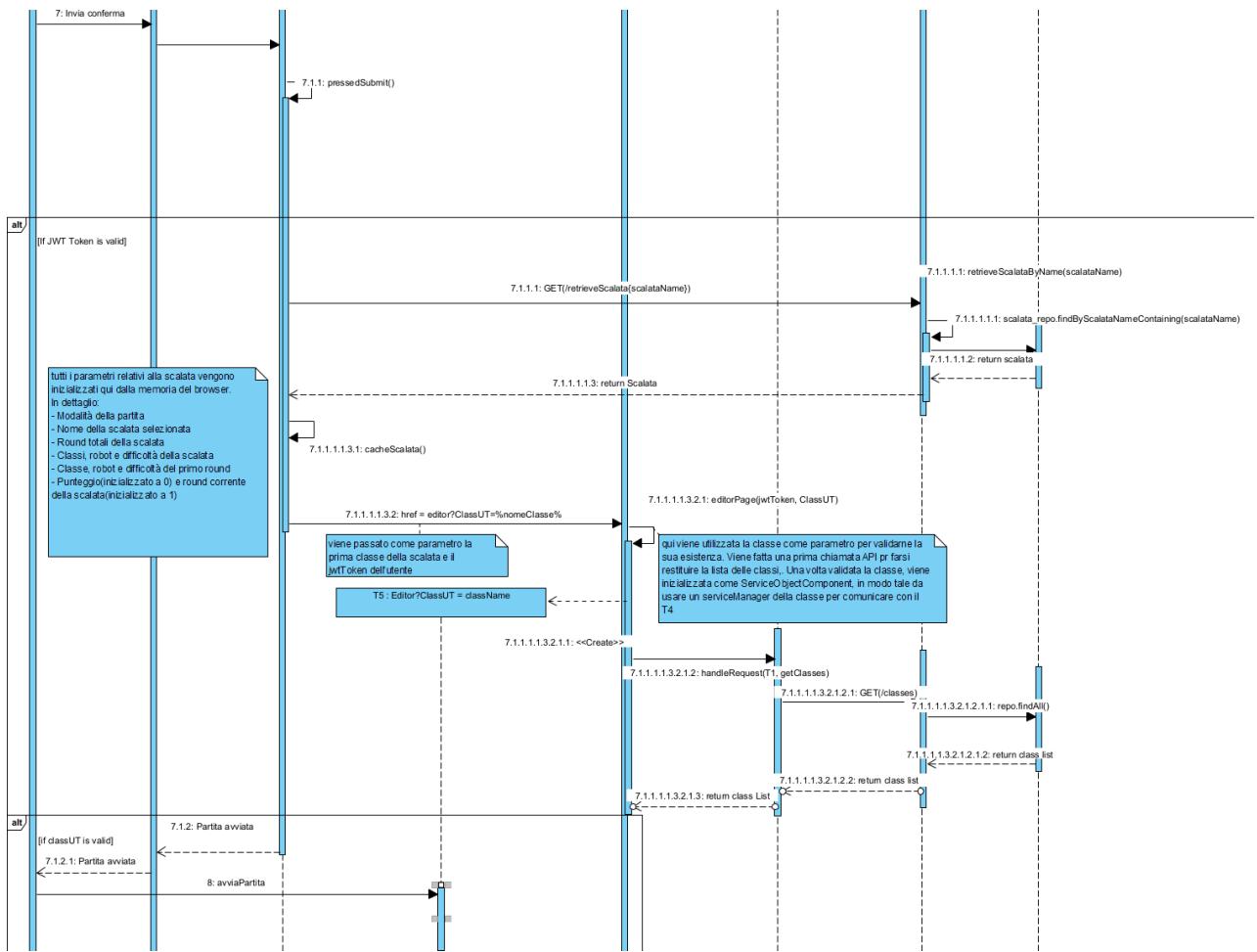
Per la visione in alta risoluzione si consiglia di cliccare su ... e poi su "view original"

La nuova versione del sequence diagram si discosta dalla vecchia a causa della scelta dell'editor. Ciò ha portato ad alcune chiamate API, come ad esempio save-scalata, a non essere più utilizzate, così come il salvataggio delle partite: questo avviene non appena viene aperto il nuovo editor.

Un'aggiunta che è stata fatta è la possibilità di ottenere informazioni relativamente al tipo di scalata. È possibile capire se la scalata è personalizzata dall'admin oppure se ha usato un preset sui robot.



Altra differenza è l'uso dell'URL parametrizzato per la classe. Questo permette di validare se la classe passata come parametro esiste, prima di creare effettivamente la pagina di gioco



4.4.4 Sequence Diagram: Inizia Partita (Vecchia versione)

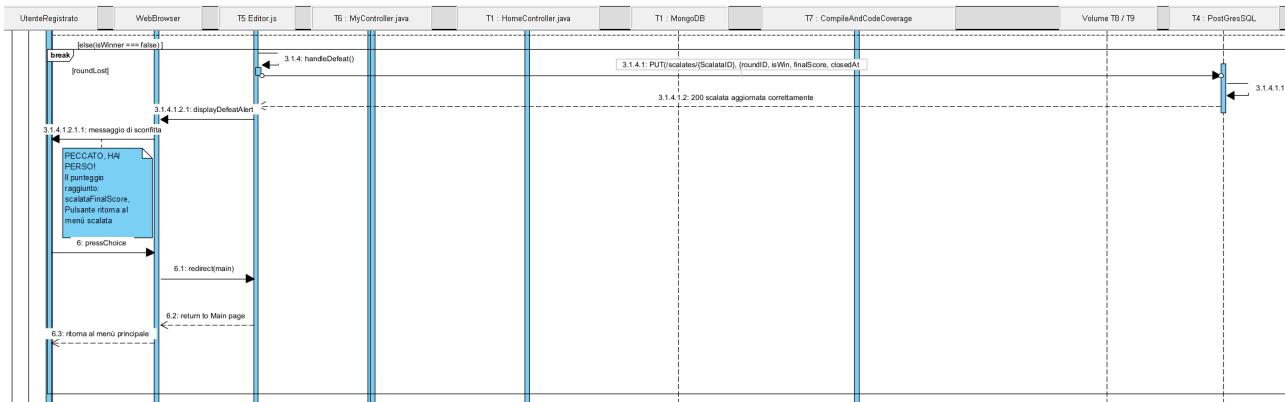
Per la visione in alta risoluzione si consiglia di cliccare su ... e poi su "view original"

Lo scenario relativo alla partita ha tantissime parti in movimento. È possibile dividere in più blocchi

- Recupero della classe da testare
- Compilazione, dove viene contattato il T6 per l'esecuzione dei test e il T7 per le coperture
- Recupero del report in T8
- Aggiornamento dei dati all'interno del T4
- Fine partita: può avviarsi un metodo calculateScalataFinalScore, che calcola tutti i risultati relativi alle partite giocate, per poi reindirizzare l'utente all'interno della leaderboard della scalata, passare alla prossima partita caricandola nel database oppure chiudere la partita.

Due scenari importanti, che evidenziano le problematiche di questo scenario, riguardano la fine della partita. In caso di vittoria del round, avveniva sempre il passaggio di evosuite di livello 1 al round successivo.

Il secondo scenario è quello relativo alla chiusura della scalata. Viene evidenziato quello relativo alla sconfitta, ma anche per quanto riguarda la gestione della vittoria avviene uno scenario simile



Questo scenario presenta un problema di tipo architetturale, dove è il front-end a contattare il database sull'aggiornamento dei dati. Ciò viola i principi del pattern MVC applicato per questo progetto, dove la view (il front-end dell'editor) deve contattare il model (ovvero il database Postgres) solamente per richiedere lo stato attuale, e non per aggiornarlo tramite una PUT.

4.4.5 Sequence Diagram: Inizia Partita (Nuova versione)

Per la visione in alta risoluzione si consiglia di cliccare su ... e poi su "view original"

Lo scenario relativo al nuovo editor presenta parecchie differenze con il vecchio editor. Spicca l'assenza del T6, presente nel diagramma precedente (il T7 non è stato rappresentato, ma viene chiamato quando vengono chieste le coperture all'interno della chiamata API /run).

Nel vecchio scenario di inizio partita, il salvataggio di partita sia per la scalata che per la partita veniva gestita da GuiEditor. Nella nuova versione, ora GuiEditor si preoccupa solo di fornire le pagine al front-end, mentre il salvataggio delle partite viene effettuato all'interno di SaveGame.

Un'altra differenza è che nel front-end si è usato solo due chiamate API per l'inizio della partita e per la sua esecuzione. Tali chiamate sono le stesse implementate per le altre modalità. Invece, nella vecchia soluzione, il salvataggio della partita e della scalata avviene in due chiamate API diverse. Questa strategia rende il flusso di comunicazioni di più semplice comprensione, sia in caso di modalità scalata, sia in modalità sfida.

4.5 CHIAMATE API

Sono state modificate i parametri richiesti dai seguenti endpoint

T5

- **/StartGame:** vengono aggiunti altri parametri opzionali alla richiesta
 - o Scalata_Name
 - o Scalata_classes
 - o Scalata_Robot
 - o Scalata_Difficulties

Inoltre, all'interno della risposta ritorna un JSON con ID di scalata, ID di game, ID di turno e ID di round nel caso di modalità scalata

- **/Run:** Aggiunto come parametro da restituire nella risposta il punteggio totale della scalata

Per la nostra soluzione, questi endpoint non sono più utilizzati all'interno dell'esecuzione di una scalata

T5:

- /Save-game
- /Save-data

T6:

- /run
- /calculateScalataFinalScore

5 – IMPLEMENTAZIONE

5.1 - T1 MODIFICHE E AGGIUNTE

MEMORIZZAZIONE:

Model.SelectedClasses.java: è stato creato un model per l'archiviazione di classe, robot e difficoltà all'interno di una scalata. In questo modo, solo i dettagli essenziali di una classe in questo contesto vengono salvati all'interno della scalata.

