UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
Corso di Laurea Magistrale in Ingegneria Informatica

REPOSITORY
CLASSI UNDER TEST

Relazione Finale

Software Architecture Design

Task: 1 – Gruppo: 20

Candidati

Luca Cardellicchio – M63001511

Roberto Sannino – M63001497

Docente
Prof.ssa Anna Rita Fasolino

ANNO ACCADEMICO 2022-2023

INDICE

1 Descrizione del Task

Per facilitare la stesura del documento per quanto concerne la progettazione del componente, i requisiti e le storie utente sono divise in due classi:

- *Requisiti specifici del task;*
- *Requisiti aggiuntivi*, inseriti per permettere la gestione degli amministratori.

1.1 Requisiti specifici del task

L'applicazione deve mantenere un insieme di Classi Java da testare e deve offrire la possibilità ai giocatori di consultare l'elenco delle classi disponibili e di fare il download del codice di una di esse. L'applicazione deve permettere ad un amministratore anche di aggiornare l'insieme di classi disponibili mediante aggiunta di classi e relativo salvataggio del file di codice. Sarebbe auspicabile anche prevedere funzioni per la ricerca di classi in base a specifici requisiti, come ad esempio la complessità della classe, o altri attributi.

1.2 Requisiti aggiuntivi

1.2.1 Registrazione degli amministratori

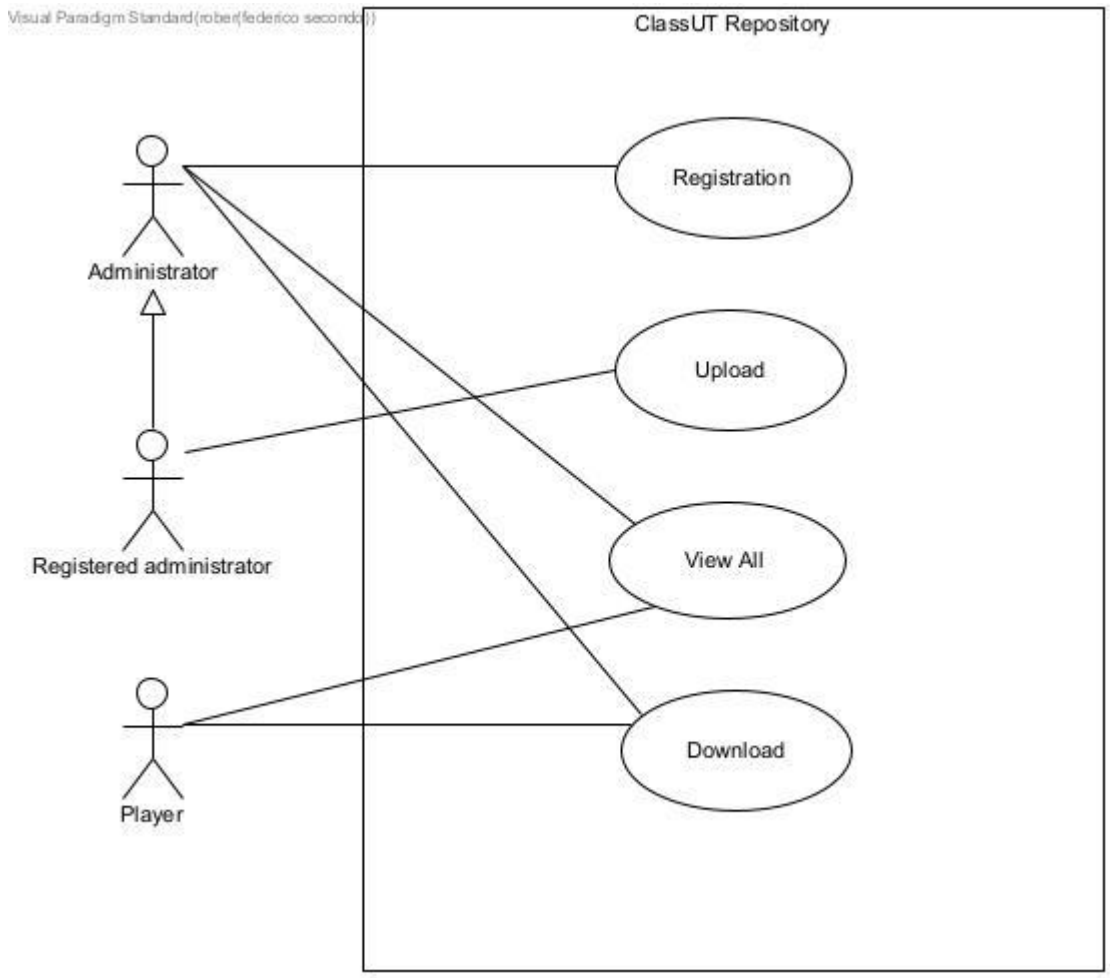
L'applicazione deve consentire agli amministratori di registrarsi fornendo nome, cognome, un indirizzo e-mail valido ed una password.

1.2.2 Autenticazione degli amministratori

L'amministratore fornisce l'indirizzo e-mail specificato in fase di registrazione e la relativa password; dopo l'autenticazione è possibile caricare le Classi Java da testare.

2 Analisi

2.1 Use Case Diagram



2.2 Storie utente

ID	*Descrizione*	*Classe*
1	Come giocatore, voglio visualizzare l'elenco di tutte le classi disponibili.	Specifico
2	Come giocatore, voglio scaricare il codice di una classe.	Specifico
3	Come amministratore autenticato, voglio accedere alla pagina per caricare una nuova classe, in modo da aggiornare l'insieme delle classi disponibili.	Specifico
4	Come amministratore non registrato, voglio accedere alla pagina di registrazione.	Aggiuntivo
5	Come amministratore registrato, voglio accedere alla pagina di autenticazione.	Aggiuntivo

2.3 Scenari

2.3.1 Registration

Attore Primario	Administrator
Attore Secondario	-
Descrizione	Permette ad un amministratore di registrarsi per poter caricare una classUT
Pre-Condizioni	L'amministratore non deve essere già registrato
Sequenza di eventi principale	1. L'amministratore apre la home page 2. L'amministratore accede alla schermata di registrazione 3. Inserisce i dati richiesti e conferma
Post-Condizioni	L'amministratore può autenticarsi
Casi d'uso correlati	-
Sequenza di eventi alternativi	La registrazione fallisce se l'e-mail inserita è già stata usata

2.3.2 Upload

Attore Primario	Registered Administrator
Attore Secondario	-
Descrizione	Permette ad un amministratore di caricare una nuova classUT
Pre-Condizioni	L'amministratore deve autenticarsi
Sequenza di eventi principale	1. L'amministratore apre la schermata per l'upload 2. L'amministratore inserisce il file della classe UT e la sua complessità e conferma.
Post-Condizioni	La classeUT viene aggiunta al repository
Casi d'uso correlati	-
Sequenza di eventi alternativi	L'upload fallisce e viene mostrato un alert con il messaggio di errore

2.3.3 View All

Attore Primario	Administrator, Player
Attore Secondario	-
Descrizione	Permette di vedere tutte le classUT disponibili nel repository
Pre-Condizioni	-
Sequenza di eventi principale	1. L'attore apre la schermata per visualizzare le classUT
Post-Condizioni	Vengono visualizzate tutte le classUT contenute nel repository
Casi d'uso correlati	-
Sequenza di eventi alternativi	-

2.3.4 Download.

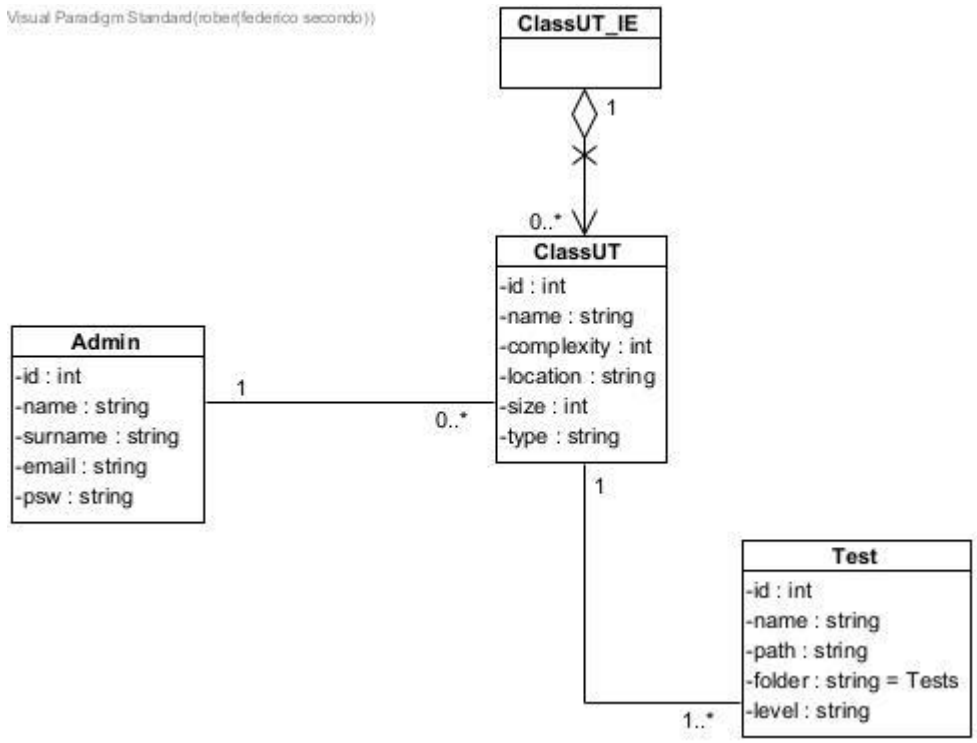
Attore Primario	Administrator, Player
Attore Secondario	-
Descrizione	Permette di scaricare il codice di una determinata classUT
Pre-Condizioni	Conoscere il nome della classe che si vuole scaricare
Sequenza di eventi principale	1. L'attore apre la schermata per fare il download 2. L'attore specifica il nome della classUT da scaricare
Post-Condizioni	Si ottiene il file .java della classe scelta
Casi d'uso correlati	-
Sequenza di eventi alternativi	Il download fallisce perché non è presente la classe specificata

2.4 Glossario dei termini

Termine	Descrizione
ClassUT	Classe Java da testare
Admin	Amministratore registrato
Test	Insieme di Test relativi ad una specifica classUT

2.5 Diagrammi di analisi

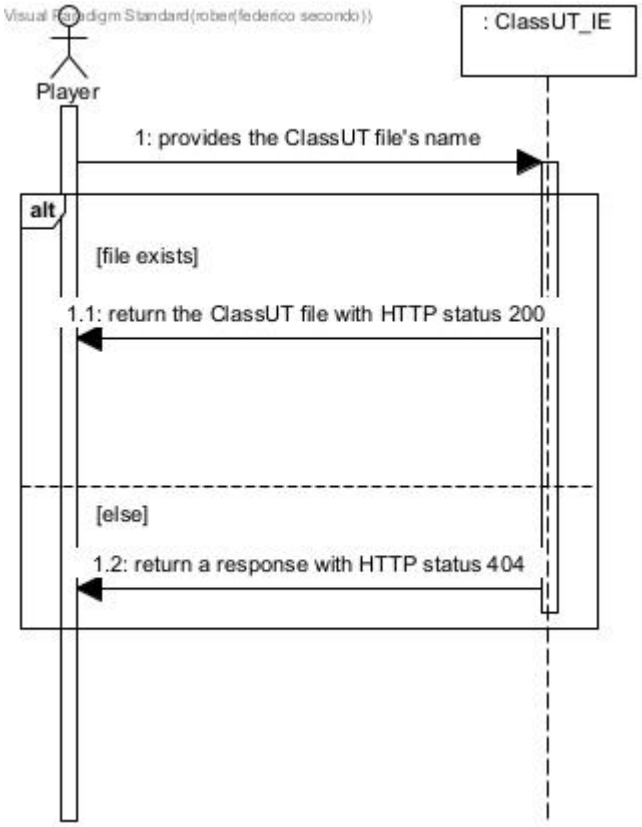
2.5.1 Class Diagram di analisi



La classe ClassUT_IE fornisce i servizi per gestire le operazioni CRUD delle classi under test.

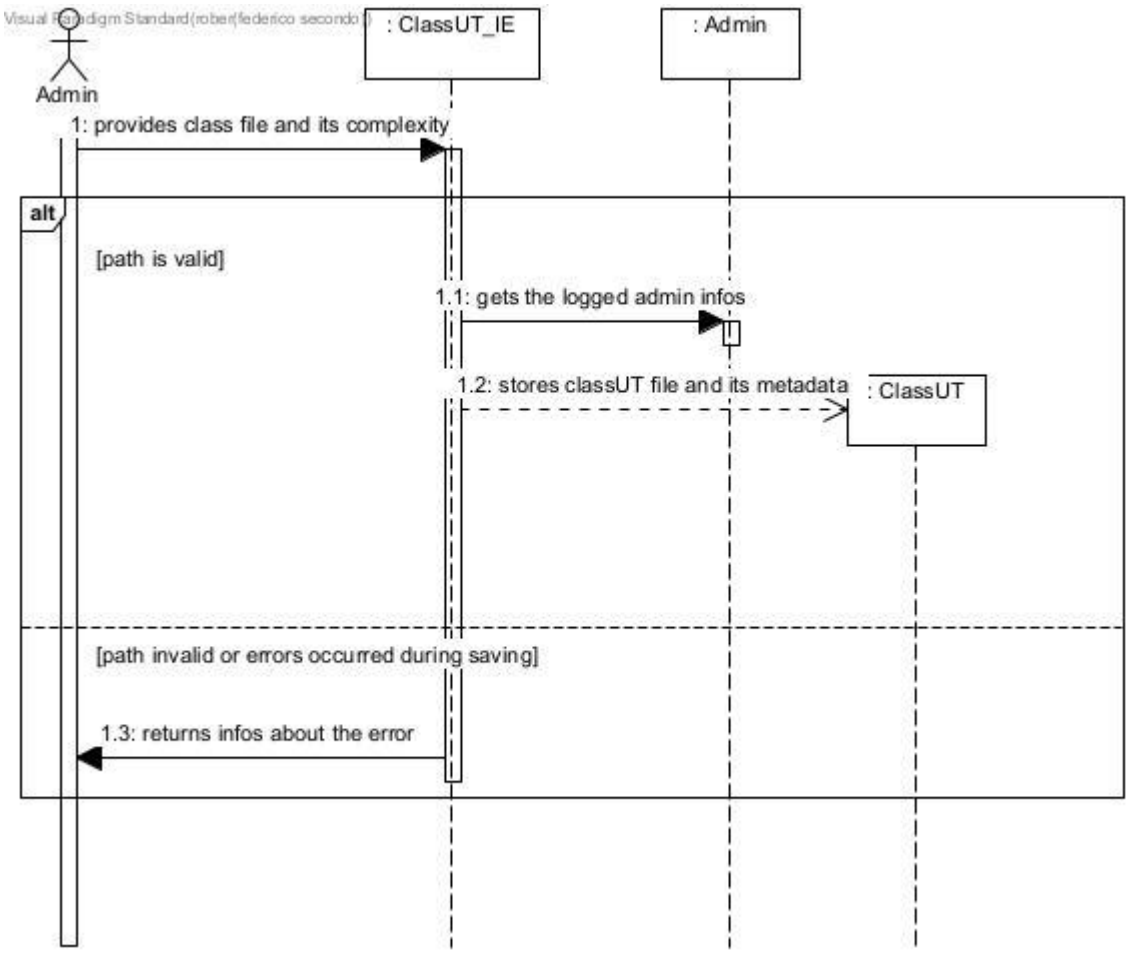
2.5.2 Sequence Diagram: Download del file di una Class Under Test

Sequence diagram relativo al caso d'uso Download.



