Università degli Studi di Salerno

Corso di Ingegneria del Software

Object Design Document 2.0

A.A. 2025/26

Progetto: EarthLocals	Versione: 2.0
Documento: Object Design Document	Data: 19/01/2026

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Abbatiello Simone	0512119659
Niemiec Francesco	0512118999
Rega Maristella	0512119032
Squitieri Andrea	0512119008

Scritto da:	Abbatiello Simone, Niemiec Francesco, Rega Maristella, Squitieri Andrea

Revision History

Data	Versione	Descrizione	Autore
25/11/2025	0.1	Stesura iniziale dell'Object Design Document	Abbatiello Simone, Niemiec Francesco, Rega Maristella, Squitieri Andrea
30/11/2025	0.2	Definizioni, acronimi e abbreviazioni	Abbatiello Simone, Rega Maristella
5/12/2025	0.3	Gestione Utente	Abbatiello Simone, Niemiec Francesco, Rega Maristella, Squitieri Andrea
6/12/2025	0.4	Gestione Missione	Abbatiello Simone, Niemiec Francesco, Rega Maristella, Squitieri Andrea
10/12/2025	0.5	Gestione Candidature	Abbatiello Simone, Niemiec Francesco, Rega Maristella, Squitieri Andrea
12/12/2025	0.6	Gestione Recensioni	Niemiec Francesco, Andrea Squitieri
19/12/2025	0.7	Sotto consiglio del tutor e del professore definito Gestione Email	Rega Maristella
22/12/2025	1.0	Correzioni finali e fix estetici	Abbatiello Simone, Niemiec Francesco, Rega Maristella, Squitieri Andrea
02/01/2026	1.1	Aggiunta Trade-off	Niemiec Francesco
03/01/2026	1.2	Definizione linee guida	Squitieri Andrea
10/01/2026	1.3	Formazione pacchetti	Rega Maristella,

			Squitieri Andrea
11/01/2026	1.4	Correzione pacchetti	Abbatiello Simone
12/01/2026	1.5	Aggiornamenti interfacce	Abbatiello Simone, Squitieri Andrea
19/01/2026	2.0	Correzioni finali	Niemiec Francesco, Rega Maristella

Contenuti

1. Introduzione

1.1 Definizioni, acronimi e abbreviazioni

Nel seguente documento sono presenti diversi acronimi, qui riportiamo il significato di ciascuno di essi:

- MVC: Model View Controller.
- DB: Database.
- JDBC: Java DataBase Connectivity.
- RAD: Requirements Analysis Document.
- PS: Problem Statement.

1.2 Riferimenti

Per la stesura di questo documento si fa riferimento ai termini utilizzati ed ampiamente descritti nei documenti di Problem Statement, Requirements Analysis e System Design.

1.3 Trade-offs

- **1.3.1** Rapid Development vs Functionality: per riuscire a consegnare una prima versione del sistema perfettamente funzionante abbiamo sacrificato il sistema delle recensioni e le pagine personali dell'utente.
- **1.3.2** Minimum number of Error vs Functionality: la scelta delle funzionalità da implementare nasce anche dalla necessità di voler avere un primo sistema che miri a minimizzare il numero di errori riscontrabili ed offra comunque funzionalità soddisfacenti.

- **1.3.3** Rapid Development vs Robustness: nonostante i tempi contenuti siamo riusciti a rendere il codice robusto anche grazie all'ampio testing effettuato.
- **1.3.4** Facilità d'uso (& user-friendliness) vs Free-roaming: il design del sito è minimale, con funzionalità essenziali, per guidare l'utente nella navigazione e facilitarlo nell'utilizzo nel sito anziché lasciargli piena libertà di ricerca e navigazione.
- **1.3.5** Prestazioni vs Costi: l'interfaccia è data da semplici fogli di stile, ciò permette di ottenere prestazioni più che discrete, invece che usare complesse librerie a pagamento.
- **1.3.6** Tracciabilità dei requisiti vs Costo: nel corso dei vari documenti (e delle loro versioni) è possibile osservare l'evoluzione che le varie funzionalità implementate nel sito hanno avuto, ognuna delle quali è facilmente riconducibile al proprio requisito funzionale.
- **1.3.7** Maintenability vs Backwards compatibility: abbiamo scelto di utilizzare le ultime versioni dei vari moduli e delle tecnologie necessarie per lo sviluppo, anche a discapito dell'esclusione del supporto alla retrocompatibilità.
- **1.3.8** Efficiency vs Rapid Development: seppur ponendo enfasi su uno sviluppo rapido, data la natura delle funzionalità da implementare, si è scelto di minimizzare il quantitativo di pagine generate dinamicamente (ristretto solamente per nuove pagine delle missioni e pagine personali).

