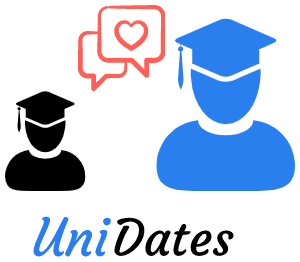
|  |
| --- |
| **Università degli Studi di Salerno Corso di Ingegneria del Software** |

**UniDates  
Object Design Document  
Version 1.5**



Data: 5/02/2020

**Coordinatore del progetto:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Andrea De Lucia |  |

**Partecipanti:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Alessandro Pio D’Antuono [APD] | 0512105700 |
| Antonio Fasulo [AF] | 0512105904 |
| Raimondo Rapacciuolo [RR] | 0512105778 |
| Roberta Gambardella [RG] | 0512106012 |

|  |  |
| --- | --- |
| **Scritto da:** |  |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 10/12/2020 | 1.1 | Aggiunta di un template per il documento e introduzione | Tutti |
| 5/01/2020 | 1.2 | Packages | Tutti |
| 9/01/2020 | 1.3 | Interfaccia delle classi | Tutti |
| 10/01/2020 | 1.4 | Contratti | Tutti |
| 5/02/2021 | 1.5 | Class Diagram e Revisione | Tutti |

**Object Design**

1. **Introduzione**

**1.1 Object Design trade-offs**

**[Ottimizzazione vs Portabilitá]**

Al costo di presentare una minore ottimizzazione del sistema, quest’ultimo andrá a favorire una maggiore portabilitá, potendo essere eseguito sulla maggior parte dei dispositivi, con le medesime caratteristiche.

**[Manutenzione vs Disponibilitá]**

Al costo di avere una minore disponibilità dei servizi durante il periodo di manutenzione, sarà possibile effettuare manutenzioni più veloci e dirette mirata a risolvere i problemi presenti.

**[Scalabilitá vs Costi]**

Il sistema presenterà una scalabilità elevata anche se questo dovrebbe inficiare su maggiori costi di gestione.

**[Sviluppo rapido vs Funzionalitá]**

Al costo di velocizzare la fase di sviluppo, il sistema è stato ridotto di alcune funzionalitá.

**1.2 Linee guida per la documentazione**

Al fine di offrire una facile leggibilità del codice è stata utilizzata la notazione CamelCase, attribuendo alle variabili e alle classi nomi evocativi rispetto al ruolo che svolgono.

La scrittura delle classi segue una struttura di indentazione, ovvero, dopo un ciclo while, for o if else sono presenti parentesi graffe il new line e un tab

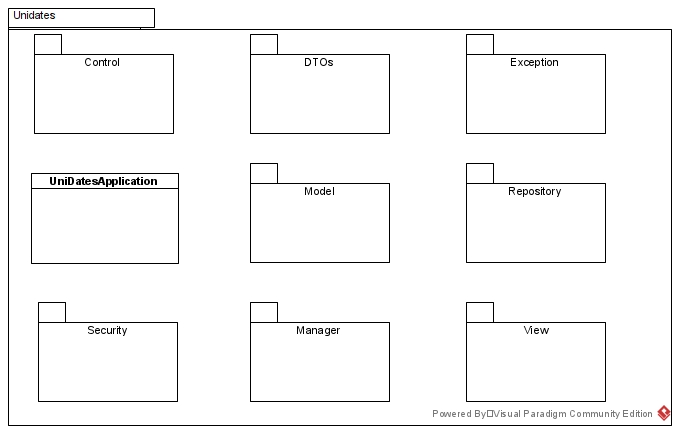
**1.3**  **Componenti off the shelf**

Andremo ad utilizzare due componenti off the shelf:

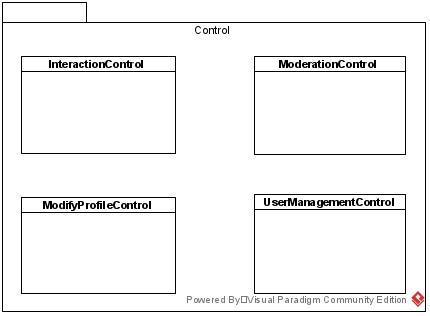
* **Spring Boot**: un framework open source basato su Apache Tomcat e JavaEE che permette di scrivere in modo semplice applicazioni web. Abbiamo scelto di utilizzare questo componente perché rende di facile utilizzo componenti propri di Java EE come JPA, JPQL, Web Services aggiungendo livelli di astrazione e semplificando la configurazione.
* **Vaadin**: un framework open source che consente di creare interfacce web in linguaggio Java. Risulta efficiente utilizzare questo componente perché ci consente di utilizzare lo stesso linguaggio (Java) su tutta l’applicazione, sia front-end che back-end.