2.5.3 Sequence Diagram: Upload di una Classe Under Test

Sequence diagram relativo al caso d'uso Upload.

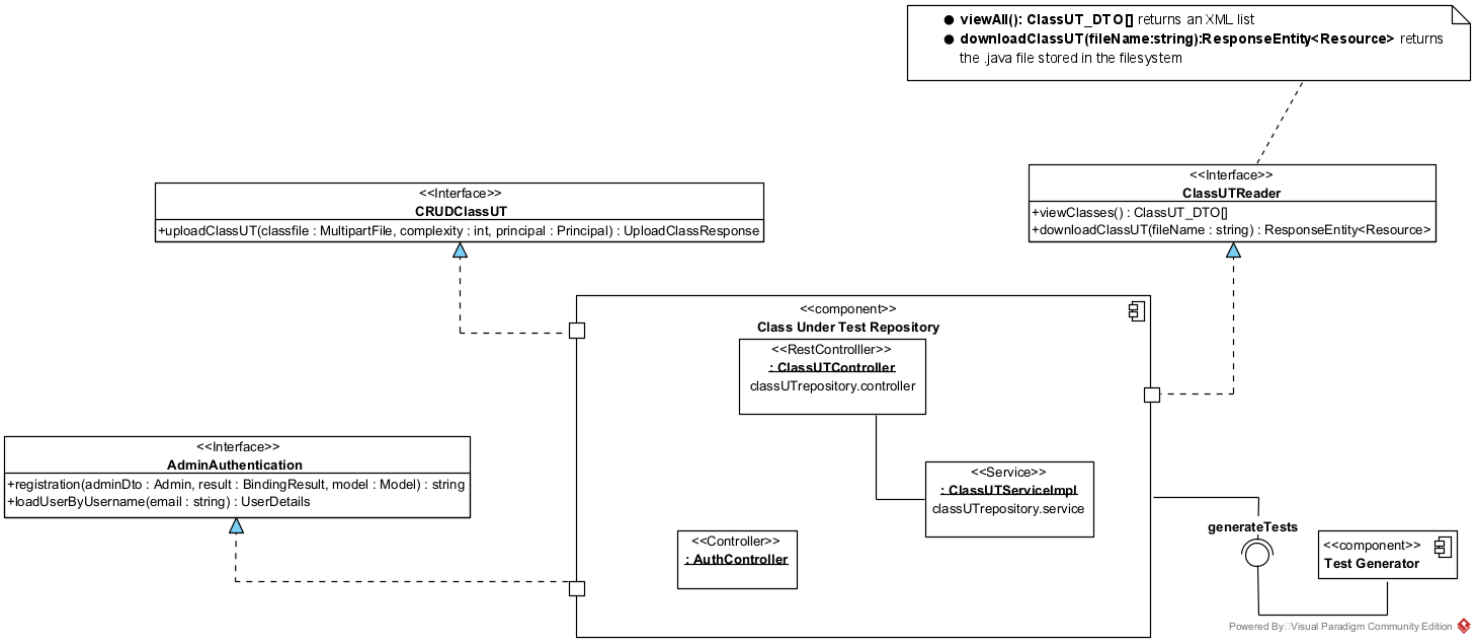


3 Progettazione

La tabella riportata di seguito evidenzia tutte le tecnologie utilizzate per l'implementazione del componente.

Tecnologia	***Motivazione***
Spring Data JPA	Integrato con *Hibernate*, fornisce supporto per l'ORM.
Spring Security	Supporto all'autenticazione degli amministratori ed alla protezione delle APIs realizzate
MySQL	Gestione della persistenza dei metadati inerenti alle *ClassUTs*, dei *Test* e degli *Admin*.
Lombok	Facilita la scrittura delle classi generando automaticamente *setter*, *getter* e *costruttori*.
Model Mapper	Fornisce dei metodi per effettuare il mapping tra una classe ed il *DTO* associato.
FasterXML	Fornisce supporto alla rappresentazione di oggetti Java in XML.
Maven	Supporto alla *build-automation*.
Thymeleaf	Supporto alla gestione delle pagine web

3.1 Component Diagram

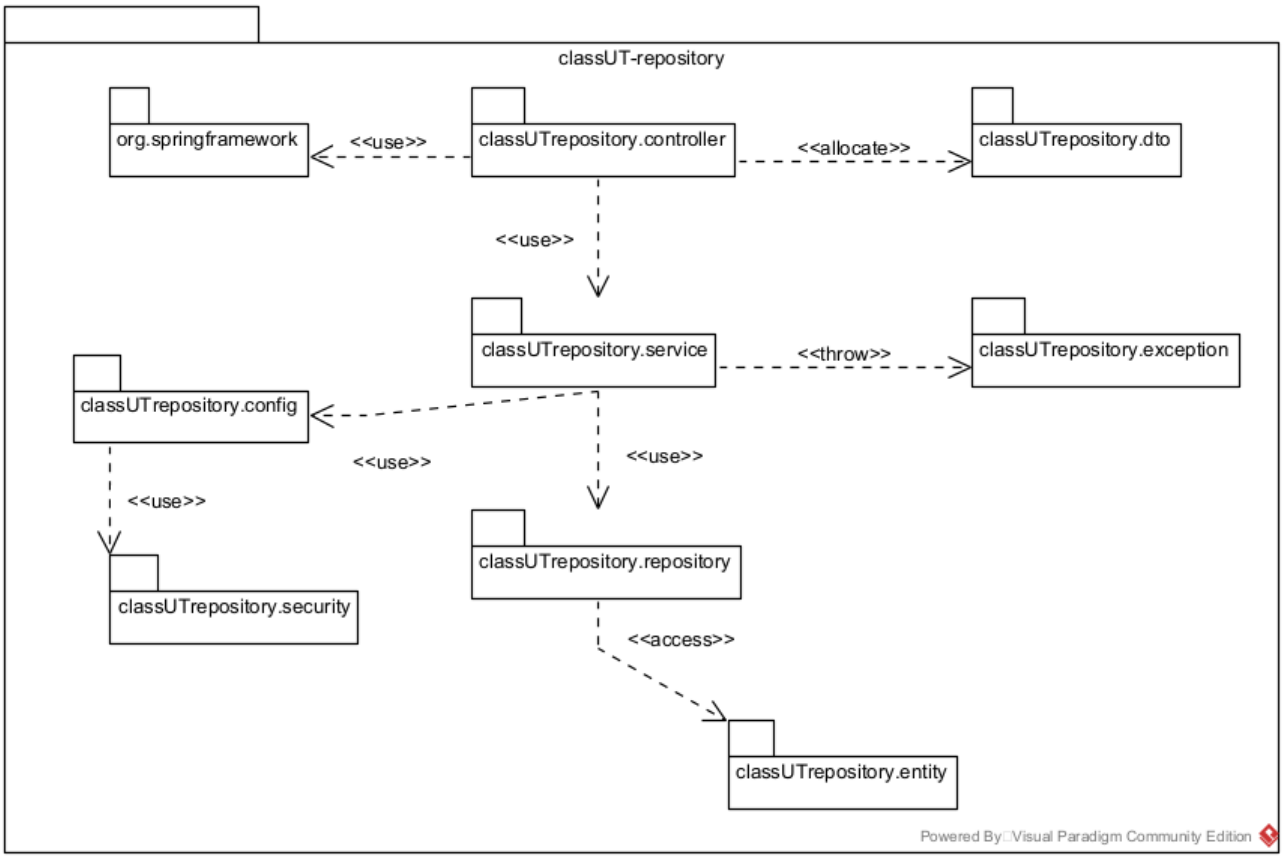


Ciascuna interfaccia abilita una o più storie utente evidenziate in fase di analisi. In particolare:

Interfaccia	***ID Storia Utente***
CRUD ClassUT	3
ClassUT Reader	1-2
Admin Authentication	4-5

3.2 Module Structures

Il componente è progettato secondo l'architettura a livelli riportata di seguito:



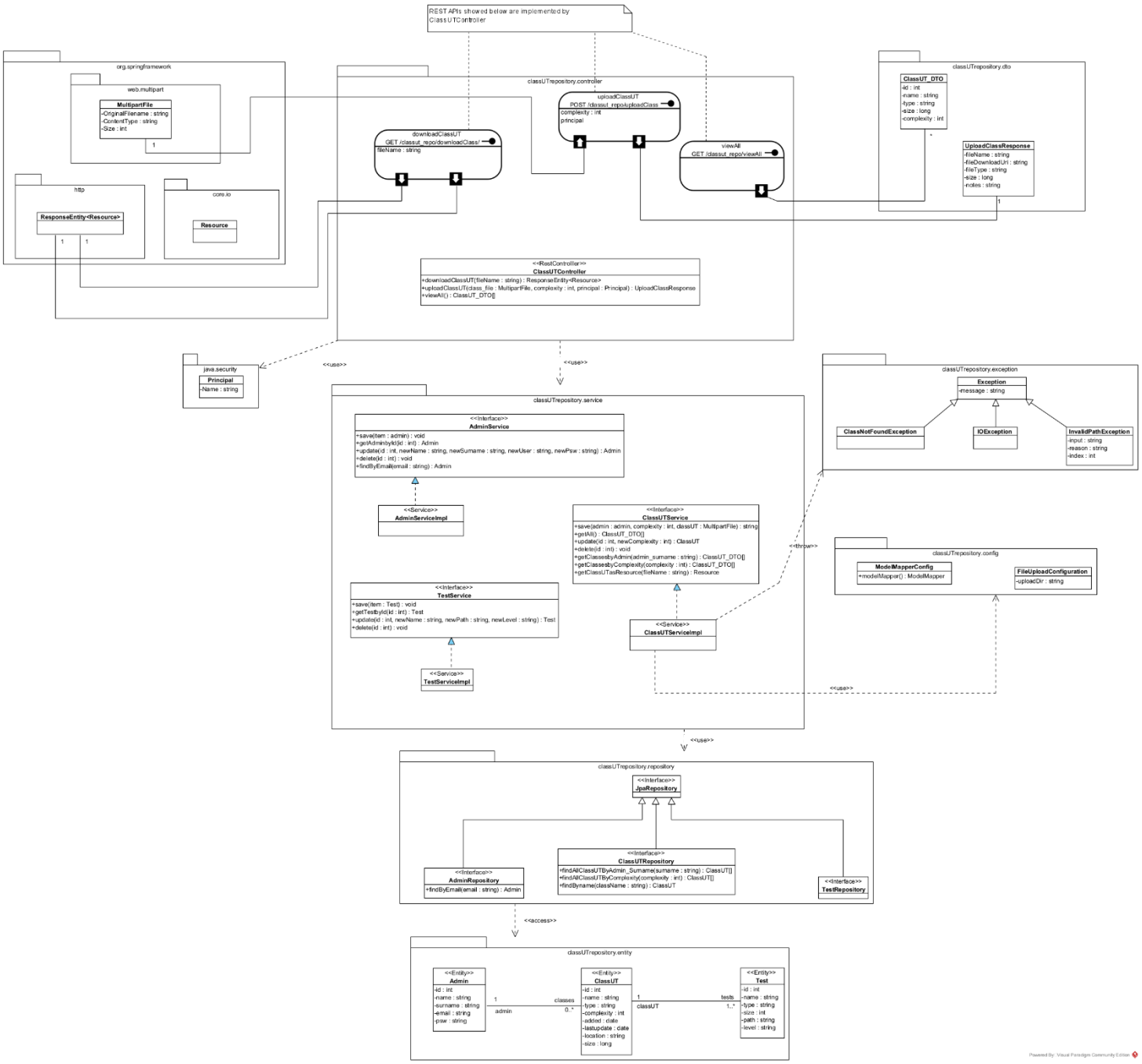
La motivazione principale di questa scelta è che si astrae il *controller*, e quindi la logica applicativa, dalle tecnologie utilizzate per persistere i dati.

Data la complessità in termini di dipendenze e numero di classi, l'architettura mostrata viene dettagliata a seconda della classe di appartenenza della storia utente in diverse viste (statiche e dinamiche), ciascuna delle quali esplicita gli elementi e le interazioni esistenti tra quest'ultimi nella realizzazione della stessa storia utente.

