|  |  |  |
| --- | --- | --- |
| Immagine che contiene disegnando  Descrizione generata automaticamenteScuola Politecnica e delle Scienze di Base  Corso di Laurea Magistrale in Ingegneria Informatica | | |
| repository  classi under test | | |
| Relazione Finale  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 1](#_Toc139645242)

[1.1 Requisiti specifici del task 1](#_Toc139645243)

[1.2 Requisiti aggiuntivi 1](#_Toc139645244)

[1.2.1 Registrazione degli amministratori 1](#_Toc139645245)

[1.2.2 Autenticazione degli amministratori 1](#_Toc139645246)

[2 Analisi 2](#_Toc139645247)

[2.1 Use Case Diagram 2](#_Toc139645248)

[2.2 Storie utente 3](#_Toc139645249)

[2.3 Scenari 4](#_Toc139645250)

[2.3.1 Registration 4](#_Toc139645251)

[2.3.2 View All 5](#_Toc139645252)

[2.4 Glossario dei termini 6](#_Toc139645253)

[2.5 Diagrammi di analisi 6](#_Toc139645254)

[2.5.1 Class Diagram di analisi 6](#_Toc139645255)

[2.5.2 Sequence Diagram: Download del file di una Class Under Test 7](#_Toc139645256)

[2.5.3 Sequence Diagram: Upload di una Classe Under Test 8](#_Toc139645257)

[3 Progettazione 9](#_Toc139645258)

[3.1 Component Diagram 10](#_Toc139645259)

[3.2 Module Structures 11](#_Toc139645260)

[3.3 Requisiti specifici 12](#_Toc139645261)

[3.3.1 Class Diagram di dettaglio 12](#_Toc139645262)

[3.3.2 Sequence Diagram: Visualizzazione della lista delle Classi disponibili 15](#_Toc139645263)

[3.3.3 Sequence Diagram: Download del file di una Class Under Test 16](#_Toc139645264)

[3.3.4 Sequence Diagram: Upload di una Classe Under Test 17](#_Toc139645265)

[3.3.5 Generazione del path della Classe Under Test 19](#_Toc139645266)

[3.4 Requisiti aggiuntivi 20](#_Toc139645267)

[3.4.1 Class Diagram di dettaglio 20](#_Toc139645268)

[3.4.2 Sequence Diagram: Registrazione di un amministratore 21](#_Toc139645269)

[21](#_Toc139645270)

[3.4.3 Sequence Diagram: Login 22](#_Toc139645271)

[3.4.4 Specifica REST APIs per AuthController 22](#_Toc139645272)

[3.5 Allocation Structures 23](#_Toc139645273)

[3.5.1 Vista d’installazione 23](#_Toc139645274)

[3.6 Deployment View 24](#_Toc139645275)

[24](#_Toc139645276)

[4 Test API 25](#_Toc139645277)

[4.1 Home Page 25](#_Toc139645278)

[4.2 Registrazione 26](#_Toc139645279)

[4.3 Login 28](#_Toc139645280)

[4.4 Upload di una Classe Under Test 29](#_Toc139645281)

[4.5 Visualizzazione della lista delle Classi disponibili 32](#_Toc139645282)

[4.6 Download del file di una Class Under Test 33](#_Toc139645283)

[5 Guida all’installazione ed utilizzo del componente 34](#_Toc139645284)

# 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.

## 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.

## Requisiti aggiuntivi

### Registrazione degli amministratori

L’applicazione deve consentire agli amministratori di registrarsi. All’atto della registrazione, l’amministratore fornirà nome, cognome, un indirizzo e-mail valido ed una password, il sistema dopo aver controllato la validità dei dati forniti, aggiungerà l’amministratore all’elenco degli amministratori registrati e gli assocerà un Id univoco.

### Autenticazione degli amministratori

All’atto della autenticazione, l’amministratore fornirà l’indirizzo e-mail fornito per la registrazione e la relativa password, il sistema dopo aver controllato la validità dei dati forniti, autenticherà l’amministratore e gli fornirà una schermata per il caricamento delle Classi Java da testare.