1.4 Linee guida

Per quanto concerne lo sviluppo verranno utilizzate le seguenti linee guida:

- **1.4.1** Naming convention generale

Devono essere rispettate queste caratteristiche

- Nomi di classi brevi ma efficaci,
- Nomi di metodi medio-corti ma esplicativi,
- Tutti i nomi devono essere significativi.

Inoltre adottiamo le seguenti convenzioni:

- Pacchetti: lower-case.
- Classi: PascalCase.
- Metodi/Variabili: camelCase.

- **1.4.2** Naming convention risorse

Per tutto ciò che non fa parte direttamente del codice java (risorse, css, pagine, immagini) si seguono due semplici regole: tutto deve essere in lower case ed in caso di nomi composti essi vengono concatenati con "_".

2. Pacchetti

L'implementazione del back-end è dominata dal massiccio uso di Spring, ciò comporta necessariamente questa strutturazione:

- pom.xml : Project Object Model necessario per configurare dipendenze e caratteristiche della build usando Maven.
- src/main/java/com/earthlocals/earthlocals che contiene tutto il codice
- src/resources per le risorse statiche
- EarthLocalsApplication.java che è lo starting point.

Inoltre l'uso di Thymeleaf comporta l'esistenza di src/resources/templates che contiene le pagine generate.

Entrando nei dettagli abbiamo stabilito la seguente divisione per quanto riguarda il codice del back-

end:

- /config : contiene classi necessarie per la corretta configurazione di diverse funzionalità
- /events: per classi relative alla gestione degli eventi
- /model: il centro dell'implementazione, contiene due tipi di elementi 1) le entità, ovvero classi memorizzate nel database, 2) repository, classi che implementano le operazioni crud.
- /service: contiene le implementazioni delle classi di servizio. La suddivisione generale è in più classi del tipo /gestioneEntita contente a sua volta obbligatoriamente una classe GestioneEntita ed opzionalmente uno o più pacchetti del tipo /dto che racchiude i vari dto necessari alla classe, /exception con le varie eccezioni, /specific contente classi che implementano funzionalità specifiche delle varie classi (e la directory prende nomi diversi a seconda di esse).
- /system: contente i vari controller (che in Spring sono componenti responsabili della gestione delle richieste e delle risposte HTTP) ed una directory /account con i controller relativi all'account
- /utility per classi di utilità, è a sua volta divisa in :
    - /costraints per quanto riguarda vincoli e validatori delle varie classi
    - /interfaces con interfacce utili alle classi.

2.1 GestioneUtente

Non sono presenti invarianti.

Servizio / Operazione	Contratti
+ registerVolunteer(Volontario volontario)	**Abstract**: Chiamato durante la registrazione per salvare il nuovo utente. **Precondizioni**: context GestioneUtente::registerVolunteer(Volontario volontario) pre: !self.utenti->includes(volontario) **Postcondizioni**: context GestioneUtente::registerVolunteer (Volontario volontario) post: self. utenti ->includes(volontario) and volontario.stato = StatoUtente::PENDING
+ registerOrganizer(Utente organizzatore)	**Abstract:** Chiamato durante la registrazione effettuata da un organizzatore **Precondizioni:** context GestioneUtente::registerOrganizer(Utente organizzatore) pre: !self.utenti->includes(organizzatore) **Postcondizioni:** context GestioneUtente::registerOrganizer (Utente organizzatore) post:

	self. utenti ->includes(organizzatore) and organizzatore.stato = StatoUtente::PENDING
+ createModerator(Utente moderatore)	**Abstract**: Chiamato per creare un nuovo utente moderatore. **Precondizioni**: context GestioneUtente::createModerator (Utente moderatore) pre: !self.utenti->includes(moderatore) **Postcondizioni**: context GestioneUtente::createModerator (Utente moderatore) post: self. utenti ->includes(moderatore) and moderatore.stato = StatoUtente::CONFIRMED
+ activateAccount(String token)	**Abstract**: Chiamato per confermare la registrazione dopo il click sul link email. **Precondizioni**: context GestioneUtente::activateAccount(token) pre: self.tokens->includes(token) **Postcondizioni**: context GestioneUtente::activateAccount(token) post: self.tokens.findByToken(token).utente.pending = false
+ verificaAccesso(String email, String password) -> Utente	**Abstract**: Verifica le credenziali al login. Restituisce utenteTrovato. **Postcondizioni**: context GestioneUtente::verificaAccesso(String email, String password) -> Utente post: (self.utenti.findByEmail(email).password.matches(password) implies result = self.utenti.findByEmail(email)) or result = null
+ editUser(Utente utenteModificato) -> Utente	**Abstract**: Aggiorna i dati del profilo utente. **Precondizioni**: context GestioneUtente::editUser(Utente utenteModificato) -> Utente pre: self.utenti->exists(u| u.id = utenteModificato.id) **Postcondizioni**: context GestioneUtente::editUser(Utente utenteModificato) -> Utente post: self.utenti->includes(utenteModificato)

+ editPassword(Utente utente, String pwCorrente, String modificaPassword) -> Utente	**Abstract**: Gestisce il cambio password, verificando la vecchia e la validità della nuova. **Precondizioni**: context GestioneUtente::editPassword(Utente utente, String pwCorrente, String modificaPassword) -> Utente pre: self.utenti->includes(utente) and utente.password.matches(pwCorrente) **Postcondizioni**: context GestioneUtente::editPassword(Utente utente, String pwCorrente, String pwNuova) -> Utente post: utente.password.matches(pwNuova) and self.utenti->includes(utente)
+editPassport(Volontario volontario, String numeroPassaporto, LocalDate dataRilascio, LocalDate dataScadenza, MultipartFile passaporto) -> Volontario	**Abstract:** Gestisce la modifica del passaporto da parte di un utente. **Precondizioni:** context GestioneUtente::editPassport(Volontario volontario, String numeroPassaporto, LocalDate dataRilascio, LocalDate dataScadenza, MultipartFile passaporto) -> Volontario pre: self.volontari->includes(volontario) and volontario.passaporto <> null **Postcondizioni:** context GestioneUtente::editPassport(Volontario volontario, String numeroPassaporto, LocalDate dataRilascio, LocalDate dataScadenza, MultipartFile passaporto) -> Volontario post: volontario.passaporto.equals(passaporto) and self.volontari->includes(volontario)
+ resetPassword(String email) -> String	**Abstract**: Avvia il recupero password verificando l'esistenza dell'email. **Precondizioni**: context GestioneUtente::resetPassword(String email) -> String pre: self.utenti->exists(u|u.email = email) **Postcondizioni**: context GestioneUtente::resetPassword(String email) -> String post:

	self.utenti->exists(u|u.email = email and u.pwTemp != null and u.scadenzaPwTemp = (currentTime() + TEMPO_MAX)
+ getProfileInformation(Integer id) -> Utente	**Abstract**: Recupera i dati pubblici del profilo di un altro utente. **Precondizioni**: context GestioneUtente::getProfiloUtente(Integer riferimento) -> Utente pre: self.utenti->exists(u|u.riferimento = riferimento)
+ createPasswordResetToken(String email) -> String	**Abstract**: Avvia il recupero password verificando l'esistenza dell'email. **Precondizioni**: context GestioneUtente::createPasswordResetToken(email : String) -> String pre: self.utenti->exists(u | u.email = email) **Postcondizioni:** context GestioneUtente::createPasswordResetToken(email : String) -> String post: self.utenti->exists(u | u.email = email and u.pwTemp = result and u.pwTemp <> null and u.scadenzaPwTemp = currentTime() + TEMPO_MAX)
+ createVerificationToken(Utente utente) -> String	**Abstract**: Crea un token di verifica per completare la registrazione. **Precondizioni**: context GestioneUtente::createVerificationToken(utente : Utente) -> String pre: self.utenti->includes(utente) and utente.stato = StatoUtente::PENDING **Postcondizioni:** context GestioneUtente::createVerificationToken(utente : Utente) post: utente.verificationToken <> null and utente.scadenzaVerificationToken = currentTime() + TEMPO_MAX
+ getPassportFile(Volontario utente)	**Abstract**: Permette di scaricare il file del passaporto. **Precondizioni**:

	context GestioneUtente::getPassportFile(utente : Volontario) pre: self.utenti->includes(utente) and utente.passportFile <> null