1. **Packages**

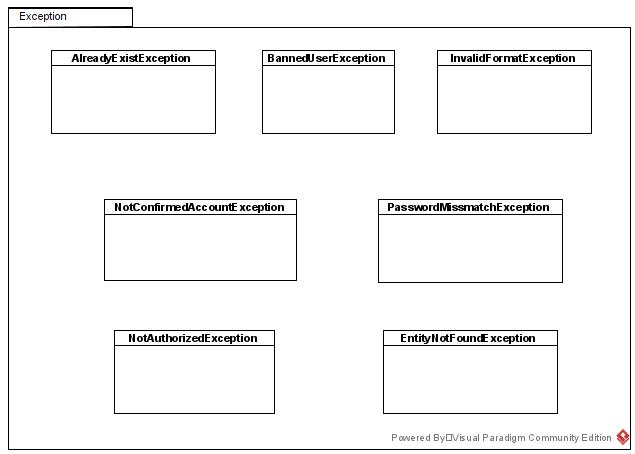
**2.1 Packages Unidates**

****

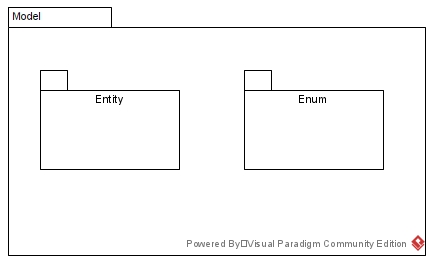
**2.2 Packages Control**

****

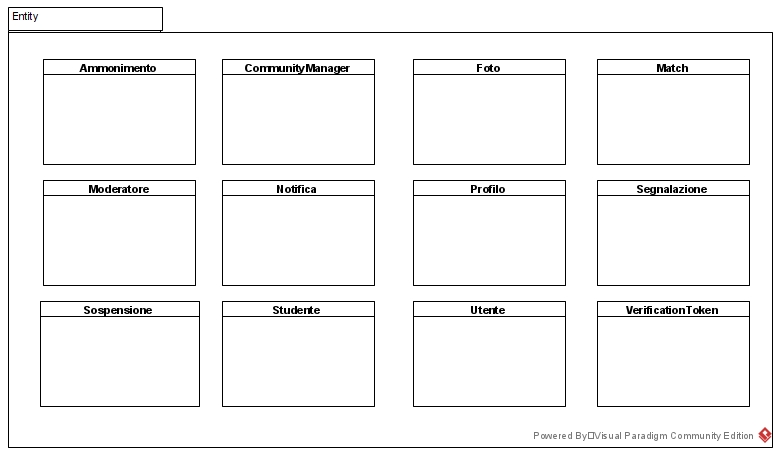
**2.3 Packages Exception**

****

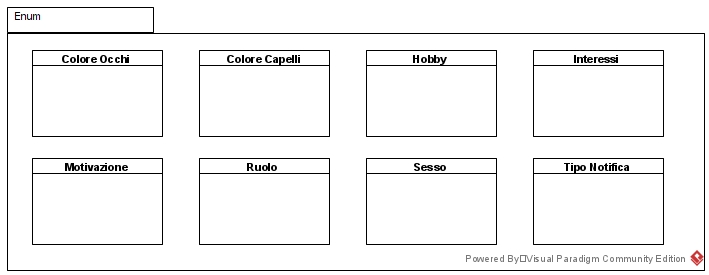
**2.4 Packages Model**

****

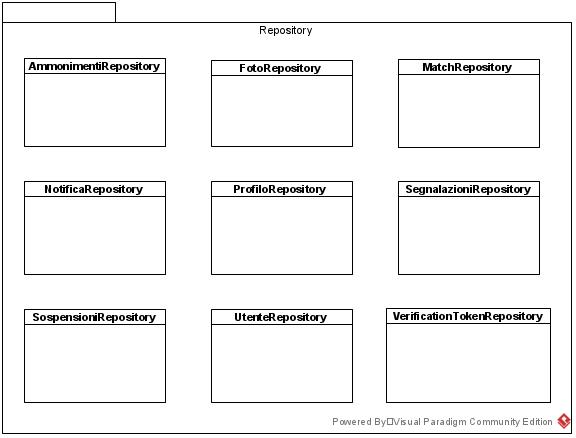
**2.4.2 Packages Entity**

****

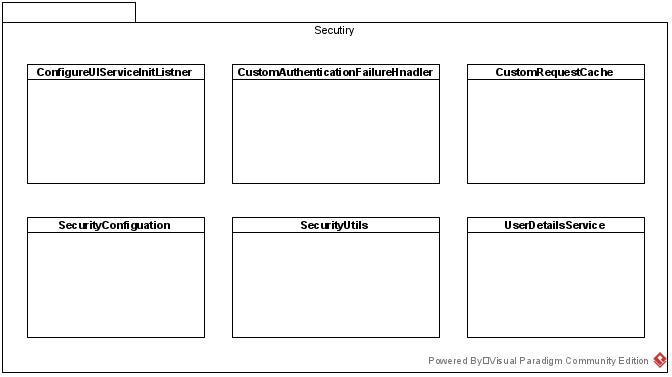
**2.4.3 Package Enum**

****

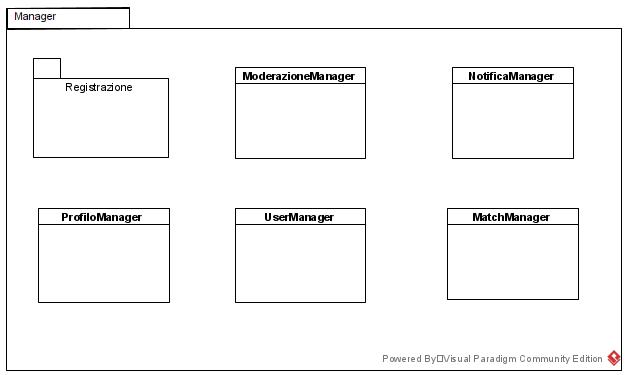
**2.5 Package Repository**

****

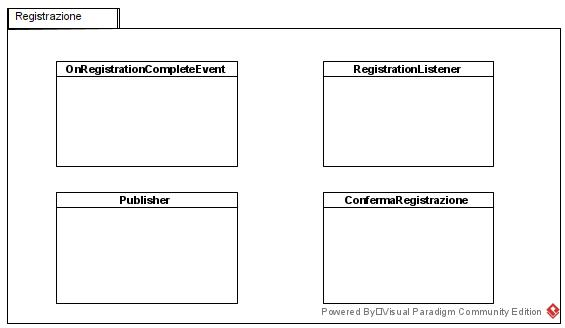
**2.6 Package Security**

****

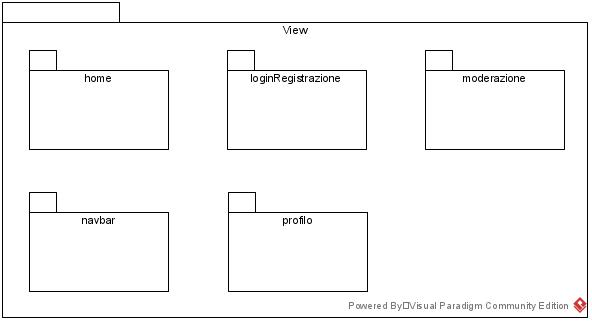
**2.7 Package Manager**

****

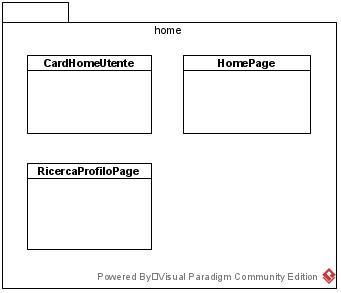
**2.7.1 Package Registrazione**

****

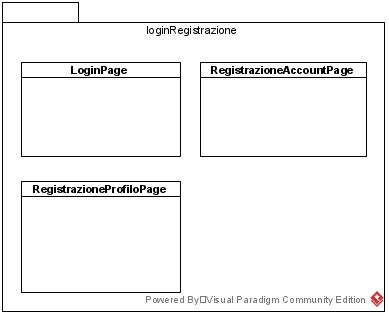
**2.8 Package View**

****

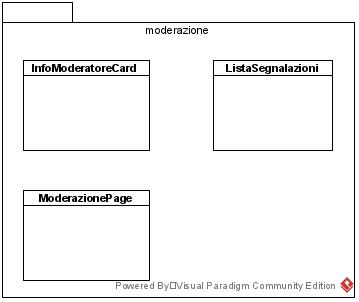
**2.8.1 Package home**

****

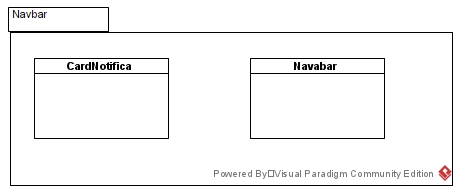
**2.8.2 Package loginRegistrazione**

****

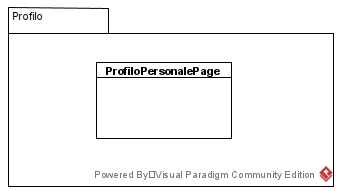
**2.8.3 Package Moderazione**

****

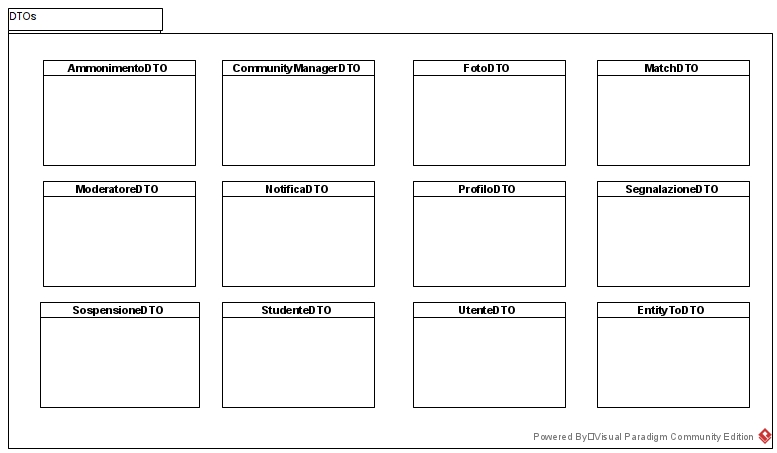
**2.8.4 Package Navbar**

****

**2.8.5 Package Profilo**

****

**2.9 Package DTOs**

**\***Non sarà presente una specifica dell’interfaccia delle classi DTO essendo queste classi POJO utilizzate per il trasferimento dati ed avendo gli stessi attributi e metodi delle classi Entity.

1. **Interfaccia delle classi**

**3.1 Ammoniemento**

|  |
| --- |
| **Ammonimento** |
| * id: Long * studente: Studente * moderatore: Moderatore * foto: Foto * motivazione: Motivazione * dettagli: String |
| * getId(): Long * setId(Long id): void * getStudente(): Studente * setStudente(Studente s): void * getModeratore(): Moderatore * setModeratore(Moderatore m): void * getFoto(): Foto * setFoto(Foto f): void * getMotivazione(): Motivazione * setMotivazione(Motivazione m): void * getDettagli(): String * setDettagli(String d): void |

**3.2 CommunityManager**

|  |
| --- |
| **CommunityManager** |
|  |
|  |

**3.3 Foto**

|  |
| --- |
| **Foto** |
| * id: Long * profilo: Profilo * segnalazioneRicevute: List<Segnalazione> * notifiche: List<Notifica> * isFotoProfilo: boolean * img: byte[] * creazione: LocalDateTime * isVisibile: boolean |
| * getId(): Long * setId(Long id): void * getProfilo(): Profilo * setProfilo(Profilo profilo): void * setImg( byte[] img): void * setSegnalazioniRicevuto(List<Segnalazione> segnalazioneRicevuto): void * getSegnalazioneRicevute(): List<Segnalazione> * setSegnalazioneRicevute(List<Segnalazione> segnalazioneRicevute): void * isVisible() : booelan * setVisible(boolean visibile) : void * getNotifiche(): List<Notifiche> * setNotifiche(List<Notica> notifiche): void * getCreazione(): LocalDateTime * setCreazione(LocalDateTime creazione): void * isFotoProfilo(): boolean * setFotoProfilo(boolean fotoProfilo): void |

**3.4 Profilo**

|  |
| --- |
| **Profilo** |
| * id: long * studente: Studente * listaFoto: List<Foto> * nome: String * cognome: String * luogoNascita: String * residenza: String * dataDiNascita: LocalDate * altezza: double * sesso: Sesso * interessi: Interessi * colori\_capelli: Colore\_capelli * colore\_occhi: Colore\_occhi * numeroTelefono: String * nickInstagram: String * hobbyList: List<Hobby> |
| * getId(): Long * getStudente(): Studente * getListaFoto(): List<Foto> * getNome(): String * getCognome(): String * getLuogoNascita: String * getResidenza(): String * getDataDiNascita(): LocalData * getAltezza(): Double * getSesso(): Sesso * getInteressi(): Interessi * getColore\_Occhi(): Colore\_Occhi * getColore\_Capelli(): Colore\_Capelli * getNumeroTelefono(): String * getNickInstagram(): String * getHobbyList(): List<Hobby> * addFoto(Foto foto, boolean isProfile): void * removeFoto(Foto foto): void * setId(Long id): void * setStudente(Studente studente): void * setListaFoto(List<Foto> listaFoto): void * setNome(Sting nome): void * setCognome(String cognome): void * setLuogoNascita(String luogoNascita): void * setResidenza(String residenza): void * setDataDiNascita(String dataDiNascita): void * setAltezza(double altezza): void * setSesso(Sesso sesso): void * setInteressi(Interessi interessi): void * setColori\_capelli(Colori\_capelli colori\_capelli): void * setColore\_occhi(Colore\_occhi colore\_occhi): void * setNumeroTelefono(String numeroTelefono):void * setNickInstagram(String nickInstagram):void * setHobbyList(List<Hobby> hobbyList): void |

**3.5 Match**

|  |
| --- |
| **Match** |
| * id: Long * studente1: Studente * studente2: Studente * likedByStudent1: boolean * likeByStudent2: boolean |
| * getId(): Long * setId(Long id): void * getStudente1(): Studente * setStudente1(Studente s1): void * getStudente2(): Studente * setStudente2(Studente s2): void * isLikedByStudent1(): boolean * setLikedByStudent1(boolean liked): void * isLikedByStudent2(): boolean * setLikedByStudent2(boolean liked): void |

**3.6 Moderatore**

|  |
| --- |
| **Moderatore** |
| * ammonimentoInviati : List<Ammonimento> * segnalazioniRicevute : List <Segnalazione> |
| * getAmmonimetoInviati(): List<Amminimento> * setAmmonimentoInviati(List<Ammoniemento ammonimentoInviati>): void * getSegnalazioneRicevute(): List<Segnalazione> * addSegnalazione(Segnalazione segnalazione): void * setSegnalazioneRicevute(List<Segnalazione> segnalazioneRicevute): List<Segnalazione> |

**3.7 Notifica**

|  |
| --- |
| **Notifica** |
| * id: Long * utente: Utente * emailToMatchWith: String * foto: Foto * testoNotifica: String * tipo\_notifica: Tipo\_Notifica * ammonimento: Ammonimento * creationTime: LocalDate |
| * getId(): Long * setId(Long id): void * getUtente(): Utente * setUtente(Utente u): void * getEmailToMatchWith(): String * setEmailToMatchWith(String e): void * getFoto(): Foto * setFoto(Foto f): void * getTestoNotifica(): String * setTestoNotifica(String t): void * getTipo\_notifica(): String * setTipo\_notifica(Tipo\_Notifica tn): void * getAmmonimento(): Ammonimento * setAmmonimento(Ammonimento a): void * getCreationTime(): LocalDate * setCreationTime(LocalDate ld): void |