# Analisi

## Use Case Diagram

Immagine che contiene diagramma, linea, Parallelo

Descrizione generata automaticamente

## 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, in modo da potermi registrare. | Aggiuntivo |
| 5 | Come amministratore registrato, voglio accedere alla pagina di autenticazione, in modo da potermi autenticare. | Aggiuntivo |

## Scenari

### 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 clicca su “Register” 3. L’amministratore, nella schermata di registrazione, inserisce i dati richiesti e clicca nuovamente su “Register” |
| 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 |

### 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 i dati nei vari campi e clicca su “Upload” |
| 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 |

### 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 | - |

### 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 |

## Glossario dei termini

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| ClassUT | Classe Java da testare |
| Admin | Amministratore registrato |
| Test | Insieme di Test generati automaticamente tramite Randoop o Evosuite relativi ad una specifica classUT |

## Diagrammi di analisi

### Class Diagram di analisi

Immagine che contiene testo, diagramma, Piano, Parallelo

Descrizione generata automaticamente

La classe ClassUT\_IE fornisce i servizi per gestire le operazioni CRUD delle classUT

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

Sequence diagram relativo al caso d’uso Download.

Immagine che contiene testo, diagramma, linea, schermata

Descrizione generata automaticamente

### Sequence Diagram: Upload di una Classe Under Test

Sequence diagram relativo al caso d’uso Upload.

Immagine che contiene testo, diagramma, Parallelo, linea

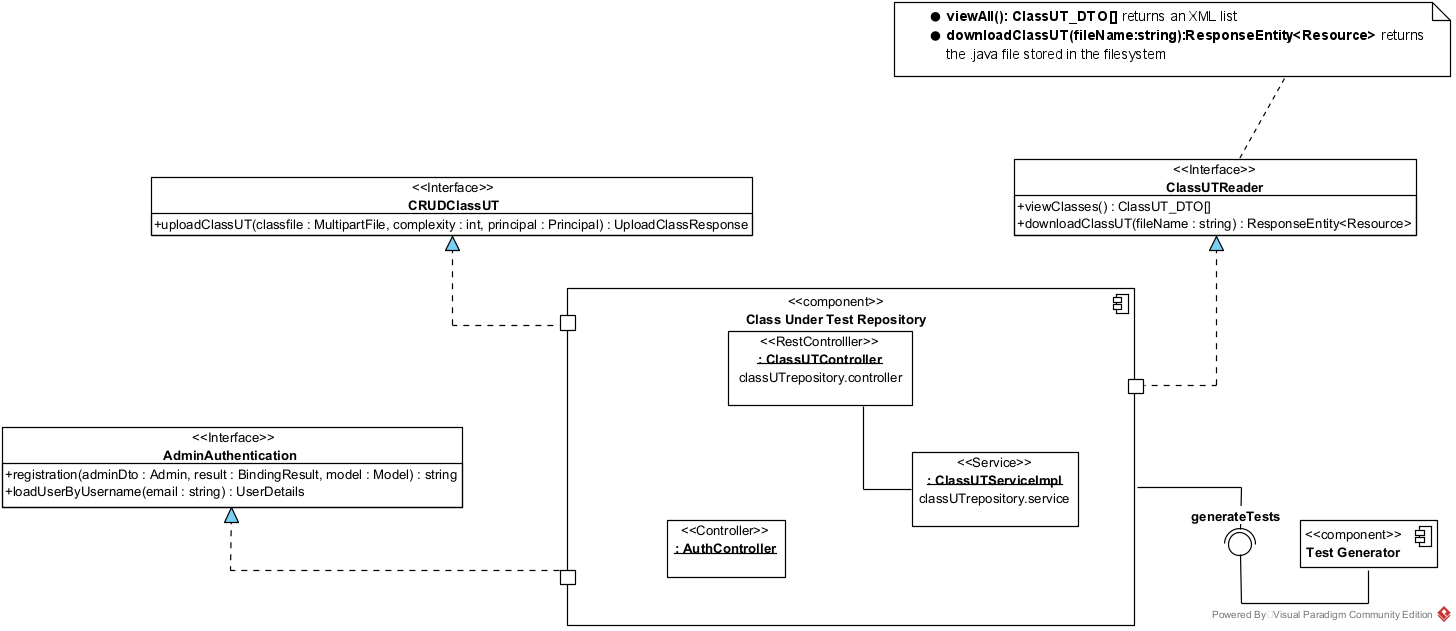
Descrizione generata automaticamente

# 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 |

## 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 |

## Module Structures

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

Immagine che contiene testo, schermata, diagramma, schermo

Descrizione generata automaticamente

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 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.