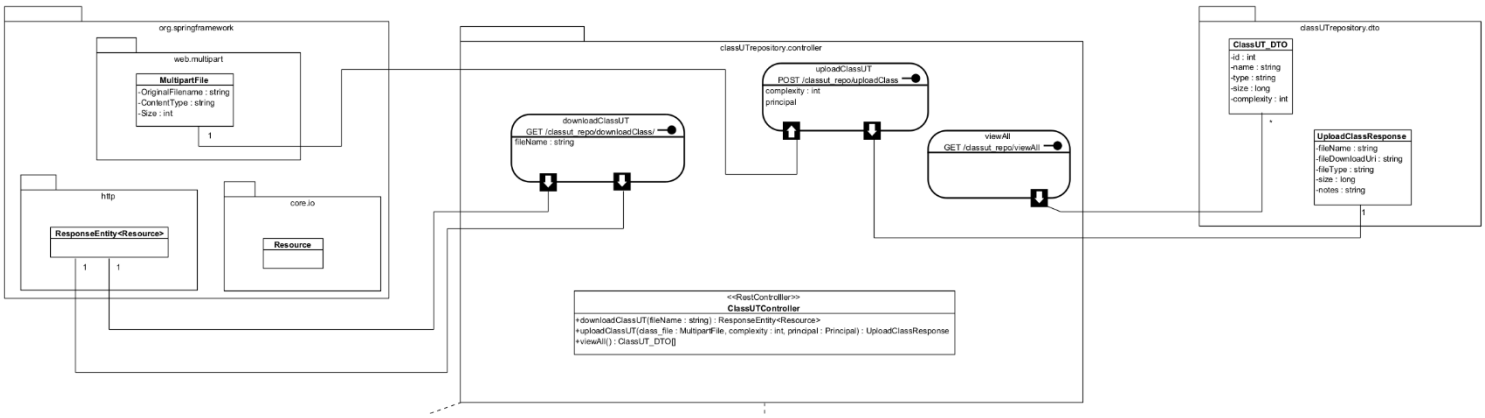
3.3 Requisiti specifici

In questa sezione è dettagliata l'implementazione delle storie utente da **1** a **3** riferite per ***ID***.

3.3.1 Class Diagram di dettaglio



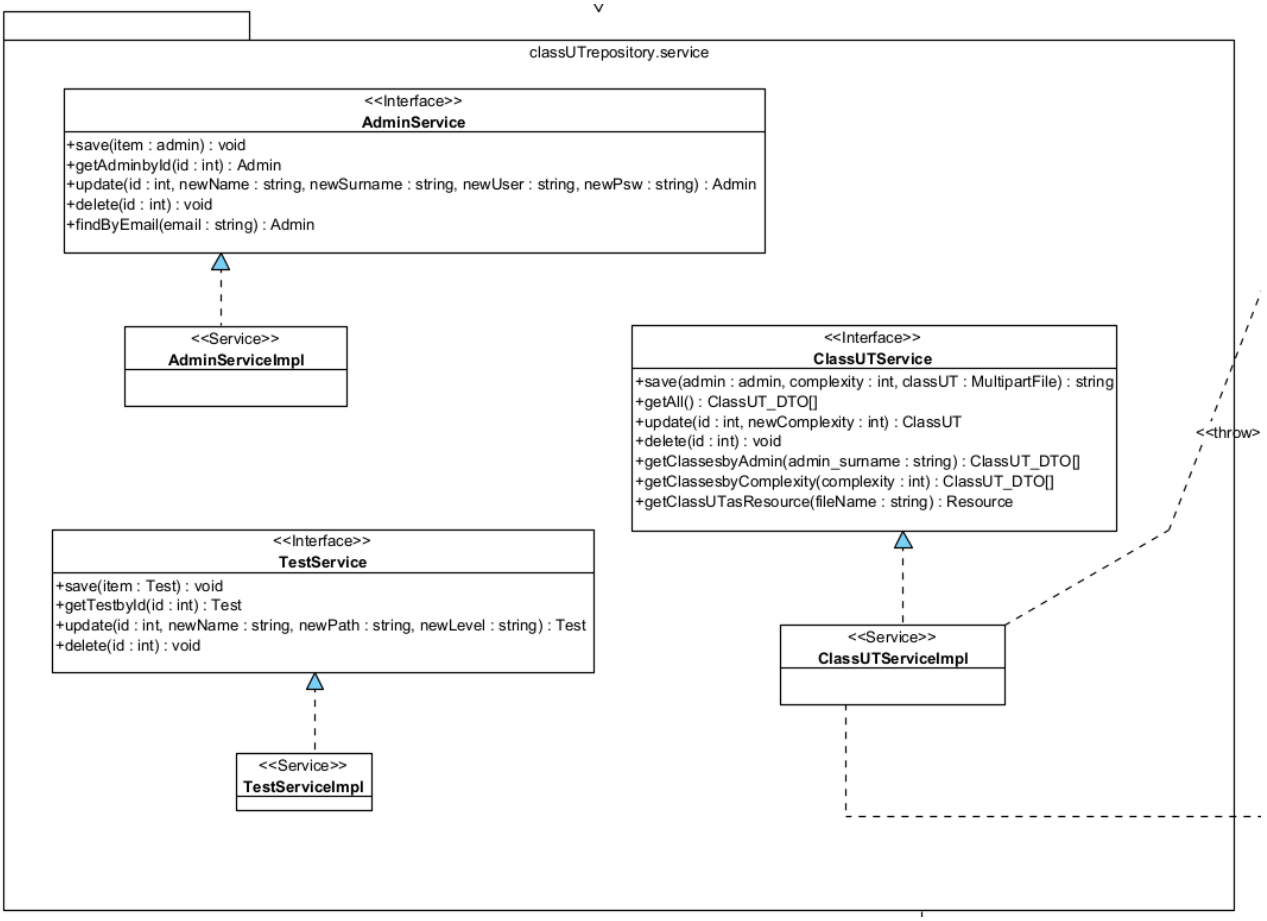
Il package *controller* include il **REST Controller** per l'interfacciamento con il repository delle classi Java.



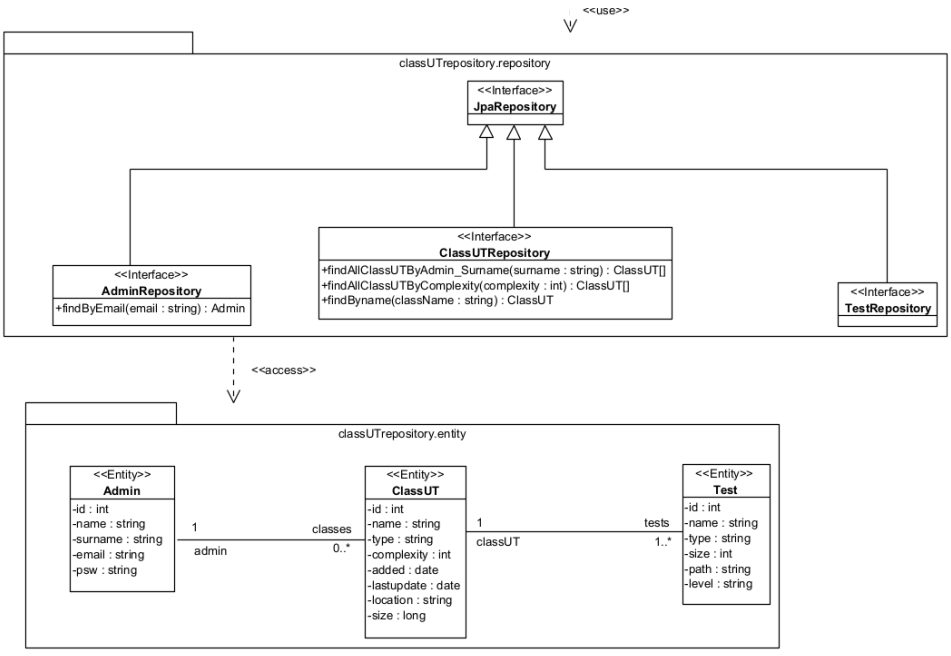
Le API fornite dal Controller sono:

Nome	***Parametri***	***Ritorno***
uploadClassUT	- Complessità della classe Java - File della classe Java - Admin loggato	Oggetto contenente le informazioni sull'esito
downloadClassUT	-Il nome della classe da scaricare	Il file .java della classe scelta
viewAll	-	La lista in formato XML di tutte le classi presenti nel repository

Il package *service* fornisce le API al package superiore, ovvero il *controller*, astraendo i meccanismi utilizzati per memorizzare metadati e file delle classi Java.



Si avvale delle API offerte dal package *repository*: Le interfacce presenti ereditano dall'interfaccia *JPA Repository* presente nel modulo Spring Data JPA citato in precedenza. Per utilizzare tale interfaccia bisogna specificare a quale classe stereotipata



come *Entity* si fa riferimento ed il tipo della chiave primaria utilizzata. Nello specifico, per gestire la persistenza dei metadati delle ClassiUT:

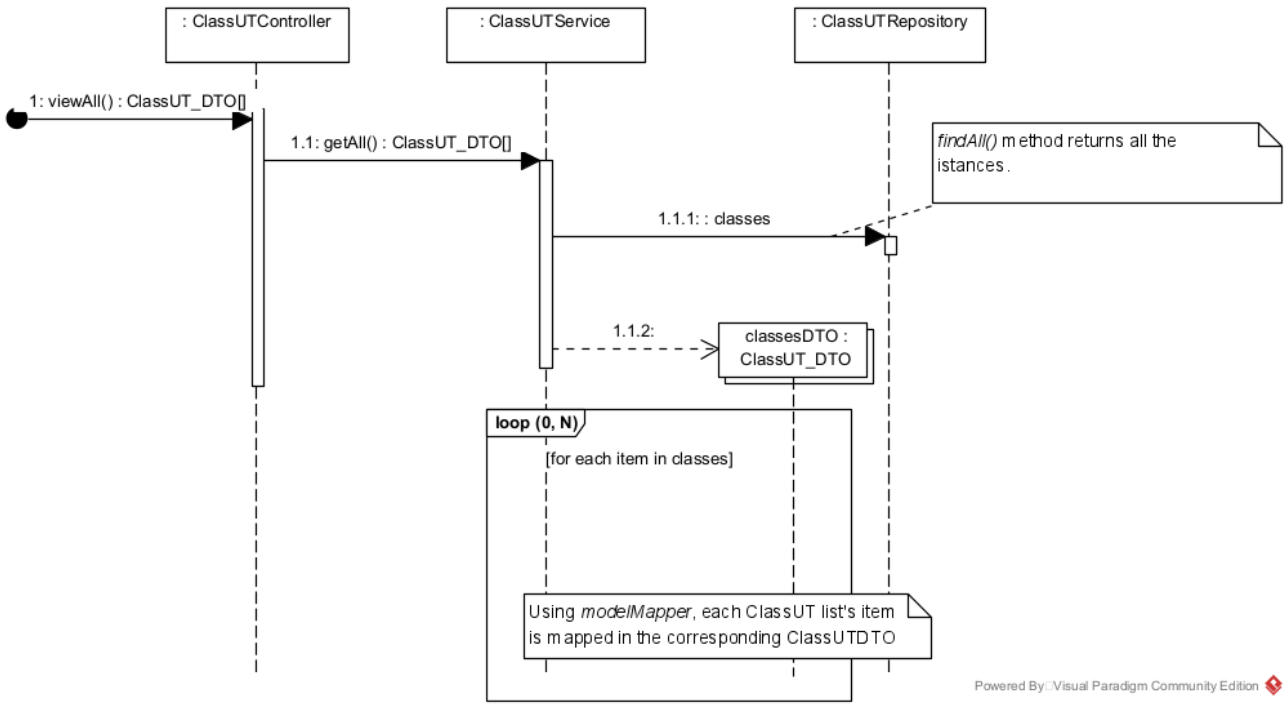
ClassUTRepository extends JpaRepository <ClassUT, Integer>

Il vantaggio di utilizzare tale interfaccia risiede nel fatto che non è necessario implementare metodi che gestiscano la persistenza dei dati. Inoltre, nel caso si vogliano definire dei metodi personalizzati è sufficiente utilizzare un template standard per la firma del metodo. Tale template prevede che:

- venga specificata l'operazione da effettuare, in questo caso *find*/*findAll*.
- venga specificato il campo su cui effettuare la selezione tramite la clausola *By*.

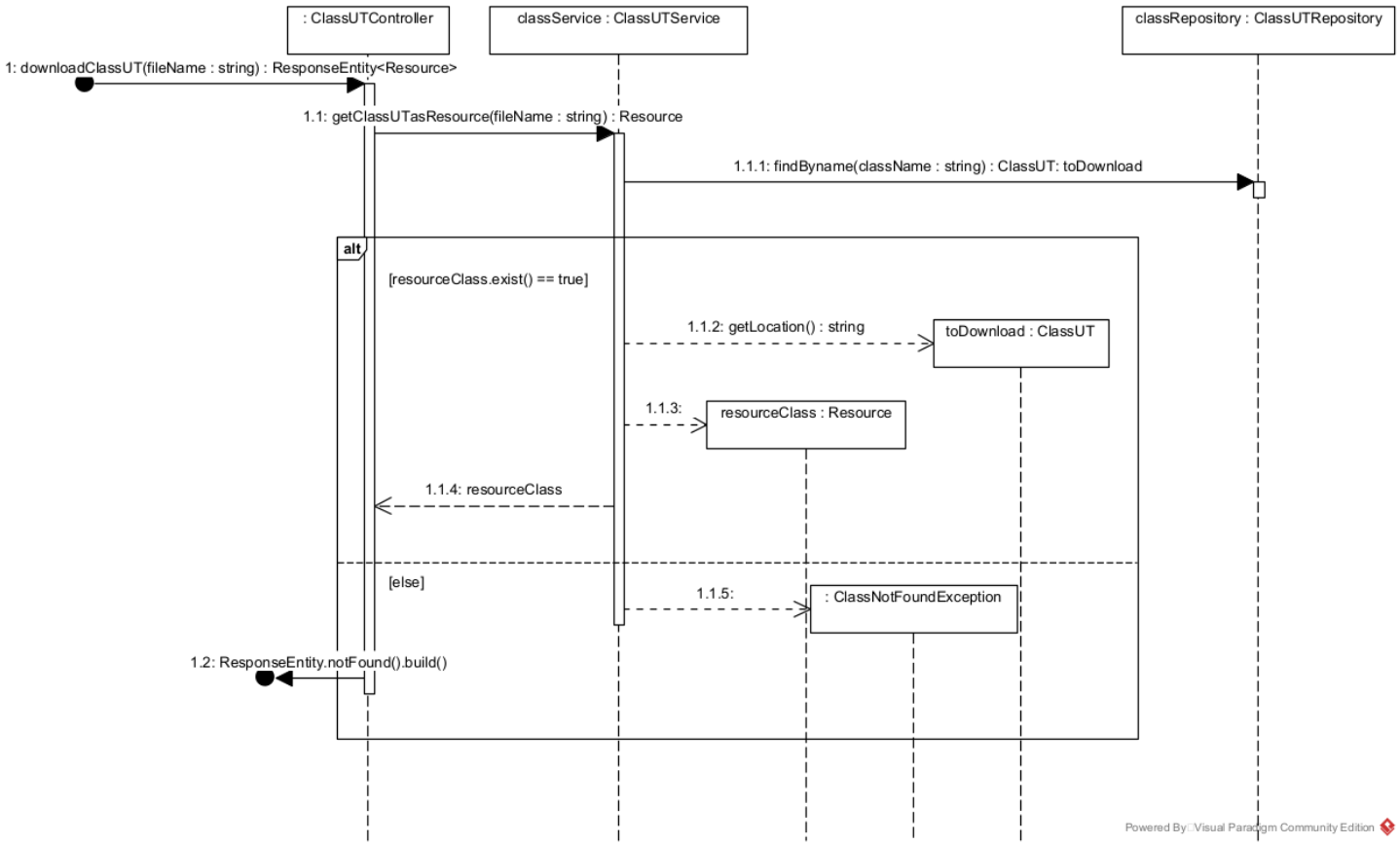
3.3.2 Sequence Diagram: Visualizzazione della lista delle Classi disponibili

