Università degli studi di Salerno

Corso di laurea in Informatica

AskToReply

(Object Design Document)

Progetto IS - A.A. 2020/21

Membri del progetto

Cognome	Nome	Matricola
Bellogrado	Vincenzo	0512105336
Di Benedetto	Carmine	0512105186
Memoli	Gianluigi	0512105204

Indice

Introduzione

Trade-off

In questo documento saranno trattati gli aspetti relativi all'implementazione del sistema, in particolare quei dettagli che nell'analisi non sono stati trattati perché di basso livello.
Verranno definiti trade-off e linee guida relativi all'implementazione e successivamente approfonditi i particolari delle interfacce di classe, approfondendo le signature, operazioni, tipi e argomenti dei sottosistemi presentati nel SDD.

Comprensibilità vs Tempo

Per aumentare la leggibilità e facilitarne la manutenzione si è data notevole importanza alla formattazione e ai commenti del codice sorgente. Si preferiscono implementazioni semplici.
Ciò può aumentare le tempistiche di stesura del codice facendo però guadagnare tempo successivamente durante eventuali aggiornamenti e/o correzioni.

Prestazioni vs Costi

Sono state utilizzate alcune componenti off-the-shelf per abbattere i costi di sviluppo, che però potrebbero minare le performance del sistema in quanto forniscono anche delle funzionalità che non vengono utilizzate appesantendo dunque l'esecuzione del codice.

Interfaccia vs Usabilità

L'interfaccia è stata progettata cercando di mantenere il numero di componenti grafico quanto più basso possibile in modo da renderla minimale e quindi più facile da utilizzare. Inoltre, l'interfaccia grafica è stata realizzata tenendo conto di dispositivi di forma diversa e dimensioni inferiori rispetto ad un comune schermo.

Sicurezza vs Efficienza

A causa del poco tempo a disposizione per la stesura del codice, sono stati inseriti solo ed esclusivamente i controlli di sicurezza lato server mentre per quanto riguarda le prestazioni, il codice non è stato ottimizzato perché è stata data priorità alla sicurezza.

Off-the-shelf components vs Implementazione

Per evitare in realizzare componenti di cui implementazione e testing sarebbero state attività estremamente costose in termini di tempo, si è preferito andare ad utilizzare implementazioni di terzi, già ampiamente testate ed altamente integrabili nella codebase, quali:

- Bootstrap 4, noto framework per il front-end, che fornisce una gestione estremamente semplificata e performante dei layout responsive. Inoltre, fornisce un'ampia suite di componenti per la creazione di UI in generale (buttons, form elements, …)
- Apache Tomcat JDBC Datasource, è stato utilizzato per sfruttare una connection pool ampiamente testata ed evitare di implementarne una da zero.

Linee guida per documentazione delle interfacce

Naming Convention

I nomi utilizzati per variabili, metodi e classi devono essere:

- semplici
- pronunciabili
- descrittivi
- ricercabili

Gli identificatori delle costanti saranno in maiuscolo, se sono composti separati da underscore.

Design pattern

I design pattern che si ritrovano nel progetto sono:

- Singleton, lo ritroviamo del DBManager, dato che è necessario avere una sola istanza attiva di quella classe nel sistema
- Facade, sono tutte le classi che terminano con "Manager" ad eccezione di DBManager. Questi classi facade orchestrano l'accesso ai metodi offerti dai vari DAO e gestiscono le relazione tra le entità chiamando metodi di diversi DAO.

Definizioni, acronimi, abbreviazioni

Definizioni

- Open Source: Software il cui codice sorgente è accessibile a tutti
- Off-the-shelf component: Componenti che forniscono servizi precisi

Acronimi

- SDD: System Design Document
- UI: User Interface
- JDBC: Java DataBase Connectivity

Riferimenti

Il materiale utilizzato come riferimento è:

- Object-Oriented Software Engineering: Using UML, Patterns and Java
- Slide fornite dal prof. A. De Lucia dell'università degli studi di Salerno

Packages

Il sistema è suddiviso in tre layer:

Presentation Layer

E' l'insieme di interfacce grafiche che consentono all'utente di interagire col sistema.

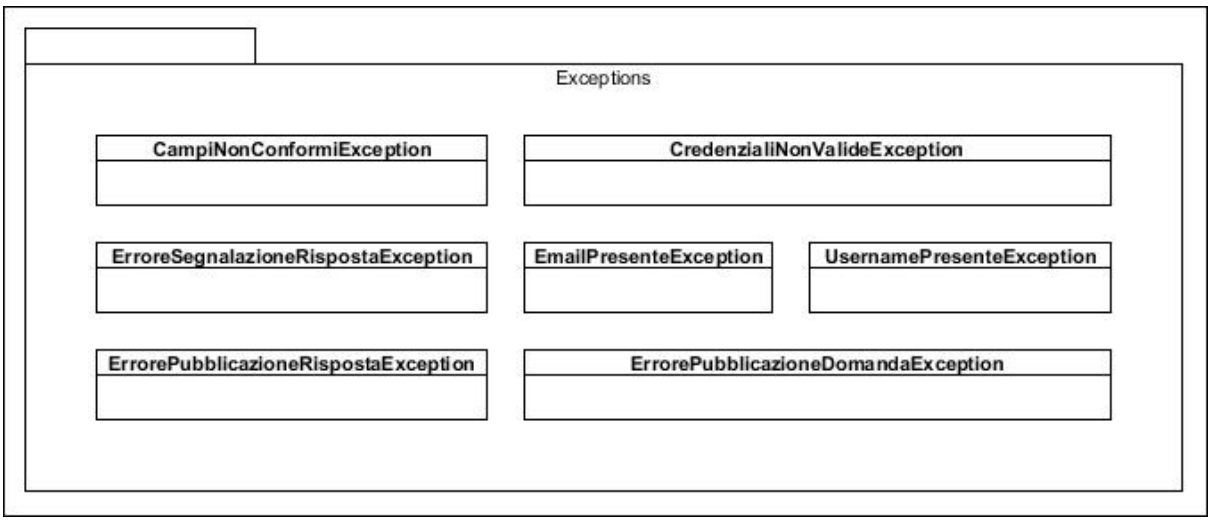
Business Layer

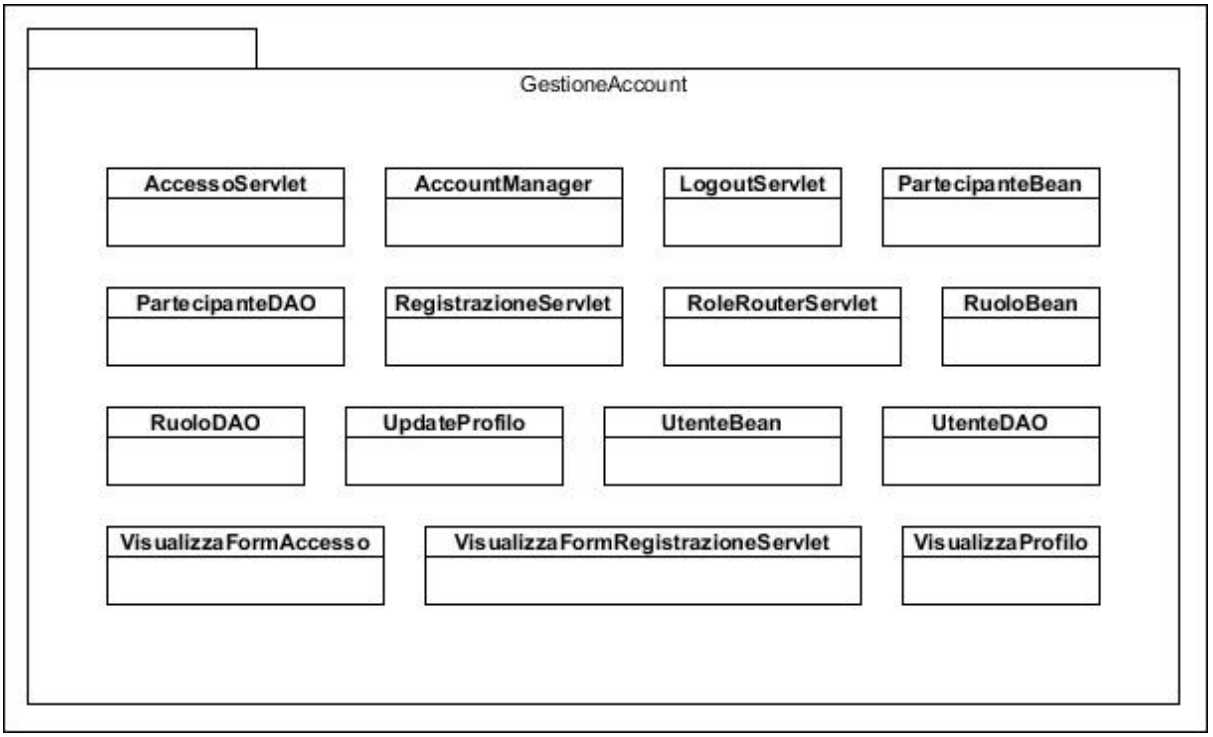
E' il livello responsabile della logica del sistema. Svolge varie funzioni raggruppate nelle seguenti gestioni:

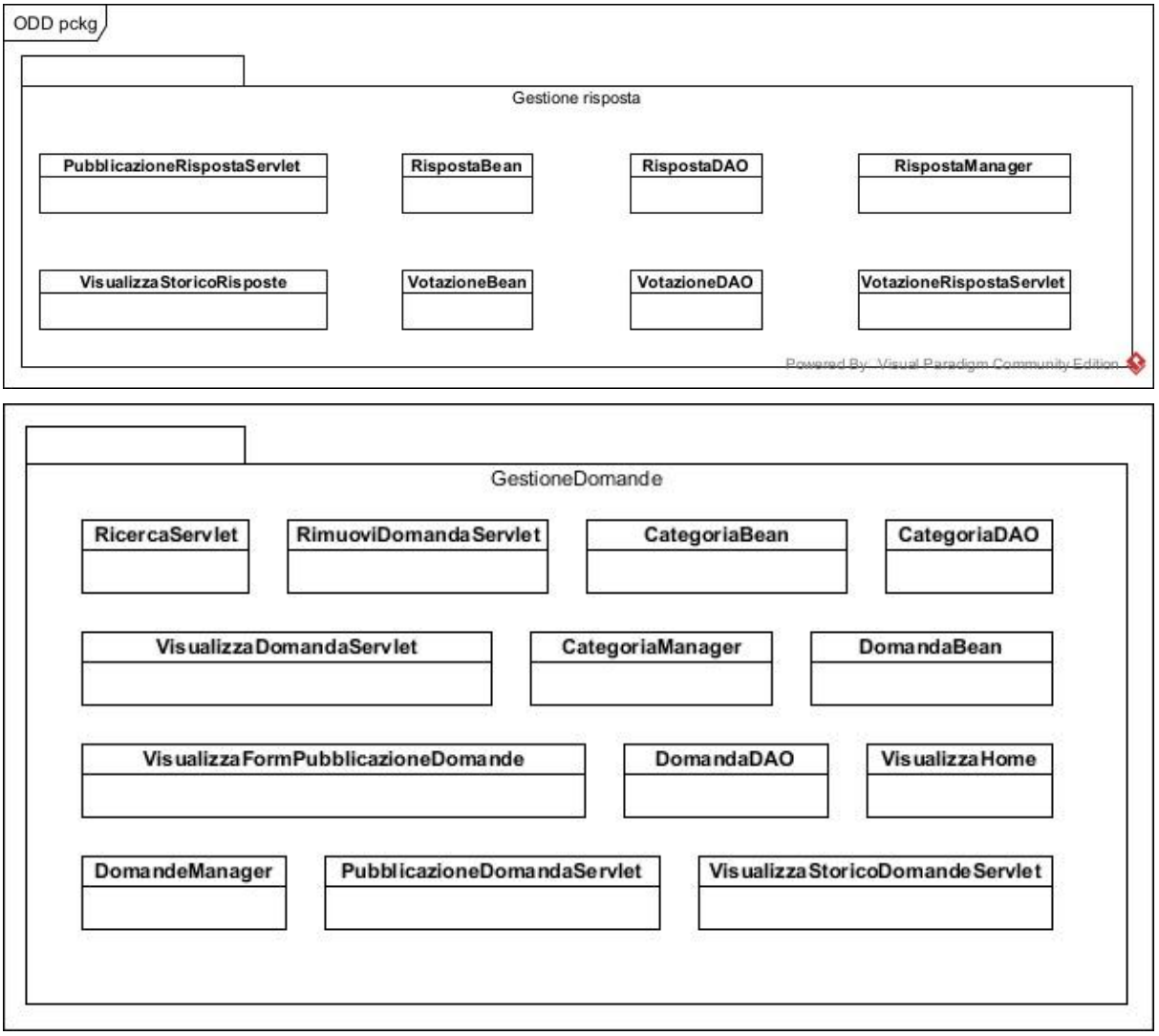
- **Gestione Account:** fornisce funzionalità di registrazione/accesso alla piattaforma ed altre operazioni relative agli account registrati.
- **Gestione Domanda:** si occupa delle funzionalità riguardanti le domande postate dagli utenti.
- **Gestione Risposta:** si occupa delle funzionalità legate alle risposte inviate dagli utenti.
- **Gestione Moderazione:** si occupa delle funzionalità che coinvolgono moderatori e master moderatori.

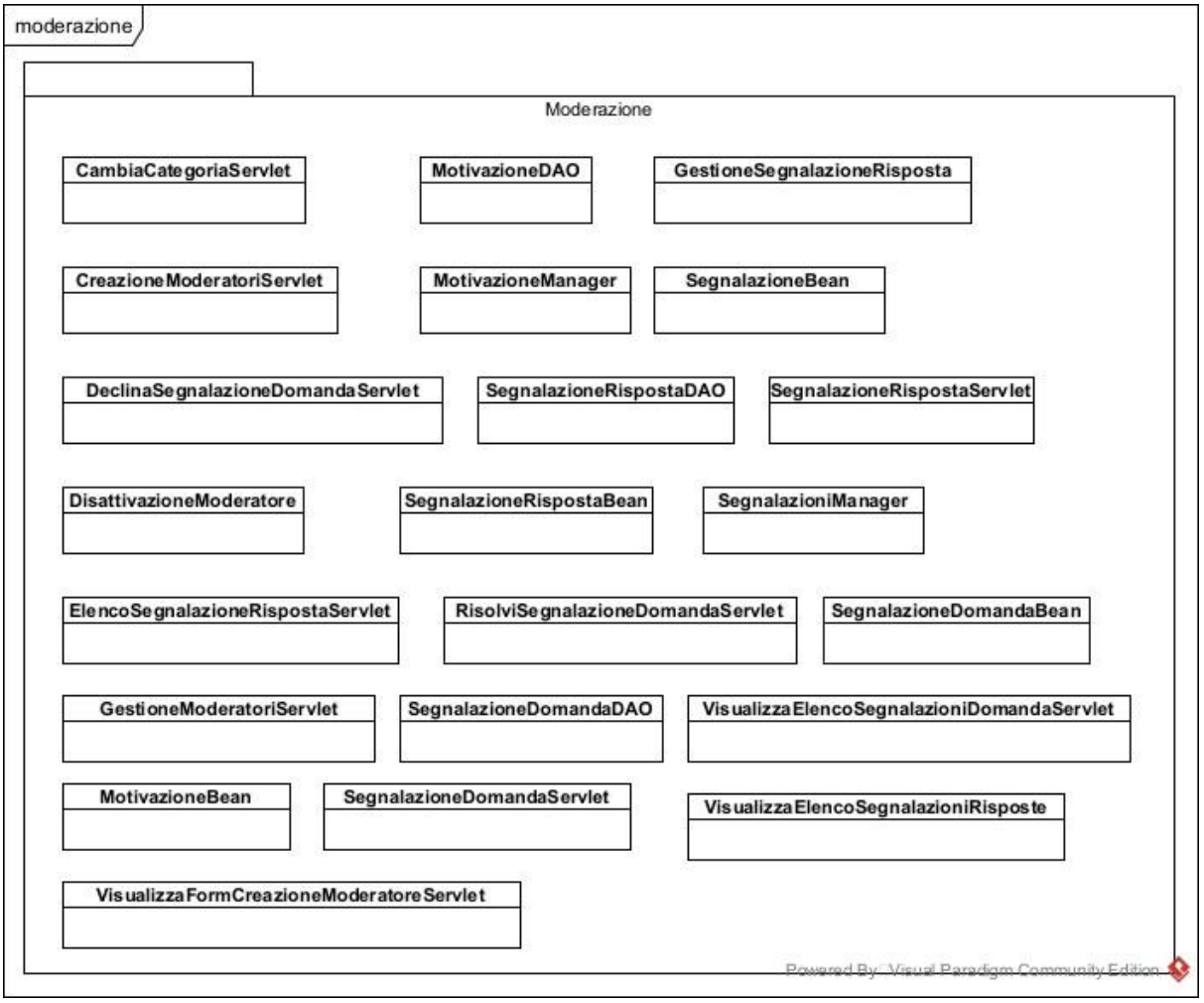
Storage Layer

E' il livello che si occupa di memorizzare e recuperare dati. Memorizza gli oggetti persistenti su un RDBMS.


Exceptions
CampiNonConformiException
CredenzialiNonValideException
ErroreSegnalazioneRispostaException
EmailPresenteException
UsernamePresenteException
ErrorePubblicazioneRispostaException
ErrorePubblicazioneDomandaException


GestioneAccount
AccessoServlet
AccountManager
LogoutServlet
PartecipanteBean
PartecipanteDAO
RegistrazioneServlet
RoleRouterServlet
RuoloBean
RuoloDAO
UpdateProfilo
UtenteBean
UtenteDAO
VisualizzaFormAccesso
VisualizzaFormRegistrazioneServlet
VisualizzaProfilo


ODD pckg
Gestione risposta
PubblicazioneRispostaServlet
RispostaBean
RispostaDAO
RispostaManager
VisualizzaStoricoRisposte
VotazioneBean
VotazioneDAO
VotazioneRispostaServlet
Powered By Visual Paradigm Community Edition
GestioneDomande
RicercaServlet
RimuoviDomandaServlet
CategoriaBean
CategoriaDAO
VisualizzaDomandaServlet
CategoriaManager
DomandaBean
VisualizzaFormPubblicazioneDomande
DomandaDAO
VisualizzaHome
DomandeManager
PubblicazioneDomandaServlet
VisualizzaStoricoDomandeServlet


moderazione
Moderazione
CambiaCategoriaServlet
MotivazioneDAO
GestioneSegnalazioneRisposta
CreazioneModeratoriServlet
MotivazioneManager
SegnalazioneBean
DeclinaSegnalazioneDomandaServlet
SegnalazioneRispostaDAO
SegnalazioneRispostaServlet
DisattivazioneModeratore
SegnalazioneRispostaBean
SegnalazioniManager
ElencoSegnalazioneRispostaServlet
RisolviSegnalazioneDomandaServlet
SegnalazioneDomandaBean
GestioneModeratoriServlet
SegnalazioneDomandaDAO
VisualizzaElencoSegnalazioniDomandaServlet
MotivazioneBean
SegnalazioneDomandaServlet
VisualizzaElencoSegnalazioniRisposte
VisualizzaFormCreazioneModeratoreServlet
Powered By: Visual Paradigm Community Edition

Class interfaces

Util

Classe	**AllegatiHandler**
Descrizione	*Questa classe si occupa del caricamento degli allegati quando viene pubblicata una domanda o una risposta e del recupero degli allegati quando tali domande e risposte vengono visualizzate.*
Signature dei metodi	+ caricaAllegati(allegati : List<Part>, destinationFolder : String) : void + getAllegati(folder : String) : File[] + convertToBase64(allegati : File[]) : ArrayList<String>
Precondizioni	**context** AllegatiHandler :: caricaAllegati(allegati : List<Part>, destinationFolder : String) **pre:** allegati <> null and allegati.size() <= AllegatiHandler.NUMERO_MAX_ALLEGATI and allegati->forAll(a | a.getSize() <= AllegatiHandler.DIMENSIONE_MAX_ALLEGATO) and allegati->forAll(a | a.getContentType = "image/gif" or a.getContentType = "image/jpeg" or a.getContentType = "image/png" or a.getContentType = "image/bmp" or a.getContentType = "image/tiff") and destinationFolder <> null **context** AllegatiHandler :: getAllegati(folder : String) **pre:** folder <> null **context** AllegatiHandler :: convertToBase64(allegati : File[]) **pre:** allegati <> null
Postcondizioni	**context** AllegatiHandler :: caricaAllegati(allegati :List<Part>, destinationFolder : String) **post:** fileSystem->exists(select(folder | folder.path = destinationFolder and folder.includes(allegati)) **context** AllegatiHandler :: getAllegati(folder : String) **post:** fileSystem->select(folder | folder.path = folder).asSet() **context** AllegatiHandler :: convertToBase64(allegati : File[]) **post:** Converte ‘allegati’ in un array di String e lo restituisce