**3.8 Segnalazione**

|  |
| --- |
| **Segnalazione** |
| * id: Long * foto : Foto * moderatore : Moderatore * motivazione : Motivazione * dettagli: String |
| * setModeratore(Moderatore moderatore) * getModeratore(): Moderatore * getId() : Long * setId(Long id) : void * getFoto(): Foto * setFoto(Foto foto): void * getMotivazione() : Motivazione * setMotivazione(Motivazione motivazione): void * getDettagli(): String * setDettagli(String dettagli) : void |

**3.9 Sospensione**

|  |
| --- |
| **Sospensione** |
| * id: Long * studente: Studente * durata: int * dettagli: String |
| * getId(): Long * setId(Long id): void * getStudente(): Studente * setStudente(Studente studente): void * getDurata(): int * setDurata(int durata): void * getDettagli(): String * setDettagli(String dettagli): void |

**3.10 Studente**

|  |
| --- |
| **Studente** |
| * isBanned: boolean * listaBloccati: List<Studente> * listaSospensioni: List<Sospensione> * listaAmmonimenti: List<Ammonimento> * listaMatch: List<Match> * listaMatchRicevuti: List<Match> * listaNotifica: List<Notifica> * profilo: Profilo * ammonimentiAttivi: int |
| * isBanned(): boolean * setBanned(boolean b): void * getProfilo(): Profilo * setProfilo(Profilo p): void * getListaBloccati(): List<Studente> * setListaBlocati(List<Studente> lb): void * getListaSospensioni(): List<Sospensione> * setListaSospensioni(List<Sospensione> ls): void * getListaAmmonimenti(): List<Ammonimento> * setListaAmmonimenti(List<Ammonimento> la): void * getListaMatch(): List<Match> * setListaMatch(List<Match> lm): void * getListaMatchRicevuti(): List<Match> * setListaMatchRicevuti(List<Match> lmr): void * addBloccato(Studente s): void * removeBloccato(Studente s): void * addAmmonimentoattivo(): void * removeAmmonimentoattivo(): void * resetAmmonimentoattivi(): void * getAmmonimentiAttivi(): int * getListaNotifica(): List<Notifica> * setListaNotifica(List<Notifica> listaNotifica): void |

**3.11 Utente**

|  |
| --- |
| **Utente** |
| * email :String * password : String * ruolo : Ruolo * isActive : boolean * verificationTocken: VerificationTocken |
| * getEmail(): String * setEmail(String emial): void * getPassword() : String * setPassword(String password): void * getRuolo(): Ruolo * setRuolo(Ruolo ruolo): void * isActive(): booelan * setActive(boolean active): void |

**3.12 VerificationToken**

|  |
| --- |
| **VerificationToken** |
| * id: Long * token: Stirng * utente: Utente * expiryDate: Date * calculateExpiryDate: Date |
| * getEXPIRATION(): int * getId(): Long * setId(Long id): void * getToken(): String * setToken(String token): void * getUtente(): Utente * setUtente(Utente utente): void * getExpiryDate(): Date * setExpiryDate(Date expiryDate): void |

**3.13 UserManager**

|  |
| --- |
| **Nome classe**: UserManager/UserService |
| **Descrizione**: Classe composta da metodi che permettono di: creare nuovi utenti, modificarne le informazioni e recuperarle |
| **Metodi:**   * + regiastrazioneStudente (Studente s, Profilo p) * + registrazioneModeratore (Moderatore m, Profilo p) * + registrazioneCommunityManager(CommunityManager cm, Profilo p) (x) * + trovaUtente(String email) * + trovaStudente(String email) * + bloccaStudenete(String emailStudenteBloccante, String emailStudenteBloccato) * + sbloccaStudente(String emailStudenteSbloccante, String emailStudenteSbloccato) * + isPresent(String email) * + getUtenteByVerificationToken(String verificationToken) * + getVerificationToken(String token) * + createVerificationToken(Utente utente, String token) * + deleteUtente(Utente utente) * + salvaUtenteRegistrato(Utente utente) * + findAllStudenti() * + cambiPassword(String emailUtente, String nuovaPassword) |

|  |  |
| --- | --- |
| **Nome Metodo:** + registrazioneStudente(Studente s, Profilo p): Studente | |
| **Descrizione metodo:** Metodo che permette di rendere un utente non attivo persistente nel sistema, criptandone la password. | |
| **PreCondizione** | **Context:** UserManager::registrazioneStudente(Studente s, Profilo p).  **Pre:** isPresent(s.getEmail) == false |
| **PostCondizione** | **Context:** UserManager:: registrazioneStudente(Studente s , Profilo p).  **Post:** lo studente risulta registrato, non attivo e con password criptata. |
| **Exception** | Se lo studente tenta di registrarsi con un email già presente (isPresente(s.getEmail()) == true) : throw **AleadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + isPresent(String email): Boolean | |
| **Descrizione metodo:** Il metodo ricerca l’utente che ha l’email che corrisponde al email passata. | |
| **PreCondizione** | **Context:** UserManager::isPresent(String email)  **Pre:** l’email ricercata risulta essere: email != null |
| **PostCondizione** | **Context:** UserManager::isPresent(String email)  **Post:** Se l’email è associata ad un utente ritorna true. |

|  |  |
| --- | --- |
| **Nome Metodo:** + registrazioneModeratore(Moderatore m, Profilo p): Moderatore | |
| **Descrizione metodo:** Metodo che permette di rendere persistente un moderatore nel sistema con la password scelta criptata. | |
| **PreCondizione** | **Context:** UserManager::registrazioneModeratore(Moderatore m, Profilo p)  **Pre:** isPresent(m.getEmail) == false |
| **PostCondizione** | **Context:** UserManager::registrazioneModeratore(Moderatore m , Profilo p)  **Post:** Il moderatore risulta registrato ed attivo, con la password criptata. |
| **Exception** | Se il moderatore tenta di registrarsi con un email già presente (isPresente(m.getEmail()) == true) : throw **AleadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + registrazioneCommunityManager(CommunityManager cm, Profilo p): CoomunityManager | |
| **Descrizione metodo:** consente ad un community manager di registrarsi al sistema | |
| **PreCondizione** | **Context:** UserManager::registrazioneCommunityManager(CommunityManager cm , Profilo p)  **Pre:** isPresent(cm.getEmail) == false |
| **PostCondizione** | **Context:** UserManager::registrazioneCommunityManager(CommunityManager cm , Profilo p)  **Post:** Il community manager attivo, viene reso persistente con la password criptata. |
| **Exception** | Se il community manager tenta di registrarti con un email già presente (isPresente(m.getEmail()) == true) : throw **AleadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + trovaUtente(String email): Utente | |
| **Descrizione metodo:** Dato un email, il metodo verifica se l’utente associato a quell email è presente nel sistema | |
| **PreCondizione** | **Context:** UserManager::trovaUtente(String email)  **Pre:** l’utente a cui è associata l’email deve essere diverso da null |
| **PostCondizione** | **Context:** UserManager::trovaUtente(String email)  **Post:** Se esiste un utente associato a quell’email allora ritorniamo l’utente |
| **Exception** | Se all’email passata non è associato nessun utente: throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + trovaStudente(String email): Studente | |
| **Descrizione metodo:** Dato un email, il metodo verifica se lo studente a cui è associata quell email è presente nel sistema | |
| **PreCondizione** | **Context:** UserManager::trovaStudente(String email)  **Pre:** lo studente a cui è associata l’email deve essere diverso da null |
| **PostCondizione** | **Context:** UserManager::trovaStudente(String email)  **Post:** Se esiste uno studente associato all’email passata ritorna lo studente |
| **Exception** | Se all’email passata non è associato nessuno studente: throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + bloccaStudente(String emailStudenteBloccante, String emailStudenteBloccato): boolean | |
| **Descrizione metodo:** Data l’email dell'utente bloccante (emailStudenteBloccante) e l’email dell'utente bloccato (emailStudenteBloccato) il sistema consente di far bloccare il secondo utente dal primo | |
| **PreCondizione** | **Context:** UserManager::bloccaStudente(String emailStudenteBloccante, String emailStudenteBloccato)  **Pre:** entrambi gli studenti associati alle due email devono essere presenti nel sistema && controlliamo se l’utente da bloccare non sia già stato bloccato dal utente bloccante |
| **PostCondizione** | **Context:** UserManager::bloccaStudente(String emailStudenteBloccante, String emailStudenteBloccato)  **Post:** Se entrambi gli utenti a cui sono associate le email sono presenti nel sistema l’utente emailStudenteBloccante blocca lo studente emailStudenteBloccato e il metodo ritorna **true**. |
| **Exception** | * Se uno dei due utenti risulta non essere presente nel sistema: throw **EntityNotFoundException** * Se l’utente che si vuole bloccare (emailStudenteBloccato) è già stato bloccato dall’utente che blocca (emailStudenteBloccante) : throw **AlreadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + sbloccaStudente(String emailStudenteSbloccante, String emailStudenteSbloccato):boolean | |
| **Descrizione metodo:** il metodo consente all’utente (emailStudenteSbloccante) di sbloccare l’utente (emailStudenteSbloccato) | |
| **PreCondizione** | **Context:** UserManager::(String emailStudenteSbloccante, String emailStudenteSbloccato)  **Pre:** entrambi gli studenti associati alle due email devono essere presenti nel sistema |
| **PostCondizione** | **Context:** UserManager::(String emailStudenteSbloccante, String emailStudenteSbloccato)  **Post:** Se entrambe gli studenti sono presenti del sistema l’utente (emailStudenteSbloccante) può sbloccare l’utente (emailStudenteSbloccato) ed il metodo ritorna **true** |
| **Exception** | * Se uno dei due utenti risulta non essere presente nel sistema: throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + getUtenteByVerificationToken(String verificationToken): Utente | |
| **Descrizione metodo:** ritorna l’utente corrispondente a quel verificationToken | |
| **PreCondizione** | **Context:** UserManager::getUtenteByVerificationToken(String verificationToken)  **Pre:** il token passato in input deve essere presente nel sistema && il token deve essere associato ad un utente. |
| **PostCondizione** | **Context:** UserManager::getUtenteByVerificationToken(String verificationToken)  **Post:** il metodo ritorna l’utente a cui è associato il token passato in input |
| **Exception** | * Se il token risulta non essere presente nel sistema || l’utente associato al token non risulta essere presente nel sistema : throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + getVerificationToken(String Token): VerificationToken | |
| **Descrizione metodo:** il metodo ritorna il verificationToken che è associato al Token | |
| **PreCondizione** | **Context:** UserManager::getVerificationToken(String Token)  **Pre:** il verificationToken deve essere presente nel sistema |
| **PostCondizione** | **Context:** UserManager::getVerificationToken(String Token)  **Post:** il metodo ritorna il VerificationToken associato al Token passato |
| **Exception** | * Se il token risulta non essere presente nel sistema : throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + createVerificationToken(Utente utente, String token): VerificationToken | |
| **Descrizione metodo:** il metodo va a creare un VerificationToken che verrà associato ad un utente ed ad un token. | |
| **PreCondizione** | **Context:** UserManager:: createVerificationToken(Utente utente, String token)  **Pre: //** |
| **PostCondizione** | **Context:** UserManager:: createVerificationToken(Utente utente, String token)  **Post:** il Verification risulta creato e il sistema ritorna il VerificationToken |

