Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Corso di Software Architecture Design Laurea Magistrale in Ingegneria Informatica

A.A. 2022-2023

TASK 4

Mantenimento delle Partite Giocate

Gruppo 15:

Matteo De Ieso – M63001561

Matteo Salvatore Pierno – M63001439

Antonio Russo – M63001460

Michele Vitagliano – M63001535

Sommario

Processo di Sviluppo

Per la realizzazione del prodotto software si è scelto di utilizzare il framework agile SCRUM.

Si è scelto di utilizzare un processo di sviluppo agile soprattutto per gestire al meglio gli eventuali e inevitabili cambiamenti architetturali e sotto il profilo dei requisiti, che un progetto di tale portata può comportare.

Durante la fase di pianificazione prevista dal framework, si sono stabilite le tecnologie da impiegare: il Product Backlog e gli obiettivi da realizzare per il primo Sprint. In totale si sono svolti quattro Sprint, ognuno dei quali della durata media di due settimane. Gli artefatti, da produrre e raffinare durante ciascuno Sprint, sono stati fissati in un'apposita seduta preliminare di Sprint Planning, che ha portato all'elaborazione dello Sprint Backlog per la particolare iterazione. Backlog che successivamente venivano condivisi con il Team utilizzando la piattaforma Trello.

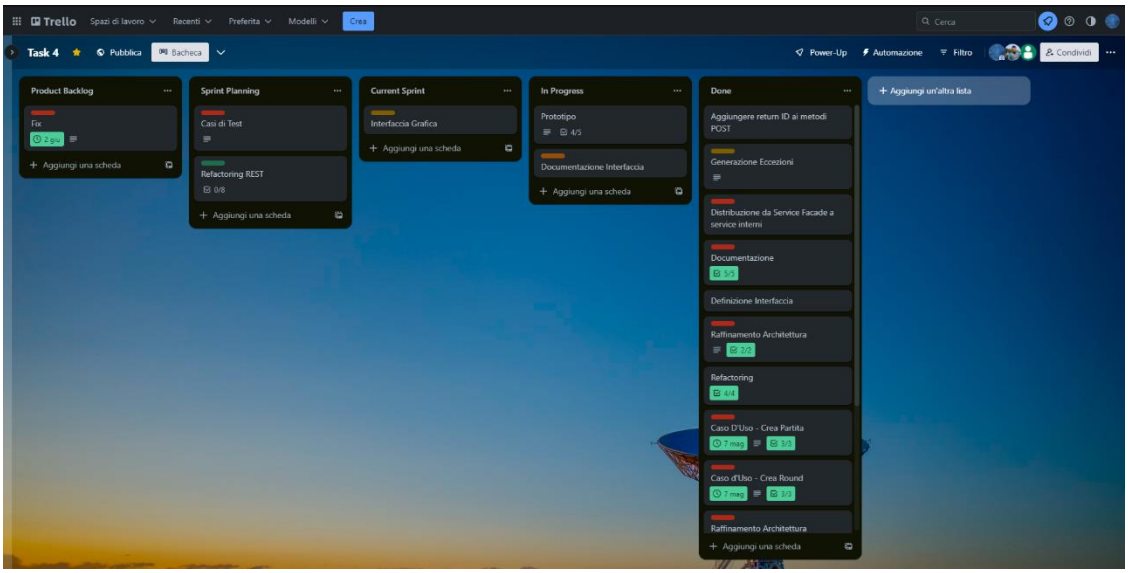


Figura 1 - Trello Board

Si è deciso di integrare al framework Scrum alcune pratiche tipiche di eXtreme Programming, come il *Refactoring*, il *Pair Programming* ed il *Collective Ownership*.

La pratica del Pair Programming si è rivelata particolarmente utile per ridurre il numero di *bug* durante lo sviluppo del primo prototipo eseguibile di *Games Repository*.

Durante ogni riunione, in presenza o tramite la piattaforma Microsoft Teams, si è tenuto un Daily Scrum per discutere brevemente il lavoro svolto e gli eventuali errori incontrati durante il periodo che intercorreva tra una riunione e l'altra.

Infine, è stato mantenuto un Diario di bordo per riassumere il lavoro fatto nel corso dei diversi Sprint.

Tecnologie Utilizzate

Per coordinare il lavoro dei diversi membri del Team si è fatto un uso intensivo delle piattaforme Microsoft Teams e Trello, inoltre è stata utilizzata una Github Repository per facilitare la produzione di codice in parallelo tra i diversi membri del Team, minimizzando i conflitti di scrittura.

Per la documentazione UML si è scelto di utilizzare il software Visual Paradigm Community Edition, mentre per lo sviluppo software è stato utilizzato JDK 17 (Ambiente di sviluppo per applicazioni Java) e in particolare Spring Boot.

Spring Boot è uno strumento che semplifica lo sviluppo di *web services* basati sul framework Spring, introducendo la possibilità di effettuare una configurazione automatica e personalizzabile per creare applicazioni completamente autonome anche attraverso l'uso di librerie *third-party*.

In particolare, insieme a Spring Boot, si è utilizzato l'ecosistema di librerie Hibernate, un framework open-source che si occupa di *Object Relational Mapping* in grado di *mappare* le entity del dominio su un database relazionale, attraverso l'uso delle specifiche JPA e le *JPA Repositories,* per raggiungere la persistenza dei dati.

Tutte queste dipendenze sono salvate in un pom.xml (Project Object Model) un file che definisce identità e struttura del progetto e che sarà consultato al momento dell'*application building*. Questo perchè, il framework Spring Boot prevede anche una funzione di *Dependency Injection*, cioè, consente ai prodotti software di definire le proprie dipendenze che poi il container Spring proverà ad inserire.

Per il Database si è utilizzato un database relazionale open-source chiamato PostgreSQL, acceduto attraverso l'interfaccia pgAdmin 4.

L'intero progetto è stato poi *buildato* in un file JAR utilizzando Maven: uno strumento di *build automation,* che sfruttando il pom.xml, citato in precedenza, riesce ad automatizzare il processo di *build* dell'applicazione. Il JAR è stato poi successivamente installato su un Docker Desktop in ambiente Windows.

Github: https://github.com/Testing-Game-SAD-2023/T4-G15

Infine, per il testing di unità si è utilizzato il framework *Mockito,* per realizzare delle classi *mock* e verificare il funzionamento dei vari casi d'uso implementati.

Analisi dei Requisiti

Task T4 Text Anlaysis

Requisiti sul mantenimento delle partite giocate

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

No.	Candidate Class	Type	Occurrence
1	Giocatore	Class	2
2	Casi di test creati	Class	2
3	Risultati	Class	2
4	Robot	Class	2
5	Il sistema deve mantenere lo storico di tale partita	Use Case	1
6	La classe testata	Class	1
7	Partita giocata	Class	1

Requisiti Funzionali

1. Il sistema deve garantire la persistenza delle partite giocate.
2. Il sistema deve fornire la possibilità di effettuare operazioni CRUD sulle partite giocate.
3. Il sistema deve fornire la possibilità di effettuare operazioni CRUD sui round giocati.

Requisiti sui Dati

1. La partita è caratterizzata da: tipo partita, id giocatore, data e ora di inizio e di fine, classe testata, insieme dei casi di test creati dai giocatori e dal robot, robot e risultati.
2. La partita è caratterizzata da più round di gioco.

Modello dei casi d'uso

In primo luogo, si è definito il *boundary* del sistema individuandone gli attori. Nel caso in esame, l'unico attore primario che interagisce direttamente con il sistema è il *Game Engine*.

Come si evince dalla Figura 2, inizialmente, il sistema prevedeva una più stretta interazione con i componenti di *Student Repository (T2-T3)* e *Test Run Enviroment (T7)*.

Tuttavia, durante lo sviluppo del prodotto software, l'interazione tra Student Repository e Game Repository si è ridotta a una semplice condivisione del database relazionale.

I casi d'uso CRUD individuati di Gestione Partita e Gestione Round si scompongono nei diversi *use-cases* descritti in seguito usando il livello di formalità *fully-dressed*.

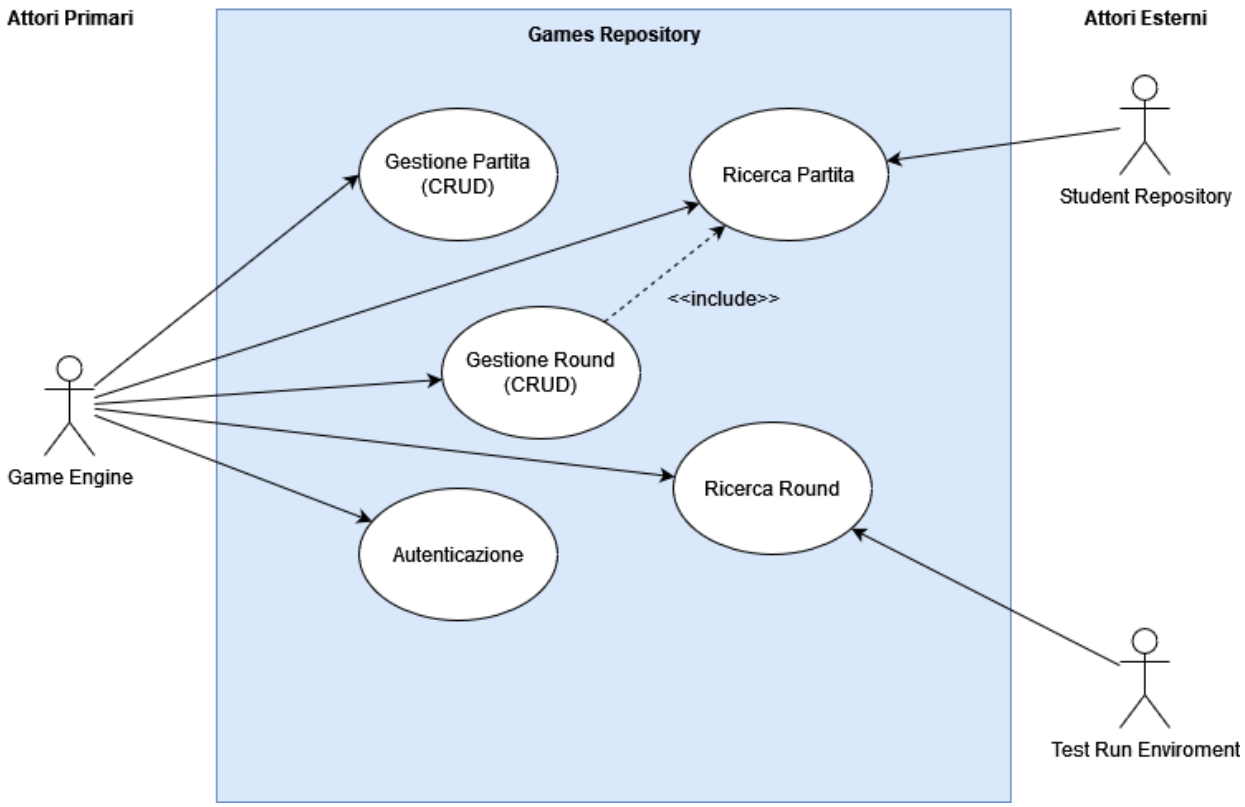


Figura 2 Use Case Diagram

Use Case ID 01: Crea Partita

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	Nessuna
Success Post Condition	Viene creato uno Storico Partita
Failed Post Condition	Non è stato possible creare uno Storico Partita

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole creare uno Storico Partita
2	Il Game Engine fornisce l'ID del Giocatore
3	Il Game Engine fornisce le informazioni relative allo Storico Partita da creare
4	Il Game Engine viene notificato che la creazione è andata a buon fine
5	Al Game Engine viene restituito l'ID dello Storico Partita appena creato

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi:
2a	Non è possibile creare lo Storico Partita ed il fallimento della creazione viene notificato al Game Engine;
3a	Lo Storico Partita viene creato con successo ma viene notificato al Game Engine la presenza di dati non validi;
	*b. Il Game Engine fornisce dati in input nulli:
3b	Lo Storico Partita viene creato con successo ma viene notificato al Game Engine la presenza di dati nulli;

Related Informations	
Frequency	Alta
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	Nessuno
Open Issues	1. In caso di partita multigiocatore viene creata un'istanza del caso d'uso per ogni giocatore partecipante o il Game Engine fornisce gli ID di tutti i giocatori alla stessa istanza? 2. I Dati da salvare all'interno dello Storico Partita potrebbero variare
Due Date	08/05/2023

Figura 3 Use Case Crea Partita

Use Case ID 02: Crea Round

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	Deve esistere almeno uno storico partita a cui assegnare il Round
Success Post Condition	Il nuovo Round e i dati ad esso associati sono salvati nello Storico Partita
Failed Post Condition	Non è stato possible salvare il nuovo Round nello Storico Partita

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole registrare un Round all'interno di uno Storico Partita
2	Il Game Engine fornisce l'ID dello Storico Partita in cui registrare il Round
3	Il Game Engine fornisce l'ID del Giocatore associato allo Storico Partita
4	Il Game Engine fornisce i risultati del Giocatore e del Robot
5	Il Game Engine viene notificato che il salvataggio è andato a buon fine
6	Al Game Engine viene restituito l'ID del Round appena salvato

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi:
2a	Non è possibile registrare il Round nello Storico Partita fornito ed il fallimento del salvataggio viene notificato al Game Engine;
3a	Non è possibile registrare il Round nello Storico Partita fornito ed il fallimento del salvataggio viene notificato al Game Engine;
4a	Il Round viene creato con successo ma viene notificato al Game Engine la presenza di dati non validi;
	*b. Il Game Engine fornisce dati in input nulli:

4b	Il Round viene creato con successo ma viene notificato al Game Engine la presenza di dati nulli;

Related Informations	
Frequency	Alta
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	ID 01: Crea Partita
Open Issues	1. In caso di partita multigiocatore viene creata un'istanza del caso d'uso per ogni giocatore partecipante o il Game Engine fornisce gli ID di tutti i giocatori alla stessa istanza? 2. I Dati da salvare all'interno del Round potrebbero variare
Due Date	08/05/2023

Figura 4 Use Case Crea Round

Use Case ID 03: Ricerca Partita

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	Nessuna
Success Post Condition	Viene restituita una Partita conclusa
Failed Post Condition	Non è stato possibile restituire una Partita

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole cerca una Partita
2	Il Game Engine fornisce l'ID della Partita
3	Al Game Engine viene restituita la Partita con ID corrispondente a quello dato in input