Util.RobotUtil.java: modifiche nel metodo *SaveRobots*: oltre a salvare le coperture in T4, salviamo anche in T1. La funzione modificata si occupa del salvataggio e della gestione di file e coperture relative ai test automatici generati per ogni classe. Gli "edit" introdotti mirano a integrare la funzionalità di memorizzazione delle percentuali di copertura dei test (coverage) all'interno di un oggetto *classUT*, che viene poi salvato nel database MongoDB. È utile notare che l'inserimento all'interno dell'array delle coperture è codificato in modo tale che

- Siano ordinati in facile, medio e difficile (1,2,3)
- Le prime tre coperture sono allocate a Randoop, e le seconde coperture per EvoSuite

Model.ClassUT: aggiunta dei parametri *robotList*, *robotDifficulty*, *coverage*.

Model.Scalata: aggiunta del parametro *List<SelectedClasses> selectedClasses*, così da memorizzare correttamente le informazioni rilevanti sulle classi presenti nella scalata: *selectedClasses* è una versione di *classUT* dove vengono memorizzati al suo interno solamente il nome, il robot e la difficoltà. Questa scelta ci permette di ottenere solamente i parametri di interesse di una classe all'interno della scalata, e non di tutte le generalità.

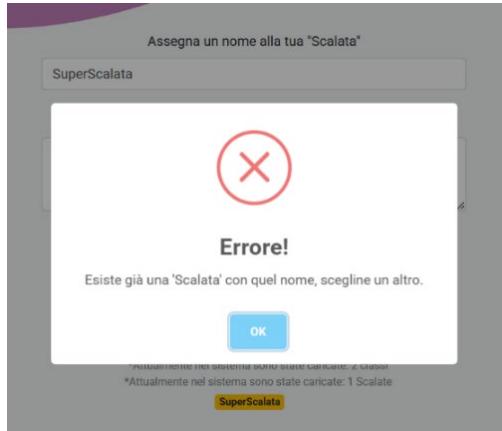
ScalataFrontEnd: modificata *submitScalataData* per includere robot e difficoltà.

RISULTATO OTTENUTO:

```
name : "FTPFile"
date : "2024-12-05"
difficulty : "Advanced"
code_Uri : "Files-Upload/FTPFile/FTPFile.java"
description : ""
  ▾ category : Array (3)
    0: ""
    1: ""
    2: ""
  ▾ robotList : Array (2)
    0: "Randoop"
    1: "Evosuite"
  ▾ robotDifficulty : Array (3)
    0: "Beginner"
    1: "Intermediate"
    2: "Advanced"
  ▾ coverage : Array (6)
    0: "68"
    1: "70"
    2: "72"
    3: "77"
    4: "100"
    5: "100"
class : "com.groom.manyclass.model.ClassUT"
```

ROBUSTEZZA:

Scalata.js: Spostato il vincolo relativo al nome della scalata direttamente alla conferma dell'inserimento di tale attributo. I cambiamenti sono stati fatti in scalata.js - *handleRoundsChange*, che si occupa del controllo degli attributi.



Controllo scalata nome aggiornato

L'eliminazione della classe presenta un controllo in più, per evitare di eliminarne una presente all'interno di una scalata creata in precedenza.

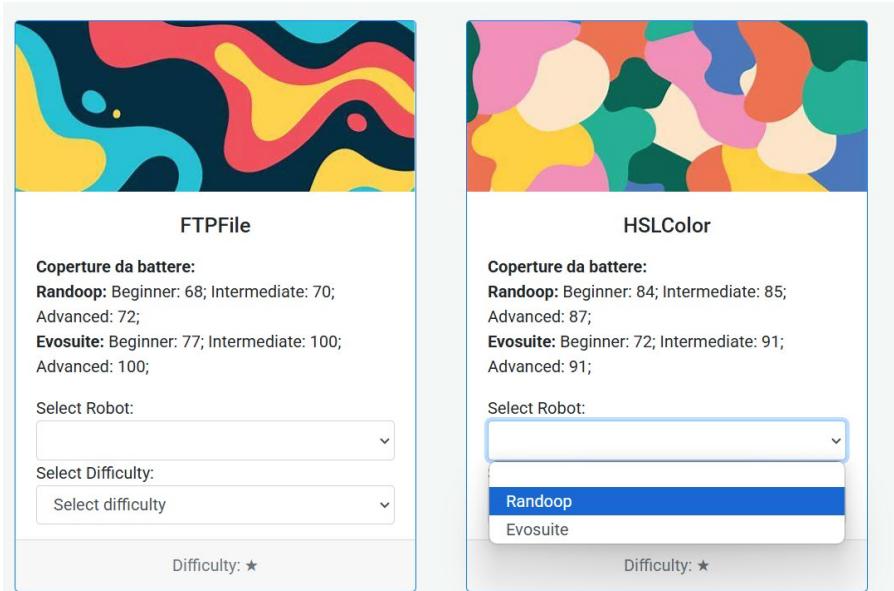


Controllo relazione classe-scalata aggiornato

PERSONALIZZAZIONE:

Scalata.js: Problema della scarsa personalizzazione permessa all'amministratore per la creazione di una scalata: la modifica delle impostazioni grafiche delle card, è stata implementata in modo tale da poter condensare al suo interno le informazioni, e la scelta di un menù a tendina per la selezione di robot e difficoltà di ogni classe di test che forma la scalata limita i passaggi che deve fare l'amministratore ed evita problematiche che può portare un inserimento errato, a differenza di opzioni selezionabili già prestabilite, che escludono inoltre, per ragioni di robustezza, la possibilità di scegliere una difficoltà nulla che un robot potrebbe avere.

L'implementazione del requisito costruito per fornire all'amministratore, all'atto della creazione di una scalata, anche le coverage dei robot, si basa principalmente sull'aggiunta di codice per visualizzarle insieme alle card che rappresentano le classi di test. I cambiamenti sono stati quindi effettuati in scalata.js - *DisplayClasses*, che si occupa della visualizzazione delle card. Poiché alcune coperture hanno valore null, abbiamo inserito un vincolo su quest'ultime, così da far visualizzare nel menu a tendina solo le coverage con valori validi.



Card aggiornate

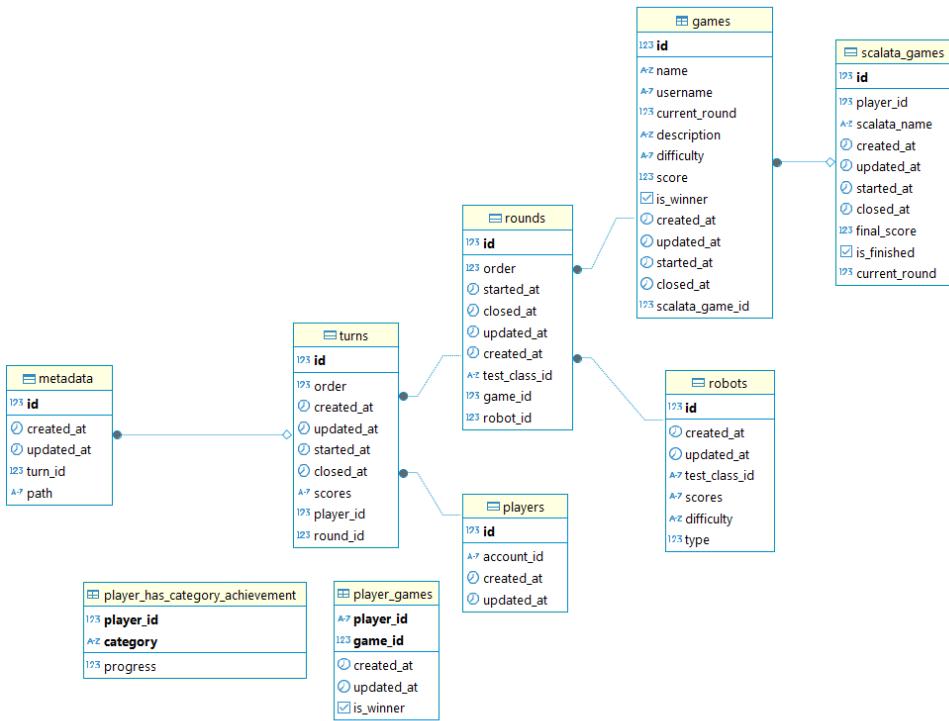
3) L'implementazione del requisito che permette all'amministratore di decidere se creare una scalata personalizzabile oppure predefinita, riduce i tempi di selezione e migliora l'esperienza. Ciò è stato effettuato aggiungendo, all'interno del gestore degli eventi, un gestore per il cambio modalità con #modeRobotSelect: se si seleziona una modalità predefinita, imposta automaticamente il robot corrispondente su tutte le card e disabilita la selezione manuale.



5.2 - T4 MODIFICHE E AGGIUNTE

Round.go, /round/service.go: È stata implementata la relazione tra Robots e Rounds, in modo da poter archiviare, all'interno di un round, l'identificativo del robot contro cui ha giocato l'utente.

Notiamo la presenza di `robot_id` in rounds, mentre l'utilità che può fornire l'attributo dell'oggetto Robot è data dall'espressione `Preload("Rounds").First(&robot, robotID)` per caricare i round associati al robot, inclusa in service.go di robot: così facendo, se hai bisogno dei dettagli del robot nei round, non devi recuperarli manualmente utilizzando RobotID.



5.3 - T5 MODIFICHE E AGGIUNTE

Prima di parlare delle modifiche implementative attuali, è utile discutere della strategia adottata per effettuare tali aggiustamenti.

All'interno di GameController, una partita viene archiviata mediante un'hashmap, dove vi è un'associazione chiave-valore con l'ID del giocatore. Questa viene istanziata mediante un oggetto di tipo GameLogic, usato per generalizzare le varie modalità di gioco.

All'inizio di questo processo, questo è risultato essere un blocco, siccome richiedeva fin da subito l'inizializzazione di una modalità scalata. All'interno di *StartGame*, la chiamata API usata per inizializzare una nuova partita, viene effettuato un controllo sulla modalità. Se questa modalità non fosse risultata esistente, sarebbe ritornato un errore.

Alla luce della necessità di memorizzare i dati relativi al robot e la difficoltà contro cui si effettuava una partita all'interno del T4, oltre all'insostenibilità a lungo termine del vecchio editor, si è deciso di effettuare il porting della scalata al nuovo editor.

