Documentazione Progetto SAD

Gruppo D6

Anno 2025/2026

Contents

1 Identificazione del Team e Obiettivi

- **ID Team:** D6
- **Task Assegnato:** TaskR2 - Caricamento Suggerimenti e Migrazione MongoDB
- **Componenti Gruppo:**

Nome Cognome	**Email Istituzionale**
Sorrentino Simone	simone.sorrentino6@studenti.unina.it
Antonio Varese	an.varese@studenti.unina.it
Rosa Palmentieri	ros.palmentieri@studenti.unina.it
Fabio Chiacchio	fab.chiacchio@studenti.unina.it

- **URL Versione di Partenza:**
  `https://github.com/Testing-Game-SAD-2023/A13`
- **URL Repository Consegnato:**
  `https://github.com/Simonesorrentino/D6-taskR2`

1.1 Descrizione Task

La Task R2 consiste in una evoluzione del microservizio T1 relativo all'admin con 2 obiettivi principali:

1. Caricamento e gestione di suggerimenti di carattere generico e associati a Classi Under Test.
2. Migrazione del database da MongoDB ad un database SQL, definendo uno schema relazionale con vincoli di integrità e mantenendo la compatibilità con le API esistenti.

1.2 Descrizione Approccio di Sviluppo

L'approccio di sviluppo adottato combina tecniche di Bottom-Up Refactoring per la migrazione tecnologica e Component-Based Development per l'implementazione delle nuove funzionalità.

Il lavoro è stato strutturato in fasi sequenziali per garantire la stabilità del sistema:

- **Analisi e Definizione Modello Relazionale**: a partire dal modello dati originale NoSQL è stato progettato il corrispettivo schema ER da realizzare su PostgreSQL, definendo opportunamente vincoli di integrità e relazioni tra le tabelle prima assenti.

- **Refactoring del Codice**: è stata effettuata una riscrittura completa delle entità Java, utilizzando annotazioni JPA, e sostituzione dei Repository MongoDB con interfacce JpaRepository, risolvendo le criticità legate all'Impedance Mismatch (es. gestione liste e tipi complessi).
- **Adattamente del Service Layer**: è stata adattata la logica di business preesistente per gestire transazioni SQL atomiche e i nuovi tipi di dato identificativi (Long Serial).
- **Gestione Suggerimenti**: sulla base dell'architettura rinnovata, è stato sviluppato il modulo per la gestione dei suggerimenti. La logica di caricamento è stata progettata per gestire dati in formato JSON, trasformarli in entità relazionali strutturate, garantendo il soddisfacimento dei vincoli di integrità tra la nuova entità Hint e le entità preesistenti Admin e ClassUT.

2 Analisi dei Requisiti

2.1 Requisiti Funzionali

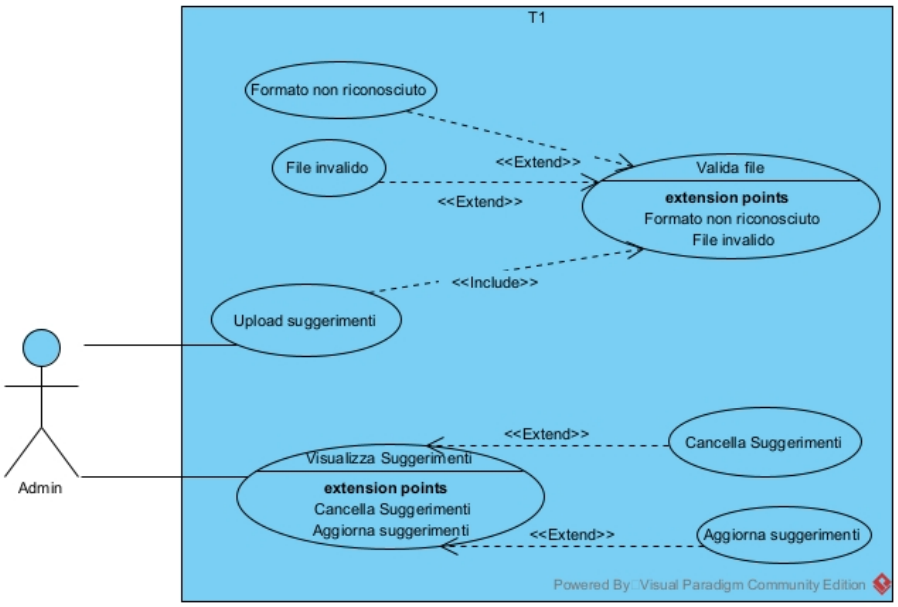
- **RF1 – Caricamento Suggerimenti**: Il sistema deve consentire all' amministratore di effettuare l'upload di un file JSON contenente suggerimenti generici oppure associati ad una specifica classe da testare, validandone il formato e salvando ciascun suggerimento nel database SQL.
- **RF2 – Validazione File**: Il sistema deve rilevare e segnalare errori nel file di input, quali: formato non valido, campi mancanti, struttura non compatibile con la classe selezionata.
- **RF3 – Parsing Suggerimenti**: Il sistema deve estrarre i suggerimenti dal file secondo il formato previsto JSON e generare un elenco di entità *hints* con campi validi e coerenti.
- **RF4 – Memorizzazione Suggerimenti**: I suggerimenti devono essere salvati in un database SQL con le seguenti informazioni: nome, contenuto, tipo e ordine di visualizzazione, email dell'amministratore che crea il suggerimento, eventuale nome della classe UT di riferimento.
- **RF5 – Gestione Suggerimenti**: Il sistema deve consentire all' amministratore di gestire i suggerimenti in termini di creazione, visualizzazione, modifica ed eliminazione.
- **RF6 – Filtraggio Suggerimenti**: Il sistema deve consentire il recupero dei suggerimenti filtrandoli in base alla tipologia, distinguendo tra suggerimenti generici e suggerimenti associati a una specifica Classe Under Test.
- **RF7 – Ordinamento Suggerimenti**: Il sistema deve consentire all' amministratore di indicare l'ordine con cui i suggerimenti dovranno essere visualizzato da parte del servizio T5.
- **RF8 – Migrazione Database**: Il sistema deve migrare l'attuale persistenza di T1 da MongoDB a un database relazionale (SQL) senza modificare il comportamento preesistente del servizio.

2.2 Requisiti Non Funzionali

- **RNF1 – Manutenibilità**: La soluzione deve favorire la manutenibilità del sistema, garantendo una chiara separazione tra livello di persistenza, logica applicativa e interfacce, e adottando uno schema di database relazionale chiaro e normalizzato.
- **RNF2 – Integrità e affidabilità dei dati**: Il sistema deve garantire l'integrità e l'affidabilità dei dati persistiti, mediante l'utilizzo di vincoli di database quali chiavi primarie, chiavi esterne e vincoli di integrità.

- **RNF3 – Compatibilità architetturale**: La soluzione deve essere compatibile con l'architettura complessiva del sistema, mantenendo la coerenza delle API esistenti e garantendo la corretta integrazione con gli altri servizi del sistema.
- **RNF4 – Testabilità**: Le funzionalità del servizio devono essere testabili in modo indipendente dal frontend, consentendo la verifica delle API REST tramite strumenti di test dedicati (es. Postman).
- **RNF5 – Estendibilità**: La progettazione deve consentire l'estensione futura del sistema, permettendo l'introduzione di nuove tipologie di suggerimenti o funzionalità senza richiedere modifiche strutturali invasive.
- **RNF6 – Sicurezza e controllo degli accessi**: Il sistema deve garantire il controllo degli accessi alle funzionalità di gestione dei suggerimenti, consentendo le operazioni di creazione, visualizzazione, aggiornamento ed eliminazione dei suggerimenti esclusivamente agli utenti con ruolo di amministratore.