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi:
2a	Non è possibile trovare la Partita ed il fallimento della ricerca viene notificato al Game Engine;
	*b. Il Game Engine fornisce dati in input nulli:
3b	Non è possibile trovare la Partita ed il fallimento della ricerca viene notificato al Game Engine;

Related Informations

Frequency	Medio-Alta
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	Nessuno
Open Issues	Nessuna
Due Date	15/05/2023

Figura 5 Use Case Ricerca Partita

Use Case ID 06: Ricerca Round

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	Nessuna
Success Post Condition	Viene restituito il Round corretto
Failed Post Condition	Non è stato possibile trovare il Round

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole ricercare un Round
2	Il Game Engine fornisce l'ID della Partita contenente il Round
3	INCLUDE Use Case ID 05 Ricerca Partita
4	Il Game Engine fornisce l'ID del Round
5	Al Game Engine viene restituito il Round con ID corrispondente a quello dato in input

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi o nulli:
2a	Non è possibile trovare il Round ed il fallimento della ricerca viene notificato al Game Engine;

Related Informations

Frequency	Medio-Alta
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	Ricerca Partita
Open Issues	Nessuna
Due Date	16/05/2023

Figura 6 Use Case Ricerca Round

Use Case ID 04: Update Partita

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	1. Deve esistere almeno una Partita da modificare 2. L'Attore Primario deve conoscere l'ID della Partita da modificare
Success Post Condition	La Partita è modificata con successo
Failed Post Condition	Non è stato possibile modificare la Partita

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole modificare una Partita
2	Il Game Engine fornisce l'ID della Partita da modificare
3	Il Game Engine fornisce le informazioni da modificare nella Partita
4	Il Game Engine viene notificato che la modifica è andata a buon fine

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi:
2a	Non è possibile modificare la Partita ed il fallimento della modifica viene notificato al Game Engine;

Related Informations

Frequency	Alta
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	Nessuno
Open Issues	Nessuna
Due Date	15/05/2023

Figura 7 Use Case Update Partita

Use Case ID 05: Update Round

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	1. Deve esistere almeno un Round da modificare 2. L'Attore Primario deve conoscere l'ID della Partita e del Round da modificare 3. Deve esistere una Partita che contenga il Round da modificare
Success Post Condition	Il Round viene modificato con successo
Failed Post Condition	Non è stato possibile modificare il Round

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole modificare un Round
2	Il Game Engine fornisce l'ID della Partita da modificare
3	Il Game Engine fornisce l'ID del Round da modificare
4	Il Game Engine fornisce le informazioni da modificare nel Round
5	Il Game Engine viene notificato che la modifica è andata a buon fine

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi:
2a	Non è possibile modificare il Round ed il fallimento della modifica viene notificato al Game Engine;

Related Informations

Frequency	Alta
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	Nessuno
Open Issues	Nessuna
Due Date	16/05/2023

Figura 8 Use Case Update Round

Use Case ID 10: Delete Partita

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	1. Deve esistere almeno una Partita da eliminare 2. L'Attore Primario deve conoscere l'ID della Partita da eliminare
Success Post Condition	La Partita viene eliminata con successo
Failed Post Condition	Non è stato possibile eliminare la Partita

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole eliminare una Partita
2	Il Game Engine fornisce l'ID della Partita
3	Al Game Engine viene notificato che la cancellazione è avvenuta con successo

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi o nulli:
2a	Non è possibile eliminare la Partita ed il fallimento dell'eliminazione viene notificato al Game Engine;

Related Informations

Frequency	Medio
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	Nessuno
Open Issues	Nessuna
Due Date	16/05/2023

Figura 9 Use Case Delete Partita

Use Case ID 11: Delete Round

General Characteristics

Primary Actor	Game Engine
Secondary Actors	Nessuno
Preconditions	1. Deve esistere almeno un Round da eliminare 2. L'Attore Primario deve conoscere l'ID della Partita e del Round da eliminare
Success Post Condition	Il Round viene eliminato con successo
Failed Post Condition	Non è stato possibile eliminare il Round

Main Success Scenario

Step	Action
1	Il caso d'uso inizia quando il Game Engine vuole eliminare un Round
2	Il Game Engine fornisce l'ID della Partita
3	Il Game Engine fornisce l'ID del Round
4	Al Game Engine viene notificato che la cancellazione è avvenuta con successo

Extension Scenario

Step	Branching Action
	*a. Il Game Engine fornisce dati di input non validi o nulli:
2a	Non è possibile eliminare il Round ed il fallimento dell'eliminazione viene notificato al Game Engine;

Related Informations

Frequency	Medio
Concurrency	Potrebbero esserci chiamate concorrenti del caso d'uso
Superordinate Use Cases	Nessuno
Open Issues	Nessuna
Due Date	16/05/2023

Figura 10 Use Case Delete Round

Class Diagram di Prima Analisi

Vista la natura del Task 4 è stato necessario produrre un Diagramma delle classi per individuare i diversi dati che il sistema *Games Repository* dovrà gestire.

Questo primo artefatto si è successivamente rivelato inadeguato ed è stato soggetto, durante le diverse iterazioni, a numerose revisioni, fino a raggiungere un risultato soddisfacente condiviso da tutti i membri del Team, in fase di Design dell'architettura software.

In particolare:

- Robot, per restrizioni dovute ai requisiti dell'avvio dello scenario di gioco Task 5, non può essere associato a Test Case;
- La realizzazione dello scenario multigiocatore attraverso l'utilizzo di un Giocatore, nelle veci di proprietario dello storico partita, provocava eccessiva ridondanza all'interno del database relazionale nel momento in cui occorreva salvare la partita giocata da più giocatori.

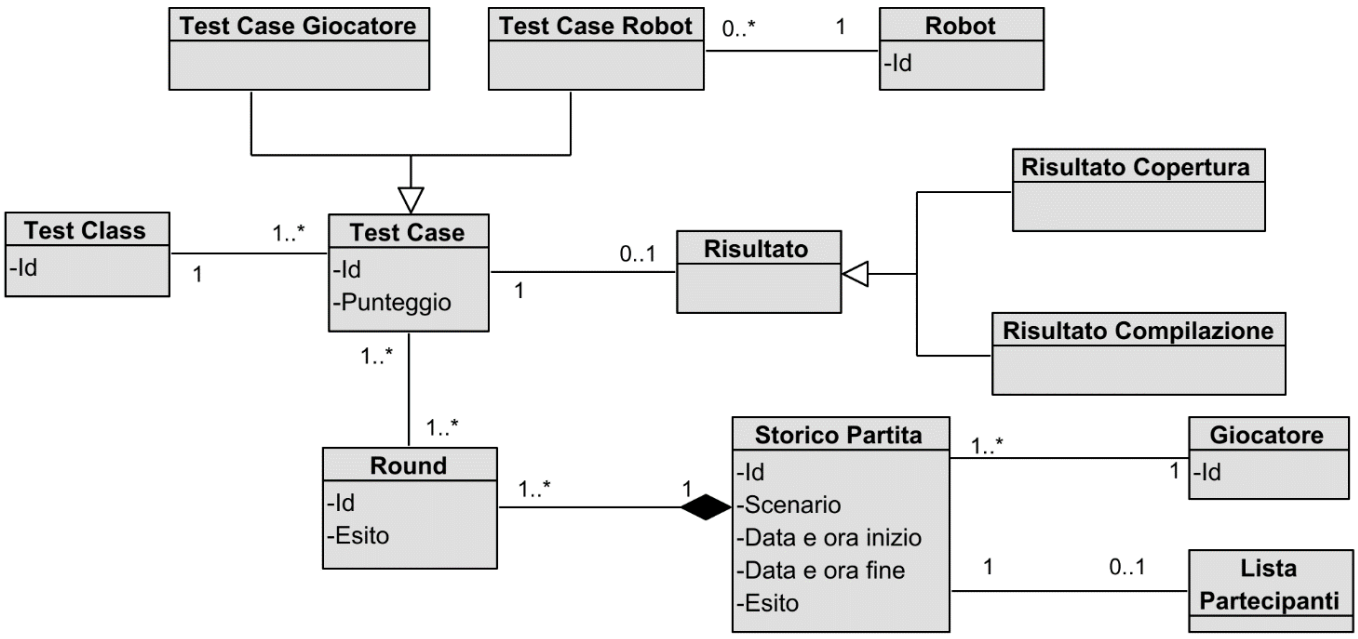


Figura 11 Class diagram di Analisi

Sequence Diagram di Analisi

Grazie alla specifica dei casi d'uso e ai legami esplicitati nel Class Diagram in Figura 11 è stato possibile produrre i Sequence Diagram relativi ai casi d'uso precedentemente descritti, permettendo una prima esplorazione delle possibili interazioni tra classi che occorrerà realizzare.