Inizialmente, il passaggio da una modalità all'altra ha presentato dei blocchi dovuti dall'assenza di una modalità scalata all'interno del GameController in T5. A causa dei conflitti precedentemente citati, il port è avvenuto in tre step

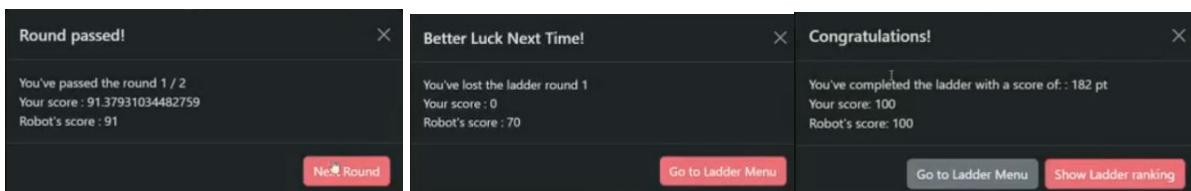
1. “Aggirando” il blocco sulle modalità di gioco, facendo passare la scalata come serie di sfide; reindirizzando così l'attenzione sulle modifiche integrative da effettuare sul front-end
2. Modificare ciò che è necessario nel back-end per conservare la partita come sessione di gioco
3. Memorizzare i dati statistici della sfida e della scalata all'interno di T4

MODIFICHE FRONTEND:

RestEditor:

il punto principale è ovviamente il trattamento separato per la modalità "Scalata": viene effettuato infatti un controllo che verifica se la modalità è "Scalata" e se il punteggio della scalata è in corso. In base a questi controlli, vengono passati più dati nel ritorno della funzione `getGameData()`, come ad esempio `scalata_name`, `scalata_classes`, `scalata_robots`. Se la Scalata è in corso, vengono utilizzati solo alcuni parametri.

Per la gestione della situazione di una scalata, è stato utilizzato principalmente il metodo già presente di gestione di fine partita, `handleEndGame`, il quale mostra il punteggio dell'utente e del robot, tramite un modal, gestendo separatamente i casi di fine round e fine gioco, con la possibilità di passare al round successivo nella modalità Scalata.



Finestre di dialogo durante una partita scalata

UtilEditor:

Nel vecchio codice le chiamate di aggiornamento e chiusura della "Scalata" erano completamente gestita nel client: veniva infatti inviata una richiesta AJAX per aggiornare lo stato della Scalata e creare un nuovo round. Le chiamate ora vengono gestite tramite un servizio Java che fa richieste a T4 per gestire la creazione, l'aggiornamento e la chiusura della Scalata: la logica è centralizzata nel backend, che gestisce le operazioni direttamente, senza necessità di inviare richieste tramite JavaScript al frontend per la maggior parte delle operazioni. Al suo interno ci sono anche dei metodi di utilità per la gestione della scalata lato front-end.

Gamemode scalata:

All'interno della modalità scalata, è stato modificato il modo di prelevare i parametri all'interno della scalata, così da poter adattarsi al cambio di tipo di rappresentazione di una classe all'interno di una scalata. Inizialmente, le classi all'interno di una scalata venivano memorizzate sotto forma di stringa; tuttavia, con l'inserimento di robot e difficoltà per ciascuna classe, è stato necessario rappresentarle con un tipo di dato differente, e pertanto andavano gestite in altro modo.

È stata poi cambiata la chiamata verso l'editor, modificando il riferimento verso quello nuovo.

Per raggiungere l'obiettivo proposto nel requisito uno, abbiamo implementato un nuovo bottone nel frontend lato utente, aggiungendolo prima nel `gamemode_scalata.html` e poi implementando la sua funzione nel `gamemode_scalata.js`: poiché la scalata non ha un attributo che ci fornisce direttamente il tipo di scalata, abbiamo deciso di analizzare il tipo di tutti i robot presenti in quella specifica scalata, così che, se sono tutti di una stessa tipologia, allora possiamo definire quella scalata come una scalata predefinita: inoltre la stampa del messaggio in questione è stata definita pensando anche alle aggiunte da fare nei

vari file.properties, che si occupano di far visualizzare i messaggi a stampa nelle diverse lingue impostate.



MODIFICHE BACKEND:

GameLogic

- adesso che T4 contiene il robot associato al round, dobbiamo aggiungere alle chiamate che vengono fatte ad ogni creazione della partita, una chiamata al T4 per ottenere l'id del robot: questo ID è poi utilizzato per altre operazioni, come la creazione di un round nel gioco, in quanto ogni round è associato a un robot specifico.

```
protected void CreateGame() {
    String Time = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
    this.GameID = (int) serviceManager.handleRequest(serviceName:"T4", action:"CreateGame", Time, this.difficulty, this.classeUT, this.gamemode, this.PlayerID);
    int robot_id =(int) serviceManager.handleRequest(serviceName:"T4", action:"GetRobotID", this.ClasseUT, this.type_robot, this.difficulty);
    this.RoundID = (int) serviceManager.handleRequest(serviceName:"T4", action:"CreateRound", this.GameID, this.ClasseUT, Time, robot_id);
    this.TurnID = (String) serviceManager.handleRequest(serviceName:"T4", action:"CreateTurn", this.PlayerID, this.RoundID, Time);
    System.out.println("ROUND:"+getRoundID()+ " GAME"+getGameID() + " TURN" + getTurnID());
}
```

- Al metodo *PlayTurn* verrà richiesto anche un booleano, *isRoundEnd*, per capire se è terminato o meno un round. Ad esempio, all'interno della classe sfida ci si è accorti che la funzione *isGameEnd* ritornava sempre false, e passando questo booleano come parametro possiamo comunicare propriamente il termine o meno di una sfida.
- Sono stati aggiunti inoltre dei metodi set per il turno, il robot, la classe sotto test e il playerID.

ScalataGame:

La precedente ScalataGame riscontrava un'implementazione poco adatta e carente per l'utilizzo polimorfico tramite GameLogic.

Inoltre, è stato necessario dotarla di un modo di leggere i parametri prelevati tramite chiamata API.

- Sono state aggiunte delle variabili per conservare l'ID della scalata, il punteggio complessivo e il nome della scalata
- È stato aggiunto un metodo statico per la bonifica dei parametri in input relativi alla scalata passati tramite chiamate API.
- Riscritti i metodi get, in modo da restituire i parametri relativi al round in corso. Ad esempio, *getScore* prima restituiva il punteggio della scalata, ma visto che tale chiamata veniva utilizzata per calcolare il punteggio della partita, è stato modificato per ritornare il punteggio di ogni partita definita dal *currentRoundIndex*.

- Aggiunto un sistema più strutturato per la gestione dello stato della scalata tramite l'enumerazione ScalataGamestatus (IN_PROGRESS, WIN, LOST).
- Il metodo principale, *PlayTurn*, è stato rivisto completamente: nella nuova versione, recupera l'oggetto Sfida corrispondente al round attuale e controlla se la sfida è terminata (*isGameEnd*). Se la coverage è sufficiente per battere il robot, il round viene considerato vinto e si chiude la partita (*EndGame*) registrando la vittoria, e il round (*EndRound*). Passa quindi al round successivo, aggiungendo il punteggio dell'utente al totale, e ciò avviene attivando il flag *nextRoundTransition* per indicare la transizione. Se tutti i round sono stati completati con successo il gioco aggiorna lo stato a *ScalataGamestatus.WIN* e chiude la scalata con *CloseScalata*. Se invece il giocatore perde il round, lo stato diventa *ScalataGamestatus.LOST*, e la scalata viene chiusa immediatamente.
- Di conseguenza, è stato riscritto il metodo *isGameEnd*, il quale ritorna *true* se lo stato corrente di *ScalataGame* è diverso da IN_PROGRESS.
- È stato riscritto anche il metodo *checkGame* per il recupero di una partita non conclusa. I parametri in ingresso vengono comparati con quelli posti all'interno del round corrente.

```

@Override
//con isGameEnd ci si riferisce al termine del round all'interno di una scalata, e non alla scalata stessa.
public void playTurn(int userScore, int robotScore, boolean isRoundEnd) {

    String now = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
    if (currentRoundIndex < games.size()) {
        Sfida currentGame = games.get(currentRoundIndex);
        currentGame.playTurn(userScore, robotScore, isRoundEnd);

        int gameCoverage = currentGame.GetCoverage();
        // Verifica se il gioco corrente è finito. Per gioco s'intende il round della scalata
        // Non so se per il passaggio al turno successivo serva il punteggio o la copertura.
        //In tal caso, la scalata ha un sistema per recuperare entrambi
        if (currentGame.isGameEnd() && gameCoverage >= robotScore) {
            currentGame.EndGame(now, robotScore, isWinner:true);
            currentGame.EndRound(now);

            System.out.println("[SCALATAGAME][T5] Round " + currentRound + " completed.");

            currentRoundIndex++; // Passa al gioco successivo
            currentRound++;
            this.totalScore += userScore;
            this.toggleRoundTransition();
            //aggiornamento del prossimo turno

            //adesso viene controllato se la scalata è terminata
            if(currentRoundIndex >= games.size()){
                System.out.println(x:[SCALATAGAME][T5] Scalata completed.");
                currentStatus = ScalataGamestatus.WIN;
            }
            else{
                loadNextRound();
                games.get(currentRoundIndex).CreateGame(id_scalata);
                this.updateScalata(id_scalata, currentRound);
            }
        }
        // Non so se per il passaggio al turno successivo serva il punteggio o la
        //copertura. In tal caso, la scalata ha un sistema per recuperare entrambi
        else if (currentGame.isGameEnd() && gameCoverage < robotScore) {
            this.currentStatus = ScalataGamestatus.LOST;
            currentGame.EndGame(now, robotScore, isWinner:false);
            currentGame.EndRound(now);
            currentGame.EndTurn(getTurnID(), now, userScore);

            System.out.println("[SCALATAGAME][T5] Round " + currentRound + " lost.");

            //gestione scalata persa
            this.closeScalata(id_scalata, getRoundID(), is_win:false, final_score:0);
        }
        else {
            currentGame.CreateTurn(now, userScore);
            System.out.println(x:[SCALATAGAME][T5] Compilation command detected.");
        }
    }
}

