



UNIVERSITA' DEGLI STUDI DI  
NAPOLI FEDERICO II

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

# *Progetto finale di Software Architecture Design*

Anno Accademico 2022/2023

Professoressa  
**Anna Rita Fasolino**

Studenti  
**Lorenzo Poli matr. M63001560**  
**Francesco Della Valle matr. M63001500**  
**Michele Morgillo matr. M63001467**

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Progetto . . . . .	1
1.2	Architettura a microservizi . . . . .	2
1.3	Analisi dell'architettura complessiva . . . . .	4
1.3.1	Component Diagram . . . . .	4
1.3.2	Diagramma dei Casi d'Uso . . . . .	5
1.3.3	Composite Diagram . . . . .	6
1.3.4	Deployment Diagram . . . . .	7
<b>2</b>	<b>Sviluppo e Obiettivi</b>	<b>9</b>
2.1	Daily SCRUM . . . . .	9
2.2	Strumenti utilizzati . . . . .	10
2.3	Obiettivi del lavoro . . . . .	11
2.4	Iterazioni . . . . .	12
2.5	Problematiche affrontate . . . . .	13
<b>3</b>	<b>Refactoring T1</b>	<b>15</b>
3.1	Component Diagram . . . . .	17
3.2	Package Service . . . . .	18
3.3	Sequence Diagram . . . . .	21

4	Issue reCAPTCHA e messaggio Upload Class	23
5	Sviluppi futuri	25

# Capitolo 1

## Introduzione

La seguente documentazione descrive le metodologie utilizzate e le varie fasi di sviluppo per la modifica e l'aggiunta di funzionalità che compongono il gioco educativo "Man vs Automated Testing Tools challenges", all'interno del progetto ENACTEST (European iNnovative AllianCe for TESTing). Per dettagli sui vari task integrati si rimanda alla documentazione individuale dei team coinvolti, dove è stata condotta un'analisi specifica.

### 1.1 Progetto

Il task assegnato prevede il **refactoring del task T1** a partire dalla versione A13 del progetto, reperibile al seguente link [Testing-Game-SAD-2023/A13](#), oltre ad aggiustamenti e risoluzioni di bug quali il ripristino del **reCAPTCHA in fase di registrazione dei Players**, e

l'aggiunta di un **messaggio di corretto Upload Class** quando l'admin carica una classe (sia da sola che con i test pre-generati), seguito dal reindirizzamento automatico alla pagina dell'home admin. L'intento principale del refactoring è quello di rendere il codice più efficiente, modulare e adattabile a modifiche future. Di seguito si riportano i diagrammi del sistema complessivo, col fine di avere un quadro generale della struttura e dei componenti che popolano il sistema e su cui siamo andati ad intervenire per i nostri obiettivi.

## 1.2 Architettura a microservizi

Nell'architettura a microservizi della nostra web application, i client possono utilizzare l'applicazione solo attraverso due punti di accesso principali tramite internet, noti come gateway. I microservizi che costituiscono l'applicazione non sono visibili o accessibili direttamente dall'esterno della loro "rete interna". Questa configurazione aggiunge un ulteriore strato di sicurezza al sistema. In altre parole, i microservizi sono nascosti all'esterno, e questo rende il sistema più sicuro. In figura viene riportata l'architettura, in cui si evince l'utilizzo del **Gateway Pattern**, che consiste nell'introdurre dei componenti che costituiscono gli unici punti di accesso ai microservizi: ciò permette di implementare dei meccanismi di autenticazione, autorizzazione, routing, load-balancing e API composition (una sorta di facade) delle richieste molto più versatili ed efficienti. L' **UI Gateway** funge

da reverse proxy: la sola responsabilità di questo gateway è quella di effettuare il routing delle richieste relative al frontend (UI). Ciò ha permesso di rendere disponibile l'intera applicazione alla porta 80 (porta predefinita per le connessioni HTTP), ottenendo così una certa uniformità. L'**API Gateway** funge da reverse proxy ma con maggiori responsabilità: si occupa di effettuare il routing, di gestire autenticazione autorizzazione delle richieste relative alle REST API. Ciò ha permesso di rendere disponibili tutte le API al di sotto del percorso URL `"/api/"`, ottenendo così anche qui una certa uniformità.