## Requisiti specifici

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

### Class Diagram di dettaglio

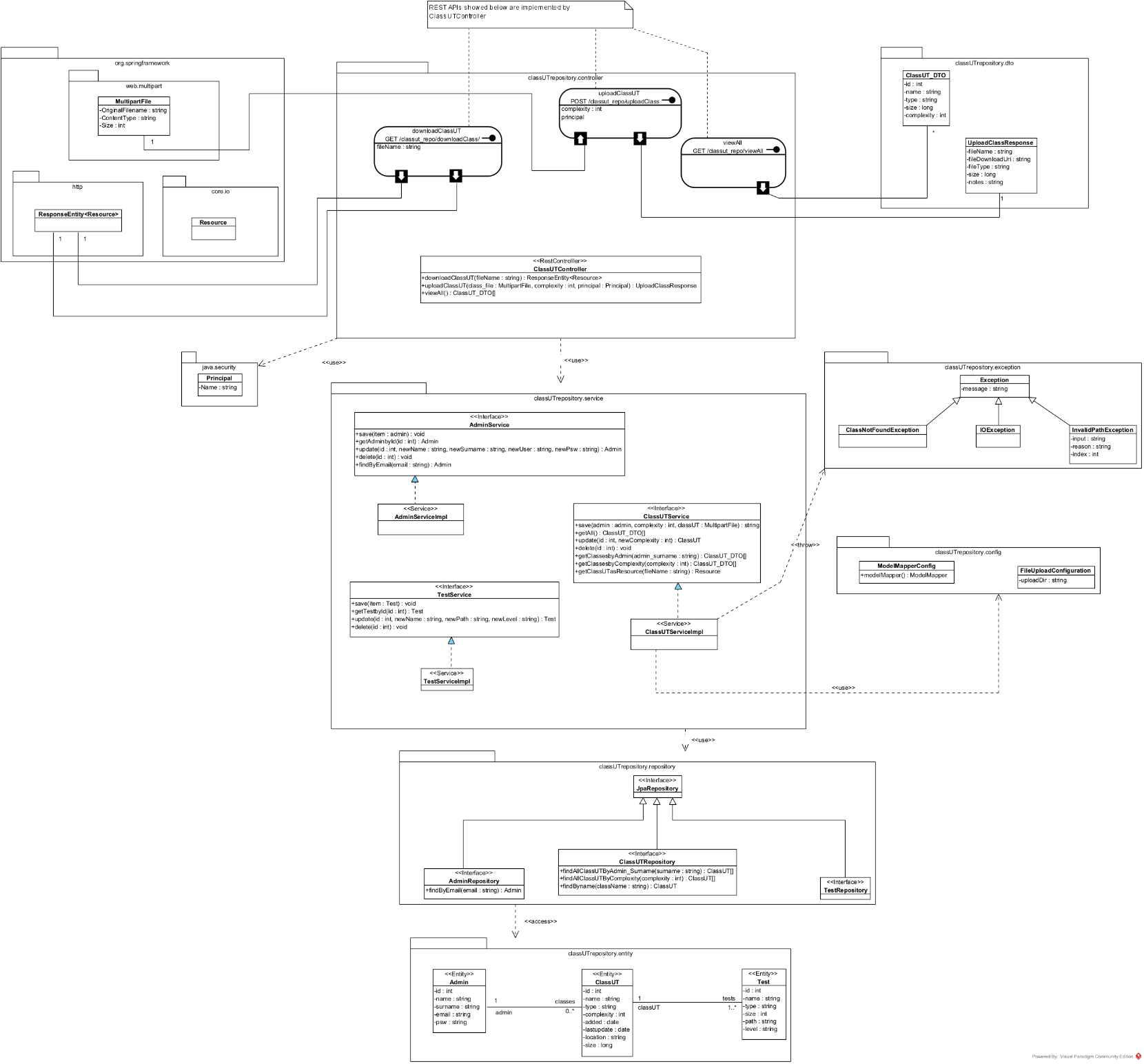


Immagine che contiene testo, schermata, diagramma, Parallelo

Descrizione generata automaticamenteIl 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 |

Immagine che contiene testo, schermata, diagramma, Parallelo

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

Immagine che contiene testo, schermata, diagramma, Parallelo

Descrizione generata automaticamenteSi 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*.