ID Storia Utente	***1***



3.3.3 Sequence Diagram: Download del file di una Class Under Test

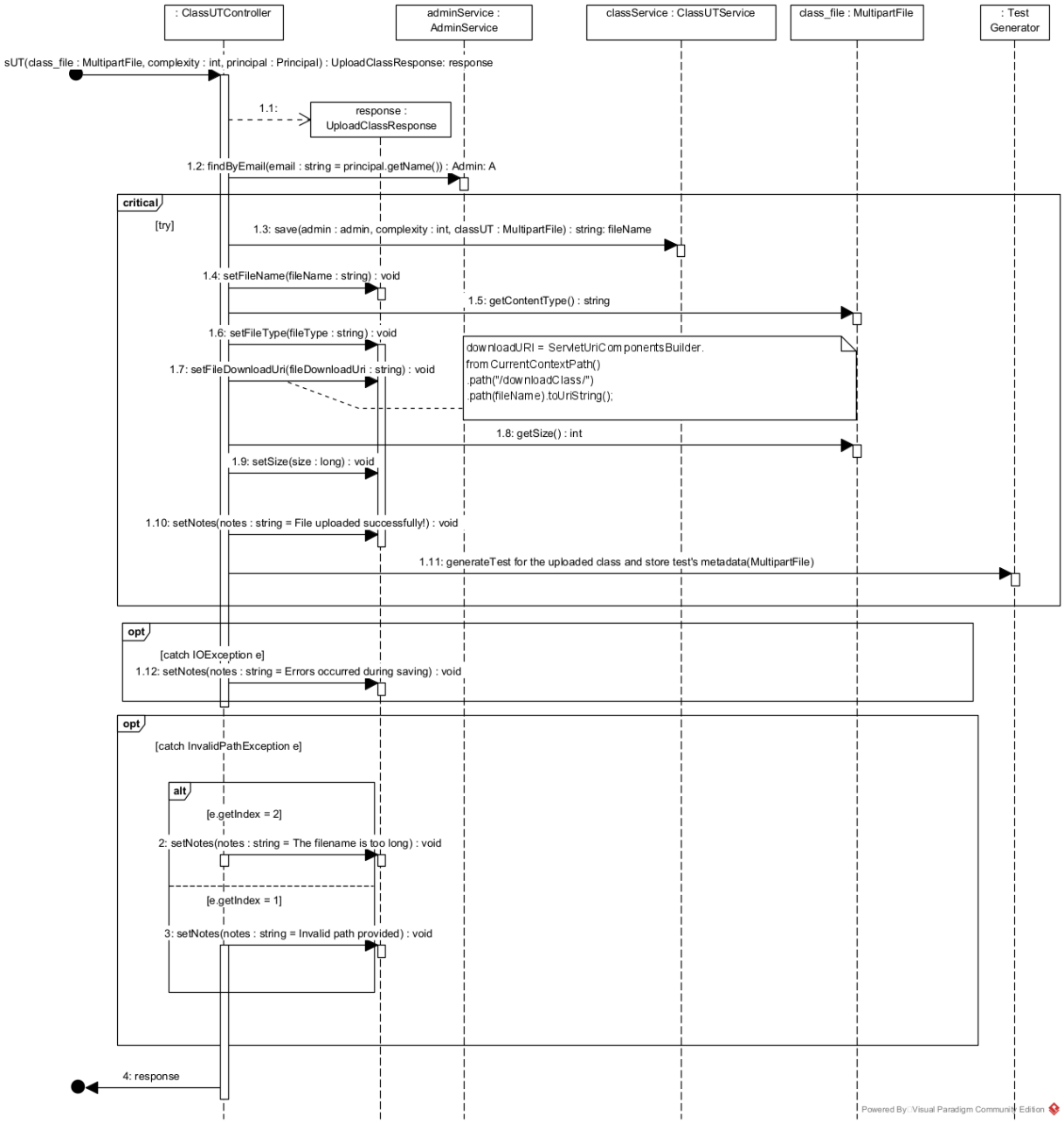
ID Storia Utente	*2*



Se la classe specificata esiste viene effettuato il download del relativo file, altrimenti viene ritornata una *ResponseEntity* con il codice http **404**.

3.3.4 Sequence Diagram: Upload di una Classe Under Test

ID Storia Utente	*3*



Tale funzionalità è disponibile soltanto per gli amministratori che effettuano il login. Infatti, tra i parametri di input del metodo è presente un oggetto *Principal* presente nel package *java.security*. Tale oggetto contiene le informazioni sull'utente loggato, in particolare l'e-mail di registrazione.

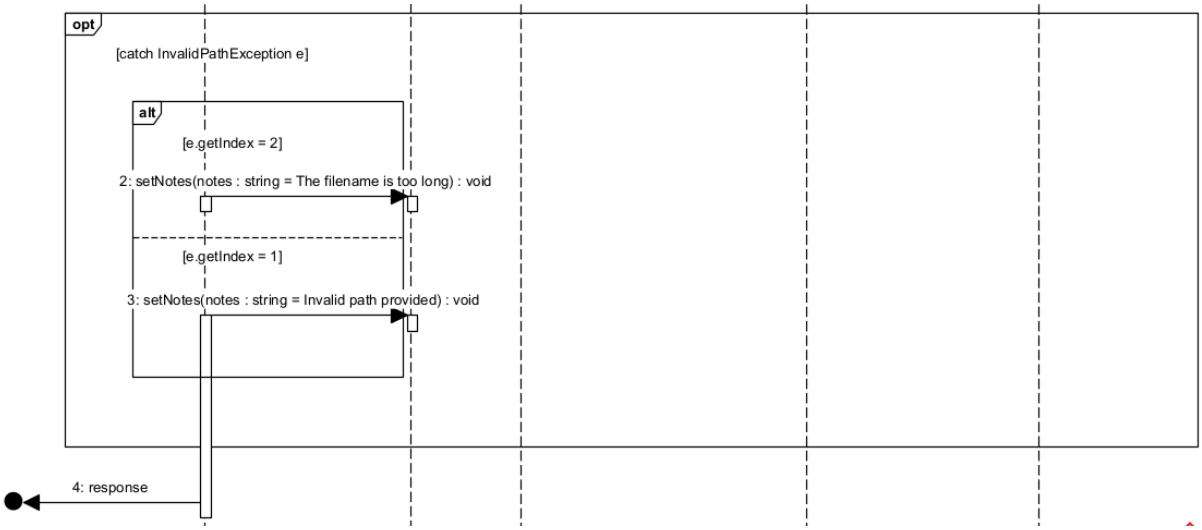
Lo scenario di successo prevede che il file ed i suoi metadati vengano memorizzati nel repository restituendo un oggetto *UploadClassResponse* riportante:

- Il nome con cui il file è stato salvato nel repository
- Il tipo di file caricato
- La dimensione del file
- L'URI per il download
- Le note inerenti all'esito del caricamento

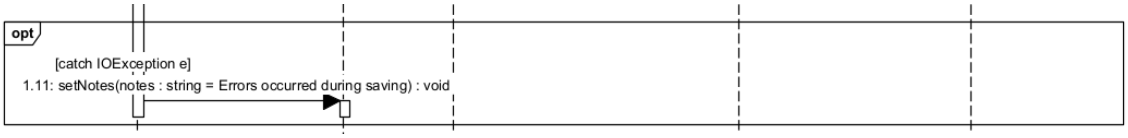
Durante il salvataggio si possono verificare due scenari di errore:

1. Errori dovuti al path del file:
    a. Il nome del file contiene caratteri non ammessi e quindi non è possibile salvare sul repository il file.
    b. Il nome del file non contiene caratteri ammissibili ma il path generato è troppo lungo per essere memorizzato nel database.

    In questo caso il *service* lancia un'eccezione di tipo *InvalidPathException* specificando l'indice: "1" nel primo caso e "2" nel secondo. Il *controller* la intercetta restituendo le informazioni sull'errore verificatosi in base all'indice dell'eccezione.



2. Errori durante il salvataggio sul filesystem:



3.3.5 Generazione del path della Classe Under Test

La regola di generazione del path del file appena caricato può essere schematizzata come segue:

root_directory+ClassUT_directory+ClassUT_filename

dove:

- *root_directory* è la cartella dove vengono memorizzati tutti i file .java delle Classi Under Test. Tale directory è specificata dalla proprietà

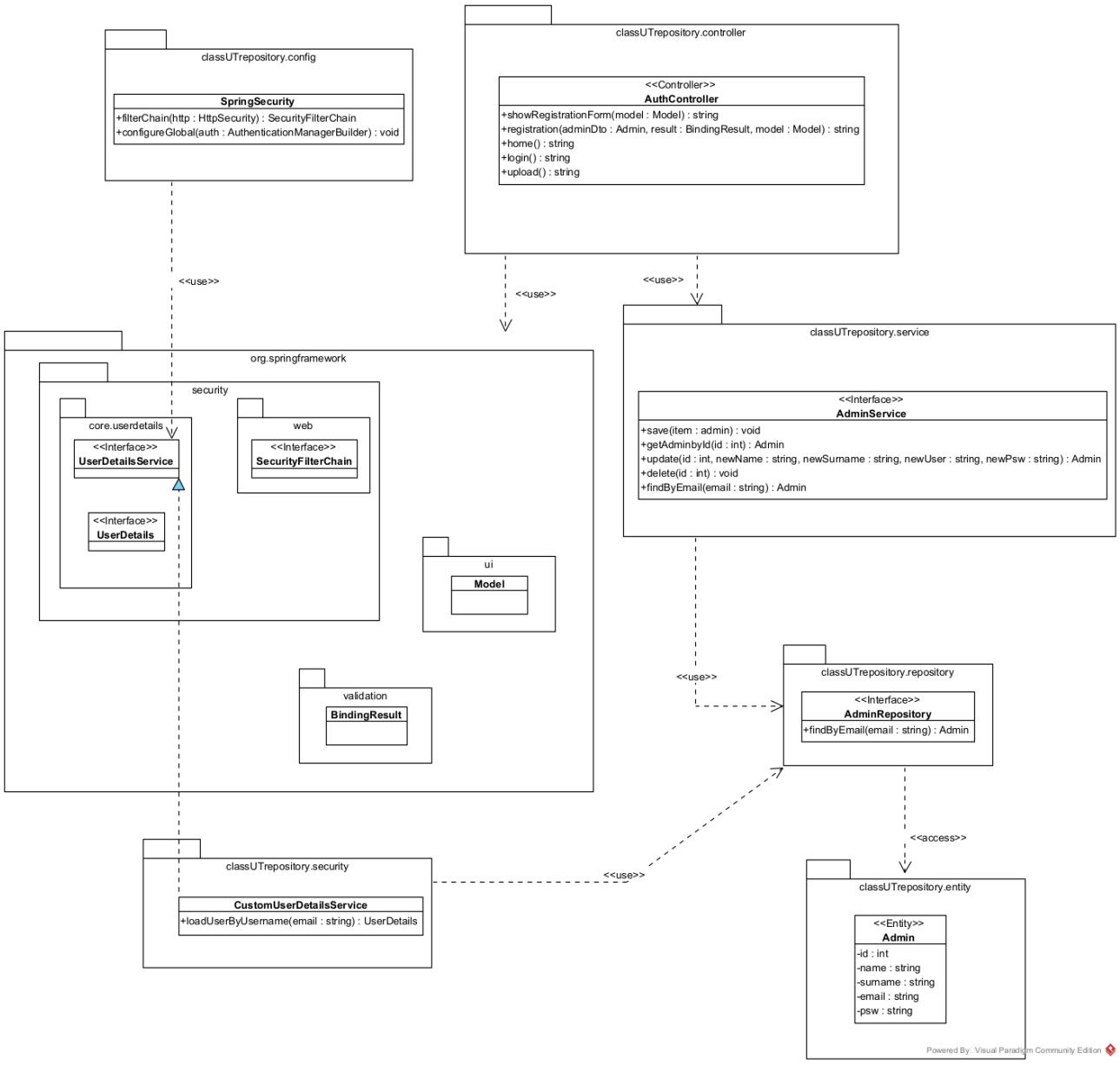
  file.upload-dir

  nel file ***application.properties*** del progetto o nel ***docker-compose.yml*** se il componente è eseguito in un ambiente Docker.
- *ClassUT_directory* si ottiene a partire dal nome file caricato ed eliminando l'estensione *.java*

3.4 Requisiti aggiuntivi

In questa sezione è dettagliata l'implementazione delle storie utente **4** e **5** riferite per ***ID***.