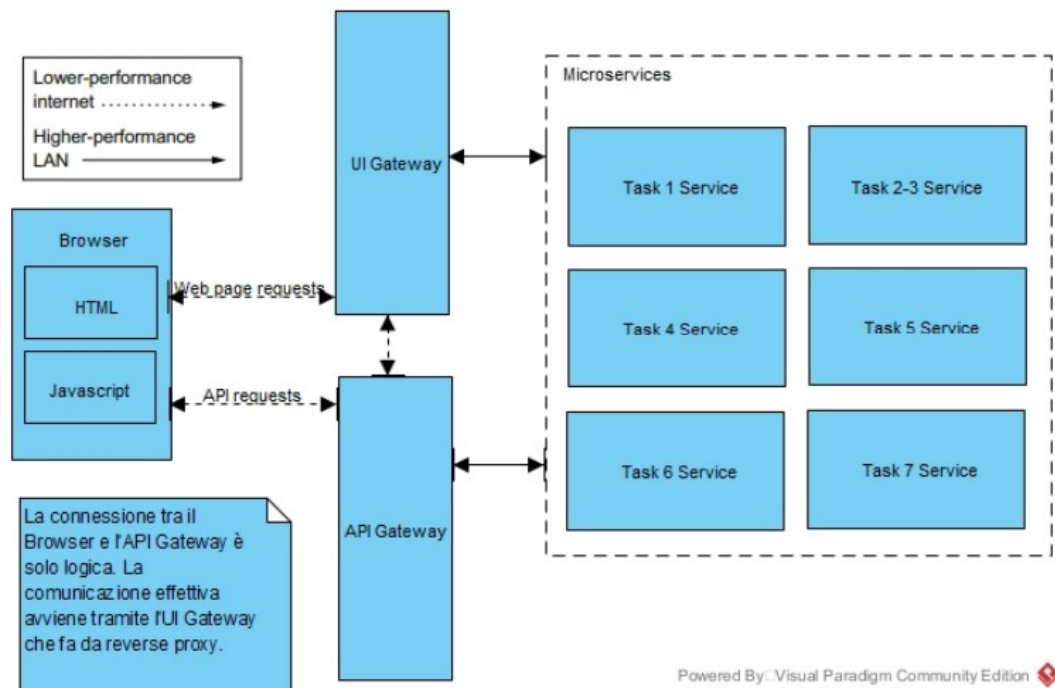


Figura 1.1: Architettura del sistema

## 1.3 Analisi dell'architettura complessiva

Il capitolo prosegue con una descrizione generale dell'architettura, evidenziando i punti in cui sono stati apportati gli interventi. Per supportare questa analisi, verranno presentati un diagramma dei casi d'uso, un diagramma dei componenti, un composite diagram e un deployment diagram, ciascuno dei quali illustrerà le aree e le interazioni coinvolte nei vari interventi.

### 1.3.1 Component Diagram

Il Diagramma dei Componenti rappresenta uno strumento fondamentale per diversi motivi:

- Fornisce una panoramica dell'architettura, illustrando come i vari componenti sono organizzati e collegati, facilitando agli stakeholder la comprensione complessiva del progetto
- Mette in evidenza le dipendenze tra i componenti, rendendole più facilmente analizzabili
- Supporta l'identificazione di punti critici e potenziali problematiche di progettazione

Di seguito viene presentato il diagramma dei componenti dell'architettura attuale, con i componenti modificati evidenziati in rosso:

*Admin Front End, Test Class Manager, Java Class Under Test Repository e Student Registration Service.*

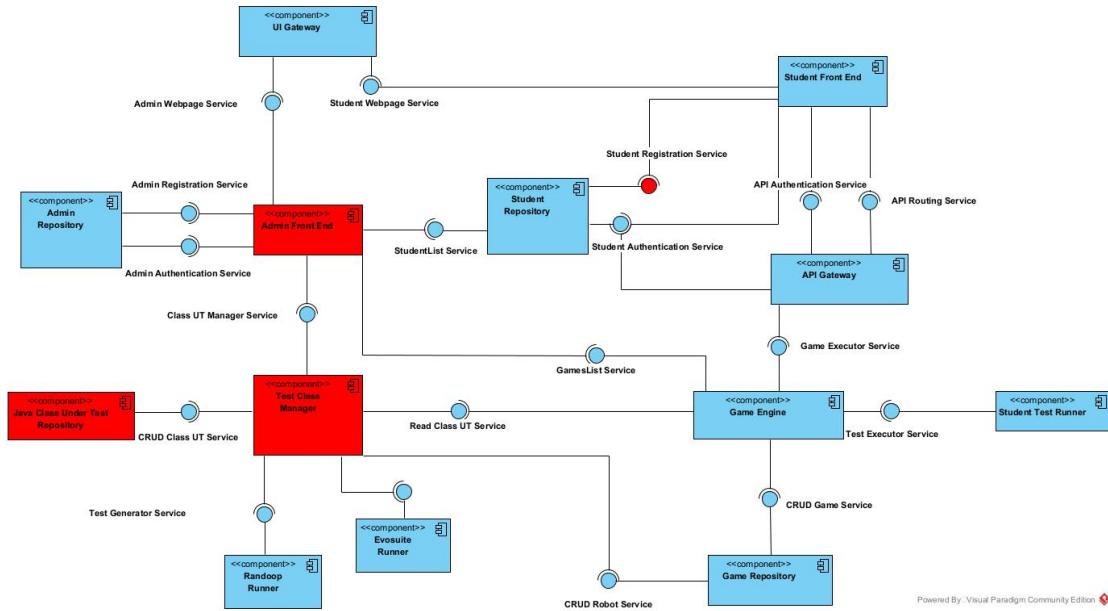


Figura 1.2: Component Diagram

### 1.3.2 Diagramma dei Casi d'Uso

Il Diagramma dei Casi d'Uso è uno strumento essenziale per rappresentare le interazioni tra gli utenti e il sistema, delineando le principali funzionalità e le relazioni tra di esse. Questo tipo di diagramma aiuta gli stakeholder a comprendere come gli utenti finali interagiscono con il sistema e quali sono le operazioni disponibili.

Nel nostro caso, il diagramma dei casi d'uso mette in evidenza solo i casi modificati, ossia *Register* e *View Classes*, quest'ultimo arricchito con punti di estensione, tra cui *Add Class*. Questi elementi evidenziati



riflettono gli interventi effettuati, rendendo più chiaro il contributo delle modifiche alle funzionalità del sistema.