### Sequence Diagram: Visualizzazione della lista delle Classi disponibili

|  |  |
| --- | --- |
| ***ID Storia Utente*** | ***1*** |

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente

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

|  |  |
| --- | --- |
| ***ID Storia Utente*** | ***2*** |

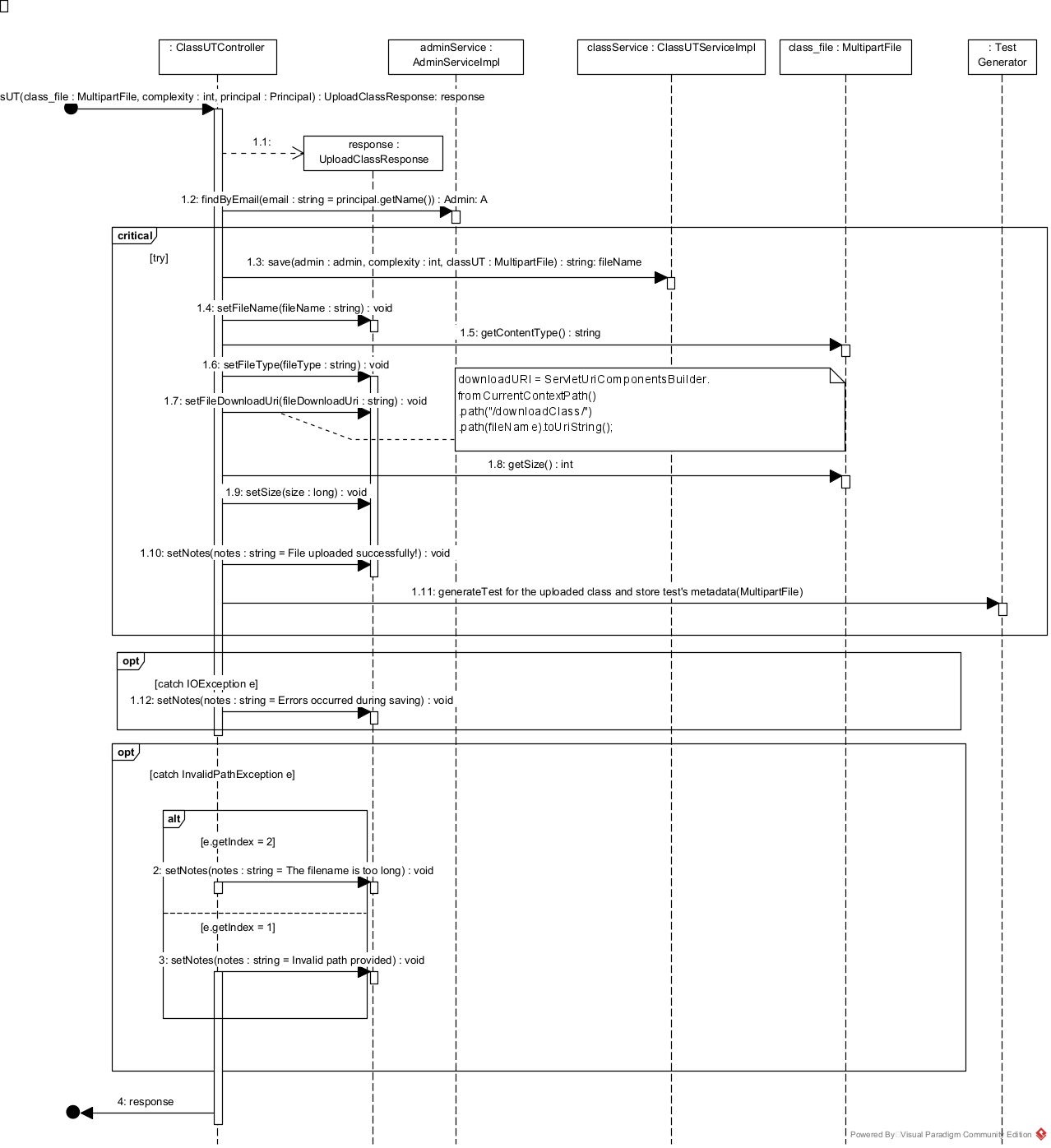
Immagine che contiene schermata, testo, diagramma

Descrizione generata automaticamente

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

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

Immagine che contiene schermata, testo, diagramma, design

Descrizione generata automaticamenteIn 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.

