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 le seguenti 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 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.

Nome metodo	+registerVolunteer(Volontario volontario)
Descrizione	Chiamato durante la registrazione per salvare il nuovo utente.
Pre-condizioni	*context GestioneUtente::registerVolunteer(volontario: Volontario)* **pre**: !self.utenti->includes(volontario)
Post-condizioni	*context GestioneUtente::registerVolunteer (volontario: Volontario)* **post**: self. utenti ->includes(volontario) and volontario.stato = StatoUtente::PENDING

Nome metodo	+registerOrganizer(Utente organizzatore)
Descrizione	Chiamato durante la registrazione effettuata da un organizzatore
Pre-condizioni	*context GestioneUtente::registerOrganizer(organizzatore: Utente)* **pre***:* !self.utenti->includes(organizzatore)
Post-condizioni	*context GestioneUtente::registerOrganizer (organizzatore: Utente)* **post***:* self. utenti ->includes(organizzatore) and organizzatore.stato = StatoUtente::PENDING

Nome metodo	+createModerator(Utente moderatore)
Descrizione	Chiamato per creare un nuovo utente moderatore.
Pre-condizioni	*context GestioneUtente::createModerator (moderatore: Utente)* **pre:** !self.utenti->includes(moderatore)
Post-condizioni	*context GestioneUtente::createModerator (moderatore: Utente)* **post:** self. utenti ->includes(moderatore) and moderatore.stato = StatoUtente::CONFIRMED

Nome metodo	+activateAccount(String token)
Descrizione	Chiamato per confermare la registrazione dopo il click sul link email.
Pre-condizioni	*context GestioneUtente::activateAccount(token: String)* **pre**: self.tokens->includes(token)
Post-condizioni	*context GestioneUtente::activateAccount(token: String)* **post**: self.tokens.findByToken(token).utente.pending = false

Nome metodo	+verificaAccesso(String email, String password) : Utente
Descrizione	Verifica le credenziali al login. Restituisce utenteTrovato.
Pre-condizioni	
Post-condizioni	*context GestioneUtente::verificaAccesso(email: String, password: String) : Utente* ***post****:* *self.utenti.findByEmail(email).password.matches(password) implies* *result = self.utenti.findByEmail(email)* *or result = null*

Nome metodo	+editUser(Utente utenteModificato) : Utente
Descrizione	Aggiorna i dati del profilo utente.
Pre-condizioni	*context GestioneUtente::editUser(utenteModificato: Utente) : Utente* **pre**: self.utenti->exists(u| u.id = utenteModificato.id)
Post-condizioni	*context GestioneUtente::editUser(utenteModificato: Utente) : Utente* **post**: self.utenti->includes(utenteModificato)

Nome metodo	+editPassword(Utente utente, String pwCorrente, String modificaPassword) : Utente
Descrizione	Gestisce il cambio password, verificando la vecchia e la validità della nuova.
Pre-condizioni	*context GestioneUtente::editPassword(utente: Utente, pwCorrente: String, modificaPassword: String) : Utente* **pre**: self.utenti->includes(utente) and utente.password.matches(pwCorrente)
Post-condizioni	*context GestioneUtente::editPassword(Utente utente, String pwCorrente, String pwNuova) : Utente* **post**: utente.password.matches(pwNuova) and self.utenti->includes(utente)

Nome metodo	+editPassport(Volontario volontario, String numeroPassaporto, LocalDate dataRilascio, LocalDate dataScadenza, MultipartFile passaporto) : Volontario
Descrizione	Gestisce la modifica del passaporto da parte di un utente.
Pre-condizioni	*context GestioneUtente::editPassport(volontario: Volontario, numeroPassaporto: String, dataRilascio: LocalDate, dataScadenza: LocalDate, passaporto: MultipartFile) : Volontario* **pre**: self.volontari->includes(volontario) and volontario.passaporto <> null
Post-condizioni	*context GestioneUtente::editPassport(volontario: Volontario, numeroPassaporto: String, dataRilascio: LocalDate, dataScadenza: LocalDate, passaporto: MultipartFile) : Volontario* **post**: volontario.passaporto.equals(passaporto) and self.volontari->includes(volontario)

Nome metodo	+resetPassword(String email) : String
Descrizione	Avvia il recupero password verificando l'esistenza dell'email.
Pre-condizioni	*context GestioneUtente::resetPassword(email: String) : String* **pre**: self.utenti->exists(u|u.email = email)
Post-condizioni	*context GestioneUtente::resetPassword(String email) : String* **post***:* self.utenti->exists(u| u.email = email and u.pwTemp != null and u.scadenzaPwTemp = (currentTime() + TEMPO_MAX))