```

PlayTurn – Controllo del round

- Metodi come *EndGame* ed *EndRound* sono utilizzati grazie all'estensione con *GameLogic*, mentre metodi più specifici per scalata, come *CloseScalata*, devono essere implementati: chiamando poi:

```
@Override
protected void CreateGame(){
    createScalata(getPlayerID(),this.scalata_name);
    games.get(currentRoundIndex).CreateGame();
}
```

- La chiamata al metodo *CreateGame* di *GameLogic* viene utilizzata per riutilizzare la logica generica di gioco. Questo metodo assicura che il processo di inizializzazione sia completamente gestito dal backend.

```
//Chiamate a T4
public void createScalata(String player_id, String scalata_name){
    String creationDate = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_LOCAL_DATE);
    String creationTime = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
    String id_scalata_string = (String)serviceManager.handleRequest(
        serviceName:"T4",
        action:"CreateScalata",
        player_id,
        scalata_name,
        creationTime,
        creationDate);
    this.id_scalata = Integer.parseInt(id_scalata_string);
}

public void updateScalata(int scalata_id, int round_id){
    String updateTime = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
    String response = (String)serviceManager.handleRequest(
        serviceName:"T4",
        action:"UpdateScalata",
        scalata_id,
        round_id,
        updateTime
    );
}

public void CloseScalata(int scalata_id, int round_id, boolean is_win, int final_score){
    String closeTime = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
    String response = (String)serviceManager.handleRequest(
        serviceName:"T4",
        action:"CloseScalata",
        scalata_id,
        round_id,
        is_win,
        final_score,
        closeTime
    );
}
```

Sfida

Per ottenere una miglior integrazione con la scalata, è stata modificata la sfida di conseguenza. Oltre ad alcuni miglioramenti, come ad esempio le chiamate a *endGame*, *endRound* e *endTurn* all'interno del *playTurn*, si è dotata la modalità sfida di una variabile per memorizzare la copertura di codice raggiunta su una classe, al fine di stabilire la condizione di vittoria/sconfitta all'interno della scalata.

Tale copertura viene salvata all'atto della chiamata della funzione di *getScore*.

Inoltre, è stata modificata la logica di termine di una sfida, tramite la suddetta citata modifica a *gameLogic* di *playTurn* con l'aggiunta del booleano del comando di submit ricevuto come parametro. Tramite questo, è stato possibile terminare la sfida in maniera appropriata e restituire alla funzione *isGameEnd* un comportamento consono

```

@Override
public void playTurn(int userScore, int robotScore, boolean isRoundEnd) {
    String Time = ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
    currentTurn++;
    //CreateTurn(Time, userScore);
    this.isGameEnd = isRoundEnd;
    System.out.println("[GAME] Turn " + currentTurn + " played. User Score: " + userScore + ", Robot Score: " + robotScore);

    this.EndTurn(this.getTurnID(), Time, userScore);

    if(isGameEnd){
        this.EndRound(Time);
        //può essere sostituito con userScore
        this.EndGame(Time, userScore, gameCoverage > robotScore);
    }
}

@Override
public Boolean isGameEnd() {
    return this.isGameEnd;
}

```

Metodo *playTurn* di Sfida

GameController:

principali differenze implementate riguardano:

- *StartGame* introduce il supporto per parametri opzionali legati alla modalità di gioco "Scalata", utilizzati per configurare dettagli avanzati della modalità di gioco, permettendo una maggiore integrazione con il registro di giochi (*activeGames*) tramite il *gameConstructor* per la creazione della partita. La versione vecchia non faceva uso di un tale registro e la creazione di una partita era più statica, legata esclusivamente ai parametri di input. I controlli sulla modalità Scalata non portano solo ad una risposta positiva come per Sfida, ma si costruisce una risposta JSON con i dettagli della scalata e la invia al client. Per quanto riguarda i round successivi, se durante il recupero della modalità della partita attuale (*currentMode*), quest'ultima è Scalata, allora controlla se è in corso una transizione di round: se sì, invia una risposta con i dati aggiornati e resetta il flag *isRoundTransition*.

Si è cercato poi di integrare, quanto più possibile, la possibilità di recuperare una partita modalità scalata lato back-end, ma senza successo.

```

if (isGameExisting) {
    if (currentMode.equals(anObject: "Scalata")) {
        if (((ScalataGame)gameLogic).isRoundTransition()) {
            System.out.println(x:"NON sono io il problema");
            errorMessage = "Transizione al round successivo in corso";
            errorCode = "6";
            ((ScalataGame)gameLogic).toggleRoundTransition();

            JSONObject obj = buildScalataResponse(gameLogic);

            return ResponseEntity.ok(obj.toString());
        }
    }
}

```

- *GestisciPartita* gestisce il **flusso globale** della partita, rimuovendo nel caso la partita dai giochi attivi. Cerca di mantenere la stessa logica applicata da sfida, sfruttando il fatto che la scalata gestisce internamente la condizione di termine partita. Nella pratica chiama *playTurn* per aggiornare lo stato della partita; la differenza principale è nella gestione del punteggio: se è una scalata, il controller recupera il punteggio totale (*GetTotalScore*), perché la partita non si chiude finché

non finiscono tutti i round. Se è una partita normale, il controller usa semplicemente il punteggio finale del turno.

```
// Salvo i dati del turno
gameLogic.playTurn(userScore, robotScore, isGameEnd);
/*
 * La seguente funzione cerca di eliminare la partita corrente in base all'ID del giocatore,
 * per la scalata invece bisogna verificare gli stati della stessa per poter proseguire
 */
if (gameLogic.isGameEnd()) {
    //la fine partita della scalata non coincide con la fine della scalata stessa, siccome essa può avere più turni
    if (gameLogic.getMode().equals(anobject:"Scalata")) {
        totalScore = ((ScalataGame) activeGames.get(playerId)).GetTotalScore();
    } else {
        //per le partite normali, viene semplicemente rimossa la partita attiva all'interno del DB
        totalScore = userScore;
    }
    activeGames.remove(playerId);
    logger.info(msg:"[GAMECONTROLLER] /run: risposta inviata con GameEnd true");
```

MODIFICHE T4SERVICE:

oltre ad alcune modifiche a chiamate presenti in T4Service, per un miglior passaggio dei parametri da inviare con le chiamate REST, l'implementazione più interessante riguarda l'aggiunta e/o modifica di *CreateScalata*, *UpdateScalata* e *CloseScalata*, necessarie per continuare il flusso delle chiamate dei metodi in *ScalataGame*, interconnesso dalla chiamata *handleRequest*, che invoca il servizio con il comando "T4" e il nome del metodo: il suo scopo è quello di instradare correttamente la richiesta verso il servizio giusto nel back-end. Nello specifico, *UpdateScalata* aggiorna lo stato di una scalata, indicando tra le varie cose l'ID del round corrente, ed invia una richiesta PUT all'endpoint /scalates/{scalata_id} con i dati aggiornati.

CloseScalata chiude la scalata, segnando se l'utente ha vinto o perso ed il punteggio finale, ed invia una richiesta PUT con i dati relativi alla chiusura della scalata.

6 – TESTING

Nel mondo dello sviluppo software, il testing rappresenta una fase tanto critica quanto imprescindibile. Non si tratta semplicemente di "provare" il software per vedere se funziona, ma di un processo strutturato e metodico volto a garantire la qualità, l'affidabilità e la robustezza del prodotto finale. Un software ben testato non solo soddisfa i requisiti funzionali, ma offre un'esperienza utente positiva, riduce i costi di manutenzione e aumenta la fiducia nel prodotto.

6.1 - TESTING MOCK

In T5 è stato necessario effettuare modifiche e aggiunte nel T4ServiceTest.java, che si occupa del verificare il corretto funzionamento delle chiamate API che svolgono il ruolo di collegamento tra i due task. Questi test vengono chiamati integration test. Alcune modifiche durante l'implementazione in chiamate in T4Service, già esistenti, hanno impattato sui test già presenti, dove sono quindi state necessarie delle piccole modifiche. Sono stati poi aggiunti anche dei nuovi test, dovuti all'inserimento di nuove funzioni che completano il compito che sta dietro alla Modalità Scalata: UpdateScalata e CloseScalata.

```
@Test
public void testUpdateScalata_ValidUpdate() {
    // Parametri di input del metodo
    int scalataId = 567;
    int roundId = 3;
    String updateDate = "2024-10-30";

    String expectedUrl = String.format(format:"%s/scalates/%d", Base_URL, scalataId); //URL di endpoint atteso
    String mockResponse = "{\"status\": \"updated\"}"; //Mock della risposta attesa

    //Configurazione del mock server
    mockServer.expect(requestTo(expectedUrl))
        .andExpect(method( HttpMethod.PUT ))
        .andExpect(content().json(expectedJsonContent:"{\\"CurrentRound\\": 3, \\"updateDate\\": \"2024-10-30\\\"}")) // Controlla JSON inviato
        .andRespond(withSuccess(mockResponse, MediaType.APPLICATION_JSON));

    //Chiamata al metodo sotto test
    String response = (String) T4Service.handleRequest(action:"UpdateScalata", scalataId, roundId, updateDate);

    assertEquals(mockResponse, response); //Verifica della risposta
    mockServer.verify(); //Verifica delle aspettative del mock server
}

@test
public void testCloseScalata_ValidClose() {
    // Parametri di input del metodo
    int scalataId = 567;
    int roundId = 3;
    boolean isWin = true;
    int finalScore = 2500;
    String closeTime = "2024-10-30T15:30:00Z";

    String expectedUrl = String.format(format:"%s/scalates/%d", Base_URL, scalataId); //URL di endpoint atteso
    String mockResponse = "{\"status\": \"closed\", \"FinalScore\": 2500, \"isFinished\": true}"; //Mock della risposta attesa

    //Configurazione del mock server
    mockServer.expect(requestTo(expectedUrl))
        .andExpect(method( HttpMethod.PUT ))
        .andExpect(content().json(expectedJsonContent:"{\\"CurrentRound\\": 3, \\"isFinished\\": true, \\"FinalScore\\": 2500, \\"ClosedAt\\": \"2024-10-30T15:30:00Z\\\"}"))
        .andRespond(withSuccess(mockResponse, MediaType.APPLICATION_JSON));

    // Chiamata al metodo sotto test
    String response = (String) T4Service.handleRequest(action:"closeScalata", scalataId, roundId, isWin, finalScore, closeTime);

    assertEquals(mockResponse, response); //Verifica della risposta
    mockServer.verify(); //Verifica delle aspettative del mock server
}
```

