



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

Software Architecture Design

Documentazione Team B14 Task R5

Anno Accademico 2024/2025

Prof.ssa: **Anna Rita Fasolino**

Membri del Team:

Francesco Prisco M63001738
Christian Morelli M63001715
Antonio Tufo M63001727
Matteo Cuomo M63001756

Abstract

Questo elaborato descrive le metodologie, le fasi di sviluppo e gli artefatti prodotti durante il corso di Software Architecture Design, anno accademico 2024/25, per la realizzazione di uno specifico requisito all'interno del progetto ENACTEST. In particolare, il progetto ha come obiettivo la creazione e lo sviluppo di un gioco con lo scopo di promuovere l'attività di testing e permettere ai partecipanti di cimentarsi in sfide contro sistemi automatizzati ed eventualmente altri giocatori.

L'elaborato è stato suddiviso nei seguenti capitoli:

- Capitolo 1: Introduzione

Il capitolo presenta il progetto ENACTEST, descrivendo la Web Application, lo stato iniziale del sistema e il requisito R5 per la gestione dei team.

- Capitolo 2: Metodologie di sviluppo

Qui vengono illustrate le metodologie, gli strumenti e le tecnologie che ci hanno accompagnato nel nostro lavoro di sviluppo software;

- Capitolo 3: Requisito R5 - Analisi preliminare

Viene analizzato il requisito R5, che introduce la gestione dei team e delle challenge. Si definiscono i casi d'uso, le user stories e i criteri di accettazione. Vengono aggiornati diagrammi chiave per modellare le nuove funzionalità e valutare l'impatto delle modifiche sul sistema.

- Capitolo 4: Requisito R5 - Nuove implementazioni

Questo capitolo descrive lo sviluppo delle nuove funzionalità, come la creazione e gestione dei team e delle challenge. Si introducono nuovi componenti backend (modelli, repository, servizi) e vengono integrate API REST per migliorare l'interazione tra le varie parti del sistema.

-
- Capitolo 5: Diagrammi di progetto per le nuove implementazioni
Vengono presentati i diagrammi aggiornati per le nuove implementazioni, tra cui i Sequence Diagram per le operazioni principali e il Package Service, che evidenzia la modularità dell'architettura.
 - Capitolo 6: Evoluzione del Front-End
Viene presentata una raccolta di screenshot delle principali interfacce del front end dedicate alla gestione dei team e delle challenges. Ogni schermata è accompagnata da una spiegazione dettagliata che descrive le funzionalità disponibili, le azioni che l'Admin può eseguire e il comportamento dell'interfaccia utente.
 - Capitolo 7: Organizzazione del lavoro con Scrum
Viene descritto l'approccio Scrum, con focus su sprint, review e organizzazione del team per garantire un flusso di lavoro efficace.
 - Capitolo 8: Glossario
Il capitolo finale fornisce un glossario dei termini principali utilizzati nella documentazione per facilitare la comprensione del testo.

Indice

1	Introduzione	1
1.1	Descrizione della Web Application	2
1.2	Descrizione del requisito assegnato	3
1.2.1	Requisito R5	4
1.3	Stato iniziale della Web Application	5
1.3.1	Deployment Diagram della Web Application . .	6
1.3.2	Composite Diagram della Web Application . .	8
1.3.3	Stato Container T1	9
1.3.4	Class Diagram	12
1.3.5	Deployment Diagram del T1	13
1.3.6	Refactoring Container T1	14
1.3.7	Component Diagram T1 aggiornato	15
1.3.8	Package Service del T1	18
2	Metodologie di sviluppo	20
2.1	Daily Scrum	20
2.2	Sprint Backlog	21
2.3	Strumenti	22
2.3.1	Microsoft Teams	22

2.3.2	Discord	22
2.3.3	GitHub	23
2.3.4	Visual Paradigm	24
2.3.5	GoodNotes	24
2.3.6	Keynote	25
2.4	Tecnologie e ambienti di sviluppo	25
2.4.1	Visual Studio Code	25
2.4.2	Java Spring e Spring Boot	26
2.4.3	Bootstrap	27
2.4.4	Maven	27
2.4.5	MongoDB Compass	28
2.4.6	Docker Desktop	28
3	Requisito R5: Analisi preliminare	29
3.1	Osservazioni e problematiche	29
3.2	Analisi d’Impatto	31
3.3	Analisi dei requisiti	32
3.4	User Story	33
3.5	Criteri di Accettazione	34
3.6	Use Case Diagram aggiornato	41
3.7	Diagramma ER aggiornato	42
3.8	Package Diagram	44
3.9	Component Diagram aggiornato	45
4	Requisito R5: Nuove Implementazioni	47
4.1	Implementazione della Funzionalità di Gestione del Team	47
4.1.1	Team.java	48

4.1.2	TeamSearchImpl.java	50
4.1.3	TeamService.java	51
4.1.4	TeamRepository.java	58
4.2	Implementazione della Funzionalità di Gestione delle Challenges	58
4.2.1	Challenge.java	59
4.2.2	ChallengeSearchImpl.java	60
4.2.3	ChallengeService.java	61
4.2.4	ChallengeRepository.java	69
4.3	Modifiche nell'HomeController	70
4.3.1	Sviluppo delle Rotte API per i Team	70
4.3.2	Sviluppo delle Rotte API per le Challenges	72
5	Diagrammi di Progetto per le Nuove Implementazioni	73
5.1	Sequence Diagram createTeam	74
5.2	Sequence Diagram deleteTeam	75
5.3	Sequence Diagram addMemberToTeam in createTeam	76
5.4	Sequence Diagram addMemberToTeam	77
5.5	Sequence Diagram removeMemberFromTeam	78
5.6	Sequence Diagram createChallenge	79
5.7	Sequence Diagram deleteChallenge	80
5.8	Sequence Diagram getAllChallenges	81
5.9	Package Service Aggiornato	82
6	Evoluzione del Front-End	83
6.1	Front-End Gestione Team	84
6.1.1	Sezione di Crea Team	86

6.1.2	Sezione di Aggiungi Membro	89
6.1.3	Sezione di Lista Team	91
6.1.4	Sezione di Dettagli Team	92
6.1.5	Sezione di Cancella Team	93
6.2	Front-End Gestione Challenges	94
6.2.1	Sezione di Crea Challenge	95
6.2.2	Sezione di Lista Challenges	98
6.2.3	Sezione di Dettagli Challenge	100
6.2.4	Sezione di Cancella Challenge	101
7	Organizzazione del lavoro con Scrum	103
7.1	Adozione di Scrum per il nostro progetto	103
7.2	Prima Iterazione	104
7.3	Seconda Iterazione	105
7.4	Prima Review	105
7.5	Seconda Review	106
7.6	Terza Review	107
8	Glossario	108
8.1	Admin	108
8.2	Container	108
8.3	Team	108
8.4	Challenge	109
8.5	Studente	109

Capitolo 1

Introduzione

Il progetto ENACTEST (European iNnovative AllianCe for TESTing) nasce per promuovere l'importanza del testing nello sviluppo software, migliorando la qualità delle applicazioni web attraverso strumenti e metodologie innovative. Nel contesto moderno, il testing è spesso trascurato, portando a vulnerabilità non identificate prima del rilascio. ENACTEST affronta questa sfida sensibilizzando sul valore del testing e introducendo approcci innovativi.

Tra le sue iniziative principali, il gioco educativo “Man vs Automated Testing Tools Challenges” permette ai partecipanti di sviluppare test manuali più efficaci rispetto a quelli generati automaticamente, unendo apprendimento pratico e gamification. Il progetto, sviluppato con metodologie agili, coinvolge studenti universitari e internazionali, offrendo un’esperienza formativa basata su progetti reali.

L’obiettivo finale è integrare queste metodologie nei curricula accade-

mici e professionali, migliorando la qualità del testing e riducendo la necessità di formazione supplementare in ambito aziendale.

1.1 Descrizione della Web Application

“Man vs Automated Testing Tools Challenges” è una Web Application educativa progettata per introdurre gli studenti al testing del software attraverso un approccio basato sulla gamification. Il gioco prevede una competizione tra studenti e strumenti di testing automatizzati, come Randoop ed Evosuite, entrambi in grado di generare casi di test JUnit in modo autonomo.

I partecipanti devono sviluppare casi di test di unità per classi Java con l’obiettivo di raggiungere metriche specifiche, tra cui:

- Copertura delle Linee di Codice (LOC);
- Copertura dei Branch e delle Decisioni;
- Gestione delle Eccezioni;
- Copertura della Weak Mutation.

Tutti i casi di test vengono implementati utilizzando JUnit 4, favorendo così un approccio pratico e standardizzato.

La piattaforma si basa su un modello collaborativo e iterativo, in cui ciascun gruppo partecipante è responsabile dell’implementazione di task associati a requisiti funzionali definiti. L’intero progetto, insieme

alla documentazione dettagliata prodotta dai team, è disponibile su GitHub, offrendo un'analisi trasparente del lavoro svolto e facilitando la condivisione delle conoscenze.

Questa applicazione non solo stimola lo sviluppo di competenze tecniche avanzate, ma incoraggia anche una sana competizione e l'apprendimento pratico del testing attraverso un confronto diretto con strumenti automatizzati, rendendo l'esperienza educativa coinvolgente e innovativa.

1.2 Descrizione del requisito assegnato

L'obiettivo principale di questa fase del progetto è migliorare la Web Application, arricchendola con nuove funzionalità per ottimizzarne l'efficacia e l'usabilità. Per raggiungere questo scopo, il lavoro è stato suddiviso tra diversi team, ciascuno incaricato di implementare specifici requisiti funzionali. Questa suddivisione non solo garantisce una gestione più efficiente dello sviluppo, ma favorisce anche un approccio collaborativo e mirato, essenziale per il successo del progetto.

Il requisito assegnato al nostro team, identificato come requisito R5, è parte integrante del processo iterativo di sviluppo del progetto. I dettagli tecnici e le specifiche relativi al requisito R5 sono descritti nel sottoparagrafo successivo.

1.2.1 Requisito R5

Il requisito R5 prevede l'implementazione di una funzionalità che consenta all'Admin di gestire e creare team di studenti all'interno della Web Application. Questa feature rappresenta un elemento centrale per migliorare l'organizzazione e il monitoraggio delle attività didattiche, offrendo strumenti avanzati per la gestione delle sfide educative.

Le principali attività incluse in questo requisito sono:

- Creazione e gestione dei team: l'Admin deve avere la possibilità di creare nuovi team, aggiungere o rimuovere studenti e gestire i dettagli dei team in modo semplice ed efficace;
- Assegnazione di sfide: l'Admin può assegnare sfide o compiti agli studenti appartenenti ad un team, specificando vincoli temporali per la loro esecuzione;
- Monitoraggio delle performance: attraverso una dashboard dedicata, l'Admin può verificare e analizzare i risultati dei team.

L'obiettivo di questa funzionalità è migliorare la capacità di gestione del sistema e contribuire ad una maggiore personalizzazione e ottimizzazione delle attività formative.

1.3 Stato iniziale della Web Application

Per comprendere appieno il lavoro svolto, è importante considerare ciò che abbiamo ricevuto "in eredità" dai colleghi che hanno lavorato alle prime fasi di questo progetto. Prima di analizzare il progetto nel dettaglio, è fondamentale soffermarsi su un aspetto cruciale: l'installazione del programma. Questa fase si è rivelata particolarmente complessa, principalmente a causa dell'incompletezza delle guida fornite, che non erano sufficientemente dettagliate per supportarci efficacemente. Nonostante le difficoltà iniziali, siamo riusciti a superare questi ostacoli e a procedere con successo nelle fasi successive del lavoro.

L'architettura scelta è basata su microservizi, che permettono di isolare e scalare singole funzionalità. Questo approccio facilita lo sviluppo autonomo e la distribuzione rapida dei componenti.

1.3.1 Deployment Diagram della Web Application

Riportiamo di seguito il corrispondente Deployment Diagram 1.1 necessario per rappresentare la disposizione fisica dei componenti hardware e software in un sistema distribuito. Questa immagine è stata estratta dalla documentazione A13. Di seguito, è riportato il link al repository GitHub contenente la documentazione: https://github.com/Testing-Game-SAD-2023/A13/tree/main/Documentazione/Documentazione_A13.

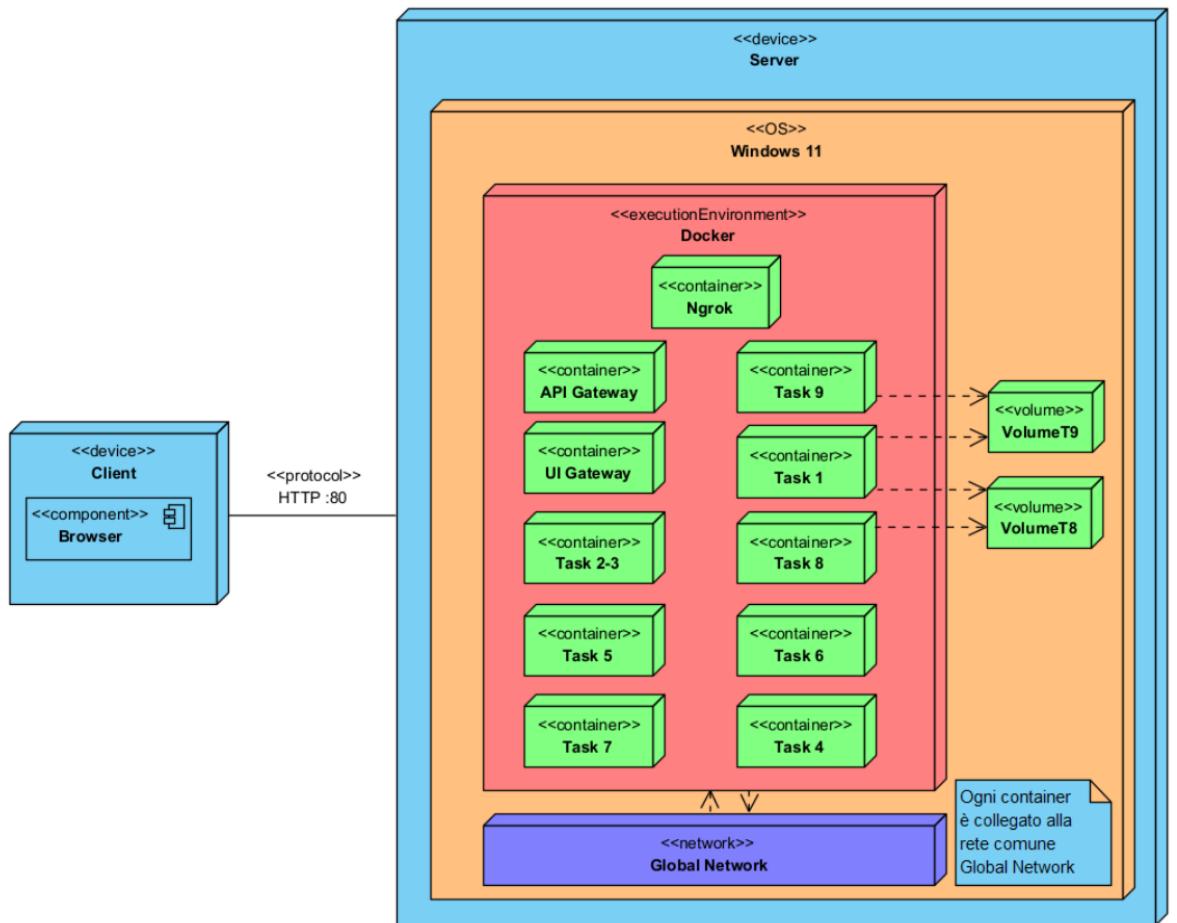


Figura 1.1: Deployment Diagram della Web Application

Di seguito analizziamo più nel dettaglio i container di tale diagramma:

- Container T1: Si occupa della gestione delle classi di programmazione da testare e della visualizzazione della classifica dei giocatori. Include funzionalità di caricamento, modifica e ricerca delle classi di programmazione per gli amministratori;
- Container T23: Responsabile per la registrazione e autenticazione dei giocatori, consente la gestione delle credenziali e include funzioni per il reset della password;
- Container T4: Repository dati del gioco, per salvare e aggiornare i dati di gioco;
- Containers T5 e T6: Front-end di selezione e gestione della partita;
- container T7: Si occupa di compilare ed eseguire i casi di test prodotti dallo studente in partita;
- Containers T8 e T9: Compilazione ed esecuzione dei test prodotti da EvoSuite e Randoop.

Ci concentreremo sul container T1, di cui abbiamo studiato attentamente la documentazione, che sarà il principale oggetto di modifica al fine di soddisfare il Requisito R5 assegnato.

1.3.2 Composite Diagram della Web Application

Forniamo ora una vista generale sul sistema nel complesso attraverso il Composite Diagram in Figura 1.2 . Questa immagine è stata estratta dalla documentazione A13.

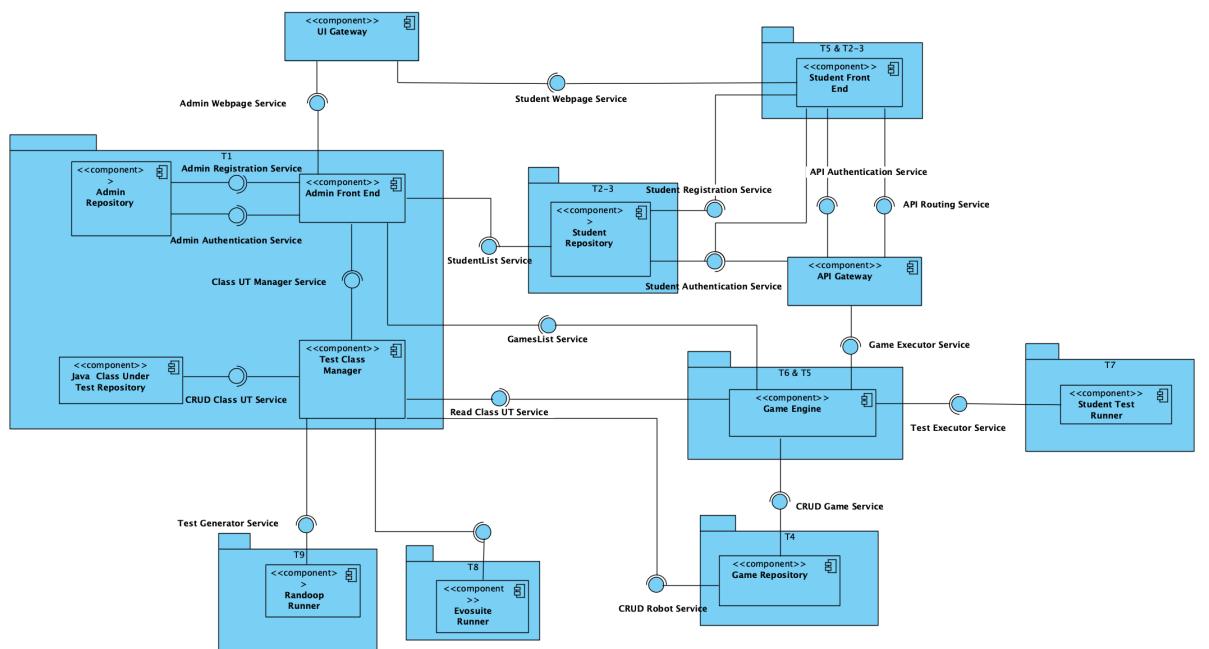


Figura 1.2: Composite Diagram della Web Application

1.3.3 Stato Container T1

Di seguito presentiamo la struttura della prima versione del container T1 per facilitarne la comprensione. La Figura 1.3 mostra il relativo diagramma dei componenti di T1. Questa immagine è stata estratta dalla documentazione A13. Di seguito, è riportato il link al repository GitHub contenente la documentazione: https://github.com/Testing-Game-SAD-2023/A13/tree/main/Documentazione/Doc_T1/Documentazione_T1_G11/documentazione.

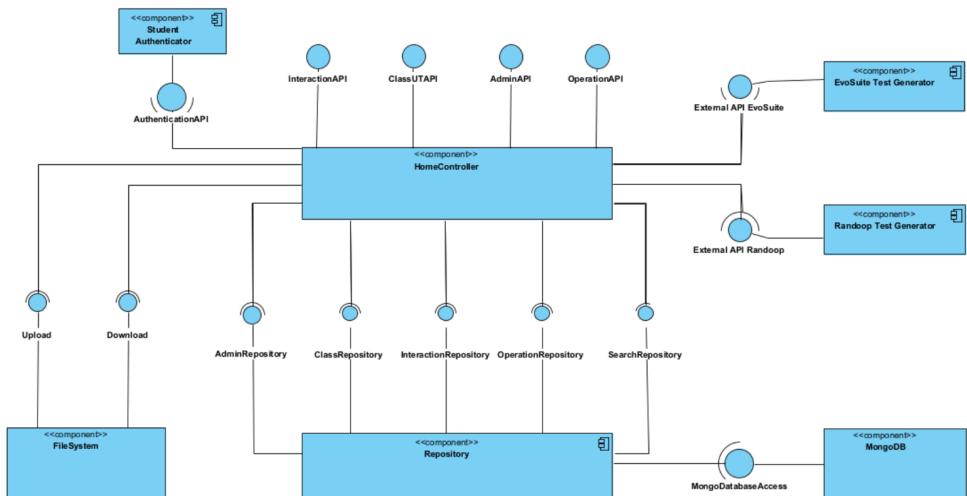


Figura 1.3: Component Diagram del Container T1

Il Component Diagram è un utile strumento per descrivere l'architettura di un sistema software e le interazioni tra i suoi componenti. Esso presenta i seguenti componenti:

- **HomeController**: rappresenta il container centrale del sistema,

responsabile della gestione delle operazioni di inserimento, cancellazione, modifica e visualizzazione delle Classi UT. Inoltre, coordina le interazioni tra studenti e classi, i dati degli amministratori e le operazioni che questi ultimi possono eseguire sulle classi. Il container si interfaccia con il MongoRepository attraverso le interfacce offerte dal container Repository e con il FileSystem tramite le funzionalità di download e upload. Oltre a ciò, HomeController comunica con componenti esterni, come RandoopTestGenerator, EvosuiteTestGenerator e Student Authenticator, utilizzando apposite interfacce esterne.

- Repository: è il container incaricato di interagire con il MongoDB, fornendo un set di interfacce per le operazioni CRUD e per attività di ricerca e filtraggio delle collezioni presenti nel database. Il MongoDB, a sua volta, rappresenta il database fisico in cui vengono archiviati i metadati relativi alle classi, le informazioni degli amministratori, e i dati sulle interazioni e operazioni che coinvolgono le classi.
- FileSystem: si occupa della gestione della directory condivisa in cui vengono archiviati fisicamente i file .java relativi alle classi da testare. Espone interfacce dedicate per le operazioni di upload e download dei file nella repository.
- Student Authenticator: consente di recuperare i parametri dello

studente attualmente autenticato nella sessione attiva, utilizzando l’interfaccia AuthenticatorAPI.

- EvosuiteTestGenerator: è responsabile della generazione automatica dei test Evosuite, attivata ogni volta che una nuova classe viene caricata per il testing.
- RandoopTestGenerator: si occupa della generazione automatica dei test Randoop, eseguita quando una nuova classe viene caricata per il testing.