Nome metodo	+getProfileInformation(Integer id) : Utente
Descrizione	Recupera i dati pubblici del profilo di un altro utente.
Pre-condizioni	*context GestioneUtente::getProfiloUtente(riferimento: Integer) : Utente* **pre**: self.utenti->exists(u|u.riferimento = riferimento)
Post-condizioni	

Nome metodo	+createPasswordResetToken(String email) : String
Descrizione	Avvia il recupero password verificando l'esistenza dell'email.
Pre-condizioni	*context GestioneUtente::createPasswordResetToken(email : String) : String* **pre**: self.utenti->exists(u | u.email = email)
Post-condizioni	*context GestioneUtente::createPasswordResetToken(email : String) : String* **post**: self.utenti->exists(u| u.email = email and result = u.pwTemp and u.pwTemp <> null and u.scadenzaPwTemp = currentTime() + TEMPO_MAX)

Nome metodo	+createVerificationToken(Utente utente) : String
Descrizione	Crea un token di verifica per completare la registrazione.
Pre-condizioni	*context GestioneUtente::createVerificationToken(utente : Utente) : String* **pre**: self.utenti->includes(utente) and utente.stato = StatoUtente::PENDING
Post-condizioni	*context GestioneUtente::createVerificationToken(utente : Utente)* **post**: utente.verificationToken <> null and utente.scadenzaVerificationToken = currentTime() + TEMPO_MAX

Nome metodo	+getPassportFile(Volontario utente)
Descrizione	Permette di scaricare il file del passaporto.
Pre-condizioni	*context GestioneUtente::getPassportFile(utente : Volontario)* **pre**: self.utenti->includes(utente) and utente.passportFile <> null
Post-condizioni	*context GestioneUtente::createVerificationToken(utente : Utente)* **post**: utente.verificationToken <> null and utente.scadenzaVerificationToken = currentTime() + TEMPO_MAX

2.2 Gestione Missione

Non sono presenti invarianti

Nome metodo	+registerMissione(Missione missione) -> Missione
Descrizione	Salva la missione nel catalogo (stato iniziale *Pending*).
Pre-condizioni	*context GestioneMissione::registerMissione(Missione missione)* **pre:** !self.missioni->includes (missione)
Post-condizioni	*context GestioneMissione::registerMissione(Missione missione)* **post:** self.missioni->includes (missione) and missione.stato = StatoMissione::PENDING

Nome metodo	+approvaMissione(Missione missione)
Descrizione	Il moderatore approva la missione. Cambia stato e invia l'email.
Pre-condizioni	*context GestioneMissione::approvaMissione(Missione missione)* **pre:** self.missioni->includes (missione) and missione.stato = StatoMissione::PENDING
Post-condizioni	*context GestioneMissione::approvaMissione(Missione missione)* **post:** self.missioni->includes (missione) and missione.stato = StatoMissione::CONFIRMED

Nome metodo	+rifiutaMissione(Missione missione)
Descrizione	Il moderatore rifiuta la missione.
Pre-condizioni	*context GestioneMissione::rifiutaMissione(Missione missione)* **pre:** self.missioni->includes (missione)
Post-condizioni	*context GestioneMissione::rifiutaMissione(Missione missione)* **post:** missione.stato = StatoMissione::DENIED

Nome metodo	+annullaMissione(Missione missione)
Descrizione	L'organizzatore annulla la missione.
Pre-condizioni	*context GestioneMissione::annullaMissione(Missione missione)* **pre:** self.missioni->includes (missione)
Post-condizioni	*context GestioneMissione::annullaMissione(Missione missione)* **post:** !self.missioni->includes (missione)

Nome metodo	+getMissioniAperte(Paese paese) -> Collection<Missione>
Descrizione	Restituisce la lista delle missioni che soddisfano i criteri di ricerca.
Pre-condizioni	*context GestioneMissione::getMissioniAperte(paese : Paese) : Collection<Missione>* **pre:** paese <> null
Post-condizioni	*context GestioneMissione::getMissioniAperte(paese : Paese) : Collection<Missione>* **post:** result = self.missioni->select(m | m.paese = paese and m.stato = StatoMissione::CONFIRMED)

Nome metodo	+getMissioneById(Long id) -> Missione
Descrizione	Recupera i dettagli di una missione (usato anche prima di una recensione).
Pre-condizioni	*context GestioneMissione::getMissioneById(Long id) -> Missione* **pre:** self.missioni->exists(m|m.id = id)
Post-condizioni	*context GestioneMissione::getMissioneById(Long id) -> Missione* **post:** self.missioni->exists(m|m.id = id and result = m)