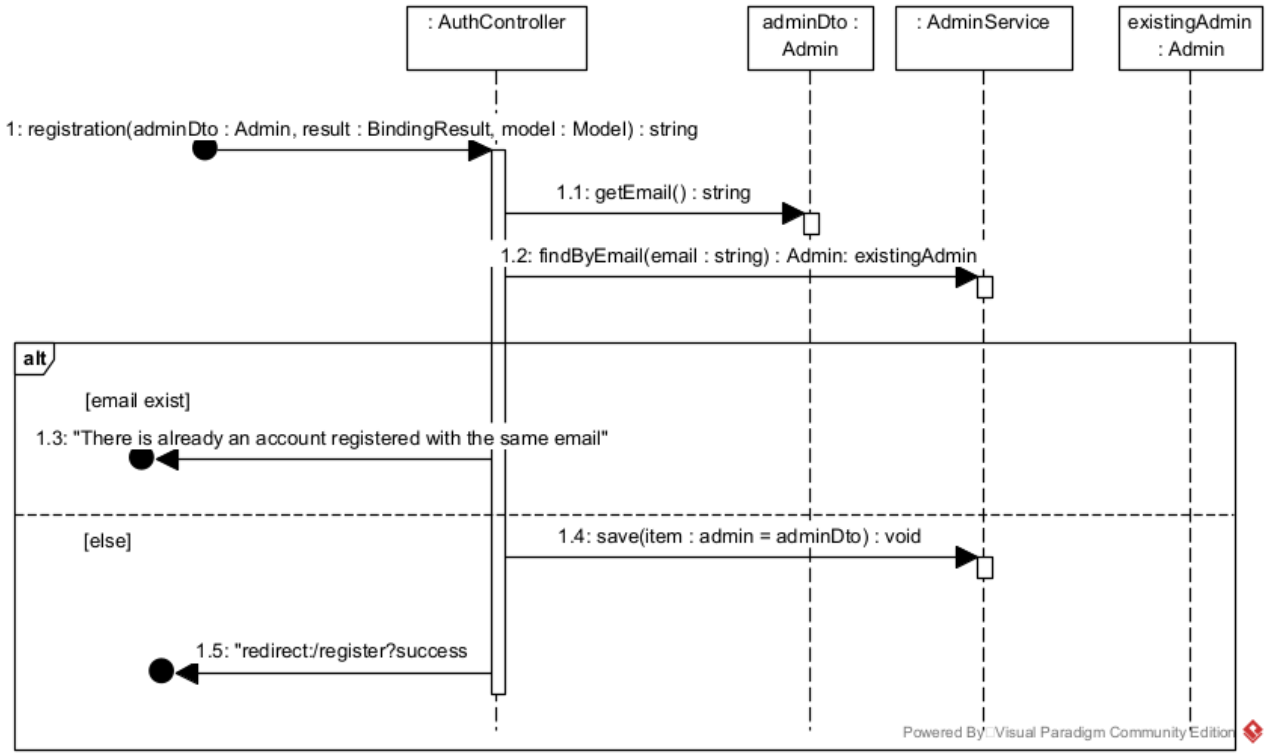
3.4.1 Class Diagram di dettaglio



- La classe *Spring Security* nel package *config*:
    - Configura i filtri di sicurezza in modo che l'accesso alla homepage, alla pagina di login, alla pagina di registrazione e alle funzionalità di download e visualizzazione delle ClassiUT nel repository sia accessibile a chiunque.
    - Configura i filtri di sicurezza in modo che l'accesso alla pagina di upload e relativa funzionalità sia consentito solo agli amministratori che hanno effettuato il login.
- La classe *CustomUserDetailsService* nel package *security* fornisce il metodo per effettuare il login.

3.4.2 Sequence Diagram: Registrazione di un amministratore

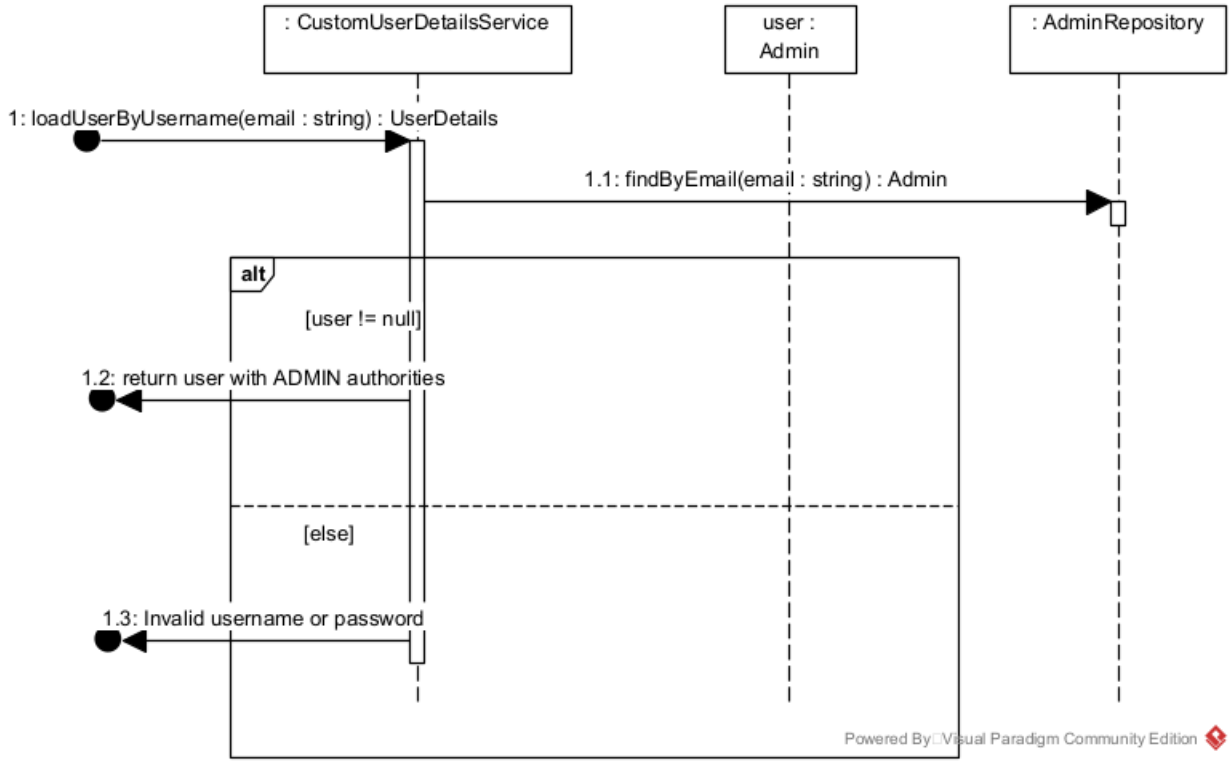
ID Storia Utente	***4***



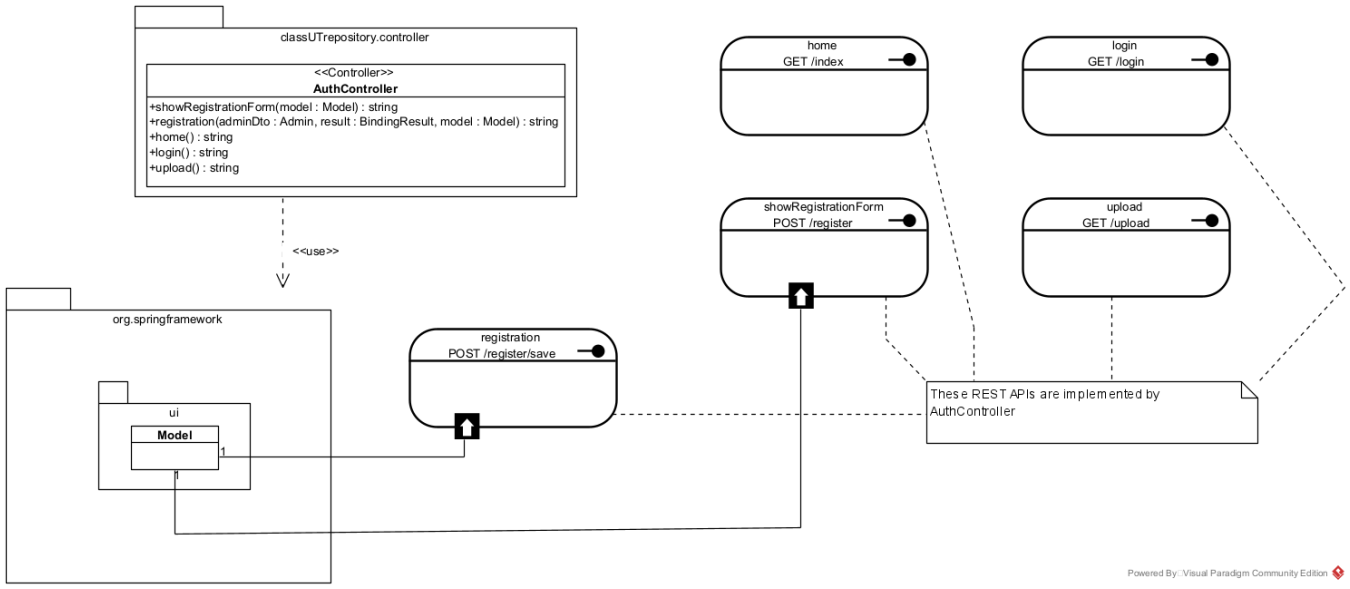
L'oggetto *result* valida i dati inseriti nel form di registrazione, in particolare, controlla che nel campo e-mail venga fornito un dominio. L'oggetto *model*, invece, serve per passare i dati dalle viste, in questo caso, quelli per la registrazione.

3.4.3 Sequence Diagram: Login

ID Storia Utente	*5*



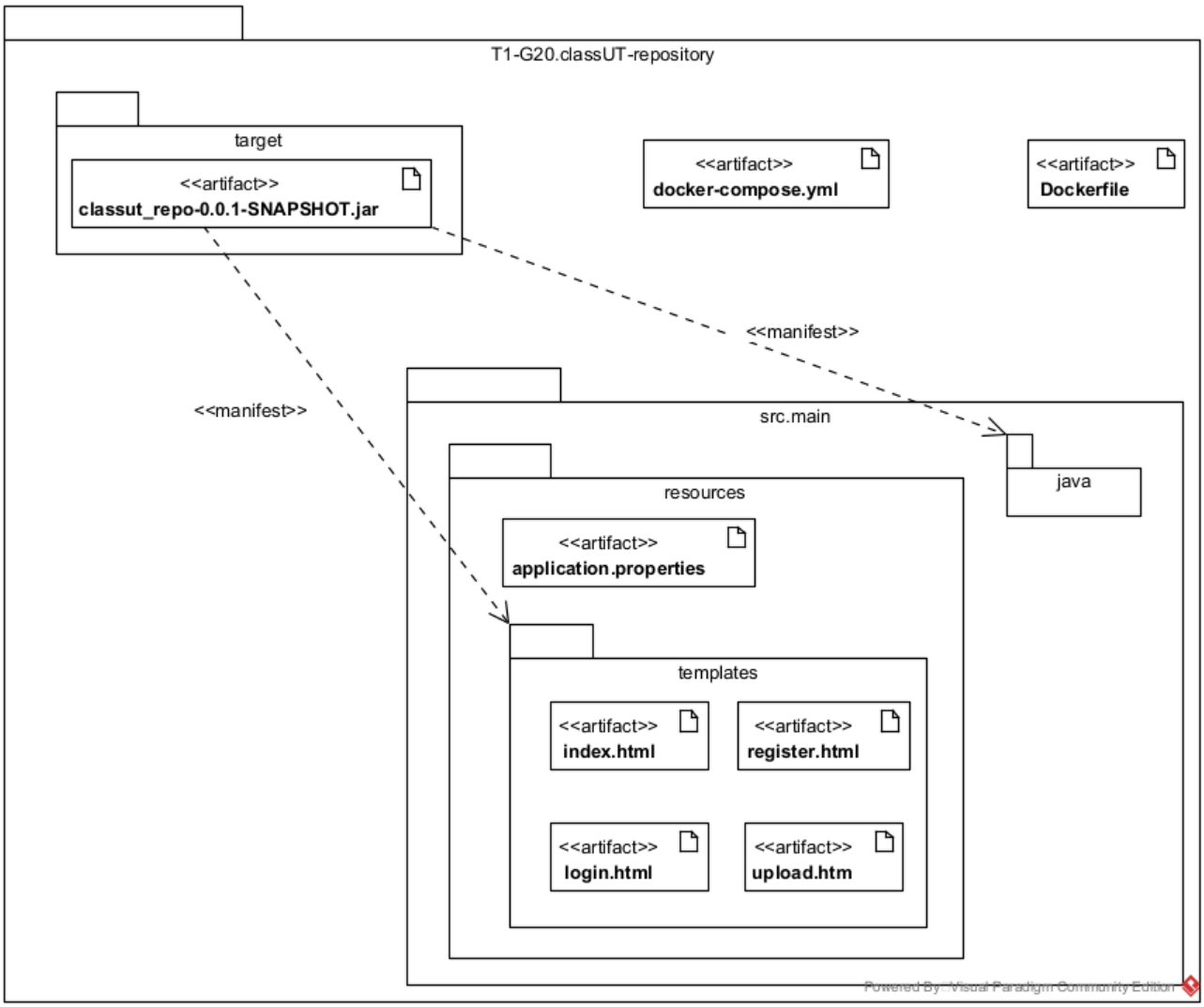
3.4.4 Specifica REST APIs per AuthController