1. Errori durante il salvataggio sul filesystem:

Immagine che contiene schermata, testo, diagramma, design

Descrizione generata automaticamente

### 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.

* *ClassUT\_directory* si ottiene a partire dal nome file caricato ed eliminando l’estensione *.java*

## Requisiti aggiuntivi

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

### Immagine che contiene testo, schermata, Rettangolo, diagramma Descrizione generata automaticamenteClass 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.

### Sequence Diagram: Registrazione di un amministratore

|  |  |
| --- | --- |
| ***ID Storia Utente*** | ***4*** |

### Immagine che contiene schermata, testo, Carattere, diagramma Descrizione generata automaticamente

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.

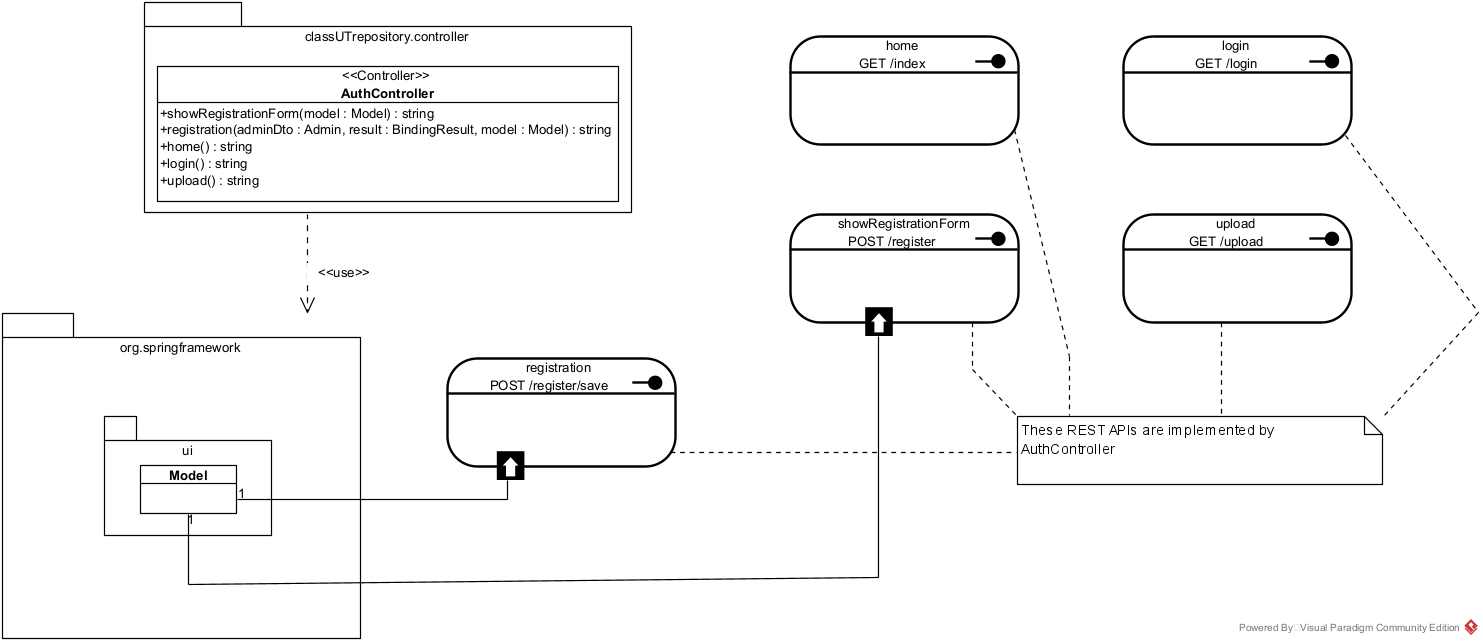
### Sequence Diagram: Login

|  |  |
| --- | --- |
| ***ID Storia Utente*** | ***5*** |

Immagine che contiene schermata, testo, Carattere

Descrizione generata automaticamente

### Specifica REST APIs per AuthController



## Allocation Structures

### Vista d’installazione

Immagine che contiene testo, diagramma, schermata, Parallelo

Descrizione generata automaticamenteIl diagramma seguente è riferito al repository presente su GitHub