ID01: Crea Partita e Crea Round

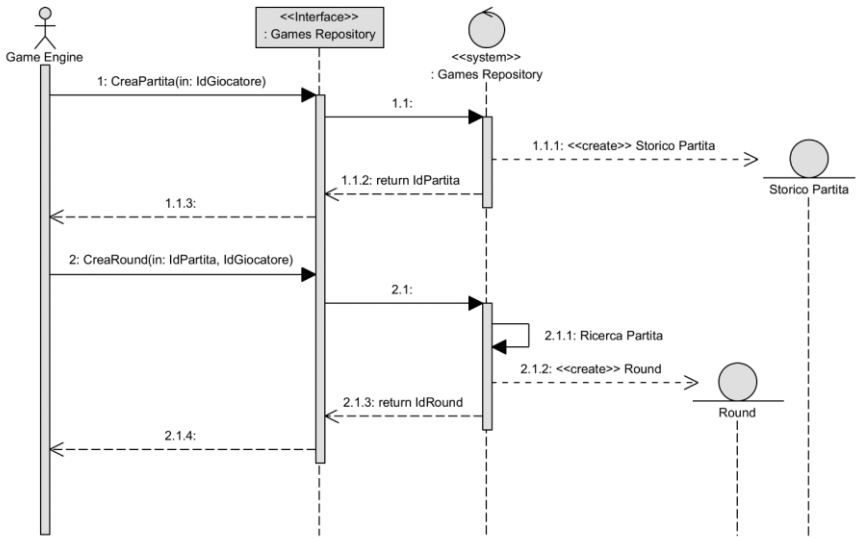


Figura 12 Sequence Diagram Crea Partita e Crea Round

ID02: Ricerca Partita

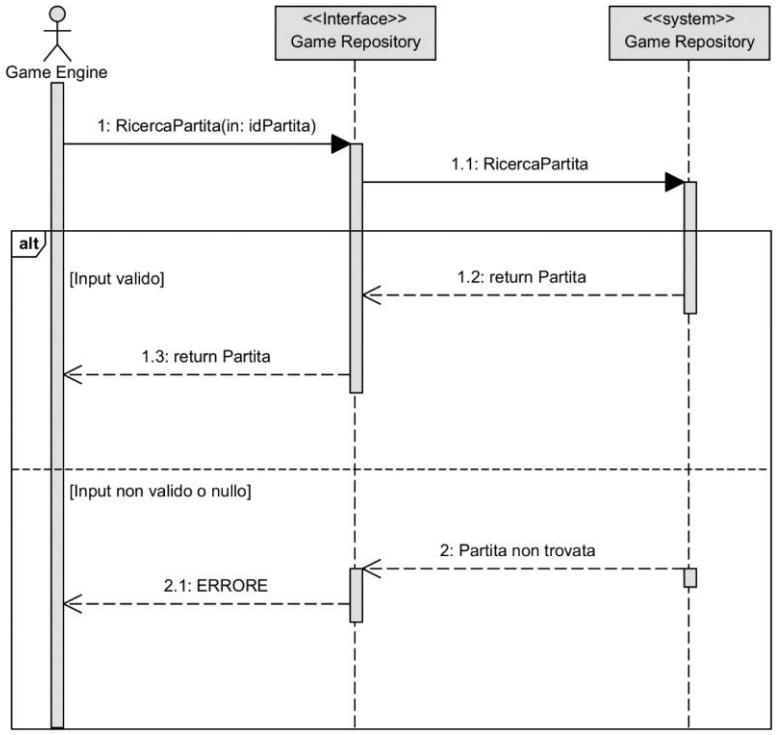


Figura 13 Sequence Diagram Ricerca Partita

ID03: Ricerca Round

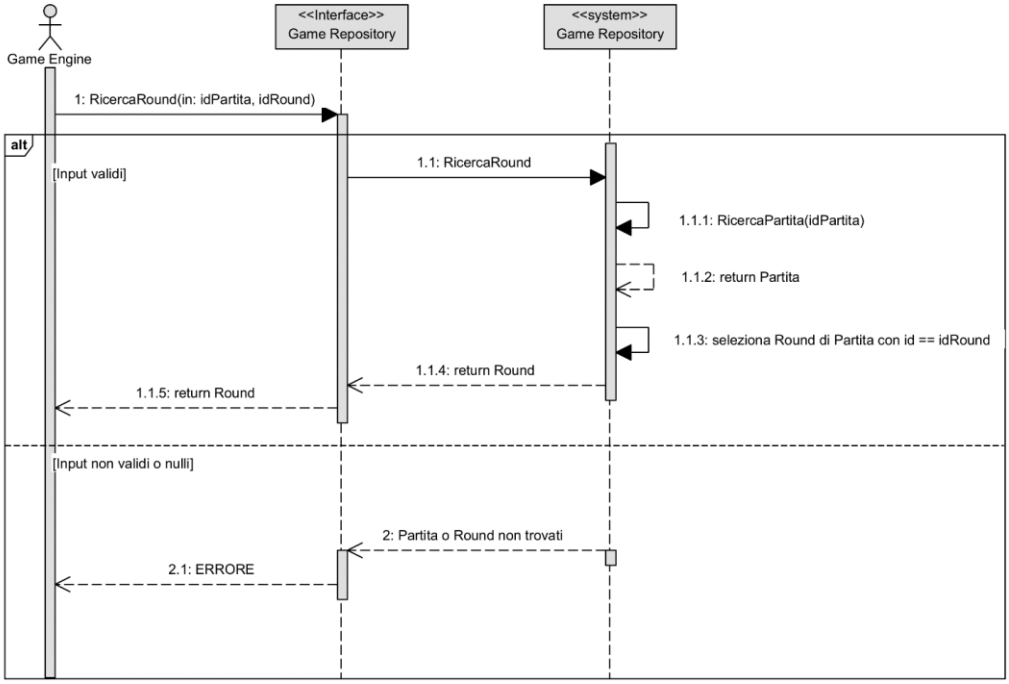


Figura 14 Sequence Diagram Ricerca Round

ID03: Update Partita

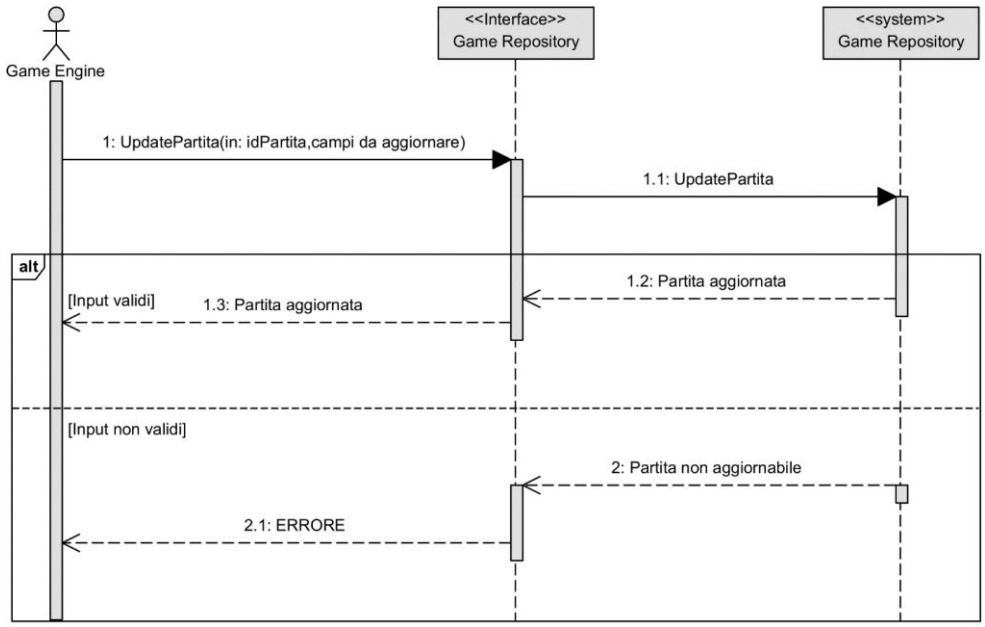


Figura 15 Sequence Diagram Update Partita

ID04: Update Round

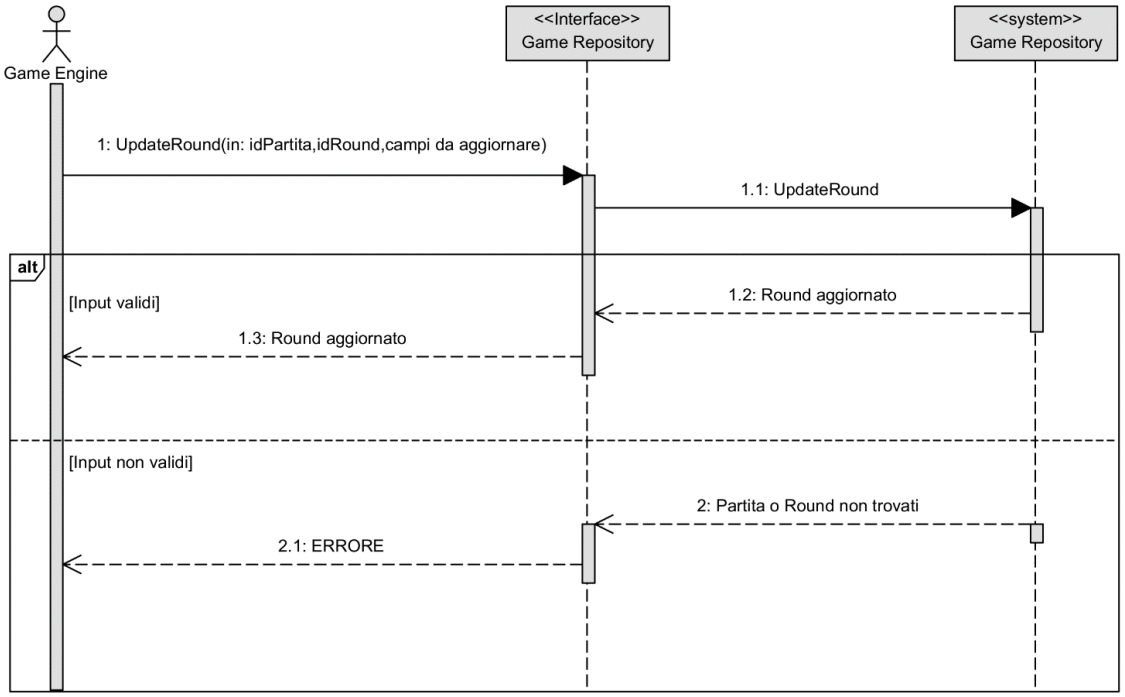


Figura 16 Sequence Diagram Update Round

ID05: Delete Partita

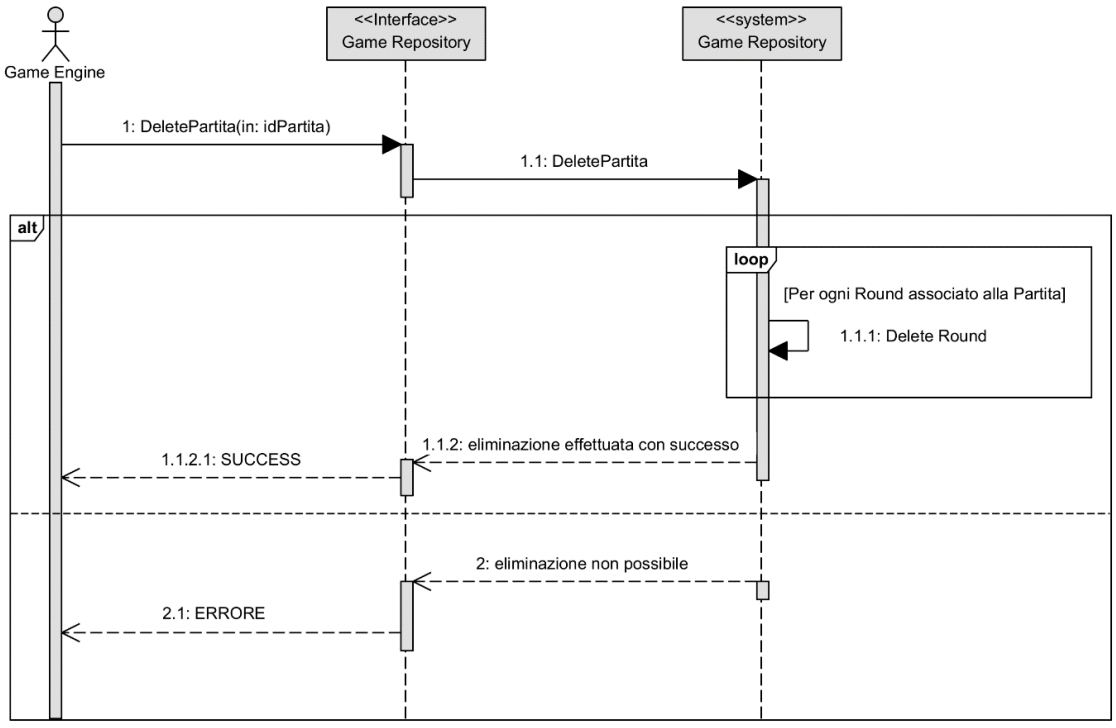


Figura 17 Sequence Diagram Delete Partita

ID06: Delete Partita

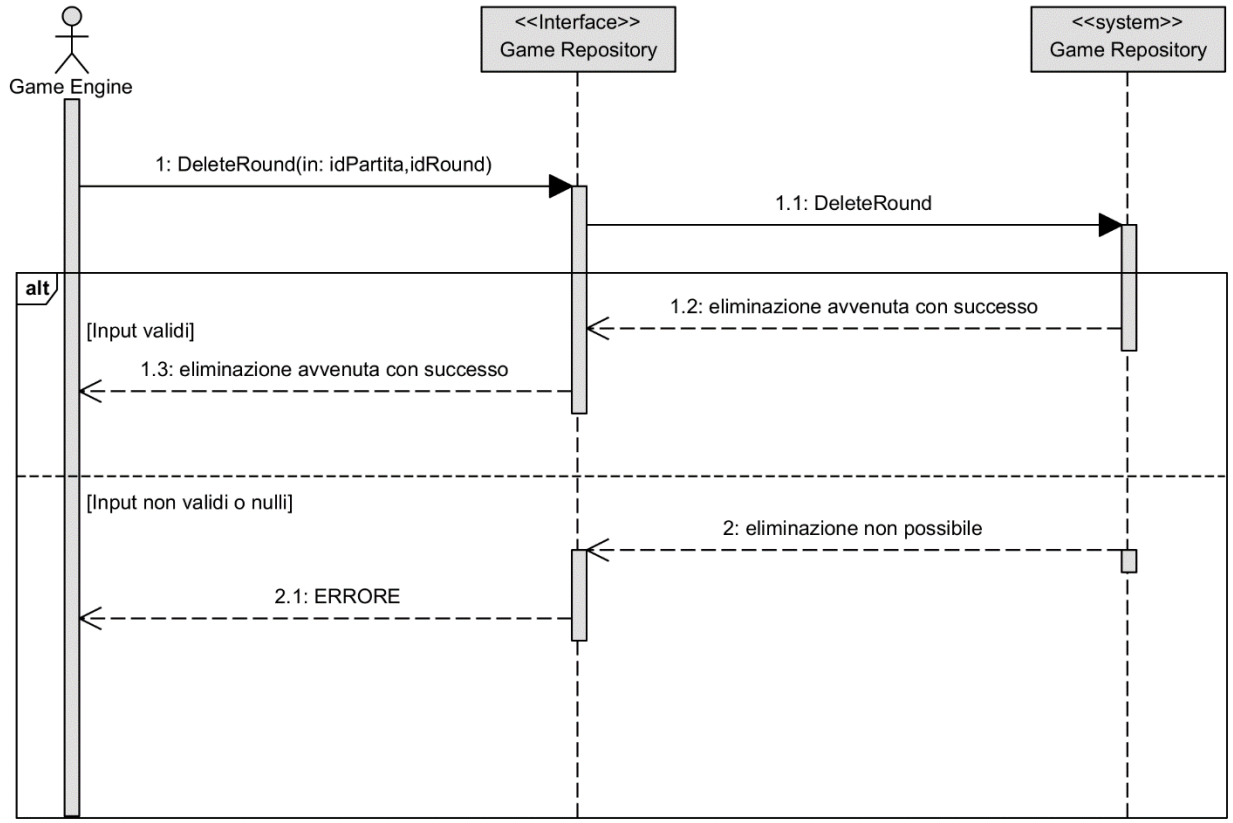


Figura 18 Sequence Diagram Delete Partita

Architettura e Software Design

L'architettura scelta è di tipo Client-Server dove il Tier Client (Game Engine) comunica con il Tier Server (Games Repository) attraverso un'API REST.

La scelta architetturale è stata guidata dall'esigenza di avere una maggiore flessibilità, riusabilità, indipendenza e astrazione dal resto dei servizi.

In particolare, il Tier Game Repository è un'architettura Multi-Layered CLA (Closed Layer Architecture) con tre strati software:

- Data Access Layer - Fornisce l'accesso ai dati e la permanenza di quest'ultimi.
- Service Layer - Racchiude l'implementazione dei Casi d'Uso implementati.
- Controller Layer - Implementa i metodi HTTP utilizzando i servizi REST.

L'architettura a layer presenta numerosi vantaggi, tra cui: alta modificabilità (Ogni layer può essere modificato in maniera indipendente dall'altro), testabilità facilitata, (Ogni layer può essere testato in maniera indipendente), agevole riutilizzazione dei layer (I "contratti" tra i layer sono definiti tramite interfacce) e virtualizzazione.

Static View: Package Diagram

Il package diagram seguente esplicita la struttura del prodotto software Games Repository.

Per facilitare la lettura e comprensione dei diagrammi architetturali, si sfrutta una colorazione specifica per i diversi layer e quanto contenuto al loro interno.

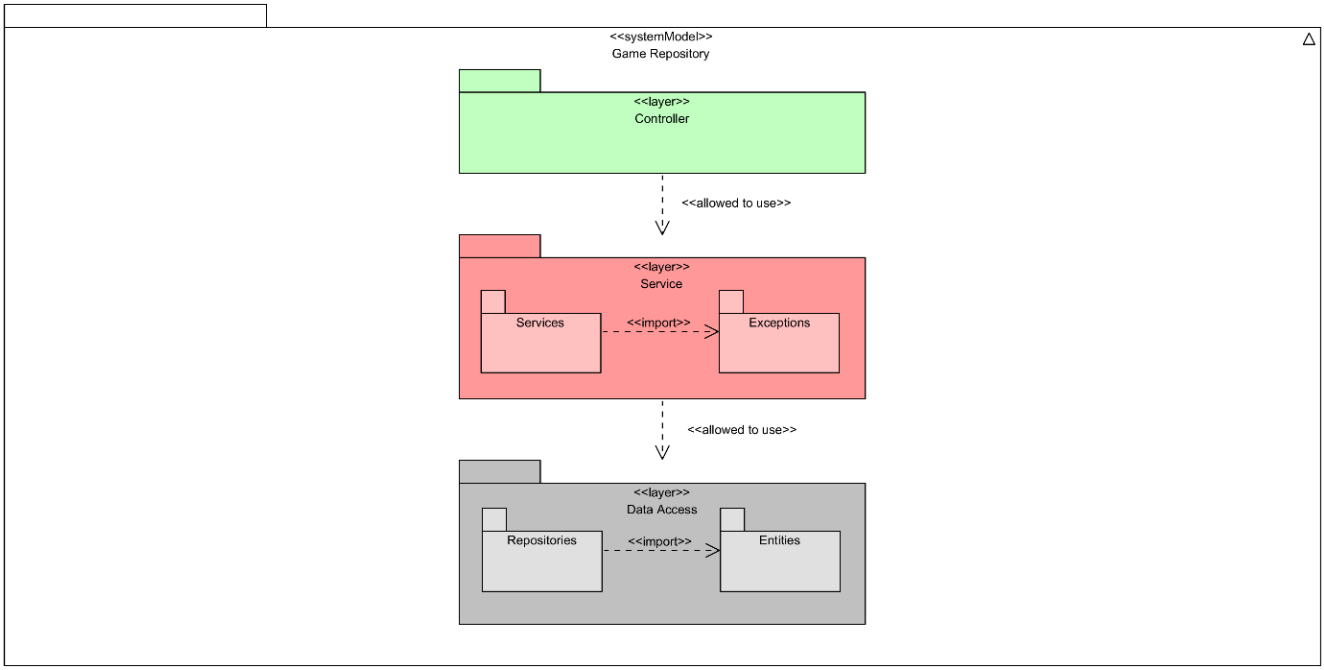


Figura 19 - Package Diagram

Controller Layer

All'interno del package *Controller* sono presenti le seguenti classi:

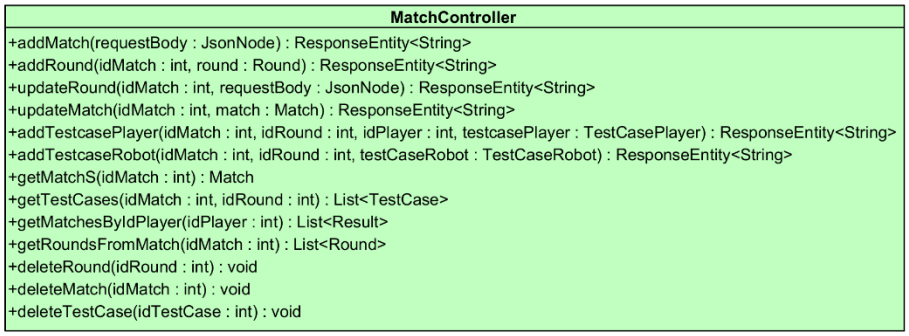


Figura 20 - Match Controller

Service Layer

All'interno del package *Services* sono presenti le seguenti classi:

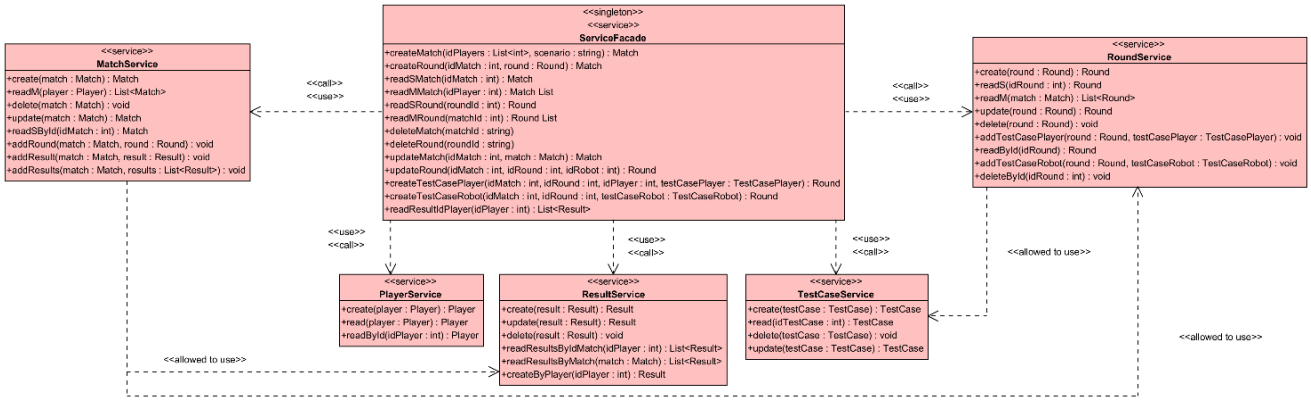


Figura 21 - Services Package

Data Access Layer

All'interno del package *Repositories* sono presenti le seguenti classi:

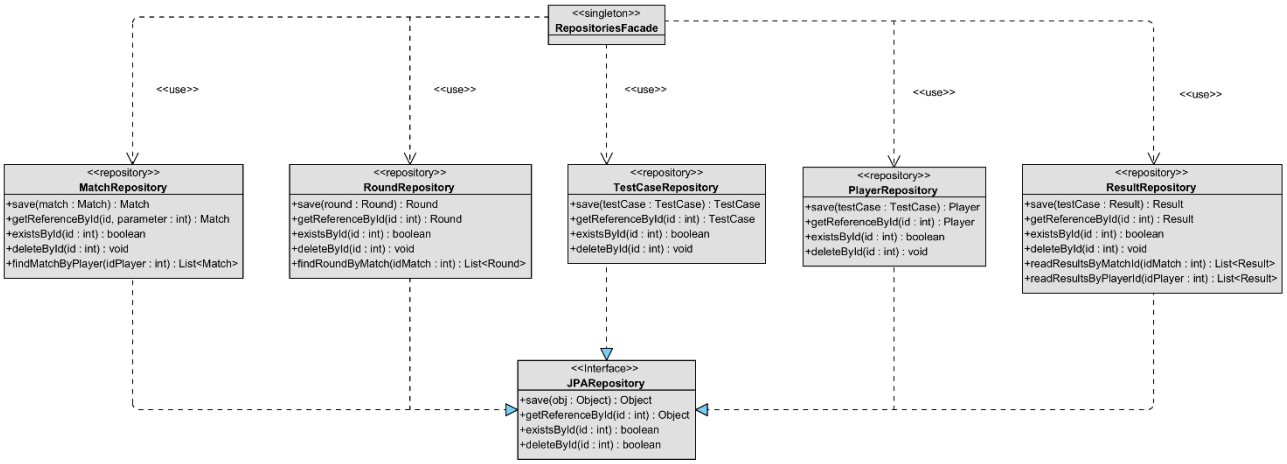


Figura 22 - Repositories Package

All'interno del package *Entities* sono presenti le seguenti classi:

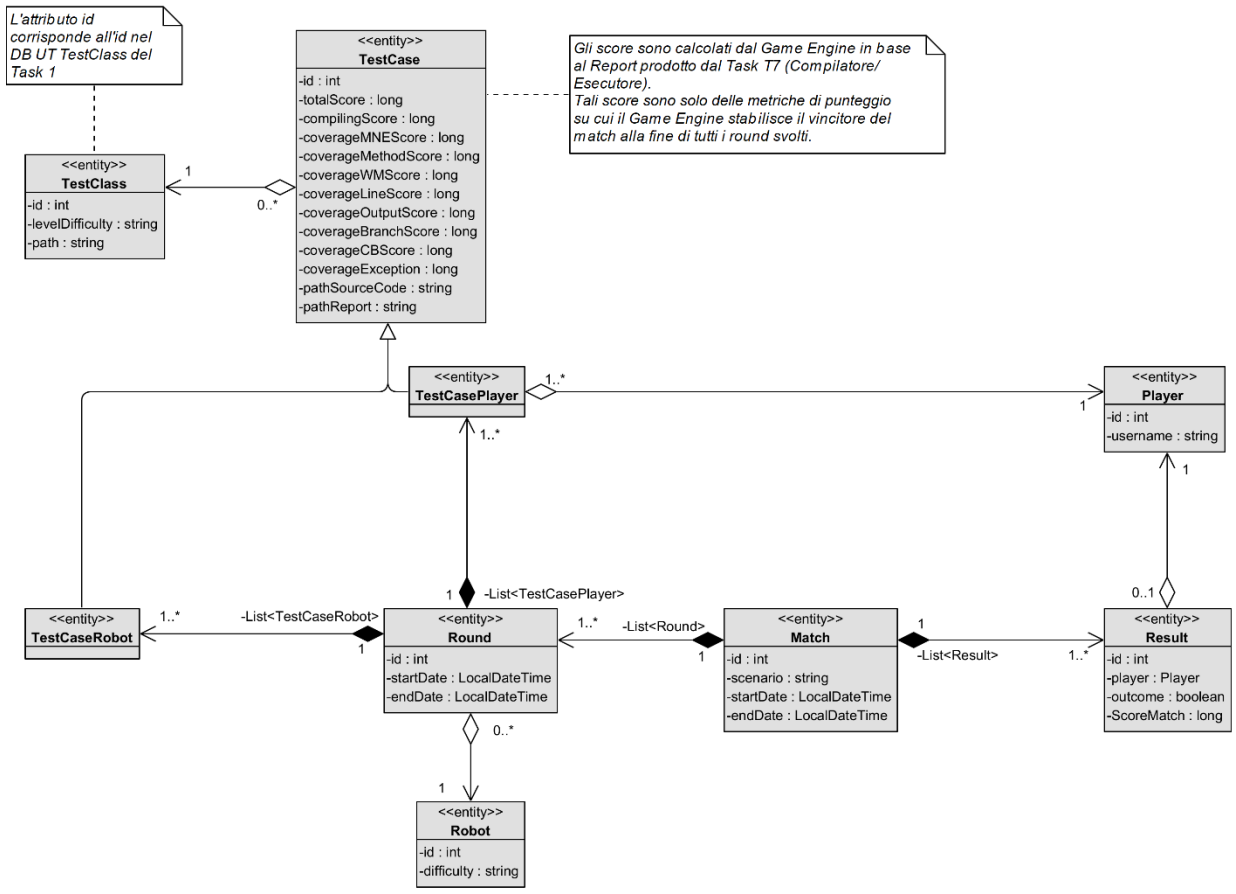


Figura 23 - Entities Package

Component Diagram

Di seguito, il *component diagram* mostra l'interfaccia esposta dal Game Repository e come potrebbe essere utilizzata dal resto dei componenti dell'architettura Test Game. Inoltre, sono anche rappresentate le diverse interfacce ("contratti") utilizzate dai layer per scambiarsi "messaggi".

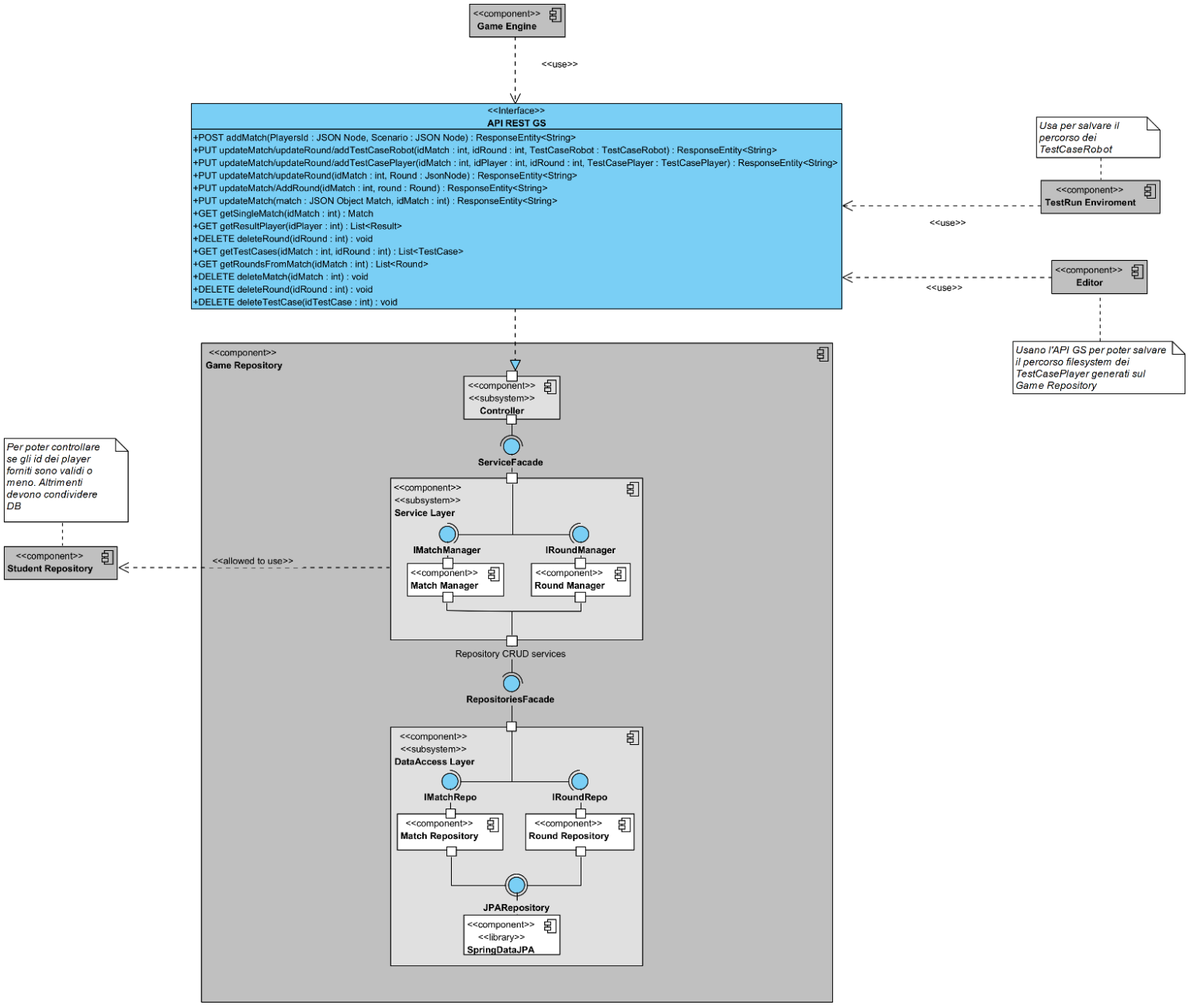


Figura 24 - Component Diagram

Interaction View: Communication Diagram

Attraverso il *communication diagram* si è ipotizzato un eventuale flusso di chiamate REST da parte dei diversi Task (T4, T5, T6 e T7 corrispettivamente *Game Repository*, *Game Engine*, *Editor* e *TestRun Env*. Nel diagramma), al fine di simulare il numero di operazioni necessarie, durante lo svolgimento di una partita, a mantenere un livello accettabile di persistenza dei dati.

In sintesi, il Game Engine, avviando lo scenario di gioco, Crea un Match e, per ogni giocatore partecipante, crea un Result con outcome per il momento nullo. Il Game Engine è in grado, durante lo svolgimento della partita, di Creare Round e Allocare TestCase, che saranno poi aggiornati con i relativi *score* e *path* di salvataggio da parte dei componenti TestRun, Editor oppure dal Game Engine stesso. Alla fine della partita il Game Engine calcolerà in base agli *score* il vincitore e aggiornerà l'outcome in Result di quel determinato Player vincente.