6.2 - TESTING MANUALE

Un **Test Case** è un insieme di condizioni e variabili utilizzate per verificare che un software funzioni come previsto. La **descrizione** fornisce un riepilogo del test e del comportamento atteso. Le **classi di equivalenza coperte** rappresentano i gruppi di input validi o non validi considerati nel test. Le **pre-condizioni** sono i requisiti che devono essere soddisfatti

prima di eseguire il test, come ad esempio l'autenticazione dell'utente o la presenza di dati nel sistema. L'**input** è il valore o il set di valori forniti al sistema per il test. L'**output atteso** è il risultato previsto in base alla specifica del software, mentre le **post-condizioni attese** descrivono lo stato del sistema dopo l'esecuzione del test. L'**output ottenuto** e le **post-condizioni ottenute** documentano i risultati effettivi della prova. Infine, l'**esito** del test viene valutato come **PASS** se il comportamento osservato corrisponde a quello atteso o **FAIL** se il sistema non si comporta come previsto. Questa metodologia permette di identificare e correggere errori, migliorando la qualità complessiva del software.

Creazione di una scalata:

Test Case Id	Descrizione	Classi di equivalenza coperte	Pre-condizioni	Input	Output attesi	Post-condizioni attese	Output ottenuti	Post-condizioni ottenute	Esito(FAIL, PASS)
1	Un amministratore autenticato crea una scalata con nome, descrizione e numero di rounds valido.	Creazione scalata valida	-L'utente è autenticato come amministratore -sono presenti classi caricate nel sistema	-Nome: "Scalata1" -Descrizione: "prima scalata" -Numero di round: 2 -Classi selezionate: [FTPFile,HSLColor]	Risposta 200 Created con messaggio: "Scalata creata con successo!"	La scalata viene memorizzata nel database.	Risposta 200 La tua 'Scalata' è stata configurata con successo.	La scalata viene memorizzata nel database.	PASS
2	L'amministratore prova a creare una scalata con un numero di round negativo o non valido.	Numero di round non valido.	-L'utente è autenticato come amministratore.	-Nome: "Scalata1" -Descrizione: "prima scalata" -Numero di round: -2	Risposta 400 Bad Request con messaggio: "Numero di round non valido"	Nessuna scalata viene salvata nel database.	Il sistema non permette di inserire numeri minori di 0, per i valori 0 e 1 restituisce: La modalità 'Scalata' prevede un numero minimo di rounds pari a 2, imposta un valore corretto	Nessuna scalata viene salvata nel database.	PASS
3	L'amministratore prova a creare una scalata con un numero di round maggiore del numero di classi presenti nel sistema	Numero di round non valido.	-L'utente è autenticato come amministratore.	-Nome: "Scalata1" -Descrizione: "prima scalata" -Numero di round: 100	Risposta 400 Bad Request con messaggio: "Numero di round non valido"	Nessuna scalata viene salvata nel database.	Il sistema non permette all'utente di selezionare un numero di round maggiore del numero di classi presenti	Nessuna scalata viene salvata nel database.	PASS
4	L'amministratore prova a creare una scalata con un nome già presente nel database.	Nome duplicato.	-L'utente è autenticato come amministratore.	- Nome: "Scalata1" -Descrizione: "prima scalata" -Numero di round: 2 -Classi selezionate: [FTPFile,HSLColor]	Risposta 409 Conflict con messaggio: "Esiste già	Nessuna scalata duplicata viene salvata nel database.	Esiste già una 'Scalata' con quel nome, sceglie un altro.	Nessuna scalata duplicata viene salvata nel database.	PASS

			-Esiste già una scalata con il nome "Scalata1".		una scalata con questo nome"				
5	L'amministratore prova a creare una scalata senza nome o con nome vuoto.	Nome non valido.	-L'utente è autenticato come amministratore.	-Nome: "" -Descrizione: "prima scalata" -Numero di round: 2 -Classi selezionate: [FTPFile,HSLColor]	Il nome della scalata non può essere vuoto	Nessuna scalata salvata nel database.	Inserisci un nome per la tua 'Scalata' prima di procedere.	Nessuna scalata salvata nel database.	PASS
6	L'amministratore tenta di creare una scalata senza selezionare classi.	Scalata senza classi.	- L'utente è autenticato come amministratore	- Nome: "Scalata1" - Descrizione: "prima scalata" - Numero di round: 2 - Classi selezionate: []	Devi selezionare una classe per ogni round	Nessuna scalata viene salvata nel database.	Seleziona un numero di classi pari al numero di rounds definiti in precedenza prima di procedere.	Nessuna scalata viene salvata nel database.	PASS
7	L'amministratore prova a creare una scalata con un numero di classi inferiore al numero di round richiesto.	Numero di classi insufficiente.	- L'utente è autenticato come amministratore	- Nome: "Scalata1" - Descrizione: "prima scalata" - Numero di round: 2 - Classi selezionate: [FTPFile]	Devi selezionare una classe per ogni round	Nessuna scalata viene salvata nel database.	Seleziona un numero di classi pari al numero di rounds definiti in precedenza prima di procedere.	Nessuna scalata viene salvata nel database.	PASS

Partita in modalità scalata:

Test Case Id	Descrizione	Classi di equivalenza coperte	Pre-condizioni	Input	Output attesi	Post-condizioni attese	Output ottenuti	Post-condizioni ottenute	Esito(FAIL, PASS)
1	L'utente avvia una partita con successo.	Partita valida.	- L'utente è autenticato. - Ha selezionato una scalata valida.	-Scalata scelta: "Scalata1".	La partita inizia dal round 1.	La partita viene inizializzata con la prima classe.	la partita inizia dal round 1.	La partita viene inizializzata con la prima classe.	PASS
2	L'utente prova ad avviare una partita senza aver scelto una scalata.	Scalata non selezionata.	- L'utente è autenticato.	- Nessuna scalata selezionata.	Devi selezionare una scalata	La partita non viene avviata.	Devi selezionare una 'Scalata' dall'elenco per poter proseguire.	La partita non viene avviata.	PASS
3	L'utente avvia la partita e completa un round con successo.	Round superato.	- L'utente è autenticato. - La partita è stata avviata.	- Test scritti correttamente. - L'utente ha ottenuto un punteggio \geq del robot.	-il round avanza.	Il round passa al successivo.	Round Superato! -Hai superato il round ½ -Il tuo punteggio: 91 -Il punteggio del robot: 72	Il round passa al successivo.	PASS

4	L'utente perde un round con punteggio inferiore al robot.	Partita persa.	- L'utente è autenticato. - La partita è stata avviata.	- Test scritti con punteggio < del robot.	"Hai perso la partita".	La partita termina con una sconfitta.	Peccato! -Hai perso la scalata al round 1 -il tuo punteggio: 0 -il punteggio del robot: 72	La partita termina con una sconfitta.	PASS
5	La copertura del robot affrontato corrisponde alla copertura in possesso	Partita valida	-L'utente è autenticato - La partita è stata avviata in modalità scalata - L'utente affronta il robot Randoop di livello 3 sulla classe HSLColor	- Test inviato(anche vuoto)	-Copertura robot: 87		Il punteggio del robot : 87		PASS
6	L'utente vince la partita completando tutti i round.	Partita vinta.	- L'utente è autenticato. - Ha completato tutti i round con successo.	-Ultimo round superato.	"Hai vinto la partita!".	La partita termina con una vittoria.	Complimenti! -Hai completato la scalata con un punteggio di:191	La partita termina con una vittoria.	PASS

							-il tuo punteggio: 100 -il punteggio del robot: 70		
7	Alla fine della partita, le statistiche vengono salvate.	Salvataggio statistiche.	- L'utente ha terminato la partita (vittoria o sconfitta).	-Fine partita.	statistiche salvate nel database.	Le statistiche dell'utente vengono aggiornate.	Le statistiche salvate nel database.	Le statistiche dell'utente vengono aggiornate.	PASS

6.3 - USER SESSION-BASED TESTING

Per registrare i procedimenti necessari alla creazione di una scalata all'interno della nostra applicazione, utilizziamo Katalon Recorder. Si tratta di un potente strumento per l'automazione dei test, particolarmente utile per la registrazione, riproduzione e analisi delle azioni eseguite su un'interfaccia utente. Grazie alla sua semplicità d'uso e alla capacità di generare script testabili e riproducibili, Katalon Recorder ci permette di documentare in modo preciso le operazioni richieste per la realizzazione della scalata, assicurando che ogni passaggio venga eseguito correttamente e in maniera ripetibile.

Command	Target	Value
open	http://localhost/scalata	
click	id=form-name	
type	id=form-name	montagna
click	id=FormControlTextareaDescription	
type	id=FormControlTextareaDescription	scalata a salire
type	id=rounds	1
click	id=rounds	
type	id=rounds	2
click	id=addRoundButton	
click	xpath=(//*[normalize-space(text()) and normalize-space(.)='Rounds info'])[1]/following::button[1]	
click	xpath=/div[@id='classUTList']/div/div/select	
select	xpath=/div[@id='classUTList']/div/div/select	label=Randoop
click	xpath=/div[@id='classUTList']/div/div/select[2]	
select	xpath=/div[@id='classUTList']/div/div/select[2]	label=Beginner
click	xpath=/div[@id='classUTList']/div[2]/div/select	
select	xpath=/div[@id='classUTList']/div[2]/div/select	label=Randoop
click	xpath=/div[@id='classUTList']/div[2]/div/select[2]	

chooseOkOnNextConfirmation		
select	xpath=/div[@id='classUTList']/div[2]/div/select[2]	label=Beginner
click	id=confirmButton	
assertConfirmation	Vuoi procedere con l'operazione?	
click	xpath=(//*[normalize-space(text()) and normalize-space(.)=concat('La tua ', " ", 'Scalata', " ", 'è stata configurata con successo.'))][1]/following::button[1]	

Rieseguendo il test case con KatalonRecorder, dovremmo aspettarci che fallisca, poichè simulando nuovamente significa che reinserirà lo stesso nome alla scalata dato durante la registrazione del test case, cosa che non è ammessa.