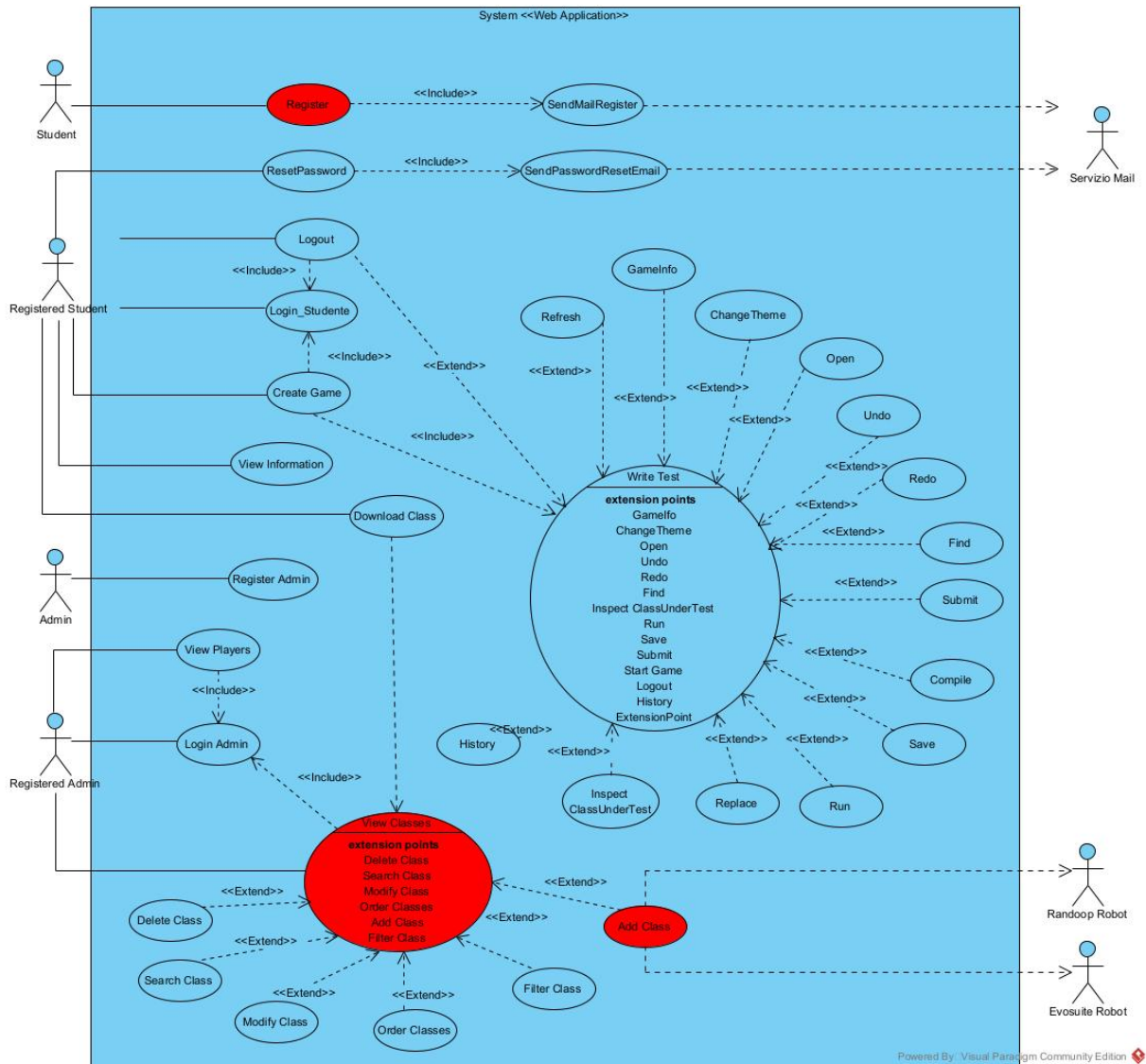


Figura 1.3: Diagramma dei Casi d'Uso del sistema complessivo

### 1.3.3 Composite Diagram

Il Composite Diagram è uno strumento prezioso per rappresentare la struttura interna di un sistema, illustrando come le diverse parti o

componenti di una classe o oggetto interagiscono tra loro. Questo diagramma mette in evidenza le relazioni e interazioni tra gli elementi interni di un sistema complesso, inclusi oggetti, componenti e connettori, facilitando la comprensione di come questi collaborano per fornire specifiche funzionalità. Inoltre, permette di visualizzare chiaramente i compiti relativi allo sviluppo e alla gestione di ciascun componente, evidenziando in rosso e verde i package e componenti modificati.

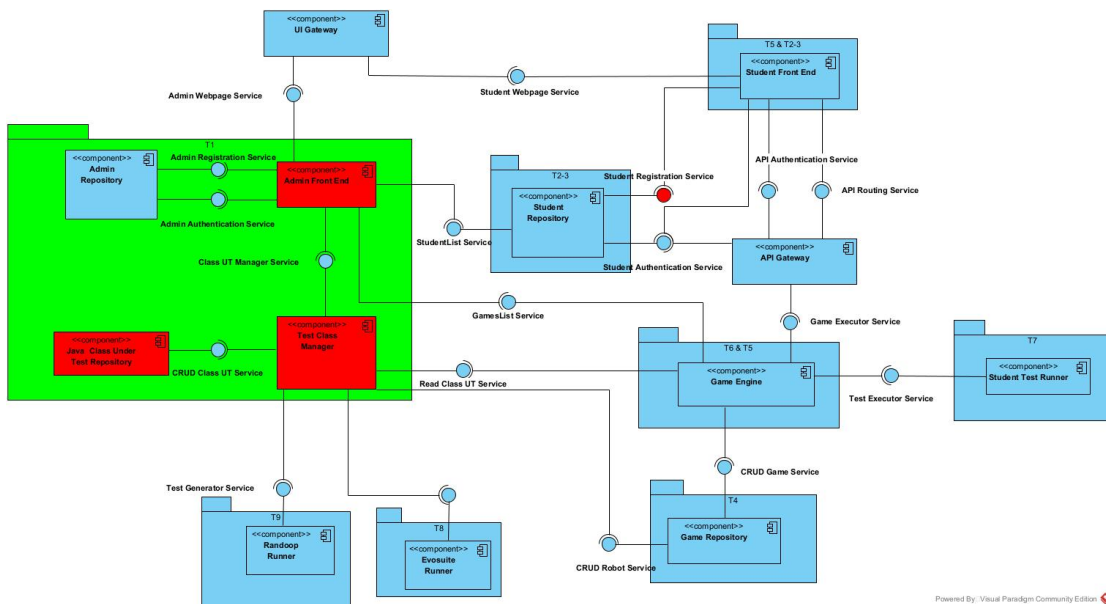


Figura 1.4: Composite Diagram

### 1.3.4 Deployment Diagram

Il Deployment Diagram qui illustrato mette in risalto la disposizione fisica dei componenti di un sistema software e le loro interazioni all'interno dell'ambiente di esecuzione. Sul lato server, l'applicazione utilizza Docker per eseguire i componenti, garantendo un elevato livello

di integrabilità. Ogni artefatto è inserito in un container dedicato per ottimizzare l'ambiente di esecuzione. Inoltre, è stato necessario creare connessioni tra i vari container, per permettere la comunicazione tra gli artefatti, come il *UI Gateway* e l'*API Gateway*. I task evidenziati in rosso rappresentano gli interventi effettuati per il refactoring e la correzione di bug.