|  |  |
| --- | --- |
| **Nome Metodo:** + deleteUtente(String email): Utente | |
| **Descrizione metodo:** consente di eliminare l’utente corrispondente all’email, dal sistema | |
| **PreCondizione** | **Context:** UserManager:: deleteUtente(String email)  **Pre:** l’utente deve essere presente nel sistema |
| **PostCondizione** | **Context:** UserManager:: deleteUtente(String email)  **Post**: l’utente risulta essere eliminato dal sistema e il sistema ritorna l’utente eliminato |
| **Exception** | Se l’utente non risulta essere presente nel sistema : throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + attivaUtenteRegistrato(String email): Utente | |
| **Descrizione metodo:** consente di ativare l’utente corrispondende all’email. | |
| **PreCondizione** | **Context:** UserManager:: salvaUtenteRegistrato(String email)  **Pre:** l’utente corrispondente all’email deve essere presente nel sistema |
| **PostCondizione** | **Context:** UserManager:: salvaUtenteRegistrato(String email)  **Post**: l’utente verrá attivato e il sistema ritornerá l’utente attivato |
| **Exception** | Se l’utente non risulta essere presente nel sistema : throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + findAllStudenti():List<Studente> | |
| **Descrizione metodo:** il metodo ritorna tutti gli utenti registrati al sistema. | |
| **PreCondizione** | **Context:** UserManager::findAllStudenti()  **Pre:** // |
| **PostCondizione** | **Context:** UserManager::findAllStudenti()  **Post**: vengono ritornati tutti gli utenti registrati al sistema. |

|  |  |
| --- | --- |
| **Nome Metodo:** + cambiaPassword(String emailUtente, String nuovaPassword): Utente | |
| **Descrizione metodo:** il metodo consente di cambiare la password ad un utente presente nel sistema | |
| **PreCondizione** | **Context:** UserManager::cambiaPassword(String emailUtente, String nuovaPassword)  **Pre:** l’utente a cui è associata l’email passata (emailUtente) deve essere presente nel sistema |
| **PostCondizione** | **Context:** UserManager:: cambiaPassword(String emailUtente, String nuovaPassword)  **Post**: la password dell'utente risulta essere stata modificata e ritorniamo tale utente. |
| **Exception** | Se l’utente non risulta essere presente nel sistema : throw **EntityNotFoundException** |

**3.14 ProfiloManager**

|  |
| --- |
| **Nome classe:** ProfiloManager |
| **Descrizione:** Classe composta da metodi che permettono di effettuare operazioni sul proprio profilo, aggiungere ed eliminare foto. |
| **Metodi:**   * + modificaProfilo (String email, Profilo p) * + setFotoProfilo (String email, Long idFoto) * + aggiungiFotoLista (String email, Foto f) * + aggiungiFotoProfilo(String email, Foto f) * + eliminaFotoLista (String email, Long idFoto) * + eliminaFotoProfilo (String email) * + findFotoById (Long idFoto) * + findProfiloById (Long idProfilo) |

|  |  |
| --- | --- |
| **Nome Metodo:** + modificaProfilo (String email, Profilo p) : Profilo | |
| **Descrizione metodo:** il metodo modifica il profilo dell’utente corrispondente all’email con i nuovi dati inseriti e tale profilo viene ritornato. | |
| **PreCondizione** | **context:** ProfiloManager::modificaProfilo (String email, Profilo p)  **pre:** Lo studente corrispondente all’email deve essere presente nel database |
| **PostCondizione** | **context:** ProfiloManager::modificaProfilo (String email, Profilo p)  **post:** il profilo ritornato presenta i campi con le informazioni aggiornate. |
| **Eccezioni** | * se non è presente nessuno studente con l’email inserita throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + setFotoProfilo (String email, Long idFoto) : Foto | |
| **Descrizione metodo:** il metodo aggiorna la foto profilo dello studente e la nuova foto viene ritornata | |
| **PreCondizione** | **context:** ProfiloManager::setFotoProfilo (String email, Long idFoto)  **pre:** La foto che si vuole usare come foto profilo deve essere presente nel database |
| **PostCondizione** | **context:** ProfiloManager::setFotoProfilo (String email, Long idFoto)  **post:** la foto profilo è stata aggiornata e viene ritornata |
| **Eccezioni** | * se la foto non è presente nel database throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + aggiungiFotoLista (String email, Foto f) : Foto | |
| **Descrizione metodo:** il metodo aggiunge, e ritorna , una foto alla lista di foto dello studente | |
| **PreCondizione** | **context:** ProfiloManager::aggiungiFotoLista (String email, Foto f)  **pre:** Lo studente corrispondente all’email deve essere presente nel database && la foto non deve essere già presente tra le foto dello studente |
| **PostCondizione** | **context:** ProfiloManager::aggiungiFotoLista (String email, Foto f)  **post:** la foto è stata aggiunta alla lista foto e viene ritornata |
| **Eccezioni** | * se non è presente nessuno studente con l’email inserita throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + aggiungiFotoProfilo(String email, Foto f) : Foto | |
| **Descrizione metodo:** il metodo aggiorna , e ritorna , la foto profilo dello studente | |
| **PreCondizione** | **context:** ProfiloManager::aggiungiFotoProfilo(String email, Foto f)  **pre:** Lo studente corrispondente all’email deve essere presente nel database && la foto non deve essere già presente tra le foto dello studente |
| **PostCondizione** | **context:** ProfiloManager::aggiungiFotoProfilo(String email, Foto f)  **post:** la foto profilo è stata aggiornata e viene ritornata |
| **Eccezioni** | se la foto è già presente tra le foto dello studente throw **AlreadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + eliminaFotoLista (String email, Long idFoto) : Foto | |
| **Descrizione metodo:** il metodo elimina, e ritorna, una foto dalla lista delle foto dello studente | |
| **PreCondizione** | **context:** ProfiloManager::eliminaFotoLista (String email, Long idFoto)  **pre:** Lo studente corrispondente all’email deve essere presente nel database && la foto deve essere presente nella lista foto dello studente |
| **PostCondizione** | **context:** ProfiloManager::eliminaFotoLista (String email, Long idFoto)  **post:** la foto è stata eliminata e viene ritornata |
| **Eccezioni** | * se non è presente nessuno studente con l’email inserita throw **EntityNotFoundException** * se la foto non è presente nella lista delle foto dello studente throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + eliminaFotoProfilo (String email, Long idFoto) : Studente | |
| **Descrizione metodo:** il metodo elimina la foto profilo dello studente e ritorna tale studente | |
| **PreCondizione** | **context:** ProfiloManager::eliminaFotoProfilo (String email, Long idFoto)  **pre:** Lo studente corrispondente all’email deve essere presente nel database && lo studente deve avere una foto profilo |
| **PostCondizione** | **context:** ProfiloManager::eliminaFotoProfilo (String email, Long idFoto)  **post:** la foto è stata eliminata e ritorniamo lo studente a cui è stata eliminata. |
| **Eccezioni** | * se non è presente nessuno studente con l’email inserita throw **EntityNotFoundException** * se lo studente non ha una foto profilo throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + findFotoById (Long idFoto) : Foto | |
| **Descrizione metodo:** il metodo restituisce la foto corrispondente all’id passato | |
| **PreCondizione** | **context:** ProfiloManager::findFotoById (Long idFoto)  **pre:** la foto corrispondente all’id deve essere presente nel database |
| **PostCondizione** | **context:** ProfiloManager::findFotoById (Long idFoto)  **post:** la foto è stata trovata e viene ritornata. |
| **Eccezioni** | * se non è presente nessuna foto con l’id inserito throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + findProfiloById (Long idProfilo) : Profilo | |
| **Descrizione metodo:** il metodo restituisce il profilo corrispondente all’id | |
| **PreCondizione** | **context:** ProfiloManager::findProfiloById (Long idProfilo)  **pre:** il profilo corrispondente all’id deve essere presente nel database |
| **PostCondizione** | **context:** ProfiloManager::findProfiloById (Long idProfilo)  **post:** il profilo è stato trovato e viene ritornato. |
| **Eccezioni** | * se non è presente nessun profilo con l’id inserito throw **EntityNotFoundException** |