1.3.4 Class Diagram

Per maggiore chiarezza mostriamo il diagramma delle classi del T1 in Figura 1.4

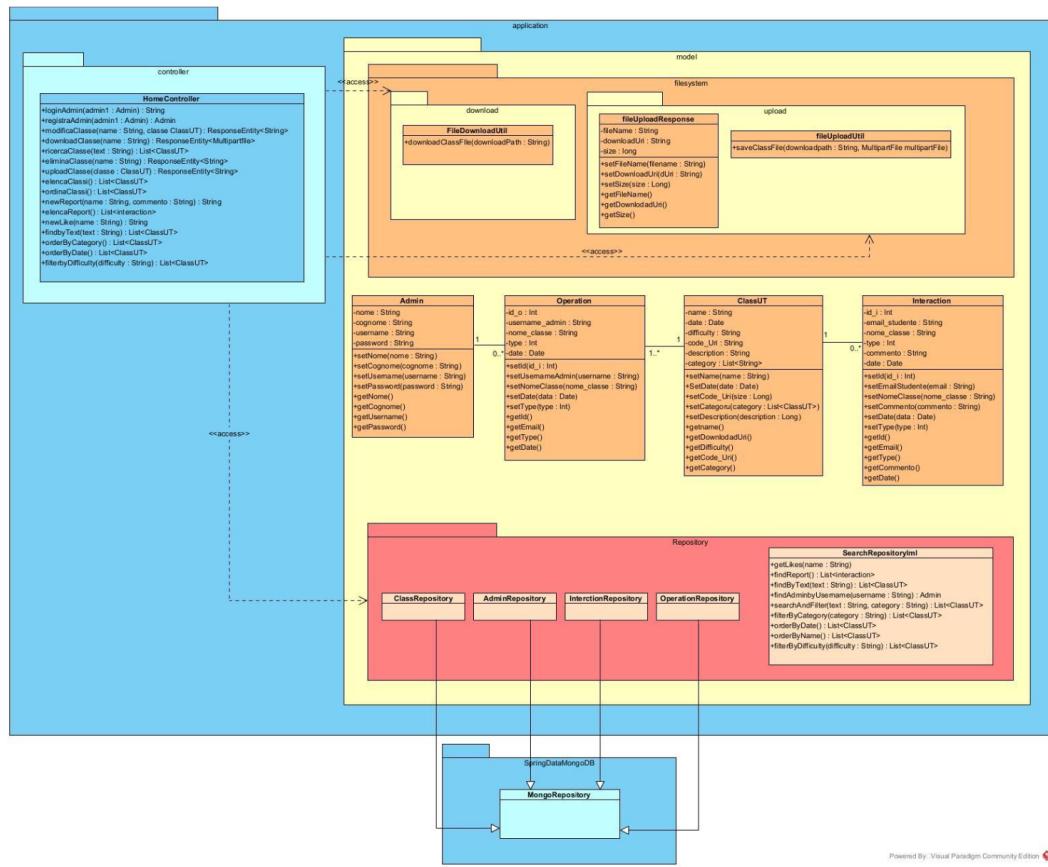


Figura 1.4: Diagramma delle classi di T1

Come evidenziato dai diagrammi forniti, il container T1 presenta un'architettura poco organizzata, caratterizzata da un elevato livello di complessità e da un forte accoppiamento tra i componenti. Per affrontare queste problematiche, è stato effettuato un refactoring di T1, come verrà illustrato nel paragrafo successivo.

1.3.5 Deployment Diagram del T1

Dalla figura 1.5, estratta dalla documentazione situata al seguente link https://github.com/Testing-Game-SAD-2023/A13/tree/main/Documentazione/Doc_T1/Documentazione_T1_G11/documentazione, possiamo notare che l'applicazione viene eseguita su un computer con sistema operativo Windows 11, dove è installato Docker Desktop per la gestione dei container. Due container Docker vengono creati per ospitare rispettivamente l'applicazione e il database. L'ambiente di esecuzione dell'applicazione è configurato utilizzando la JVM21, mentre il database è accessibile tramite il protocollo TCP sulla porta 27017. I container comunicano tra loro attraverso una rete privata configurata da Docker Desktop. Per consentire l'accesso all'applicazione da parte degli utenti, il container dell'applicazione è esposto all'esterno tramite il protocollo HTTP sulla porta 8080, rendendolo accessibile da qualsiasi browser e sistema operativo. Un elemento importante di questo setup è la gestione dei file. Viene creata una cartella dedicata al salvataggio dei file, che viene clonata nell'ambiente Windows 11 per essere utilizzata come volume condiviso. Questo approccio offre numerosi vantaggi:

- Persistenza dei file: i file delle classi salvati nel volume condiviso rimangono accessibili anche quando i container sono spenti o riavviati, garantendo che non vadano persi.
- Condivisione dei file: il volume condiviso consente ad altri task

o servizi di accedere ai file delle classi senza dipendere esclusivamente dal funzionamento dei container.

Questa configurazione assicura che l'applicazione venga eseguita in modo isolato all'interno del container Docker, mantenendo al contempo un accesso efficiente e persistente ai file necessari per il funzionamento del sistema.

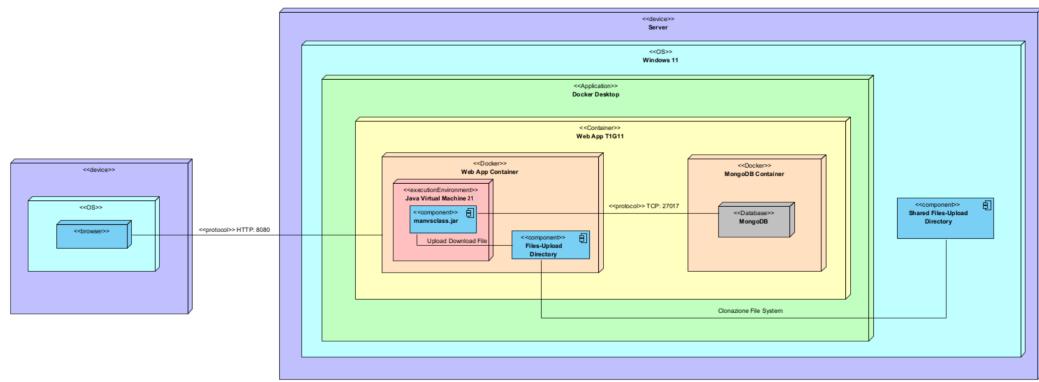


Figura 1.5: Deployment Diagram del T1

1.3.6 Refactoring Container T1

Il lavoro di refactoring del container T1, portato avanti dai nostri colleghi, ha affrontato le problematiche di complessità e accoppiamento evidenziate nella versione iniziale. Questo intervento è stato fondamentale per rendere il sistema più modulare, scalabile e manutenibile, introducendo un livello di servizi intermedi tra il controller e i repository. Le modifiche hanno consentito una migliore organizzazione delle

responsabilità e una maggiore flessibilità nell'aggiunta di nuove funzionalità, come richiesto dal Requisito R5. Di seguito verrà fornito il link alla repository GitHub contenente i dettagli di questo lavoro:
<https://github.com/lor3nzopoly/A13>.

Analizzeremo ora le motivazioni che hanno reso necessario il refactoring:

- Migliorare la modularità: Separando le responsabilità tra controller, servizi e repository, si ottiene un'architettura modulare in cui ogni container ha un ruolo specifico e limitato;
- Aumentare la scalabilità: La nuova struttura consente di aggiungere nuovi servizi o repository senza influire direttamente sul controller, facilitando l'estensione del sistema;
- Facilitare la manutenzione: Riducendo l'accoppiamento tra i componenti e centralizzando la logica applicativa nei servizi, le modifiche al codice diventano più facili da implementare e testare.

1.3.7 Component Diagram T1 aggiornato

Di seguito in Figura 1.6 viene presentato il Component Diagram che illustra il lavoro di refactoring effettuato.

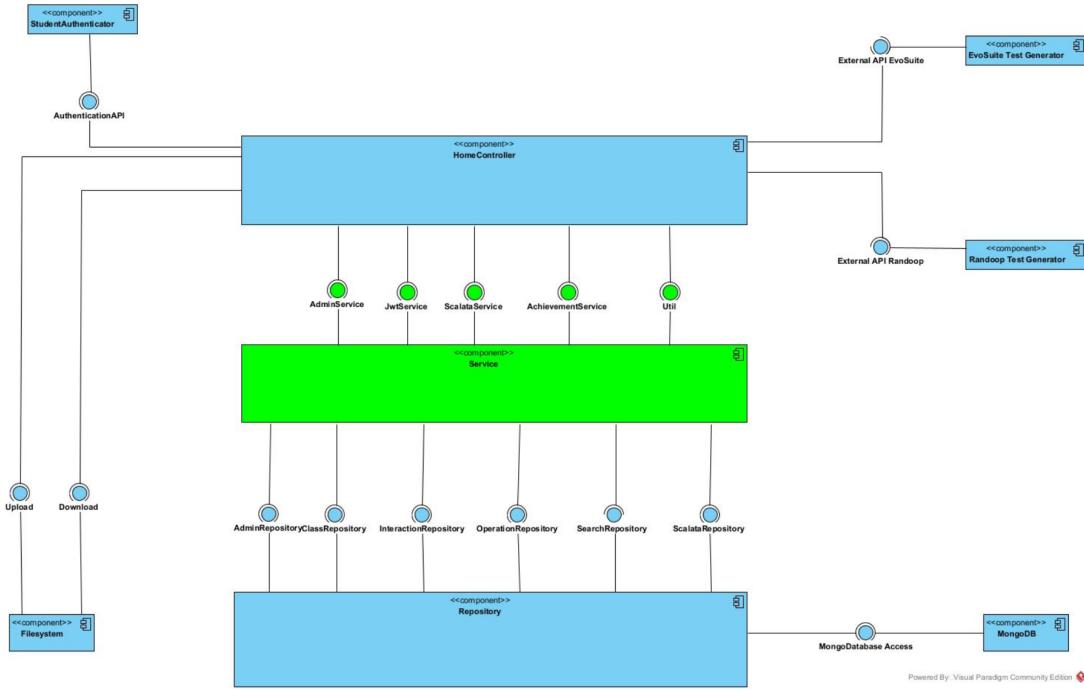


Figura 1.6: Component Diagram del Refactoring di T1

L’immagine mostra il Component Diagram aggiornato del Task T1 dopo il refactoring, che evidenzia una struttura più organizzata e modulare rispetto alla versione precedente. I componenti principali sono:

- Service: È stato aggiunto un nuovo livello intermedio, chiamato Service, che separa l’HomeController dai repository. Questo livello gestisce la logica applicativa e centralizza le operazioni tra il controller e i repository, consentendo una chiara separazione delle responsabilità;
- HomeController: Ora il controller si occupa esclusivamente di ricevere richieste dall’utente e di delegare la logica ai servizi.

Questo riduce il carico sul controller e rende il sistema più leggibile e manutenibile. Così facendo, il ruolo del controller si limita ad una semplice interfaccia tra il frontend e i servizi di business;

- Repository: I repository sono accessibili solo attraverso i servizi, eliminando il collegamento diretto tra il controller e il database. Questo package era già presente prima del refactoring e mantiene la sua funzione di interfacciarsi con i servizi e con il database.

1.3.8 Package Service del T1

Il Package Service (Figura 1.7) è stato aggiunto durante il processo di refactoring dell’architettura, con l’obiettivo di separare la logica di business dal controller principale (HomeController).

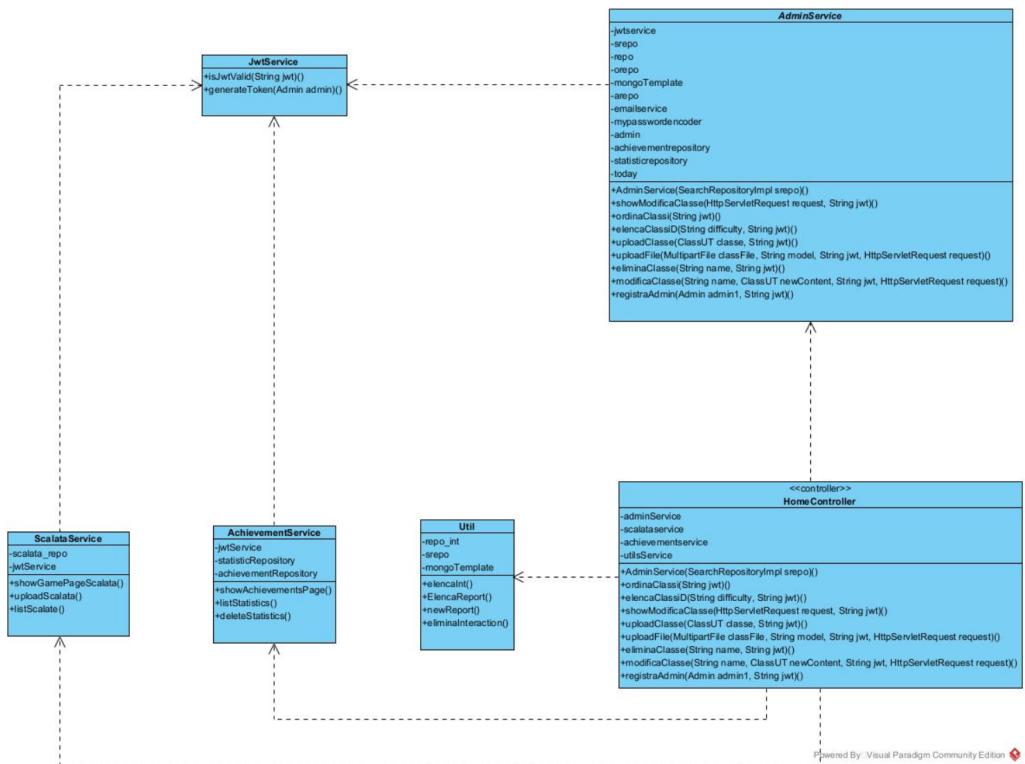


Figura 1.7: Package Service

In conclusione, il refactoring del Task T1 ha consentito di trasformare un’architettura complessa e poco organizzata in una struttura più modulare, scalabile e manutenibile. Grazie all’introduzione del livello di servizi e alla separazione delle responsabilità, il sistema risulta ora più chiaro e flessibile, rendendo più semplice l’aggiunta di nuove funziona-

CAPITOLO 1. INTRODUZIONE

lità. Noi stessi faremo leva sulla logica dei servizi introdotta durante il refactoring per implementare il nostro Requisito R5, sfruttando i vantaggi offerti da questa nuova struttura.

Capitolo 2

Metodologie di sviluppo

Il primo passo per realizzare il nostro task è stato adottare la metodologia AGILE, nota per favorire collaborazione, adattabilità e gestione iterativa delle attività. Basata su cicli di lavoro brevi (sprint), consente di suddividere il lavoro in task gestibili e di adattarsi rapidamente a cambiamenti. In questo progetto, particolare attenzione è stata data al Daily Scrum e allo Sprint Backlog, fondamentali per l'organizzazione e il monitoraggio del lavoro.

2.1 Daily Scrum

Il Daily Scrum è una breve riunione giornaliera della durata di circa 15 minuti, finalizzata a favorire la comunicazione all'interno del team e a mantenere tutti i membri allineati sullo stato di avanzamento del progetto. Inizialmente, abbiamo incontrato difficoltà nel comprendere

come gestire e implementare la nuova funzionalità richiesta. Tuttavia, grazie a queste riunioni giornaliere, siamo riusciti a confrontarci in modo efficace, individuare i problemi principali e definire il miglior approccio per affrontarli. Le riunioni si sono svolte prevalentemente sulla piattaforma Microsoft Teams, che ha agevolato la collaborazione e la comunicazione. Questo metodo ci ha permesso di chiarire tempestivamente i dubbi, approfondire la comprensione del task e ottimizzare il flusso di lavoro. Inoltre, il Daily Scrum ha facilitato il monitoraggio delle attività e il coordinamento tra i membri del team, rendendo il processo decisionale più rapido e collaborativo.

2.2 Sprint Backlog

Lo Sprint Backlog rappresenta la pianificazione dettagliata dei task da completare durante uno sprint, che solitamente ha una durata di 1-2 settimane. Grazie agli incontri regolari, è stato possibile assegnare e suddividere i vari task tra i membri del gruppo, stabilendo scadenze precise per il loro completamento. Inizialmente, le scadenze erano fissate settimanalmente o, al massimo, ogni due settimane. Tuttavia, con l'avvicinarsi della data di presentazione del progetto, è stato necessario intensificare gli incontri per garantire il completamento di tutte le attività nei tempi previsti. Questo approccio ha permesso di definire con chiarezza gli obiettivi di ogni sprint, monitorare il progresso delle attività e distribuire i compiti in modo equilibrato tra i membri del

team. Lo Sprint Backlog si è rivelato uno strumento essenziale per mantenere il team organizzato e focalizzato sulle priorità.

2.3 Strumenti

2.3.1 Microsoft Teams

Microsoft Teams è una piattaforma progettata per gestire classi virtuali, effettuare chiamate e scambiare messaggi tra utenti. Durante le fasi di sviluppo degli artefatti, si è rivelata cruciale per facilitare la comunicazione all'interno del team. Le videochiamate di gruppo hanno permesso discussioni approfondite e dettagliate, mentre la chat di gruppo ha semplificato lo scambio di file e informazioni. In definitiva, la piattaforma ha giocato un ruolo fondamentale nel mantenere costante la collaborazione tra i membri del team.

2.3.2 Discord

Discord è una piattaforma di comunicazione progettata inizialmente per i gamer, ma ormai ampiamente utilizzata da community e team di lavoro di vario tipo. Offre funzionalità come messaggistica istantanea, chiamate vocali e videochiamate, oltre alla possibilità di creare server dedicati con canali organizzati per argomenti specifici. Abbiamo utilizzato Discord creando un server dedicato, organizzato in modo da dividerci in due gruppi da due persone ciascuno. Questa struttura ha

permesso di passare rapidamente da un canale all’altro, facilitando le comunicazioni urgenti e istantanee.

2.3.3 GitHub

GitHub è una piattaforma di hosting per lo sviluppo software che si basa su Git, un sistema di controllo di versione distribuito. Questo strumento permette agli sviluppatori di monitorare e gestire le modifiche al codice sorgente durante il processo di sviluppo. Ha reso più semplice lo sviluppo parallelo, grazie all’impiego delle sue funzionalità principali:

- Repository: Un repository (o repo) è uno spazio di archiviazione per i file di un progetto, che include il codice sorgente, la documentazione e altri file correlati. I repository possono essere pubblici o privati;
- Branch: I branch rappresentano diramazioni del codice sorgente all’interno di un repository. Permettono agli sviluppatori di lavorare su nuove funzionalità o correzioni di bug senza interferire con il ramo principale;
- Pull Request: Una pull request consente di proporre modifiche al codice sorgente. Gli sviluppatori possono utilizzarla per discutere, rivedere e approvare le modifiche prima di unirle al ramo principale;

- Collaborazione: GitHub fornisce strumenti avanzati per la collaborazione tra i membri del team. È possibile commentare il codice, partecipare a discussioni sulle issue e collaborare in modo efficace sfruttando notifiche e menzioni.

2.3.4 Visual Paradigm

Visual Paradigm è un software per la modellazione e la gestione dei progetti, pensato per supportare un'ampia gamma di attività legate allo sviluppo software e alla progettazione di sistemi complessi. Ampiamente utilizzato per creare modelli UML (Unified Modeling Language), diagrammi di flusso, schemi di database e altre rappresentazioni grafiche, si distingue per la sua versatilità e le funzionalità avanzate. Grazie a queste caratteristiche, ci ha facilitato in particolare nella creazione di diagrammi UML, elemento fondamentale della documentazione del progetto.

2.3.5 GoodNotes

GoodNotes è un'applicazione progettata per prendere appunti digitali in modo organizzato e versatile. GoodNotes si è dimostrato uno strumento ideale per la gestione grafica e visiva delle attività del progetto. È stato particolarmente utile per creare diagrammi chiari e personalizzati, come quelli dei casi d'uso, e per migliorare la comunicazione e la collaborazione a distanza. Grazie a questa piattaforma, il team è stato

in grado di lavorare in modo più efficiente e organizzato, sfruttando al meglio le potenzialità del lavoro remoto.

2.3.6 Keynote

Keynote è un'applicazione sviluppata da Apple per creare presentazioni multimediali in modo professionale ed elegante. Si è rivelata fondamentale per la creazione di diapositive utilizzate durante le review.

2.4 Tecnologie e ambienti di sviluppo

2.4.1 Visual Studio Code

Visual Studio Code (o VS Code) è un editor di codice sorgente apprezzato per la sua leggerezza, versatilità e per le numerose funzionalità che lo rendono anche un IDE completo. Esso è stato utilizzato dal team di sviluppo per scrivere e compilare eventuali modifiche effettuate all'interno del codice. Tra i principali vantaggi di VS Code troviamo:

- Leggerezza e velocità: VS Code offre un'esperienza fluida e reattiva indipendentemente dall'hardware, grazie alla sua struttura leggera e ottimizzata;
- Estensibilità: Supporta quasi tutti i linguaggi di programmazione e può essere personalizzato con estensioni disponibili nel suo

Marketplace. Questo consente agli sviluppatori di adattare l'ambiente di lavoro alle loro esigenze specifiche, supportando diversi framework e tecnologie;

- Ambiente di sviluppo integrato: Oltre a essere un editor, VS Code funziona come un IDE, integrando strumenti essenziali come terminale, debugger e Git. Quest'ultimo risulta particolarmente utile per semplificare il version control.

2.4.2 Java Spring e Spring Boot

Java Spring e la sua estensione Spring Boot sono stati i principali framework utilizzati per lo sviluppo del progetto, grazie alla loro capacità di semplificare la creazione di applicazioni web robuste e scalabili. Spring Framework ha fornito strumenti fondamentali come la gestione automatica delle dipendenze e il modulo Spring MVC, che ha permesso di strutturare l'applicazione in modo chiaro, separando controller, logica applicativa e viste. Spring Boot ha ulteriormente velocizzato lo sviluppo grazie alla configurazione automatica, al supporto integrato per il deployment e alla possibilità di monitorare l'applicazione con strumenti dedicati. La sua semplicità ha eliminato la necessità di configurazioni complesse, rendendo il processo di sviluppo più rapido ed efficiente. Questo ha permesso al progetto di beneficiare di un framework flessibile e pronto per essere esteso.

2.4.3 Bootstrap

Per lo sviluppo dell’interfaccia utente, è stato utilizzato il framework Bootstrap, che ha semplificato la creazione di un design moderno, responsivo e coerente. Grazie alle sue classi predefinite e ai componenti già pronti, è stato possibile costruire rapidamente layout adattabili a diverse risoluzioni e dispositivi, migliorando l’esperienza utente. L’uso di Bootstrap ha ridotto significativamente il bisogno di scrivere codice CSS personalizzato, garantendo uno stile uniforme e un’interfaccia utente professionale. Integrato con il back-end sviluppato in Spring Boot, ha contribuito a offrire un sistema completo e ben strutturato.

2.4.4 Maven

Maven è uno strumento di gestione e automazione dei progetti, progettato principalmente per semplificare lo sviluppo di applicazioni Java. Attraverso il file di configurazione pom.xml, consente di gestire automaticamente le dipendenze, scaricandole da repository remoti, e di automatizzare fasi cruciali del ciclo di vita del software, come la compilazione, i test e il packaging. Promuove una struttura standardizzata del progetto, facilitando la collaborazione tra i membri del team. Grazie alla sua estensibilità tramite plugin, offre funzionalità aggiuntive come il controllo della qualità del codice e la generazione di documentazione. In sintesi, Maven garantisce un ambiente organizzato, efficiente e coerente per lo sviluppo software.

2.4.5 MongoDB Compass

MongoDB Compass è un’interfaccia grafica per MongoDB che consente di esplorare, analizzare e gestire i database in modo intuitivo. Con Compass, gli utenti possono visualizzare i dati memorizzati in MongoDB, eseguire query interattive, analizzare la struttura dei documenti e ottenere statistiche sui dati. Durante il progetto, abbiamo utilizzato MongoDB Compass per verificare che le nostre funzioni memorizzassero correttamente i dati all’interno del database. Questo strumento si è rivelato essenziale per validare il corretto funzionamento delle operazioni di scrittura e per garantire l’integrità dei dati memorizzati.

2.4.6 Docker Desktop

Docker Desktop è uno strumento essenziale per la gestione e l’esecuzione di applicazioni containerizzate. Fornisce un’interfaccia intuitiva per interagire con Docker Engine, semplificando la creazione, il deployment e la gestione dei container in ambienti locali. Nel contesto del nostro progetto, Docker Desktop è stato utilizzato per configurare e avviare i container dei vari microservizi, consentendo di replicare un ambiente di sviluppo coerente e isolato.

Capitolo 3

Requisito R5: Analisi preliminare

Il requisito assegnatoci, identificato come Requisito R5, prevede l'implementazione di una nuova funzionalità che consenta all'Admin di creare e gestire team di studenti, assegnare loro sfide o compiti con specifici vincoli temporali e monitorare il progresso delle attività. L'obiettivo principale del requisito è fornire agli Admin uno strumento per strutturare e organizzare in modo più efficace i team di studenti, rendendo il sistema più flessibile e semplice da utilizzare.