2.2 Gestione Missione

Non sono presenti invarianti

Servizio / Operazione	Contratto
+ registerMissione(Missione missione) -> Missione	**Abstract**: Salva la missione nel catalogo (stato iniziale Pending). **Precondizioni**: context GestioneMissione::registerMissione(Missione missione) pre: !self.missioni->includes (missione) **Postcondizioni**: context GestioneMissione::registerMissione(Missione missione) post: self.missioni->includes (missione) and missione.stato = StatoMissione::PENDING
+ approvaMissione(Missione missione)	**Abstract**: Il moderatore approva la missione. Cambia stato e invia l'email. **Precondizioni**: context GestioneMissione::approvaMissione(Missione missione) pre: self.missioni->includes (missione) and missione.stato = StatoMissione::PENDING **Postcondizioni**: GestioneMissione::approvaMissione(Missione missione) post: self.missioni->includes (missione) and missione.stato = StatoMissione::CONFIRMED
+ rifiutaMissione(Missione missione)	**Abstract**: Il moderatore rifiuta la missione.

	Precondizioni: context GestioneMissione::rifiutaMissione(Missione missione) pre: self.missioni->includes (missione) **Postcondizioni**: context GestioneMissione::rifiutaMissione(Missione missione) post: missione.stato = StatoMissione::DENIED
+ annullaMissione(Missione missione)	**Abstract**: L'organizzatore annulla la missione. **Precondizioni**: context GestioneMissione::annullaMissione(Missione missione) pre: self.missioni->includes (missione) **Postcondizioni**: context GestioneMissione::annullaMissione(Missione missione) post: !self.missioni->includes (missione)
+getMissioniAperte(Paese paese) -> Collection<Missione>	**Abstract**: Restituisce la lista delle missioni che soddisfano i criteri di ricerca. **Precondizioni**: context GestioneMissione::getMissioniAperte(paese : Paese) : Collection<Missione> pre: paese <> null **Postcondizioni**: context GestioneMissione::getMissioniAperte(paese : Paese) : Collection<Missione> post: result = self.missioni->select(m | m.paese = paese and m.stato = StatoMissione::CONFIRMED)
+ getMissioneById(Long id) -> Missione	**Abstract**: Recupera i dettagli di una missione (usato anche prima di una recensione). **Precondizioni**: context GestioneMissione::getMissioneById(Long id) -> Missione pre: self.missioni->exists(m|m.id = id) **Postcondizioni**: context GestioneMissione::getMissioneById(Long id) -> Missione post: self.missioni->exists(m|m.id = id and result = m)

+ getMissioniOrganizzatore (Utente organizzatore) -> Collection<Missione>	**Abstract**: Recupera le missioni dell'organizzatore per la sua area personale. **Precondizioni**: context GestioneMissione::getMissioniOrganizzatore(Utente organizzatore) -> Collection<Missione> pre: self.missioni.organizzatori->includes(organizzatore) **Postcondizioni**: context GestioneMissione::getMissioniOrganizzatore(Utente organizzatore) -> Collection<Missione> post: self.missioni->forAll(m| m.organizzatore = organizzatore => result.includes(m))
+ getMissioniPending() -> Collection<Missione>	**Abstract:** Restituisce la liste delle missioni che si trovano nello stato pending. **Postcondizioni**: context GestioneMissione::getMissioniPending() : Collection<Missione> post: result = self.missioni->select(m |m.stato = StatoMissione::PENDING)

2.3 GestioneCandidature

Non sono presenti invarianti.
Tutti i seguenti metodi sono contenuti nella classe GestioneCandidatura.