Command	Target	Value
open	http://localhost/scalata	
click	id=form-name	
type	id=form-name	montagna
click	id=FormControlTextareaDescription	
type	id=FormControlTextareaDescription	scalata a salire
type	id=rounds	1
click	id=rounds	
type	id=rounds	2
click	id=addRoundButton	
click	xpath=(//*[normalize-space(text()) and normalize-space(.)='Rounds info'])[1]/following::button[1]	
click	xpath=/div[@id='classUTList']/div/div/select	
select	xpath=/div[@id='classUTList']/div/div/select	label=Randoop

Per testare una simulazione durante una scalata giocata dall'utente, possiamo testare dapprima il caso di una scalata persa. Katalon Recorder dà la possibilità di valutare l'output tramite asserzioni, un'affermazione che viene utilizzata nei test per verificare se una condizione è vera o falsa durante l'esecuzione di un programma o di un test. In particolare usiamo **assertText**, un comando che viene utilizzato per verificare che un determinato testo sia presente in una pagina web. È utilizzato per validare che il sistema stia restituendo o mostrando il contenuto corretto durante l'esecuzione del test. Nello specifico di una scalata persa, ho compilato senza scrivere nessun metodo di test, e quindi mi aspetto, quando il confronto con la coverage del robot è terminato, che il sistema mostri che l'utente ha perso, col suo punteggio (un valore di LOC uguale a zero) e con quello del robot affrontato. Inoltre utilizziamo un comando di attesa: Katalon propone ad esempio "waitForText", per fare in modo che il test aspetti che un determinato testo sia visibile o presente in una pagina web prima di procedere con il passo successivo del test: è particolarmente utile quando si lavora con applicazioni web dinamiche, dove il contenuto della pagina può essere aggiornato o caricato in modo asincrono (ad esempio, dopo una richiesta AJAX o una risposta del server). In questi casi, è importante sincronizzare il test con l'apparizione di un testo specifico, per evitare che il test fallisca prima che la pagina sia pronta o aggiornata con il contenuto atteso. Noi utilizzeremo invece "pause", attendendo un numero fisso di millisecondi prima di passare al comando successivo.

Command	Target	Value
click	id=coverageButton	
pause		40000
assertText	xpath="//div[@id='section_result']/div/div[6]/div/div/div/div[5]/pre[2]	Il tuo punteggio: Opt
click	id=loading_run	
pause		40000
storeEval	document.getElementById('Modal_body').innerText.trim()	cleanText
echo	\$(cleanText)	
assertText	id=Modal_body	\$(cleanText)
click	link=Vai alla home	
open	http://localhost/main	

echo:

Hai perso la scalata al round 1 Il tuo punteggio: 0 Il punteggio del robot : 84

7 – STRUMENTI UTILIZZATI

L'utilizzo di strumenti come DBeaver, MongoDB, Docker, VS Code, Trello, Microsoft Teams, Microsoft Word, Notepad++, Visual Paradigm, GitHub e GitHub Desktop è stato essenziale per la gestione e lo sviluppo del progetto:

- **DBeaver:** è un potente client database utilizzato per gestire, visualizzare e interrogare database, tra cui MySQL e Postgres, database di nostra attenzione. Facilita la visualizzazione, gestione e debugging dei dati dei player, delle partite e delle varie parti che la compongono, essenziale per monitorare il repository dei dati di gioco (T4).
- **MongoDB Compass:** è un GUI ufficiale di MongoDB, progettato per semplificare la gestione, l'analisi e la manipolazione dei dati in database MongoDB, un database NoSQL che gestisce dati in formato JSON-like, ideale per un sistema basato su microservizi. Offre un'interfaccia intuitiva che consente di: visualizzare i dati in modo interattivo, senza bisogno di usare solo comandi da terminale, sotto forma di documenti JSON strutturati o tabelle interattive, eseguire query in modo visuale con

filtri e ordinamenti, e gestire indici, documenti e collezioni senza dover scrivere codice.

- **Docker:** è fondamentale per containerizzare i microservizi e garantire un ambiente di esecuzione coerente. Permette di orchestrare e scalare i servizi indipendenti senza dipendenze tra di loro.
- **VS Code:** è un IDE versatile, perfetto per sviluppare il front-end e il back-end dei microservizi: supporta estensioni per Java, JavaScript, Docker e MongoDB, semplificando lo sviluppo e il debugging, e inoltre può essere utilizzato per scrivere e testare codice in JUnit.
- **Trello:** piattaforma di project management fondamentale per la pianificazione, il tracciamento e la gestione agile delle attività del team durante le iterazioni. Permette di definire i task a partire dal Product Backlog, assegnare compiti, coordinare il team e rispettare le scadenze, supportando il processo di sviluppo scelto.
- **Microsoft Teams:** Una piattaforma di comunicazione e collaborazione che facilita l'interazione tra i membri del team. Microsoft Teams offre funzionalità di chat, videoconferenza, condivisione di file e strumenti di gestione dei progetti, contribuendo a mantenere il team allineato e produttivo.
- **Microsoft Word:** Un software di elaborazione testi utilizzato per la creazione e la gestione della documentazione del progetto, inclusi i requisiti, le specifiche, i report e altri documenti. Microsoft Word offre strumenti di formattazione, revisione e collaborazione che semplificano la creazione di documenti professionali e completi.
- **NotePad++:** un editor di testo avanzato e leggero, particolarmente utile per la modifica rapida di file di configurazione, script e codice sorgente. Supporta la colorazione della sintassi per numerosi linguaggi di programmazione e offre funzionalità avanzate come il confronto tra file, la ricerca con espressioni regolari e la modifica simultanea di più linee. È uno strumento utile per operazioni rapide che non richiedono un IDE completo. Nel nostro progetto è stato particolarmente utile grazie al suo plugin per comparare i testi.
- **Visual Paradigm:** un potente strumento di modellazione UML e progettazione software, utile per la creazione di diagrammi ER, diagrammi delle classi, diagrammi di sequenza e altri modelli architetturali. Facilita la progettazione del sistema, consentendo di definire visivamente l'architettura e il flusso di dati tra i componenti.
- **GitHub e GitHub Desktop:** GitHub è una piattaforma di hosting di codice e collaborazione che offre un sistema di controllo di versione basato su Git. GitHub Desktop è un'applicazione desktop che semplifica l'interazione con GitHub, permettendo al team di gestire il codice sorgente, collaborare su progetti condivisi e tenere traccia delle modifiche in modo efficiente

Di seguito sono riportate tutte le boards di Trello per ogni iterazione:

Prima iterazione:



Seconda iterazione:



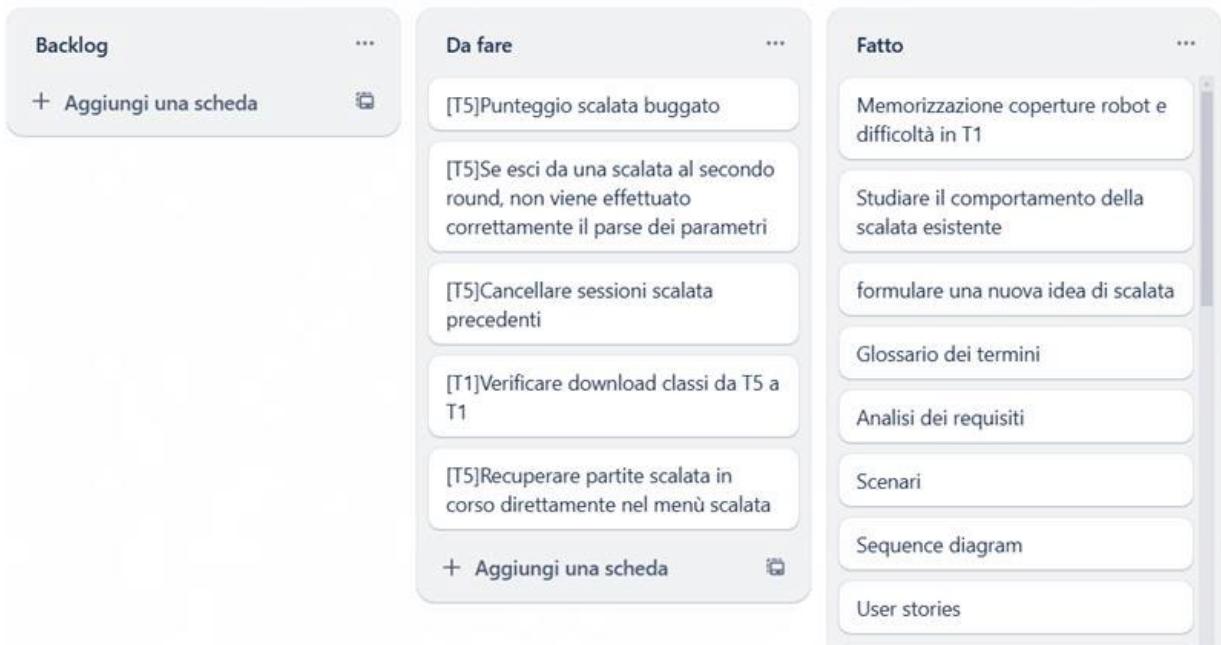
Terza iterazione:



Quarta iterazione:



Quinta iterazione:



8 - SVILUPPI FUTURI

L'aspettativa su questo progetto è stata, fin da subito, di non poter effettuare un lavoro a 360 gradi sulla modalità scalata, ma cercare di rendere il tutto più integrabile possibile anche con il lavoro svolto sia in precedenza, sia svolto dagli altri.