3.1 Osservazioni e problematiche

I nostri primi passi per comprendere meglio il requisito da implementare sono stati la consultazione delle documentazioni esistenti relative

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

al task T1. Tuttavia, abbiamo riscontrato diverse difficoltà legate alla poca chiarezza e incompletezza delle informazioni fornite, che hanno rallentato l'avvio del nostro lavoro. Durante questa fase preliminare, ci siamo dedicati all'installazione della web application in locale, ma anche qui abbiamo affrontato problemi significativi legati alla configurazione degli ambienti di sviluppo. In particolare, i path per JAVA_HOME e MAVEN_HOME non erano correttamente definiti, causando errori nell'esecuzione dei container Docker e ostacolando il corretto avvio del sistema. Dopo un'analisi approfondita, siamo riusciti a risolvere queste configurazioni e a procedere con lo sviluppo. Infine, abbiamo dovuto confrontarci con un problema legato alla struttura del codice nel container T1. L'HomeController, prima del refactoring effettuato da alcuni colleghi (vedi sezione 1.3.6), era difficilmente gestibile a causa della sua complessità e della scarsa separazione della logica di business. Durante i ricevimenti e le review con la professoressa, ci è stato suggerito di basarci sul lavoro di refactoring già avviato, che aveva migliorato la modularità del codice introducendo servizi dedicati per ogni funzionalità. Questa nuova organizzazione ci ha permesso di implementare il Requisito R5 in modo chiaro e manutenibile, senza compromettere la stabilità del sistema. Grazie a queste analisi e interventi, siamo riusciti a superare le difficoltà iniziali e a integrare con successo le nuove funzionalità richieste.

3.2 Analisi d’Impatto

Una delle sfide principali è stata capire dove implementare le nuove funzionalità legate alla gestione dei team. Dopo diverse review e confronti interni, abbiamo deciso di integrarle nel container T1, poiché è quello dedicato alla gestione amministrativa del sistema. Questa scelta ha richiesto un’attenta analisi della struttura del container per garantire una corretta integrazione con le logiche già presenti, evitando conflitti e mantenendo una coerenza complessiva. Un ulteriore problema affrontato è stato introdurre per la prima volta una chiamata REST tra i container T1 e T23. Nel container T1, infatti, non erano mai state implementate chiamate REST verso altri container, il che ha rappresentato una difficoltà significativa. Sebbene nel container T23 fosse già presente un’API REST per ottenere la lista degli utenti, è stato necessario comprendere come utilizzarla correttamente e integrarla con il sistema esistente. Abbiamo configurato la chiamata REST in T1 per inviare la richiesta all’API di T23 e gestire la risposta, integrandola nella logica di creazione dei team. Questo passaggio è stato fondamentale per garantire una comunicazione fluida tra i due container, migliorando l’interoperabilità senza la necessità di creare nuove API. All’interno del container T1 è presente un’altra chiamata REST al container T4, utilizzata per recuperare le statistiche relative alle partite giocate da un determinato giocatore. Queste statistiche vengono poi utilizzate per verificare se il giocatore ha effettivamente

superato la challenge.

In sintesi, i container impattati da queste modifiche sono T1, che gestisce le nuove funzionalità e le chiamate REST, T23, che fornisce l'API per recuperare la lista degli utenti, e T4, che fornisce le statistiche necessarie per la verifica delle challenge.

3.3 Analisi dei requisiti

La prima fase che abbiamo affrontato è stata l'analisi dei requisiti. In questa fase, ci siamo concentrati sulla modellazione delle nuove funzionalità, tra cui le operazioni CRUD per i Team, la gestione di aggiunta e rimozione di membri dai Team, e l'assegnazione delle sfide. Per raggiungere questi obiettivi, abbiamo esaminato i diagrammi esistenti nella documentazione del container T1 e avviato una fase di brainstorming per sviluppare idee utili alla creazione di nuovi diagrammi o alla modifica di quelli già presenti.

Nei sottoparagrafi successivi, analizzeremo nel dettaglio i risultati ottenuti in questa prima fase.

3.4 User Story

Con il susseguirsi delle iterazioni si è arrivati all'ideazione delle User Stories per stabilire gli obiettivi finali che si vogliono raggiungere. Ogni storia rappresenta un requisito di sistema e seguendo il paradigma SCRUM la struttura con la quale vengono descritte è:

- Come: descrive il tipo di utente coinvolto
- Vorrei: rappresenta il requisito/funzionalità di sistema
- In modo tale: indica l'obiettivo da raggiungere

Definizione delle user story relative alla Gestione del Team

- Come amministratore, voglio creare nuovi team, in modo tale da organizzare i membri in gruppi specifici per le challenge
- Come amministratore, voglio visualizzare un elenco completo dei team, in modo tale da avere una panoramica di quelli disponibili.
- Come amministratore, voglio visualizzare i dettagli di un singolo team, in modo tale da gestirne le informazioni e i membri.
- Come amministratore, voglio poter eliminare un team, in modo tale da rimuovere team non più necessari o obsoleti dal sistema.
- Come amministratore, voglio aggiungere un membro a un team, in modo tale da includere nuovi giocatori nel team.

- Come amministratore, voglio rimuovere un membro da un team, in modo tale da escludere giocatori non più coinvolti.

Definizione delle user story relative alla Gestione delle Challenge

- Come amministratore, voglio creare una nuova challenge, in modo tale da organizzare sfide specifiche per i team.
- Come amministratore, voglio visualizzare l'elenco completo delle challenge, in modo tale da avere una panoramica delle sfide in corso o completate.
- Come amministratore, voglio visualizzare i dettagli relativi ad una determinata challenge, in modo tale da avere una panoramica su di essa
- Come amministratore, voglio poter eliminare una challenge, in modo tale da rimuovere challenge non più necessarie o obsolete dal sistema.

3.5 Criteri di Accettazione

Successivamente abbiamo definito i criteri di accettazione seguendo il modello Given-When-Then in SCRUM che aiuta a definire chiaramente le funzionalità, stabilendo condizioni iniziali, azioni, risultati attesi ed eventuali condizioni aggiuntive (AND). Questo approccio allinea il lavoro del team alle aspettative dell'utente, garantendo chiarezza sui

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

requisiti e fornendo metriche per valutare il successo o il fallimento delle funzionalità. Ogni criterio assicura che una user story sia considerata completata con successo, evitando ambiguità e fainfendimenti. Di seguito vengono elencati i criteri di accettazione per ciascuna storia creata.

Creazione dei Team da parte dell'Admin Registrato

- GIVEN: L'Admin è autenticato e ha accesso alla Home Admin nella sezione di gestione dei team.
- WHEN: L'Admin inserisce i dettagli del nuovo team (nome, descrizione, ID leader e membri) e clicca sul pulsante aggiungi Team.
- THEN: Un nuovo team viene creato e salvato nel sistema.
- AND: L'elenco dei team nella sezione di gestione dei team viene aggiornato, e gli utenti possono visualizzare il nuovo team nella lista.

Operazione Read Team da parte dell'Admin Registrato

- GIVEN: L'Admin è autenticato e si trova nella sezione Lista Team ottenuta entrando prima nella sezione di gestione dei team della Home Admin.
- WHEN: L'Admin, attraverso un menù a tendina, sceglie il team che vuole visualizzare.

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

- THEN: Vengono mostrati i dettagli completi del team selezionato, inclusi nome, descrizione, ID leader, elenco dei membri e data di creazione.
- AND: L'Admin può visualizzare i membri e le sfide assegnate al team, con possibilità di esplorare ulteriori dettagli o azioni relative.

Delete Team da parte dell'Admin Registrato

- GIVEN: L'Admin è autenticato e si trova nella sezione Cancellare Team ottenuta entrando prima nella sezione di gestione dei team della Home Admin.
- WHEN: L'Admin seleziona un team dalla lista e clicca sul pulsante elimina Team, confermando l'azione nella finestra di dialogo.
- THEN: Il team selezionato viene eliminato dal sistema.
- AND: L'elenco dei team nella sezione di gestione dei team viene aggiornato, e il team eliminato non è più visibile nella lista.

Aggiungi Membro da parte dell'Admin Registrato

- GIVEN: L'Admin è autenticato e si trova nella sezione Aggiungi Membro ottenuta entrando prima nella sezione di gestione dei team della Home Admin.

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

- WHEN: L'Admin seleziona un team dal menu a tendina, attraverso un altro menu a tendina sceglie il membro da aggiungere e conferma l'operazione.
- THEN: Il membro selezionato viene aggiunto al team scelto nel sistema.
- AND: L'elenco dei membri del team selezionato viene aggiornato e mostrato nella sezione dei dettagli del team, includendo il nuovo membro.

Rimuovi Membro da parte dell'Admin Registrato

- GIVEN: L'Admin è autenticato e si trova nella sezione Aggiungi Membro ottenuta entrando prima nella sezione di gestione dei team della Home Admin.
- WHEN: L'Admin seleziona un team dal menu a tendina, attraverso un altro menu a tendina sceglie il membro da rimuovere e conferma l'operazione.
- THEN: Il membro selezionato viene rimosso dal team scelto nel sistema.
- AND: L'elenco dei membri del team selezionato viene aggiornato e mostrato nella sezione dei dettagli del team, escludendo il nuovo membro.

Visualizzazione della lista dei Team da parte dell'Admin Registrato

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

- GIVEN: L'Admin è autenticato e ha accesso alla Home Admin nella sezione di gestione dei team.
- WHEN: L'Admin accede alla sezione “Lista Team”.
- THEN: Il sistema recupera dal database l'elenco completo dei team disponibili.
- AND: I team vengono visualizzati sottoforma di elenco con le relative informazioni chiave come nome, descrizione, ID leader.

Create Challenge da parte dell'Admin Registrato

- GIVEN: L'Admin è autenticato e ha accesso alla Home Admin nella sezione di gestione delle challenges.
- WHEN: L'Admin inserisce i dettagli della nuova challenge (nome, descrizione, ID del team associato, ID del creatore, data di inizio e fine, tipo e condizione di vittoria) e clicca sul pulsante “Crea Challenge”.
- THEN: Il sistema valida i dati forniti, li converte in un documento MongoDB e salva il nuovo challenge nel database.
- AND: Il nuovo challenge appare nell'elenco delle challenge disponibili con tutte le informazioni chiave inserite.

View Challenge da parte dell'Admin Registrato

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

- GIVEN: L'Admin è autenticato e ha accesso alla Home Admin nella sezione di gestione delle challenge.
- WHEN: L'Admin accede alla sezione “Lista Challenge”.
- THEN: Il sistema interroga il database e recupera l'elenco completo dei challenge disponibili.
- AND: L'elenco delle challenge viene visualizzato sotto forma di tabella, con informazioni chiave come nome, descrizione, ID team, ID creatore, date di inizio e fine, stato, tipo e condizione di vittoria.

Criteri di accettazione per la visualizzazione di una singola challenge da parte dell'Admin Registrato:

- GIVEN: L'amministratore è autenticato e ha accesso alla sezione di gestione delle challenge.
- WHEN: L'amministratore seleziona una challenge specifica dall'elenco delle challenge disponibili.
- THEN: Il sistema visualizza i dettagli completi della challenge selezionata, inclusi il nome, descrizione, team, Creatore, data di inizio e data di fine, stato della challenge e i dettagli della condizione di vittoria
- AND: I dati sono presentati in modo chiaro e leggibile all'interno dell'interfaccia Admin.

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

Criteri di accettazione per l'eliminazione di una challenge da parte dell'Admin Registrato:

- GIVEN: L'amministratore è autenticato e ha accesso alla sezione dedicata alla gestione delle challenge.
- WHEN: L'amministratore seleziona una challenge dall'elenco e conferma l'operazione di eliminazione.
- THEN: La challenge viene rimossa dal sistema.
- AND: La challenge eliminata non appare più nell'elenco delle sfide disponibili.

3.6 Use Case Diagram aggiornato

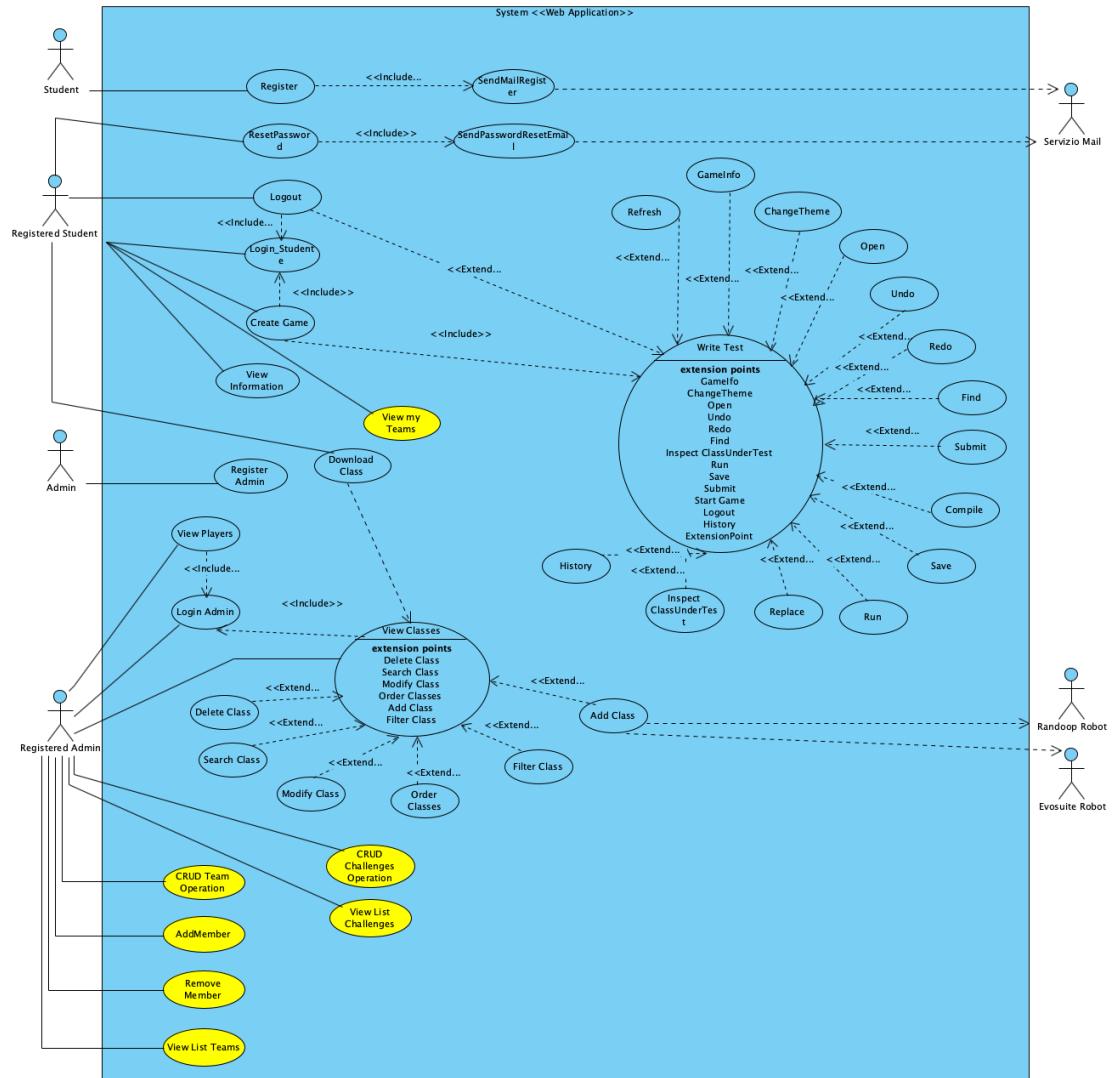


Figura 3.1: Diagramma dei casi d'uso aggiornato

In questo Diagramma dei Casi D'Uso (Figura 3.1) notiamo in giallo le funzionalità da noi implementate legate alla gestione dei team e delle challenges. A differenza di View Class, che utilizza un unico caso d'uso grande e complesso per gestire tutte le operazioni sulle classi, noi abbiamo deciso di suddividere le funzionalità legate alla gestione

dei team e dei challenge in più casi d'uso specifici e distinti. Questa scelta è stata motivata da diverse ragioni:

- Chiarezza e modularità: Separare le funzionalità in casi d'uso più piccoli e mirati rende il sistema più comprensibile, sia per chi lo sviluppa che per chi lo utilizza.
- Facilità di manutenzione: La suddivisione consente di intervenire su una funzionalità specifica senza rischiare di compromettere altre parti del sistema.
- Riutilizzabilità: Ogni caso d'uso rappresenta una funzionalità atomica che può essere riutilizzata in altri contesti o integrata con altre parti del sistema in futuro.
- Scalabilità: Con casi d'uso separati il sistema è più scalabile, poiché è possibile aggiungere nuove funzionalità senza complicare ulteriormente un unico caso d'uso monolitico.

3.7 Diagramma ER aggiornato

Il diagramma rappresenta il modello del database del container T1, includendo sia le entità originali sia le modifiche apportate per migliorare e ampliare le funzionalità. In particolare, sono state aggiunte

CAPITOLO 3. REQUISITO R5: ANALISI PRELIMINARE

alcune tabelle e relazioni, evidenziate in giallo, per supportare nuove funzionalità legate alla gestione dei team e delle challenge. Il sistema modellato pone l'Admin al centro delle operazioni di gestione: ogni admin può creare sia Challenge che Team, ma ciascun challenge e ciascun team possono essere associati a un solo admin. Questo garantisce una gestione chiara e centralizzata, in cui ogni attività o gruppo è sotto la responsabilità di un amministratore specifico. Una relazione fondamentale è quella tra Challenge e Team: una challenge può coinvolgere più team e, allo stesso tempo, un team può partecipare a più challenge. Questa relazione bidirezionale, gestita attraverso la tabella intermedia ChallengeTeam, offre una grande flessibilità. Nel modello, ogni User rappresenta un player generico e può essere associato, attraverso una relazione multi-a-molti, alle entità Team e Challenge. La tabella User è stata progettata per memorizzare i player estratti dalla chiamata REST effettuata al container T23.

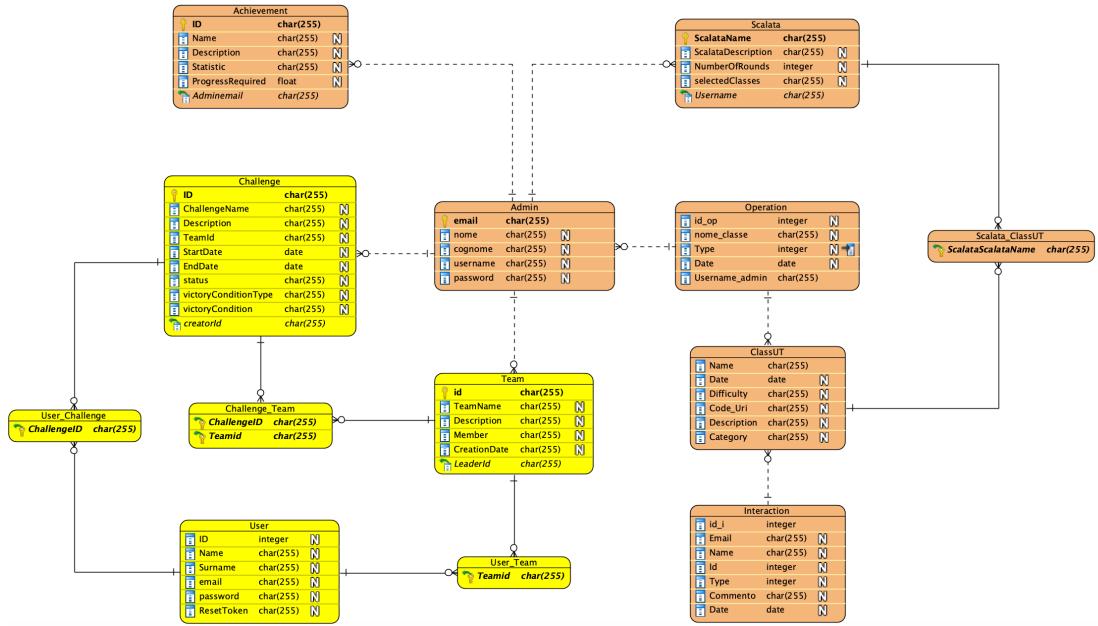


Figura 3.2: Diagramma ER di T1

3.8 Package Diagram

Il Package Diagram (Figura 3.3) illustra l'architettura dell'applicazione organizzata secondo il pattern Model-View-Controller (MVC), fornendo una visione chiara delle principali componenti e delle loro interazioni. Il Model rappresenta il cuore della logica applicativa e della gestione dei dati, includendo il collegamento con il Database MongoDB e un sistema di gestione file che interagisce con un File System Condiviso. La View è costituita dalle risorse statiche e dai template, che si occupano di presentare le informazioni all'utente in modo chiaro e accessibile. I Controller fungono da intermediari, orchestrando il flusso di dati tra la View, il Model e i Service, dove risiede la logica

aziendale che supporta tutte le operazioni dell'applicazione. Inoltre, il sistema interagisce con un'External API per estendere le funzionalità attraverso servizi esterni. Questa struttura permette all'applicazione di essere modulare, scalabile e facilmente manutenibile, grazie alla chiara suddivisione delle responsabilità tra i vari componenti.

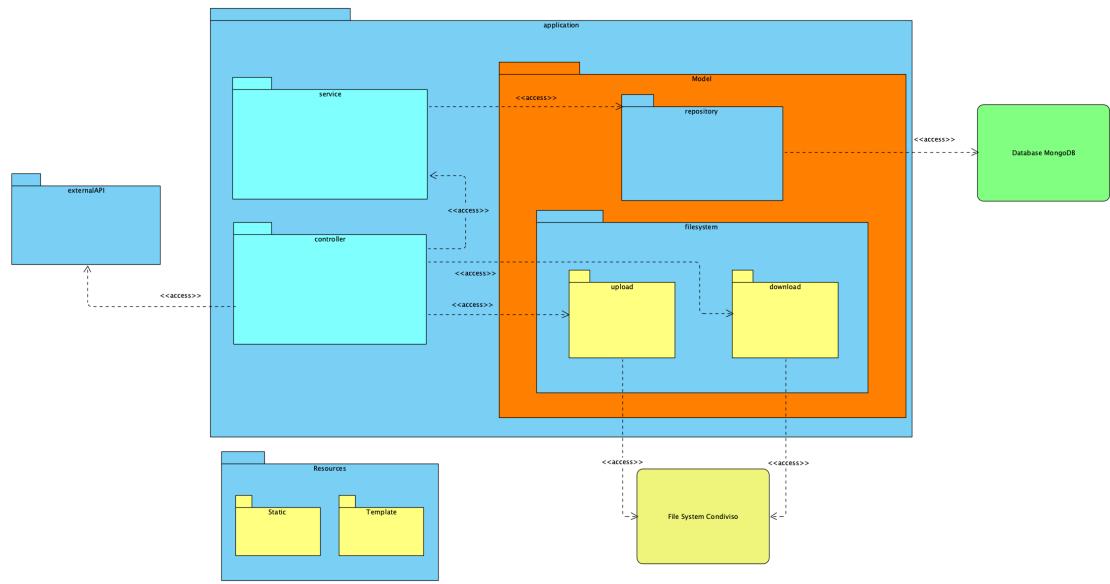


Figura 3.3: Package Diagram di T1

3.9 Component Diagram aggiornato

Nel Component Diagram aggiornato (Figura 3.4), gli elementi evidenziati in giallo rappresentano le nuove implementazioni introdotte per soddisfare il Requisito R5.

Sono stati aggiunti due nuovi service: TeamService, che gestisce operazioni come la creazione e gestione dei membri del team, e ChallengeService, responsabile della logica di business relativa alle sfide, come

la creazione e l’assegnazione.

Inoltre, sono state introdotte quattro nuove repository: TeamRepository e ChallengeRepository, che estendono il repository Mongo standard e implementano le operazioni CRUD di base; TeamSearchImpl e ChallengeSearchImpl, che aggiungono operazioni avanzate come la gestione personalizzata dei dati dei team e delle sfide. Queste aggiunte migliorano la modularità del sistema e separano chiaramente la logica di business dall’accesso ai dati, rendendo il sistema più scalabile e manutenibile.