Il *path* di salvataggio è il percorso del FileSystem dove i TestCase.java sia dei giocatori che dei robot e i relativi report sono salvati, per un eventuale consultazione *offline.*

I Sequence successivi al Communication Diagram invece, esprimono in maniera molto dettagliata le interazioni tra classi e layer all'interno del sistema Games repository. Osservarli tenendo bene in mente il Package Diagram precedente (nel caso non dovessero essere visibili per via della bassa risoluzione, nel Github sono presenti delle versioni .svg ad alta risoluzione).

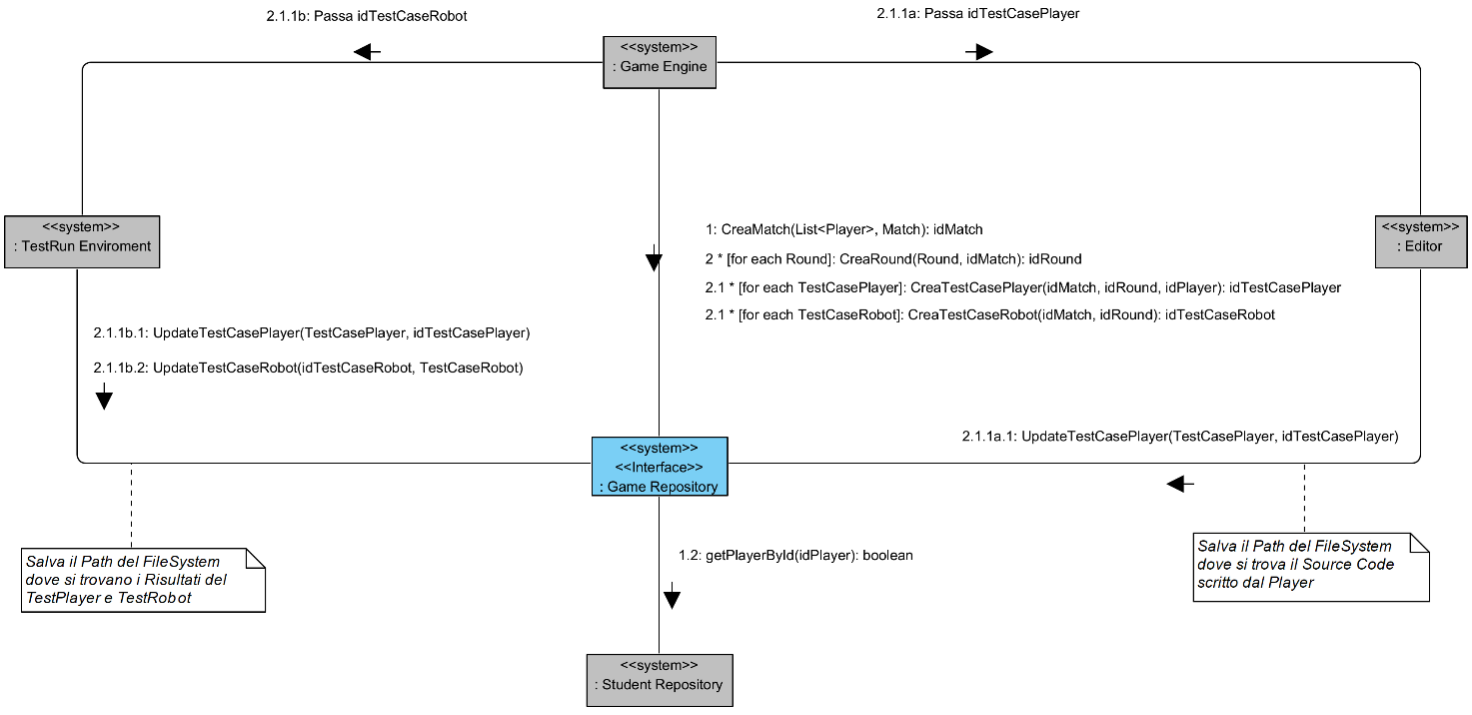


Figura 25 - Communication Diagram

Design Sequence Diagrams

ID01: Crea Partita

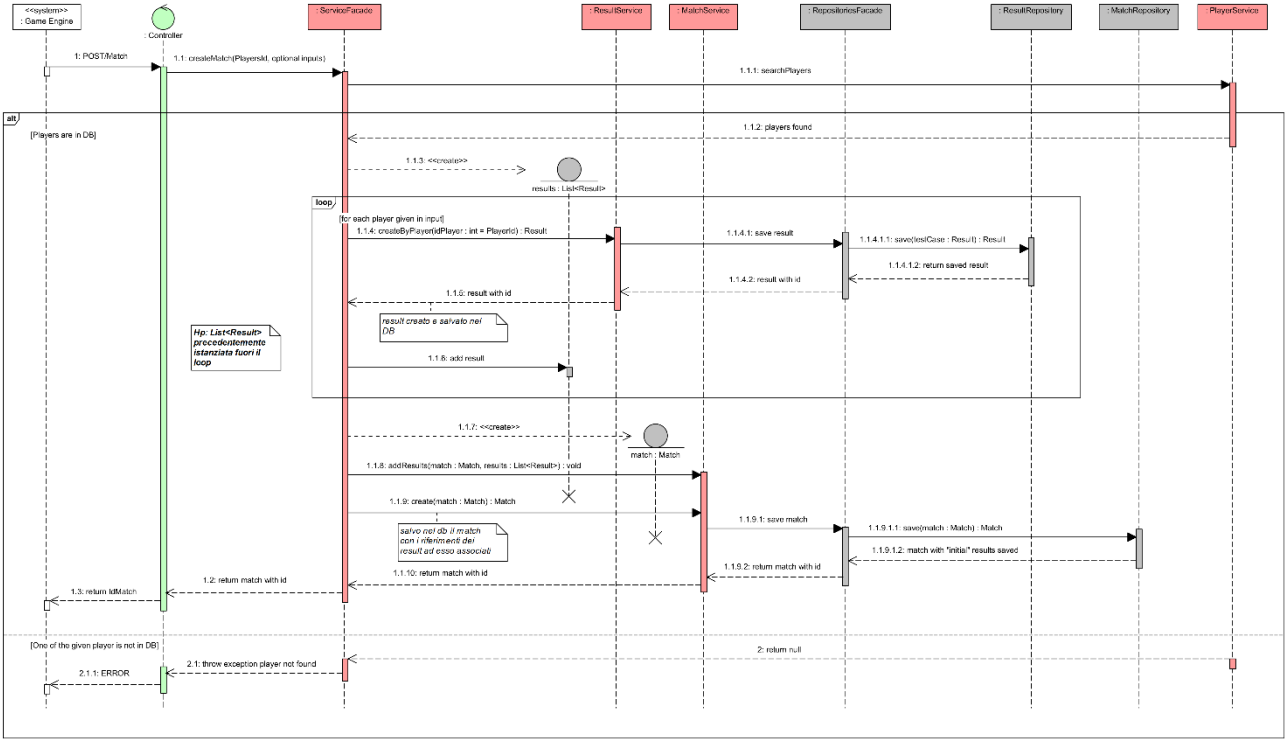


Figura 26 - SD Crea Partita

ID02: Crea Round

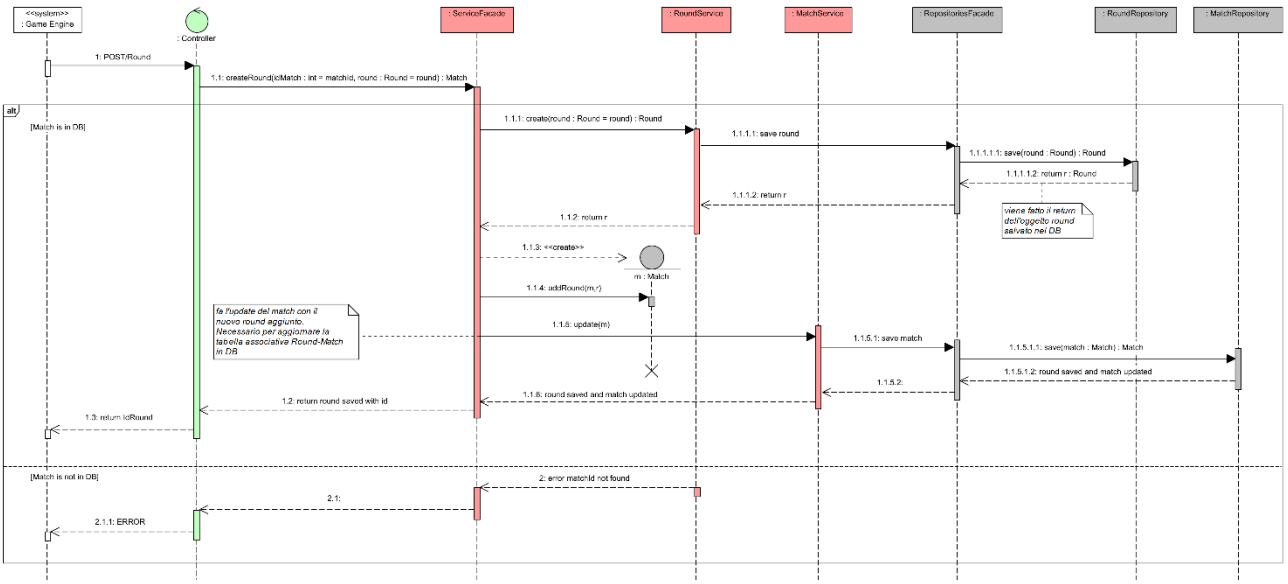


Figura 27 - SD Crea Round