Servizio / Operazione	Contratto
+ registerCandidatura(Candidatura candidatura)	**Abstract**: Crea una nuova istanza di candidatura associando il volontario alla missione scelta. **Precondizioni**: context GestioneCandidature::registerCandidatura(Candidatura candidatura) pre: !self.candidature->includes(candidatura) **Postcondizioni**:

	context GestioneCandidature::registerCandidatura(Candidatura candidatura) post: self.candidature->includes(candidatura) and candidature.stato = StatoCandidatura::PENDING
+ removeCandidatura(Candidatura candidatura)	**Abstract**: Elimina (o annulla) una candidatura esistente su richiesta del volontario. **Precondizioni**: context GestioneCandidature::removeCandidatura(Candidatura candidatura) pre: self.candidature ->includes(candidatura) **Postcondizioni**: context GestioneCandidature::removeCandidatura(Candidatura candidatura) post: !self.candidature ->includes(candidatura)
+ acceptCandidatura(Candidatura candidatura)	**Abstract**: Registra l'approvazione del volontario da parte dell'organizzatore. **Precondizioni**: context GestioneCandidature::acceptCandidatura(Candidatura candidatura) pre: self.candidature ->includes(candidatura) and candidatura.stato = StatoCandidatura::PENDING **Postcondizioni**: context GestioneCandidature::acceptCandidatura(Candidatura candidatura) post: self.candidature ->includes(candidatura) and candidatura.stato = StatoCandidatura::CONFIRMED
+ rejectCandidatura(Candidatura candidatura)	**Abstract**: Registra il rifiuto della candidatura da parte dell'organizzatore. **Precondizioni**: context GestioneCandidature::rejectCandidature(Candidatura candidatura) pre: self.candidature->includes(candidatura) and candidatura.stato = StatoCandidatura::PENDING

	Postcondzioni: context GestioneCandidature::rejectCandidature(Candidatura candidatura) post: candidatura.stato = StatoCandidatura::DENIED
+getRichiesteCandidatura(Utente organizzatore) -> Collection<Candidature>	**Abstract**: Ritorna tutte le candidature relative alle missioni di un organizzatore. **Precondizioni**: context GestioneCandidature::getRichiesteCandidatura(Utente organizzatore) - > Collection<Candidature> organizzatore.roles->includes(Role::ORGANIZZATORE) **Postcondzioni**: context GestioneCandidature::getRichiesteCandidatura(Utente organizzatore) - > Collection<Candidature> post: self.candidature->forAll(c | c.missione.organizzatore = organizzatore implies result->includes(c)) and result->forAll(c | self.candidature->includes(c) and c.missione.organizzatore = organizzatore)
+getEsperienzeVolontario(Volontario volontario) -> Collection<Candidature>	**Abstract:** Ritorna tutte le candidature accettate del volontario. **Precondizioni**: context GestioneCandidature::getEsperienzeVolontario(Volontario volontario) -> Collection<Candidature> pre: self.candidature->exists(c | c.volontario = volontario) **Postcondizioni:** context GestioneCandidature::getEsperienzeVolontario(Volontario volontario) post: result = self.candidature->select(c | c.volontario = volontario and c.stato = StatoCandidatura::CONFIRMED)
+getCandidatureVolontario(Volontario volontario) -> Collection<Candidature>	**Abstract:** Ritorna tutte le candidature del volontario. **Precondizioni:** context GestioneCandidature::getCandidatureVolontario(Volontario Volontario) -> Collection<Candidature> pre: self.candidature->exists(c | c.volontario = volontario)

	Postcondizioni: GestioneCandidature::getCandidatureVolontario(Volontario volontario) -> Collection<Candidature> post: result = self.candidature->select(c |c.volontario = volontario)

2.4 GestioneRecensioni

Non sono presenti invarianti.