Nome metodo	+getMissioniOrganizzatore (Utente organizzatore) -> Collection<Missione>
Descrizione	Recupera le missioni dell'organizzatore per la sua area personale.
Pre-condizioni	*context GestioneMissione::getMissioniOrganizzatore(Utente organizzatore) -> Collection<Missione>* **pre:** self.missioni.organizzatori->includes(organizzatore)
Post-condizioni	*context GestioneMissione::getMissioniOrganizzatore(Utente organizzatore) -> Collection<Missione>* ***post:*** self.missioni->forAll(m| m.organizzatore = organizzatore => result.includes(m))

Nome metodo	+getMissioniPending() -> Collection<Missione>
Descrizione	Restituisce la lista delle missioni che si trovano nello stato pending.
Pre-condizioni	
Post-condizioni	*context GestioneMissione::getMissioniPending() : Collection<Missione>* **post:** result = self.missioni->select(m |m.stato = StatoMissione::PENDING)

2.3 GestioneCandidature

Non sono presenti invarianti.

Nome metodo	+registerCandidatura(Candidatura candidatura)
Descrizione	Crea una nuova istanza di candidatura associando il volontario alla missione scelta.
Pre-condizioni	*context GestioneCandidature::registerCandidatura(Candidatura candidatura)* **pre:** !self.candidature->includes(candidatura)
Post-condizioni	*context GestioneCandidature::registerCandidatura(Candidatura candidatura)* **post:** self.candidature->includes(candidatura) and candidature.stato = StatoCandidatura::PENDING

Nome metodo	+removeCandidatura(Candidatura candidatura)
Descrizione	Elimina (o annulla) una candidatura esistente su richiesta del volontario.
Pre-condizioni	*context GestioneCandidature::removeCandidatura(Candidatura candidatura)* **pre:** self.candidature ->includes(candidatura)
Post-condizioni	*context GestioneCandidature::removeCandidatura(Candidatura candidatura)* **post:** !self.candidature ->includes(candidatura)

Nome metodo	+acceptCandidatura(Candidatura candidatura)
Descrizione	Registra l'approvazione del volontario da parte dell'organizzatore.
Pre-condizioni	*context GestioneCandidature::acceptCandidatura(Candidatura candidatura)* **pre:** self.candidature ->includes(candidatura) and candidatura.stato = StatoCandidatura::PENDING
Post-condizioni	*context GestioneCandidature::acceptCandidatura(Candidatura candidatura)* **post:** self.candidature ->includes(candidatura) and candidatura.stato = StatoCandidatura::CONFIRMED

Nome metodo	+rejectCandidatura(Candidatura candidatura)
Descrizione	Registra il rifiuto della candidatura da parte dell'organizzatore.
Pre-condizioni	*context GestioneCandidature::rejectCandidature(Candidatura candidatura)* **pre:** self.candidature->includes(candidatura) and candidatura.stato = StatoCandidatura::PENDING
Post-condizioni	*context GestioneCandidature::rejectCandidature(Candidatura candidatura)* **post:** candidatura.stato = StatoCandidatura::DENIED

Nome metodo	+getRichiesteCandidatura(Utente organizzatore) -> Collection<Candidature>
Descrizione	Ritorna tutte le candidature relative alle missioni di un organizzatore.
Pre-condizioni	*context GestioneCandidature::getRichiesteCandidatura(Utente organizzatore) - > Collection<Candidature>* **pre:** organizzatore.roles->includes(Role::ORGANIZZATORE)
Post-condizioni	*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)

Nome metodo	+getEsperienzeVolontario(Volontario volontario) -> Collection<Candidature>
Descrizione	Ritorna tutte le candidature accettate del volontario.
Pre-condizioni	*context GestioneCandidature::getEsperienzeVolontario(Volontario volontario) -> Collection<Candidature>* **pre:** self.candidature->exists(c | c.volontario = volontario)
Post-condizioni	*context GestioneCandidature::getEsperienzeVolontario(Volontario volontario)* **post:** result = self.candidature->select(c | c.volontario = volontario and c.stato = StatoCandidatura::CONFIRMED)

Nome metodo	+getCandidatureVolontario(Volontario volontario) -> Collection<Candidature>
Descrizione	Ritorna tutte le candidature del volontario.
Pre-condizioni	*context GestioneCandidature::getCandidatureVolontario(Volontario Volontario) -> Collection<Candidature>* **pre:** self.candidature->exists(c | c.volontario = volontario)
Post-condizioni	*context GestioneCandidature::getCandidatureVolontario(Volontario volontario) -> Collection<Candidature>* ***post:*** result = self.candidature->select(c |c.volontario = volontario)

2.4 GestioneRecensioni

Non sono presenti invarianti.