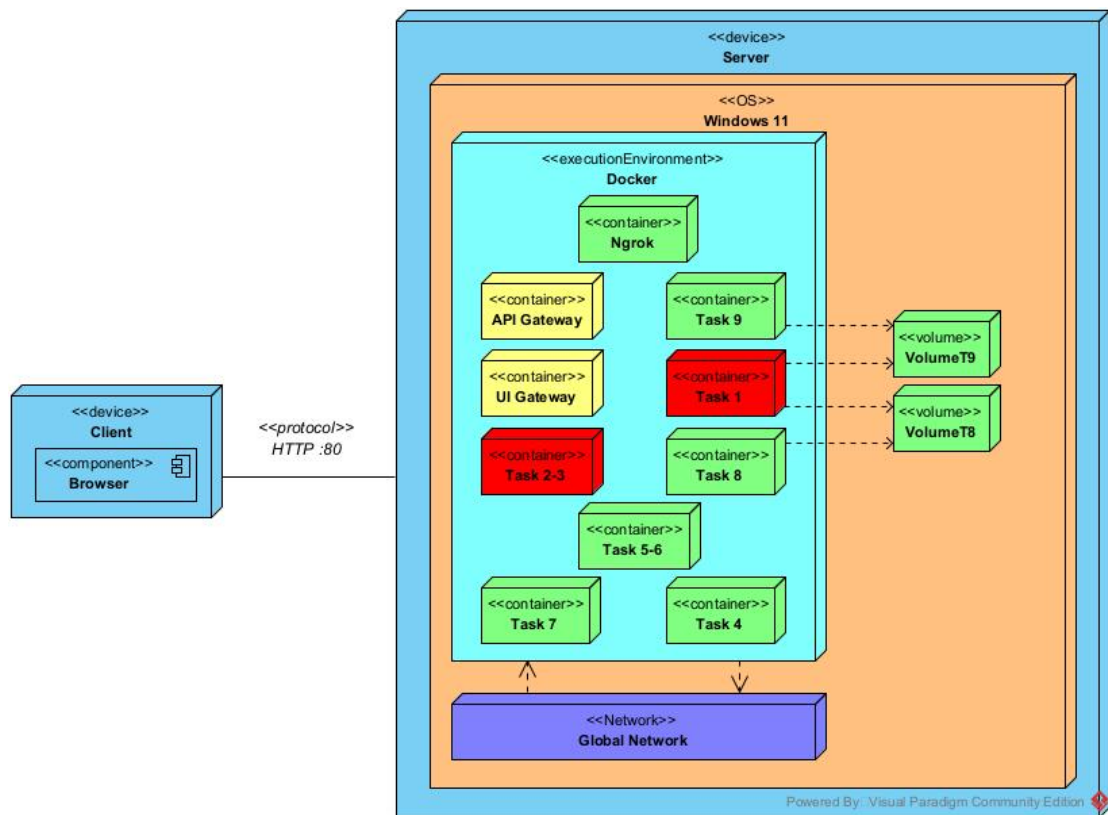


Figura 1.5: Deployment Diagram

## Capitolo 2

# Sviluppo e Obiettivi

In questa sezione sono descritti gli approcci e gli strumenti utilizzati durante lo svolgimento del task, insieme agli obiettivi prefissati e alle problematiche riscontrate.

### 2.1 Daily SCRUM

Nel contesto dello sviluppo del nostro software, abbiamo adottato un approccio agile basato sul framework **SCRUM**. Questo framework rappresenta una metodologia di gestione del ciclo di sviluppo del software caratterizzata da iterazioni e incrementi. Nel nostro processo di lavoro, abbiamo adottato la pratica di pianificare e condurre sprint settimanali. Durante ciascun sprint, i membri del team si concentrano su un insieme specifico di funzionalità o obiettivi definiti in precedenza. Ogni sprint è preceduto da un processo di pianificazione in cui

vengono stabiliti gli obiettivi dell'iterazione e vengono identificate le attività necessarie per raggiungerli. Questo approccio ci consente di gestire in modo efficiente il ciclo di sviluppo del software e di adattarci in modo flessibile alle esigenze in evoluzione del progetto.

## 2.2 Strumenti utilizzati

Durante lo sviluppo del progetto, abbiamo utilizzato vari strumenti per semplificare la collaborazione, la gestione del codice e la modellazione delle funzionalità:

- **Microsoft Teams:** per coordinare il lavoro e mantenere aggiornamenti costanti sui progressi. Teams è stato anche utile per lo scambio immediato di file e messaggi, agevolando un coordinamento efficace tra i membri del team.
- **GitHub:** per una gestione efficiente del codice e per mantenere un flusso di sviluppo incrementale, abbiamo scelto GitHub come repository centrale, garantendo un controllo di versione aggiornato e condiviso per tutto il team.
- **Visual Paradigm:** Utilizzato nella fase di progettazione e analisi dei requisiti, questo software di modellazione è stato impiegato per la creazione dei diagrammi UML inclusi nella documentazione. Visual Paradigm ha permesso di rappresentare

graficamente l'architettura e i flussi del sistema, facilitando la comprensione della struttura del progetto.

- **Visual Studio Code:** Questo editor di codice è stato adottato da tutto il team per scrivere e testare il codice. Con l'aggiunta di estensioni, come "Spring Boot Dashboard," abbiamo potuto lavorare agilmente con il framework Spring Boot, seguendo i requisiti definiti nelle prime fasi del progetto. Inoltre, l'integrazione nativa di GitHub ha semplificato la gestione e il mantenimento dei branch direttamente all'interno dell'ambiente di sviluppo.