2.3 Diagramma dei Casi d'Uso



2.4 Scenari

ID e Nome	**UC1 – Caricamento suggerimenti**
Attore	Amministratore
Precondizioni	Login effettuato come Admin; la classe target esiste.
Scenario Principale	1. L'Amministratore accede alla sezione "Suggerimenti". 2. Seleziona la voce "Carica suggerimenti". 3. Carica il file JSON contenente i suggerimenti. 4. Carica l'eventuale immagine associata (se prevista). 5. Il sistema valida i file caricati (formato e contenuto). 6. Il sistema estrae i dati e salva i suggerimenti nel database. 7. Il sistema mostra un messaggio di successo.
Estensioni	• UC1-E1: File invalido o corrotto → Messaggio di errore. • UC1-E2: Suggerimenti vuoti o campi obbligatori mancanti. • UC1-E3: Immagine referenziata nel JSON ma non allegata nell'upload.

Table 1: Caso d'Uso UC1: Caricamento Suggerimenti

ID e Nome	**UC2 – Recupero suggerimenti**
Attori	Amministratore, Player, Servizio T5
Scenario	1. Il giocatore richiede i suggerimenti nel Game Engine. 2. T5 invia richiesta `GET /hints/` a T1. 3. T1 recupera i dati dal DB SQL. 4. T1 restituisce l'elenco ordinato. 5. T5 visualizza i suggerimenti.

Table 2: Caso d'Uso UC2: Recupero Suggerimenti

ID e Nome	**UC2 – Modifica suggerimenti**
Attore	Amministratore
Precondizioni	Login effettuato.
Scenario	1. Accesso sezione “Suggerimenti”. 2. Selezione di uno tipo di suggerimento. 3. Selezione di uno specifico suggerimento. 4. Visualizzazione dettaglio. 5. Click sul bottone modifica. 6. Modifica dei dati. 7. Salvataggio modifiche. 8. Messaggio di successo.
Estensioni	• File/Dati invalidi → Messaggio di errore.

Table 3: Caso d’Uso UC2: Modifica Suggerimenti

ID e Nome	**UC3 – Eliminazione suggerimenti**
Attore	Amministratore
Precondizioni	Login effettuato.
Scenario	1. Accesso sezione “Suggerimenti”. 2. Selezione tipo e suggerimento target. 3. Click su “Elimina”. 4. Conferma eliminazione. 5. Cancellazione dal DB.

Table 4: Caso d’Uso UC3: Eliminazione Suggerimenti

2.5 Analisi dell’Impatto

Le modifiche impatteranno unicamente il microservizio T1 (Gestione Admin).

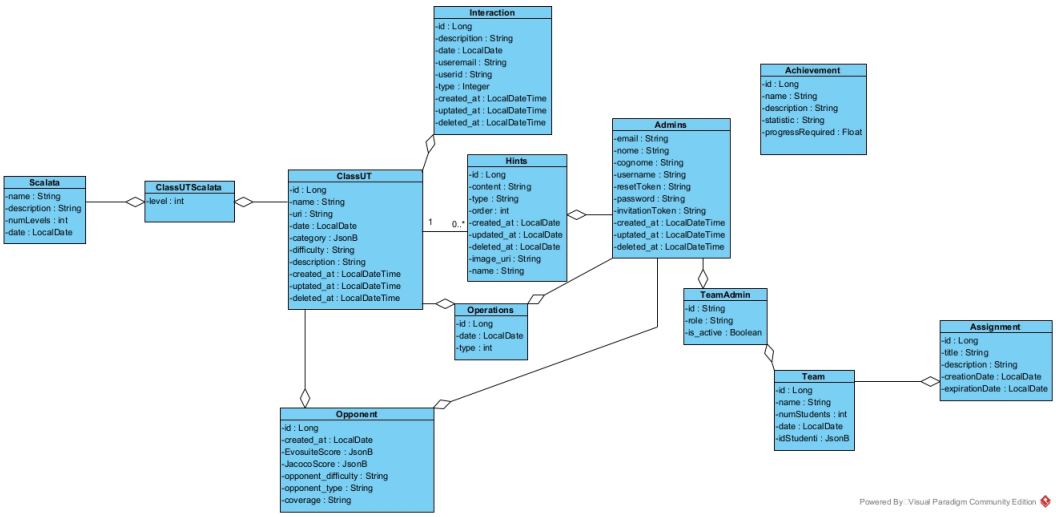
La migrazione verso un database SQL richiede una ridefinizione del modello dati, passando da una struttura a documenti, propria del database MongoDB, ad uno schema relazionale rigido con vincoli di integrità referenziale.

I componenti del microservizio T1 che saranno impattati nelle modifiche sono:

- **Model**: tutte le classi che rappresentano le entità del dominio;

- **Repository**: conversione di repository basati su Mongo in repository basati su SQL.
- **Service**: adattamento della logica di business in seguito alle modifiche effettuate su entità e repository.

Di seguito è riportato il diagramma delle classi.



La gestione dei suggerimenti comporterà delle modifiche sui seguenti componenti:

- **Model**: aggiunta della nuova entità suggerimento;
- **Repository**: aggiunta del nuovo repository per la persistenza dei dati relativi alla nuova entità;
- **Service**: aggiunta di un nuovo service per implementare la logica di business relativa alla gestione dei suggerimenti;
- **Controller**: aggiunta di un nuovo controller per la definizione ed esposizione delle API gi gestione dei suggerimenti;
- **View**: aggiunta di una nuova view per la visualizzazione e gestione dei suggerimenti lato interfaccia utente.

3 Progettazione della Soluzione

3.1 Decisioni di Progetto

Per la migrazione verso un database relazionale, la scelta del database è ricaduta su **PostgreSQL** per i seguenti motivi:

- Si tratta di un database relazionale orientato agli oggetti (ORDBMS), pertanto risulta più semplice l'interfacciamento con il back-end, orientato agli oggetti.
- L'insieme dei tipi di dati è esteso, includendo un supporto JSON più avanzato.
- Presenta un sistema di vincoli tra le tabelle più robusto, garantendo l'integrità dei dati e le proprietà ACID per le transazioni.
- È considerato il database più avanzato, robusto e conforme a livello tecnico, ideale per carichi di lavoro complessi e integrità dei dati.
- La natura open source del software ha permesso l'utilizzo senza costi di licenza, mantenendo l'architettura allineata con gli standard dei progetti accademici e open source.

La scelta è stata effettuata anche ragionando in un'ottica di robustezza più a lungo termine. Il progetto in questione è in continua evoluzione, sia in termini di funzionalità aggiuntive sia in termini di modifica delle funzionalità esistenti. PostgreSQL anche se richiede un utilizzo più elevato di memoria e presenta una velocità di lettura leggermente più lenta rispetto ad altri database, consente di gestire carichi di lavoro più complessi e garantisce un'integrità dei dati più robusta.