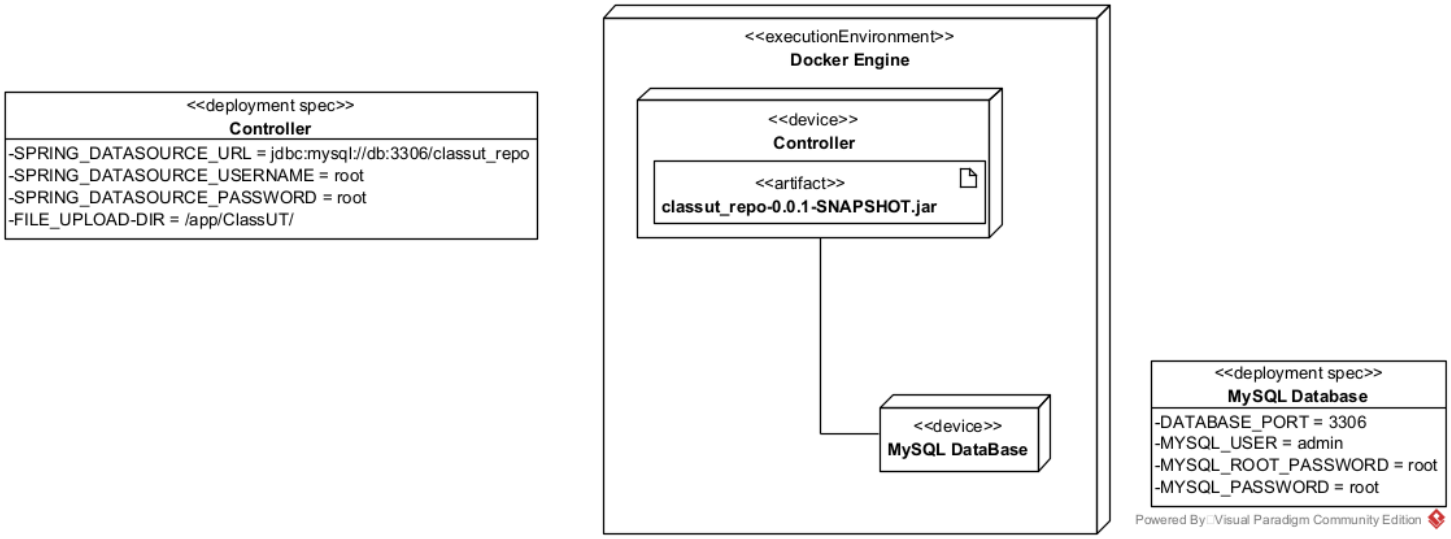
3.5 Allocation Structures

3.5.1 Vista d'installazione

Il diagramma seguente è riferito al repository presente su GitHub



3.6 Deployment View

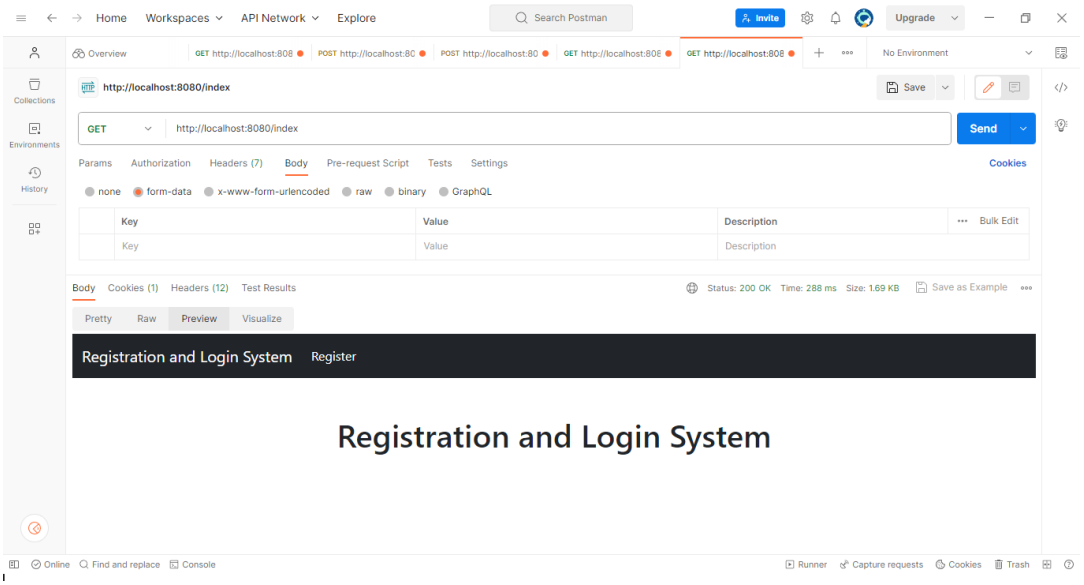


4 Test API

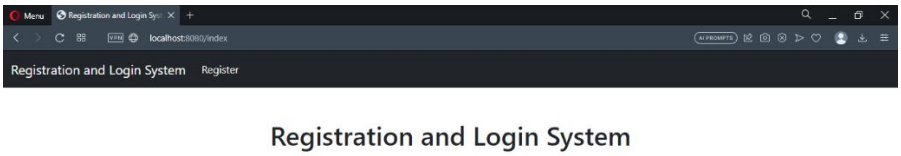
Di seguito sono riportati i test ed alcuni screenshot delle richieste effettuate tramite API con Postman.

4.1 Home Page

REST API	VALORE RITORNATO	TIPO CHIAMATA
home():string	*/index*	*GET*



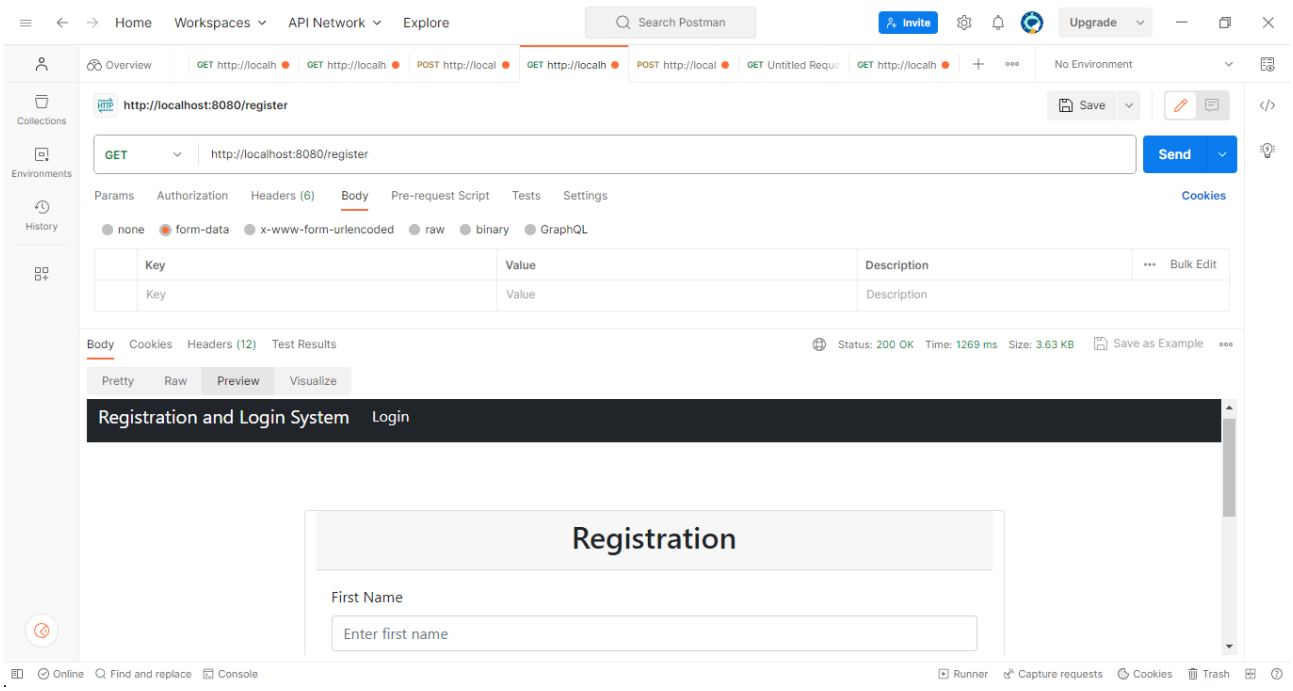
Screenshot con Postman



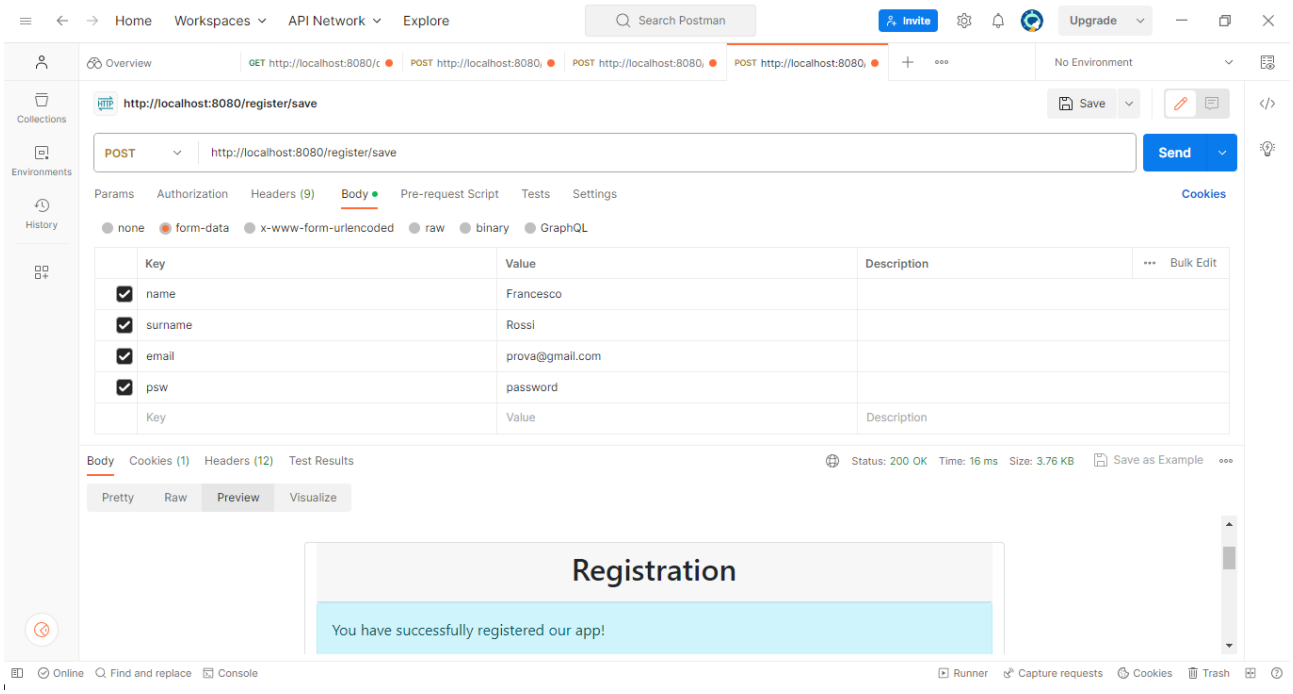
Front-end

4.2 Registrazione

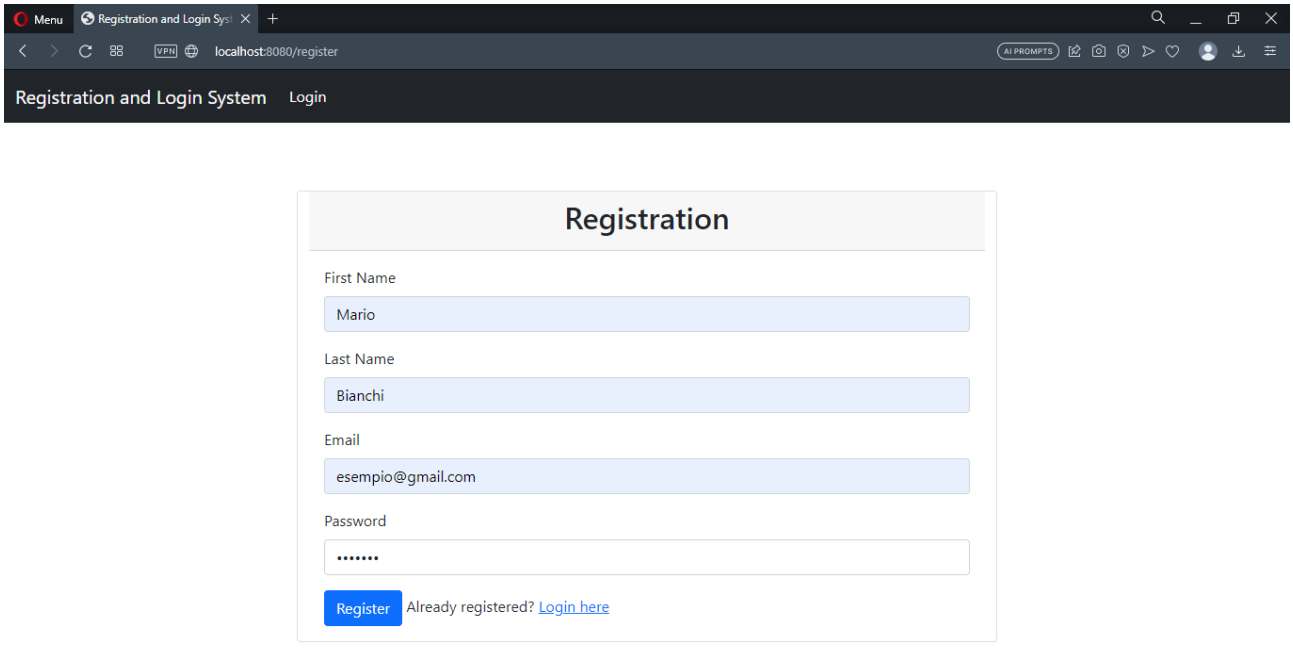
REST API	VALORE RITORNATO	TIPO CHIAMATA
showRegistrationForm(model:Model):string	*/register*	*GET*
registration(adminDto:Admin, result:BindingResult, model:Model):string	**In caso di errore**	*POST*
	/register	
	In caso di successo	
	/redirect:/register?success	