### 2.3 Obiettivi del lavoro

L'obiettivo principale del **refactoring del T1** è stato migliorare la manutenibilità, la testabilità e la scalabilità dell'applicazione, andando a lavorare sul controller il quale, gestendo insieme sia il routing sia la logica di business, comprometteva la chiarezza del codice e la modularità dell'applicazione stessa. Delegando la logica di business ai nuovi servizi, l'applicazione risulta migliorata strutturalmente, più agile, pronta per futuri sviluppi. In una fase successiva, il focus è stato sul fixing di alcuni bug e alcuni miglioramenti:

- **reCaptcha:** è stato riabilitato il controllo reCaptcha al momento del login dell'utente che risultava disabilitato (sia a livello

frontend che backend), e non proteggeva l'accesso alla webapplication.

- **Messaggio di corretto Upload Class e reindirizzamento automatico:** è stato implementato un messaggio di corretto caricamento della classe, in precedenza assente e che quindi comportava il ritorno manuale alla pagina di admin, seguito dal reindirizzamento automatico dopo aver terminato l'upload di un classe, visto che alla versione attuale l'applicazione rimaneva bloccata nella pagina che richiedeva i parametri per il caricamento della classe.

## 2.4 Iterazioni

Le attività sono state eseguite in diverse iterazioni, ognuna delle quali si prefissava degli obiettivi specifici e prevedeva un ciclo di pianificazione, sviluppo, revisione e rifinitura:

- **Fase 1- Analisi del sistema (versione A13):** Il primo passo è stato l'analisi della documentazione prodotta dal gruppo A13. Ciò ci ha permesso di comprendere lo stato attuale del sistema e identificare le eventuali criticità legate al loro sviluppo. Abbiamo analizzato il codice esistente per identificare le principali problematiche e le aree che richiedevano refactoring del task T1.

- **Fase 2- Refactoring T1:** Durante questa iterazione, ci siamo concentrati sullo sviluppo del refactoring del codice esistente, con l'obiettivo di rendere codice del Controller più modulare e facilmente manutenibile, e separato dalla logica di business delegata ai vari Service implementati.
- **Fase 3- Ottimizzazione, Testing e Miglioramento :** In questa fase, il focus è stato sull'ottimizzazione del codice sviluppato a partire dalla vecchia versione monolitica del Controller, per poi procedere al Testing per la verifica della correttezza dei Service e dei collegamenti della versione migliorata del Controller. Infine, sono stati apportati dei miglioramenti al sistema complessivo, con l'implementazione del controllo reCaptcha in fase di registrazione dell'utente, e con il reindirizzamento automatico dopo l'uploadClass dopo un messaggio di corretto caricamento della classe.

## 2.5 Problematiche affrontate

Il Controller del task T1 alla versione presa in considerazione al momento del nostro intervento presenta diverse criticità e problematiche:

- **Alta Complessità e Assenza di Modularità:** il Controller conteneva logiche complesse che si intrecciavano tra loro senza una chiara separazione. Ciò implica per ogni modifica l'analisi



attenta dell'intero blocco di codice, poiché alterare una parte poteva facilmente avere ripercussioni in altre aree.

- **Duplicazioni e scarsa Modularità:** poiché il Controller era responsabile di tutte le funzionalità, diverse operazioni presentavano frammentazioni e duplicazioni, soprattutto nelle logiche di autenticazione e nelle verifiche dei permessi.
- **Difficoltà nell'Evoluzione e nell'Aggiunta di Nuove Funzionalità:** la struttura monolitica e poco modulare del HomeController ostacolava l'aggiunta di nuove funzionalità. Inoltre, ogni nuova funzionalità introduceva una complessità aggiuntiva, aumentando ulteriormente la dimensione del controller e rendendo il codice meno leggibile e più soggetto a errori.

Oltre queste problematiche strutturali al task T1 e su cui siamo intervenuti, bisogna segnalare che la webapplication ha ricevuto, in una precedente fase di sviluppo, un'importante ristrutturazione, che ha portato però a diversi issues e bug ancora da risolvere al momento dell'intervento del refactoring T1. Quest ultimo, tuttavia, risulta essere indipendente dal resto del sistema, e grazie agli interventi effettuati più comprensibile, scalabile e semplice da gestire, e per questo da tenere in considerazione dopo la fase di debug e di risoluzione delle varie criticità presenti nel sistema attualmente.

## Capitolo 3

# Refactoring T1

Il focus principale di questo progetto è stato il miglioramento del task T1, che si presentava al momento del nostro intervento con un unico controller (*HomeController*), contenente oltre 1500 righe di codice, e che gestiva sia il routing (GET, POST e DELETE mapping) sia la logica applicativa. Tale approccio presenta diverse criticità, che sono state affrontate e risolte durante il processo di refactoring:

- **Responsabilità eccessive:** il Controller gestiva più responsabilità, infrangendo il principio di Single Responsibility Principle (SRP) a causa della presenza di logica di routing e logica applicativa nello stesso file.
- **Difficoltà nella Manutenzione:** con oltre 1500 righe di codice, il controller risultava difficile da comprendere e mantenere, con le modifiche ad una sezione del codice potevano influenzare altre parti senza chiarezza.

- **Scalabilità Limitata e Mancanza di Riutilizzabilità:** Con l'aumentare della complessità delle funzionalità, la crescita del codice avrebbe reso la gestione ancora più difficile, mentre Le logiche di business incorporate nel controller non erano facilmente riutilizzabili in altri contesti.

In particolare, per effettuare il refactoring del T1 è stata seguita la seguente strategia:

- **Separazione delle funzionalità:** sono state separate dal controller le funzionalità di routing da quella della logica, che invece è stata delegata a servizi specifici.
- **Implementazione di Servizi:** la logica di business è stata estratta in servizi dedicati, facilitando la testabilità e il riutilizzo. Così facendo le operazioni vengono affidate ai servizi, mantenendo il codice separato e focalizzato.
- **Ristrutturazione del Routing:** i *mapping* sono stati riorganizzati per riflettere meglio la nuova struttura del task T1, semplificando la gestione delle rotte.
- **Testing e Documentazione:** verifica delle funzionalità dei Service implementati, e dei collegamenti lato Controller. Infine stesura della documentazione progettuale.