Classe	**DBManager**

Descrizione	*Classe che gestisce l'accesso al database*
Signature dei metodi	+ getInstance() : DBManager - getSQLConnection() : Connection + prepareStoredProcedureCall(procedureName : String, parametersAmount : int) : CallableStatement + createPreparedStatement(query : String) : PreparedStatement + executeFromScript(filePath : String) : void
Precondizioni	
Postcondizioni	**context** DBManager :: getInstance() **post:** DBManager.getIstance() <> null **context** DBManager :: getSQLConnection() **post:** Restituisce un oggetto Connection dalla connection pool **context** DBManager :: prepareStoredProcedureCall(procedureName, parametersAmount) **post:** Restituisce un oggetto Connection dalla connection pool **context** DBManager :: createPreparedStatement() **post:** Restituisce un oggetto PreparedStatement

Classe	**CustomServlet**
Descrizione	*Questa classe è una classe che contiene metodi che verificano se un particolare utente (partecipante, moderatore e master moderatore) ha effettuato il login. Le classi che hanno bisogno di questi metodi, estendono questa classe invece di HttpServlet.*
Signature dei metodi	+ getLoggedUser(session : HttpSession) : UtenteBean + isUserLogged(session : HttpSession) : boolean - isUserRoleLogged(session : HttpSession, role : String) : boolean + isPartecipanteLogged(session : HttpSession) : boolean + isModeratoreLogged(session : HttpSession) : boolean + isMasterModeratoreLogged(session : HttpSession) : boolean + checkPartecipante(session : HttpSession, resp : HttpServletResponse) : void + checkModeratore(session : HttpSession, resp : HttpServletResponse) : void + checkMasterModeratore(session : HttpSession, resp : HttpServletResponse) :void

Precondizioni	**context** CustomServlet :: isUserRoleLogged(session, role) **pre:** role <> null **context** CustomServlet :: checkPartecipante(session, resp) **pre:** session.getAttribute(“utenteLoggato”).getRuolo().getId() = RuoloBean.ROLE_PARTECIPANTE **context** CustomServlet :: checkModeratore(session, resp) **pre:** session.getAttribute(“utenteLoggato”).getRuolo().getId() = RuoloBean.ROLE_MODERATORE **context** CustomServlet :: checkMasterModeratore(session, resp) **pre:** session.getAttribute(“utenteLoggato”).getRuolo().getId() = RuoloBean.ROLE_MASTER_MODERATORE
Postcondizioni	**context** CustomServlet :: getLoggedUser(session) **post:** request.getAttribute(“utenteLoggato”) **context** CustomServlet :: isUserLogged(session) **post:** true if request.getAttribute(“utenteLoggato”) <> null **context** CustomServlet :: isUserRoleLogged(session, role) **post:** true if request.getAttribute(“utenteLoggato”) <> null and request.getAttribute(“utenteLoggato”).getRuolo().getId() = RuoloDAO.getRuoloByNome(role).getId() **context** CustomServlet :: isPartecipanteLogged(session) **post:** true if request.getAttribute(“utenteLoggato”) <> null and request.getAttribute(“utenteLoggato”).getRuolo().getId() = RuoloBean.ROLE_PARTECIPANTE **context** CustomServlet :: isModeratoreLogged(session) **post:** true if request.getAttribute(“utenteLoggato”) <> null and request.getAttribute(“utenteLoggato”).getRuolo().getId() = RuoloBean.ROLE_MODERATORE **context** CustomServlet :: isMasterModeratoreLogged(session) **post:** true if request.getAttribute(“utenteLoggato”) <> null and request.getAttribute(“utenteLoggato”).getRuolo().getId() = RuoloBean.ROLE_MASTER_MODERATORE

Gestione domanda

Classe	CategoriaBean
Descrizione	**Classe che rappresenta l'oggetto Categoria, che può essere associato a Domanda e Partecipante**
Signature dei metodi	+getId(): String +getNome(): String +setId(id:String): void +setNome(nome:String): void
Precondizioni	
Postcondizioni	

Classe	CategoriaDAO
Descrizione	**La classe si occupa di realizzare l'accesso alla tabella "Categorie",**
Signature dei metodi	+addCategoria(categoria: CategoriaBean) +getAll(): Collection +getCategoriaByNome(nome: String): CategoriaBean +getCategorieByUtente(id: String): Collection +getCategorieDomandaByIdDomanda(id: String): Collection
Precondizioni	**context** CategoriaDAO::addCategoria(categoria) **pre** categoria != null
Postcondizioni	**context** CategoriaDAO::addCategoria(categoria) **post**database.categorie->includes(categoria)

Classe	VisualizzaHome
Descrizione	**Servlet che fa redirect alla home di Partecipante, in cui visualizza tutte le domande attinenti ai suoi interessi**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** doPost(request, response) **pre:** request.getSession().getAttribute("utenteLoggato").getRuolo().getNome() = "Parteciapante"
Postcondizioni	

Classe	**PubblicazioneDomandaServlet**

Descrizione	*Controller che si occupa di gestire l'inserimento di una domanda nel db e di reindirizzare l'utente alla domanda pubblicata*
Signature dei metodi	+ service(req : HttpServletRequest , resp : HttpServletResponse) : void + doGet(request : HttpServletRequest , response HttpServletResponse) : void + doPost(request : HttpServletRequest , response HttpServletResponse) : void
Precondizioni	**context:** PubblicazioneDomandaServlet:: service(req : HttpServletRequest , resp : HttpServletResponse) **pre:** request.getSession().getAttribute("utenteLoggato").getRuolo().getNome() = "Partecipante"
Postcondizioni	**context:** PubblicazioneDomandaServlet :: doPost(req : HttpServletRequest , resp : HttpServletResponse) **post:** database.domande->includes(d | d.id = request.getParameter("id"))

Classe	**VisualizzaDomandaServlet**
Descrizione	*Controller che si occupa di reindirizzare l'utente alla pagina che mostra la domanda che l'utente ha aperto*
Signature dei metodi	+ service(req : HttpServletRequest , resp : HttpServletResponse) : void + doGet(request : HttpServletRequest , response : HttpServletResponse) : void + doPost(request : HttpServletRequest , response HttpServletResponse) : void
Precondizioni	**context:** VisualizzaDomandaServlet :: service(req : HttpServletRequest , resp : HttpServletResponse) **pre:** request.getSession().getAttribute("utenteLoggato").getRuolo().getNome() = "Partecipante"
Postcondizioni	**context:** VisualizzaDomandaServlet :: doGet (request HttpServletRequest, response : HttpServletResponse) **post:** request.getAttribute("domanda")<>null AND request.getAttribute("domandeRisposte")<>null AND request.getAttribute("motivazioni")<>null AND request.getAttribute("risposteApprezzate")<>null AND request.getAttribute("risposteNonApprezzate")<>null AND

	request.getAttribute(“next”)<>null AND request.getAttribute(“risposte”)<>null AND request.getAttribute(“utenteLoggato”)<>null

Classe	**CategorieManager**
Descrizione	*Questa classe si occupa del recupero dal database di tutte le categorie.*
Signature dei metodi	+getAll() : ArrayList<CategoriaBean>
Precondizioni	
Postcondizioni	**context** CategoriaManager :: getAll() **post:** database->categorie

Classe	**DomandaDAO**
Descrizione	*Questa classe si occupa dell’inserimento, recupero, modifica e cancellazione delle domande nel database.*
Signature dei metodi	+ addDomanda(domanda : DomandaBean) : DomandaBean + addCategoriaDomanda(domanda : DomandaBean, categoria : CategoriaBean) + updateCategorieDomanda(domanda : DomandaBean) : void + getDomandeRisposte(idUtente : String) : ArrayList<DomandaBean> + getDomandeByUtente(idUtente : String, start : int, end : int) : ArrayList<DomandaBean> + getDomandaById(id : String) : DomandaBean + getDomandeCercate(testo : String, isArchiviata : Boolean, categorie : String[]) : ArrayList<DomandaBean> + removeDomanda (idDomanda : String) : void + getDomandeRecenti(start : int, end : int) : ArrayList<DomandaBean> + getNumeroDomandePertinenti(categorie : ArrayList<CategoriaBean>) : int + getDomandePertinenti(categorie : ArrayList<CategoriaBean>, page : int, offset : int) : ArrayList<DomandaBean> + getNumeroDomandeByAutore(idAutore : String) : int
Precondizioni	**context** DomandaDAO :: addDomanda(domanda) **pre:** domanda <> null **context** DomandaDAO :: addCategoriaDomanda(domanda, categoria) **pre:** domanda <> null and categoria <> null

	context DomandaDAO :: updateCategorieDomanda(domanda) **pre:** domanda <> null **context** DomandaDAO :: getDomandeRisposte(idUtente) **pre:** idUtente <> null **context** DomandaDAO :: getDomandeByUtente(idUtente, start, end) **pre:** idUtente <> null and start < end **context** DomandaDAO :: getDomandaById(id) **pre:** id <> null **context** DomandaDAO :: removeDomanda (idDomanda) **pre:** idDomanda <> null **context** DomandaDAO :: getDomandeRecenti(start, end) **pre:** start <= end **context** DomandaDAO :: getNumeroDomandePertinenti(categorie) **pre:** categorie <> null **context** DomandaDAO :: getDomandePertinenti(categorie, page, offset) **pre:** categorie <> null **context** DomandaDAO :: getNumeroDomandeByAutore(idAutore) **pre:** idAutore <> null
Postcondizioni	**context** DomandaDAO :: addDomanda(domanda) **post:** database.domande->includes(select(d | d.id = domanda.getId()) **context** DomandaDAO :: addCategoriaDomanda(domanda, categoria) **post:** database.categoriedomande->includes(select(cd | cd.idDomanda = domanda.getId() and cd.idCategoria = categoria.getId()) **context** DomandaDAO :: updateCategorieDomanda(domanda) **post:** @pre.domanda.getCategorie() <> @post.domanda.getCategorie() **context** DomandaDAO :: getDomandeRisposte(idUtente) **post:** database.domande->select(d | database.risposte->exists(r | r.getAutore().getid() = idUtente and r.getDomanda().getId() = d.getId()) **context** DomandaDAO :: getDomandeByUtente(idUtente, start, end)

	post: database.domande->select(d | domande.autore.id = idUtente and domande.isNascosta = false) **context** DomandaDAO :: getDomandaById(id) **post:** database.domande->select(d | domande.id = d and isNascosta = false) **context** DomandaDAO :: removeDomanda (idDomanda) **post:** database.domande->exists(d | domande.id = idDomanda and domande.isNascosta = true) **context** DomandaDAO :: getDomandeRecenti(start, end) **post:** **context** DomandaDAO :: getNumeroDomandePertinenti(categorie) **post:** database.domande->select(d | categorie->exists(c | database->categoriedomande.exists(c.getId)).size() **context** DomandaDAO :: getDomandePertinenti(categorie, page, offset) **post:** **context** DomandaDAO :: getNumeroDomandeByAutore(idAutore) **post:**