**3.15 MatchManager**

|  |
| --- |
| **Nome classe:** MatchManager |
| **Descrizione:** Classe composta da metodi che permettono di gestire i match tra gli studenti |
| **Metodi:**   * + aggiungiMatch(String emailStudente1, String emailStudente2) * + trovaMatch(String emailStudente1, String emailStudente2) * + isvalidMatch(String emailStudente1, String emailStudente2) * + eliminaMatch(String emailStudente1, String emailStudente2) |

|  |  |
| --- | --- |
| **Nome Metodo:** + aggiungiMatch(String emailStudente1, String emailStudente2) : boolen | |
| **Descrizione metodo:** il metodo crea un match tra i due studenti che si sono scambiati il mi piace | |
| **PreCondizione** | **context:** MatchManager::aggiungiMatch(String emailStudente1, String emailStudente2)  **pre:** gli studenti corrispondenti alle email devono essere presenti nel database |
| **PostCondizione** | **context:** MatchManager::aggiungiMatch(String emailStudente1, String emailStudente2)  **post:** il match tra i due utenti viene creato e il metodo ritorna true. |
| **Eccezioni** | * se uno o più studenti non sono presenti nel database throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + trovaMatch(String emailStudente1, String emailStudente2) : Match | |
| **Descrizione metodo:** il metodo trova un Match tra due studenti se esiste | |
| **PreCondizione** | **context:** MatchManager::trovaMatch(String emailStudente1, String emailStudente2)  **pre:** gli studenti corrispondenti alle email devono essere presenti nel database && il match tra i due studenti deve essere presente nel database |
| **PostCondizione** | **context:** MatchManager::trovaMatch(String emailStudente1, String emailStudente2)  **post:** viene trovato e ritornato il match corrispondente ai due studenti |
| **Eccezioni** | * se uno o più studenti non sono presenti nel database **||** se il match tra i due studenti non esiste throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + isValidMatch(String emailStudente1, String emailStudente2) : boolean | |
| **Descrizione metodo:** il metodo controlla se un Match tra due studenti è valido, cioè se tutt'e due gli studenti hanno ricambiato il mi piace. | |
| **PreCondizione** | **context:** MatchManager::isvalidMatch(String emailStudente1, String emailStudente2)  **pre:** gli studenti corrispondenti alle email devono essere presenti nel database |
| **PostCondizione** | **context:** MatchManager::isvalidMatch(String emailStudente1, String emailStudente2)  **post:** return true se il match è valido , false altrimenti |
| **Eccezioni** | * se uno o più studenti non sono presenti nel database **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + eliminaMatch(String emailStudente1, String emailStudente2) : boolean | |
| **Descrizione metodo:** il metodo permette di cancellare un metodo tra due studenti se questo esiste | |
| **PreCondizione** | **context:** MatchManager::eliminaMatch(String emailStudente1, String emailStudente2)  **pre:** gli studenti corrispondenti alle email devono essere presenti nel database |
| **PostCondizione** | **context:** MatchManager::eliminaMatch(String emailStudente1, String emailStudente2)  **post:** il match risulta eliminato ed il metodo ritorna true |
| **Eccezioni** | * se uno o più studenti non sono presenti nel database throw **EntityNotFoundException** |

**3.16 NotificaManager**

|  |
| --- |
| **Nome classe:** NotificaManager |
| **Descrizione:** Classe composta da metodi che permettono di gestire le notifiche |
| **Metodi:**   * + generateNotificaMatch (String emailStudente1, String emailStudente2) * + generateNotificaWarning (String emailStudente1, String emailStudente2) * + eliminaNotificaMatch (String emailBloccante, String emailBloccato) |

|  |  |
| --- | --- |
| **Nome Metodo:** + generateNotificaMatch (String emailStudente1, String emailStudente2) : void | |
| **Descrizione metodo:** Il metodo aggiunge le notifiche di match ai due studenti | |
| **PreCondizione** | **context:** NotificaManager::generateNotificaMatch (String emailStudente1, String emailStudente2)  **pre:** gli studenti corrispondenti alle email devono essere presenti nel database |
| **PostCondizione** | **context:** NotificaManager::generateNotificaMatch (String emailStudente1, String emailStudente2)  **post:** le notifiche di match sono state aggiunte a gli studenti |
| **Eccezioni** | * se uno o più studenti non sono presenti nel database throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + generateNotificaWarning (String emailStudente1, String emailStudente2) : void | |
| **Descrizione metodo:** Il metodo aggiunge la notifica di ammonimento allo studente | |
| **PreCondizione** | **context:** NotificaManager::generateNotificaWarning (String emailStudente1, String emailStudente2)  **pre:** lo studente corrispondente all’email deve essere presente nel database && la foto oscurata deve essere presente nel database && la foto oscurata deve essere stata ammonita |
| **PostCondizione** | **context:** NotificaManager::generateNotificaWarning (String emailStudente1, String emailStudente2)  **post:** la notifica di ammonimento è stata aggiunta allo studente |
| **Eccezioni** | * se lo studente non è presente nel database || se la foto non ha un ammonimento ||se la foto non è presente nel database throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + eliminaNotificaMatch (String emailBloccante, String emailBloccato) : void | |
| **Descrizione metodo:** Il metodo elimina la notifica di match se un utente è stato bloccato | |
| **PreCondizione** | **context:** NotificaManager::eliminaNotificaMatch (String emailBloccante, String emailBloccato)  **pre:** lo studente corrispondente all’email deve essere presente nel database && il match tra i due studenti deve essere presente nel database |
| **PostCondizione** | **context:** eliminaNotificaMatch (String emailBloccante, String emailBloccato)  **post:** la notifica di match è stata eliminata |
| **Eccezioni** | * se uno o più studenti non sono presenti nel database ||se il match non è presente nel database throw **EntityNotFoundException** |

**3.17 ModerazioneManager**

|  |
| --- |
| **Nome classe:** ModerazioneManager |
| **Descrizione:** Classe composta da metodi che permettono di gestire segnalazioni, ammonimenti e sospensioni. |
| **Metodi:**   * + inviaSegnalazione (Segnalazione s, Long idFoto) * + inviaSegnalazioneCommunityManager (Segnalazione s, Long idFoto) * + inviaAmmonimento (Ammonimento a, String emailModeratore, String emailStudenteAmmonito) * + inviaSospensione(Sospensione sp, String emailSospeso) * + nascondiFoto (Long idFoto) * + checkAmmonimentiStudente (String emailStudente) |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaSegnalazione (Segnalazione s, Long idFoto) : Segnalazione | |
| **Descrizione metodo:** Il metodo permette di segnalare la foto di un altro studente | |
| **PreCondizione** | **context:** ModerazioneManager::inviaSegnalazione (Segnalazione s, Long idFoto)  **pre:** la foto corrispondente all’id deve essere presente nel database && la foto non deve avere già una segnalazione |
| **PostCondizione** | **context:** ModerazioneManager::inviaSegnalazione (Segnalazione s, Long idFoto)  **post:** La segnalazione è stata aggiunta alla foto |
| **Eccezioni** | * se la foto non è presente nel database throw **EntityNotFoundException** * se la foto ha già una segnalazione throw **AlreadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaSegnalazioneCommunityManager (Segnalazione s, Long idFoto) : Segnalazione | |
| **Descrizione metodo:** Il metodo permette di inviare una segnalazione dal moderatore al community manager | |
| **PreCondizione** | **context:** ModerazioneManager::inviaSegnalazioneCommunityManager (Segnalazione s, Long idFoto)  **pre:** la foto corrispondente all’id deve essere presente nel database && la foto non deve avere già una segnalazione verso il community manager |
| **PostCondizione** | **context:** ModerazioneManager::inviaSegnalazioneCommunityManager (Segnalazione s, Long idFoto)  **post:** La segnalazione è stata aggiunta alla foto |
| **Eccezioni** | * se la foto non è presente nel database throw **EntityNotFoundException** * se la foto ha già una segnalazione throw **AlreadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaAmmonimento (Ammonimento a, String emailModeratore, String emailStudenteAmmonito) : Ammonimento | |
| **Descrizione metodo:** Il metodo permette di inviare un ammonimento a uno studente | |
| **PreCondizione** | **context:** ModerazioneManager::inviaAmmonimento (Ammonimento a, String emailModeratore, String emailStudenteAmmonito)  **pre:** lo studente corrispondente all’email deve essere presente nel database && l’ammonimento non deve essere già stato inviato allo studente |
| **PostCondizione** | **context:** ModerazioneService::inviaAmmonimento (Ammonimento a, String emailModeratore, String emailStudenteAmmonito)  **post:** L’ammonimento è stato inviato allo studente |
| **Eccezioni** | * se lo studente non è presente nel database throw **EntityNotFoundException** * se lo studente ha già ricevuto l’ammonimento throw **AlreadyExistException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaSospensione(Sospensione sp, String emailSospeso) : Sospensione | |
| **Descrizione metodo:** Il metodo permette di inviare una sospensione a uno studente | |
| **PreCondizione** | **context:** ModerazioneManager::inviaSospensione(Sospensione sp, String emailSospeso)  **pre:** lo studente corrispondente all’email deve essere presente nel database |
| **PostCondizione** | **context:** ModerazioneManager::inviaSospensione(Sospensione sp, String emailSospeso)  **post:** Lo studente è stato sospeso |
| **Eccezioni** | * se lo studente non è presente nel database throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + nascondiFoto (Long idFoto) : boolean | |
| **Descrizione metodo:** il metodo permette di nascondere la foto di uno studente | |
| **PreCondizione** | **context:** ModerazioneManager::nascondiFoto (Long idFoto)  **pre:** la foto corrispondente all’id deve essere presente nel database |
| **PostCondizione** | **context:** ModerazioneManager::nascondiFoto (Long idFoto)  **post:** ritorna true se la foto è stata nascosta |
| **Eccezioni** | * se la foto non è presente nel database throw **EntityNotFoundException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + checkAmmonimentiStudente (String emailStudente) : boolean | |
| **Descrizione metodo:** Il metodo permette di sospendere uno studente se ha ricevuto 3 ammonimenti | |
| **PreCondizione** | **context:** ModerazioneManager::checkAmmonimentiStudente (String emailStudente)  **pre:** lo studente corrispondente all’email deve essere presente nel database |
| **PostCondizione** | **context:** ModerazioneManager::checkAmmonimentiStudente (String emailStudente)  **post:** restituisce true se l’utente è stato sospeso |
| **Eccezioni** | * se lo studente non è presente nel database throw **EntityNotFoundException** |