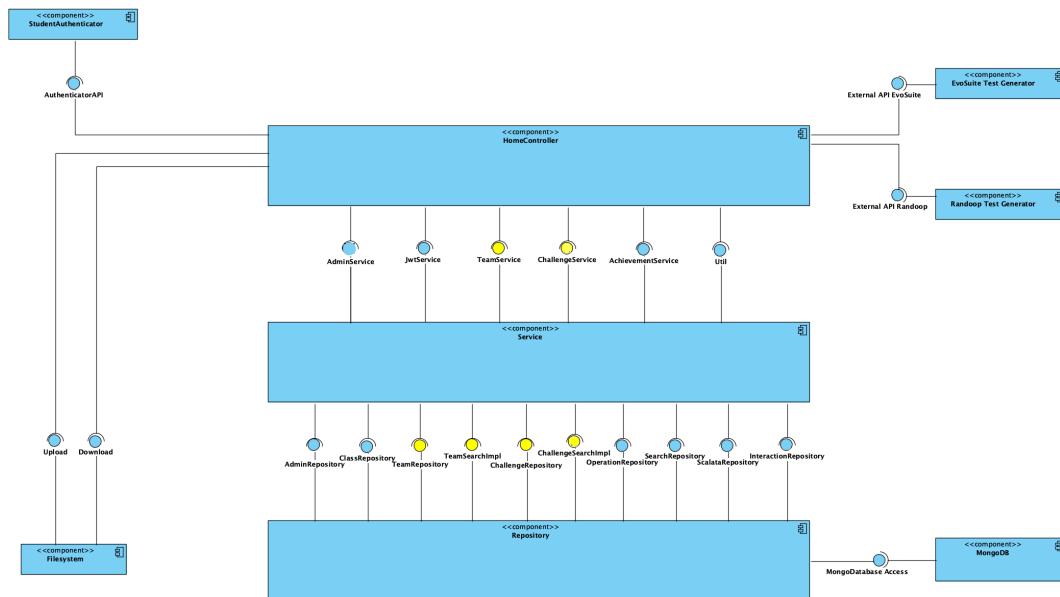


Figura 3.4: Component Diagram aggiornato di T1

Nel capitolo 4 vedremo più nel dettaglio i nuovi componenti.

Capitolo 4

Requisito R5: Nuove Implementazioni

4.1 Implementazione della Funzionalità di Gestione del Team

In questa sezione vengono descritti tutti i file di codice recentemente aggiunti o modificati nel container T1, relativi alle implementazioni per la gestione dei team. Per ciascuna implementazione, verranno mostrati gli screenshot del codice sorgente, accompagnati da una breve descrizione delle funzionalità implementate e del ruolo specifico del file all'interno del sistema. Questi aggiornamenti includono:

- Modelli: Rappresentazioni delle entità principali per garantire coerenza con il database MongoDB.

- Repository: Interfacce per l'accesso diretto ai dati e la gestione delle operazioni CRUD.
- Servizi: Contengono la logica di business per orchestrare le operazioni e integrare i diversi componenti del sistema.
- Controller: Gestione delle API REST per il flusso di dati tra frontend e backend.
- File per il frontend, come moduli HTML e script JavaScript, per un'interfaccia utente più intuitiva e funzionale.

4.1.1 Team.java

Le prime modifiche effettuate riguardano il Model; in particolare, per garantire la coerenza dei dati salvati abbiamo dovuto aggiungere la classe Team presente nel package Model per adattarla alla struttura del Database di T1. La classe Team include i seguenti campi: *id*, che rappresenta l'identificatore unico del documento gestito automaticamente da MongoDB; *teamName*, che memorizza il nome del team; *description*, che contiene una breve descrizione del team; *leaderId*, che identifica il leader o amministratore del team; *member*, una lista di stringhe che rappresentano i membri del team; e *creationDate*, che registra la data di creazione del team come stringa. La classe Team include un costruttore che consente di creare un oggetto inizializzando i campi principali come *teamName*, *description*, *leaderId*, *member* e

creationDate, mentre il campo id è gestito automaticamente da MongoDB (Vedi figura 4.1).

```
@Document(collection = "Team")
public class Team {
    @Id
    private String id;
    private String teamName;
    private String description;
    private String leaderId; // ID dell'amministratore o leader del team
    private List<String> member; // Lista dei membri del team
    private String creationDate; // Data di creazione del team

    public Team(String teamName, String description, String leaderId, List<String> member, String creationDate) {
        this.teamName = teamName;
        this.description = description;
        this.leaderId = leaderId;
        this.member = member;
        this.creationDate = creationDate;
    }
}
```

Figura 4.1: Classe Team nel Model di T1

I metodi getter e setter, aggiunti per ogni campo, forniscono un modo standard per accedere e modificare i valori degli attributi dell'oggetto, garantendo l'incapsulamento dei dati. Il metodo `toString` (Fig 4.2) è implementato per restituire una rappresentazione testuale dell'oggetto `Team`, elencando i valori dei suoi campi in un formato leggibile, utile per scopi di debug o logging.

```
@Override
public String toString() {
    return "Team{" +
        "teamName='" + teamName + '\'' +
        ", description='" + description + '\'' +
        ", leaderId='" + leaderId + '\'' +
        ", member=" + member +
        ", creationDate='" + creationDate + '\'' +
        '}';
}
```

Figura 4.2: Metodo `toString()` nella classe `Team`

4.1.2 TeamSearchImpl.java

La classe TeamSearchImpl è stata creata per gestire operazioni avanzate e personalizzate sui documenti della collezione Team nel database MongoDB, che non possono essere facilmente realizzate con i metodi standard di un repository Spring, come quelli forniti da MongoRepository. Di seguito elenchiamo due funzioni principali che verranno richiamate anche in TeamService (Vedi sezione 4.1.3).

```
public void addTeam(Team team) {
    MongoDatabase database = client.getDatabase("manvsclass");
    MongoCollection<Document> collection = database.getCollection("Team");

    // Converti la lista di membri in un formato compatibile con MongoDB
    List<String> members = team.getMember(); // Assumi che 'getMember' restituisca una lista di stringhe

    Document teamDoc = new Document()
        .append("teamName", team.getTeamName()) // Usato teamName come identificatore
        .append("description", team.getDescription())
        .append("leaderId", team.getLeaderId())
        .append("member", members) // Inserisci direttamente l'array dei membri
        .append("creationDate", team.getCreationDate());

    collection.insertOne(teamDoc);
}
```

Figura 4.3: addTeam in TeamSearchImpl

La funzione addTeam (Figura 4.3) prende in input un oggetto Team e lo inserisce nella collezione Team del database MongoDB. Per farlo, converte l'oggetto in un documento MongoDB, includendo informazioni come il nome del team, la descrizione, l'ID del leader, la lista dei membri e la data di creazione. Una volta trasformato, il documento viene aggiunto alla collezione con il metodo insertOne. Questo consente di creare nuovi team nel database in modo strutturato e persistente. La funzione addMemberToTeam (Figura 4.4) permette di aggiungere un membro specifico a un team esistente. Utilizza il nome del team come filtro per individuare il documento corrispondente nel databa-

```

// Metodo per aggiungere un membro al team
public void addMemberToTeam(String teamName, String memberId) {
    MongoDatabase database = client.getDatabase("manvsclass");
    MongoCollection<Document> collection = database.getCollection("Team");

    // Filtra il documento del team
    Bson filter = Filters.eq(fieldName:"teamName", value:teamName);

    // Aggiungi il nuovo membro alla lista dei membri
    Bson update = new Document("$addToSet", new Document("members", memberId));

    // Aggiorna il documento del team
    collection.updateOne(filter, update);
}

```

Figura 4.4: addMemberToTeam in TeamSearchImpl

se. Una volta trovato, applica l'operazione addToSet per aggiungere l'ID del nuovo membro alla lista dei membri del team, evitando duplicati. Infine, il documento aggiornato viene salvato nella collezione Team, garantendo che il nuovo membro sia correttamente aggiunto senza modificare gli altri dati.

4.1.3 TeamService.java

Il TeamService è stato introdotto con l'obiettivo di centralizzare e orchestrare la logica di business relativa alla gestione dei team, migliorando la coerenza complessiva con il refactoring del container T1. Questo approccio consente di separare la logica di business dal controller, rendendo il codice più leggibile, manutenibile e modulare. Inoltre, garantisce un'architettura coerente e scalabile, supportando l'introduzione di nuove funzionalità come la gestione avanzata dei membri e le interazioni con altre parti del sistema. Un aspetto fondamentale è l'integrazione della validazione dei token JWT per ogni operazione, assicurando che solo gli utenti autorizzati possano interagire con i dati.

Attraverso queste due chiamate:

@Autowired private TeamRepository teamRepository

@Autowired private TeamSearchImpl searchRepository

TeamService può gestire le operazioni CRUD di base sui documenti Team nel database e iniettare automaticamente un’istanza della classe TeamSearchImpl. Questo significa che la classe può utilizzare i metodi personalizzati definiti in TeamSearchImpl per interagire con il database MongoDB. Infine, il TeamService è progettato per gestire le interazioni con componenti esterni, come le chiamate REST a container come T2-3, facilitando operazioni cruciali come il recupero degli elenchi di studenti.

Tra le operazioni implementate nel TeamService, spiccano due funzionalità fondamentali: createTeam e getAllTeamsAsHtml.

```
public ResponseEntity<Team> createTeam(Team team, String jwt) {
    if (!jwtService.isJwtValid(jwt)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
    if (teamRepository.existsByTeamName(team.getTeamName())) {
        return ResponseEntity.status(HttpStatus.CONFLICT).body(null); // Team already exists
    }
    searchRepository.addTeam(team);
    return ResponseEntity.status(HttpStatus.CREATED).body(team);
}
```

Figura 4.5: createTeam situato in TeamService

La funzione createTeam (Figura 4.5) è stata progettata per consentire la creazione di nuovi team nel sistema in modo sicuro e controllato. Prima di procedere con l’operazione, il servizio verifica la validità del token JWT dell’utente, assicurandosi che solo gli amministratori autorizzati possano eseguire questa azione. Successivamente, utilizza la

funzione existsByTeamName di MongoDB per controllare se il nome del team è già presente nel database, prevenendo così eventuali conflitti con i dati esistenti. Se tutte le verifiche vengono superate con successo, il team viene aggiunto al database tramite il repository, garantendo un'operazione fluida e sicura. Alla fine, il servizio restituisce una risposta chiara che indica il risultato dell'operazione, fornendo al chiamante tutte le informazioni necessarie. Il risultato dell'operazione può essere un 201 (Created) se il team viene creato con successo, un 401 (Unauthorized) se il token JWT fornito non è valido o un 409 (Conflict) se esiste già un team con lo stesso nome nel database, impedendo la creazione per evitare conflitti.

```

public ResponseEntity<String> getAllTeamsAsHtml(String jwt) {
    // Controlla se il JWT è valido
    if (!jwtService.isJwtValid(jwt)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Accesso non autorizzato");
    }

    // Recupera tutti i team dal repository
    List<Team> teams = teamRepository.findAll();

    // Genera il contenuto HTML per il corpo della tabella
    StringBuilder htmlBuilder = new StringBuilder();
    for (Team team : teams) {
        htmlBuilder.append(str:"<tr>")
            .append(str:"<td>").append(team.getTeamName()).append(str:"</td>")
            .append(str:"<td>").append(team.getDescription()).append(str:"</td>")
            .append(str:"<td>").append(team.getLeaderId()).append(str:"</td>")
            .append(str:"<td>").append(String.join(delimiter:", ", team.getMember())).append(str:"</td>")
            .append(str:"</tr>");
    }
    return ResponseEntity.ok(htmlBuilder.toString());
}

```

Figura 4.6: getAllTeamsAsHtml situato in TeamService

La funzione getAllTeamsAsHtml (Figura 4.6), invece, si concentra sulla generazione dinamica di contenuti HTML. Questa operazione consente di recuperare tutti i team presenti nel sistema e di convertirli in una stringa HTML formattata, ideale per essere utilizzata direttamente in una vista o in una risposta lato frontend.

Anche qui, la validità del token JWT è una precondizione indispensabile, garantendo che l'accesso ai dati sia riservato agli utenti autorizzati. Il risultato è un contenuto HTML che elenca i dettagli di ciascun team, pronto per essere visualizzato in una tabella o in un'interfaccia utente.

Altre due funzioni importantissime sono quelle di getStudentsList e addMemberToTeam.

```
public ResponseEntity<String> getStudentsList(String jwt) {
    String t2_Url = "http://t23-g1-app-1:8080/students_list";
    List<User> allStudents;

    try {
        HttpEntity<Void> entity = jwtService.createJwtRequestEntity(jwt);

        ResponseEntity<List<User>> response = restTemplate.exchange(
            t2_Url,
            HttpMethod.GET,
            entity,
            new ParameterizedTypeReference<List<User>>() {});
    }

    if (response.getStatusCode() != HttpStatus.OK) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Error retrieving students from T2-3");
    }

    allStudents = response.getBody();
    return ResponseEntity.ok(allStudents);
} catch (Exception e) {
    return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error connecting to T2-3");
}
```

Figura 4.7: getStudentsList situato in TeamService

La funzione getStudentsList (Figura 4.7) è stata progettata per recuperare l'elenco degli studenti da un container esterno, il T2-3, rappresentando una nuova implementazione unica all'interno del container T1. Questa chiamata al container T2-3, mai utilizzata prima, è stata sviluppata appositamente per soddisfare il requisito di integrazione con un sistema esterno, fornendo un'importante espansione delle funzionalità del servizio.

Come al solito, in ogni funzione viene effettuata una verifica preliminare per garantire che l'operazione possa essere eseguita

correttamente e in sicurezza. Una volta confermata l'autorizzazione, viene utilizzato il RestTemplate per inviare una richiesta HTTP GET al servizio students list del container T2-3. Il token JWT viene incluso come intestazione della richiesta, garantendo che il container T2-3 possa convalidare l'autenticità dell'operazione. La risposta del container T2-3 viene quindi analizzata: se l'elenco degli studenti viene recuperato correttamente, viene restituito con uno stato di successo; altrimenti, vengono gestiti eventuali errori, come problemi di connessione o assenza di dati, con risposte adeguate che segnalano la natura dell'errore. La funzione addMemberToTeam (Figura 4.8)

```

public ResponseEntity<String> addMemberToTeam(String teamName, String memberId, String jwt) {
    // Verifica la validità del token JWT
    if (!jwtService.isJwtValid(jwt)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Unauthorized");
    }

    // Recupera il team dal repository
    Team team = searchRepository.findTeamByName(teamName);

    // Verifica se il membro è già nel team
    if (team.getMembers().contains(memberId)) {
        return ResponseEntity.status(HttpStatus.CONFLICT).body("Membro già nel team: " + memberId);
    }

    // Aggiungi il membro al team
    try {
        searchRepository.addMemberToTeam(teamName, memberId);
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error updating the team with member " + memberId);
    }

    return ResponseEntity.ok(body:"Studente aggiunto con successo!");
}

```

Figura 4.8: addMemberToTeam situato in TeamService

consente di aggiungere un nuovo membro a un team specifico, garantendo sicurezza e controllo sull'integrità dei dati. Dopo aver verificato il token. Se il token è valido, la funzione recupera il team associato al nome fornito tramite il searchRepository. Una volta ottenuto il team, controlla se il membro è già presente nella lista dei membri, evitando duplicati. Se il membro è già nel team, restituisce

una risposta che informa del conflitto e blocca l'operazione. In caso contrario, tenta di aggiungere il membro utilizzando il metodo dedicato del repository. L'intera operazione di aggiornamento è gestita in modo sicuro con un blocco try-catch per catturare eventuali errori durante l'interazione con il database. Se si verifica un errore, restituisce un messaggio adeguato con uno stato di errore interno del server. Quando l'operazione viene completata con successo, restituisce una risposta positiva indicando che il membro è stato aggiunto correttamente.

Questa funzione addMemberToTeam (Figura 4.9) è pensata per aggiungere più membri contemporaneamente a un team durante la fase di creazione. La sua particolarità sta proprio qui: viene utilizzata quando un team sta nascendo e c'è bisogno di inserire subito un gruppo di membri in un'unica operazione. È diverso dall'altra funzione, che ha lo stesso nome ma accetta parametri differenti: quella serve ad aggiungere un solo membro alla volta, ed è utilizzata dopo che il team è già stato creato, in una sezione apposita lato frontend. La funzione inizia verificando il token JWT per assicurarsi che l'operazione sia autorizzata, cioè che l'Admin abbia effettuato correttamente l'accesso. Superato questo controllo, procede a contattare un altro sistema, il container T2-3, che contiene la lista aggiornata di tutti gli studenti disponibili. Una chiamata viene fatta a questo container per recuperare l'elenco completo degli

CAPITOLO 4. REQUISITO R5: NUOVE IMPLEMENTAZIONI

```
public ResponseEntity<String> addMemberToTeam(String teamName, List<String> selectedMemberIds, String jwt) {
    // Verifica la validità del token JWT
    if (!jwtService.isJwtValid(jwt)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Unauthorized");
    }

    // Chiamata al container T2-3 per ottenere tutti gli studenti
    String t2_3Url = "http://t23-g1-app-1:8080/students_list"; // URL per ottenere tutti gli studenti
    List<User> allStudents;

    try {
        // Crea l'header con il JWT usando il metodo di JwtService
        HttpEntity<Void> entity = jwtService.createJwtRequestEntity(jwt); // Qui non è necessario fare altro, è già un HttpEntity

        // Esegui la chiamata GET per ottenere gli studenti
        ResponseEntity<List<User>> response = restTemplate.exchange(
            t2_3Url,
            HttpMethod.GET,
            entity,
            new ParameterizedTypeReference<List<User>>() {}
        );

        if (response.getStatusCode() != HttpStatus.OK) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Error retrieving students from T2-3");
        }

        allStudents = response.getBody();
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error connecting to T2-3");
    }

    // Visualizza tutti gli studenti e aggiungi i selezionati al team in MongoDB (T1)
    for (String memberId : selectedMemberIds) {
        // Verifica che il membro selezionato esista tra gli studenti di T2-3
        boolean memberFound = allStudents.stream()
            .anyMatch(student -> student.getID().equals(Integer.parseInt(memberId)));

        if (!memberFound) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("User not found in T2-3: " + memberId);
        }

        // Aggiungi il membro al team in MongoDB (T1)
        try {
            searchRepository.addMemberToTeam(teamName, memberId);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error updating the team with member " + memberId);
        }
    }
}

return ResponseEntity.ok(body:"Members added successfully");
}
```

Figura 4.9: addMemberToTeam durante la funzione createTeam

studenti. Una volta ottenuto questo elenco, la funzione verifica che ogni membro nella lista passata dal frontend esista effettivamente nel sistema degli studenti. Se anche solo un membro non viene trovato, la funzione interrompe l'operazione e restituisce un errore, specificando quale ID non è stato trovato. Se invece tutti i membri sono validi, li aggiunge uno per uno al team nel database principale (T1), utilizzando un metodo dedicato del repository.

4.1.4 TeamRepository.java

Il file TeamRepository definisce un’interfaccia che funge da collegamento tra l’applicazione e il database MongoDB, consentendo di gestire le operazioni di persistenza per la classe Team. Estendendo MongoRepository, eredita metodi standard per operazioni CRUD.

Inoltre, aggiunge un metodo personalizzato:

boolean existsByTeamName(String teamName)

Questo metodo, utilizzato nella funzione create team (Vedi figura 4.1), verifica l’esistenza di un team in base al suo nome, garantendo unicità e prevenendo conflitti durante la creazione di nuovi team.

4.2 Implementazione della Funzionalità di Gestione delle Challenges

In questa sezione vengono descritti tutti i file di codice recentemente aggiunti o modificati nel container T1, relativi all’implementazione della funzionalità di gestione delle challenges. Per ciascuna implementazione, verranno mostrati gli screenshot del codice sorgente, accompagnati da una breve descrizione delle funzionalità implementate e del ruolo specifico del file all’interno del sistema.

4.2.1 Challenge.java

Questo file definisce la classe Challenge (Figura 4.10), utilizzata per rappresentare una sfida nell'applicazione. La classe è annotata con @Document per indicare che corrisponde a un documento nella collezione Challenge del database MongoDB. Ogni istanza della classe contiene informazioni dettagliate sulla sfida, tra cui un identificativo univoco, il nome della sfida, una descrizione, l'ID del team associato, l'ID del creatore, le date di inizio e fine, lo stato attuale, il tipo di condizione di vittoria e il dettaglio della condizione di vittoria.

```
@Document(collection = "Challenge")
public class Challenge {
    @Id
    private String id;
    private String challengeName; // Nome della challenge
    private String description; // Descrizione della challenge
    private String teamId; // ID del team assegnato alla challenge
    private String creatorId; // ID dell'utente o amministratore che ha creato la challenge
    private String startDate; // Data di inizio della challenge
    private String endDate; // Data di fine della challenge
    private String status; // Stato della challenge (es. "In Progress", "Completed", "Pending")
    private VictoryConditionType victoryConditionType; // Tipo di condizione di vittoria
    private String victoryCondition; // Dettaglio della condizione di vittoria (es. numero di partite)

    public Challenge(String challengeName, String description, String teamId, String creatorId,
                     String startDate, String endDate, String status, VictoryConditionType victoryConditionType, String victoryCondition) {
        this.challengeName = challengeName;
        this.description = description;
        this.teamId = teamId;
        this.creatorId = creatorId;
        this.startDate = startDate;
        this.endDate = endDate;
        this.status = status;
        this.victoryConditionType = victoryConditionType;
        this.victoryCondition = victoryCondition;
    }
}
```

Figura 4.10: Classe Challenge nel Model di T1

La classe include un costruttore per creare oggetti con tutti gli attributi necessari, metodi getter e setter per accedere e modificare i dati, e un metodo `toString` sovrascritto per fornire una rappresentazione leggibile dell'oggetto. La struttura della classe consente una facile integrazione con MongoDB, rendendo possibile la

gestione e la manipolazione dei dati relativi alle sfide in modo efficiente.

Facendo un focus sull'attributo victoryConditionType, questo è implementato come un enum (Figura 4.11) che rappresenta i diversi tipi di condizioni di vittoria possibili per una sfida. L'utilizzo di un enum garantisce una maggiore chiarezza e sicurezza tipologica, limitando i valori accettati a quelli definiti. Attualmente, l'enum include il tipo GAMES_PLAYED, che indica che per completare la sfida è necessario giocare un determinato numero di partite. Questa struttura rende il sistema flessibile ed estendibile, consentendo di aggiungere facilmente nuovi tipi di condizioni di vittoria senza introdurre complessità aggiuntive o alterare la logica esistente.

```
package com.groom.mansclass.model;

public enum VictoryConditionType {
    GAMES_PLAYED // Gioca un certo numero di partite
    // Altri tipi potranno essere aggiunti qui in futuro
}
```

Figura 4.11: VictoryConditionType Enum

4.2.2 ChallengeSearchImpl.java

La classe ChallengeSearchImpl è stata progettata per gestire operazioni avanzate e personalizzate sui documenti della collezione Challenge nel database MongoDB, andando oltre le funzionalità

standard fornite dai repository di Spring, come quelli basati su MongoRepository. La funzione addChallenge (Figura 4.12) prende in input un oggetto Challenge e lo inserisce nella collezione Challenge del database. Per farlo, l'oggetto viene convertito in un documento MongoDB (Document), includendo informazioni come il nome della challenge, la descrizione, l'ID del team, l'ID del creatore, le date di inizio e fine, lo stato, il tipo di condizione di vittoria e i relativi dettagli. Una volta trasformato in documento, il metodo insertOne viene utilizzato per aggiungerlo alla collezione. Questo approccio consente di inserire nuove challenge nel database in modo strutturato, garantendo che tutte le informazioni necessarie siano salvate correttamente e pronte per essere utilizzate da altre parti del sistema.

```
public void addChallenge(Challenge challenge) {
    MongoDatabase database = client.getDatabase("mansclass");
    MongoCollection<Document> collection = database.getCollection("Challenge");

    Document challengeDoc = new Document()
        .append("challengeName", challenge.getChallengeName())
        .append("description", challenge.getDescription())
        .append("teamId", challenge.getTeamId())
        .append("creatorId", challenge.getCreatorId())
        .append("startDate", challenge.getStartDate())
        .append("endDate", challenge.getEndDate())
        .append("status", challenge.getStatus())
        .append("victoryConditionType", challenge.getVictoryConditionType().toString())
        .append("victoryCondition", challenge.getVictoryCondition());
}

    collection.insertOne(challengeDoc);
}
```

Figura 4.12: addChallenge situato in ChallengeSearchImpl

4.2.3 ChallengeService.java

Il ChallengeService è un componente fondamentale che implementa la logica di business per la gestione delle sfide nel sistema. Questo

servizio è responsabile di orchestrare operazioni complesse che coinvolgono sia la creazione e il monitoraggio delle sfide, sia l’interazione con altri container tramite chiamate REST. Grazie alla modularità offerta da questa classe, il sistema garantisce un’elevata manutenibilità, separando nettamente la logica di business dall’accesso ai dati. Un aspetto fondamentale è l’integrazione della validazione dei token JWT per ogni operazione, assicurando che solo gli utenti autorizzati possano interagire con i dati. Utilizza il pattern di iniezione delle dipendenze per accedere ai dati e gestire operazioni avanzate attraverso i repository definiti di seguito:

@Autowired private ChallengeSearchImpl searchRepository;

@Autowired private ChallengeRepository challengeRepository;

Di seguito vengono descritte tre funzioni principali implementate nel servizio: createChallenge, getPlayerGames e isChallengeCompleted. La funzione createChallenge consente all’amministratore di creare una nuova sfida e aggiungerla al database.