ID03: Ricerca Partita

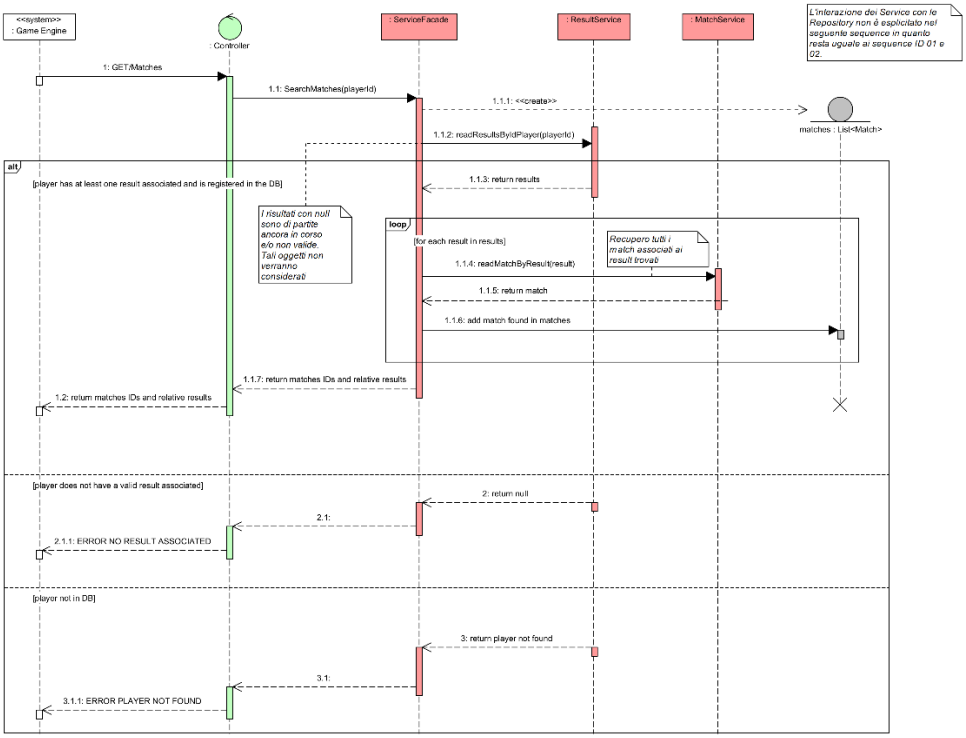


Figura 28 - SD Ricerca Partita

ID04: Ricerca Round

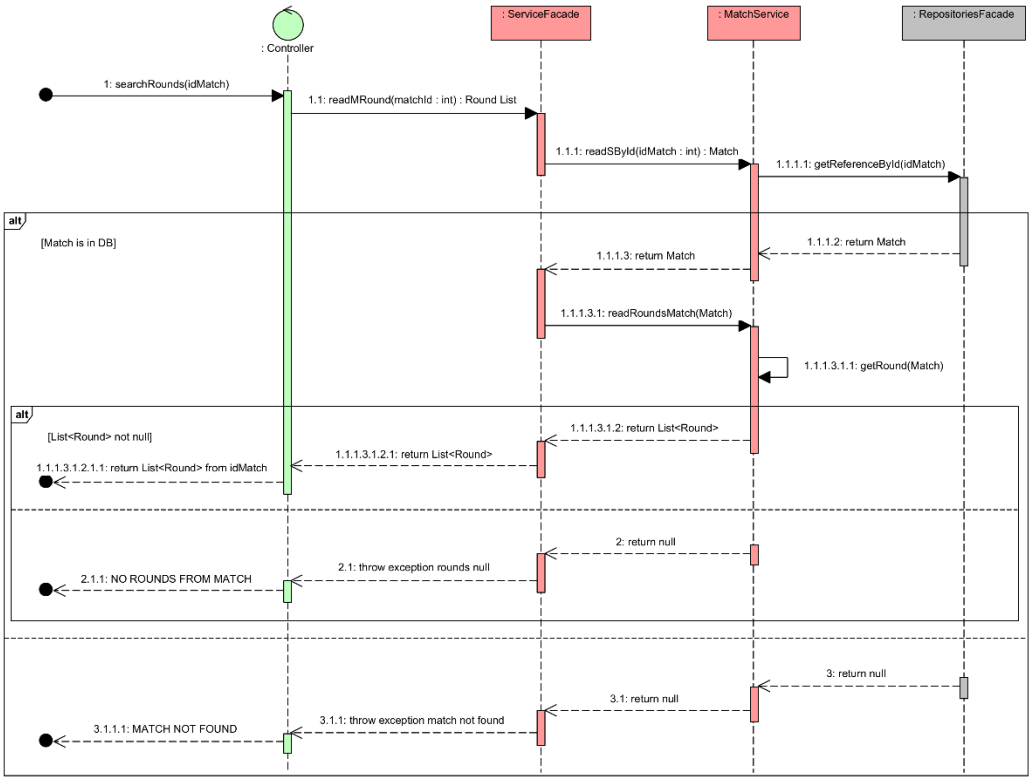


Figura 29 - SD Ricerca Round

ID05: Update Partita

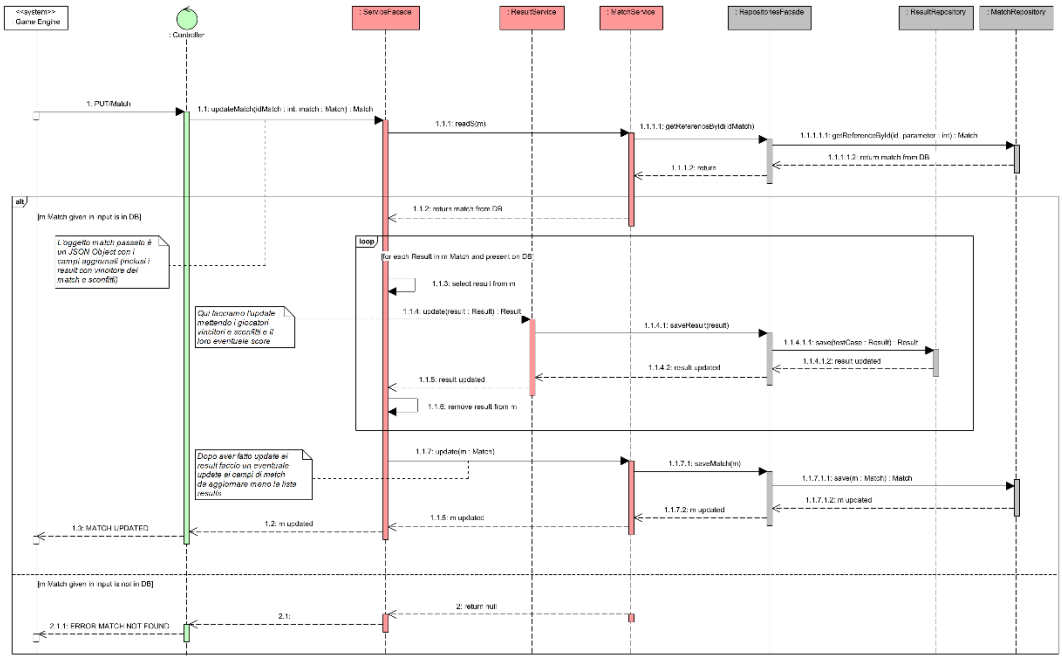


Figura 30 - SD Update Partita

ID06: Update Round

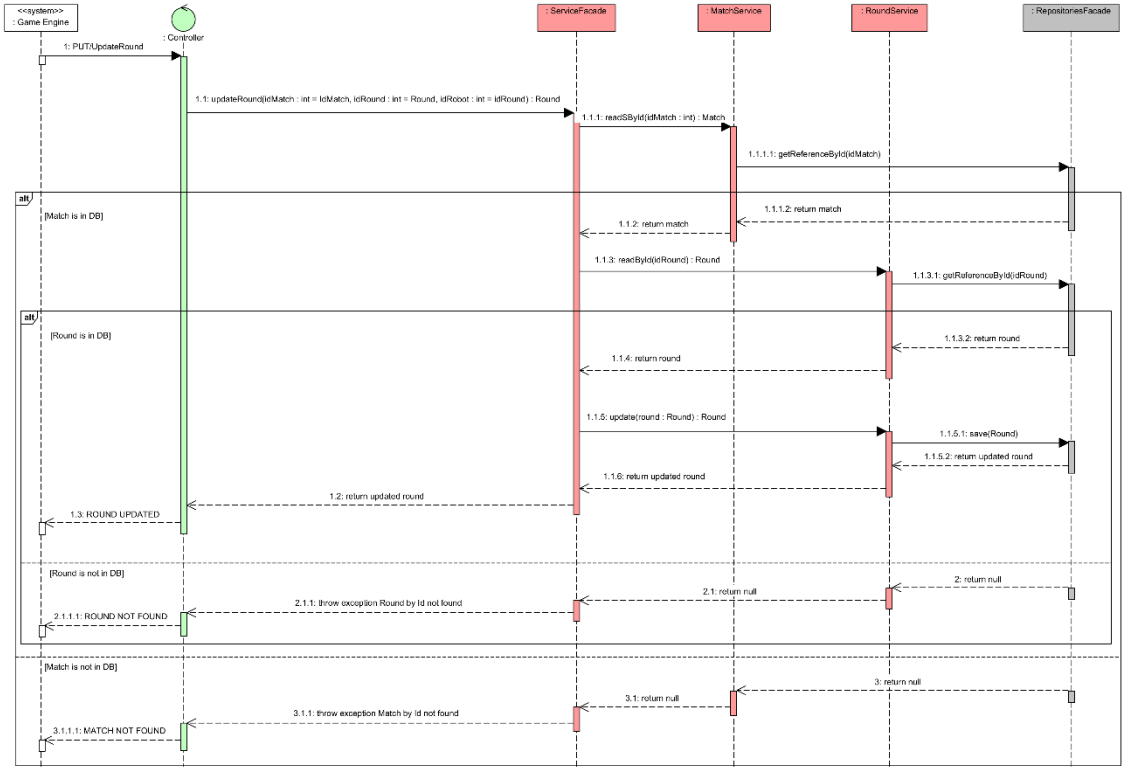


Figura 31 - SD Update Round

ID07: Crea TestCase

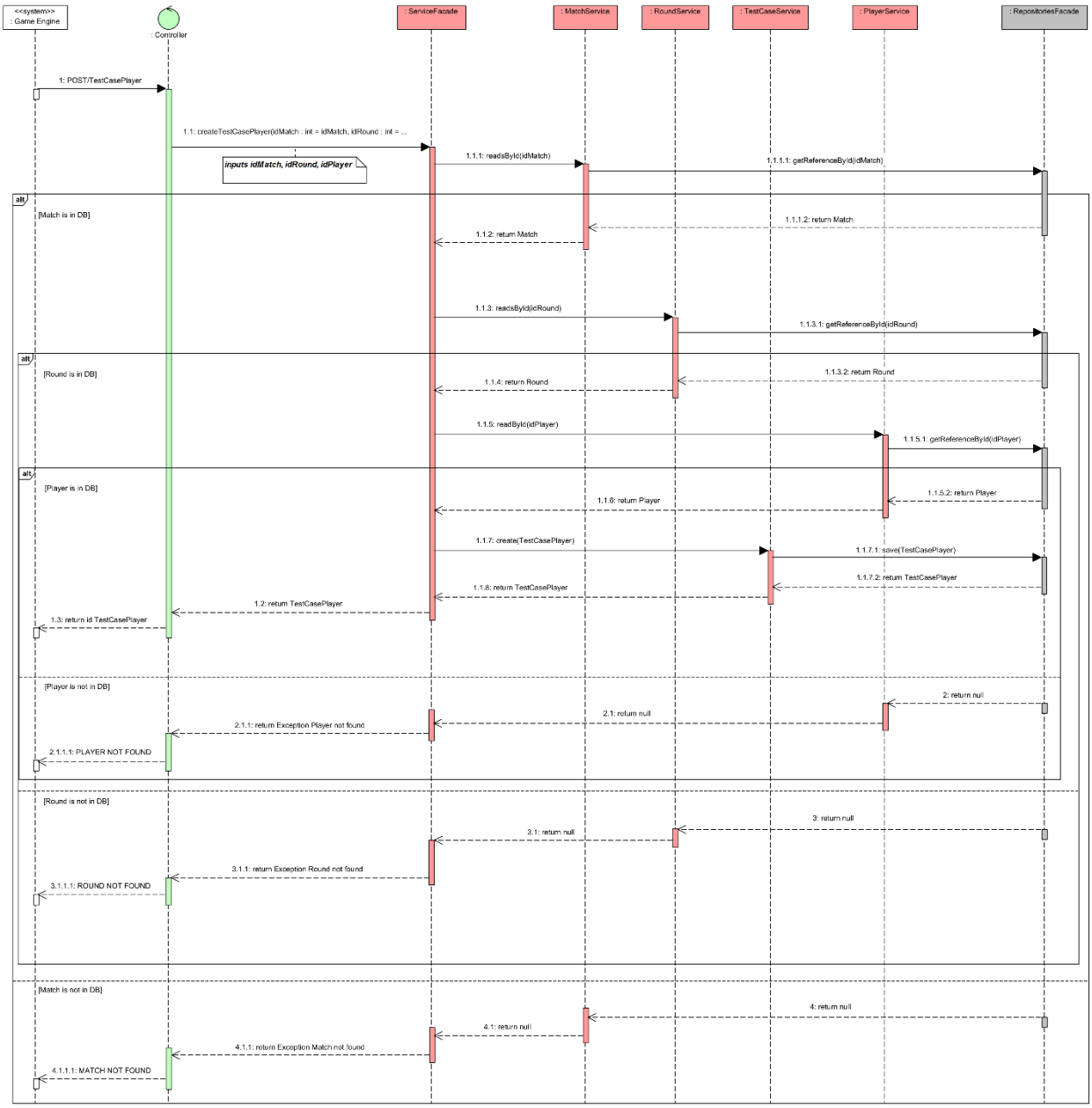


Figura 32 - SD Crea TestCase

ID08: Ricerca TestCase

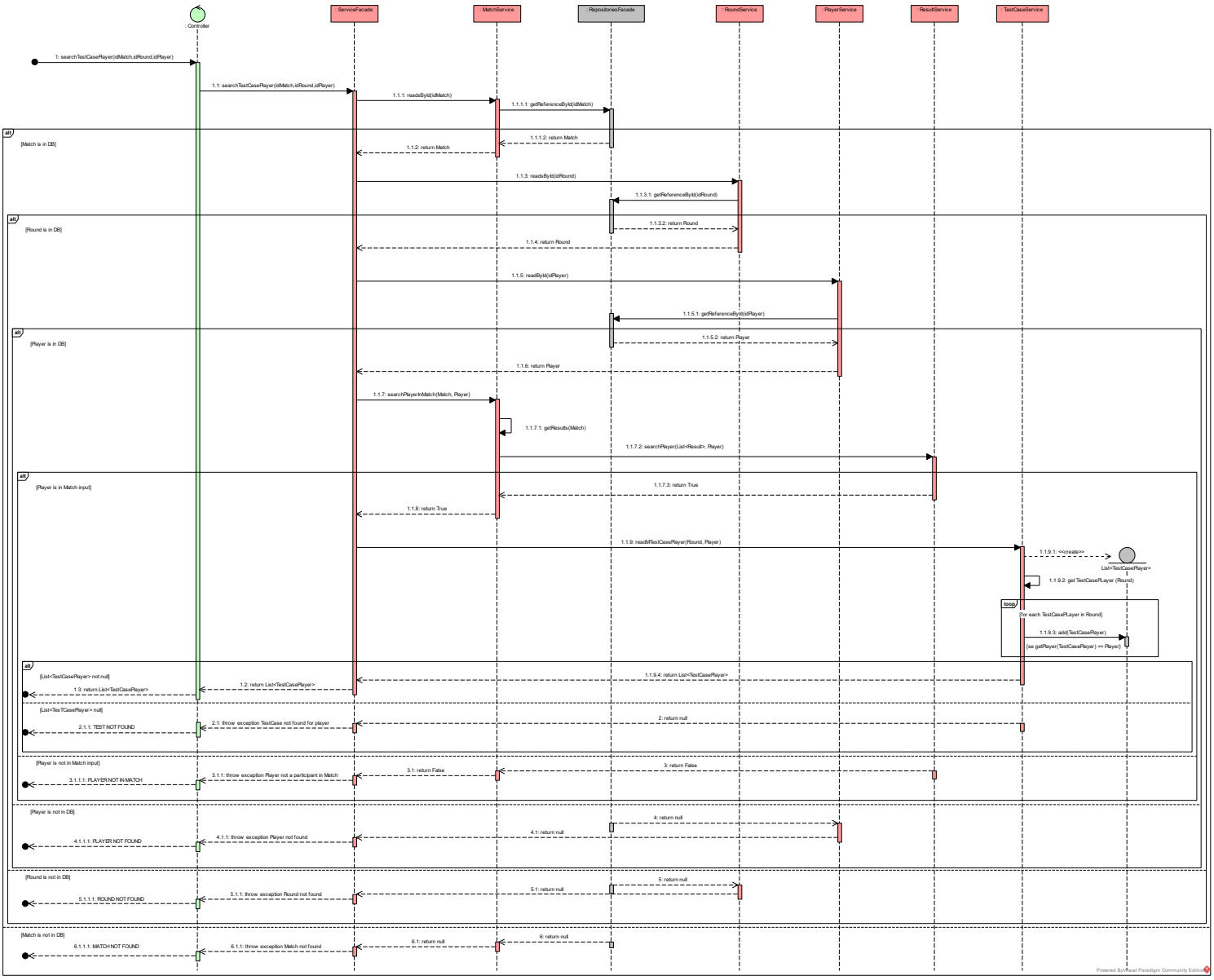


Figura 33 - SD Ricerca TestCase

ID09: Update TestCase

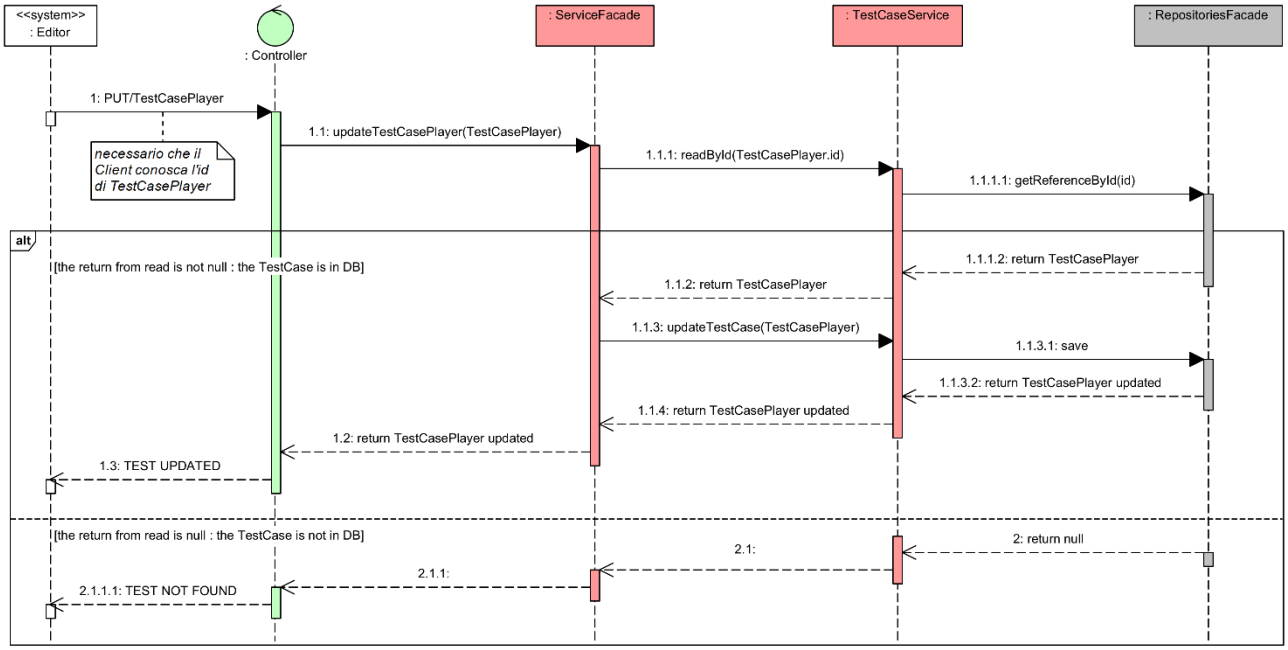


Figura 34 - SD Update TestCase

Legenda Interfaccia

I metodi esposti dall'interfaccia REST adottata sono elencati in seguito, esplicitando i relativi URI, parametri ed eccezioni, lanciate in caso di errore.