Nome metodo	+aggiungiRecensioneOrganizzatore(Recensione recensione)
Descrizione	Salva una recensione scritta da un volontario per un organizzatore.
Pre-condizioni	c*ontext GestioneRecensioni::aggiungiRecensioneOrganizzatore(Recensione recensione)* **pre:** !self.recensioni->includes(recensione)
Post-condizioni	*context GestioneRecensioni::aggiungiRecensioneOrganizzatore(Recensione recensione)* **post:** self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC

Nome metodo	+ aggiungiRecensioneVolontario(Recensione recensione)
Descrizione	Salva una recensione scritta da un organizzatore per un volontario.
Pre-condizioni	*context GestioneRecensioni::aggiungiRecensioneVolontario(Recensione recensione)* **pre:** !self.recensioni->includes(recensione)
Post-condizioni	*context GestioneRecensioni::rimuoviRecensione(Recensione recensione)* **post:** !self.recensioni->includes(recensione)

Nome metodo	+ rimuoviRecensione(Recensione recensione)
Descrizione	Elimina una recensione segnalata ritenuta inappropriata dal moderatore.
Pre-condizioni	*context GestioneRecensioni::rimuoviRecensione(Recensione recensione)* **pre:** self.recensioni->includes(recensione)
Post-condizioni	*context GestioneRecensioni::aggiungiRecensioneVolontario(Recensione recensione)* **post:** self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC

Nome metodo	+ approvaRecensione(Recensione recensione)
Descrizione	Conferma la validità di una recensione segnalata, mantenendola pubblicata.
Pre-condizioni	*context GestioneRecensioni::approvaRecensione(Recensione recensione)* **pre:** self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::REPORTED
Post-condizioni	*context GestioneRecensioni::approvaRecensione(Recensione recensione)* **post:** self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC

Nome metodo	+ segnalaRecensione(Recensione recensione)
Descrizione	Segnala una recensione.
Pre-condizioni	*context GestioneRecensioni::segnalaRecensione(Recensione recensione)* **pre:** self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::PUBLIC
Post-condizioni	*GestioneRecensioni::segnalaRecensione(Recensione recensione)* **post:** self.recensioni->includes(recensione) and recensione.stato = StatoRecensione::REPORTED

2.5 GestioneEmail

Non sono presenti invarianti e postcondizioni.

Nome metodo	+inviaConferma(Long id, String token)
Descrizione	Invia link di attivazione account.
Pre-condizioni	*context GestioneEmail::inviaConferma(id:Long, token : String)* **pre**: self.utenti->exists(u | u.id = id and u.stato = StatoUtente::PENDING)

Nome metodo	+inviaEmailRecupero(String email, String token)
Descrizione	Invia credenziali per reset password.
Pre-condizioni	*context GestioneEmail::inviaEmailRecupero(email: String, token : String)* **pre**: self.utenti->exists(u | u.email = email)

Nome metodo	+inviaEmailCandidatura(Long id)
Descrizione	Invia nuova candidatura dell'organizzatore.
Pre-condizioni	*context GestioneEmail::inviaEmailCandidatura(id : Long)* **pre**: self.candidature->exists(c | c.id = id and c.missione.organizzatore <> null)

Nome metodo	+inviaEmailAnnullamentoCandidatura(Long id)
Descrizione	Notifica annullamento candidatura (dal volontario).
Pre-condizioni	*context GestioneEmail::inviaEmailAnnullamentoCandidatura(id : Long)* **pre**: self.candidature->exists(c | c.id = id and c.missione.organizzatore <> null)

Nome metodo	+ inviaEmailAccettazioneCandidatura (Long id)
Descrizione	Notifica accettazione candidatura.
Pre-condizioni	*context GestioneEmail::inviaEmailAccettazioneCandidatura(id : Long)* **pre**: self.candidature->exists(c | c.id = id)

Nome metodo	+ inviaEmailRifiutoCandidatura (Long id)
Descrizione	Notifica rifiuto candidatura.
Pre-condizioni	*context GestioneEmail::inviaEmailRifiutoCandidatura(id : Long)* **pre**: self.candidature->exists(c | c.id = id)

Nome metodo	+ inviaEmailApprovazioneMissione(Long id)
Descrizione	Notifica approvazione missione.
Pre-condizioni	*context GestioneEmail::inviaEmailApprovazioneMissione(id : Long)* **pre**: self.missioni->exists(m | m.id = id and m.stato = StatoMissione::CONFIRMED and m.organizzatore <> null)

Nome metodo	+ inviaEmailViolazioneRecensione(Long id)
Descrizione	Notifica rimozione recensione.
Pre-condizioni	*context GestioneEmail::inviaEmailViolazioneRecensione(id : Long)* **pre**: self.recensioni->exists(r | r.id = id and r.utente <> null)

Nome metodo	+ inviaEmailCreazioneModeratore(Long id)
Descrizione	Invia credenziali al nuovo moderatore creato.
Pre-condizioni	*context GestioneEmail::inviaEmailCreazioneModeratore(id : Long)* **pre**: self.utenti->exists(u | u.id = id and u.oclIsKindOf(Moderatore))