**3.18 UserManagementControl**

|  |
| --- |
| **Nome classe:** UserManagementControl |
| **Descrizione:** Controller che si occupa di gestire tutte le richieste del pacchetto “Gestione Utenti” |
| **Metodi:**   * + registrazioneStudente (StudenteDTO s, HttpServletRequest r) * + isAlreadyRegistered (String email) * + studenteInSessione () * + confermaRegistrazione (String token) * + trovaTuttiStudenti () * - checkEmail (String email) * - checkProfilo (ProfiloDTO p) * - checkStudente (StudenteDTO s) |

|  |  |
| --- | --- |
| **Nome Metodo:** + registrazioneStudente (StudenteDTO s, HttpServletRequest r) : Void | |
| **Descrizione metodo:** Il metodo permette di registrare uno studente | |
| **PreCondizione** | **Contex:** UserManagementControl::registrazioneStudente(StudenteDTO s, HttpServletRequest r)  **Pre:** (checkStudente(s) == true) && (checkProfilo(p) == true) |
| **PostCondizione** | **Contex:** UserManagementControl::registrazioneStudente(StudenteDTO s, HttpServletRequest r)  **Post:** Lo studente s risulta registrato con il profilo p |
| **Eccezioni** | * se uno o più campi del profilo non rispettano il formato throw **InvalidFormatException** * se email e/o password non rispettano il formato throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + isAlreadyRegistered (String email) : boolean | |
| **Descrizione metodo:** Il metodo controlla se l’email è già associata ad uno studente | |
| **PreCondizione** | **Contex:** UserManagementControl::isAlreadyRegistered (String email)  **Pre:** checkEmail(email) == true |
| **PostCondizione** | **Contex:** UserManagementControl::isAlreadyRegistered (String email)  **Post:** ritorna true se l’email è già associata ad un account, false altrimenti |
| **Eccezioni** | * se l’email non rispetta il formato throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + studenteInSessione(): studenteDTO | |
| **Descrizione metodo:** Il metodo restituisce lo studente in sessione | |
| **PreCondizione** | **Contex:** UserManagementControl::studenteInSessione(): studenteDTO  **Pre:** // |
| **PostCondizione** | **Contex:** UserManagementControl::studenteInSessione(): studenteDTO  **Post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** + confermaRegistrazione(String token): String | |
| **Descrizione metodo:** Il metodo attiva l’account dell’utente | |
| **PreCondizione** | **Contex:** UserManagementControl::confermaRegistrazione(String token)  **Pre:** token != null && il token non è scaduto |
| **PostCondizione** | **Contex:** UserManagementControl::confermaRegistrazione(String token)  **Post:** l’utente viene attivato |

|  |  |
| --- | --- |
| **Nome Metodo:** + trovaTuttiStudenti(): List<StudenteDTO> | |
| **Descrizione metodo:** Metodo che restituisce la lista di tutti gli studenti presenti nel database | |
| **PreCondizione** | **Contex:** UserManagementControl::trovaTuttiStudenti(): List<StudenteDTO>  **Pre:** // |
| **PostCondizione** | **Contex:** UserManagementControl::trovaTuttiStudenti(): List<StudenteDTO>  **Post:** // |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkEmail(String email): boolean | |
| **Descrizione metodo:** Metodo che controlla la correttezza del formato di un email | |
| **PreCondizione** | **Contex:** UserManagementControl::checkEmail(String email)  **Pre:** email != null && email rispetta il formato della regex |
| **PostCondizione** | **Contex:** UserManagementControl::checkEmail(String email)  **Post:** Se viene rispettata la correttezza del formato ritorna true, altrimenti false |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkProfilo(ProfiloDTO p): boolean | |
| **Descrizione metodo:** Metodo che controlla la correttezza del formato dei campi del profilo | |
| **PreCondizione** | **Contex:** UserManagementControl::checkProfilo(ProfiloDTO p)  **Pre:** i valori associati al profilo quali: nome,cognome,colore capelli, colore occhi,altezza,sesso, data si nascita, luogo di nascita, residenza, interessi,lista topic devono essere diversi da null. |
| **PostCondizione** | **Contex:** UserManagementControl::checkProfilo(ProfiloDTO p)  **Post:** Se viene rispettata la correttezza del formato ritorna true, altrimenti false |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkStudente(StudenteDTO s): boolean | |
| **Descrizione metodo:** Metodo che controlla la correttezza del formato dei campi dello studente | |
| **PreCondizione** | **Contex:** UserManagementControl::checkStudente(StudenteDTO s)  **Pre:** (s.getEmail != null) && (s.getPassword != null) && (checkEmail(s.getEmail()) == true) |
| **PostCondizione** | **Contex:** UserManagementControl::checkStudente(StudenteDTO s)  **Post:** Se viene rispettata la correttezza del formato ritorna true, altrimenti false |

**3.19 InteractionControl**

|  |
| --- |
| **Nome classe:** InteractionControl |
| **Descrizione:** Controller che si occupa di gestire tutte le richieste del pacchetto “Interazioni Utenti” |
| **Metodi:**   * + aggiungiMatch (String email1, String email2) * + isValidMatch(String email1, String email2) * + bloccaStudente(String emailBloccante, String emailBloccato) * +sbloccaStudente(String emailSbloccante, String emialSbloccato) * +ricercaStudente(String email) * - checkEmail (String email) |

|  |  |
| --- | --- |
| **Nome Metodo:** +aggiungiMatch (String emailStudente1, String emailStudente2) : boolean | |
| **Descrizione metodo:** il metodo permette di aggiungere un match tra lo studente con emailStudente1 e lo studente con emailStudente2. | |
| **PreCondizione** | **Contex:** InteractionControl::aggiungiMatch(String emailStudente1, String emailStudente2)  **Pre:**  ((checkEmail(emailStudente1) == true) && (checkEmail(emailStudente2) == true)) && lo studente è loggato && (emailStudente1 != emailStudente2) && il match è valido da entrambe le parti |
| **PostCondizione** | **Contex:** InteractioControl::aggiungiMatch (String emailStudente1, String emailStudente2)  **Post:** Ritorna true se il match è avvenuto, false altrimenti |
| **Exception** | Se emailStudente1 == emailStudente2 || l’utente loggato non è l’utente con emailStudente1 : throw **NotAuthorizedException**  Se emailStudente1 e/o emailStudente2 non rispettano il formato : throw **InvalidFormartException** |

|  |  |
| --- | --- |
| **Nome Metodo:** +isValidMatch (String emailStudente1, String emailStudente2) : boolean | |
| **Descrizione metodo:** il metodo consente di verificare se esiste un Match tra gli utenti con le email passate come parametri. | |
| **PreCondizione** | **Contex:** InteractionControl::isValidMatch(String emailStudente1, String emailStudente2)  **Pre:** (checkEmal(emailStudente1) == true) && (checkEmail(emailStudente2) == true) |
| **PostCondizione** | **Contex:** InteractionControl::isValidMatch (String emailStudente1, String emailStudente2)  **Post:** ritorna true se il match esiste, false altrimenti |
| **Exception** | Se emailStudente1 e/o emailStudente2 non rispettano il formato : throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** +bloccaStudente (String emailBloccante, String emailBloccato) : boolean | |
| **Descrizione metodo:** il metodo permette di far bloccare l’utente con emailBloccato dall’utente con emailBloccante | |
| **PreCondizione** | **Contex:** InteractionControl::aggiungiMatch(String emailBloccante, String emailBloccato)  **Pre:** (checkEmail(emailBloccante) == true) && (checkEmail(emailBloccato) == true) && l’utente con emailBloccante deve essere l’utente loggato && emailBloccante != emailBloccato |
| **PostCondizione** | **Contex:** InteractioControl::aggiungiMatch (String emailBloccante, String emailBloccato)  **Post:** ritorna true sel’utente con emailBloccato risulta essere stato bloccato dall’utente con emailBloccante, false altrimenti |
| **Exception** | Se l’utente con emailBloccante non è quello loggato || emailBloccante == emailBloccato : throw **NotAuthorizedException**  se emailBloccante e/o emailBloccato non rispettano il formato throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** +sbloccaStudente (String emailSbloccante, String emailSbloccato) : boolean | |
| **Descrizione metodo:** il metodo permette di far sbloccare l’utente con emailSbloccato dall’utente con emailSbloccante | |
| **PreCondizione** | **Contex:** InteractionControl::aggiungiMatch(String emailSbloccante, String emailSbloccato)  **Pre:** (checkEmail(emailSbloccante) == true) && (checkEmail(emailSbloccato) == true) && l’utente con emailSbloccante è l’utente loggato && emailSbloccante != emailSbloccato |
| **PostCondizione** | **Contex:** InteractioControl::aggiungiMatch (String emailSbloccante, String emailSbloccato)  **Post:** ritorna true se l’utente con emailSboccato risulta essere stato sbloccato dall’utente con emailSbloccante, false altrimenti |
| **Exception** | Se l’utente con emailSbloccante non è quello loggato || emailSbloccante == emailSbloccato : throw **NotAuthorizedException**  se emailSbloccante e/o emailSbloccato non rispettano il formato throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** +ricercaStudente (String emailStudente) : StudenteDTO | |
| **Descrizione metodo:** il metodo permette di ricercare l’utente con emailStudente | |
| **PreCondizione** | **Contex:** InteractionControl::ricercaStudente (String emailStudente)  **Pre:**chekEmail(emailStudente) == true |
| **PostCondizione** | **Contex:** InteractioControl:: ricercaStudente (String emailStudente)  **Post:** ritorna una StudenteDTO |
| **Exception** | Se checkEmail(emailStudente) == false : throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkEmail(String email): boolean | |
| **Descrizione metodo:** Metodo che controlla la correttezza del formato di un email | |
| **PreCondizione** | **Contex:** InteractionControl::checkEmail(String email)  **Pre:** email != null && email rispetta il formato della regex |
| **PostCondizione** | **Contex:** InteractionControl::checkEmail(String email)  **Post:** Se viene rispettata la correttezza del formato ritorna true, altrimenti false |

**3.20 ModifyProfileControl**

|  |
| --- |
| **Nome classe: ModifyProfileControl** |
| **Descrizione:** Controller che si occupa di gestire tutte le richieste del pacchetto “Gestione Profilo” |
| **Metodi:**   * + aggiungiFotoLista (String emailFotoToAdd, FotoDTO fotoDTO) * + eliminaFotoLista (Lond idFoto f, String email) * + aggiungiFotoProfilo(String emailFotoToAdd, FotoDTO fotoDTO ) * + setFotoProfilo(String emailStudenteToModify, Long fotoId) * + modificaProfilo(String emailStudenteToModify, ProfiloDTO profiloDTO) * + cambiaPassword(String emailUtente, String nuovaPassword, String vecchiaPassword) * + cancellaAccountPersonale(String email, String password) * + trovaFoto(Long fotoId) * + trovaProfilo(Long profiloId) * - checkFoto(FotDTO f) * - checkProfilo(Profilo p) * - checkPassword(String password) * - checkEmail(String email) |