La funzione createChallenge (Figura 4.13) è progettata per consentire all’admin di creare una nuova sfida e salvarla nel database. Il processo inizia con una verifica della validità del token JWT fornito nella richiesta, per garantire che l’operazione sia eseguita da un utente autenticato. Successivamente, vengono effettuati controlli preliminari sul nome della sfida, assicurandosi che non esista già un’altra sfida con lo stesso nome nel database, prevenendo così

```

public ResponseEntity<Challenge> createChallenge(Challenge challenge, String jwt) {
    if (!jwtService.isJwtValid(jwt)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }

    if (challengeRepository.existsById(challenge.getChallengeName())) {
        return ResponseEntity.status(HttpStatus.CONFLICT).body(null); // La challenge esiste già
    }

    // Aggiungi il tipo di condizione di vittoria e la condizione
    if (challenge.getVictoryConditionType() == null || challenge.getVictoryCondition() == null) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null); // Parametri mancanti
    }

    searchRepository.addChallenge(challenge);
    return ResponseEntity.status(HttpStatus.CREATED).body(challenge);
}

```

Figura 4.13: createChallenge situato in ChallengeService

duplicati. Un ulteriore passo fondamentale è la validazione dei parametri necessari alla creazione della sfida. In particolare, vengono controllati il tipo di condizione di vittoria e il relativo valore associato, assicurandosi che siano entrambi presenti e corretti. Se uno di questi parametri non è valido, la funzione restituisce un errore, impedendo l'inserimento di dati incompleti o errati nel sistema. Una volta superate tutte le validazioni, la sfida viene salvata nel database utilizzando il repository ChallengeSearchImpl, che gestisce l'inserimento del documento nella collezione MongoDB. Al termine, la funzione restituisce una risposta con lo stato CREATED, confermando che la sfida è stata aggiunta correttamente. Questo flusso garantisce che solo sfide valide e ben definite vengano salvate, mantenendo l'integrità e l'affidabilità del sistema.

La funzione getPlayerGames (Figura 4.14) è progettata per recuperare, tramite una chiamata REST, la lista delle partite giocate da un determinato giocatore. Questo processo è fondamentale per

CAPITOLO 4. REQUISITO R5: NUOVE IMPLEMENTAZIONI

```
public ResponseEntity<?> getPlayerGames(int playerId, String jwt) {
    try {
        // Prepara l'entity con l'header JWT
        HttpEntity<Void> entity = jwtService.createJwtRequestEntity(jwt);

        // Costruisce l'URL per la richiesta REST
        String url = "http://t4-g18-app-1:3000/games/player/" + playerId;

        // Esegue la chiamata REST
        ResponseEntity<List<Map<String, Object>>> response = restTemplate.exchange(
            url,
            HttpMethod.GET,
            entity,
            new ParameterizedTypeReference<List<Map<String, Object>>>() {}
        );

        // Restituisce i dati se la chiamata ha successo
        if (response.getStatusCode().is2xxSuccessful()) {
            return ResponseEntity.ok(response.getBody());
        } else {
            return ResponseEntity.status(response.getStatusCode()).body("Error retrieving games from T4");
        }
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error connecting to T4: " + e.getMessage());
    }
}
```

Figura 4.14: getPlayerGames situato in ChallengeService

consentire al sistema di accedere ai dati provenienti dal container T4, integrandoli nelle logiche del backend. La funzione inizia creando un’`HttpEntity` contenente un token JWT nell’header della richiesta. Questo token è essenziale per autenticare la chiamata REST, garantendo che solo utenti autorizzati possano accedere ai dati. Successivamente, costruisce dinamicamente l’URL dell’endpoint del container T4, includendo l’ID del giocatore come parametro per specificare la richiesta. La chiamata REST viene eseguita utilizzando il metodo GET attraverso il `restTemplate.exchange()`, che invia la richiesta all’endpoint e riceve come risposta una lista di partite in formato JSON. Una volta ricevuti i dati, la funzione verifica lo stato della risposta. Se la chiamata è andata a buon fine, restituisce i dati delle partite al chiamante. In caso di errore, come un problema di connessione o un’operazione non autorizzata, la funzione restituisce

un messaggio di errore adeguato.

```

public boolean isChallengeCompletedByMember(Challenge challenge, String playerName, String jwt) {
    LocalDate challengeStartDate = LocalDate.parse(challenge.getStartDate(), DateTimeFormatter.ISO_DATE);
    LocalDate challengeEndDate = LocalDate.parse(challenge.getEndDate(), DateTimeFormatter.ISO_DATE);
    String victoryCondition = challenge.getVictoryCondition();
    VictoryConditionType type = challenge.getVictoryConditionType();

    try {
        // Recupera le partite filtrate per il giocatore specifico
        ResponseEntity<List<Map<String, Object>>> response = getPlayerGames(playerName, jwt);

        if (response.getStatusCode().is2xxSuccessful() && response.getBody() != null) {
            List<Map<String, Object>> gamesInRange = response.getBody().stream()
                .filter(game -> {
                    String startedAtStr = (String) game.get(key:"startedAt");
                    if (startedAtStr != null) {
                        LocalDate startedAt = LocalDate.parse(startedAtStr, DateTimeFormatter.ISO_DATE);
                        return !startedAt.isBefore(challengeStartDate) && !startedAt.isAfter(challengeEndDate);
                    }
                    return false;
                })
                .collect(Collectors.toList());

            // Verifica la Victory Condition
            if (type == VictoryConditionType.GAMES_PLAYED) {
                int requiredGames = Integer.parseInt(victoryCondition);
                return gamesInRange.size() >= requiredGames;
            }

            throw new UnsupportedOperationException("Tipo di condizione non supportato: " + type);
        }
    } catch (Exception e) {
        System.err.println("Errore durante la verifica della challenge: " + e.getMessage());
    }
}

return false;
}

```

Figura 4.15: isChallengeCompletedByMember situato in ChallengeService

La funzione isChallengeCompletedByMember (Figura 4.15), verifica se un giocatore specifico ha completato una determinata challenge.

Per farlo, il metodo utilizza le date di inizio e fine della challenge, il tipo di condizione di vittoria e il numero di partite necessarie.

Inizialmente, la funzione recupera la lista delle partite giocate dal giocatore tramite una chiamata a un servizio esterno. Dopo aver verificato che la risposta sia valida, filtra le partite basandosi sulla data di inizio e fine della challenge: solo le partite giocate all'interno di questo intervallo temporale vengono prese in considerazione.

Successivamente, il metodo confronta il numero di partite giocate con la condizione di vittoria della challenge. Se il tipo di condizione è “GAMESPLAYED”, controlla che il numero di partite giocate sia

almeno pari al valore richiesto dalla condizione. Se questa verifica ha successo, la funzione restituisce true, indicando che il giocatore ha completato la challenge. In caso contrario, restituisce false. Infine, la funzione gestisce eventuali errori durante l'operazione, come problemi con il formato dei dati o con il tipo di condizione, stampando un messaggio di errore in caso di fallimento.

```
public boolean isChallengeCompletedByTeam(Challenge challenge, Team team, String jwt) {
    List<String> teamMembers = team.getMember(); // Recupera la lista dei nomi dei membri (String)

    if (teamMembers == null || teamMembers.isEmpty()) {
        throw new IllegalArgumentException("Il team non ha membri.");
    }

    for (String memberName : teamMembers) {
        try {
            // Verifica se il membro (identificato dal nome) ha completato la challenge
            boolean isMemberCompleted = isChallengeCompletedByMember(challenge, memberName, jwt);

            if (!isMemberCompleted) {
                System.out.println("Il membro " + memberName + " non ha completato la challenge.");
                return false; // Restituisce false appena trova un membro che non ha completato
            }
        } catch (Exception e) {
            System.err.println("Errore durante la verifica della challenge per il membro: " + memberName);
            e.printStackTrace();
            throw new RuntimeException("Errore nella verifica della challenge per il membro: " + memberName);
        }
    }

    return true; // Restituisce true se tutti i membri hanno completato la challenge
}
```

Figura 4.16: isChallengeCompletedByTeam situato in ChallengeService

La funzione isChallengeCompletedByTeam (Figura 4.16) a differenza di isChallengeCompletedByMember verifica la challenge per un intero team e non si limita a controllare un singolo membro. La lista dei membri viene recuperata e per ciascun membro viene chiamata la funzione precedente isChallengeCompletedByMember. Se anche uno solo dei membri non ha completato la challenge, la funzione interrompe immediatamente l'operazione e restituisce false. Solo se tutti i membri superano la verifica, la funzione conclude con true. La funzione getAllChallengesAsHtml (Figura 4.17) è progettata per

```

public ResponseEntity<String> getAllChallengesAsHTML(String jwt) {
    // Controlla se il JWT è valido
    if (!jwtService.isJwtValid(jwt)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Accesso non autorizzato");
    }

    // Recupera tutti i Challenge dal repository
    List<Challenge> challenges = challengeRepository.findAll();

    // Genera il contenuto HTML per il corpo della tabella
    StringBuilder htmlBuilder = new StringBuilder();
    for (Challenge challenge : challenges) {
        htmlBuilder.append(str:"<tr>")
            .append(str:"<td>").append(challenge.getChallengeName()).append(str:"</td>")
            .append(str:"<td>").append(challenge.getDescription()).append(str:"</td>")
            .append(str:"<td>").append(challenge.getTeamId()).append(str:"</td>")
            .append(str:"<td>").append(challenge.getStartDate()).append(str:"</td>")
            .append(str:"<td>").append(challenge.getEndDate()).append(str:"</td>")
            .append(str:"<td>").append(challenge.getVictoryCondition()).append(str:"</td>")
            .append(str:"</tr>");
    }
    return ResponseEntity.ok(htmlBuilder.toString());
}

```

Figura 4.17: getAllChallengeHTML situato in ChallengeService

recuperare tutte le sfide presenti nel database e generare un output in formato HTML, che può essere utilizzato per visualizzare i dati in una tabella nel frontend. Il processo inizia verificando la validità del token JWT fornito nella richiesta. Questo controllo assicura che solo utenti autenticati possano accedere alle informazioni sensibili. Se il token non è valido, la funzione restituisce una risposta con stato UNAUTHORIZED e un messaggio di errore. Dopo la verifica, la funzione utilizza il ChallengeRepository per recuperare tutte le sfide memorizzate nel database MongoDB. I dati estratti vengono iterati e trasformati in righe di una tabella HTML utilizzando un oggetto StringBuilder. Ogni sfida viene rappresentata da una riga (<tr>) che include tutte le informazioni rilevanti, come il nome della sfida, la descrizione, l'ID del team associato, le date di inizio e fine, e la condizione di vittoria. Infine, la funzione restituisce il contenuto

HTML generato all'interno di una risposta con stato OK. Questo approccio consente di integrare facilmente i dati delle sfide nel frontend, offrendo un metodo strutturato per la loro visualizzazione in formato tabellare.

```
public ResponseEntity<String> updateExpiredChallenges(String jwt) {
    // Verifica la validità del token JWT
    if (!jwtService.isJwtValid(jwt)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body("Accesso non autorizzato: JWT non valido o assente.");
    }

    LocalDate today = LocalDate.now(); // Ottiene la data di oggi
    // Recupera tutte le challenge dal database
    List<Challenge> challenges = challengeRepository.findAll();

    int count = 0; // Contatore delle challenge aggiornate
    for (Challenge challenge : challenges) {
        LocalDate endDate = LocalDate.parse(challenge.getEndDate(), DateTimeFormatter.ISO_DATE);
        if (today.isAfter(endDate) && !"Scaduta".equals(challenge.getStatus())) {
            challenge.setStatus("Scaduta");
            challengeRepository.save(challenge);
            count++;
        }
    }
    return ResponseEntity.ok("Aggiornate " + count + " challenge a stato 'Scaduta'.");
}
```

Figura 4.18: updateExpiredChallenges situato in ChallengeService

Questa funzione updateExpiredChallenges controlla e aggiorna lo stato delle challenge scadute nel sistema. Per prima cosa, verifica la validità del token JWT ricevuto, assicurandosi che l'utente sia autorizzato ad accedere. Se il token non è valido o assente, restituisce una risposta di errore “Accesso non autorizzato”. Una volta convalidato l'accesso, la funzione ottiene la data corrente utilizzando LocalDate.now(). Recupera poi tutte le challenge presenti nel database tramite il challengeRepository. Per ciascuna challenge, confronta la sua data di fine con quella corrente: se la data corrente è successiva alla data di fine e lo stato della challenge non è già “Scaduta”, lo stato viene aggiornato a “Scaduta”. La challenge modificata viene quindi salvata nuovamente nel database e viene incrementato un contatore che tiene traccia delle challenge

aggiornate. Al termine del processo, la funzione restituisce una risposta con il numero totale di challenge che sono state aggiornate allo stato “Scaduta”. Questo metodo è utile per mantenere aggiornato lo stato delle challenge in base alla loro scadenza.

4.2.4 ChallengeRepository.java

Il file ChallengeRepository.java è un’interfaccia Spring Data che gestisce l’accesso ai dati dell’entità Challenge, fornendo metodi CRUD e la possibilità di definire query personalizzate. Estende MongoRepository e grazie a Spring, automatizza l’implementazione delle operazioni, garantendo modularità e una chiara separazione tra logica applicativa e accesso ai dati.

4.3 Modifiche nell'HomeController

4.3.1 Sviluppo delle Rotte API per i Team

Le API REST in questo Home Controller (Figura 4.19) servono a fornire un’interfaccia per la gestione dei team all’interno dell’applicazione. Attraverso queste API, il client può comunicare con il server per:

- Visualizzare i dati: Recuperare la lista dei team e degli studenti, ottenere la visualizzazione dei team in formato HTML;
- Modificare i dati: Creare nuovi team, aggiungere o rimuovere membri da un team, eliminare un team esistente.

Queste API seguono il paradigma REST, che permette di effettuare operazioni di Create, Read, Update e Delete (CRUD) sui dati in modo standardizzato, utilizzando richieste HTTP come GET e POST. Le API sono inoltre protette tramite token JWT, garantendo che solo gli utenti autenticati possano accedere alle risorse.

CAPITOLO 4. REQUISITO R5: NUOVE IMPLEMENTAZIONI

```
@GetMapping("/team_view")
@ResponseBody
public ResponseEntity<List<Team>> getAllTeams(@CookieValue(name = "jwt", required = false) String jwt) {
    return teamService.getAllTeams(jwt);
}

@GetMapping("/teams_view") //aggiunta rotta per visualizzare i team in formato html
public ResponseEntity<String> getAllTeamsAsHtml(@RequestHeader("Authorization") String jwt) {
    String token = jwt.replace("Bearer ", ""); // Rimuove il prefisso Bearer
    return teamService.getAllTeamsAsHtml(token);
}

//Modifica 06/12/2024
@GetMapping("/students_list")//aggiunta rotta per visualizzare la lista degli studenti
@ResponseBody
public ResponseEntity<Team> getStudentsList(@CookieValue(name = "jwt", required = false) String jwt) {
    return teamService.getStudentsList(jwt);
}

@PostMapping("/team_create")//aggiunta rotta per creare un team
@ResponseBody
public ResponseEntity<Team> createTeam(@RequestBody Team team, @CookieValue(name = "jwt", required = false) String jwt) {
    return teamService.createTeam(team, jwt);
}

// Aggiungere un membro al team
@PostMapping("/team_add_member")
@ResponseBody
public ResponseEntity<String> addMemberToTeam(
    @RequestBody Map<String, String> payload,
    @CookieValue(name = "jwt", required = false) String jwt) {
    String teamName = payload.get("teamName");
    String memberId = payload.get("memberId");
    return teamService.addMemberToTeam(teamName, memberId, jwt);
}

@PostMapping("/team_remove_member")//aggiunta rotta per rimuovere un membro dal team
@ResponseBody
public ResponseEntity<String> removeMemberFromTeam(@RequestBody Map<String, String> requestBody, @CookieValue(name = "jwt", required = false) String jwt){
    // Estrai i dati dalla richiesta
    String teamName = requestBody.get("teamName");
    String memberId = requestBody.get("memberId");
    // Verifica che i parametri siano presenti
    if (teamName == null || memberId == null) {
        return ResponseEntity.badRequest().body("Team name or member ID is missing.");
    }
    // Chiama il TeamService per rimuovere il membro
    return teamService.removeMemberFromTeam(teamName, memberId, jwt);
}

@PostMapping("/team_delete")
@ResponseBody
public ResponseEntity<String> deleteTeam(@RequestBody Map<String, String> requestBody, @CookieValue(name = "jwt", required = false) String jwt) {
    // Estrai i dati dalla richiesta
    String teamName = requestBody.get("teamName");
    // Verifica che il parametro sia presente
    if (teamName == null || teamName.isEmpty()) {
        return ResponseEntity.badRequest().body("Nome team mancante.");
    }
    // Chiama il TeamService per cancellare il team
    return teamService.deleteTeam(teamName, jwt);
}

//Chiamata da bottone nella pagina home admin
@GetMapping("/teams_show")
@ResponseBody
public ModelAndView showTeamManagementPage(HttpServletRequest request, @CookieValue(name = "jwt", required = false) String jwt) {
    return adminService.showTeamManagementPage(request, jwt);
}
```

Figura 4.19: Rotte API per i Team

4.3.2 Sviluppo delle Rotte API per le Challenges

Le API REST elencate nella figura 4.20 forniscono le funzionalità principali per la gestione delle challenges. Le API consentono di creare nuove challenge, eliminare challenge esistenti tramite il loro nome, recuperare la lista completa delle challenge, aggiornare lo stato delle challenge sia come dati JSON che in formato HTML, per essere visualizzate nell’interfaccia Admin. Inoltre, forniscono le tipologie di condizioni di vittoria disponibili, utili per configurare le regole di completamento delle challenge.

```

/* Crea una nuova challenge.
 */
@PostMapping("/challenge_create")
public ResponseEntity<Challenge> createChallenge(
    @RequestBody Challenge challenge,
    @CookieValue(name = "jwt", required = false) String jwt) {
    return challengeService.createChallenge(challenge, jwt);
}

/* Elimina una challenge.
 */
@PostMapping("/challenges_ChallengesByName")
@RequestBody
public ResponseEntity<String> deleteChallenge(
    @RequestBody Map<String, String> payload,
    @CookieValue(name = "jwt", required = false) String jwt) {
    String challengeName = payload.get("challengeName");
    return challengeService.deleteChallenge(challengeName, jwt);
}

//route botone della challenge
@GetMapping("/challenges_show")
@RequestBody
public ModelAndView showChallengeManagementPage(HttpServletRequest request, @CookieValue(name = "jwt", required = false) String jwt) {
    return adminService.showChallengeManagementPage(request, jwt);
}

@GetMapping("/challenge_view")
@ResponseEntity<String> getAllChallengesAsHtml(@RequestHeader("Authorization") String jwt) {
    String token = jwt.replace("target=\"Bearer ", "replacement"); // Rimuove il prefisso Bearer
    return challengeService.getAllChallengesAsHtml(token);
}

@GetMapping("/challenges_view")
@ResponseEntity
public ResponseEntity<List<Challenge>> getAllChallenges(@CookieValue(name = "jwt", required = false) String jwt) {
    return challengeService.getAllChallenges(jwt);
}

// Route per ottenere la victoryConditionType
@GetMapping("/victoryConditionTypes")
@ResponseEntity
public ResponseEntity<VictoryConditionType[]> getVictoryConditionTypes() {
    return ResponseEntity.ok(VictoryConditionType.values());
}
/* Endpoint per aggiornare lo stato delle challenge scadute.
 */
@PostMapping("/challenges/update_expired")
@ResponseEntity<String>
public ResponseEntity<String> updateExpiredChallenges(
    @CookieValue(name = "jwt", required = false) String jwt) {
    return challengeService.updateExpiredChallenges(jwt);
}

```

Figura 4.20: Rotte API per le Challenges

Capitolo 5

Diagrammi di Progetto per le Nuove Implementazioni

In questa sezione vengono presentati i diagrammi di progetto relativi alle nuove funzionalità implementate all'interno del sistema. Ogni diagramma è stato realizzato seguendo uno schema chiaro e strutturato, volto a rappresentare in maniera dettagliata il flusso delle operazioni, le interazioni tra i componenti principali e le condizioni di validazione necessarie.

5.1 Sequence Diagram createTeam

Osserviamo ora il diagramma di sequenza relativo alla funzione `createTeam` (Figura 5.1).

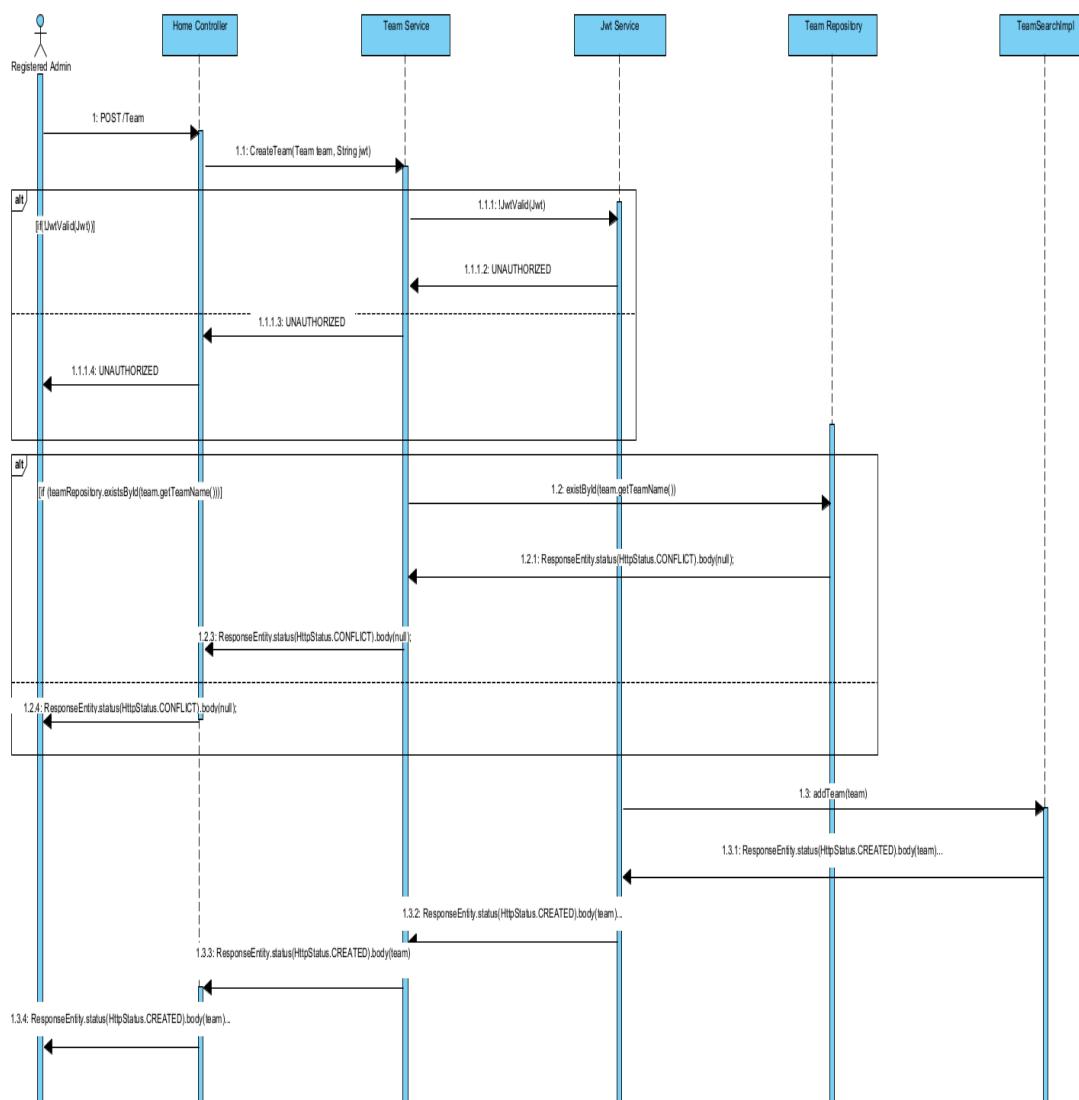


Figura 5.1: Sequence Diagram della funzione `createTeam`

5.2 Sequence Diagram deleteTeam

Osserviamo ora il diagramma di sequenza relativo alla funzione deleteTeam (Figura 5.2).