Classe	**DomandaBean**
Descrizione	*Questa classe rappresenta l'oggetto domanda.*
Signature dei metodi	+ getId() : String + setDataPubblicazione(dataPubblicazione : Date) : void + getDataPubblicazione() : Date + setId(id : String) : void + setAllegati(allegati : ArrayLis<String>t) : void + setCategorie(categorie : ArrayList<CategoriaBean>) : void + getCorpo() : String + setTitolo(titolo : String) : void + setAutore(autore : UtenteBean) : void + setRisposte(risposte : ArrayList<RispostaBean>) : void + isArchiviata() : boolean + getRisposte() : ArrayList<RispostaBean> + getAllegati() : ArrayList<String> + getAutore() : UtenteBean + getCategorie() : ArrayList <CaetgoriaBean> + setCorpo(corpo : String) : void + getTitolo() : String + setArchiviata(isArchiviata : boolean) : void
Precondizioni	

Postcondizioni	

Classe	**DomandeManager**
Descrizione	*Questa classe implementa il Design Pattern Facade e gestisce l'entita domanda.*
Signature dei metodi	+ pubblicaDomanda(autore :AutoreBean, titolo : String, corpo : String, dataPubblicazione : Date, idCategorie : String[], allegati : List<Part>) : DomandaBean + ricerca(testo : String, categorie : String[], isArchiviata : Boolean) : ArrayList<DomandaBean> + getDomandaById(idDomanda : String) : DomandaBean + getDomandeByAutore(idAutore : String, start : int, end : int) : ArrayList<DomandaBean> + removeDomanda(idDomanda : String) : void + updateCategorieDomanda(domanda : DomandaBean) : void - DomandaBean populateAutore(domanda : DomandaBean) : DomandaBean - DomandaBean populateCategorie(domanda : DomandaBean) : DomandaBean - DomandaBean populateReferencedEntities(domanda : DomandaBean) : DomandaBean + DomandaBean populateAllegati(domanda : DomandaBean) : DomandaBean + getDomandeRecenti(start : int, end : int) : ArrayList<DomandaBean> + getDomandePertinenti(utente : PartecipanteBean, page : int, offset : int) : ArrayList<DomandaBean> + getNumOfDomandePertinenti(utente : PartecipanteBean) : int + getNumeroDomandeByAutore(idAutore : String) : int + getDomandeRisposte(utente : PartecipanteBean) : HashSet<DomandaBean>
Precondizioni	**context** DomandeManager :: pubblicaDomanda(autore, titolo, corpo, dataPubblicazione, idCategorie, allegati) **pre:** titolo <> null and titolo.trim.length() >= 5 and titolo.trim.length() <= 90 and corpo <> null and if allegati.size = 0 corpo.trim.length >= 5 and corpo.trim.lenght <= 250 and if allegati.size() <> 0 and corpo.trim.lenght() <> 0 then corpo.trim.length >= 5 and corpo.trim.lenght <= 250 and idCategorie <> null and idCategorie.lenght > 0 and allegati.size() <= 5 and allegati->forAll(a | allegati.getSize <= 5MB) and allegati->forAll(a | allegati.getContentType() = "image/gif" or allegati.getContentType() = "image/jpeg" or allegati.getContentType() = "image/png" or allegati.getContentType() = "image/bmp" or allegati.getContentType() = "image/tiff") **context** DomandeManager :: ricerca(testo, categorie, isArchiviata)

	pre: if testo <> null then testo >= 3 **context** DomandeManager :: getDomandaById(idDomanda) **pre:** idDomanda <> null **context** DomandeManager :: getDomandeByAutore(idAutore, start, end) **pre:** idAutore <> null **context** DomandeManager :: removeDomanda(idDomanda) **pre:** idDomanda <> null **context** DomandeManager :: updateCategorieDomanda(domanda) **pre:** domanda <> null **context** DomandeManager :: populateAutore(domanda) **pre:** domanda <> null **context** DomandeManager :: populateCategorie(domanda) **pre:** domanda <> null **context** DomandeManager :: populateReferencedEntities(domanda) **pre:** domanda <> null **context** DomandeManager :: populateAllegati(domanda) **pre:** domanda <> null **context** DomandeManager :: getDomandeRecenti(start, end) **pre:** **context** DomandeManager :: getDomandePertinenti(utente, page, offset) **pre:** utente <> null **context** DomandeManager :: getNumOfDomandePertinenti(utente) **pre:** utente <> null **context** DomandeManager :: getNumeroDomandeByAutore(idAutore) **pre:** idAutore <> null **context** DomandeManager :: getDomandeRisposte(utente) **pre:** utente <> null
Postcondizioni	**context** DomandeManager :: pubblicaDomanda(autore, titolo, corpo, dataPubblicazione, idCategorie, allegati) **post:** database.domande->exists(d | d.autore = autore and d.titolo = titolo and d.corpo = corpo and d.dataPubblicazione = dataPubblicazione and d.categorie->forAll(c | idCategorie->includes(c.getId())

	context DomandeManager :: getDomandaById(idDomanda) **post:** database.domande->select(d | domande.id = idDomanda) **context** DomandeManager :: getDomandeByAutore(idAutore, start, end) **post:** database.domande->select(d | domande.autore.id = idAutore) and database.domande->select(d | domande.autore.id = idAutore).size() = (end - start) **context** DomandeManager :: removeDomanda(idDomanda) **post:** database.domande->exists(d | domande.id = idDomanda and domande.isNascosta = true) **context** DomandeManager :: updateCategorieDomanda(domanda) **post:** @pre.domanda.categorie <> @post.domanda.categorie **context** DomandeManager :: getDomandeRecenti(start, end) **post:** **context** DomandeManager :: getDomandePertinenti(utente, page, offset) **post:** database.domande->select(d | d.getCategorie.exists(utente,getInteressi()) and d.isNascosta = false) **context** DomandeManager :: getNumOfDomandePertinenti(utente) **post:** database.domande->select(d | d.getCategorie = utente,getInteressi() and d.isNascosta = false).size() **context** DomandeManager :: getDomandeRisposte(utente) **post:** DomandaDAO.getDomandeRisposte(utenti)

Classe	**RicercaServlet**
Descrizione	*Servlet che restituisce un elenco di domande in base a determinati filtri compilati dall'utente.*
Signature dei metodi	# doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** RicercaServlet :: doGet(request, response) **pre:** if request<> null then testo.length >= 3
Postcondizioni	**context** RicercaServlet :: doGet(request, response) **post:** request.getAttribute("risultatoRicerca") = managerDomande.ricerca(testo, categorieDomanda, isArchiviata)

Classe	**RimuoviDomandaServlet**
Descrizione	*Servlet che permette ad un utente che ha pubblicato una domanda di rimuoverla.*
Signature dei metodi	# service(req : HttpServletRequest, resp : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void # doPost(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** RimuoviDomandaServlet :: service(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”) <> null and request.getSession().getAttribute(“utenteLoggato”).getRuolo = RuoloBean.ROLE_PARTECIPANTE **context** RimuoviDomandaServlet :: doGet(request, response) **pre:** request.getParameter("idDomanda") <> null and domandeManager.getDomandeByid(request.getParameter("idDomanda")) <> null and domandeManager.getDomandeByid(request.getParameter("idDomanda")).getAutore().getId() = request.getSession(“utenteLoggato”).getId()
Postcondizioni	**context** RimuoviDomandaServlet :: doGet(request, response) **post:** domandeManager.getDomandaById(request.getParameter(“idDomanda”)) = null

Classe	**VisualizzaFormPubblicazioneDomandaServlet**
Descrizione	*Servlet che mostra ad un partecipante un form per pubblicare una domanda.*
Signature dei metodi	# service(req : HttpServletRequest, resp : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void # doPost(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** VisualizzaFormPubblicazioneDomandaServlet :: service(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”) <> null and request.getSession().getAttribute(“utenteLoggato”).getRuolo = RuoloBean.ROLE_PARTECIPANTE
Postcondizioni	

Classe	**VisualizzaStoricoDomandeServlet**
Descrizione	*Servlet che permette ad un utenteloggato di visualizzare l'elenco di domande che ha pubblicato.*
Signature dei metodi	# service(req : HttpServletRequest, resp : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void # doPost(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** VisualizzaStoricoDomandeServlet :: service(request, response) **pre:** request.getSession().getAttribute("utenteLoggato") <> null and request.getSession().getAttribute("utenteLoggato").getRuolo = RuoloBean.ROLE_PARTECIPANTE
Postcondizioni	**context** VisualizzaStoricoDomandeServlet :: doGet(request, response) **post:**

Gestione Risposta

Classe	**VotazioneDAO**
Descrizione	*Classe che si occupa di realizzare l'accesso alla tabella "Votazioni"*
Signature dei metodi	+ addVotazioneRisposta(votazione : VotazioneBean) : void + removeVotazioneRisposta(votazione : VotazioneBean) : void + getVotazioniByIdRisposta(idRisposta : String) : ArrayList<VotazioneBean>
Precondizioni	**context** VotazioneDAO :: addVotazioneRisposta(votazione : VotazioneBean) **pre:** votazione <> null **context** VotazioneDAO :: removeVotazioneRisposta(votazione : VotazioneBean) **pre:** votazione <> null **context** VotazioneDAO :: getVotazioniByIdR(votazione : VotazioneBean) **pre:** votazione <> null
Postcondizioni	**context** VotazioneDAO :: addVotazioneRisposta(votazione : VotazioneBean) **post:** database.votazioni->exists(v|v.idUtente = votazione.getUtente().getId() AND v.risposta = votazione.getRisposta().getId()) **context** VotazioneDAO :: removeVotazioneRisposta(votazione : VotazioneBean) **post:** database.votazioni->select(v|v.idUtente = votazione.getUtente().getId() AND v.risposta = votazione.getRisposta().getId())->isEmpty()

Classe	**VotazioneRispostaServlet**
Descrizione	*Controller che gestisce l'inserimento e la rimozione delle votazioni alle risposte*
Signature dei metodi	+ service(request : HttpServletRequest , response HttpServletResponse) : void + doPost(request : HttpServletRequest , response