## 3.1 Component Diagram

Il diagramma dei componenti fornisce una visione di come sono stati suddivisi i vari moduli, fornendo una visione del sistema nel suo insieme. Come mostrato in immagine il task T1 refattorizzato contiene:

1. **Controller:** dopo il refactoring, il controller questo package ha una responsabilità esclusivamente orientata al routing e alla gestione delle richieste HTTP (GET, POST, DELETE, ecc.), delegando la logica di business al package service, limitando il ruolo del controller a una semplice interfaccia tra il frontend e i servizi di business.
2. **Service:** contiene i vari servizi che implementano la logica di business dell'applicazione, ognuno progettato per coprire un'area specifica delle funzionalità, consentendo una chiara separazione delle responsabilità. I servizi collaborano tra loro per eseguire operazioni complesse, ed interagiscono con il package Model per accedere e manipolare i dati persistenti, separando ulteriormente il layer di business dal layer di accesso ai dati.
3. **Model:** rappresenta il layer di accesso ai dati e contiene le classi di dominio (model), che mappano le entità e le relazioni del database. Questo package era già presente prima del refactoring e mantiene la sua funzione di interfacciarsi con i servizi e con il database.

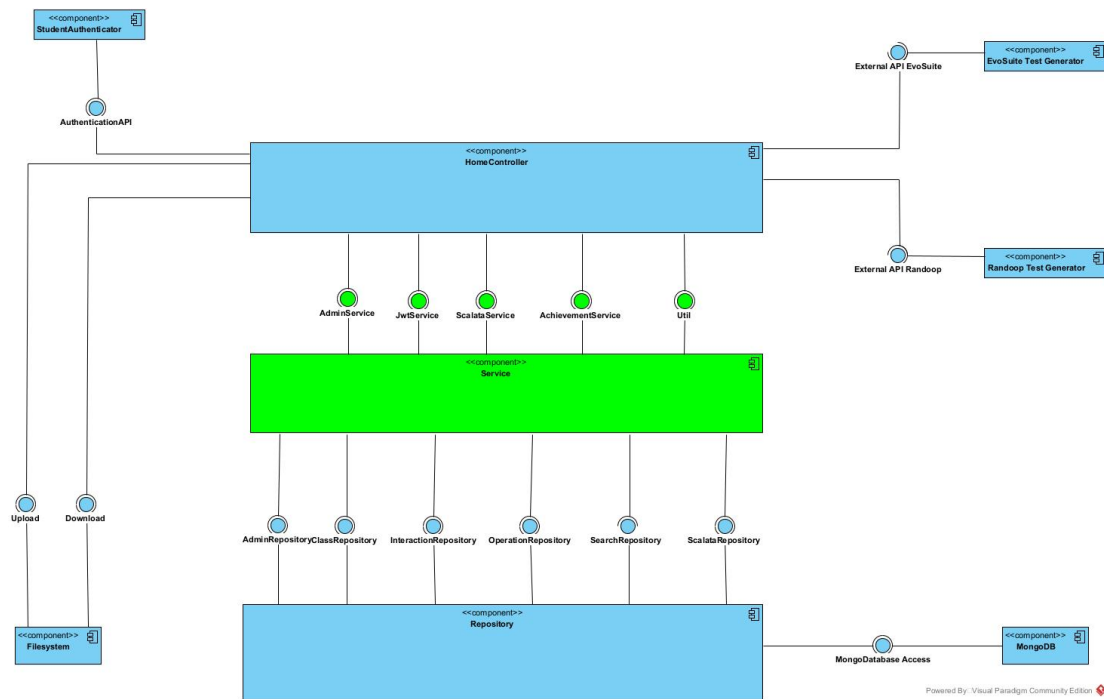


Figura 3.1: Component Diagram relativo al Task T1

L'organizzazione del task T1 prodotta risulta a livello architetturale più robusta e modulare, pronta per supportare le esigenze di crescita e di qualità dell'applicazione.

## 3.2 Package Service

Il package Service è stato introdotto come parte del refactoring dell'architettura per separare la logica di business dal controller principale (HomeController). Questa nuova suddivisione permette di mantenere i controller leggeri, con il solo compito di gestire le richieste HTTP, mentre tutta la logica applicativa è delegata ai servizi, consentendo un approccio modulare e orientato alla responsabilità singola. Il pac-

kage Service è composto dai seguenti servizi, ognuno progettato per contenere classi che gestiscono diverse aree funzionali dell'applicazione:

1. **AdminService**: è pensato per essere utilizzato dai controller dell'applicazione che gestiscono le richieste amministrative, fornendo una logica centralizzata per l'amministrazione delle classi e degli utenti admin.
  - Contiene i metodi di Autenticazione, tramite inclusione della gestione di token JWT da *JwtService* per l'autenticazione e l'autorizzazione degli amministratori, e la registrazione di nuovi amministratori.
  - Si occupa della gestione delle Classi tramite una serie di operazioni CRUD (Create, Read, Update, Delete), oltre a filtri e ordinamenti delle classi stesse.
2. **JwtService**: è responsabile della generazione e della validazione dei token JWT (JSON Web Token) per gestire l'autenticazione e l'autorizzazione degli utenti amministrativi all'interno dell'applicazione. Questo servizio garantisce che solo gli utenti autenticati possano accedere a determinate funzionalità, verificando la validità dei token JWT associati alle richieste. In particolare:
  - Fornisce il metodo *generateToken* per generare un token JWT per un utente amministrativo, che può essere uti-

lizzato come strumento di autenticazione per le richieste successive.

- Il metodo *isJwtValid* consente di validare i token JWT ricevuti, assicurandosi che siano autentici e non scaduti.