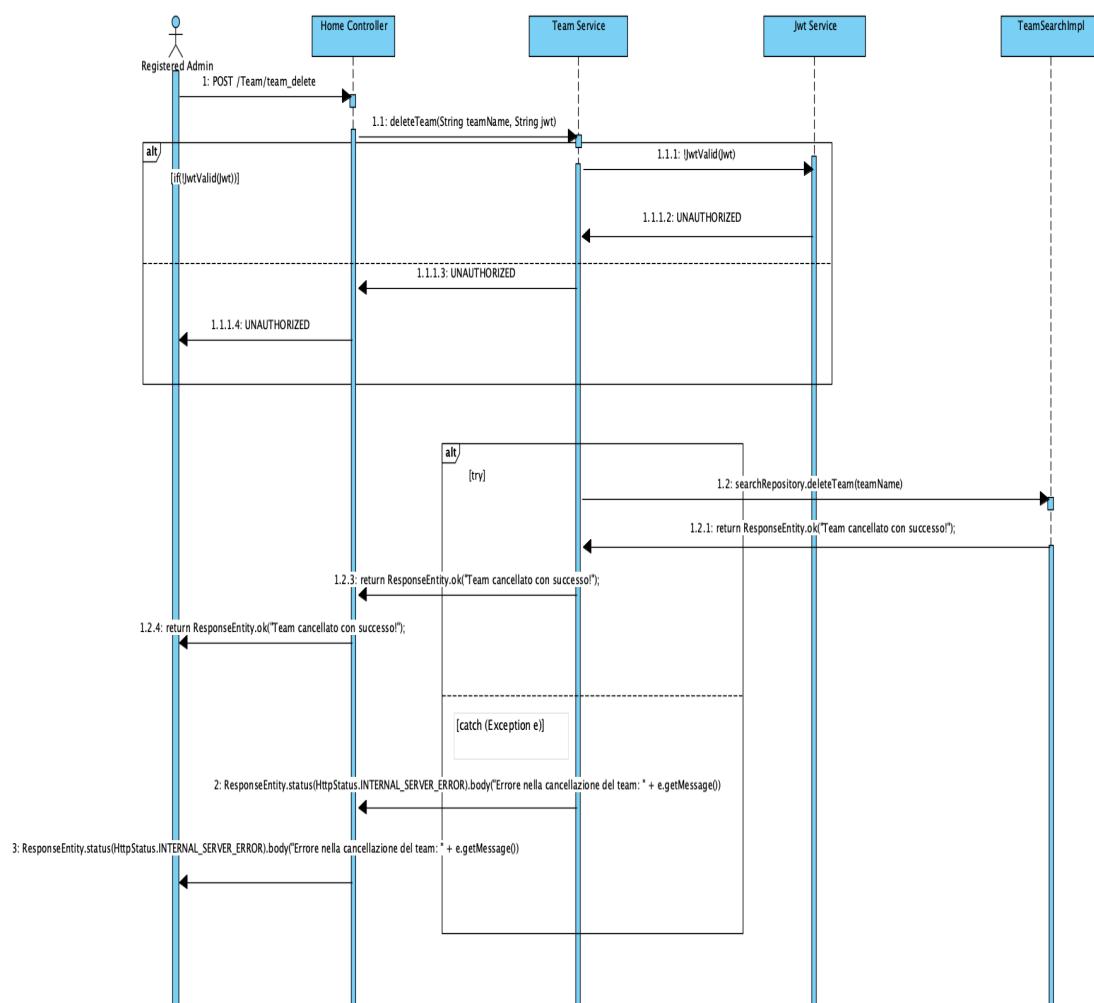


Figura 5.2: Sequence Diagram della funzione deleteTeam

5.3 Sequence Diagram

addMemberToTeam in createTeam

Osserviamo ora il diagramma di sequenza relativo alla funzione addMemberToTeam in createTeam (Figura 5.3).

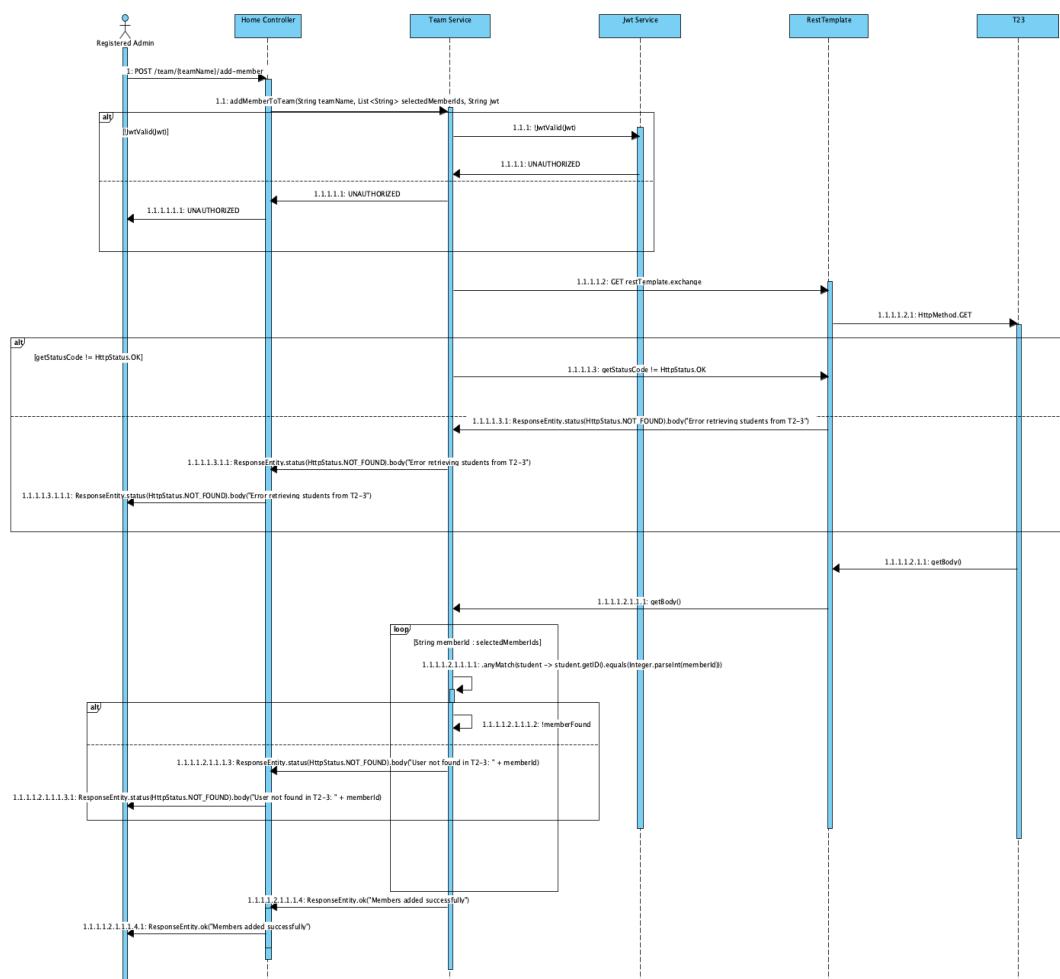


Figura 5.3: Sequence Diagram della funzione addMemberToTeam in createTeam

5.4 Sequence Diagram

addMemberToTeam

Osserviamo ora il diagramma di sequenza relativo alla funzione addMemberToTeam (Figura 5.4).

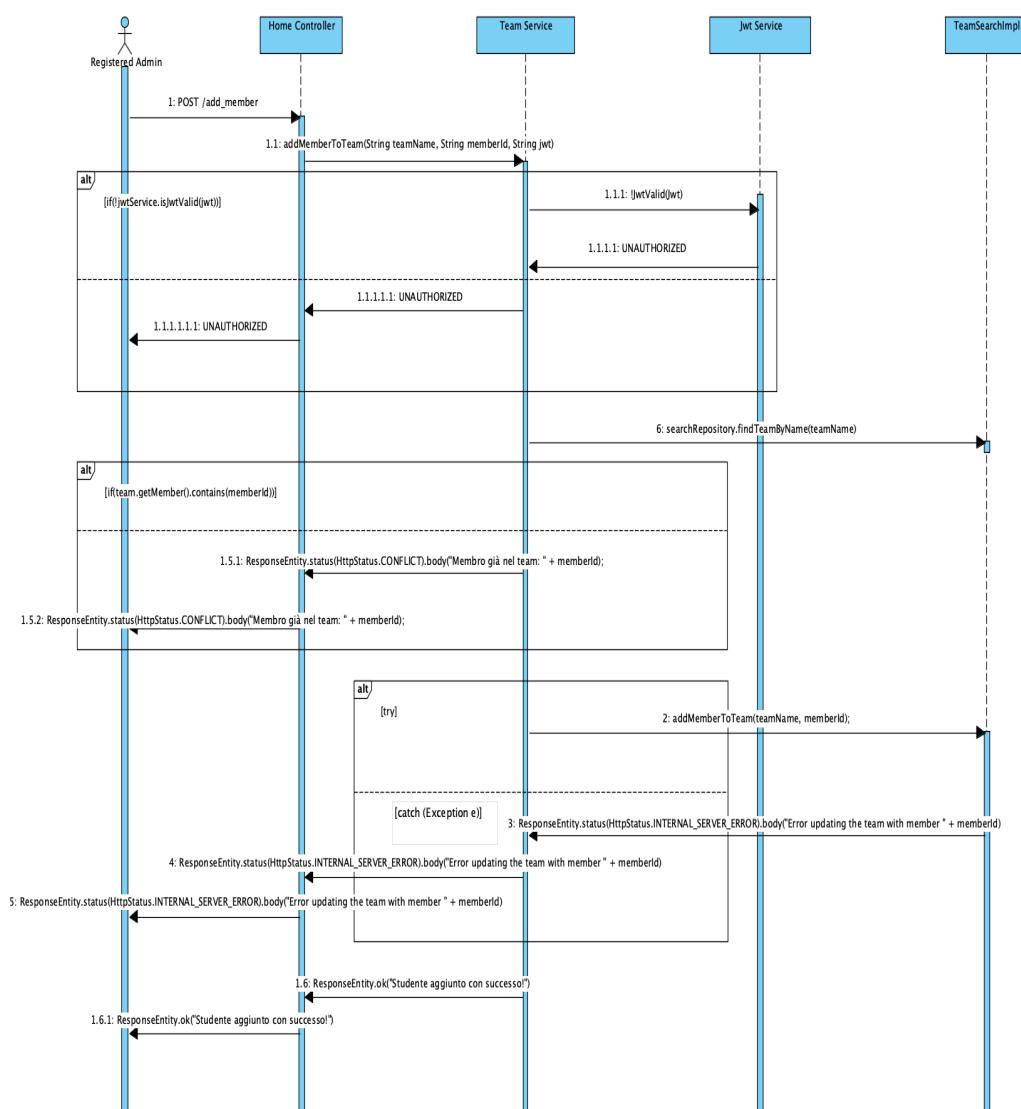


Figura 5.4: Sequence Diagram della funzione addMemberToTeam

5.5 Sequence Diagram

removeMemberFromTeam

Osserviamo ora il diagramma di sequenza relativo alla funzione removeMemberFromTeam (Figura 5.5).

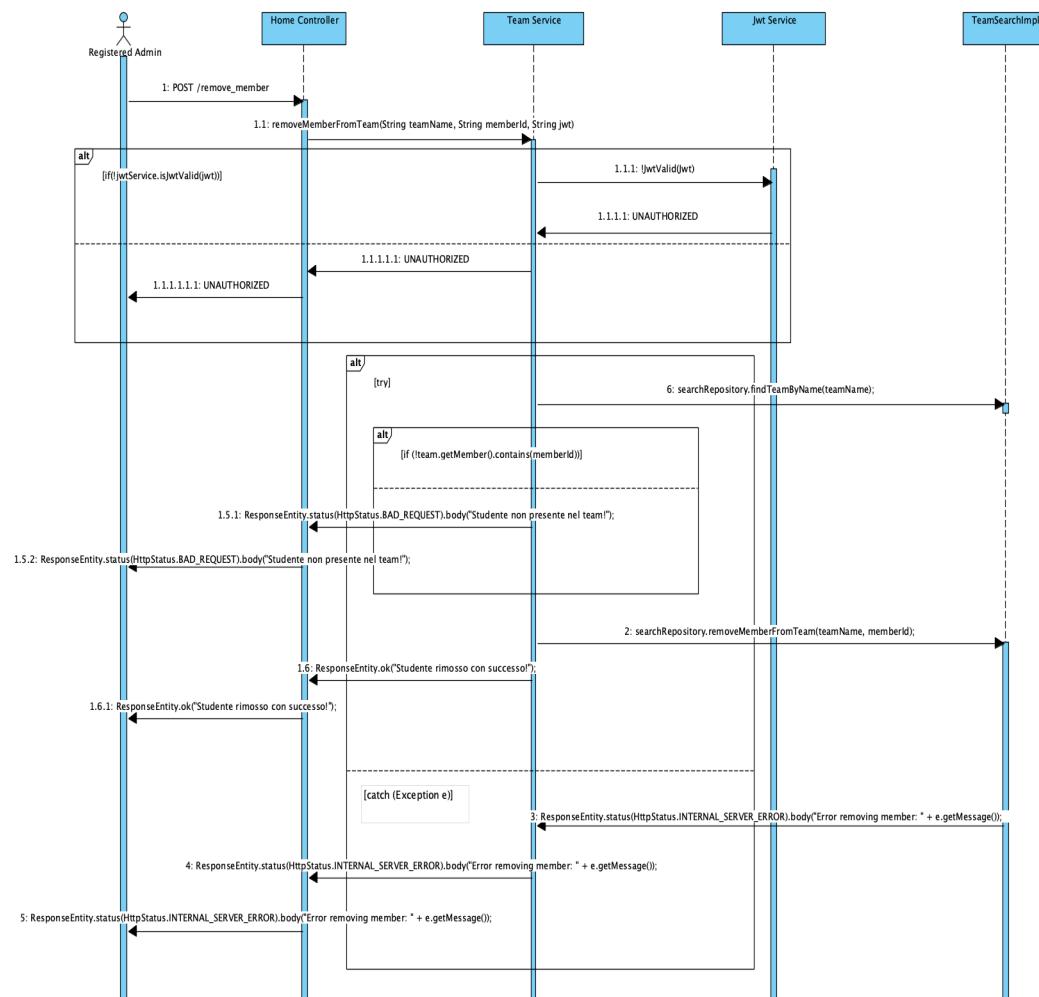


Figura 5.5: Sequence Diagram della funzione removeMemberFromTeam

5.6 Sequence Diagram createChallenge

Osserviamo ora il diagramma di sequenza relativo alla funzione `createChallenge` (Figura 5.6).

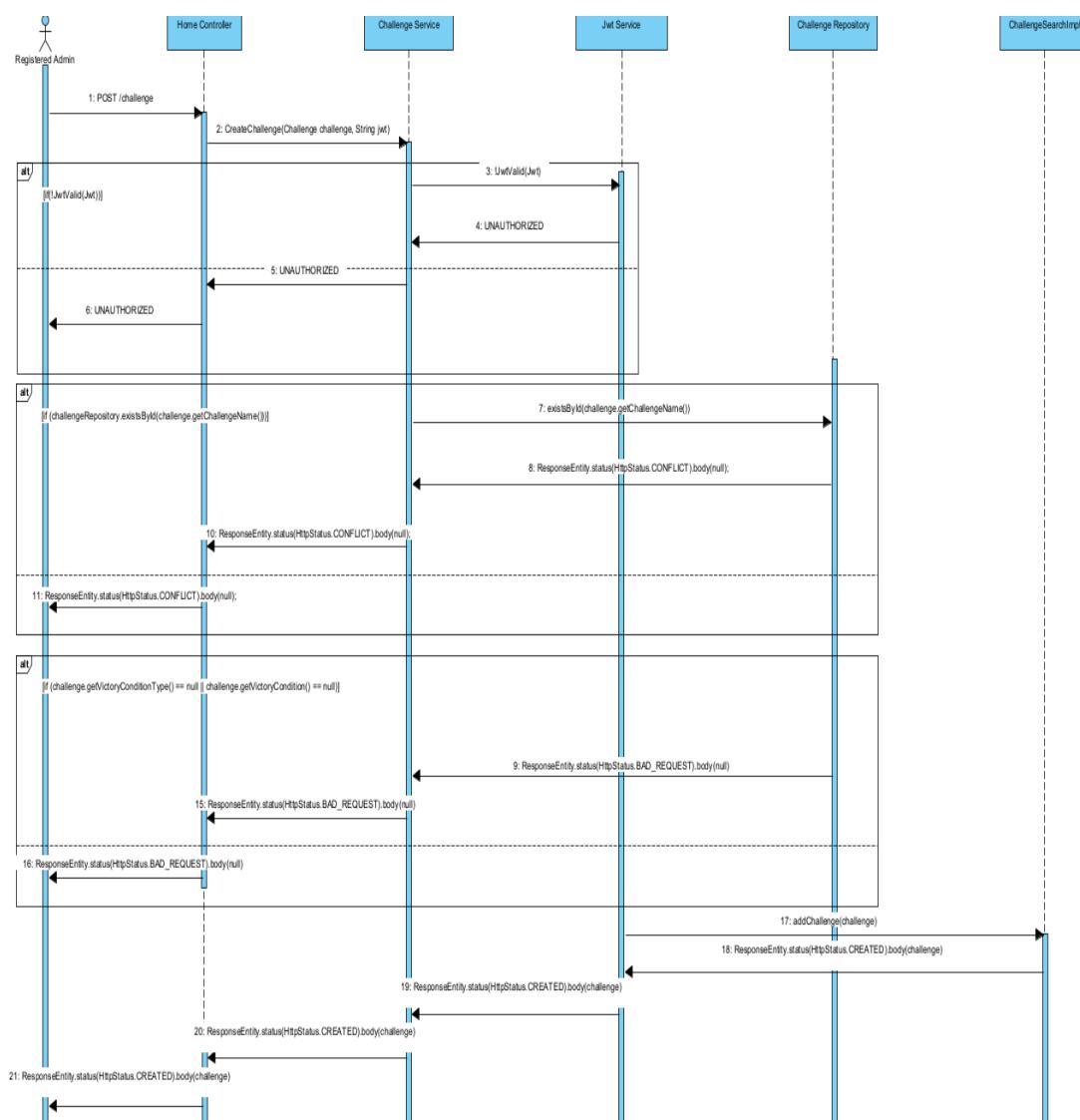


Figura 5.6: Sequence Diagram della funzione `createChallenge`

5.7 Sequence Diagram deleteChallenge

Osserviamo ora il diagramma di sequenza relativo alla funzione deleteChallenge (Figura 5.7).

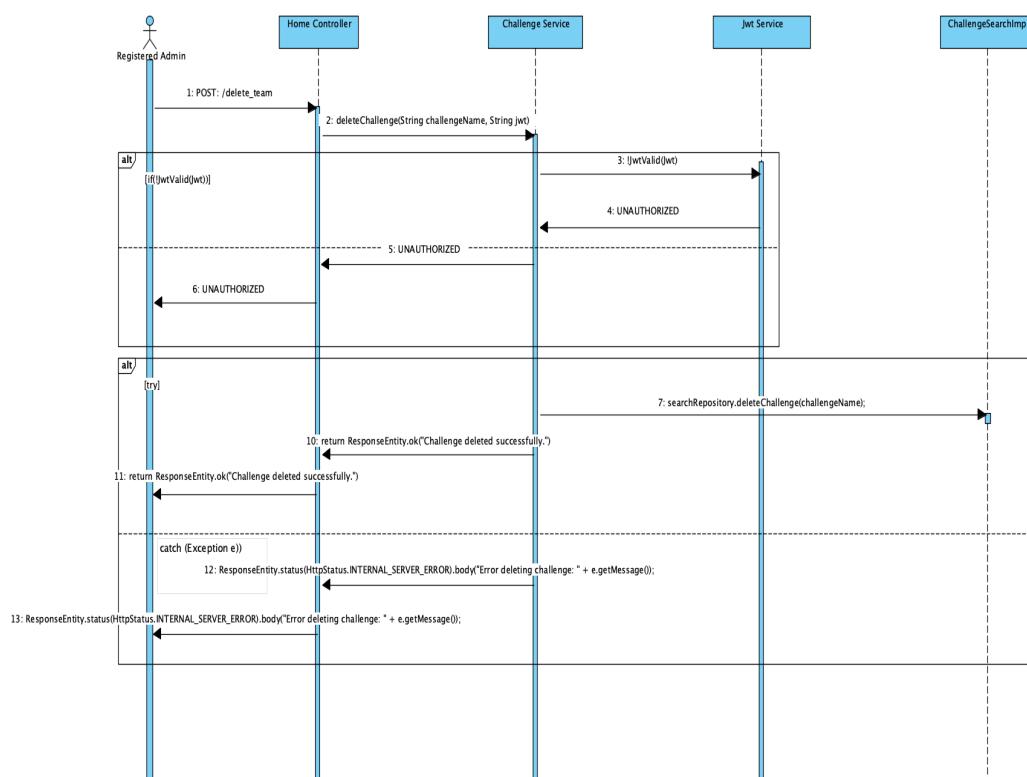


Figura 5.7: Sequence Diagram della funzione deleteChallenge

5.8 Sequence Diagram getAllChallenges

Osserviamo ora il diagramma di sequenza relativo alla funzione getAllChallenges (Figura 5.8).

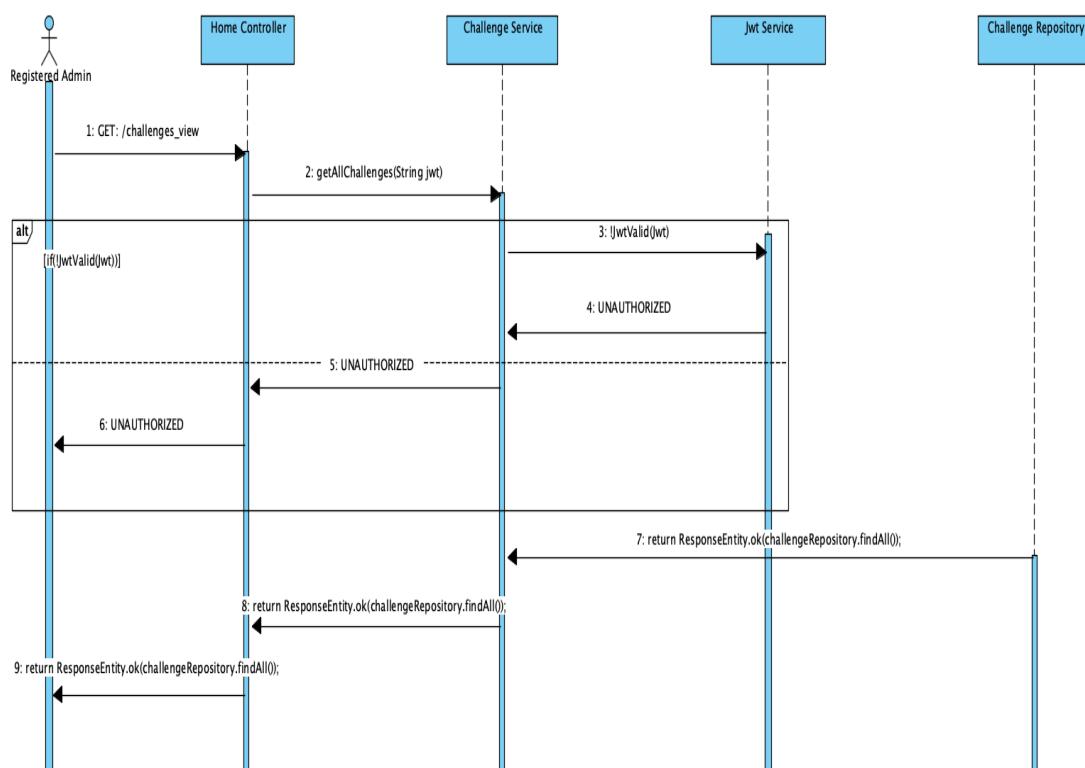


Figura 5.8: Sequence Diagram della funzione getAllChallenges

5.9 Package Service Aggiornato

Osserviamo ora il package service aggiornato con le nostre implementazioni (Figura 5.9).