	HttpServletResponse) : void
Precondizioni	**context:** VotazioneRisposta:: service(req : HttpServletRequest , resp : HttpServletResponse) **pre:**request.getSession().getAttribute(“utenteLoggato”).getRuolo(). g etNome() = “Partecipante” **context** VotazioneRispostaServlet :: doPost(request HttpServletRequest, response : HttpServletResponse) **pre:** request.getParameter("idRisposta")<>null AND request.getParameter("idDom")<>null
Postcondizioni	**context** VotazioneRispostaServlet :: doPost(request HttpServletRequest, response : HttpServletResponse) **post:** viene aggiunta/rimossa/cambiata la votazione ad una risposta

Classe	**RispostaDAO**
Descrizione	*La classe si occupa di fornire i metodi per accedere in scrittura e lettura ai record della tabella “risposte”*
Signature dei metodi	+ addRisposta(risposta : RispostaBean) : RispostaBean + removeRisposta(risposta : RispostaBean) : void + getStoricoRisposteByUtente(utente : UtenteBean, start : int, offset : int) : ArrayList<RispostaBean> + getRispostaById(id : String) : RispostaBean + getNumeroRisposteByUtente(user : PartecipanteBean) : int + getRisposteByIdDomanda(idDomanda : String, numPagina : int) : ArrayList<RispostaBean> + countRisposteByDomandaId(id : String) : int + getRisposteApprezzate(idutente : String) : ArrayList<RispostaBean> + getRisposteNonApprezzate(idutente : String) : ArrayList<RispostaBean>
Precondizioni	**context** RispostaDAO :: addRisposta(risposta : RispostaBean) **pre:** risposta <> null **context** RispostaDAO :: removeRisposta(risposta : RispostaBean) **pre:** risposta <> null **context** RispostaDAO :: getStoricoRisposteByUtente(utente : UtenteBean, start :int, offset : int) **pre:** utente <> null **context** RispostaDAO :: getRispostaById(id : String) **pre:** id <> null **context** RispostaDAO :: getRisposteByIdDomanda(idDomanda : String) **pre:** idDomanda <> null **context** RispostaDAO :: countRisposteByIdDomanda(idDomanda : String) **pre:** idDomanda <> null **context** RispostaDAO :: getRisposteApprezzate(idUtente : String) **pre:** idUtente <> null

	context RispostaDAO :: getRisposteNonApprezzate(idUtente : String) **pre:** idUtente <> null
Postcondizioni	**context** RispostaDAO :: addRisposta(risposta : RispostaBean) **post:** database.risposte->exists(r| r.idDomanda = risposta.getDomanda().getId() AND r.corpo = risposta.getCorpo() AND r.idAutore = risposta.getAutore().getId()) **context** RispostaDAO :: removeRisposta(risposta : RispostaBean) **post:** database.risposte->exists(r | r.id = risposta.getId() AND r.isNascosta=1)

Classe	**RisposteManager**
Descrizione	*Facade usato per la gestione dell'accesso ai DAO che contribuiscono alla gestione dell'entità RispostaBean*
Signature dei metodi	+ pubblicaRisposta(idDomanda : String , corpo : String , allegati : List<Part> , idAutore : String , idAutoreDomanda : String , dataPubblicazione : Date) : void + getRispostaById(idRisposta : String) : RispostaBean + getRisposteByIdDomanda(idDomanda : String , numPagina : int) : ArrayList<RispostaBean> + caricaAllegati(allegati : List<Part> , risposta RispostaBean) : void + removeRisposta(risposta : RispostaBean) : void + getNumeroRisposte(domanda : DomandeBean) : int + getNumeroRisposteByUtente(user : PartecipanteBean) : int + getStoricoRisposte(user : PartecipanteBean , page : int , offset : int) : ArrayList<RispostaBean> + getRisposteApprezzate(utente : UtenteBean) : HashSet<RispostaBean> + getRisposteNonApprezzate(utente : UtenteBean) : HashSet<RispostaBean>
Precondizioni	**context** RisposteManager :: pubblicaRisposta(idDomanda : String , corpo : String , allegati : List<Part> , idAutore : String , idAutoreDomanda : String , dataPubblicazione : Date) **pre:** idAutoreDomanda<>idAutore AND database.risposte->select(r | r.idDomanda = idDomanda AND r.idAutore = idAutore)->isEmpty() AND corpo.length()>1 AND allegati.size() <= 5 AND allegati->forAll(a | allegati.getSize <= 5MB) AND allegati->forAll(a | allegati.getContentType() = "image/gif" or allegati.getContentType() = "image/jpeg" or allegati.getContentType() = "image/png" or allegati.getContentType() = "image/bmp" or allegati.getContentType() = "image/tiff") **context** RisposteManager :: removeRisposta(risposta : RispostaBean) **pre:** risposta<>null
Postcondizioni	**context** RisposteManager :: pubblicaRisposta(idDomanda : String , corpo : String , allegati : List<Part> , idAutore : String , idAutoreDomanda : String , dataPubblicazione : Date)

	post: database.risposte->exists(r | r.idDomanda = idDomanda AND r.corpo = corpo AND r.idAutore = idAutore) **context** RisposteManager :: removeRisposta(risposta : RispostaBean) **post:** database.risposte->exists(r | r.id = risposta.getId() AND r.isNascosta=1)

Classe	**VotazioneBean**
Descrizione	*Classe che rappresenta l'oggetto Votazione*
Signature dei metodi	+ getUtente() : PartecipanteBean + setUtente(utente : PartecipanteBean) : void + getRisposta() : RispostaBean + setRisposta(risposta : RispostaBean) : void + getValore() : int + setValore(valore : int) : void
Precondizioni	
Postcondizioni	

Classe	**PubblicazioneRispostaServlet**
Descrizione	*Controller che si occupa di gestire l'inserimento di una risposta nel db e di reindirizzare l'utente alla risposta pubblicata*
Signature dei metodi	+ service(request : HttpServletRequest , response HttpServletResponse) : void + doPost(request : HttpServletRequest , response HttpServletResponse) : void
Precondizioni	**context:** PubblicazioneRispostaServlet :: service(req : HttpServletRequest , resp : HttpServletResponse) **pre:**request.getSession().getAttribute("utenteLoggato").getRuolo(). g etNome() = "Partecipante" **context:** PubblicazioneRispostaServlet :: doPost(request HttpServletRequest, response : HttpServletResponse) **pre:** request.getParameter("idDom")<>null AND request.getParameter("corpo")

Postcondizioni	**context:** PubblicazioneRispostaServlet :: doPost(request HttpServletRequest, response : HttpServletResponse) **post:** database.risposte>includes(r | r.id = request.getParameter(risposta.getId()))

Classe	**RispostaBean**
Descrizione	*Classe che rappresenta l'oggetto Risposta (può essere associato a Domanda, Partecipante e SegnalazioneRisposta)*
Signature dei metodi	+ getId() : String + setId(id : String) : void + getCorpo() : String + setCorpo(corpo : String) : void + getDataPubblicazione() : Date + setDataPubblicazione(dataPubblicazione : Date) : void + getMiPiace() : int + setMiPiace(miPiace : int) : void + getNonMiPiace() : int + setNonMiPiace(nonMiPiace : int) : void + getVoti() : ArrayList<VotazioneBean> + setVoti(voti : ArrayList<VotazioneBean>) : void + getAllegati() : ArrayList<String> + setAllegati(allegati : ArrayList<String>) : void + getAutore() : PartecipanteBean + setAutore(autore : PartecipanteBean) : void + getDomanda() : DomandaBean + setDomanda(domanda : DomandaBean) : void
Precondizioni	
Postcondizioni	

Classe	**VisualizzaStoricoRisposte**
Descrizione	*Controller che si occupa di reindirizzare l'utente alla pagina che mostra l'elenco delle risposte pubblicate dall'utente*
Signature dei metodi	+ service(req : HttpServletRequest , resp : HttpServletResponse) : void + doGet(request : HttpServletRequest , response : HttpServletResponse) : void + doPost(request : HttpServletRequest , response HttpServletResponse) : void
Precondizioni	**context:** VisualizzaStoricoRisposte :: service(req : HttpServletRequest , resp : HttpServletResponse) **pre:** request.getSession().getAttribute("utenteLoggato").getRuolo().getNome() = "Partecipante"
Postcondizioni	**context:** VisualizzaStoricoRisposte:: doGet (request HttpServletRequest, response : HttpServletResponse) **post:** request.getAttribute("storicoRisposte") <> null

Gestione account

Classe	UtenteBean
Descrizione	**Rappresenta l'oggetto Utente**
Signature dei metodi	+ getNome() : String + getCognome(): String + getEmail(): String + getId(): String + getPasswordHash(): String + getRuoloId(): Integer +getUsername(): String + isDisattivato(): Boolean + setEmail(email: String) : void + setCognome(cognome: String): void + setNome(Nome: String): void + setId(id:String): void + setPasswordHash(pwdHash: String): void + setRuoloId(ruoloID: Integer): void

	+ setUsername(username: String): void
Precondizioni	**context** **pre:**
Postcondizioni	**context** **post:**

Classe	RegistrazioneServlet
Descrizione	**Controller che si occupa di gestire l'inserimento di un Partecipante nel db**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** RegistrazioneServlet:: doPost(request, response) **pre:** request.getParameter("email") != null and request.getParameter("nome") != null and request.getParameter("cognome") != null and request.getParameter("username") != null and request.getParameter("password") != null
Postcondizioni	**context** RegistrazioneServlet:: doPost(request, response) **post:** database.utenti.includes(u | u.email = request.getParameter("email")

Classe	VisualizzaProfilo
Descrizione	**Servlet che fa redirect alla pagina per visualizzare e modificare i dati del profilo Partecipante**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** doPost(request, response) **pre:** request.getSession().getAttribute("utenteLoggato").getRuolo().getNome() = "Parteciapante
Postcondizioni	

Classe	VisualizzaFormAccesso
Descrizione	**Servlet che fa redirect alla pagina di accesso**
Signature dei metodi	+doPost(request: HttpServletRequest, response:

	HttpServletResponse)
Precondizioni	
Postcondizioni	

Classe	VisualizzaFormRegistrazione
Descrizione	**Servlet che fa redirect alla pagina di registrazione**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	
Postcondizioni	

Classe	AccessoServlet
Descrizione	**Controller per gestire l'autenticazione degli utenti**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** AccessoServlet:: doPost(request, response) **pre:** database.utenti->exists(u | u.email = request.getParameter("email") and u.passwordHash = getPasswordHash(request.getParameter("password")))
Postcondizioni	**context** AccessoServlet:: doPost(request, response) **post:** request.getSession().getAttribute("utenteLoggato") != null