Alcuni aspetti relativi a questa modalità non rientravano nelle nostre mansioni, ma siccome si è fatto tanto lavoro per restituire una modalità quanto più vicina possibile allo stato attuale, possiamo suggerire come è possibile integrare migliorie che sono state apportate dagli altri gruppi.

Ad esempio, si poteva migliorare tanto la leaderboard della scalata, ma sapendo che c'erano altri gruppi che avevano come obiettivo una classifica che riguardasse anche la modalità sfida, la si è lasciata immutata, consapevoli che in iterazioni future questa verrà rimpiazzata. Tuttavia, per quanto possiamo aver descritto al meglio il nostro lavoro, molte idee e aggiustamenti sono stati accantonati per questioni di tempo. Viene sfruttato questo spazio

per parlare quindi di possibili idee su cosa può essere integrato, e per alcuni aggiustamenti verrà discusso il come.

8.1 Fix necessari

Viene segnalato che il comportamento del recupero della scalata precedente è lontano dall'essere considerato robusto. Il recupero di una scalata precedente può essere effettuato solamente all'interno della modalità sfida. Se si avvia una partita all'interno di modalità scalata, dopo aver lasciato una partita precedente in sospeso, possono verificarsi degli errori lanciati all'interno del metodo Runner in T5.

Si segnala, inoltre, che il download dei file effettuato dall'editor presenta un bug nel momento in cui si ricompila il T1. Si pensa che una delle cause sia dovute al fatto che il download viene effettuato all'interno di T1, il quale non memorizza le classi in maniera persistente, a differenza del T8.

8.2 Possibili aggiunte

Scalata dinamica: il sistema, nel caso in cui l'utente perda una partita, ma ha già battuto una classe di livello 1 in una scalata in cui ci sono più classi dello stesso livello, deve dare la possibilità all'utente di ripetere la scalata, ripartendo da una classe di livello 1 randomicamente selezionata tra quelle presenti nella scalata.

Possibilità di rimpiazzare una nuova classifica con l'attuale leaderboard della scalata: è sufficiente modificare il link di redirezione all'interno del bottone “visualizza leaderboard scalata” all'interno di REST_Editor con quello della nuova classifica

Possibilità di avere un'anteprima della scalata sia nel menù di creazione della scalata in T1, sia nel menù di gioco.

Scalata hardcore: possibilità di fare la scalata senza ausilio di aiuti esterni come possono essere il caricamento dei file oppure disabilitando il comando di incolla all'interno dell'editor.

Scalata a tempo: come dietro all'idea della scalata hardcore, si vuole dare la possibilità a giocatori più esperti di cimentarsi in modalità differenti e più stimolanti. Una soluzione innovativa potrebbe quindi essere una scalata dove per ogni round è assegnato un tempo limite, magari impostato a seconda della difficoltà del round stesso. Si potrebbe inserire anche un modo per incrementare il tempo, nel momento in cui ad ogni controllo della test suite attuale, se è stato coperto nuovo codice e/o è stato raggiunto un livello di coverage prestabilito, allora il timer s'incrementerà: così facendo la modalità aumenterebbe la tensione del giocatore, incentivando una strategia più efficace. Infine si potrebbe aggiungere un'altra sezione nella leaderboard già implementata da altri studenti, ovvero una “classifica dei migliori tempi”, incentivando la competizione.

Al termine di una scalata, all'interno di REST_Editor.js nel metodo di handleEndGame, c'è un riferimento alla vecchia leaderboard della scalata. In questo punto, noi pensiamo che sia più appropriato inserire un riferimento a una classifica migliore.

Condizione di vittoria di un round: Al momento, la “winning condition” è contesa tra

- il confronto fra la copertura ottenuta dall'utente contro quella del robot,
- il confronto fra il punteggio utente con la copertura del robot.

Il punteggio utente è ottenuto a partire dalla copertura ottenuta, e viene aggiunta una penalità ogni qual volta si andasse a compilare. Questa logica è presente all'interno del back-end di Sfida, ed è stata utilizzata anche per la Scalata.

Vista l'induzione su una tematica così importante, riteniamo doveroso precisare che si è optati per la prima strada, ovvero si vince il round usando la copertura dell'utente come condizione di vittoria.

Nel caso in cui si dovesse cambiare approccio, è sufficiente sostituire il punteggio utente con la copertura dell'utente, effettuando le modifiche all'interno di questi file

- playTurn di Sfida(T5 back-end)
- playTurn di ScalataGame(T5 back-end)
- handleEndGame di REST_Editor(T5 front-end)

Appendice

Appendice A: ChangeLog

Di seguito viene riportato, in forma tabellare, tutte le modifiche effettuate all'interno della modalità scalata

T5 Backend:

Nome	Tipo	Modifiche
Game/GameController.java		
GameFactoryFunction	Interfaccia	Aggiunta dei parametri per la gestione della scalata.
RegisterGames	Metodo	Aggiunta della registrazione di scalata.
StartGame	Metodo	Aggiunti parametri opzionali per la scalata Aggiunta gestione della modalità "Scalata" con creazione dell'oggetto ScalataGamee restituzione dati utili a UtilEditor. Il controllo della partita esistente tiene conto se la scalata sta effettuando una transizione al round successivo.
TotalScore	Variabile	Fondamentale per calcolare il punteggio totale della scalata.
processScalataParameters	Metodo	Chiamata al metodo implementata ScalataGame.

buildScalataResponse	Metodo	Costruisce la risposta da inviare in caso di partita in modalità scalata.
gestisciPartita	Metodo	Migliorato controllo di fine partita, in modo da distinguere termine sfida e termine scalata. Aggiunta restituzione del punteggio totale della scalata.
Game/GameLogic.java		
PlayTurn	Metodo Astratto	Aggiunto il booleano isRoundEnd per verificare la fine di un round.
CreateGame	Metodo	Aggiornato il vecchio metodo. Aggiunto un CreateGame(int scalataID) per gestire la creazione della partita con un ID Scalata.
GameLogic	Classe	Aggiunta setter/getter. Ritorna il PlayerID, tipo di robot, difficoltà, classe.
Game/ScalataGame.java		
PlayTurn	Metodo	Aggiunto boolean isRound End per controllare la fine del round. Modificato per gestire la logica della scalata nei turni con l'aggiunta del controllo sullo stato della scalata (IN_PROGRESS, WIN, LOST) e gestione dell'aggiornamento del turno.
GetScore	Metodo	Aggiunta dell'aggiornamento del totalScore.
ScalataGame	Classe	Aggiunta metodi getter/setter per id_scalata, currentRoundID, currentGameID, currentTurnID
ProcessScalataParameters	Metodo	Aggiunta del metodo per “normalizzare” i parametri che sono passati come array json.
CheckGame	Metodo	Viene fatto il controllo mediante round corrente.
nextRoundTransition	Attributo/ Metodo	Permette di notificare all'esterno quando la scalata passa al round successivo.
toggleRoundTransition	Metodo	Imposta il booleano nextRoundTransition rendendolo true quando si passa al round successivo in playTurn di ScalataGame, e false quando viene fatto il controllo al round successivo in startGame nel GameController.

isRoundTransition	Metodo	Ritorna il valore di nextRoundTransition
Game/Sfida.java		
PlayTurn	Metodo	<p>Abbiamo effettuato la modifica perché è un metodo ereditato da GameLogic e dato che abbiamo il playTurn in GameLogic abbiamo adattato il metodo con l'aggiunta del booleano isRoundEnd.</p> <p>Sono state aggiunte le chiamate a createTurn, EndRound ed endGame.</p>
GameCoverage	Variabile	Aggiunta variabile per ottenere la copertura del codice tramite getScore. Aggiunto anche il relativo get.
GetScore	Metodo	Imposta la variabile gameCoverage con la copertura ricevuta in ingresso.
Interfaces/BaseService.java		
CallRestPut	Metodo	Aggiunte per un put di un jsonObject.
Interfaces/T4Service.java		
registerAction("CreateGame")	Registrazione azione REST	Aggiunta del parametro scalataId di tipo Optional<Integer> per la creazione di un gioco, modificando la chiamata REST per includerlo.
registerAction("CreateRound")	Registrazione azione REST	Aggiunto il parametro robot_id di tipo int per la creazione di un round, per includere l'ID del robot.
registerAction("EndTurn")	Registrazione azione REST	Modifica dei parametri (ordine cambiato da (String, String, int) a (int, String, String)) e aggiunta la gestione del robot ID.
registerAction("GetRobotID")	Registrazione azione REST	Aggiunta di una nuova azione REST per recuperare l'ID del robot.
CreateTurn	Metodo privato REST (chiamata POST)	Aggiunta gestione del response JSON.
EndTurn	Metodo privato REST (chiamata PUT)	Conversione del payload da MultiValueMap<String, String> a JSONObject per inviare dati in formato JSON. L'ID del turno (turnId) ora è una String invece di int.

registerAction("UpdateScalata")	Registrazione azione REST	Aggiunta la registrazione per l'aggiornamento della scalata.
registerAction("CloseScalata")	Registrazione azione REST	Aggiunta la registrazione per la chiusura di una scalata.
CreateGame	Metodo privato REST (chiamata POST)	Aggiunta della logica per inviare i dati di creazione del gioco, inclusa la gestione del parametro scalataId opzionale.
EndGame	Metodo privato REST (chiamata POST)	Metodo rimasto invariato ma invece di utilizzare MultiValueMap<String, String> formData abbiamo utilizzato un JSONObject obj.
CreateRound	Metodo privato REST (chiamata POST)	Aggiunta del parametro robot_id per la creazione di un round.
EndTurn	Metodo privato REST (chiamata PUT)	Modificato per inviare i dati di fine turno come JSON tramite una chiamata PUT.
GetRobotID	Metodo privato REST (chiamata GET)	Nuovo metodo per ottenere l'ID del robot, usando i parametri classUT, robot_type e difficulty in una chiamata GET a /robots.
CreateScalata	Metodo privato REST (chiamata POST)	Nuovo metodo per creare una scalata, che invia i dati come JSON e restituisce l'ID della scalata creata.
UpdateScalata	Metodo privato REST (chiamata PUT)	Aggiunta la logica per aggiornare una scalata, modificando il round corrente e la data di aggiornamento.