INTERFACCIA CRUD SERVICES GAME REPOSITORY				
Funzione	**Parametri**	**Info Parametri**	**Eccezioni**	**URI**
AddMatch	P1: JsonNode	Value Of IdPlayers and Values Of Match	E1: idPlayer not found	/addMatch
AddRound	P1: JsonObject	Value Of Round	E1: idMatch not found E2: idRobot not found E3: idRobot not specified	/updateMatch/{idMatch}/ addRound
UpdateRound	P1: JsonNode	Value Of IdRound Value Of EndDate	E1: idMatch not found E2: idRound not found	/updateMatch/{idMatch}/ updateRound
UpdateMatch	P1: Integer P2: JsonObject	Value Of IdMatch Value Of Match	E1: idMatch not found	/updateMatch/{idMatch}
AddTestCasePlayer	P1: Integer P2: Integer P3: Integer P4: JsonObject	Value Of IdMatch Value Of IdPlayer Value Of IdRound Value Of TestCasePlayer	E1: idMatch not found E2: idRound not found E3: idPlayer not found	/updateMatch/{idMatch}/ updateRound/{idRound}/ addTestCasePlayer/{idPlayer}
AddTestCaseRobot	P1: Integer P2: Integer P3: JsonObject	Value Of IdMatch Value Of IdRound Value Of TestCaseRobot	E1: idMatch not found E2: idRound not found	/updateMatch/{idMatch}/ updateRound/{idRound}/ addTestCaseRobot
ReadResultIdPlayer	P1: Integer P2: Output	Value Of IdPlayer Return List Match	E1: idPlayer not found E2: idResult not found	/getSingleMatch/{idMatch} /getTestCasesByRound/ {idRound}
ReadSMatch	P1: Integer P2: Output	Value Of IdMatch Return Match	E1: idMatch not found	/getSingleMatch/{idMatch}
ReadMRounds	P1: Integer P2: Output	Value Of IdMatch Return List Round	E1: idMatch not found	/getSingleMatch/{idMatch}/ rounds
ReadMTestCases	P1: Integer P2: Output	Value Of IdRound	E1: idRound not found	/getSingleMatch/{idMatch}/ getTestCasesByRound/{idRound}

		Return List TestCase	E2: idMatch not found E3: TestCase not found	
DeleteMatch	P1: Integer	Value Of IdMatch	E1: idMatch not found	/deleteMatch/{idMatch}
DeleteRound	P1: Integer	Value Of IdRound	E1: idRound not found	/deleteRound/{idRound}
DeleteTestCase	P1: Integer	Value Of IdTestCase	E1: idTestCase not found	/deleteTestCase/{idTestCase}

LOCALLY DEFINED DATA TYPES			
Funzione	**Parametri**	**Definizione Parametri**	**Esempio**

AddMatch	JsonNode	Nel JsonNode deve essere specificata una lista di Studenti e uno Scenario	{
			"idStudents": [value1, value2],
			"scenario": "exampleScenario",
			"idRobot": 1
			}

AddRound	JsonObject	Nel JsonObject deve essere specificato un JsonObject della classe Round	{
			"Robot": {
			"id": 1
			}
			optional,
			}

UpdateRound	JsonNode	Nel JsonNode deve essere specificato un id di una classe Round e un endDate in formato LocalDateTime	{
			"idRound": 1
			"end_date": "2023-06-02T21:00:00
			}

UpdateMatch	Integer	Common Type	

	JsonObject	Deve essere specificato un JsonObject della classe Match	{
			"id": 1,
			"scenario": "scenario",
			"endDate": "2023-06-02T21:00:00",
			"results": [
			{
			"id": 1,
			"result": "sconfitta"
			}
			{
			"id": 2,
			"result": "vittoria"
			}
			]
			}

AddTestCasePlayer	Integer	Common Type	
	JsonObject	Deve essere specificato un JsonObject della classe TestCasePlayer	{ "totalResult" : 12568, "compilingResult" : 1212, ecc...

		con tutti i campi, meno id e Player, not null	}
AddTestCaseRobot	Integer	Common Type	

	JsonObject	Deve essere specificato un JsonObject della classe TestCasePlayer con tutti i campi, meno id, not null	{
			"totalResult" : 12568,
			"compilingResult" : 1212,
			ecc...
			}

ReadResultById	Integer	Common Type	

	Output	Viene restituita una lista di JsonObject Result con l'idMatch specificato	[
			{
			"idResult": 1,
			"idMatch": 3,
			"Outcome": "vittoria"
			}
			{
			"idResult": 2,
			"idMatch": 6,
			"Outcome": "sconfitta"
			}
			ecc
			]

ReadSMatch	Integer	CommonType	

	Output	Viene restituito un JsonObject Match	{
			"id": 1,
			"scenario": "scenario",
			"endDate": "2023-06-02T21:00:00",
			"results": [
			{
			"id": 1,
			"result": "sconfitta"
			}
			{
			"id": 2,
			"result": "vittoria"
			}
			]
			}

ReadRound	Integer	CommonType	

	Output	Viene restituito un JsonObject Round	{
			"idRobot":value1,
			optional,
			}

ReadMRounds	Integer	Common Type	

	Output	Viene restituita una lista di JsonObject Rounds	{
			"id": 1,
			"idRobot":value1,
			optional
			}
			{
			"idRound: 2,
			"idRobot":value2,
			optional
			}

ReadTestCase	Integer	Common Type	

	Output	Viene restituito un JsonObject TestCase	{
			“id”: 1,
			“idPlayer”: 4,
			"totalResult" : 12568,
			"compilingResult" : 1212,
			ecc...
			}

ReadMTestCases	Integer	Common Type	

	Output	Viene restituita una lista di JsonObject TestCase	[
			{
			“id”: 1,
			“idPlayer”: 4,
			"totalResult" : 12568,
			"compilingResult" : 1212,
			ecc...
			}
			{
			“id”: 2,
			“idPlayer”: 5,
			"totalResult" : 165568,
			"compilingResult" : 1212,
			ecc...
			}
			]

ReadTestCaseByIdPlayer	Integer	CommonType	

	Output	Viene restituita una lista di JsonObject TestCase	[
			{
			“id”: 1,
			“idPlayer”: 4,
			"totalResult" : 12568,
			"compilingResult" : 1212,
			ecc...
			}
			{
			“id”: 2,
			“idPlayer”: 5,
			"totalResult" : 165568,
			"compilingResult" : 1212,
			ecc...
			}
			]

ReadTestCaseByIdRobot	Integer	CommonType	

		Viene restituita una lista di JsonObject TestCase	[
			{
			“id”: 1,
			“idPlayer”: 4,
			"totalResult" : 12568,
			"compilingResult" : 1212,
			ecc...
			}
			{
			“id”: 2,
			“idPlayer”: 5,
			"totalResult" : 165568,
			"compilingResult" : 1212,
			ecc...

DIZIONARIO	
Funzione	**Descrizione**
AddMatch	Permette di aggiungere un Match ed inizializzare i relativi Risultati associati ad ogni Player partecipante. Tali risultati sono al momento null.
AddRound	Permette di aggiungere un Round ad un Match esistente specificato in input.
UpdateRound	Permette di modificare i campi di un round esistente specificato in input.
UpdateMatch	Permette di modificare i campi di un Match esistente specificato in input.
AddTestCasePlayer	Permette di aggiunger un TestCasePlayer ad un Round esistente spcificato in input.
AddTestCaseRobot	Permette di aggiunger un TestCaseRobot ad un Round esistente spcificato in input.
ReadResultIdPlayer	Permette di visualizzare lo storico di Risultati di un Player esistente specificato in input.
ReadSMatch	Permette di visualizzare un singolo Match esistente in base all'id specificato in input.
ReadRound	Permette di visualizzare un singolo Round esistente in base all'id specificato in input.
ReadMRounds	Permette di visualizzare tutti i Round associati ad un Match esistente specificato in input.
ReadTestCase	Permette di visualizzare un singolo TestCase esistente in base all'id specificato in input.
ReadMTestCases	Permette di visualizzare tutti i TestCase associati ad un Round esistente specificato in input.
ReadTestCaseByIdPlayer	Permette di visualizzare un singolo TestCase esistente in base all'id del Player specificato in input.
ReadTestCaseByIdRobot	Permette di visualizzare un singolo TestCase esistente in base all'id del Robot specificato in input.
DeleteMatch	Permette di eliminare un Match esistente specificato in input.
DeleteRound	Permette di eliminare un Round esistente specificato in input.
DeleteResult	Permette di eliminare un Result esistente specificato in input.
DeleteTestCase	Permette di eliminare un TestCase esistente specificato in input.

Deploy e Guida di installazione

Deploy

Nel primo diagramma si è previsto un deploy su macchine separate, dove *TestGame.jar* rappresenta una possibile integrazione dei servizi (Game Engine, Student Repository, ecc.), escluso Game Repository, all'interno di un unico archivio java.

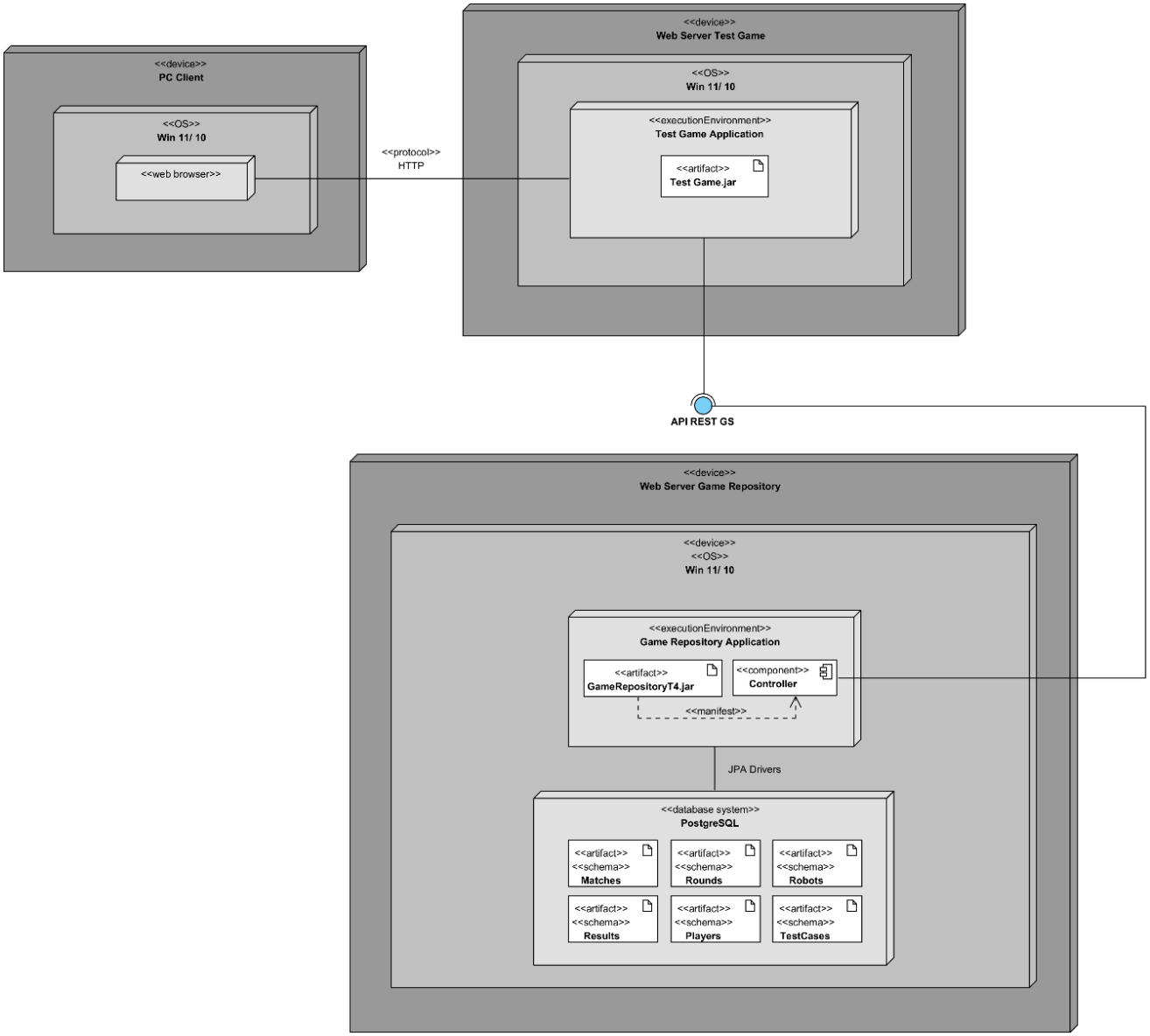


Figura 35 - Deployment Diagram Distributed

Visto l'impiego reale, dove l'installazione dei diversi servizi avviene sulla stessa macchina, si è previsto anche il seguente diagramma di deploy.