Classe	AccountManager
Descrizione	**Si tratta di Facade per l'accesso ai DAO che contribuiscono alla gestione delle entità UtenteBean e PartecipanteBean**
Signature dei metodi	-isEmailAvailable(email: String): boolean -isUsernameAvailable(username:String): boolean -generateUtenteBean(nome:String , cognome:String , email:String , username:String , password:String): UtenteBean -getPasswordHash(password:String): String -getUserInstance(ruoloId:int , email:String) -updateInteressiUtente(utente:PartecipanteBean , nomiCategorie: Collection)

	+addInteresseParteciapante(user:ParteciapanteBean, interesse:CategoriaBean): void +autenticaUtente(email:String , password:String): UtenteBean +deleteUtente(id:String): void +getAllModeratori(): Collection moderatori +RegisterUser(nome:String , cognome:String , email:String , username:String , password:String , interessi:Collection): void +RegistraModeratore(nome:String , cognome:String , email:String , username:String , password:String): void +removeInteresseParteciapante(partecipante:ParteciapanteBean, interesse:CategoriaBean): void +updateUtente(user:ParteciapanteBean , newNome:String , newCognome:String , newUsername:String , newEmail:String , interessi:Collection , password:String)
Precondizioni	**context** AccountManager :: addInteresseParteciapante (user, interesse) **pre:** database.categorie->exists(c | c.nome = interesse.getNome()) **context** AccountManager::autenticaUtente(String email, String password): UtenteBean **pre:** database.utenti->exists(u | u.email = email and u.passwordHash = self.getPasswordHash(password) and not u.isDisattivato) **context** AccountManager:: generateUtenteBean(nome, cognome, username, email, password) **pre:** database.utenti->select(u | u.email = email) -> isEmpty() and database.utenti->select(u | u.username = username) -> isEmpty() nome.size() > 2 and (password.size() > 6 and password.size() < 32) and (username.size() > 3 and username.size() < 10) nome.matches('([A-Za-z'À-ÖØ-öø-ÿ]+\\s*)+') and cognome.matches('([A-Za-z'À-ÖØ-öø-ÿ]+\\s*)+') and username.matches('^[a-zA-Z]+[a-zA-Z0-9]*') and email.matches('^[a-zA-Z0-9_!#$%&'*+/=?`{\|}~^.-]+@[a-zA-Z0-9.-]+$') **context** AccountManager:: RegisterUser(nome, cognome, username, email, password, interessi) **pre:** interessi->size() > 0

context
AccountManager::removeInteresseParteciapante(partecipante, categoria)
pre:
database.categorie->exists(c | c.nome = categoria.getNome())

context
AccountManager::updateUtente(user, newNome, newCognome, newUsername, newEmail, interessi, password)

pre:
interessi <> null
newNome.size() > 2 and (newPassword.size() > 6 and password.size() < 32) and (newUsername.size() > 3 and newUsername.size() < 10)

newNome.matches('([A-Za-z'À-ÖØ-öø-ÿ]+\\s*)+') and newCognome.matches('([A-Za-z'À-ÖØ-öø-ÿ]+\\s*)+') and newUsername.matches('^[a-zA-Z]+[a-zA-Z0-9]*') and newEmail.matches('^[a-zA-Z0-9_!#$%&'*+/=?`{|}~^.-]+@[a-zA-Z0-9.-]+$')

Postcondizioni

context AccountManager::addInteresseParteciapante(user, interesse)
post: database.utenti->getInteressi()->includes(interesse)

context AccountManager:: deleteUtente(id)
post database.utenti->exists(u | u.id = id and u.isDisattivato = true)

context AccountManager:: RegisterUser(nome, cognome, username, email, password, interessi)
post
database.utenti->exists(u | u.nome = nome and u.cognome = cognome and u. email= email and u.passwordHash = self.getPasswordHash(password) and u.username = username and u.getInteressi() = interessi and u.getRuolo().getNome() = "Partecipante")

context AccountManager:: RegisterModeratore(nome, cognome, username, email, password)
post
database.utenti->exists(u | u.nome = nome and u.cognome = cognome and u. email= email and u.passwordHash = self.getPasswordHash(password) and u.username = username and u.getInteressi() = interessi and u.getRuolo().getNome() = "Moderatore")

	context AccountManager::removeInteresseParteciapante(partecipante, categoria) **post** partecipante->interessi->not includes(c | c.getID() = categoria.getId()) **context** updateUtente(user, newNome, newCognome, newUsername, newEmail, interessi, password) **post** user.id = user'.id and user'.nome = newNome and user'.cognome = newCognome and user'.username = newUsername and user'.passwordHash = getPasswordHash(newPassword) and user'.email = newEmail and user'.interessi = interessi

Classe	RuoloDAO
Descrizione	**La classe si occupa di fornire i metodi per accedere in scrittura e lettura ai record della tabella Ruoli**
Signature dei metodi	+getRuoloById(id: int): RuoloBean +getRuoloByName(name: String): RuoloBean
Precondizioni	
Postcondizioni	

Classe	UtenteDAO
Descrizione	**La classe si occupa di fornire i metodi per accedere in scrittura e lettura ai record della tabella Utenti**
Signature dei metodi	+doAddModeratore(mod: UtenteBean) +doAddUtente(utente: UtenteBean) +doDeactivateUser(id: String) +doGetAllModeratori(): Collection +getUtenteByEmail(String): UtenteBean +getUtenteById(String): UtenteBean +getUtenteByUsername(username: String): UtenteBean
Precondizioni	
Postcondizioni	**context** UtenteDAO::doAddModeratore(mod) **post** database.utenti->includes(m| m.id = mod.id)

	context UtenteDAO::doAddUtente(user) **post** database.utenti->includes(u | user.id = u.id) **context** UtenteDAO::doDeactivateUser(id) **post** database->exists(u | u.id = id and u.isDisattivato)

Classe	RuoloBean
Descrizione	**Classe che rappresenta i ruoli che possono essere associati ad un Utente**
Signature dei metodi	+getId(): int +getNome(): String +setId(id: Integer): void +setNome(nome: String): void
Precondizioni	
Postcondizioni	

Classe	PartecipanteBean
Descrizione	**Una specializzazione della classe UtenteBean**
Signature dei metodi	+getInteressi(): Collection + setInteressi(interessi:Collection): void + getNumeroSegnalazioni(): int + setNumeroSegnalazioni(): void
Precondizioni	
Postcondizioni	

Classe	RoleRouterServlet
Descrizione	**Servlet che viene invocata da AccessoServlet, per reindirizzare l'utente su una homepage diversa a seconda del Ruolo**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** doPost(request, response) **pre:** request.getSession().getAttribute("utenteLoggato") != null
Postcondizioni	**context** doPost(request, response) **post:**

	if(utenteLoggato.getRuolo().getNome() = “Partecipante”)
	redirect to “VisualizzaHomeServlet”
	if(utenteLoggato.getRuolo().getNome() = “Moderatore”)
	redirect to “VisualizzaElencoSegnalazioniRisposte”
	if(utenteLoggato.getRuolo().getNome() = “MasterModeratore”)
	redirect to “GestioneModeratori”

Classe	UpdateProfilo
Descrizione	**Servlet che va ad aggiornare i campi del record utente**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** doPost(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”) != null and request.getSession().getAttribute(“utenteLoggato”).ruolo.getNome( ) = “Partecipante” and request.getParameter(“email”) != null and request.getParameter(“nome”) != null and request.getParameter(“cognome”) != null and request.getParameter(“username”) != null and request.getParameter(“password”) != null
Postcondizioni	**context** doPost(request, response) **post:** @pre. request.getSession().getAttribute(“utenteLoggato”) != @post request.getSession().getAttribute(“utenteLoggato”)

Classe	PartecipanteDAO
Descrizione	**La classe si occupa di fornire i metodi per accedere in scrittura e lettura ai record della tabella Partecipante**
Signature dei metodi	+addInteresse(user: PartecipanteBean, categoria: CategoriaBean) +getPartecipanteByEmail(email: String): PartecipanteBean +removeInteresse(user: PartecipanteBean, interesse: CategoriaBean) +removePartecipanteById(user: PartecipanteBean) +updateUtente(user : PartecipanteBean)
Precondizioni	**context** PartecipanteDAO::addInteresse(user, categoria) **pre** categoria != null and user != null **context** PartecipanteDAO::removeInteresse(user, categoria) **pre** categoria != null and user != null **context** PartecipanteDAO::removePartecipante(user)

	pre user != null
Postcondizioni	**context** PartecipanteDAO::addInteresse(user, categoria) **post** database.interessi->includes(i | i.idUtente = user.getId() and i.idCategoria = categoria.getId()) **context** PartecipanteDAO::removeInteresse(user, categoria) **post** database.interessi-> not includes(i | i.idCategoria = categoria and i.idUtente = user.getId()) **context** removePartecipanteById(user) **post** user->isDisattivato = true database.utenti->exists(u | u.id = user.getId() and u.isDisattivato = 1) **context** PartecipanteDAO::updateUtente(user) **post** @pre.user <> @post.user

Classe	LogoutServlet
Descrizione	**Controller che si occupa di rimuovere dalla sessione, l'istanza di utente loggato**
Signature dei metodi	+doPost(HttpServletRequest: request, HttpServletResponse: response)
Precondizioni	
Postcondizioni	**context** LogoutServlet::doPost(request,response) **post:** request.getSession().getAttribute("utenteLoggato") = null

Moderazione

Classe	**SegnalazioneRispostaBean**
Descrizione	*Classe che rappresenta l'oggetto SegnalazioneRisposta*

	(specializzazione di SegnalazioneBean)
Signature dei metodi	+ getIdSegnalazione() : String + setIdSegnalazione(idSegnalazione : String) : void + getDataSegnalazione() : Date + setDataSegnalazione(dataSegnalazione : Date) : void + getStato() : int + setStato(stato : int) : void + getCommento() : String + setCommento(commento : String) : void + getRispostaSegnalata() : RispostaBean + setRispostaSegnalata(rispostaSegnalata : RispostaBean) : void + getMotivazione() : MotivazioneBean + setMotivazione(motivazione : MotivazioneBean) : void
Precondizioni	
Postcondizioni	

Classe	**SegnalazioneBean**
Descrizione	*Classe che rappresenta l'oggetto Segnalazione*
Signature dei metodi	+ getId() : String + setId(id : String) : void + getMotivazione() : MotivazioneBean + setMotivazione(motivazione : MotivazioneBean) : void + getDataSegnalazione() : Date + setDataSegnalazione(dataSegnalazione : Date) : void + getStato() : int + setStato(stato : int) : void