3. **ScalataService**: gestisce le operazioni relative all'entità Scalata, includendo operazioni CRUD (Creazione, Lettura e Cancellazione), e utilizza la validazione JWT per limitare l'accesso.
4. **Util**: offre diversi metodi di utilità per gestire le interazioni con l'entità interaction, gestendo le interazioni come "like" e "report" associati a una determinata classe, oltre ad ottenere un elenco di report.
5. **AchievementService**: gestisce la logica relativa agli achievement (obiettivi) e alle statistiche dell'applicazione. Questa classe fornisce metodi per visualizzare, creare, elencare e cancellare achievement e statistiche, assicurando che le operazioni siano eseguite solo se il token JWT è valido (tramite *JwtService*).

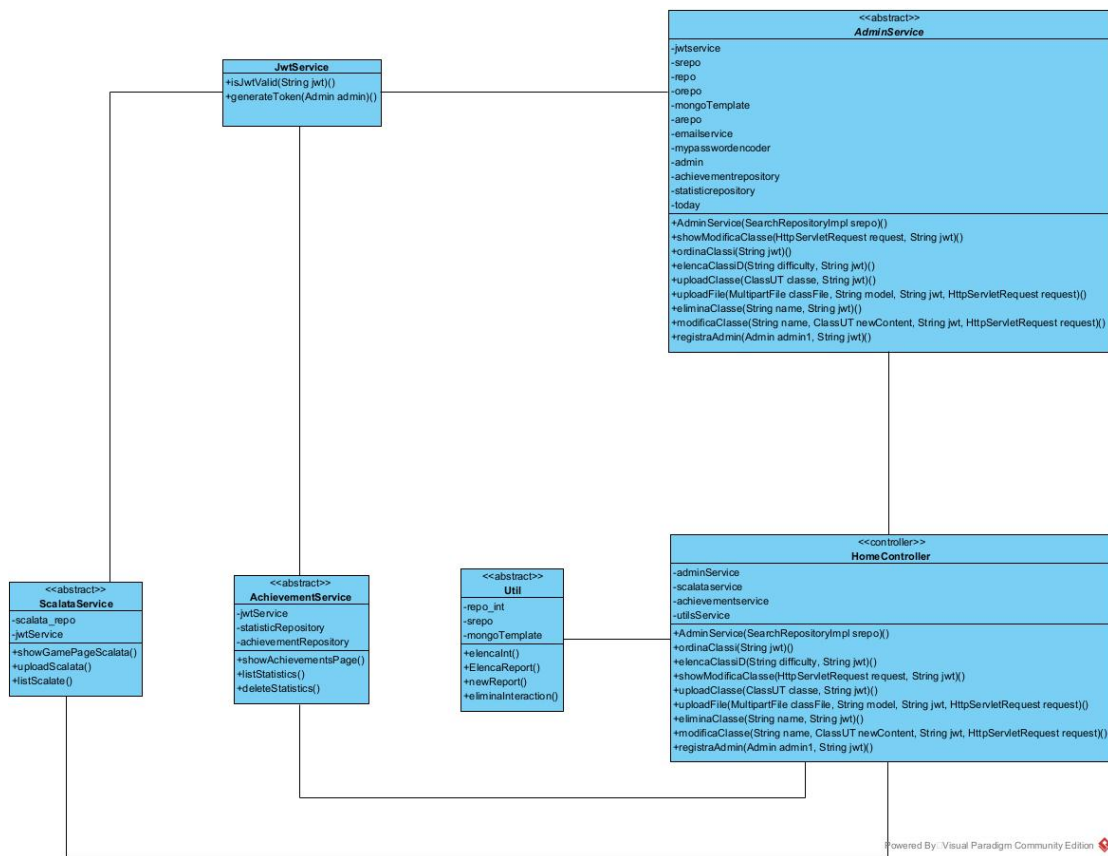


Figura 3.2: Package Service

### 3.3 Sequence Diagram

Riportiamo il Sequence Diagram per il caricamento di una classe col fine di mostrare come il flusso di esecuzione è strutturato dopo il refactoring del task T1:

1. Admin invia una richiesta di Upload Class al Controller.
2. Il Controller, ricevuta la richiesta, inoltra la chiamata al Admin Service per gestire l'operazione richiesta.
3. Admin Service avvia un controllo del token JWT (JSON Web



Token) per verificare che l'utente abbia i permessi amministrativi.

4. Se il token non è valido o non ha i permessi, il flusso termina qui con un errore di autorizzazione.
5. Se il token è valido e il ruolo è confermato come admin, il flusso continua.
6. Superato il controllo, Admin Service procede con l'upload vero e proprio della classe (caricamento dei dati necessari, salvataggio in database, ecc.).