## Deployment View

## Immagine che contiene testo, schermata, design Descrizione generata automaticamente

# Test API

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

## Home Page

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

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

*Screenshot con Postman*

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente*Front-end*

## Registrazione

|  |  |  |
| --- | --- | --- |
| **REST API** | **VALORE**  **RITORNATO** | **TIPO**  **CHIAMATA** |
| showRegistrationForm( model:Model):string | */register* | *GET* |
| registration(adminDto:Admin, result:BindingResult, model:Model):string | **Se ci sono stati errori** | *POST* |
| */register* |
| **Se non ci sono stati errori** |
| */redirect:/register?success* |

Immagine che contiene testo, schermata, software, Icona del computer

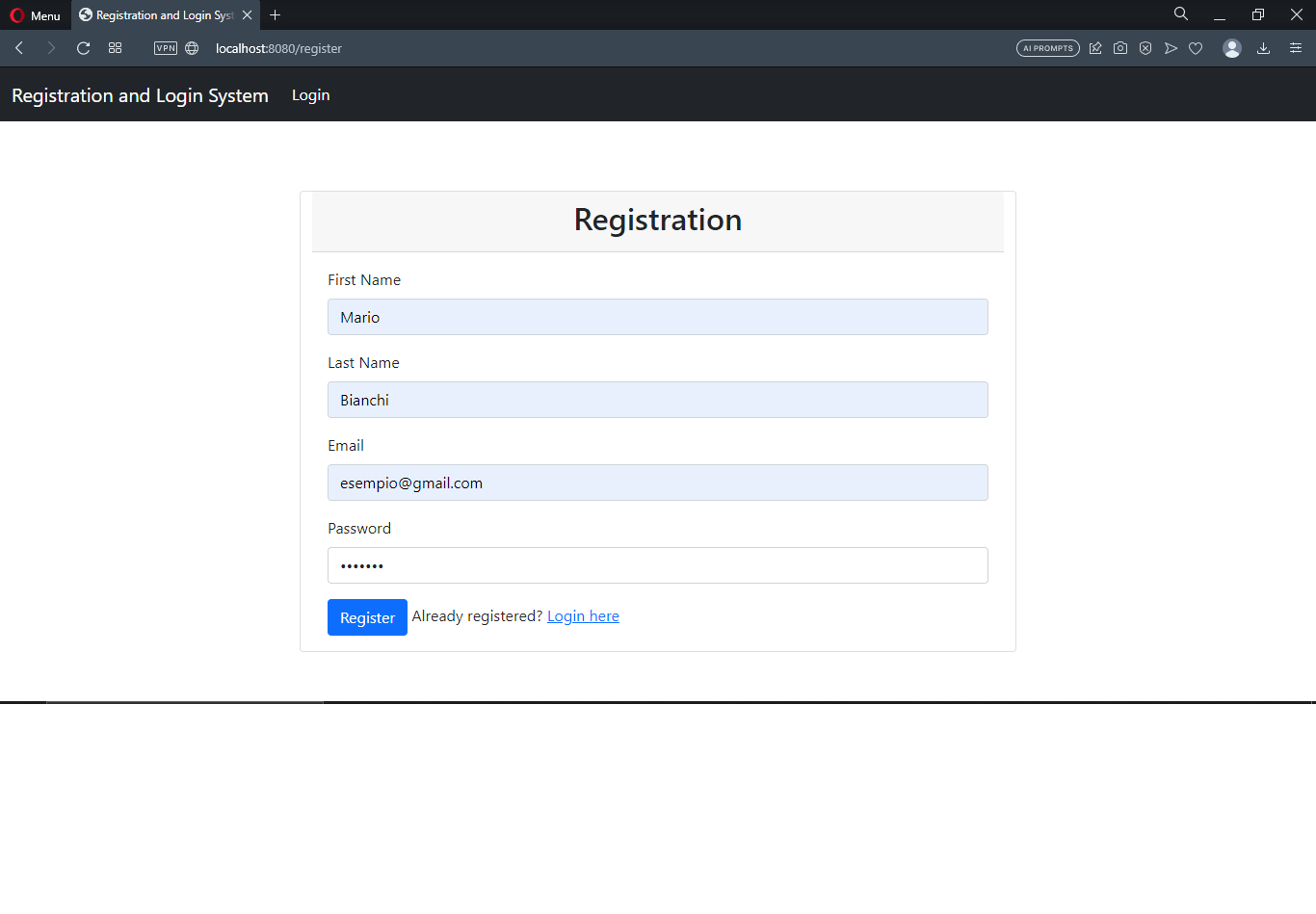
Descrizione generata automaticamente

*Screenshot http GET con Postman*

Immagine che contiene testo, schermata, numero, software

Descrizione generata automaticamente

*Screenshot http POST con Postman*



*Front-end*

## Login

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

Immagine che contiene testo, schermata, software, numero

Descrizione generata automaticamente

*Screenshot http GET con Postman*

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

*Screenshot http POST con Postman*

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

*Front-end*