|  |  |
| --- | --- |
| **Nome Metodo:** + aggiungiFotoLista (String emailFotoToAdd, FotoDTO fotoDTO): FotoDTO | |
| **Descrizione metodo:** il metodo permette di aggiungere una foto al profilo personale. | |
| **PreCondizione** | **Contex:** ModifyProfileControl::aggiungiFotoLista(String emailFotoToAdd, FotoDTO fotoDTO)  **Pre:** checkEmail(emailFotoToAdd) == true && checkFoto(fotoDTO) == true && emailFotoToAdd è uguale all’email dello studente loggato |
| **PostCondizione** | **Contex:** ModifyProfileControl::aggiungiFotoLista(String emailFotoToAdd, FotoDTO fotoDTO)  **Post:** La fotoDTO f è stata aggiunta al profilo p e viene ritornata.. |
| **Exception** | * Se checkEmail(emailFotoToAdd) == false ||   checkFoto(fotoDTO) == false  throw **InvalidFormatException**   * Se emailFotoToAdd è diverso dall’email dello studente loggato throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + eliminaFotoLista (Lond idFoto f, String email) : FotoDTO | |
| **Descrizione metodo:** il metodo permette di eliminare una foto dal proprio profilo | |
| **PreCondizione** | **Contex:** ModifyProfileControl::eliminaFotoLista (Long idFoto , String email)  **Pre:** (checkEmail(email) == true && idFoto != null) && (l’utente loggato è uno studente && email deve essere uguale all’email dell’utente loggato) |
| **PostCondizione** | **Contex:** ModifyProfileControl::eliminaFotoLista (Long idFoto , String email)  **Post:** La fotoDTO con id = idFoto è stata eliminata e viene ritornata. |
| **Exception** | * Se (checkEmail(email) == false || idFoto == null) throw **InvalidFormatException** * Se l’utente loggato è uno studente e l’email non è uguale all’utente loggato throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + aggiungiFotoProfilo(String emailFotoToAdd, FotoDTO fotoDTO ) : FotoDTO | |
| **Descrizione metodo:** il metodo permette di settare una nuova foto come foto profilo. | |
| **PreCondizione** | **Contex:** ModifyProfileControl::aggiungiFotoProfilo(String emailFotoToAdd, FotoDTO fotoDTO )  **Pre:** checkEmail(emailFotoToAdd) == true && checkFoto(fotoDTO) == true && l’utente con email emailFotoToAdd è l’utente loggato |
| **PostCondizione** | **Contex:** ModifyProfileControl::aggiungiFotoProfilo(String emailFotoToAdd, FotoDTO fotoDTO )  **Post:** la fotoDTO profilo è stata aggiornata e viene ritornata. |
| **Eccezioni** | * se heckEmail(emailFotoToAdd) == false || checkFoto(fotoDTO) == false throw **InvalidFormatException** * se l’utente con email emailFotoToAdd non è l’utente loggato throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + modificaProfilo(String emailStudenteToModify, ProfiloDTO profiloDTO) : boolean | |
| **Descrizione metodo:** il metodo permette di modificare le informazioni presenti nel profilo personale | |
| **PreCondizione** | **Contex:** ModifyProfileControl::modificaProfilo(String emailStudenteToModify, ProfiloDTO profiloDTO)  **Pre:** checkEmail(emailStudenteToModify) == true && l’utente con email emailStudenteToModify è l’utente loggato && checkProfilo(p) == true |
| **PostCondizione** | **Contex:** ModifyProfileControl::modificaProfilo(String emailStudenteToModify, ProfiloDTO profiloDTO)  **Post:** Le info del profilo p sono state salvate |
| **Eccezioni** | se checkEmail(emailStudenteToModify) == false || checkProfilo(p) == false throw **InvalidFormatException**  se l’utente con email emailStudenteToModify non è l’utente loggato throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkFoto (FotoDTO f) : Boolean | |
| **Descrizione metodo:** il metodo controlla la correttezza di formato della foto f. | |
| **PreCondizione** | **Contex:** ModifyProfileContro::checkFoto (FotoDTO f)  **Pre:** f != null |
| **PostCondizione** | **Contex:** ModifyProfileContro::checkFoto (FotoDTO f)  **Post:** Se la foto f rispetta il formato corretto viene ritornato true, altrimenti false. |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkPassword (String password) : Boolean | |
| **Descrizione metodo:** il metodo controlla il formato della password | |
| **PreCondizione** | **Contex:** ModifyProfileContro::checkPassword (String password)  **Pre:** la password deve rispettare il formato. |
| **PostCondizione** | **Contex:** ModifyProfileContro::checkPassword (String password)  **Post:** la password risulta rispettare il formato. |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkProfilo (Profilo p) : Boolean | |
| **Descrizione metodo:** il metodo controlla la correttezza di formato delle modifiche al profilo p. | |
| **PreCondizione** | **Contex:** ModifyProfileContro::checkProfilo (Profilo p)  **Pre:** i valori associati al profilo quali: nome,cognome,colore capelli, colore occhi,altezza,sesso, data si nascita, luogo di nascita, residenza, interessi,lista topic devono essere diversi da null. && i valori associati al profilo rispettano i formati |
| **PostCondizione** | **Contex:** ModifyProfileContro::checkProfilo (Profilo p)  **Post:** Se le modifiche al profilo p rispettano il formato corretto ritorna true, altrimenti false |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkEmail(String email): boolean | |
| **Descrizione metodo:** Metodo che controlla la correttezza del formato di un email | |
| **PreCondizione** | **Contex:** ModifyProfileControl::checkEmail(String email)  **Pre:** email != null && email rispetta il formato della regex |
| **PostCondizione** | **Contex:** ModifyProfileControl::checkEmail(String email)  **Post:** Se viene rispettata la correttezza del formato ritorna true, altrimenti false |

|  |  |
| --- | --- |
| **Nome Metodo:** setFotoProfilo (String emailStudenteToModify, Long fotoId) : FotoDTO | |
| **Descrizione metodo:** consente di settare una foto nella lista delle foto con id=fotoId, come foto profilo | |
| **PreCondizione** | **Contex:** ModifyProfileContro:: (String emailStudenteToModify, Long fotoId)  **Pre: (**checkEmail(emailToModify) == true &&| fotoId != null ) && l’utente con emailToModify risulta essere loggato |
| **PostCondizione** | **Contex:** ModifyProfileContro:: (String emailStudenteToModify, Long fotoId)  **Post:** il sistema permette di impostare una fotoDTO già presente sul sistama come foto profilo e ritorna tale foto |
| **Exception** | Se emailStudenteToModify non rispetta il formato || fotoId != null : throw **InvalidFormatException**  Se l’utente con emailStudenteToModify non è l’utente loggato : throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + cambiaPassword(String emailUtente, String nuovaPassword, String vecchiaPassword) : void | |
| **Descrizione metodo:** Metodo che permette di cambiare password all’utente | |
| **PreCondizione** | **Contex:** ModifyProfileControl::cambiaPassword(String emailUtente, String nuovaPassword, String vecchiaPassword)  **Pre:** l’utente con email emailUtente è l’utente loggato && checkPassword(nuovaPassword) == true && vecchiaPassword corrisponde alla password dell’utente |
| **PostCondizione** | **Contex:**ModifyProfileControl::cambiaPassword(String emailUtente, String nuovaPassword, String vecchiaPassword)  **Post:** la password è stata modificata |
| **Eccezioni** | se nuovaPassword non rispetta il formato throw **InvalidFormatException**  se vecchiaPassword è diversa dalla password dell’utente throw **PasswordMissmatchException**  Se l’utente con emailUtente non è l’utente loggato : throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** trovaProfilo (Long profiloId) : ProfiloDTO | |
| **Descrizione metodo:** il metodo consente di ricercare un Profilo associato al profiloId passato in input. | |
| **PreCondizione** | **Contex:** ModifyProfileControl :: trovaProfilo (Long profiloId)  **Pre:** Se profiloId != null && (l’utente loggato è un Moderatore || l’utente loggato è un Community Manager ) |
| **PostCondizione** | **Contex:**ModifyProfileControl :: trovaProfilo (Long profiloId)  **Post:** ritrona il ProfiloDTO associato al profiloId passato in inpuit |
| **Exception** | Se l’utente loggato non è un moderatore ne un community manager : throw **NotAuthorizedException**  Se il profiloId non è associato ad alcun PorfiloDTO : throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + cancellaAccountPersonale(String email, String password) : void | |
| **Descrizione metodo:** metodo che permette di eliminare il proprio profilo personale | |
| **PreCondizione** | **Contex:** ModifyProfileControl::cancellaAccountPersonale(String email, String password)  **Pre:** l’utente con email email è l’utente loggato && la password inserita corrisponde alla password dell’utente |
| **PostCondizione** | **Contex:** ModifyProfileControl::cancellaAccountPersonale(String email, String password)  **Post:** il profilo è stato eliminato |
| **Eccezioni** | Se l’utente con email non è l’utente loggato : throw **NotAuthorizedException**  se la password inserita è diversa dalla password dell’utente throw **PasswordMissmatchException** |

|  |  |
| --- | --- |
| **Nome Metodo:** trovaFoto (Long fotoId) : FotoDTO | |
| **Descrizione metodo:** il metodo ritorna una FotoDTO associata al fotoId passato in input | |
| **PreCondizione** | **Contex:** ModifyProfileControl :: trovaFoto (Long fotoId)  **Pre:** Se fotId != null && (l’utente loggato è un Moderatore || l’utente loggato è un Community Manager ) |
| **PostCondizione** | **Contex:**ModifyProfileControl :: trovaFoto (Long fotoId)  **Post:** ritrona il FotoDTO associato al profiloId passato in inpuit |
| **Exception** | Se l’utente loggato non è un moderatore ne un community manager : throw **NotAuthorizedException**  Se il fotoId non è associato ad alcun PorfiloDTO : throw **InvalidFormatException** |