CloseScalata	Metodo privato REST (chiamata PUT)	Aggiunta la logica per chiudere una scalata, inviando il risultato finale, il punteggio e la data di chiusura tramite una chiamata PUT.
---------------------	------------------------------------	---

T5 Frontend:

Nome	Tipo	Modifiche
js/Button_Editor.js		
DOMContentLoaded	Event Listener	Aggiunto un controllo if per verificare se modalita è "Scalata". Se modalita è "Scalata", aggiunge round_corrente_info nel messaggio.
flush_localStorage	Funzione	Se modalita è "Scalata", vengono eliminati anche i dati specifici della scalata (scalataId, SelectedScalata, gameId, ecc.).
js/REST_Editor.js		
GetGameData	Funzione	Introdotto controllo per modalita == "Scalata", Aggiunto is_scalata_inprogress per verificare se la scalata è in corso, Se la scalata non è iniziata, chiama handleScalataParameters(), aggiunti nuovi parametri nel return (scalata_name, scalata_classes, ecc.). Riposizionato controllo sul parametro del link.
\$(document).ready	Event Listener	Aggiunto if(is_scalata_inprogress == false) prima di chiamare startGame(data). Aggiunto try-catch, dove in caso di eccezione viene mostrato un alert che porta al menu principale.
handleResponse	Funzione	Aggiunto totalScore tra i parametri estratti da response.
processCoverage	Funzione (async)	Aggiunto totalScore tra i parametri della funzione.
handleEndGame	Funzione	Modifiche radicali per migliorare la gestione della modalità "Scalata". Aggiunta condizione di vittoria di una partita scalata, e degli alert a seconda della vittoria, sconfitta, passaggio al round successivo. Ad ogni alert è seguito l'inserimento di un timeout di 10 secondi che reindirizza in una pagina di default.

js/Util_Editor.js

startGame	Funzione async	Se modalita === "Scalata", recupera roundID, gameID, turnID, scalataID dalla risposta JSON e li salva in localStorage.
getScalataClasse	Funzione	Estrae la classe associata al round corrente dalla JSON della scalata.
getScalataRobot	Funzione	Estrae il robot associato al round corrente dalla JSON della scalata.
getScalataDifficulty	Funzione	Estrae e converte la difficoltà del round corrente.
GetDifficulty	Funzione	Mappa la difficoltà da Beginner, Intermediate, Advanced a valori numerici 1, 2, 3.
HandleScalataParameters	Funzione	Recupera e stampa i dati relativi alla scalata da localStorage.
HandleScalataNextRound	Funzione	Controlla se la scalata è terminata, Se la scalata è ancora in corso, aggiorna localStorage con i parametri del prossimo round, e Incrementa current_round_scalata e aggiorna il punteggio della scalata per la visualizzazione nel Frontend.

js/Var_Editor.js

var editor	Variabili globali	Aggiunte le variabili per gestire la modalità Scalata (ID, nome, round corrente, totale round, robot, difficoltà, classi, punteggio totale, stato della scalata).
-------------------	-------------------	---

js/gamemode_scalata.js

gamedatawriter	AJAX e gestione scalata	Aggiunta chiamata AJAX per ottenere la lista delle scalate e pulsante "Info" per identificare se la scalata è predefinita o personalizzabile.
PressedSubmit	Funzione	Rimossa la chiamata AJAX per salvare la scalata e aggiunto il reindirizzamento all'editor basato su ClassUT in localStorage.
RetrieveScalata	Funzione	Aggiunta per recuperare i dati di una scalata, salvandoli in localStorage (classi, robot, difficoltà, numero di round, ecc.).

T4:

Nome	Tipo	Modifiche
round/round.go		
Round	Struct	Aggiungi RobotID e Robot alla struttura.
CreateRequest	Struct	Aggiunta proprietà RobotID con validazione per obbligatorietà.
CreateRequest	Funzione	Controllo sul id del robot.
UpdateRequest	Struct	Aggiunto id del robot.
FromModel	Funzione	Modificata per includere RobotID dalla struttura del modello.
round/Service.go		
Create	Funzione	Ora assegna un solo robot al round, aggiungendo il campo RobotID e utilizzando l'ID del robot nel momento della creazione del round.
Update	Funzione	Aggiunta la modifica del metodo, passando da Updates(req) a Updates(r) e modificato il trattamento dell'errore.
FindById	Funzione	Aggiunta la pre-caricamento del campo Robot.
FindByGame	Funzione	Aggiunta la pre-caricamento del campo Robot.

T1 Frontend:

Nome	Tipo	Modifiche
js/Scalata.js		
handleRoundsChange, handleConfirm	Funzioni	<p>Modifica effettuata per impedire l'inserimento del numero del round da tastiera.</p> <p>Inoltre tutti i controlli sul nome della scalata e sul numero di round sono stati spostati sul tasto aggiungi in handleRoundsChange(), di conseguenza sono stati tolti da handleConfirm() che posizionava tali controlli solo alla fine della creazione della scalata (click tasto conferma).</p>
SelectClasses	Funzione	Aggiunti gestori di eventi per i menu a tendina delle card con i relativi controlli.
DisplayClasses	Funzione	Modifiche effettuate per mostrare le coperture da battere, robot e difficoltà.
submitScalataData	Funzione	Aggiunta la riabilitazione del tasto "aggiungi" e dei componenti della parte superiore della pagina come il nome, descrizione e numero di round.
ready	Funzione	Aggiunto il codice per modificare il cambiamento di modalità scalata, nel caso di scalata predefinita setta il robot selezionato dall'amministratore.
updateDifficultySelect	Funzione	Aggiunta per cambiare le difficoltà in base al robot selezionato, perché nel caso un robot abbia una difficoltà con coverage nulla non viene proprio mostrata all'amministratore.
\$(document).on('change', '.robot-select')	Event Listener	Quando cambia il robot selezionato viene chiamata updateDifficultySelect.
js/class.js		
EliminaScalata	Funzione	Nuova funzione per eliminare una scalata tramite la chiamata DELETE all' endpoint /delete_scalata/{scalataName}.
EliminaClasse	Funzione	<p>Aggiunto controllo sullo status 409 (classe usata in una scalata).</p> <p>Analizza il testo della risposta per estrarre i nomi delle scalate associate.</p> <p>Utilizza SweetAlert (swal) per mostrare un messaggio di conferma all'utente.</p> <p>Se l'utente conferma, elimina prima le scalate associate e poi la classe.</p>

js/scalata_tour.js

#modeRobotSelect	Elemento UI	Aggiunto nuovo step nel tour per spiegare la selezione tra scalata personalizzabile e predefinita.
localStorage.getItem('tourShown')	Controllo	Ora il tour viene mostrato solo la prima volta grazie all'uso di localStorage.
#summaryButton	ID	Corretto da #summarymButton a #summaryButton.

T1 Backend:

Nome	Tipo	Modifiche
controller/HomeController.java		
EliminaClasse	Metodo	Aggiunto il controllo per verificare se la classe è presente in una scalata prima di eleminarla.
service/AdminService.java		
UploadTest	Metodo	Ora passa l'oggetto ClassUT a RobotUtil.saveRobots invece del solo nome della classe. Aggiunta lista robotList con i robot disponibili (Randoop, Evosuite) e associata all'oggetto ClassUT. Aggiunta lista robotDifficultyList con i livelli di difficoltà (Beginner, Intermediate, Advanced) e associata all'oggetto ClassUT.
modificaClasse	Metodo	Aggiunti i campi robot, robotDifficulty e coverage all'Update della classe. Ora aggiorna anche le liste di robot e difficoltà oltre alle informazioni di base.
service/ScalataService.java		
FindScalataByClassName	Metodo	Aggiunto metodo per ritornare la lista delle scalate che contengono una data classe.
repository/ScalataRepository.java		
FindBySelectedClassesClassName	Metodo	Aggiunto nuovo metodo per trovare tutte le scalate che contengono una classe specifica.
model/ClassUT.java		
RobotList	Variabile	Aggiunta lista per memorizzare i tipi di robot associati alla classe.

RobotDifficulty	Variabile	Aggiunta lista per memorizzare i livelli di difficoltà dei robot.
Coverage	Variabile	Aggiunta lista per memorizzare la copertura dei test della classe.
ClassUT	Costruttore	Modificato per accettare i nuovi parametri robotList, robotDifficulty, e coverage.
getRobotList, setRobotList	Getter/ Setter	Permette di ottenere e impostare la lista dei robot associati alla classe.
getRobotDifficulty, setRobotDifficulty	Getter/ Setter	Permette di ottenere e impostare la lista dei livelli di difficoltà dei robot.
getCoverage, setCoverage	Getter/ Setter	Permette di ottenere e impostare la lista delle coperture di test della classe.
ToString	Metodo	Modificato per includere robotList, robotDifficulty e coverage nella stringa di output.
filesystem/RobotUtil.java		
SaveRobots	Metodo	Ora riceve un oggetto ClassUT invece del solo className, permettendo di aggiornare direttamente i dati della classe. Aggiunta gestione delle coperture (coverageList[]) per memorizzare i risultati dei test in MongoDB. Ora imposta le coperture dei test (setCoverage(...)) direttamente su classUT.

Appendice B: Riferimenti a scalata

Durante la prima fase di analisi del progetto, si è deciso di raccogliere in un file di note dove trovare file relativi alla scalata, in quali percorsi del progetto, e i vari punti di compilazione degli stessi.

In questo link sono stati inseriti la maggior parte dei riferimenti, sia diretti che indiretti, alla modalità scalata, utilizzato per mappare tutti i file coinvolti nella stessa. È probabile che classi non riportate possano essere state utilizzate. Al suo interno una piccola descrizione del comportamento di ciascun riferimento, che potrebbe non rispecchiare a verità.

La versione di riferimento è quella relativa al commit 604d2b6. Non è stata aggiornata con il proseguo delle modifiche, ma è servito per tracciare il suo comportamento e analizzare in quali punti andare a effettuare le modifiche.