Un'altra decisione importante presa in riferimento alla migrazione del database riguarda il refactoring del codice preesistente. La migrazione comporterà la modifica dei *model* e dei *repository*. Tali modifiche impatteranno inevitabilmente anche i *service*. Nella struttura di partenza del microservizio i *service* sono stati gestiti direttamente come delle classi. Ma in riferimento ad uno dei principi fondamentali dell'ingegneria del software, **Information Hinding**, riteniamo sia più opportuno riscrivere tali *service* definendo prima l'interfaccia e poi la classe che la implementa.

In fase di definizione del modello del database, su ciascuna tabella si è deciso di riportare tre campi importanti:

- created_at
- updated_at
- delete_at

Si tratta di una predisposizione per eventuali sviluppi futuri. La presenza di questi tre campi su ciascuna entità consente di tenere traccia di tutte le operazioni effettuare e di quando queste sono state compiute. Inoltre, valutare la possibilità di sostituire una eliminazione fisica con una eliminazione *soft*, potrebbe essere di supporto per eventuali roll back o analisi future.

Un'altra decisione presa riguardo al database è stata quella di adottare il modulo **Flyway**, che permette di definire tutte le modifiche allo schema del database (come creazione di tabelle, aggiunta di colonne, modifica di indici e così via) come file di script SQL separati, con lo scopo di versionare il database.
Ogni script è contrassegnato con un numero di versione, per esempio V1.0.1__create_admin_table.sql, e Flyway esegue sempre gli script in ordine numerico, garantendo che le modifiche vengano applicate nello stesso modo, in ogni ambiente (sviluppo, test, produzione).
Inoltre, viene creata la tabella *flyway_schema_history* all'interno del database, che registra quali migrazioni sono state applicate e quando. Quando l'applicazione si avvia, Flyway controlla questa tabella, vede quali migrazioni mancano e applica solo quelle, senza mai rieseguire quelle già completate. Questo rende il processo ripetibile e non distruttivo.

In riferimento alla gestione dei suggerimenti, si è deciso di estendere il modello dati preesitente con una nuova tabella: ***hint***. Il suggerimento potrà essere di due tipi:

- generico
- associato ad una classe UT

Sarà possibile associare eventualmente un'immagine ad un suggerimento. Inoltre, deve essere definito l'ordine di visualizzazione dei suggerimenti, fondamentale in fase di recupero di questi da parte del microservizio T5.

Per garantire la persistenza delle informazioni relative ai suggerimenti richieste nei requisiti e sulla base delle considerazioni appena fatte, si è deciso di prevedere per tale tabella i campi:

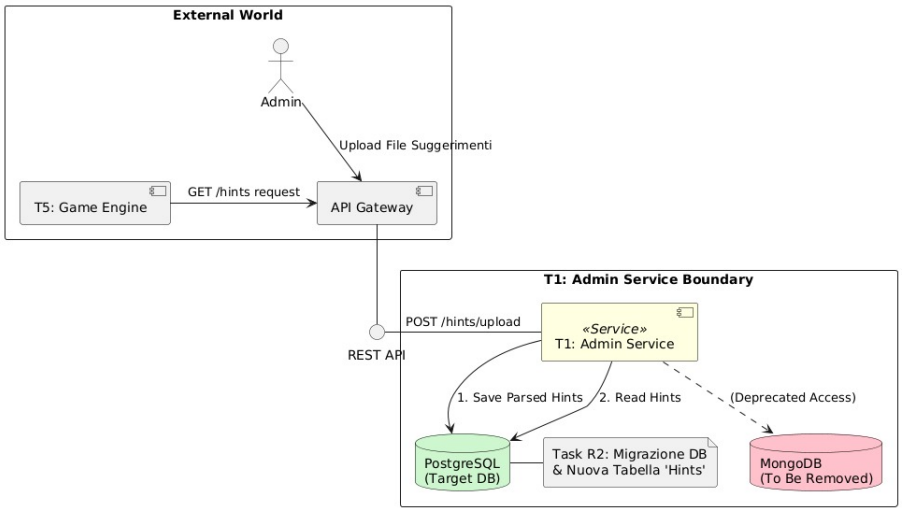
- **name**: nome del suggerimento;
- **content**: contenuto del suggerimento;
- **type**: tipo di suggerimento, *generic* oppure *class* ovvero associato ad una classe UT;
- **class_ut_name**: nome della classe UT se il suggerimento è di tipo associato ad una classe;
- **admin_email**: email dell'amministratore che ha creato il suggerimenti;
- **image_uri**: URI dell'eventuale immagine allegata al suggerimento;

- **order**: ordine di visualizzazione del suggerimento.

Per il caricamento dei suggerimenti si è deciso di adottare un file di tipo JSON perchè questo formato offre la possibilità di visualizzare il suggerimento in maniera strutturata con le informazioni di nostro interesse. Inoltre, il mapping tra un file JSON ed un oggetto Java risulta anche più semplice rispetto ad altri tipi di file come .txt o .html.

3.2 Viste Architetturali

3.2.1 Component Diagram



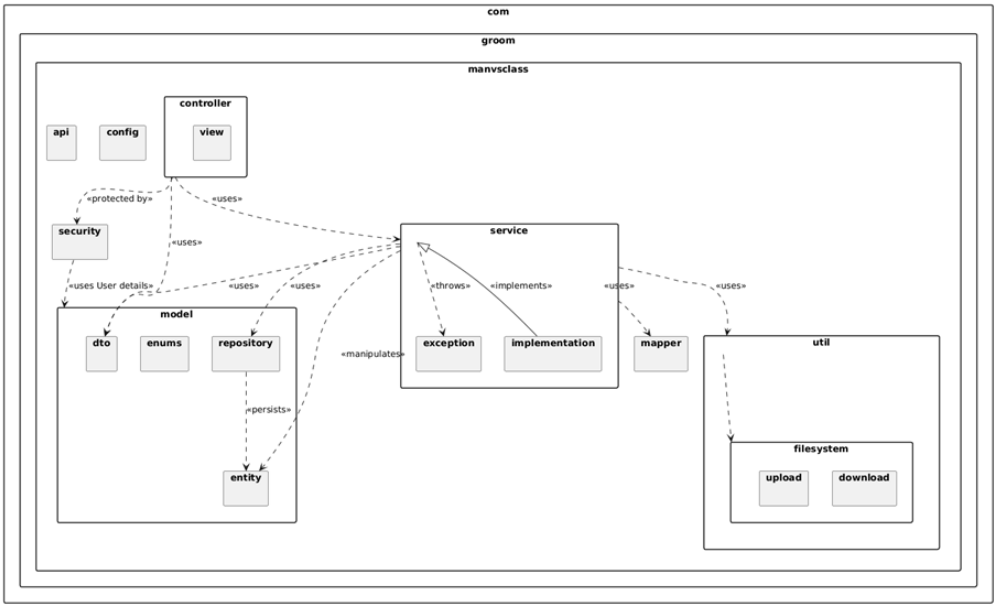
Questo diagramma rappresenta l'architettura aggiornata per le funzionalità aggiunte. Questo tipo di diagramma mostra come i vari elementi del sistema comunicano tra loro per realizzare le funzionalità.

Pertanto, l'amministratore invia il file dei suggerimenti ("Upload File Suggerimenti"). Tale richiesta non arriva direttamente a T1, ma passa per l'API Gateway, che agisce da punto di ingresso unico e smista le chiamate verso i servizi giusti.

Il Game Engine è il microservizio del gioco. Chiede i suggerimenti da mostrare al giocatore ("GET /hints request"). Anche in questo caso la richiesta arriva all'API Gateway.