**3.21 ModerationController**

|  |
| --- |
| **Nome classe:** GestioneModerazioneController |
| **Descrizione:** Controller che si occupa di gestire tutte le richieste del pacchetto “Moderazione” |
| **Metodi:**   * + inviaSegnalazione(SegnalazioneDTO segnalazioneDTO, Long fotoId) * + inviaSegnalazioneCommunityManager(SegnalazioneDTO segnalazioneDTO, Long fotoId) * + inviaAmmonimento(AmmonimentoDTO ammonimentoDTO, String emailModeratore, String emailStudenteAmmonito, FotoDTO fotoDTO) * + inviaSospensione(SospensioneDTO sospensioneDTO, String emailSospeso) * + moderatoreInSessione() * - checkSegnalazione(Segnalazione s) * - checkAmmonimento(Ammonimento a) * - checkSospensione(Sospensione sp) * - checkEmail(String email) |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaSegnalazione(SegnalazioneDTO segnalazioneDTO, Long fotoId) : SegnalazioneDTO | |
| **Descrizione metodo:** il metodo permette di inviare una segnalazione s alla foto f. | |
| **PreCondizione** | **Contex:** ModerationController::inviaSegnalazione(SegnalazioneDTO segnalazioneDTO, Long fotoId)  **Pre:** checkSegnalazione(s) == true && foto != null |
| **PostCondizione** | **Contex:** ModerationController::inviaSegnalazione(SegnalazioneDTO segnalazioneDTO, Long fotoId)  **Post:** La segnalazione è stata aggiunta alla foto |
| **Exception** | Se il formato della segnalazione non viene rispettato || il fotoId non è associato ad alcuna foto throw **InvalidFormatException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaSegnalazioneCommunityManager(SegnalazioneDTO segnalazioneDTO, Long fotoId) : SegnalazioneDTO | |
| **Descrizione metodo:** il metodo permette di inviare una segnalazione dal moderatore al community manager | |
| **PreCondizione** | **Contex:** ModerationControl::inviaSegnalazioneCommunityManager(SegnalazioneDTO segnalazioneDTO, Long fotoId)  **Pre:** checkSegnalazione(s) == true && fotoId != null && l’utente loggato è un moderatore |
| **PostCondizione** | **Contex:** ModerationControl::inviaSegnalazioneCommunityManager(SegnalazioneDTO segnalazioneDTO, Long fotoId)  **Post:** La segnalazione al community manager è stata inviata |
| **Eccezioni** | se checkSegnalazione(s) == false || fotoId == null throw **InvalidFormatException**  se l’utente loggato non è un moderatore throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaAmmonimento (AmmonimentoDTO ammonimetoDTO, String emailModeratore, String emailStudenteAmmonito, FotoDTO fotoDTO) : AmmonimentoDTO | |
| **Descrizione metodo:** il metodo permette inviare dal Moderatore con emailModeratore ad uno studente con emailStudenteAmmonito un ammonimento DTO che riguarda la fotoDTO | |
| **PreCondizione** | **Contex:** ModeretionControl :: inviaAmmonimento (AmmonimentoDTO ammonimetoDTO, String emailModeratore, String emailStudenteAmmonito, FotoDTO fotoDTO)  **Pre:** chekEmail(emailModeratore) == true && chekEmail(emailStudenteAmmonimto) == true && l’utente in sessione è un Moderatore o un Community Manager && checkAmmonimeto == true |
| **PostCondizione** | **Contex:** GModeretionControl :: inviaAmmonimento (AmmonimentoDTO ammonimetoDTO, String emailModeratore, String emailStudenteAmmonito, FotoDTO fotoDTO)  **Post:** il moderatore con emailModeratore invia un ammonimentoDTO allo studente con emailStudenteAmmonito riguardante la fotoDTO. |
| **Exception** | Se emailModeratore e emailStudenteAmmonito non rispettano il formato || il formato dell ammonimetoDTO non è rispettato : throw **InvalidFormatException**  Se l’utente in sessione non è un Moderatore o non è un Community Manager : throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** + inviaSospensione(SospensioneDTO sospensioneDTO, String emailSospeso) : SospensioneDTO | |
| **Descrizione metodo:** il metodo permette l’invio di una sospensione a uno studente | |
| **PreCondizione** | **Contex:** ModerationControl:: inviaSospensione(SospensioneDTO sospensioneDTO, String emailSospeso)  **Pre:** checkSospensione(sp) == true && l’utente loggato è un community manager && checkEmail(emailSospeso) == true |
| **PostCondizione** | **Contex:** ModerationControl:: inviaSospensione(SospensioneDTO sospensioneDTO, String emailSospeso)  **Post:** la sospensione è stata aggiunta all’utente |
| **Eccezioni** | se checkSospensione(sp) == false || checkEmail(emailSospeso) == false throw **InvalidFormatException**  se l’utente loggato non è un community manager throw **NotAuthorizedException** |

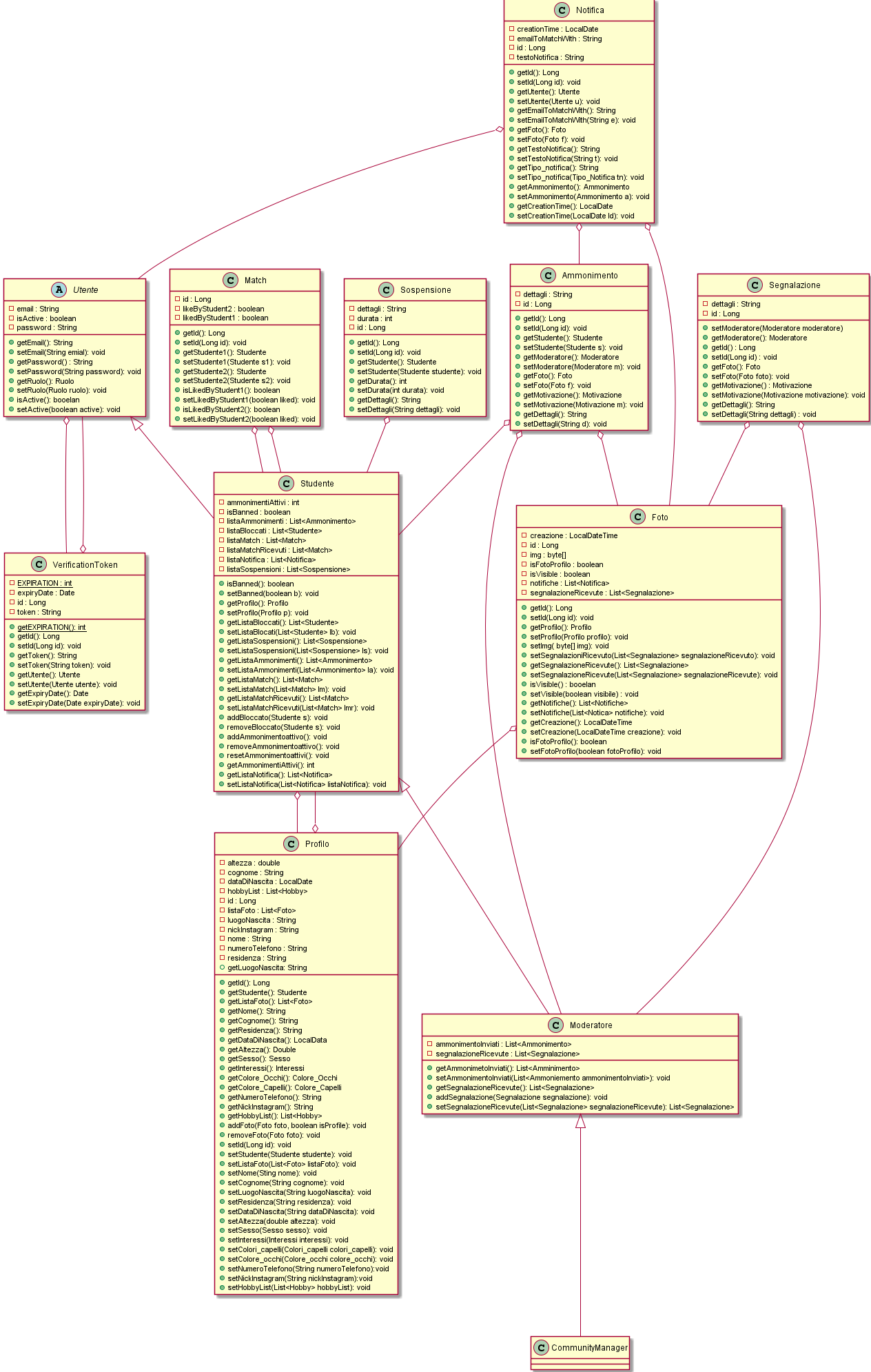
|  |  |
| --- | --- |
| **Nome Metodo:** + moderatoreInSessione() : moderatoreDTO | |
| **Descrizione metodo:** il metodo restituisce il moderatore o il community manager in sessione | |
| **PreCondizione** | **Contex:** ModerationControl::moderatoreInSessione()  **Pre:** l’utente in sessione è un moderatore || l’utente in sessione è un community manager |
| **PostCondizione** | **Contex:** ModerationControl:: ModerationControl::moderatoreInSessione()  **Post:** // |
| **Eccezioni** | se l’utente loggato non è un moderatore || l’utente loggato non è un community manager throw **NotAuthorizedException** |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkSegnalazione (Segnalazione s) : boolean | |
| **Descrizione metodo:** metodo che controlla la correttezza di formato della segnalazione s | |
| **PreCondizione** | **Contex:**ModerationControl::checkSegnalazione (Segnalazione s)  **Pre:** s.getMotivazione() != null && s.getDettagli() != null && s.getDettagli().length() > 0 && s.getDettagli().length() < 250 |
| **PostCondizione** | **Contex:** ModerationControl::checkSegnalazione (Segnalazione s)  **Post:** se il formato della segnalazione é corretto ritorna true,altrimenti false |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkSospensione (Sospensione sp) : boolean | |
| **Descrizione metodo:** metodo che controlla la correttezza di formato della sospensione sp. | |
| **PreCondizione** | **Contex:**ModerationControl::checkSospensione (Sospensione sp)  **Pre:** sp.getDurata() != 0 && sp.getDettagli() != null && sp.getDettagli().length() > 0 && sp.getDettagli().length() < 250 |
| **PostCondizione** | **Contex:** ModerationControl::checkSospensione (Sospensione sp)  **Post:** se il formato della sospensione é corretto ritorna true,altrimenti false |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkAmmonimento (Ammonimento a) : boolean | |
| **Descrizione metodo:** metodo che controlla la correttezza di formato dell’ammonimento a. | |
| **PreCondizione** | **Contex:** ModerationControl::checkAmmonimento (Ammonimento a)  **Pre:** a.getMotivazione() != null && a.getDettagli() != null && a.getDettagli().length() > 0 && a.getDettagli().length() < 250 |
| **PostCondizione** | **Contex:** ModerationControl::checkAmmonimento (Ammonimento a)  **Post:** se il formato dell’ammonimento é corretto ritorna true,altrimenti false |

|  |  |
| --- | --- |
| **Nome Metodo:** - checkEmail(String email): boolean | |
| **Descrizione metodo:** Metodo che controlla la correttezza del formato di un email | |
| **PreCondizione** | **Contex:** ModerationControl::checkEmail(String email)  **Pre:** email != null && email rispetta il formato della regex |
| **PostCondizione** | **Contex:** ModerationControl::checkEmail(String email)  **Post:** Se viene rispettata la correttezza del formato ritorna true, altrimenti false |

1. **Class Diagram**