	+ getCommento() : String + setCommento(commento : String) : void
Precondizioni	
Postcondizioni	

Classe	**SegnalazioneManager**
Descrizione	*Facade usato per la gestione dell'accesso ai DAO che contribuiscono alla gestione delle entità Segnalazione*
Signature dei metodi	+ creazioneSegnalazioneDomanda(motivazione : MotivazioneBean, dataSegnalazione : Date, commento : String, domandaSegnalata : DomandaBean, utente : PartecipanteBean) : void + creazioneSegnalazioneRisposta(srb : SegnalazioneRispostaBean) : void + getSegnalazioniDomanda(start : int, end : int) : ArrayList<SegnalazioneDomandaBean> + getAllSegnalazioniDomanda() : ArrayList<SegnalazioneDomandaBean> + getAllSegnalazioniRisposta() : ArrayList<SegnalazioneRispostaBean> + getSegnalazioneDomanda(id : int) : SegnalazioneDomandaBean + getSegnalazioneRisposta(id : String) : SegnalazioneRispostaBean + risolviSegnalazioneDomanda(segnalazioneDaRisolvere : SegnalazioneDomandaBean) : void + declinaSegnalazioneDomanda(segnalazione : SegnalazioneDomandaBean) : void + risolviSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean) : void + declinaSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean) : void + getNumeroSegnalazioniDomanda() : int
Precondizioni	**context:** SegnalazioneManager ::

	creazioneSegnalazioneDomanda(motivazione : MotivazioneBean, dataSegnalazione : Date, commento : String, domandaSegnalata : DomandaBean, utente : PartecipanteBean) **pre:** commento.length()<=256 **context:** SegnalazioneManager :: creazioneSegnalazioneRisposta(srb : SegnalazioneRispostaBean) **pre:** commento.length()<=256

Postcondizioni	**context:** SegnalazioneManager :: creazioneSegnalazioneDomanda(motivazione : MotivazioneBean, dataSegnalazione : Date, commento : String, domandaSegnalata : DomandaBean, utente : PartecipanteBean)
	post: database.segnalazioni->exists(s | s.motivazione = motivazione.id() AND s.dataSegnalazione = dataSegnalazione AND s. commento = commento AND s.utente = utente.getId() AND database.segnalazioniDomanda->exists(sd | sd.idDomanda = domandaSegnalata.getId() AND sd.id = s.id))
	
	context: SegnalazioneManager :: creazioneSegnalazioneRisposta(srb : SegnalazioneRispostaBean)
	post: database.segnalazioni->exists(s | s.motivazione = srb.getMotivazione().getId() AND s.dataSegnalazione = srb.getDataSegnalazione() AND s.commento = srb.getCommento() AND database.segnalazioniRisposte->exists(sr | sr.idRisposta = srb.getRispostaSegnalata().getId() AND sr.id = srb.getIdSegnalazione()))
	
	context: SegnalazioneManager :: risolviSegnalazioneDomanda(segnalazioneDaRisolvere : SegnalazioneDomandaBean)
	post:
	if(segnalazioneDaRisolvere.getMotivazione().getId()= MotivazioneBean.OFFTOPIC) then
	database.segnalazioni->select(s | s.idMotivazione = MotivazioneBean.OFFTOPIC AND s.stato = SegnalazioneBean.DA_GESTIRE AND database.segnalazionidomande->exists(sd | sd.idSegnalazione = s.id AND sd.idDomandaSegnalata = segnalazioneDaRisolvere.getDomandaSegnalata().getId()))
	->isEmpty()
	else
	database.segnalazioni->select(s | s.stato = SegnalazioneBean.DA_GESTIRE AND database.segnalazionidomande->exists(sd | sd.idSegnalazione = s.id AND sd.idDomandaSegnalata = segnalazioneDaRisolvere.getDomandaSegnalata().getId()))
	->isEmpty()
	
	context: SegnalazioneManager ::

declinaSegnalazioneDomanda(segnalazione : SegnalazioneDomandaBean)
post: database.segnalazioni->exists(s | s.id = segnalazione.getId() AND s.stato=3)

context: SegnalazioneManager :: risolviSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean)
post: database.segnalazioni->select(s | s.stato = SegnalazioneBean.DA_GESTIRE AND database.segnalazionirisposte->exists(sd | sd.idSegnalazione = s.id AND sd.idRispostaSegnalata = segnalazioneDaRisolvere.getRispostaSegnalata().getId())) ->isEmpty()

context: SegnalazioneManager :: declinaSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean)
post: database.segnalazioni->exists(s | s.id = segnalazione.getId() AND s.stato=3)

Classe	GestioneModeratoriServlet
Descrizione	**Servlet che fa redirect alla homepage di MasterModeratore, per gestire i Moderatori presenti**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** doPost(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”).getRuolo().getNome() = “MasterModeratore”
Postcondizioni	

Classe	CreazioneModeratori
Descrizione	**Servlet che fa redirect alla form, per creare i Moderatori**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** doPost(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”).getRuolo().getNome() = “MasterModeratore”
Postcondizioni	

Classe	CreazioneModeratoreServlet
Descrizione	**Controller che si occupa di creare un utente con ruolo Moderatore**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** CreazioneModeratoreServlet:: doPost(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”).getRuolo().getNome() = “MasterModeratore” and request.getParameter(“email”) != null and request.getParameter(“nome”) != null and request.getParameter(“cognome”) != null and request.getParameter(“username”) != null and request.getParameter(“password”) != null
Postcondizioni	**context** CreazioneModeratoreServlet:: doPost(request, response) **post:** database->utenti->includes(u | u.email = request.getParameter(“email”) and u.ruolo = “Moderatore”)

Classe	DisattivazioneModeratore
Descrizione	**Controller che va a disattivare un moderatore, attraverso il suo id**
Signature dei metodi	+doPost(request: HttpServletRequest, response: HttpServletResponse)
Precondizioni	**context** DisattivazioneModeratore:: doPost(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”).getRuolo().getNome() = “MasterModeratore” and request.getParameter(“idModeratore”) != null
Postcondizioni	**context** DisattivazioneModeratore:: doPost(request, response) **post:** moderatore->isDisattivato = true

Classe	**ElencoSegnalazioniRisposteServlet**
Descrizione	*Controller che si occupa di reindirizzare l’utente alla pagina che mostra l’elenco delle segnalazioni delle risposte*
Signature dei metodi	+ service(request : HttpServletRequest , response HttpServletResponse) : void + doGet(request : HttpServletRequest , response HttpServletResponse) : void + doPost(request : HttpServletRequest , response HttpServletResponse) : void
Precondizioni	**context:** ElencoSegnalazioniRisposteServlet :: service(request HttpServletRequest, response : HttpServletResponse) **pre:**request.getSession().getAttribute(“utenteLoggato”).getRuolo().getNome() = “Moderatore”
Postcondizioni	**context:** ElencoSegnalazioniRisposteServlet :: doGet(request : HttpServletRequest , response HttpServletResponse) **post:** request.getSession().getAttribute(“segnalazioniRisposta”)<>null

Classe	**GestioneSegnalazioneRispostaServlet**
Descrizione	*Controller che si occupa di gestire le operazioni relative alla gestione delle segnalazioni delle risposte da parte del moderatore*
Signature dei metodi	+ service(request : HttpServletRequest , response HttpServletResponse) : void + doGet(request : HttpServletRequest , response HttpServletResponse) : void + doPost(request : HttpServletRequest , response HttpServletResponse) : void
Precondizioni	**context** GestioneSegnalazioneRispostaServlet :: service(request : HttpServletRequest , response HttpServletResponse) **pre:** request.getSession().getAttribute(“utenteLoggato”).getRuolo().getNome() = “Moderatore” **context** GestioneSegnalazioneRispostaServlet :: doGet(request : HttpServletRequest , response HttpServletResponse) **pre:** request.getParameter(“idSegnalazione”) <> null AND request.getParameter(“idRisposta”) AND (request.getParameter(“approva”) XOR request.getParameter(“ignora”))
Postcondizioni	**context** GestioneSegnalazioneRispostaServlet :: doGet(request : HttpServletRequest , response HttpServletResponse) **post:** if (request.getParameter(“approva”)) then database.segnalazioni->exists (s | s.id = request.getParameter(“idSegnalazione”) AND s.stato = 2) AND database.risposte->exists (r | r.id = request.getParameter(“idRisposta”) AND r.isNascosta = 1) else then database.segnalazioni->exists (s | s.id = request.getParameter(“idSegnalazione”) AND s.stato = 3)

Classe	**SegnalazioneRispostaServlet**
Descrizione	*Controller che si occupa di inserire le segnalazioni inviate dagli utenti*
Signature dei metodi	+ service(request : HttpServletRequest , response HttpServletResponse) : void + doPost(request : HttpServletRequest , response HttpServletResponse) : void
Precondizioni	**context** SegnalazioneRispostaServlet :: service(request HttpServletRequest, response : HttpServletResponse) **pre:**request.getSession().getAttribute("utenteLoggato").getRuolo(). getNome() = "Partecipante" **context** SegnalazioneRispostaServlet:: doPost (request HttpServletRequest, response : HttpServletResponse) **pre:** request.getAttribute("idRisposta")<>null AND request.getAttribute("idMotivazione")<>null
Postcondizioni	**context:** SegnalazioneRispostaServlet :: doPost(request HttpServletRequest, response : HttpServletResponse) **post:** viene pubblicata la segnalazione (viene inserita la segnalazione nel database)

Classe	**SegnalazioneRispostaDAO**
Descrizione	*Classe che si occupa di realizzare l'accesso alle tabelle "segnalazioni" e "segnalazionirisposta"*
Signature dei metodi	+ addSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean) : SegnalazioneRispostaBean + updateStatoSegnalazioneRisposta (segnalazione : SegnalazioneRispostaBean) : void + getElencoSegnalazioniRisposte() : ArrayList<SegnalazioneRispostaBean> + getSegnalazioneRispostaById(idSegnalazione : String) : SegnalazioneRispostaBean
Precondizioni	**context** SegnalazioneRispostaDAO :: addSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean) **pre:** segnalazione <> null **context** SegnalazioneRispostaDAO :: updateStatoSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean) **pre:** segnalazione <> null
Postcondizioni	**context** SegnalazioneRispostaDAO :: addSegnalazioneRisposta(segnalazione : SegnalazioneRispostaBean) **post:** database.segnalazioni->exists(s | s.motivazione = segnalazione.getMotivazione() and s.dataSegnalazione = segnalazione.getDataSegnalazione() and s.stato = segnalazione.getStato() and s.commento = segnalazione.getCommento() and database->segnalazionirisposta.includes(sr | sr.id = segnalazione.getId() and sr.idDomanda = segnalazione.getDomandaSegnalata().getId()) **context** SegnalazioneRispostaDAO :: updateStatoSegnalazione(segnalazione : SegnalazioneRispostaBean) **post:** database.segnalazioni->exists(s | s.id = segnalazione.getId() AND s.stato <> 1)