Il microservizio T1 riceve i dati dal Gateway (tramite interfaccia REST API) e mediante la logica sviluppata salva i suggerimenti sul nuovo database Postgres.

3.2.2 Package Diagram



Il Package Diagram mostra l'organizzazione strutturale del codice sorgente del microservizio T1, suddiviso secondo il pattern architetturale a strati (**Layered Architecture**) all'interno del namespace **com.groom.manvsclass**.

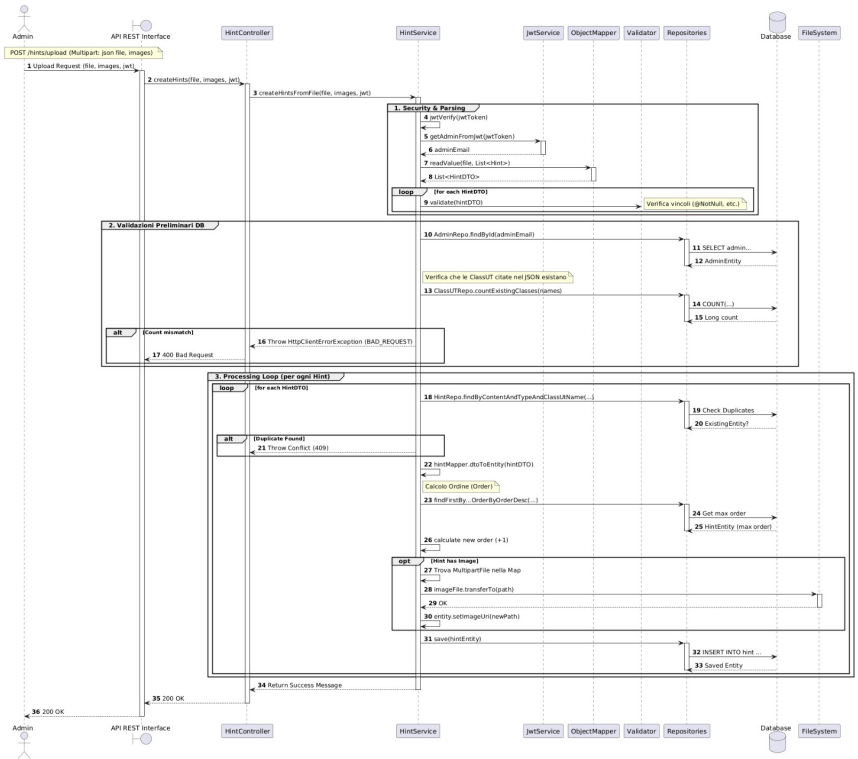
I package principali sono:

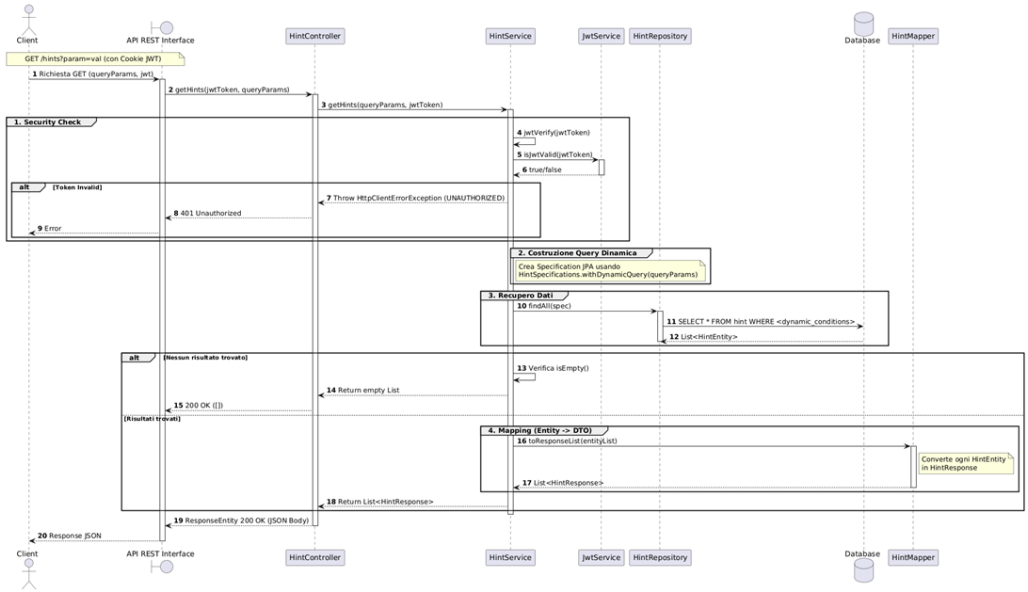
- **Controller**: Questo package gestisce l'interfaccia verso l'esterno (REST API). Il componente view riceve le richieste HTTP e delega l'elaborazione al layer sottostante (service). Il layer è protetto dal modulo security.
- **Service**: Rappresenta il cuore della logica di business. È stata adottata una separazione tra interfaccia (service) e implementazione concreta (implementation), garantendo il disaccoppiamento e facilitando la manutenibilità (Information Hiding). Il service orchestra le operazioni utilizzando il mapper per la conversione DTO-Entity e manipola le eccezioni tramite il package exception.
- **Model**: Contiene la definizione dei dati persistenti. Include le entità JPA (entity), i Data Transfer Objects (dto) e i repository (repository) che interfacciano il database. Il service manipola le entità e utilizza i repository per la persistenza.
- **Util (Filesystem)**: Questo package è stato fondamentale per l'implementazione del task di upload. Contiene le classi di utilità per la gestione del file sys-

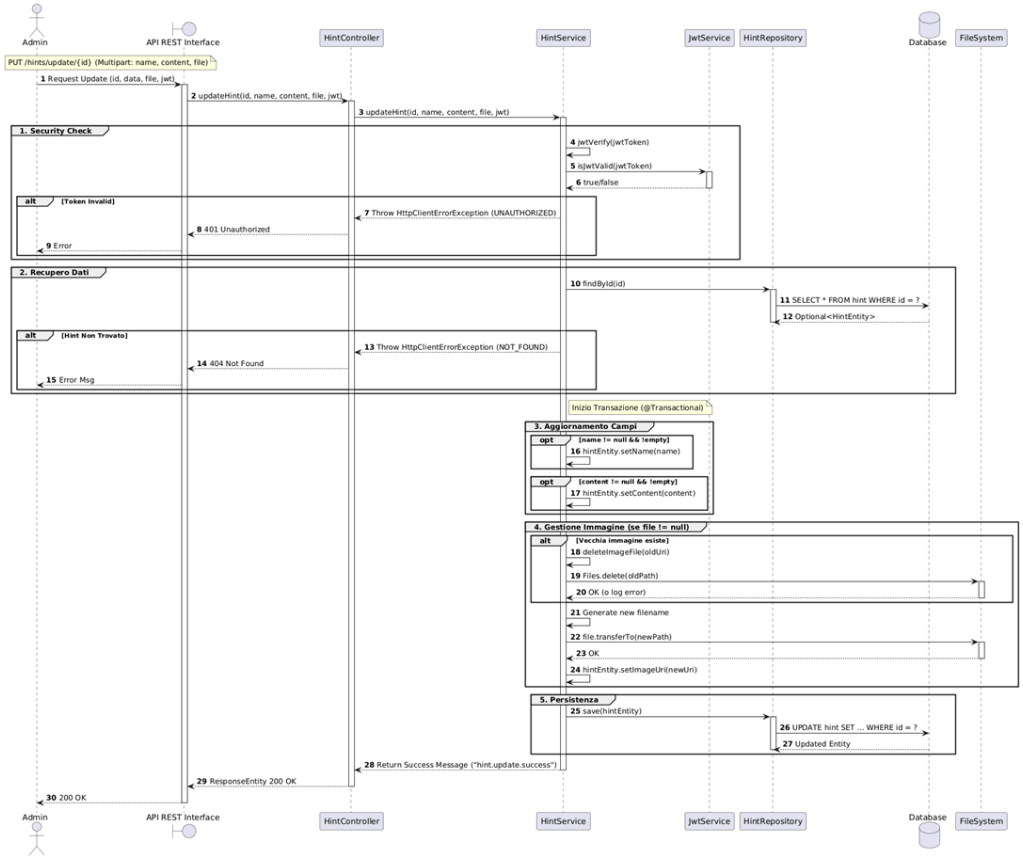
tem (upload, download), permettendo al service di salvare e recuperare fisicamente le immagini dei suggerimenti caricati.