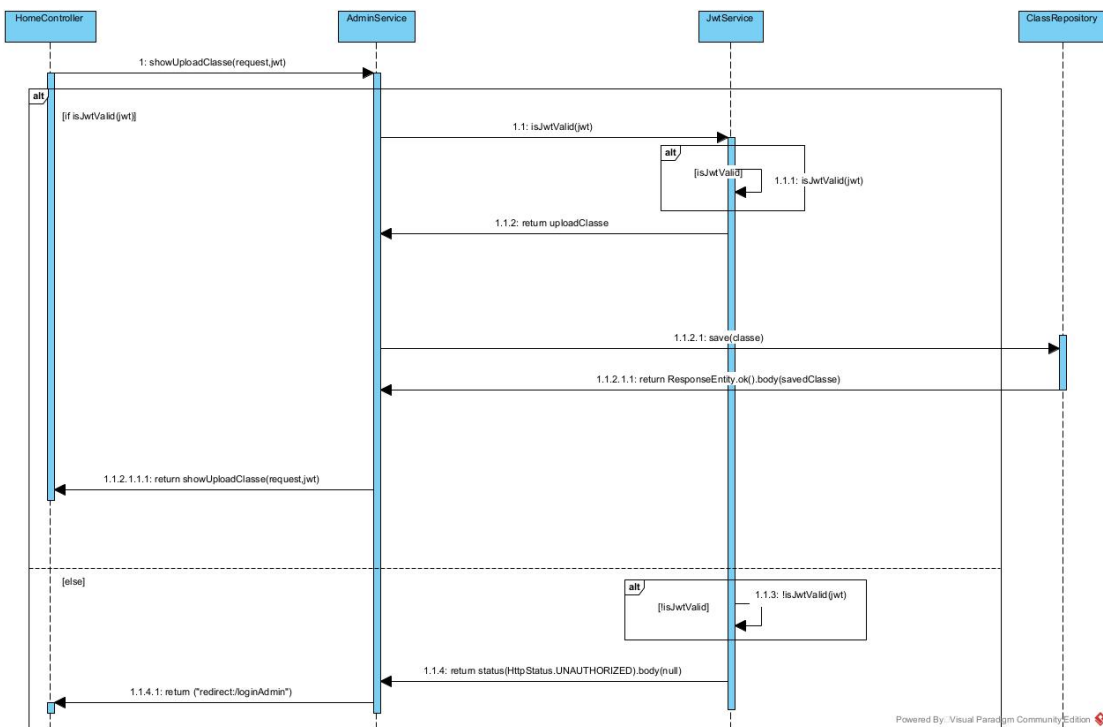


Figura 3.3: Sequence Diagram UploadClass

## Capitolo 4

# Issue reCAPTCHA e messaggio Upload Class

In una fase successiva del progetto, ci siamo occupati del ripristino del controllo reCAPTCHA (ISSUE 27) in fase di registrazione dei Players, che risultava disabilitato (sia a livello front-end che back-end), per dare maggior protezione alla webapplication. La schermata di registrazione dello studente è presente in figura qui sotto.

Oltre questo, è stato ritenuto necessario fornire un feedback quando l'admin effettua l'upload di una classe, poichè non vi era alcun messaggio a schermo e si rimaneva alla pagina dell'upload. Ecco come compare il messaggio in seguito all'upload classe:

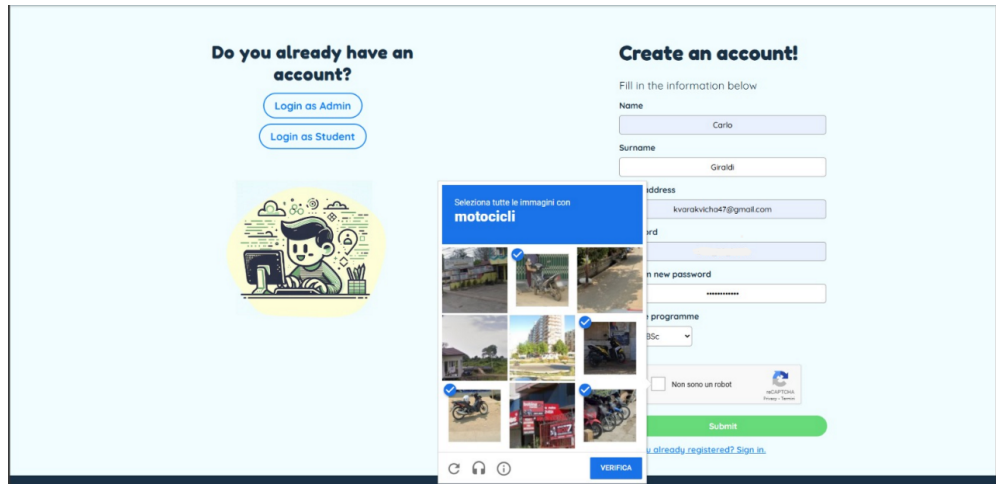


Figura 4.1: Registrazione Players con reCaptcha

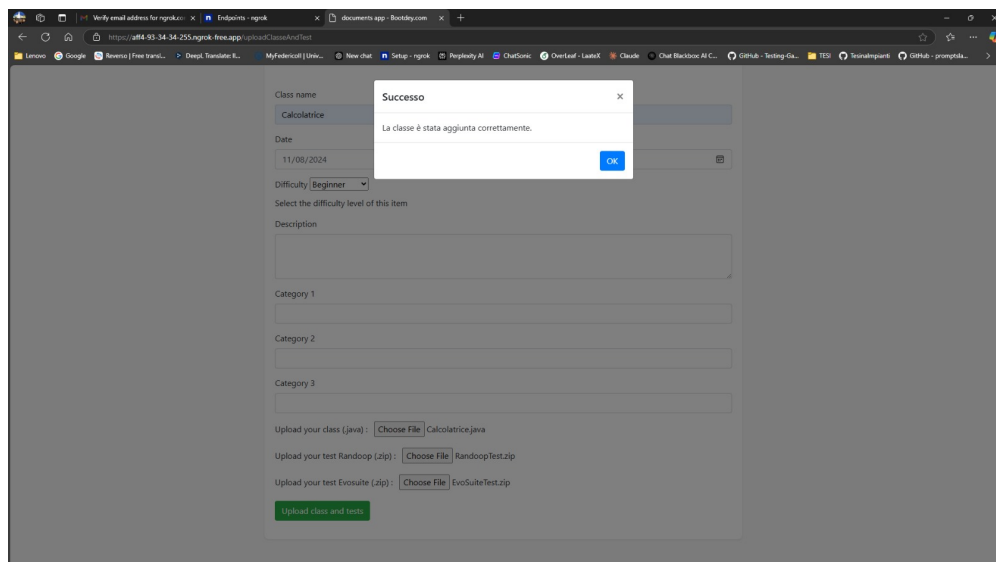


Figura 4.2: Messaggio Upload Class

## Capitolo 5

# Sviluppi futuri

Grazie al nostro lavoro è stato possibile analizzare e testare i vari componenti del sistema, da cui è emersa la necessità di continuare il lavoro di bug fixing sulla nuova versione della webapplication in seguito all'ultimo importante refactoring strutturale. In generale, da un'analisi preliminare di una versione precedente, è stato possibile riscontrare le migliorie più a livello grafico/strutturale che di performance. La strada intrapresa del refactoring dei vari componenti risulta, tuttavia, necessaria per le varie problematiche strutturali ad alcuni task evidenziate anche in questa documentazione.