Screenshot http GET con Postman



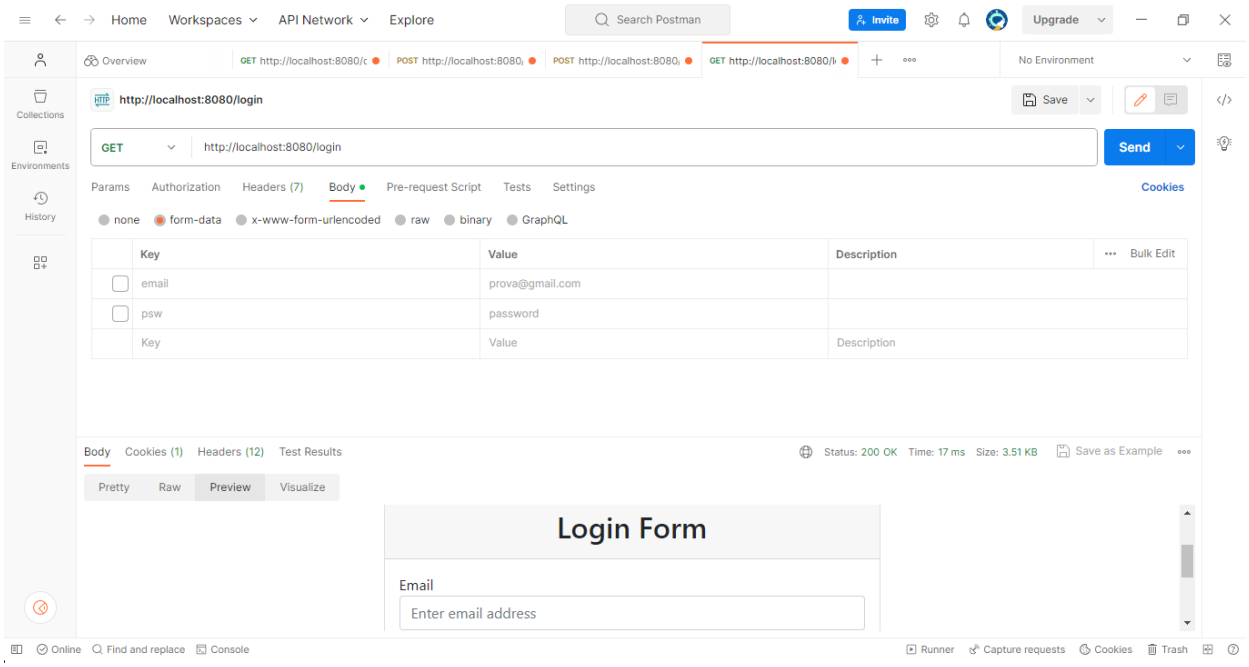
Screenshot http POST con Postman



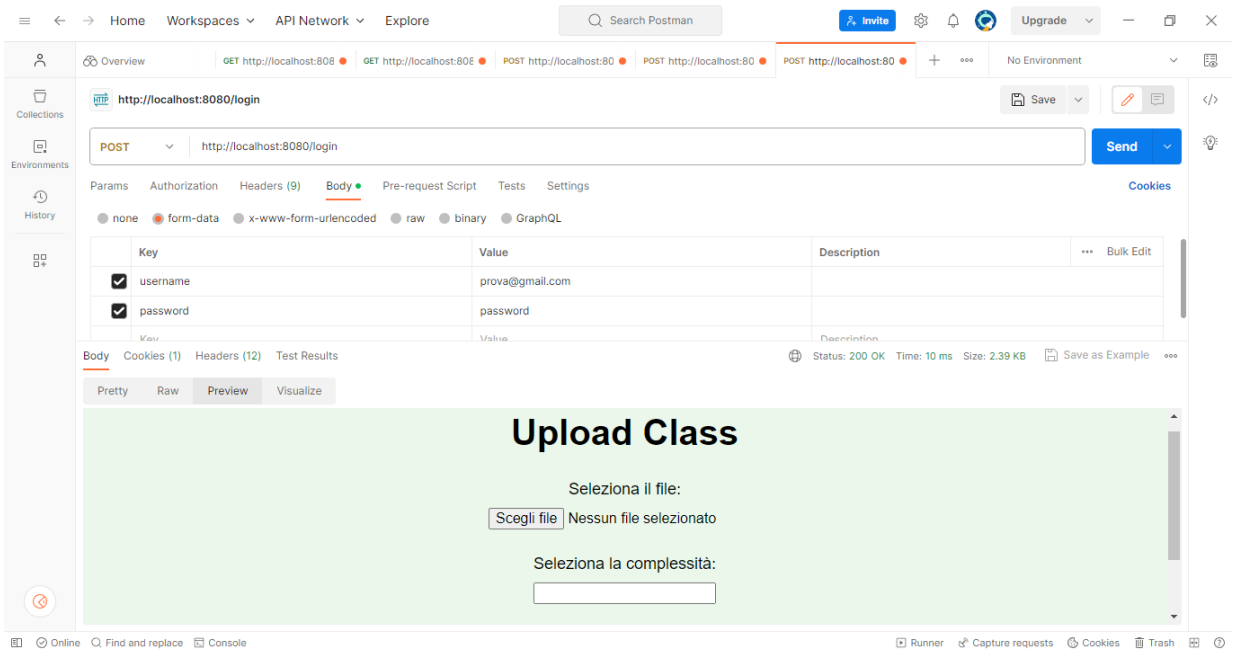
Front-end

4.3 Login

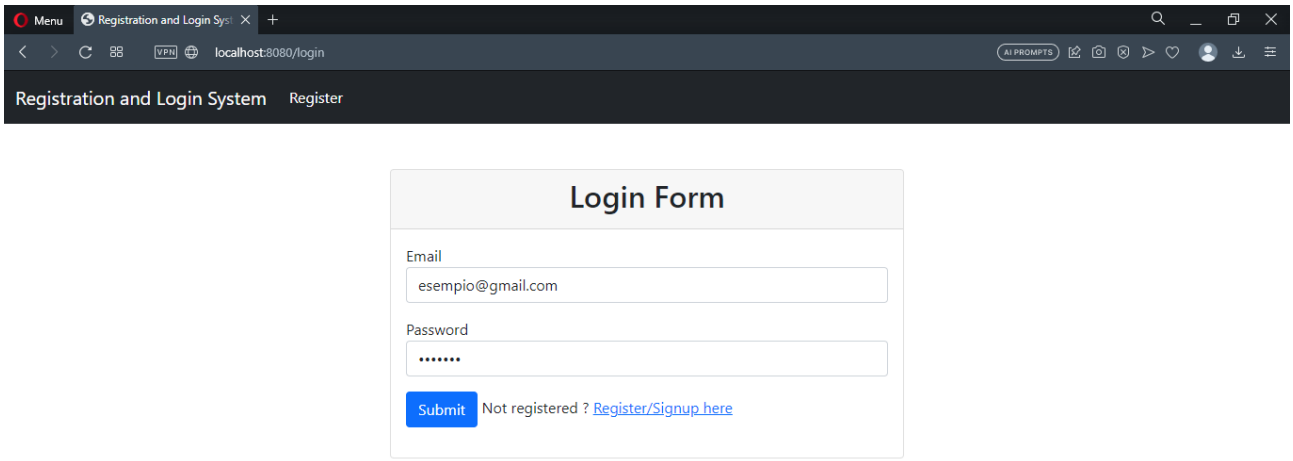
REST API	VALORE RITORNATO	TIPO CHIAMATA
login():string	*/login*	*GET*



Screenshot http GET con Postman



Screenshot http POST con Postman

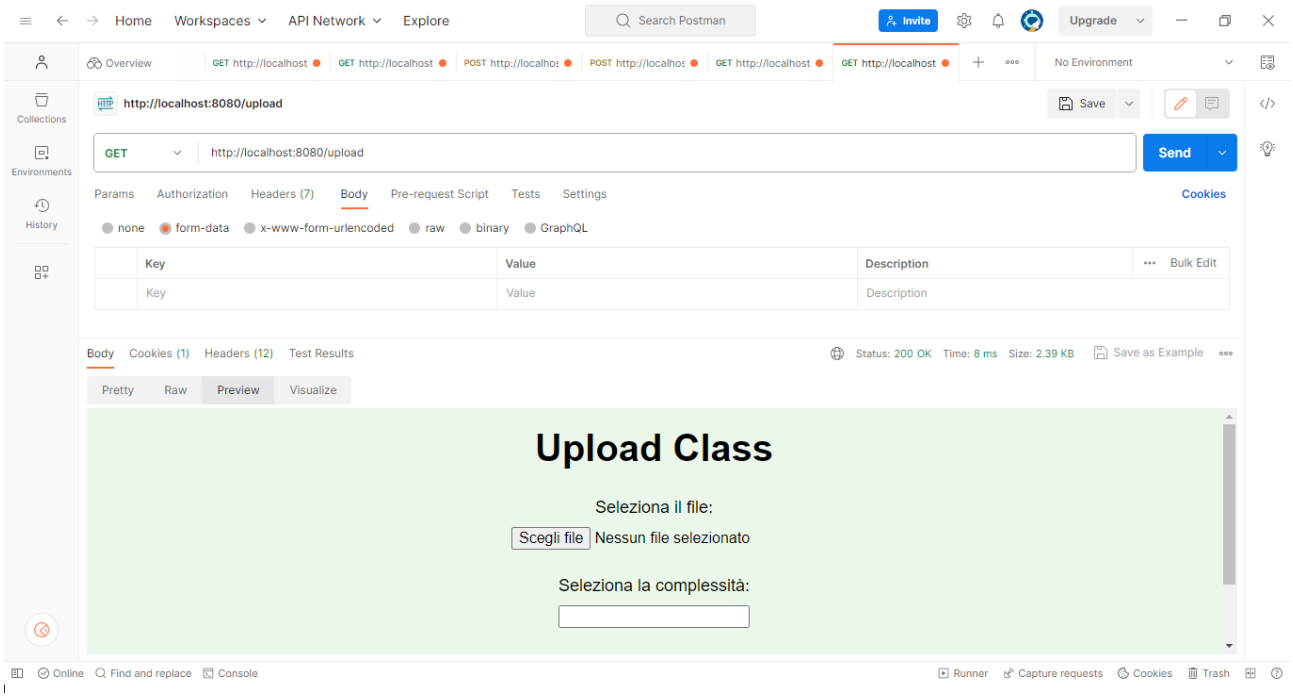

Menu
Registration and Login Sys
localhost:8080/login
Registration and Login System
Register
Login Form
Email
esempio@gmail.com
Password
Submit
Not registered ? Register/Signup here

Front-end

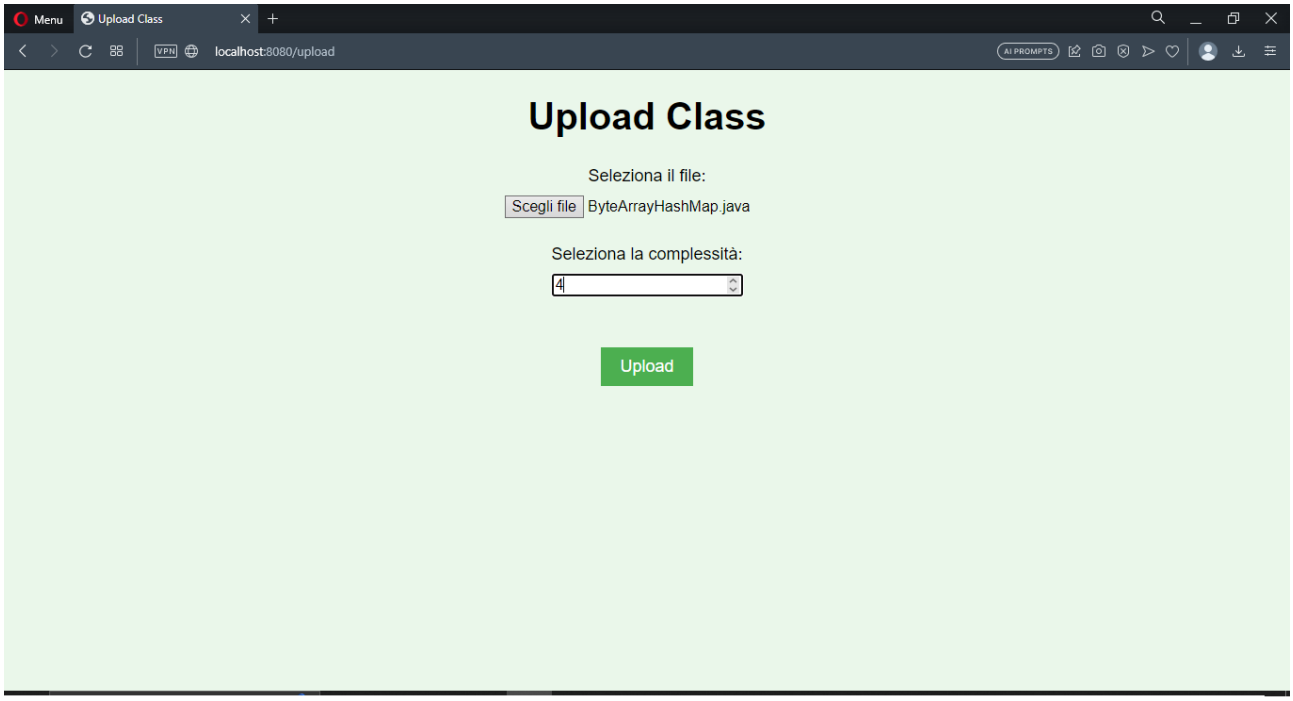
4.4 Upload di una Classe Under Test

REST API	VALORE RITORNATO	TIPO CHIAMATA
upload():string	*/upload*	*GET*
uploadClassUT(class_file:MultipartFile, complexity:int, principal:Principal):UploadClassResponse	*UploadClassResponse*	*POST*

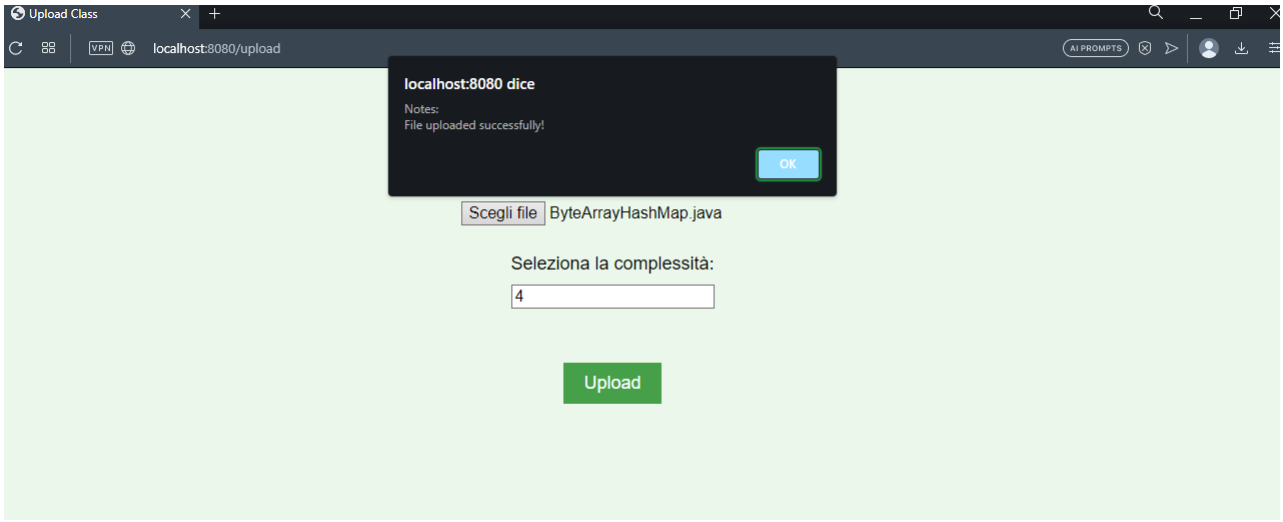
Se la richiesta ha successo, viene ottenuto il corpo della risposta dal server come oggetto JSON e quindi viene mostrato un messaggio di avviso con i dati restituiti nel campo “Notes”. Se si verifica un errore durante la richiesta, viene mostrato un alert con il messaggio di errore.