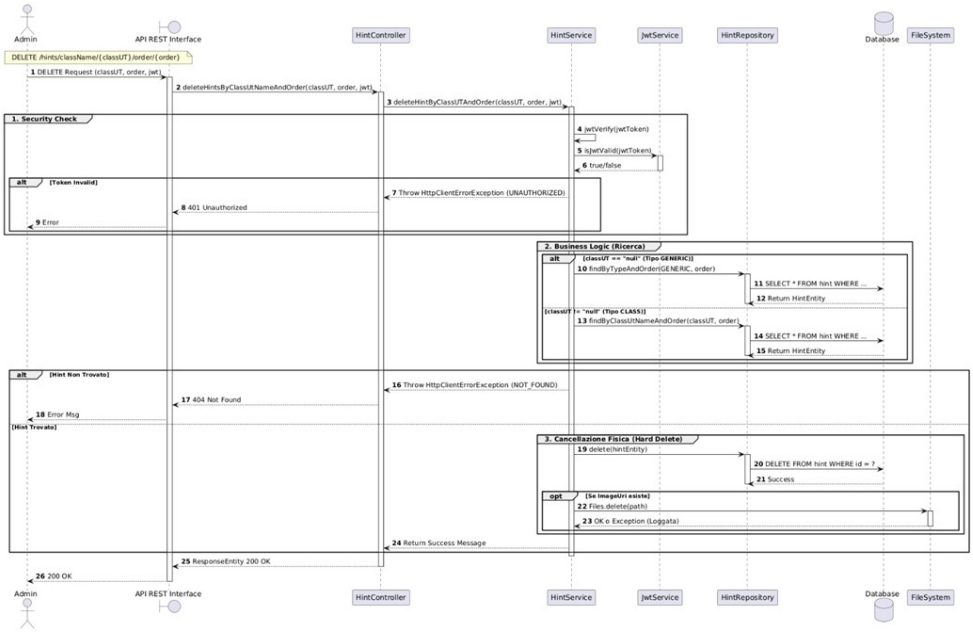
- **Security**: Interagisce trasversalmente per proteggere i controller e utilizza i dettagli utente definiti nel model per l'autenticazione e autorizzazione.

3.2.3 Sequence Diagram

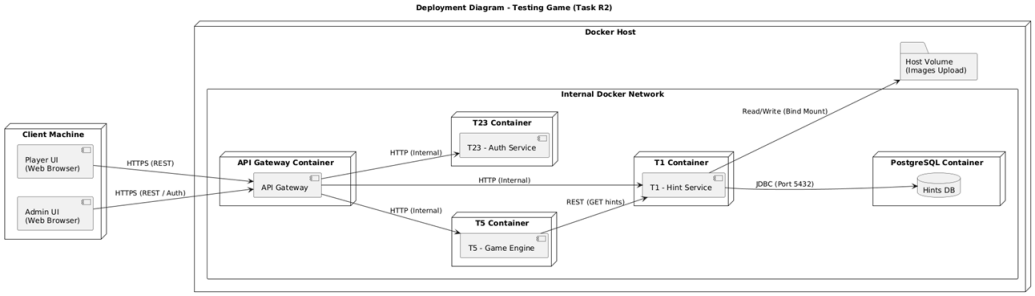



Client
API REST Interface
HintController
HintService
JwtService
HintRepository
Database
HintMapper
GET /hints?param=val (con Cookie JWT)
1 Richiesta GET (queryParams, jwt)
2 getHints(jwtToken, queryParams)
3 getHints(queryParams, jwtToken)
1. Security Check
4 jwtVerify(jwtToken)
5 isJwtValid(jwtToken)
6 true/false
alt (Token invalid)
7 Throw HttpClientErrorException (UNAUTHORIZED)
8 401 Unauthorized
9 Error
2. Costruzione Query Dinamica
Crea Specification JPA usando HintSpecifications.withDynamicQuery(queryParams)
3. Recupero Dati
10 findAll(spec)
11 SELECT * FROM hint WHERE <dynamic_conditions>
12 List<HintEntity>
alt (Nessun risultato trovato)
13 Verifica isEmpty()
14 Return empty List
15 200 OK ([])
(Risultati trovati)
4. Mapping (Entity -> DTO)
16 toResponseList(entityList)
Converte ogni HintEntity in HintResponse
17 List<HintResponse>
18 Return List<HintResponse>
19 ResponseEntity 200 OK (JSON Body)
20 Response JSON


Admin
API REST Interface
HintController
HintService
JwtService
HintRepository
Database
FileSystem
PUT /hints/update/{id} (Multipart: name, content, file)
1 Request Update (id, data, file, jwt)
2 updateHint(id, name, content, file, jwt)
3 updateHint(id, name, content, file, jwt)
1. Security Check
4 jwtVerify(jwtToken)
5 isJwtValid(jwtToken)
6 true/false
alt [Token invalid]
7 Throw HttpClientErrorException (UNAUTHORIZED)
8 401 Unauthorized
9 Error
2. Recupero Dati
10 findById(id)
11 SELECT * FROM hint WHERE id = ?
12 Optional<HintEntity>
alt [Hint Non Trovato]
13 Throw HttpClientErrorException (NOT_FOUND)
14 404 Not Found
15 Error Msg
Inizio Transazione (@Transactional)
3. Aggiornamento Campi
opt [name != null && !empty]
16 hintEntity.setName(name)
opt [content != null && !empty]
17 hintEntity.setContent(content)
4. Gestione Immagine (se file != null)
alt [Vecchia immagine esiste]
18 deleteImageFile(oldUri)
19 Files.delete(oldPath)
20 OK (o log error)
21 Generate new filename
22 file.transferTo(newPath)
23 OK
24 hintEntity.setImageUri(newUri)
5. Persistenza
25 save(hintEntity)
26 UPDATE hint SET ... WHERE id = ?
27 Updated Entity
28 Return Success Message ("hint.update.success")
29 ResponseEntity 200 OK
30 200 OK



3.2.4 Deply Diagram



Il Deployment Diagram mostra l'architettura di rilascio aggiornata, basata su container **Docker** orchestrati tramite Docker Compose.
La configurazione rispecchia le modifiche introdotte per la gestione della persistenza relazionale e dello storage dei file.

- **Docker Host e Internal Docker Networking:** Tutti i servizi backend risiedono all'interno di una rete Docker isolata. L'unico punto di accesso dall'esterno (Client Machine) è l'API Gateway, che gestisce il traffico HTTPS e lo instrada via HTTP verso i container interni.