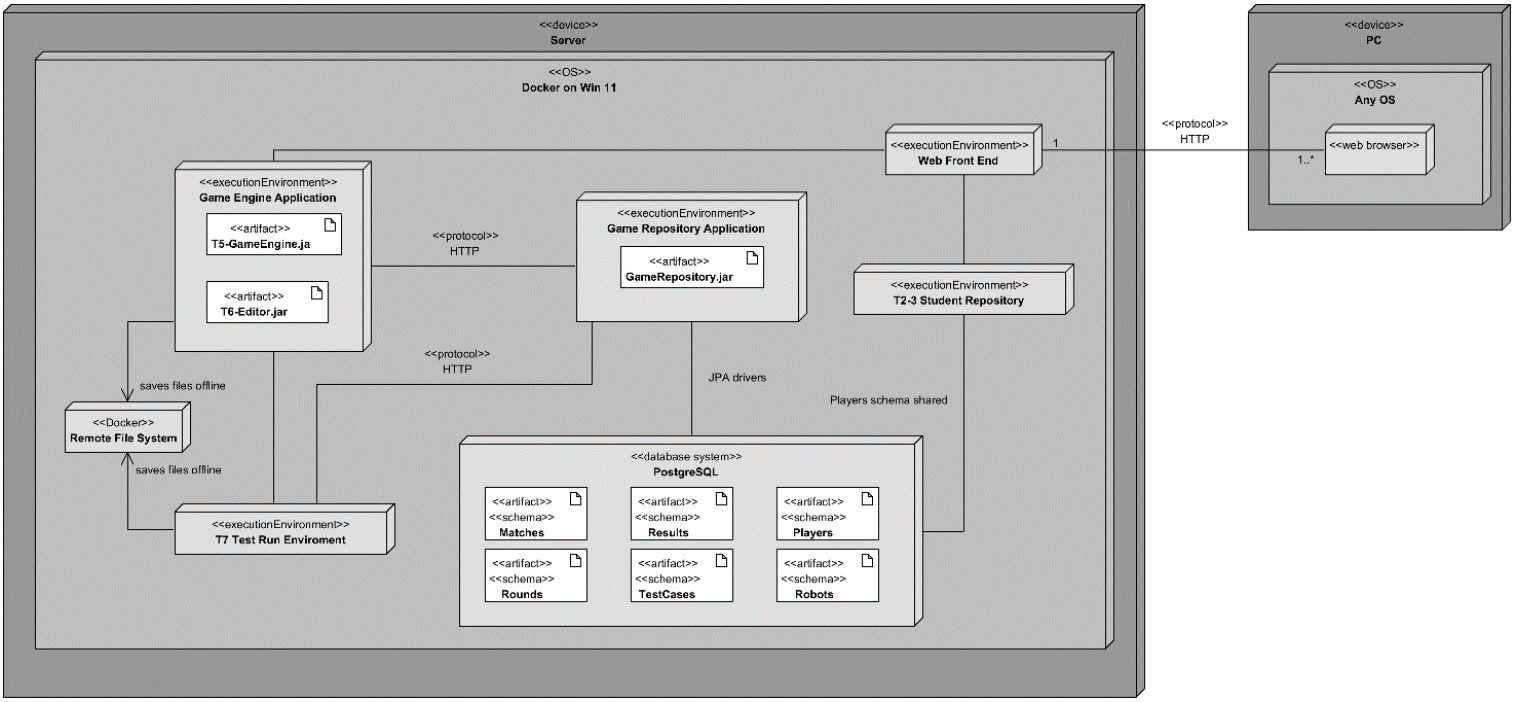


Figura 36 - Deploy Diagram not Distributed

Installation View

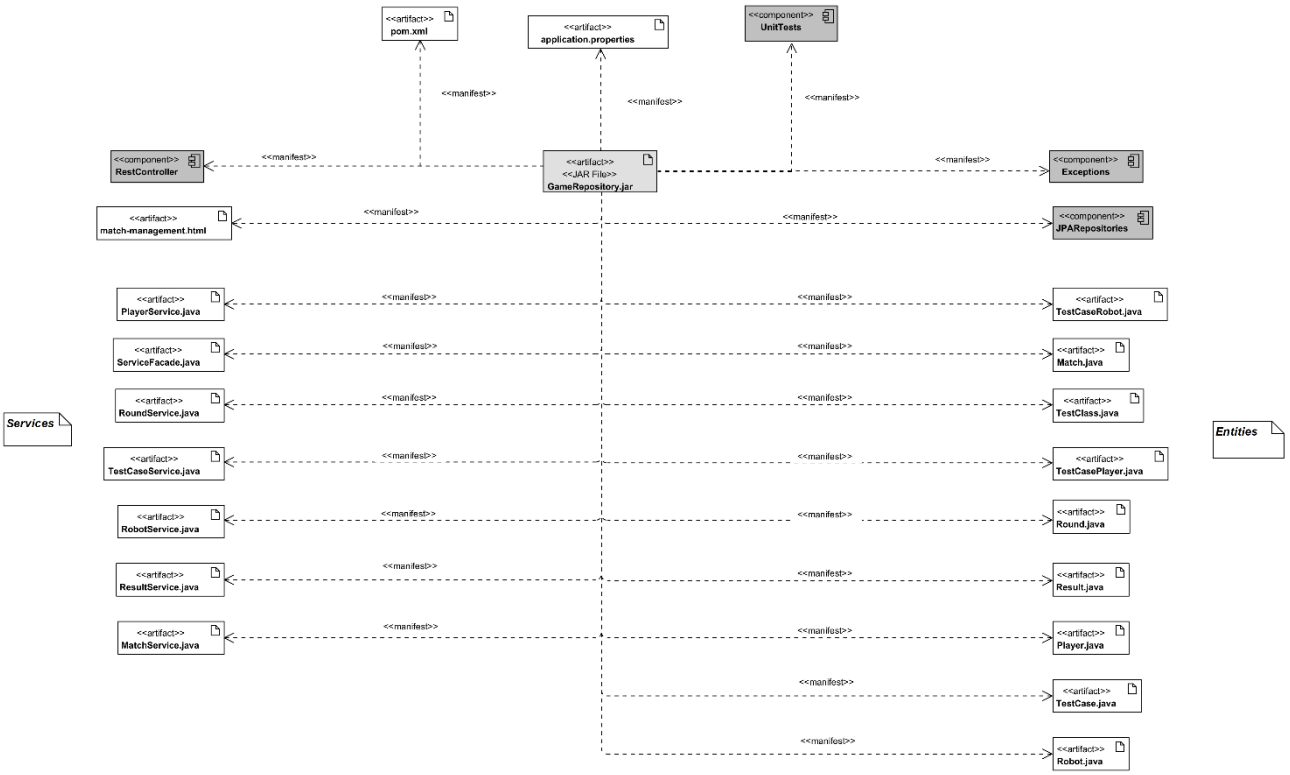


Figura 37 – Installation View Diagram

Guida di installazione

Prerequisiti per l'installazione

Prima di poter installare l'applicativo è necessario avere sulla macchina:

- Docker Desktop
- PostGreSQL 15.3

Installazione

Al fine di una corretta installazione dell'applicativo è necessario eseguire le seguenti operazioni:

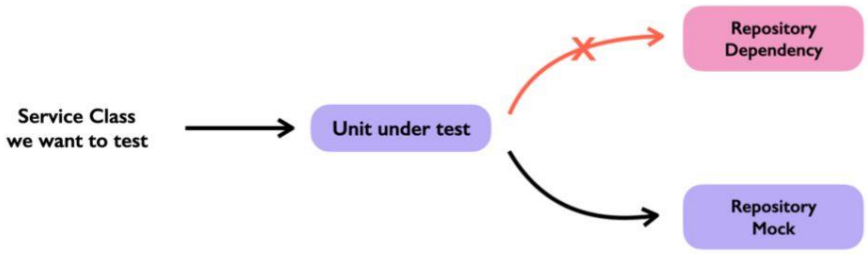
1. Importare il progetto presente sul repository Github all'indirizzo:
    a. https://github.com/Testing-Game-SAD-2023/T4-G15
2. Posizionarsi sulla directory dove si è scaricato il progetto ed accedere alla subdirectory *src/docker:*
    a. cd web-service-games-repository/src/docker
3. Avviare, da terminale, il comando:
    a. docker-compose up
4. Verificare la corretta creazione e avvio delle docker-Image attraverso Docker Desktop (*springapp* e *db* all'interno del container *docker*)
5. Accedere all'interfaccia grafica all'indirizzo:
    a. localhost:8080/match-management.html

    L'interfaccia permette di popolare il database con dei player e robot *prop* così da testare rapidamente i casi d'uso implementati.

    Per utilizzare i metodi REST consultare la documentazione ed effettuare richieste attraverso gli URI forniti (localhost:8080/URI).

Testing Funzionale

Per testare i casi d'uso implementati nel *Service Facade,* è stato utilizzato il framework *Mockito*, allo scopo di simulare il comportamento dei componenti appartenenti ai layer sottostanti. Questo ha reso possibile testare le funzionalità dell'applicativo, aggirando le *Repositories* nel *Data Access Layer.*



Nelle Test Suite riportate in seguito sono esplicitati i test effettuati con i relativi esiti (PASS or FAIL).

Test Suite

Test Case ID	Descrizione	Pre Condizioni	Input	Oracolo	Post Condizioni	Output Ottenuto	Post condizioni Ottenute	Esito(Fail, Pass)
1	Crea un match inserendo id player e tipo di scenario.	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id player, tipo di scenario	Match not null	nessuna	Match not null	Nessuna	Pass
2	Crea un Round su un match già creato	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Round, robot(Id, difficoltà)	Round not null	nessuna	Round not null	nessuna	Pass
3	Verifica dato un input sbagliato su un match non esistente	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id Match non esistente	MatchNotFoundException "The given Match does not exist!"	nessuna	MatchNotFoundException "The given Match does not exist!"	Nessuna	Pass
4	Crea un round dato un robot con id 0	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Robot(id:0)	RobotNotFoundException "Robot id not Found"	nessuna	RobotNotFoundException"Robot id not Found"	Nessuna	Pass
5	Modifica un round inserendo in un secondo momento un nuovo robot	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Input(id robot nuovo e scenario nuovo)	Id robot uguale a 789(Id del robot che intendiamo mettere nel round).	nessuna	Id robot 789	Nessuna	Pass
6	Crea un Test case Player in un match che non contiene il round indicato	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id del match, id del round, id player e il caso di test creato	RoundNotFoundException "The given match does not contain the given round!"	nessuna	RoundNotFoundException"The given match does not contain the given round!"	Nessuna	Pass

Test Case ID	Descrizione	Pre Condizioni	Input	Oracolo	Post Condizioni	Output Ottenuto	Post condizioni Ottenute	Esito(Fail, Pass)
7	Crea un Test Case Player con un Player che non partecipa a quel Match	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id del match, id del round, id player ,caso di test player	PlayerNotFoundException"The given player does not partecipate in this match"	nessuna	PlayerNotFoundException"The given player does not partecipate in this match"	Nessuna	Pass
8	Crea un test case Player	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id del match, id del round ,test case player	TestcasePlayer not null	Nessuna	TestCasePlayer not null	Nessuna	Pass
9	Crea un Test Case robot con un round non esistente	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id match, id round, test case robot	RoundNotFoundException "The given match does not contain the given round!"	nessuna	RoundNotFoundException"The given match does not contain the given round!"	Nessuna	Pass
10	Crea un test Case Robot	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id match, id round, test case robot	TestCase Robot not null	nessuna	TestCaseRobot not null	Nessuna	Pass
11	Va ad effettuare un update sul Match aggiungendo data di fine e scenario	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Scenario e fine partita	Update Match not null	nessuna	Update Match not null	Nessuna	Pass
12	Va a verificare se effettivamente lo scenario aggiunto è uguale a quello passato	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Scenario e fine partita	updateMatch.getScenario equals "scenario"	nessuna	updateMatch.getScenario equals "scenario"	Nessuna	Pass

Test Case ID	Descrizione	Pre Condizioni	Input	Oracolo	Post Condizioni	Output Ottenuto	Post condizioni Ottenute	Esito(Fail, Pass)
13	Va a verificare se la data di fine è uguale a quella passata	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Scenario e fine partita	upDateMatch.getEndDate equals “2023-06-12T10:00:00“	Nessuna	updateMatch.getEndDate equals “2023-06-12T10:00:00“	Nessuna	Pass
14	Va a verificare il metodo Read Player Result dato un player non esistente	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id player	PlayerNotFoundException”Player Not Found”	Nessuna	PlayerNotFoundException”Player Not Found”	Nessuna	Pass
15	Va a verificare il metodo Read Player Result dato un player che non ha ottenuto nessun risultato	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id player	ResultNotFoundException “No result available for this player, yet”	Nessuna	ResultNotFoundException “No result available for this player, yet”	Nessuna	Pass
16	Va a verificare il metodo Read Result Player	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id player	serviceFacade.ReadResultIdPlayer equals List<Result>	Nessuna	player.REadResultIdPlayer equals List<Result>	Nessuna	Pass
17	Va a verificare i test case effettuati dato un match ed un suo round non esistente	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id match, id Round	RoundNotFoundException “Round not Found for Match given in input”	nessuna	RoundNotFoundException “Round not Found for Match given in input”	Nessuna	Pass
18	Va a verificare i test case associati ad un round	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id match, id Round	TestNotFoundException “No Test associated to given Round”	nessuna	TestNotFoundException “No Test associated to given Round”	Nessuna	Pass

Test Case ID	Descrizione	Pre Condizioni	Input	Oracolo	Post Condizioni	Output Ottenuto	Post condizioni Ottenute	Esito(Fail, Pass)
19	Va a verificare i test case associati ad un round	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id match, id Round	serviceFacade.ReadMTestCases equals List<TestCases>	Nessuna	serviceFacade.ReadMTestCases equals List<TestCases>	Nessuna	Pass
20	Va ad eliminare un round di un match non esistente	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id Match	MatchNotFoundException "Match not found"	Nessuna	MatchNotFoundException "Match not found"	Nessuna	Pass
21	Va ad eliminare un round in un match	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id match, id Round	DeleteRound = True	Nessuna	DeleteRound = True	Nessuna	Pass
22	Va ad eliminare match non esistente	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id Match	MatchNotFoundException "Match not found"	Nessuna	MatchNotFoundException "Match not found"	Nessuna	Pass
23	Va ad eliminare i Round in un Match	Devono essere creati tutti gli oggetti Mock per simulare il comportamento del metodo	Id match	Boolean Result = true	Nessuna	Boolean result =true	Nessuna	Pass