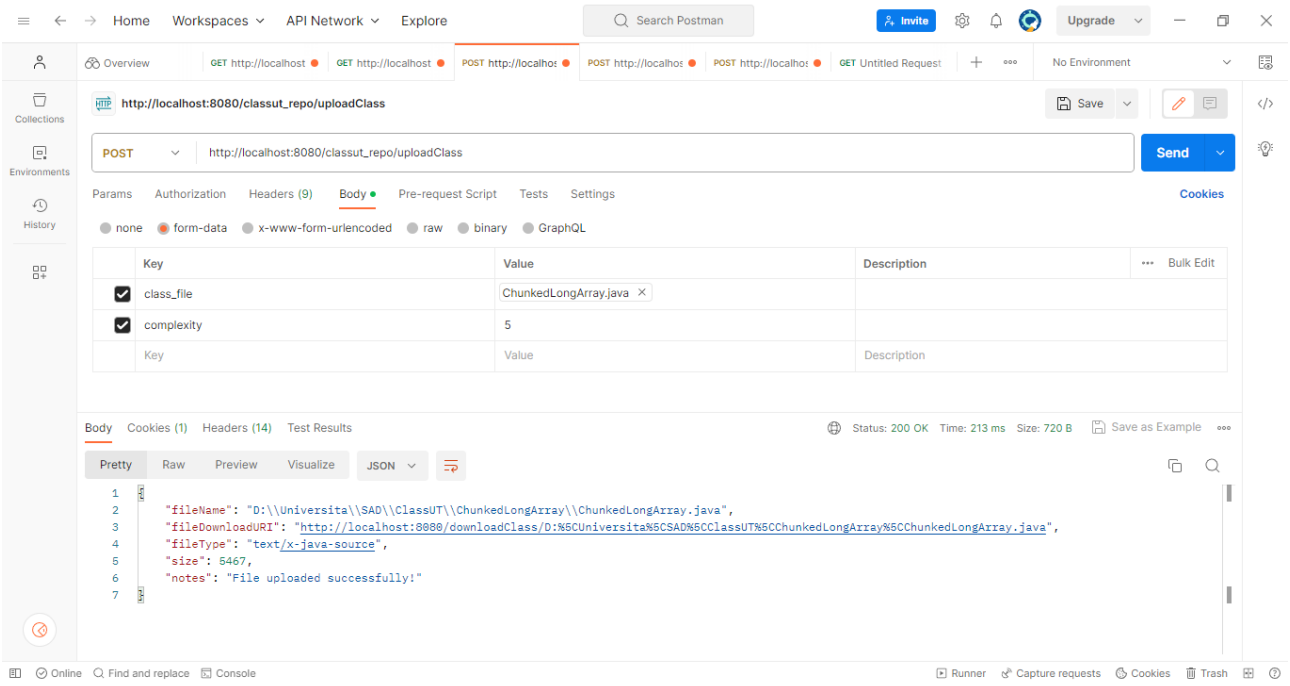
Screenshot http GET con Postman



Front-end



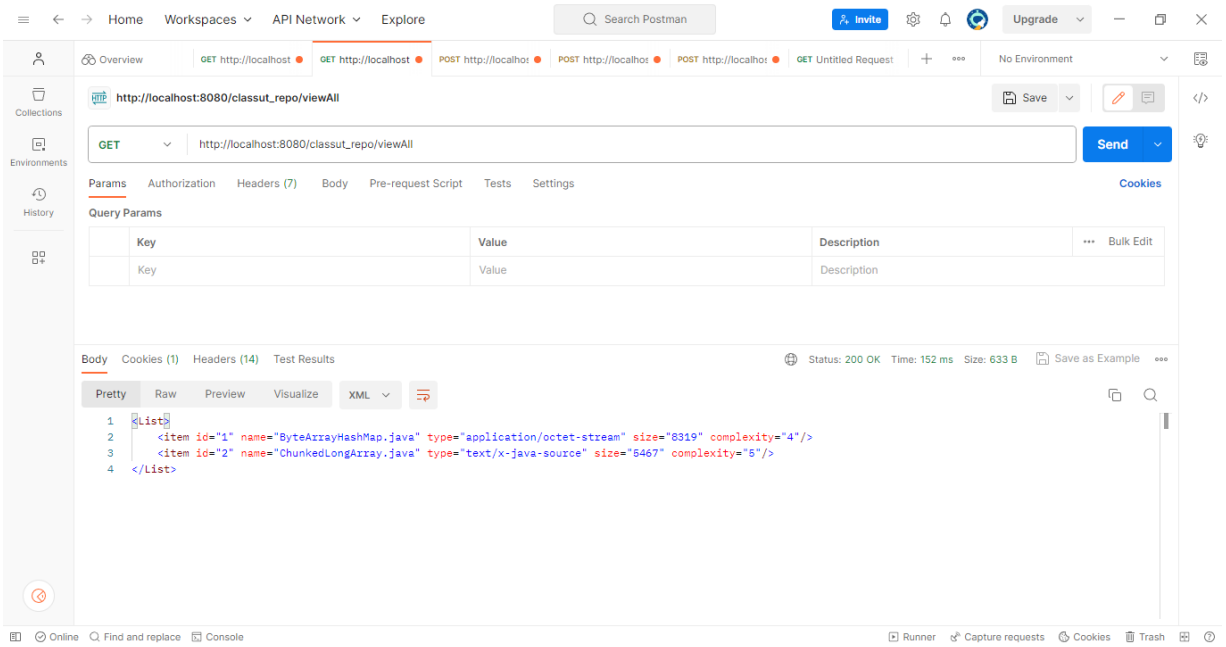
Alert che indica il corretto caricamento della classeUT



Screenshot http POST con Postman

4.5 Visualizzazione della lista delle Classi disponibili

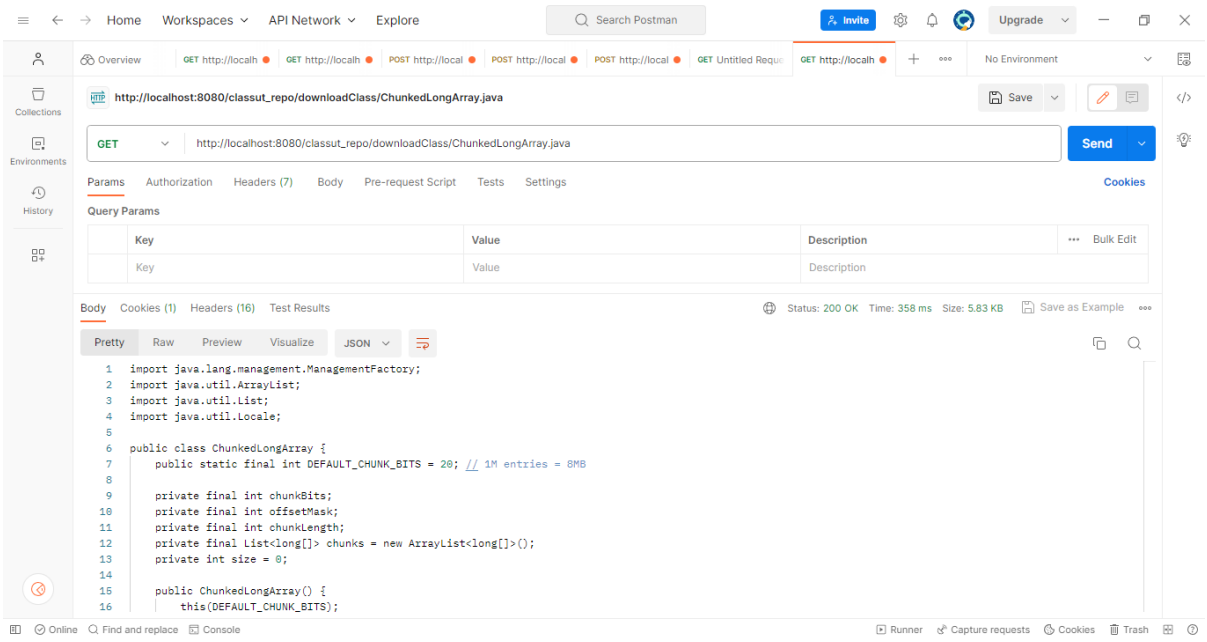
REST API	VALORE RITORNATO	TIPO CHIAMATA
viewAll():ClassUT_DTO[]	ClassUT_DTO[]	*GET*



Screenshot con Postman

4.6 Download del file di una Class Under Test

REST API	VALORE RITORNATO	TIPO CHIAMATA
downloadClassUT(fileName:string):ResponseEntity<Resource>	ResponseEntity<Resource>	*GET*



Screenshot con Postman

4.7 Test

Test Case ID	Descrizione	Precondizioni	Input	Output Attesi	Output Ottenuti	Post-condizioni Ottenute	Esito (FAIL, PASS)
1R	*Corretto inserimento dei dati di input.*	*L'amministratore non deve essere presente nella base dati.*	*Francesco Rossi prova@gmail.com password*	*"You have successfully registered our app!"*	*"You have successfully registered our app!"*	*L'amministratore si è registrato.*	*PASS*
2R	*Scorretto inserimento dei dati di input, campo e-mail non valido, assenza della @.*	*L'amministratore non deve essere presente nella base dati.*	*Mario Bianchi esempiogmail.com password*	*"Aggiungi un simbolo @ nell'indirizzo e-mail"*	*"You have successfully registered our app!"*	*L'amministratore non si è registrato.*	*FAIL*
3R	*Amministratore già registrato, effettua una registrazione con le stesse credenziali.*	*L'amministratore è già presente nella base di dati.*	*Francesco Rossi prova@gmail.com password*	*"There is already an account registered with the same email"*	*"There is already an account registered with the same email"*	*L'amministratore non si è registrato.*	*PASS*

1L	*Corretto inserimento dei dati di input.*	*L'amministratore si è già registrato.*	*prova@gmail.com password*	*Reindirizzamento alla pagina di upload*	*Reindirizzamento alla pagina di upload*	*L'amministratore può caricare il codice di una nuova classeUT.*	*PASS*
2L	*Scorretto inserimento dei dati di input, e-mail non corretta.*	*L'amministratore si è già registrato.*	*pro@gmail.com password*	*"Invalid Email or Password"*	*"Invalid Email or Password"*	*Amministratore non autenticato.*	*PASS*
3L	*Scorretto inserimento dei dati di input, password non corretta.*	*L'amministratore si è già registrato.*	*prova@gmail.com passwo*	*"Invalid Email or Password"*	*"Invalid Email or Password"*	*Amministratore non autenticato.*	*PASS*
1U	*Corretto inserimento dei dati di input.*	*L'amministratore si è autenticato.*	*ChunkedLongArray.java 5*	*"File uploaded successfully!"*	*"File uploaded successfully!"*	*La classUT è stata caricata.*	*PASS*
2U	*Corretto inserimento dei dati di input*	*L'amministratore non si è autenticato.*	*ChunkedLongArray.java 5*	*http 403 Forbidden*	*http 403 Forbidden*	*La classUT non è stata caricata.*	*PASS*
3U	*Incorretto inserimento dei dati input, classUT con lo stesso nome di una già nella base di dati.*	*L'amministratore si è autenticato.*	*ChunkedLongArray.java 5*	*"Errors occurred during saving"*	*"Errors occurred during saving"*	*La classUT non è stata caricata nuovamente.*	*PASS*
1D	*Corretto inserimento dei dati di input*	*La classe cercata è presente nella base di dati.*	*ChunkedLongArray.java*	*ChunkedLongArray.java*	*ChunkedLongArray.java*	*Il codice della classUT scelta è stato scaricato*	*PASS*
2D	*Scorretto inserimento dei dati di input, la classe specificata non è disponibile.*	*La classe cercata non è presente nella base di dati.*	*Esempio.java*	*http 404 Not Found*	*http 404 Not Found*	*La classUT non è stata trovata.*	*PASS*
1V	*Richiesta effettuata correttamente.*	*Ci sono classUT nella base di dati.*	-	*Lista delle classi disponibili*	*Lista delle classi disponibili*	*Vengono visualizzate tutte le classUT contenute nel repository.*	*PASS*
2V	*Richiesta effettuata correttamente.*	*Non ci sono classUT nella base di dati.*	-	*Lista vuota*	*Lista vuota*	*Vengono visualizzate tutte le classUT contenute nel repository.*	*PASS*

5 Guida all'installazione

5.1 Docker Desktop

Pre-requisiti:

- Docker Desktop

Di seguito sono riportati i passi per l'installazione ed esecuzione del componente sviluppato:

- Clonare il repository da GitHub
- Editare il file *docker-compose.yml* presente nella cartella *./classUT-repository* specificando:
    - la porta dell'host su cui deve essere raggiungibile il *web-server*, ad esempio **8081**. In questo modo è possibile accedere al componente dal proprio browser.
    - il *path* assoluto dove deve essere mappato il **volume** per il salvataggio dei file delle ClassiUT. Tale volume conferisce al container la capacità di salvare i file caricati su una porzione di filesystem dell'host.
    - il *path* assoluto dove devono essere memorizzati le informazioni del database al fine di essere mantenute anche dopo lo *shut-down* del container.
- Aprire un terminale nella directory sopracitata e digitare il comando:

docker-compose up

5.2 Ambiente Windows

Pre-requisiti:

- MySQL Workbench 8.0

Di seguito sono riportati i passi per l'installazione ed esecuzione del componente:

- Clonare il repository da GitHub
- Editare il file *application.properties* specificando:
    - URL del server database a cui il componente deve collegarsi
    - Il *path* assoluto dove devono essere salvati file caricati