Classe	**MotivazioneBean**
Descrizione	*Questa classe rappresenta l'oggetto Motivazione.*

Signature dei metodi	+ getId() : int + setId(id : int) : void + setNome(nome : String) : void + getNome() : String
Precondizioni	
Postcondizioni	

Classe	**MotivazioneDAO**
Descrizione	*Questa classe si occupa di svolgere operazioni riguardanti le motivazioni sul database.*
Signature dei metodi	+ getAll() : ArrayList<MotivazioneBean> + getMotivazioneById(id : int) : MotivazioneBean
Precondizioni	
Postcondizioni	**context** MotivazioneDAO :: getAll() **post:** database.motivazioni **context** MotivazioneDAO :: getMotivazioneById(id) **post:** database.motivazioni->select(m | motivazioni.id = id)

Classe	**SegnalazioneDomandaBean**
Descrizione	*Questa classe rappresenta l'oggetto SegnalazioneDomanda.*
Signature dei metodi	+ getDomandaSegnalata() : DomandaBean + setDomandaSegnalata(domandaSegnalata : DomandaBean) : void + toString() : String
Precondizioni	
Postcondizioni	

Classe	**SegnalazioneDomandaDAO**
Descrizione	*Questa classe si occupa dell'aggiunta, della modifica e del recupero delle segnalazioni delle domande all'interno del database.*
Signature dei metodi	+ addSegnalazioneDomanda(segnalazione : SegnalazioneDomandaBean) : void + getAll() : ArrayList<SegnalazioneDomandaBean> + getNumeroSegnalazioniDomanda() : int + updateStatoSegnalazioneDomanda(segnalazione :

	SegnalazioneDomandaBean) : void + getSegnalazioneDomandaById(idSegnalazione : String) : SegnalazioneDomandaBean + getSegnalazioniDomanda(start : int, end : int) : ArrayList<SegnalazioneDomandaBean>
Precondizioni	**context** SegnalazioneDomandaDAO :: addSegnalazioneDomanda(segnalazione) **pre:** segnalazione <> null **context** SegnalazioneDomandaDAO :: updateStatoSegnalazioneDomanda(segnalazione) **pre:** segnalazione <> null **context** SegnalazioneDomandaDAO :: getSegnalazioneDomandaById(idSegnalazione) **pre:** idSegnalazione <> null **context** SegnalazioneDomandaDAO :: getSegnalazioniDomanda(start, end) **pre:** start <= end
Postcondizioni	**context** SegnalazioneDomandaDAO :: addSegnalazioneDomanda(segnalazione) **post:** database.segnalazioni->exists(s | s.motivazione = segnalazione.getMotivazione() and s.dataSegnalazione = segnalazione.gerDataSegnalazione() and s.stato = segnalazione.getStato() and s.commento = segnalazione.getCommento() and database->segnalazionidomanda.includes(sd | sd.id = segnalazione.getId() and sd.idDomanda = segnalazione.getDomandaSegnalata().getId()) **context** SegnalazioneDomandaDAO :: getAll() **post:** database.segnalazionidomanda **context** SegnalazioneDomandaDAO :: getNumeroSegnalazioniDomanda() **post:** database->segnalazionidomanda.size() **context** SegnalazioneDomandaDAO :: updateStatoSegnalazioneDomanda(segnalazione) **post:** @pre.segnalazione.getStato() <> @post.segnalazione.getStato() **context** SegnalazioneDomandaDAO :: getSegnalazioneDomandaById(idSegnalazione) **post:** database->segnalazionidomanda->select(sd | segnalazionidomanda.id = idSegnalazione) **context** SegnalazioneDomandaDAO ::

	getSegnalazioniDomanda(start, end) **post:** database.segnalazionidomanda.size() = (end - start)

Classe	**CambiaCategoriaDomandaServlet**
Descrizione	*Questa classe è un Control che viene chiamato dopo che un moderatore ha approvato una segnalazione la cui motivazione è 'off-topic' ed ha selezionato da un elenco le categorie che sostituiranno le vecchie categorie della domanda la cui segnalazione è stata approvata.*
Signature dei metodi	# service(req : HttpServletRequest, resp : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void # doPost(request : HttpServletRequest, response : HttpServletResponse) : void - setStringAttributeThenRedirect(nomeAttributo : String, messaggioAttributo : String, request : HttpServletRequest, response : HttpServletResponse, path : String) : void
Precondizioni	**context** CambiaCategoriaDomandaServlet :: service(req, resp) **pre:** req.getSession().getAttribute("utenteLoggato") <> null and req.getSession().getAttribute("utenteLoggato").getRuolo().getId() = RuoloBean.ROLE_MODERATORE **context** CambiaCategoriaDomandaServlet :: doGet(request, response) **pre:** request.getParameter("idSegnalazione") <> null and request.getParameterValues("categorieDomanda") <> null and segnalazioniManager.getSegnalazioneById(request.getParameter("idSegnalazione")) <> null and segnalazioniManager.getSegnalazioneById(request.getParameter("idSegnalazione")).getMotivazione().getId() = MotivazioneBean.OFFTOPIC
Postcondizioni	**context** CambiaCategoriaDomandaServlet :: service(req, resp) **post:** L'utente è reindirizzato alla servlet per poter fare il login if not (req.getSession().getAttribute("utenteLoggato") <> null and req.getSession().getAttribute("utenteLoggato").getRuolo() = RuoloDAO.getRuoloByNome("Moderatore")) **context** CambiaCategoriaDomandaServlet :: doGet(request, response) **post:** domandeManager.getDomandaById(segnalazioniManager.getSegnalazioneDomandaById(request.getAttribute("idSegnalazione").getDomanda().getCategorie().getId() = request.getParameter("categorieDomanda")

	context CambiaCategoriaDomandaServlet :: setStringAttributeThenRedirect(nomeAttributo, messaggioAttributo, request, response, path) **post:** Viene effettuato il forward alla Servlet con il percorso path e viene settato un attributo di nome ‘nomeAttributo’ e di valore ‘messaggioAttributo’

Classe	**DeclinaSegnalazioneDomandaServlet**
Descrizione	*Servlet che viene chiamata quando un moderatore clicca sul pulsante ‘Declina’ di una segnalazione ad una domanda dall’apposito elenco.*
Signature dei metodi	- setStringAttributeThenRedirect(nomeAttributo : String, messaggioAttributo : String, request : HttpServletRequest, response : HttpServletResponse, path : String) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** DeclinaSegnalazioneDomandaServlet:: doGet(request, response) **pre:** request.getParameter("idSegnalazione") <> null and managerSegnalazioni.getSegnalazioneDomanda(request.getParameter("idSegnalazione")) <> null
Postcondizioni	**context** DeclinaSegnalazioneDomandaServlet:: doGet(request, response) **post:** managerSegnalazioni.getSegnalazioneDomanda(request.getParameter(“idSegnalazione”)).getStato() = 3

Classe	**RisolviSegnalazioneDomandaServlet**
Descrizione	*Servlet che permette ad un moderatore di risolvere una segnalazione con una motivazione diversa da ‘off-topic’, risolve tutte le altre segnalazioni che si riferiscono alla stessa domanda e rimuove la domanda con le relative risposte.*
Signature dei metodi	# service(req : HttpServletRequest, resp : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void # doPost(request : HttpServletRequest, response : HttpServletResponse) : void -

Precondizioni	**context** RisolviSegnalazioneDomandaServlet :: service(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”) <> null and request.getSession().getAttribute(“utenteLoggato”).getRuolo = RuoloBean.ROLE_MODERATORE **context** RisolviSegnalazioneDomandaServlet :: doGet(request, response) **pre:** request.getParameter("idSegnalazione") <> null and managerSegnalazioni.getSegnalazioneDomanda(request.getParameter("idSegnalazione")) <> null and managerSegnalazioni.getSegnalazioneDomanda(request.getParameter("idSegnalazione")).getMotivazione().getId() <> MotivazioneBean.OFFTOPIC
Postcondizioni	**context** RisolviSegnalazioneDomandaServlet :: doGet(request, response) **post:** domandeManager.getDomandaById(request.getParameter(“idDomanda”)) = null and *** managerSegnalazioni.getSegnalazioneDomanda(request.getParameter(“idSegnalazione”)) = null

Classe	**SegnalazioneDomandaServlet**
Descrizione	*Servlet che permette ad un utente loggato di segnalare una domanda*
Signature dei metodi	# service(req : HttpServletRequest, resp : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void # doPost(request : HttpServletRequest, response : HttpServletResponse) : void - setStringAttributeThenRedirect(nomeAttributo : String, messaggioAttributo : String, request : HttpServletRequest, response : HttpServletResponse, path : String) : void
Precondizioni	**context** SegnalazioneDomandaServlet :: service(request, response) **pre:** request.getSession().getAttribute(“utenteLoggato”) <> null and request.getSession().getAttribute(“utenteLoggato”).getRuolo = RuoloBean.ROLE_PARTECIPANTE **context** SegnalazioneDomandaServlet :: doGet(request, response) **pre:** request.getParameter("idDomandaSegnalata") <> null and request.getParameter("idMotivazione") <> null and managerDomande.getDomandaById(request.getParameter("idDomandaSegnalata")) and if request.getParameter("commento") <> null then request.getParameter("commento").length > 256

Postcondizioni	**context** SegnalazioneDomandaServlet :: doGet(request, response) **post:** Viene chiamato il metodo del manager delle segnalazioni che inserisce una segnalazione al database

Classe	**VisualizzaElencoSegnalazioniDomandaServlet**
Descrizione	*Servlet che permette ad un moderatore loggato di visualizzare l'elenco di segnalazioni alle domande.*
Signature dei metodi	# service(req : HttpServletRequest, resp : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void # doPost(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** VisualizzaElencoSegnalazioniDomandaServlet :: service(request, response) **pre:** request.getSession().getAttribute("utenteLoggato") <> null and request.getSession().getAttribute("utenteLoggato").getRuolo = RuoloBean.ROLE_MODERATORE
Postcondizioni	**context** VisualizzaElencoSegnalazioniDomandaServlet :: doGet(request, response) **post:** request.getAttribute("segnalazioniDomanda") <> null and request.getAttribute("categorie") <> null and request.getAttribute("paginaCorrente") <> null and request.getAttribute("pagineTotali") <> null