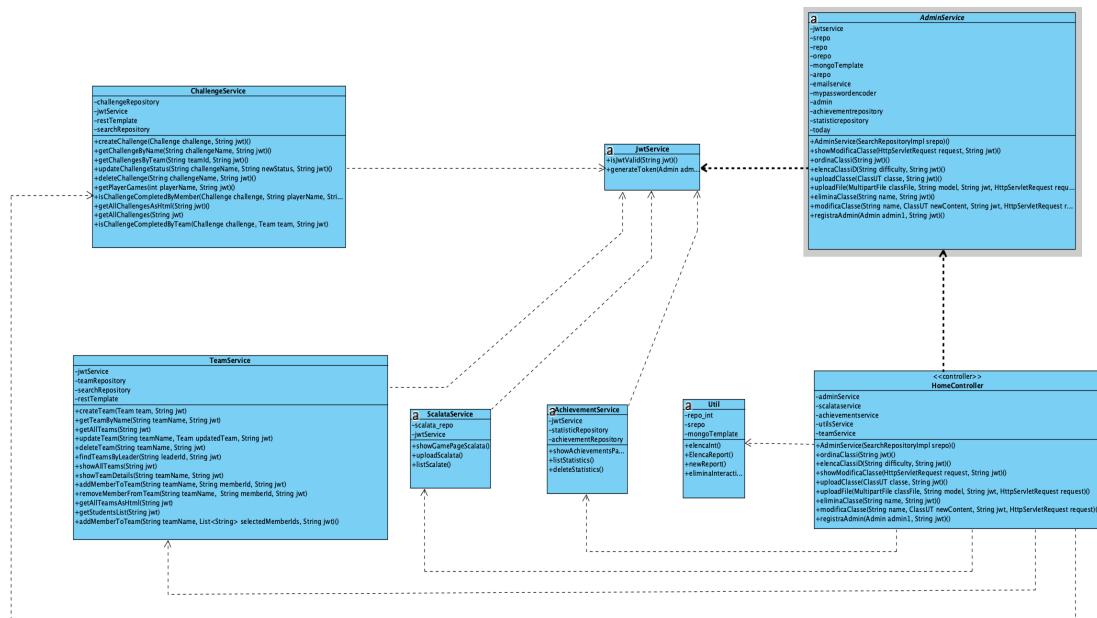


Figura 5.9: Package Service Aggiornato

Capitolo 6

Evoluzione del Front-End

In questo capitolo viene presentato un approfondimento sulle modifiche apportate al front-end dell'applicazione, con particolare attenzione alle funzionalità relative alla gestione dei team e delle challenges. Come prima modifica rilevante, è stata aggiornata la schermata iniziale della Home Admin (Figura 6.1) con l'inserimento di due nuovi pulsanti:

- Team: consente di accedere rapidamente alla sezione dedicata alla gestione dei team, dove è possibile creare, aggiornare e organizzare i dettagli dei vari team;
- Challenges: offre un accesso diretto alla gestione delle challenges, permettendo all'admin di configurare e monitorare lo stato delle challenges.

L'obiettivo principale di questa modifica è migliorare l'accessibilità e

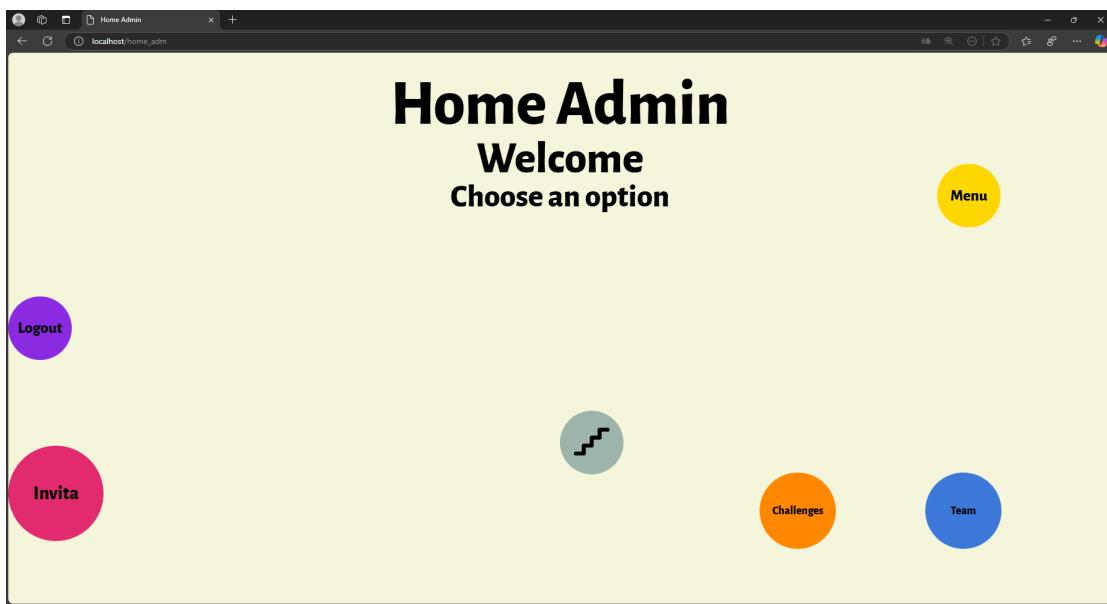


Figura 6.1: Schermata Home Admin aggiornata

la navigazione, fornendo un’interfaccia intuitiva e funzionale che guidi l’utente verso le sezioni chiave dell’applicazione.

6.1 Front-End Gestione Team

La sezione dedicata alla gestione dei team offre un’interfaccia chiara e intuitiva che permette all’admin di svolgere facilmente tutte le operazioni necessarie sui team. La pagina è organizzata in modo da migliorare l’usabilità, con un menu laterale e un contenuto centrale dinamico (Figura 6.2). Il menu laterale, posizionato sulla sinistra della schermata, fornisce un accesso rapido alle principali funzionalità della gestione team:

- Crea Team: Permette di aggiungere un nuovo team inserendo i

dati richiesti;

- Aggiungi Membro: Consente di aggiungere nuovi membri a un team esistente;
- Rimuovi Membro: Facilita la rimozione di membri dal team;
- Lista Team: Visualizza l'elenco completo dei team presenti nel sistema;
- Dettagli Team: Fornisce informazioni dettagliate su un team selezionato;
- Cancella Team: Permette di eliminare un team in modo sicuro e con conferma dell'operazione.

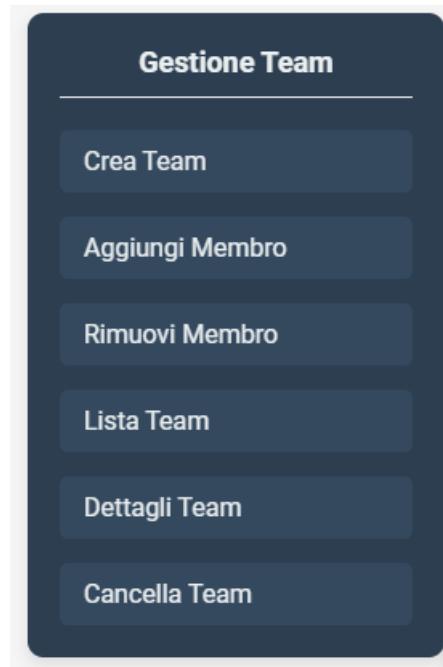


Figura 6.2: Schermata Menu Laterale di Gestione Team

6.1.1 Sezione di Crea Team

Questa schermata permette all'admin di creare un nuovo team inserendo le informazioni necessarie in modo semplice e intuitivo. Al centro della pagina è presente un form strutturato con campi dedicati per raccogliere i dettagli principali del team. L'admin può iniziare inserendo il nome del team e una breve descrizione per identificarne lo scopo o le caratteristiche. Successivamente, viene richiesto l'ID del leader, che rappresenta l'utente responsabile della gestione del team. Una funzionalità chiave è la gestione dei membri del team.

Attraverso il campo di ricerca dedicato, l'admin può cercare gli studenti da aggiungere. I risultati vengono visualizzati all'interno di una lista. Per aggiungere un membro, è sufficiente selezionare il nome dalla lista cliccando con il mouse. Allo stesso modo, se l'admin desidera rimuovere un membro, può semplicemente cliccare nuovamente sul nome selezionato per deselezionarlo. Nella parte inferiore del form sono presenti due pulsanti principali:

- “Pulisci Lista” (in rosso): consente di azzerare rapidamente la selezione dei membri, permettendo all'admin di ripartire da zero se necessario;
- “Aggiungi Team” (in verde): finalizza la creazione del team, inviando i dati inseriti al sistema.

Dopo aver compilato il form e cliccato su "Aggiungi Team",

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

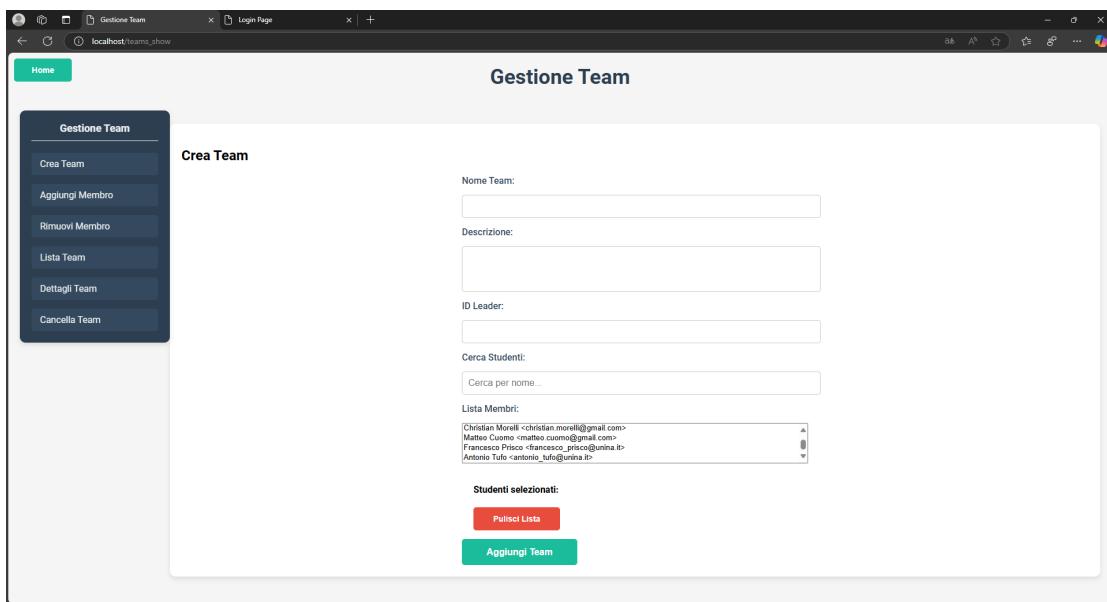


Figura 6.3: Schermata Crea Team

l'applicazione fornisce un feedback immediato tramite popup di conferma o di errore, a seconda dell'esito dell'operazione:

- Se l'operazione va a buon fine, viene mostrato un popup di conferma con il messaggio: “Team creato con successo!”. Questo avviso informa l'admin che il team è stato aggiunto correttamente al sistema (Figura 6.4);
- Nel caso in cui il nome del team inserito sia già presente nel sistema, viene visualizzato un messaggio di errore: “Errore: Il team esiste già. Inserisci un nome diverso.” Questo feedback guida l'admin a correggere l'input e inserire un nome univoco per procedere (Figura 6.5).

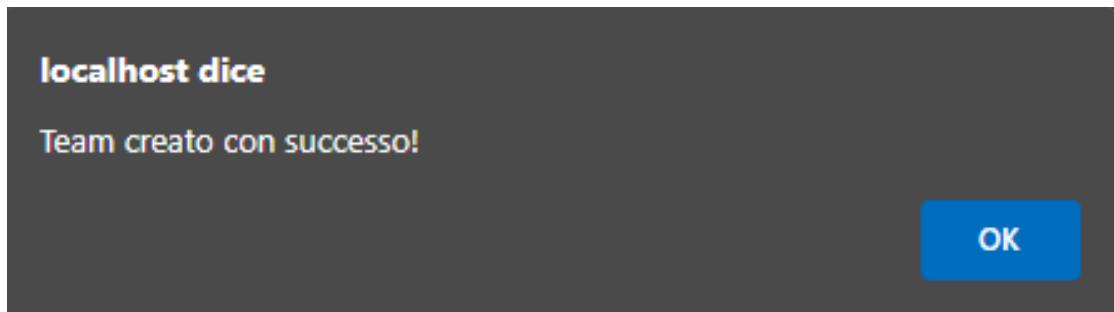


Figura 6.4: Popup di Avvenuta Creazione

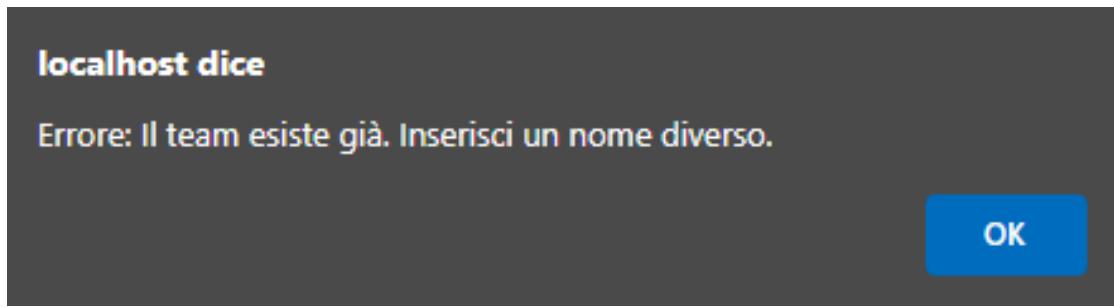


Figura 6.5: Popup di Errore Creazione

6.1.2 Sezione di Aggiungi Membro

Nella sezione principale della pagina (Figura 6.6), l’admin può aggiungere uno studente a un team seguendo questi passaggi:

- Seleziona il team: In questa sezione, l’Admin dispone di un menu a tendina che permette di scegliere il team a cui vuole aggiungere uno studente. Oltre a navigare tra le opzioni, questa funzionalità include la ricerca dei team per nome, facilitando l’operazione soprattutto quando i team presenti sono numerosi.
- Seleziona lo studente: Un altro menu a tendina elenca gli studenti disponibili, tra cui l’Admin può scegliere. Anche in questo caso la ricerca dello studente può avvenire tramite nome, cognome o email facilitando l’operazione soprattutto quando i membri presenti sono numerosi.
- Conferma l’operazione: L’Admin clicca sul pulsante “Aggiungi Membro”, che esegue l’operazione di aggiunta.

Una volta completata l’operazione, appare un messaggio di conferma in un popup che informa l’Admin del successo.

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

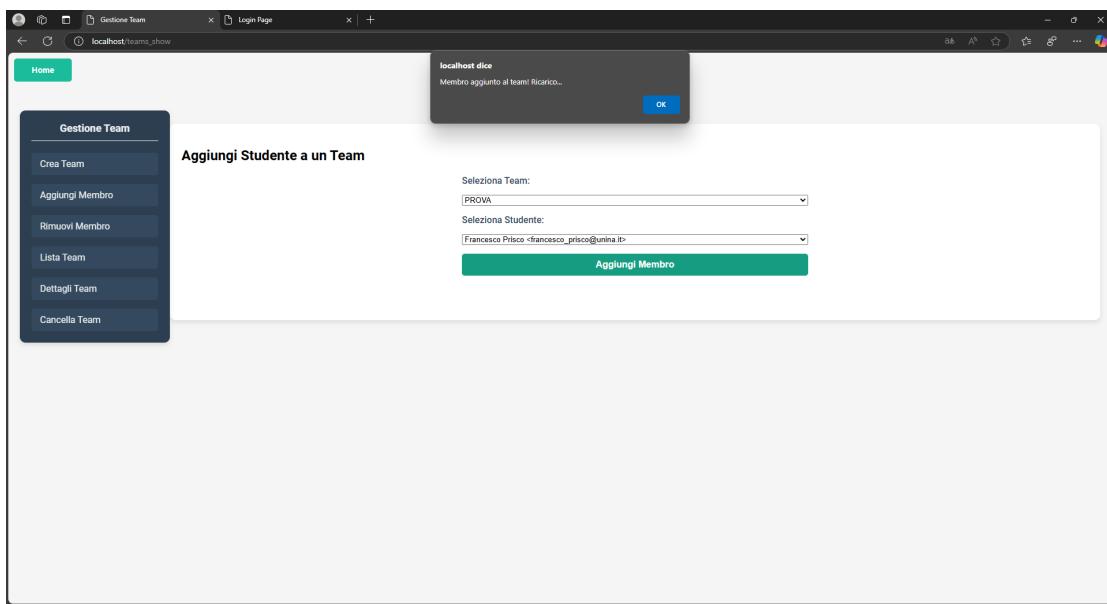


Figura 6.6: Schermata Aggiungi Membro

Durante l'operazione di aggiunta di un membro a un team, il sistema esegue una verifica per controllare se lo studente selezionato appartiene già al team. Se l'utente tenta di inserire un membro duplicato, il sistema interrompe l'operazione e mostra il seguente messaggio di errore (Figura 6.7):



Figura 6.7: Popup di Errore Aggiunta

Oltre alla funzionalità di aggiunta di un membro, è presente anche la

funzione “Rimuovi Membro”, che segue la stessa logica. Quando l’admin seleziona un team e un membro da rimuovere, il sistema verifica la presenza effettiva del membro nel team. Se il membro esiste, l’operazione viene completata con successo; in caso contrario, il sistema fornisce un messaggio di errore, informando Admin che il membro selezionato non è presente nel team. Questo approccio garantisce chiarezza e coerenza nelle operazioni di gestione dei team, prevenendo errori e duplicazioni.

6.1.3 Sezione di Lista Team

Questa schermata rappresenta la sezione Lista Team (Figura 6.8) all’interno della pagina dedicata alla gestione dei team. L’obiettivo di questa view è offrire all’admin una panoramica chiara e organizzata di tutti i team attualmente registrati nel sistema. Al centro della pagina è presente una tabella dinamica che mostra i dettagli principali di ogni team in modo ordinato e facilmente leggibile. Sopra la tabella è presente un campo “Filtra per Nome Team”, che permette all’admin di ricercare un team specifico in base al nome. Questa funzionalità migliora l’efficienza nella navigazione dei dati, soprattutto quando il numero di team registrati è elevato. L’admin può inserire una parte del nome del team e la tabella si aggiorna dinamicamente per mostrare solo i risultati corrispondenti.

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

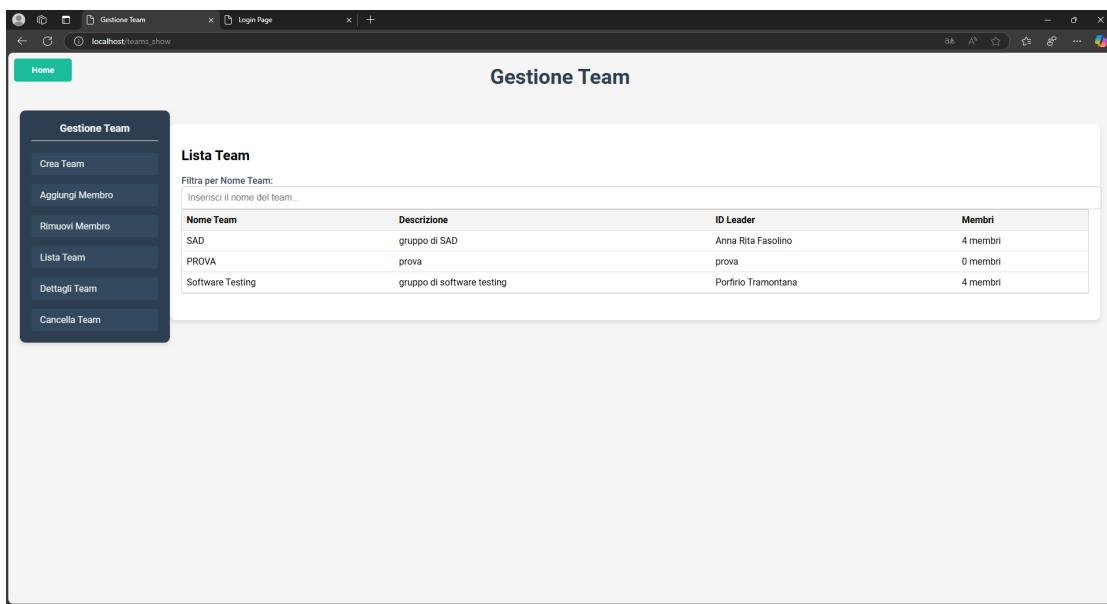


Figura 6.8: Schermata di Lista Team

6.1.4 Sezione di Dettagli Team

Questa schermata (Figura 6.9) offre all'admin la possibilità di visualizzare tutte le informazioni dettagliate relative a un singolo team selezionato. L'obiettivo principale è fornire una panoramica chiara e completa di ciascun team registrato nel sistema. Nella parte superiore, è presente un menu a tendina che consente di scegliere il team di cui si vogliono visualizzare i dettagli. Questa funzionalità permette di selezionare rapidamente un team specifico in modo semplice e intuitivo.

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

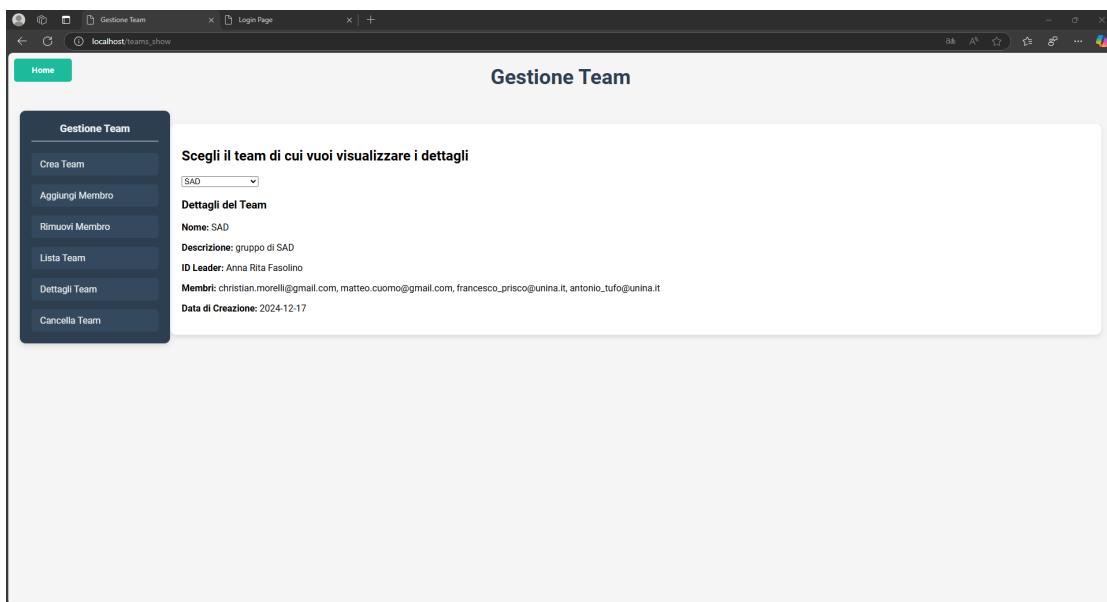


Figura 6.9: Schermata di Dettagli Team

6.1.5 Sezione di Cancella Team

Questa schermata (Figura 6.10) è dedicata alla funzionalità di cancellazione di un team all'interno della sezione di Gestione Team. L'Admin può selezionare il team desiderato attraverso un menu a tendina che elenca tutti i team esistenti nel sistema. Una volta scelto il team, premendo il pulsante “Cancella Team”, il sistema procede alla rimozione del team selezionato.

Dopo l'eliminazione, viene visualizzato un messaggio di conferma che informa l'Admin del successo dell'operazione con il testo: “Team Cancellato! Ricarico...”. La funzionalità garantisce che il team selezionato venga definitivamente rimosso dal sistema, migliorando così la gestione e l'organizzazione complessiva dei team.