- **T1 Container (Admin Service):** Il microservizio T1 è il cuore della gestione delle classi UT, dei robot e dei suggerimenti. Esso comunica con due risorse fondamentali:
  1. **PostgreSQL Container:** Tramite connessione JDBC (porta 5432), T1 persiste i metadati dei suggerimenti e le relazioni con le classi, sostituendo il precedente storage MongoDB.
  2. **Host Volume (Images Upload):** È stato configurato un volume Docker (Bind Mount) per mappare la cartella di upload del container su una directory fisica dell'Host. Questo garantisce la persistenza delle immagini caricate dall'amministratore anche in caso di riavvio o aggiornamento del container T1.
- **Interazione tra Servizi:** Il diagramma evidenzia la dipendenza a runtime del container **T5 (Game Engine)** che invoca le API REST interne di T1 per recuperare i suggerimenti da mostrare al giocatore durante la partita.

3.2.5 Nuove Interfacce REST

Di seguito vengono descritte le interfacce REST esposte dal microservizio T1.

Nome	**Upload Massivo Suggerimenti**
Metodo	`POST`
Endpoint	`/hints/upload`
Headers	`Cookie: jwt`
Body	• `file`: File JSON (array suggerimenti). • `images`: Lista immagini (opzionale).
Response	• `200 OK`: Suggerimenti caricati con successo. • `400 Bad Request`: JSON malformato, campi vuoti o mancanti, estensione file errata. • `404 Not Found`: Classe UT associata non trovata. • `409 Conflict`: Suggerimento già esistente (contenuto duplicato). • `415 Unsupported Media Type`: Formato file non supportato (non JSON o immagini non valide). • `401 Unauthorized`: Token mancante o non valido.

Table 5: API Upload Suggerimenti

Nome	**Visualizzazione Suggerimenti**
Metodo	`GET`
Endpoint	`/hints`
Params	• `classUTName`, `type`, `order`.
Response	• `200 OK`: Lista suggerimenti (o lista vuota se nessun risultato trovato). • `400 Bad Request`: Parametri di ricerca non validi. • `401 Unauthorized`: Token mancante o non valido.

Table 6: API Visualizzazione Suggerimenti

Nome	**Eliminazione Suggerimenti**
Metodo	`DELETE`
Endpoints	• `/hints/{classUTName}` (Per classe) • `/hints/className/{class}/order/{n}` (Singolo) • `/hints/type/{type}` (Tutti i generici)
Response	• `200 OK`: Eliminazione effettuata con successo. • `404 Not Found`: Risorsa da eliminare non trovata. • `401 Unauthorized`: Non autorizzato.

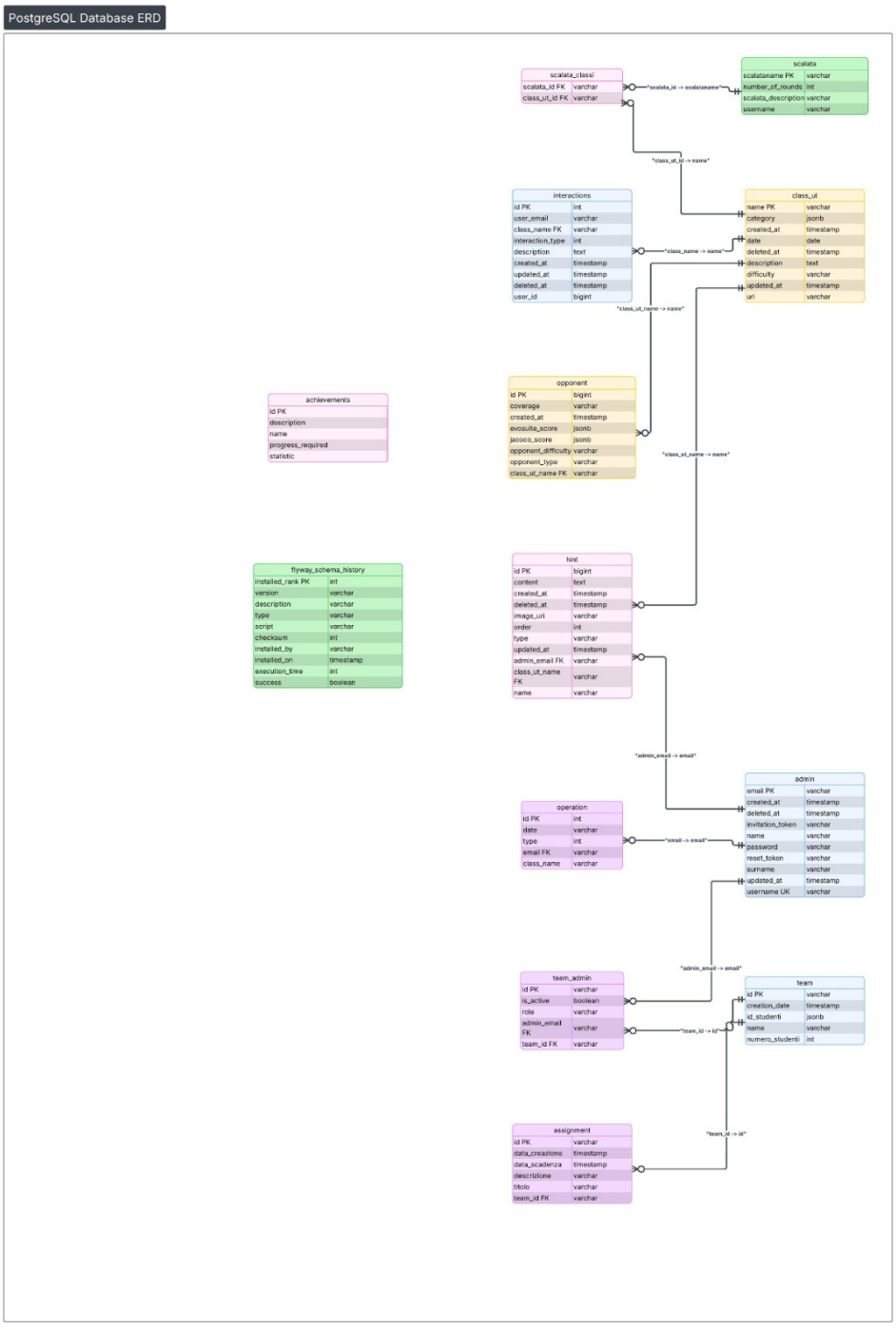
Table 7: API Eliminazione Suggerimenti

Nome	**Modifica e Ordinamento**
Update	`PUT /hints/update/{id}`
Body: Multipart (name, content, file)	
Move	`POST /hints/{id}/move`
Params: direction (UP/DOWN)	
Response	• `200 OK`: Modifica o spostamento avvenuti con successo. • `400 Bad Request`: Dati di input non validi o mancanti. • `404 Not Found`: ID suggerimento non trovato. • `409 Conflict`: La modifica crea un duplicato di un suggerimento esistente. • `401 Unauthorized`: Non autorizzato.

Table 8: API Modifica e Ordinamento

3.3 Modifiche Database

Di seguito è riportato il modello dei dati aggiornato.