Servizio / Operazione	Contratto
+aggiungiRecensioneOrganizzatore(Recensione recensione)	**Abstract**: Salva una recensione scritta da un volontario per un organizzatore. **Precondizioni**: context GestioneRecensioni::aggiungiRecensioneOrganizzatore(Recensione recensione) pre: !self.recensioni->includes(recensione) **Postcondizioni**: context GestioneRecensioni::aggiungiRecensioneOrganizzatore(Recensione recensione) post: self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC
+aggiungiRecensioneVolontario(Recensione recensione)	**Abstract**: Salva una recensione scritta da un organizzatore per un volontario. **Precondizioni**: context GestioneRecensioni::aggiungiRecensioneVolontario(Recensione recensione) pre: !self.recensioni->includes(recensione) **Postcondizioni**: context GestioneRecensioni::aggiungiRecensioneVolontario(Recensione recensione) post: self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC
+ rimuoviRecensione(Recensione recensione)	**Abstract**: Elimina una recensione segnalata ritenuta inappropriata dal moderatore. **Precondizioni**:

	context GestioneRecensioni::rimuoviRecensione(Recensione recensione) pre: self.recensioni->includes(recensione) **Postcondizioni**: context GestioneRecensioni::rimuoviRecensione(Recensione recensione) post: !self.recensioni->includes(recensione)
+ approvaRecensione(Recensione recensione)	**Abstract**: Conferma la validità di una recensione segnalata, mantenendola pubblicata. **Precondizioni**: context GestioneRecensioni::approvaRecensione(Recensione recensione) pre: self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::REPORTED **Postcondzioni**: context GestioneRecensioni::approvaRecensione(Recensione recensione) post: self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC
+ segnalaRecensione(Recensione recensione)	**Abstract**: Segnala una recensione. **Precondizioni**: context GestioneRecensioni::segnalaRecensione(Recensione recensione) pre: self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC **Postcondizioni**: GestioneRecensioni::segnalaRecensione(Recensione recensione) post: self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::REPORTED

2.5 GestioneEmail

Non sono presenti invarianti e nemmeno postcondizioni.

Tutti i seguenti metodi sono definiti all'interno di "GestioneEmail.java".

Servizio / Operazione (da Diagrammi)	Contratto
+ inviaConferma(Long id, String token)	**Abstract:** Invia link di attivazione account. **Precondizioni:** context GestioneEmail::inviaConferma(id:Long, token : String) pre: self.utenti->exists(u | u.id = id and u.stato = StatoUtente::PENDING)
+ inviaEmailRecupero(String email, String token)	**Abstract:** Invia credenziali per reset password. **Precondizioni:** context GestioneEmail::inviaEmailRecupero(email: String, token : String) pre: self.utenti->exists(u | u.email = email)
+ inviaEmailCandidatura(Long id)	**Abstract:** Notifica nuova candidatura all'organizzatore. **Precondizioni:** context GestioneEmail::inviaEmailCandidatura(id : Long) self.candidature->exists(c | c.id = id and c.missione.organizzatore <> null)
+inviaEmailAnnullamentoCandidatura (Long id)	**Abstract:** Notifica annullamento candidatura (dal volontario). **Precondizioni:** context GestioneEmail::inviaEmailAnnullamentoCandidatura (id : Long) pre: self.candidature->exists(c | c.id = id and c.missione.organizzatore <> null)
+inviaEmailAccettazioneCandidatura(Long id)	**Abstract:** Notifica accettazione candidatura. **Precondizioni:** context GestioneEmail::inviaEmailAccettazioneCandidatura(id : Long) self.candidature->exists(c | c.id = id)
+ inviaEmailRifiutoCandidatura (Long id)	**Abstract:** Notifica rifiuto candidatura. **Precondizioni:** context GestioneEmail::inviaEmailRifiutoCandidatura(id : Long) self.candidature->exists(c | c.id = id)

+invioEmailApprovazioneMissione(Long id)	**Abstract:** Notifica approvazione missione. **Precondizioni:** context GestioneEmail::invioEmailApprovazioneMissione(id : Long) pre: self.missioni->exists(m | m.id = id and m.stato = StatoMissione::CONFIRMED and m.organizzatore <> null)
+invioEmailViolazioneRecensione(Long id)	**Abstract:** Notifica rimozione recensione. **Precondizioni:** context GestioneEmail::invioEmailViolazioneRecensione(id : Long) pre: self.recensioni->exists(r | r.id = id and r.utente <> null)
+inviaEmailCreazioneModeratore(Long id)	**Abstract:** Invia credenziali al nuovo moderatore creato. **Precondizioni:** context GestioneEmail::invioEmailCreazioneModeratore (id : Long) pre: self.utenti->exists(u | u.id = id and u.oclIsKindOf(Moderatore))