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

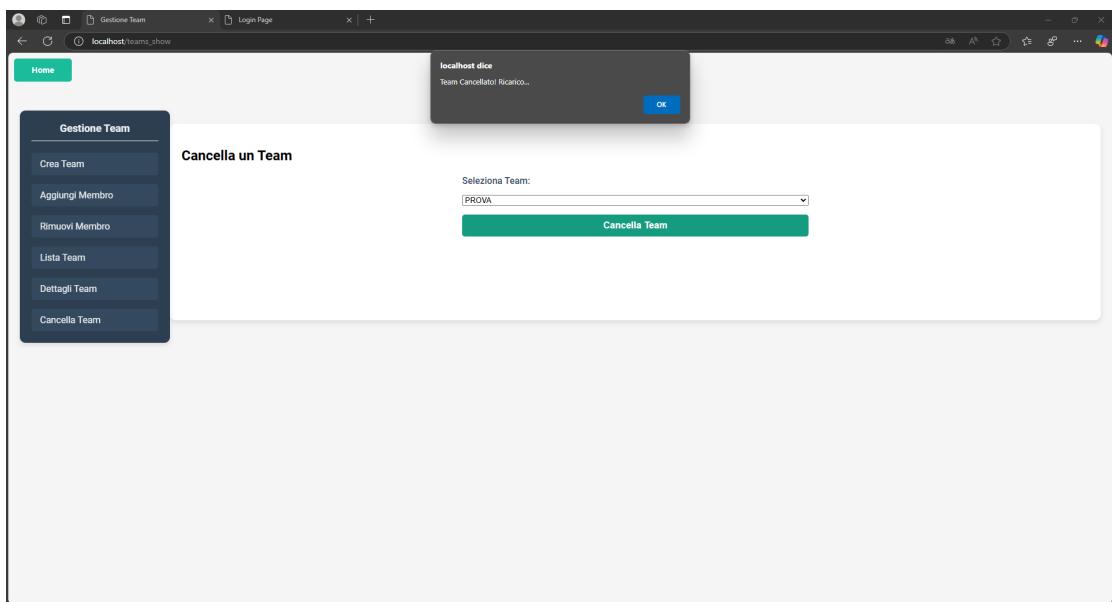


Figura 6.10: Schermata di Cancella Team

6.2 Front-End Gestione Challenges

La sezione dedicata alla gestione delle Challenges offre un’interfaccia chiara e intuitiva che permette all’admin di svolgere facilmente tutte le operazioni necessarie per la creazione, gestione e organizzazione delle challenges. La pagina è organizzata in modo da migliorare l’usabilità, con un menu laterale e un contenuto centrale dinamico (Figura 6.11). Il menu laterale, posizionato sulla sinistra della schermata, fornisce un accesso rapido alle principali funzionalità della gestione delle Challenges:

- Crea Challenge: Permette di aggiungere una nuova challenge inserendo i dati richiesti come nome, descrizione, condizioni e date di validità;

- Lista Challenges: Visualizza l'elenco completo delle challenges presenti nel sistema, con i dettagli chiave per ciascuna di esse;
- Dettagli Challenge: Fornisce informazioni specifiche su una challenge selezionata, come partecipanti, condizioni di vittoria e stato attuale;
- Cancella Challenge: Consente di eliminare una challenge in modo sicuro.



Figura 6.11: Schermata Menu Laterale di Gestione Challenges

6.2.1 Sezione di Crea Challenge

La schermata dedicata alla creazione di una nuova challenge (Figura 6.12) permette all'admin di configurare tutte le informazioni necessarie in maniera chiara e intuitiva. Al centro della pagina si trova un form strutturato che guida l'utente nell'inserimento dei dati. L'admin inizia inserendo il nome della challenge, il proprio nome

come creatore e una descrizione utile per spiegare lo scopo o i dettagli della challenge. Successivamente, è possibile associare la challenge a un team specifico selezionandolo da un menu a tendina, semplificando così l'assegnazione a gruppi già presenti nel sistema. Una particolare attenzione è stata riservata alla gestione delle date: la data di inizio della challenge deve essere odierna o successiva, impedendo l'inserimento di date passate. Questo vincolo garantisce la coerenza temporale della challenge e offre la possibilità di pianificare una challenge anche con qualche giorno di anticipo. Allo stesso modo, l'admin può specificare una data di fine, permettendo di stabilire la durata della challenge. Un elemento chiave della configurazione è la scelta della tipologia della challenge, disponibile tramite un menu a tendina. In questa versione, è stato implementato solo il tipo “GAMES_PLAYED”, che rappresenta una challenge basata sul numero di partite giocate. L'admin può inoltre definire la condizione di vittoria, specificando il numero di partite che devono essere completate per terminare con successo la challenge. Una volta inseriti tutti i dati, il pulsante “Aggiungi Challenge” consente di confermare l'operazione e salvare la challenge nel sistema.

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

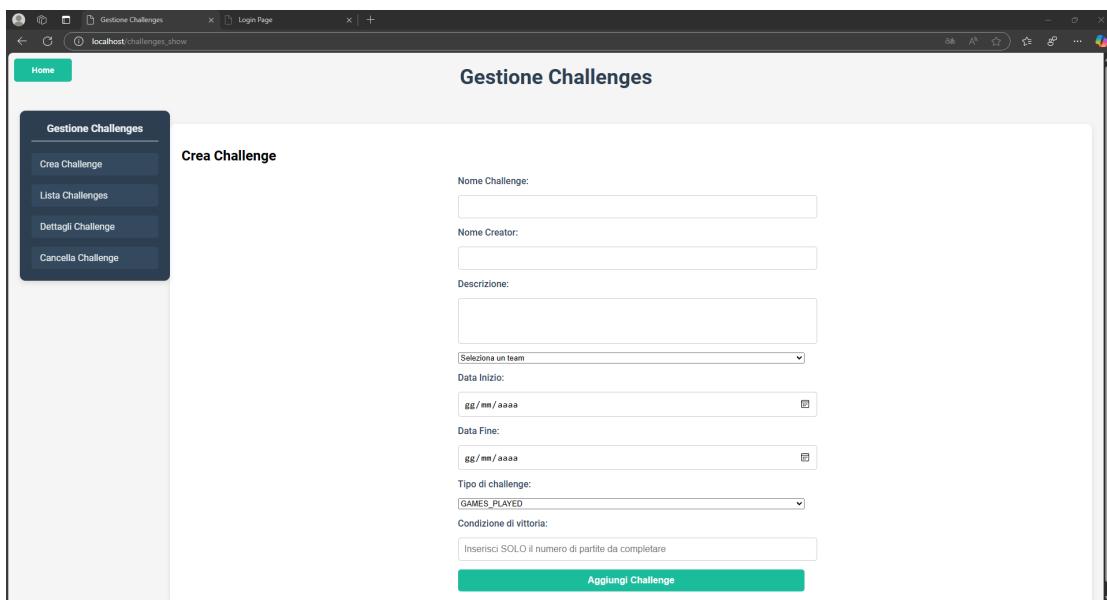


Figura 6.12: Schermata di Crea Challenge

Una volta inseriti tutti i campi richiesti nel form e cliccato sul pulsante “Aggiungi Challenge”, il sistema verifica i dati e fornisce un feedback immediato. Se la challenge viene creata con successo, appare un popup di conferma (Figura 6.13) che notifica all’admin l’esito positivo dell’operazione con il messaggio: “Challenge creata con successo!”. Questo avviso assicura che la challenge è stata correttamente salvata nel sistema e pronta per essere utilizzata.

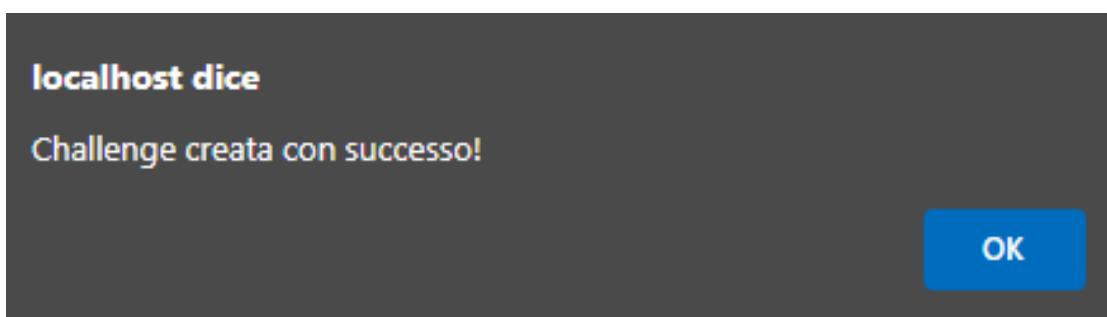


Figura 6.13: Popup di Avvenuta Creazione della Challenge

6.2.2 Sezione di Lista Challenges

La sezione dedicata alla Lista Challenges (Figura 6.14) permette all’admin di visualizzare e monitorare tutte le challenges configurate nel sistema in modo chiaro e strutturato. Al centro della schermata, è presente una tabella che raccoglie i dettagli di ciascuna challenge, consentendo di avere una panoramica completa delle challenges attive o in programma. Ogni riga della tabella rappresenta una singola challenge, riportando informazioni essenziali come il nome della challenge e una descrizione dettagliata delle condizioni per completarla. Ad esempio, nella schermata mostrata, una challenge richiede di giocare 3 partite, mentre un’altra richiede 5 partite, senza alcun vincolo relativo alla difficoltà o all’avversario. Oltre ai dettagli sulle challenges, la tabella evidenzia anche il team associato, identificato tramite il proprio nome, e le date di inizio e fine della challenge. Queste date sono fondamentali per stabilire il periodo di validità della challenge e determinare il suo stato attuale. Un aspetto rilevante è lo stato della challenge, che dipende proprio dalla data di inizio. Se la challenge è già partita, lo stato viene visualizzato come “in corso”, indicando che i partecipanti possono attivamente lavorare per completare le condizioni. Nel caso in cui la data di inizio sia futura, come nella seconda riga della tabella, lo stato è indicato come “in attesa”, segnalando che la challenge è stata configurata ma non è ancora iniziata. Questo permette all’admin di distinguere facilmente

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

tra challenges attive e programmate. Un'aggiunta importante a questa sezione è il pulsante “Aggiorna Stato”, posizionato sulla destra della barra di ricerca. Questo pulsante consente all’admin di verificare se una challenge è scaduta o meno in base alla data corrente. Quando l’admin clicca su di esso, il sistema controlla le date di fine di tutte le challenge e aggiorna automaticamente lo stato delle challenges. Questo permette di mantenere sempre aggiornate le informazioni, garantendo un monitoraggio più preciso e puntuale delle challenges. In aggiunta, è presente un campo di ricerca posizionato sopra la tabella che consente di filtrare le challenge inserendo il nome desiderato. Questa funzionalità risulta particolarmente utile per individuare rapidamente una specifica challenge quando l’elenco delle challenges diventa più lungo.

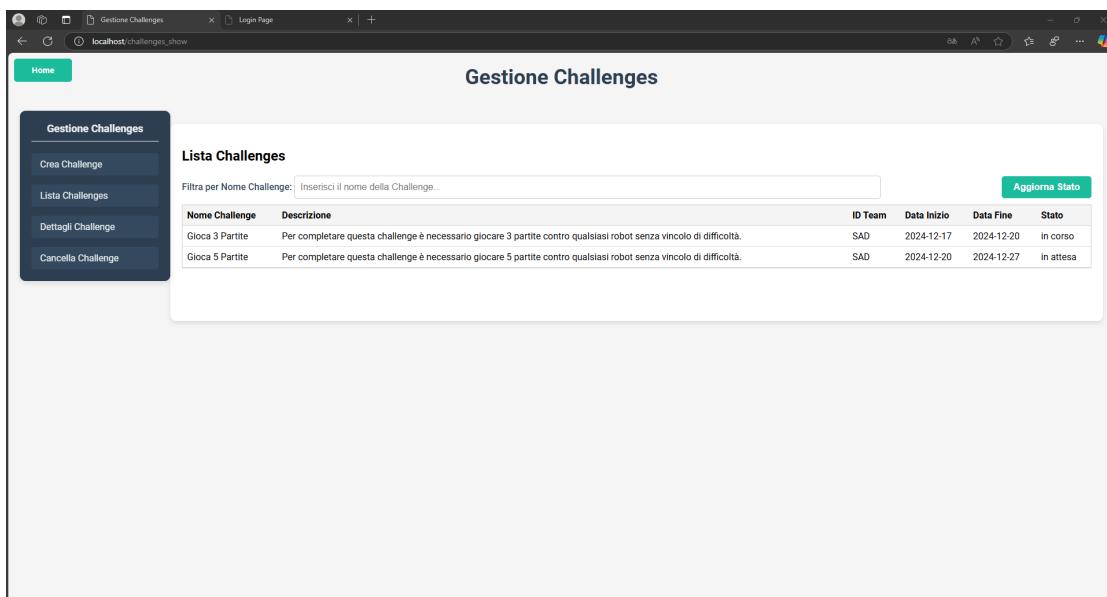


Figura 6.14: Schermata Lista Challenges

6.2.3 Sezione di Dettagli Challenge

La sezione dedicata ai Dettagli Challenge (Figura 6.15) mostra le informazioni complete della challenge selezionata tramite un menu a tendina. I dettagli della challenge includono il nome della Challenge, in questo caso (“Gioca 3 Partite”), una descrizione che spiega cosa bisogna fare per completarla, l’ID del team associato, l’ID del creatore, la data di inizio e fine, e lo stato attuale della challenge, che in questo caso è “in corso”. Vengono inoltre visualizzati il tipo di condizione di vittoria (GAMESPLAYED) e il numero di partite richieste per completarla, pari a 3.

Questa pagina è utile per fornire una panoramica dettagliata delle informazioni relative a una challenge, consentendo agli admin di monitorare facilmente il suo progresso e le condizioni richieste per il completamento.

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

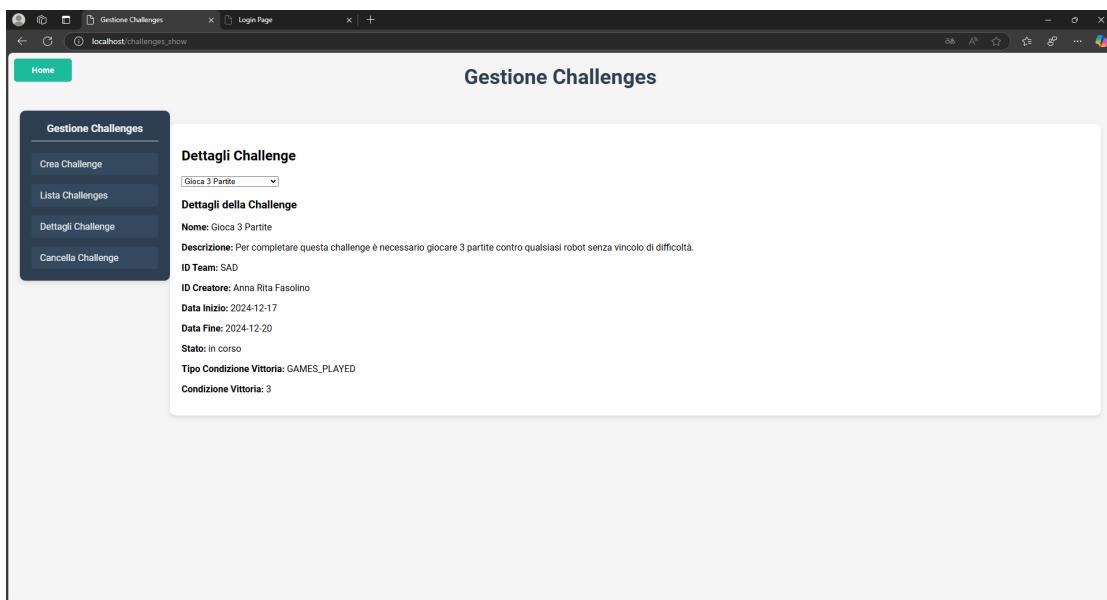


Figura 6.15: Schermata di Dettagli Challenge

6.2.4 Sezione di Cancella Challenge

La sezione Cancella Challenge (Figura 6.16) fornisce all’admin uno strumento semplice e diretto per eliminare una challenge esistente dal sistema. Al centro della pagina, è presente un menu a tendina che consente di selezionare la challenge desiderata tra quelle disponibili. Questo approccio facilita la scelta della challenge da cancellare, mostrando l’elenco aggiornato delle challenges esistenti. Una volta selezionata la challenge, l’admin può confermare l’eliminazione cliccando sul pulsante “Cancella Challenge”, visibile in verde. L’operazione è accompagnata da un popup di conferma, che notifica l’esito positivo con il messaggio: “Challenge cancellata! Ricarico...”. Questo feedback assicura che l’azione sia stata completata con

CAPITOLO 6. EVOLUZIONE DEL FRONT-END

successo.

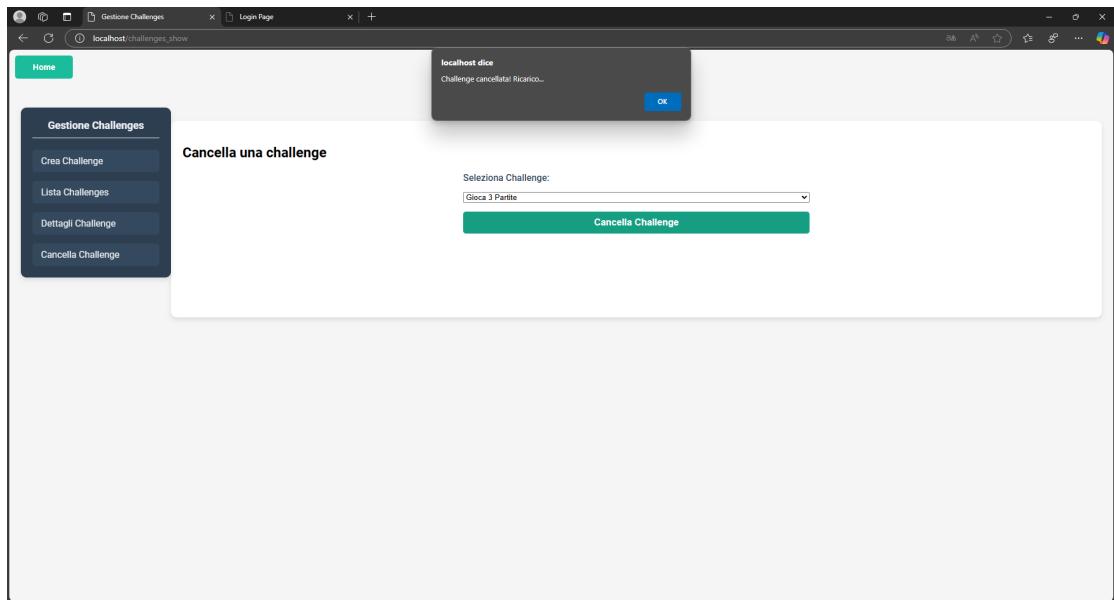


Figura 6.16: Schermata di Cancella Challenge

Capitolo 7

Organizzazione del lavoro con Scrum

Scrum è un framework Agile progettato per la gestione dei progetti e lo sviluppo del software. Basato sui principi e valori del "Manifesto Agile", Scrum offre una struttura organizzativa che aiuta i team a collaborare in modo efficace, adattandosi rapidamente ai cambiamenti e garantendo un progresso continuo verso gli obiettivi.

7.1 Adozione di Scrum per il nostro progetto

Nel nostro progetto abbiamo adottato il framework Scrum per organizzare il lavoro in modo iterativo e collaborativo. La suddivisione in piccoli incrementi (sprint) ha garantito flessibilità e

un miglior controllo sullo stato di avanzamento delle attività. Pratiche come il Daily Scrum e lo Sprint Backlog hanno mantenuto il team allineato sugli obiettivi, consentendo di identificare tempestivamente problematiche e ottimizzare i tempi. Il lavoro è stato suddiviso in due iterazioni iniziali, dedicate alla comprensione dei problemi della web application e all'assegnazione dei requisiti ai singoli gruppi. Successivamente, sono state svolte tre review durante le quali abbiamo presentato i nostri progressi e ricevuto feedback per migliorare il lavoro. L'adozione di Scrum ha migliorato la comunicazione nel team, semplificato il processo decisionale e garantito una distribuzione più equa dei task. Questo approccio ci ha permesso di soddisfare i requisiti del progetto e sviluppare una metodologia di lavoro efficace e adattabile.

7.2 Prima Iterazione

Durante la prima iterazione del progetto, ci siamo concentrati sull'analisi della documentazione esistente e sul testing dell'applicazione web, con l'obiettivo di identificare eventuali bug e proporre nuove funzionalità da integrare nel sistema. Ogni gruppo ha lavorato in modo indipendente per esaminare approfonditamente i materiali disponibili e testare le diverse funzionalità della piattaforma, cercando di individuare criticità o aree di miglioramento. Successivamente, è stato organizzato un brainstorming generale,

durante il quale i vari gruppi hanno condiviso i loro feedback e le osservazioni emerse durante l’analisi. Questo confronto ha permesso di raccogliere spunti utili per definire le priorità delle attività future, tra cui la risoluzione dei problemi individuati e la pianificazione di eventuali nuove funzionalità.

7.3 Seconda Iterazione

Partendo dalle analisi effettuate durante la prima iterazione, la professoressa ci ha presentato un elenco dettagliato dei requisiti che potevano essere implementati nel progetto. Dopo aver letto e discusso in maniera collettiva ogni requisito, ciascun gruppo ha avuto la possibilità di scegliere uno dei requisiti proposti.

Dopo un confronto interno, il nostro gruppo ha deciso di selezionare il Requisito R5 come punto focale del nostro lavoro, ritenendolo prioritario e strategico per lo sviluppo del sistema.

7.4 Prima Review

Durante la prima review, abbiamo presentato il Requisito R5, descrivendolo nel dettaglio e scomponendolo nelle diverse funzionalità da implementare. Per una maggiore chiarezza e organizzazione, abbiamo creato tavelle che individuavano i casi d’uso e i relativi requisiti funzionali, identificando l’Admin registrato come l’utente

principale che utilizzerà i servizi legati al requisito. Abbiamo inoltre definito alcuni scenari d'uso, descrivendo la sequenza degli eventi principali per illustrare come le funzionalità dovrebbero operare in pratica. Questo ci ha permesso di analizzare il flusso delle operazioni e di evidenziare i punti critici da affrontare. Infine, abbiamo discusso e concordato su quale container avrebbe impattato maggiormente l'implementazione del requisito, identificando il container T1 come il fulcro del lavoro necessario.

7.5 Seconda Review

Durante la seconda review, ci siamo concentrati sul mostrare i progressi fatti nell'implementazione del Requisito R5 attraverso i diagrammi aggiornati. In particolare, abbiamo presentato le modifiche apportate al Component Diagram, tra cui l'aggiunta del TeamService, che rappresenta una delle principali funzionalità introdotte. Abbiamo inoltre illustrato due funzioni chiave implementate, `createTeam` e `addMemberToTeam`, accompagnandole con i rispettivi Sequence Diagram per spiegare in dettaglio la sequenza di eventi e l'interazione tra i vari componenti del sistema. Alla fine della presentazione, abbiamo discusso alcune problematiche riscontrate durante lo sviluppo. In particolare, dopo l'aggiunta del bottone "Team" lato front-end, abbiamo notato che cliccandoci sopra si verificavano degli errori, impedendo il corretto funzionamento

dell’interfaccia. Grazie al supporto della professoressa e dell’ingegnere Marco De Luca, siamo riusciti a individuare e risolvere la causa del problema, ripristinando il funzionamento previsto.

7.6 Terza Review

Durante la terza review, abbiamo presentato i progressi raggiunti nel nostro progetto, con un focus particolare sull’implementazione del backend per le challenges e su alcune funzionalità del front-end, come la creazione di una challenge e la sua assegnazione a un team.

Rispetto alla review precedente, in cui avevamo riscontrato problemi nella visualizzazione della pagina di gestione dei team, siamo riusciti a risolvere tali difficoltà e a mostrare un sistema funzionante e stabile. Abbiamo illustrato tutte le nuove funzionalità implementate, tra cui la creazione di un team, l’aggiunta e la rimozione di un membro, e la visualizzazione dei membri associati. Questi progressi hanno dimostrato l’integrazione delle logiche di backend con il front-end, rendendo il sistema più completo e funzionale. Durante la review, abbiamo ricevuto feedback importanti e consigli su alcune modifiche da apportare per migliorare la comprensibilità e l’usabilità del sistema.

Capitolo 8

Glossario

8.1 Admin

Un utente registrato come amministratore, denominato "Admin", responsabile dell'organizzazione delle attività.

8.2 Container

Nel progetto, i container rappresentano i diversi microservizi che comunicano tra loro.

8.3 Team

Gruppo di utenti registrati gestito dall'Admin. I team vengono utilizzati per organizzare gli utenti registrati in base alle attività o sfide assegnate.

8.4 Challenge

Attività o compito assegnato agli utenti registrati da parte dell'Admin.

8.5 Studente

Utente registrato nel sistema che partecipa alle sfide contro i robot, testando le classi.