PostgreSQL Database ERD
scalata_classi
scalata_id FK varchar
class_ut_id FK varchar
"scalata_id -> scalataname"
scalata
scalataname PK varchar
number_of_rounds int
scalata_description varchar
username varchar
"class_ut_id -> name"
interactions
id PK int
user_email varchar
class_name FK varchar
interaction_type int
description text
created_at timestamp
updated_at timestamp
deleted_at timestamp
user_id bigint
"class_name -> name"
class_ut
name PK varchar
category jsonb
created_at timestamp
date date
deleted_at timestamp
description text
difficulty varchar
updated_at timestamp
uri varchar
"class_ut_name -> name"
achievements
id PK
description
name
progress_required
statistic
opponent
id PK bigint
coverage varchar
created_at timestamp
evosuite_score jsonb
jacoco_score jsonb
opponent_difficulty varchar
opponent_type varchar
class_ut_name FK varchar
"class_ut_name -> name"
flyway_schema_history
installed_rank PK int
version varchar
description varchar
type varchar
script varchar
checksum int
installed_by varchar
installed_on timestamp
execution_time int
success boolean
hint
id PK bigint
content text
created_at timestamp
deleted_at timestamp
image_uri varchar
order int
type varchar
updated_at timestamp
admin_email FK varchar
class_ut_name FK varchar
name varchar
"admin_email -> email"
admin
email PK varchar
created_at timestamp
deleted_at timestamp
invitation_token varchar
name varchar
password varchar
reset_token varchar
surname varchar
updated_at timestamp
username UK varchar
operation
id PK int
date varchar
type int
email FK varchar
class_name varchar
"email -> email"
"admin_email -> email"
team_admin
id PK varchar
is_active boolean
role varchar
admin_email FK varchar
team_id FK varchar
"team_id -> id"
team
id PK varchar
creation_date timestamp
id_studenti jsonb
name varchar
numero_studenti int
assignment
id PK varchar
data_creazione timestamp
data_scadenza timestamp
descrizione varchar
titolo varchar
team_id FK varchar
"team_id -> id"

3.4 Issue Corrette

Una issue individuata è la seguente. Il microservizio che si occupa di autenticazione e gestione delle utenze è T23. Quando ci si registra come nuovo amministratore, si riesce ad accedere alla dashboard e a gestire classi e suggerimenti. Però, in fase di creazione dei suggerimenti, il microservizio T1 andava in errore perchè non trovava l'utenza dell'amministratore nella tabella *admin*. Questo ci ha fatto supporre, che non esista un flusso che preveda il corretto caricamento dell'utenza sul database del microservizio T1.

4 Implementazione

4.1 Tabella Moduli Aggiunti e Modificati

Service	Tipo di Modifica
T1	Migrazione **database** da MongoDB a PostgreSQL
T1	Modifica database per aggiungere la **tabella** *hints* dei suggerimenti
T1	Tutte le entity sono state spostate sotto il **package** `/model/entity`
T1	Tutte le **entity** preesistenti sono state modificate per essere adattate al database sottostante
T1	Tutti i **repository** sono stati modificati per estendere l'interfaccia `JpaRepository`
T1	Tutti i **service** preesistenti sono stati modificati e trasformati in interfacce
T1	Per ciascuna interfaccia service è stata creata una **classe di implementazione** (`Impl`), all'interno del package `/service/implementation`
T1	Tutti i **controller** preesistenti sono stati modificati per riadattarli a entity e service ridefiniti
T1	È stata creata la nuova **entity** `HintEntity`
T1	È stato creato il nuovo **repository** `HintRepository`
T1	Sono stati creati i **DTO** `Hint`, `HintResponse` e `HintUpdateDto`
T1	È stato creato un **enum** `HintTypeEnum`
T1	È stato aggiunto il **package** `mapper`
T1	È stato creato il **mapper** `HintMapper` per convertire entity in DTO e viceversa
T1	È stata creata l'**interfaccia** `HintService`
T1	È stata creata la **classe** `HintServiceImpl` che implementa l'interfaccia `HintService`
T1	È stato creato il **controller** `HintController`
T1	È stata creata la **classe di utility** `HintSpecification` per filtrare i dati sul database
T1	È stata creata la **classe di configurazione** `JpaConfig` per la configurazione del database Postgres
T1	È stata creata la **classe di configurazione** `WebConfig`
T1	È stata creata la **classe di configurazione** `WebSecurityConfig`
T1	È stata creata la **classe di configurazione** `FlywayConfig` per la configurazione di Flyway
T1	È stata creata la **view** `HintViewController`
T1	Sono stati modificati i file `dashboard.html`, `dashboard.js` per aggiungere la card relativa ai suggerimenti
T1	È stata aggiunta la directory `resources/static/t1/js/hint` per raccogliere i file javascript relativi ai suggerimenti
T1	Sono stati creati i file `hint_main.js`, `hint_upload.js` per gestire lato frontend le logiche di visualizzazione, inserimento ed eliminazione

Service	Tipo di Modifica
T1	È stata aggiunta la directory `resources/templates/hint` per raccogliere i file html relativi ai suggerimenti
T1	Sono stati creati i file `hint_main.html`, `hint_upload.html`
T1	Sono stati modificati i file `message.properties`, `message_en.properties` e `message_it.properties` per i messaggi internazionalizzati
T1	È stato modificato il file `application.properties` per aggiungere le properties relative al nuovo database
T1	È stato aggiornato il file `.env` con le nuove variabili d'ambiente
T1	È stato aggiornato il file `pom.xml` per aggiungere le dipendenze relative a Postgres, MapStruct, Flyway

Table 9: Elenco delle principali modifiche ai microservizi

5 Test e Problematiche

N.	Nome	Precondizioni	Test	Postcondizioni	Esito atteso	Esito reale	Pass/Fail	Problema	Soluzione
1	Creazione Suggerimenti tramite file JSON	Formato file JSON. Contenuto file corretto.	**HintsWithImages.json**	Suggerimenti caricati sul database.	Suggerimenti caricati con successo.	**Suggerimenti caricati con successo!**	PASS		
2	Caricamento file non JSON	Formato file png	**FileNonJson.txt**	Suggerimenti non caricati sul database.	Formato file non valido. Assicurati che sia un JSON valido.	**Formato file non valido. Assicurati che sia un JSON valido.**	PASS		
3	Caricamento file con contenuto errato.	Formato file JSON. Contenuto file errato.	**StructureWrong.json**	Suggerimenti non caricati sul database.	Il contenuto del file non è valido.	ECCEZIONE	FAIL	Hint presenta le annotazioni @NotNull per la verifica sui campi. Però non viene considerata automaticamente da Jackson (la libreria usata da Spring Boot per objectMapper.readValue().	è stato aggiunto un controllo esplicito sui suggerimenti presenti nel file per validarne il contenuto.
4	Caricamento file senza immagine allegata.	Formato file JSON. Immagine mancante.	**HintsWithImages.json**	Suggerimenti non caricati sul database.	URI dell'immagine trovato nel JSON, ma file non allegato.	ECCEZIONE	FAIL	Mancava il throw dell'eccezione.	Eccezione gestita.
5	Caricamento dello stesso file due volte.	Formato file JSON. File già caricato in precedenza.	**HintsWithImages.json**	Suggerimenti non caricati sul database.	Suggerimento già esistente (contenuto duplicato).	**Suggerimento già esistente (contenuto duplicato).**	PASS		
6	Caricamento file vuoto	Formato file JSON. File vuoto.	**Empty.json**	Suggerimenti non caricati sul database.	Il file non contiene suggerimenti validi da caricare.	**Il file non contiene suggerimenti validi da caricare.**	PASS		
7	Caricamento file con tutti i campi con valore una stringa vuota	Formato file JSON.	**HintWithAllFieldButEmpty.json**	Suggerimenti non caricati sul database.	Il file JSON è malformato o contiene errori di sintassi.	**Il file JSON è malformato o contiene errori di sintassi.**	PASS		
8	Caricamento file con immagini allegate come file di testo.	Formato file JSON.	**HintsFTPFile.json image1.txt image2.txt**	Suggerimenti non caricati sul database.	Il file 'image2.txt' non è un'immagine valida. Sono ammessi solo .png, .jpg, .jpeg.	**Il file 'image2.txt' non è un'immagine valida. Sono ammessi solo .png, .jpg, .jpeg.**	PASS		
9	Caricamento file suggerimenti associati ad una classe UT non esistente.	Formato file JSON. Classe UT non esistente.	**ClassNotFound.json**	Suggerimenti non caricati sul database.	Classe UT non trovata.	**Classe UT non trovata.**	PASS		