## 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* |

Quando l'amministratore invia il modulo facendo clic sul pulsante "Upload" l’API crea un oggetto FormData che rappresenta i dati del modulo, e quindi invia una richiesta POST a " <http://localhost:8080/classut_repo/uploadClass> " utilizzando l'API Fetch.

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.

Immagine che contiene testo, schermata, software, Pagina Web

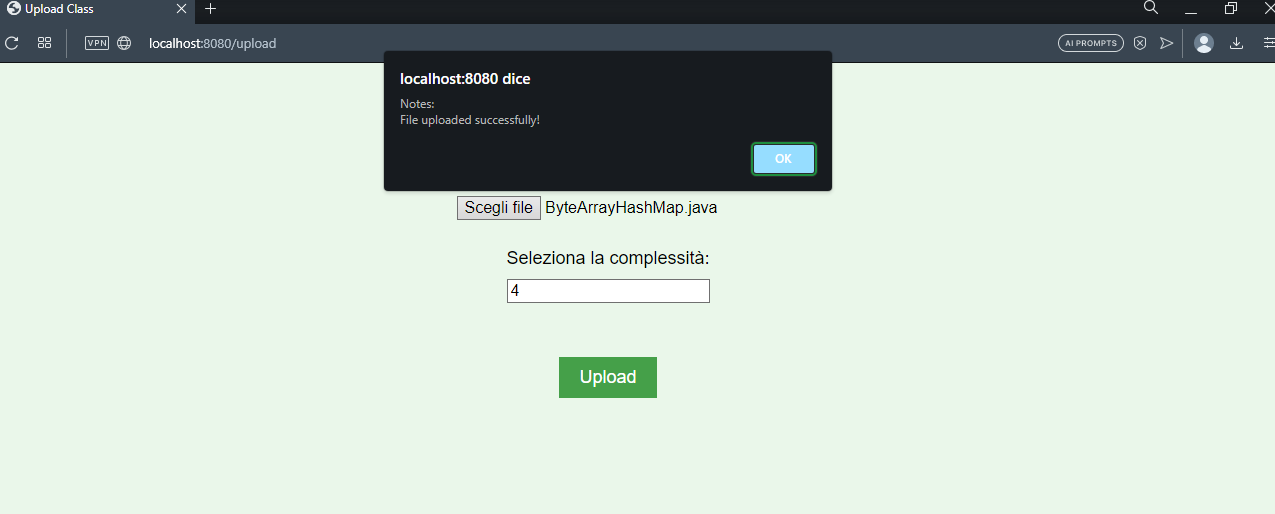
Descrizione generata automaticamente

*Screenshot http GET con Postman*

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

*Front-end*



*Alert che indica il corretto caricamento della classeUT*

Immagine che contiene testo, schermata, software, numero

Descrizione generata automaticamente

*Screenshot http POST con Postman*

## Visualizzazione della lista delle Classi disponibili

|  |  |  |
| --- | --- | --- |
| **REST API** | **VALORE**  **RITORNATO** | **TIPO CHIAMATA** |
| viewAll():ClassUT\_DTO |  | *GET* |

Immagine che contiene testo, software, Icona del computer, numero

Descrizione generata automaticamente

*Screenshot con Postman*

## Download del file di una Class Under Test

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

Immagine che contiene testo, schermata, software, numero

Descrizione generata automaticamente

*Screenshot con Postman*

# Guida all’installazione

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 host su cui deve essere raggiungibile il *web-server*, ad esempio **8081**.
  + il *path* assoluto dove deve essere mappato il **volume** per il salvataggio dei file delle ClassiUT.
  + 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*