Figure 1: Test Create

N.	Nome	Precondizioni	Test	Postcondizioni	Esito atteso	Esito reale	Pass/Fail
1	Navigazione su http://localhost/hints/main# generic non avendo caricato alcun suggerimento	Non si è caricato alcun suggerimento generico		Suggerimenti generici non visualizzati	Nessun suggerimento generico trovato.	**Nessun suggerimento generico trovato.**	PASS
2	Navigazione su http://localhost/hints/main# classes non avendone caricato alcuno.	Non si è caricato alcun suggerimento di classe		Suggerimenti di classi non visualizzati	Nessun suggerimento di classe trovato.	**Nessun suggerimento di classe trovato.**	PASS
3	Navigazione su http://localhost/hints/main# generic cercando il suggerimento generico che si è caricato.	Si è caricato un suggerimento generico	**hint-template.json**	Suggerimento generico caricato visualizzato	Suggerimento generico caricato visualizzato	**Suggerimento generico caricato visualizzato**	PASS
4	Navigazione su http://localhost/hints/main# generic cercando il suggerimento di classe che si è caricato.	Si è caricato un suggerimento di classe	**hint-template.json**	Suggerimento di classe caricato visualizzato	Suggerimento di classe caricato visualizzato	**Suggerimento di classe caricato visualizzato**	PASS
5	Navigazione su http://localhost/hints/main# generic inserendo nella barra di ricerca il nome di un suggerimento generico non avendone caricato alcuno.	Non si è caricato alcun suggerimento generico		Suggerimento generico non visualizzato	Nessun suggerimento di classe trovato.	**Nessun suggerimento di classe trovato.**	PASS
6	Navigazione su http://localhost/hints/main# classes inserendo nella barra di ricerca il nome di un suggerimento di classe non avendone caricato alcuno.	Non si è caricato alcun suggerimento di classe		Suggerimento di classe non visualizzato	Nessun suggerimento di classe trovato.	**Nessun suggerimento di classe trovato.**	PASS
7	Navigazione su http://localhost/hints/main# generic avendo caricato un suggerimento di classe	Si è caricato un suggerimento generico	**hint-template.json**	Suggerimenti generici caricati visualizzati	Visualizzazione della griglia con i suggerimenti generici caricati	**Visualizzazione della griglia con i suggerimenti generici caricati**	PASS
8	Navigazione su http://localhost/hints/main# classes avendo caricato un suggerimento generico	Si è caricato un suggerimento di classe avendo prima caricato la classe a cui associarlo.	**hint-template.json**	Suggerimenti di classe caricati visualizzati	Visualizzazione della griglia con i suggerimenti di classe caricati	**Visualizzazione della griglia con i suggerimenti di classe caricati**	PASS
9	Navigazione su http://localhost/hints/main# detail per visualizzare il dettaglio di un suggerimento generico	Si è caricato un suggerimento generico	**hint-template.json**	Dettagli dei suggerimenti generici caricati visualizzati	Dettagli dei suggerimenti generici caricati visualizzati con la possibilità di modificare o eliminare il suggerimento generico	**Dettagli dei suggerimenti generici caricati visualizzati con la possibilità di modificare o eliminare il suggerimento generico**	PASS
10	Navigazione su http://localhost/hints/main# detail per visualizzare il dettaglio di un suggerimento di classe	Si è caricato un suggerimento di classe avendo prima caricato la classe a cui associarlo.	**hint-template.json**	Dettagli dei suggerimenti di classe caricati visualizzati	Dettagli dei suggerimenti generici caricati visualizzati con la possibilità di modificare o eliminare il suggerimento di classe	**Dettagli dei suggerimenti generici caricati visualizzati con la possibilità di modificare o eliminare il suggerimento generico**	PASS

Figure 2: Test Get

Numero	Nome	Precondizioni	Test	Postcondizioni	Esito reale	Esito atteso	Pass/Fail
1	Modifica di un suggerimento generico	Si è caricato il suggerimento generico da modificare	**hint-template.json**	Il suggerimento generico viene visualizzato con i campi modificati	Il suggerimento generico viene visualizzato con i campi modificati	**Il suggerimento generico viene visualizzato con i campi modificati**	PASS
2	Modifica di un suggerimento di classe	Si è caricata la classe e il relativo suggerimento da modificare	**hint-template.json**	Il suggerimento di classe viene visualizzato con i campi modificati	Il suggerimento di classe viene visualizzato con i campi modificati	**Il suggerimento di classe viene visualizzato con i campi modificati**	PASS

Figure 3: Test Update

Numero	Nome	Precondizioni	Test	Postcondizioni	Esito atteso	Esito reale	Pass/Fail
1	Eliminazione di un suggerimento di classe	Si è caricato un suggerimento di classe avendo prima caricato la classe a cui associarlo.	**hint-template.json**	Il suggerimento non viene più visualizzato	Viene chiesto la conferma di voler eliminare il suggerimento generico, dopodiché si viene riportati alla schermata dei suggerimenti di classe dove questo non viene più mostrato.	**Viene chiesto la conferma di voler eliminare il suggerimento generico, dopodiché si viene riportati alla schermata dei suggerimenti di classe dove questo non viene più mostrato.**	PASS
2	Eliminazione di tutti suggerimenti generici	Si è caricato almeno un suggerimento generico	**hint-template.json**	Non viene visualizzato più alcun suggerimento generico	Non viene visualizzato più alcun suggerimento generico	**Non viene visualizzato più alcun suggerimento generico**	PASS
3	Eliminazione di tutti suggerimenti di classe	Si è caricato almeno un suggerimento di classe avendo prima caricato le classi a cui associarli	**hint-template.json**	Non viene visualizzato più alcun suggerimento di classe	Non viene visualizzato più alcun suggerimento di classe	**Non viene visualizzato più alcun suggerimento di classe**	PASS

Figure 4: Test Delete

6 Sviluppi Futuri

Di seguito sono riportate alcune idee di sviluppi futuri da fare:

- Aggiungere paginazione alla GET dei suggerimenti.
- Sviluppare anche una create manuale dei suggerimenti.

- Sul database Postgres su ogni tabella sono stati predisposti i campi createdAt, updatedAt, deletedAt per tenere traccia di ogni operazione fatta.
- Aggiungere dei mapper per mappare tutte le entity sui DTO senza dover assegnare a mano i campi con i setter.
- Gestire opportunamente le operazioni transazionali avendo ora un database SQL.
- Dare la possibilità di caricare anche file diversi dal JSON.