Università degli Studi di Salerno
Corso di Ingegneria del Software

SwaGGed
Object Design Document
Versione 2.0

Data: 29/12/2024

Coordinatore del progetto

Nome	Matricola

Partecipanti:

Nome	Matricola
Choaib Goumri	0512118390
Mattia Gallucci	0512116893

Scritto da:	Choaib Goumri
	Mattia Gallucci

Revision History

Data	Versione	Descrizione	Autore
2/12/2024	1.0	Prima stesura del documento	Choaib Goumri, Gallucci Mattia
25/12/2024	1.1	Aggiustamenti class interfaces	Choaib Goumri, Gallucci Mattia
29/12/2024	2.0	Finalizzazione del documento	Choaib Goumri, Gallucci Mattia

Sommario

1. INTRODUZIONE

1.1 Object design trade-offs

Comprensibilità vs Tempo

A causa dei tempi di sviluppo limitati, non sarà prioritario inserire commenti dettagliati in ogni metodo. Questa scelta mira a velocizzare l'implementazione del sistema, ma comporterà una maggiore complessità nella fase di testing e nelle eventuali modifiche future.

Riusabilità vs Costi

Il sistema sarà sviluppato senza l'utilizzo di librerie o componenti a pagamento, poiché il progetto non dispone di un budget economico.

Sicurezza vs Efficienza

Considerando le tempistiche ristrette, il sistema implementerà misure di sicurezza basate su **username e password crittografate**. Sarà garantito un controllo degli accessi mediante ruoli definiti, bilanciando sicurezza ed efficienza.

Incapsulamento vs Efficienza

L'architettura del sistema assicura la protezione dei dettagli implementativi delle classi. Gli attributi saranno manipolabili esclusivamente attraverso i metodi definiti, rispettando il principio di incapsulamento.

Trasparenza vs Efficienza

Per la gestione della persistenza dei dati, si utilizzerà un database relazionale. Grazie al supporto di un **DBMS**, sarà possibile operare in modo consistente e transazionale, accettando una minima perdita di efficienza. Le connessioni al database saranno gestite tramite un **Driver Manager**, che semplifica la sincronizzazione e l'interazione con il DBMS.

1.2 Interface documentiation guidelines

Per garantire un'elevata leggibilità e una buona manutenibilità del codice, gli sviluppatori devono seguire specifiche linee guida durante la scrittura del codice.

Stile del codice

Gli sviluppatori sono invitati ad aderire il più possibile alle convenzioni di stile del codice definite da Google Java. In particolare:

- **Indentazione:** utilizzare i **tab** invece degli spazi.
- **Formattazione:** mantenere uno stile conciso e facilmente comprensibile.

Convenzioni per i nomi

I nomi utilizzati nel codice devono essere:

- **Descrittivi:** chiari e indicativi del loro scopo.
- **Pronunciabili:** facilmente leggibili e comprensibili.
- **Di uso comune:** seguire termini standard e familiari.
- **Non abbreviati:** evitare abbreviazioni, tranne per variabili temporanee.
- **Validi:** contenere solo caratteri consentiti (A-Z, a-z, 0-9).

Classi

- I nomi delle classi devono seguire lo stile **Upper Camel Case (o Pascal Case)**, in cui la prima lettera di ogni parola è maiuscola.
- Non utilizzare acronimi o abbreviazioni nei nomi delle classi, a meno che l'abbreviazione sia più diffusa della forma estesa (ad esempio, *URL* o *HTML*).

Esempio: class User {}

Metodi e Funzioni

- I nomi dei metodi e delle funzioni devono seguire lo stile **Lower Camel Case (o Dromedary Case)**, in cui la prima lettera della prima parola è minuscola e la prima lettera di ogni parola successiva è maiuscola.
- Devono essere composti da un verbo o includere un verbo come prima parola.

Esempio: getName();

1.3 Definitions, acronyms, and abbreviations

Acronimo	Abbreviazione	Definizione
URL	Uniform Resource Locator	È' un indirizzo che specifica la posizione di una risorsa su Internet
HTML	HyperText Markup Language	È il linguaggio standard utilizzato per creare e strutturare le pagine web
DP	Design Pattern	Sono soluzioni riutilizzabili e consolidate per problemi comuni
DAO	Data Access Object	È un pattern di progettazione che fornisce un'interfaccia astratta per interagire con un database o altre sorgenti dati persistenti

1.4 References

- SDD

  Ci si riferisce al SDD quando si spiega l'organizzazione dei package, dato che quest'ultima è stata creata proprio a partire dalla suddivisione in subsystem.

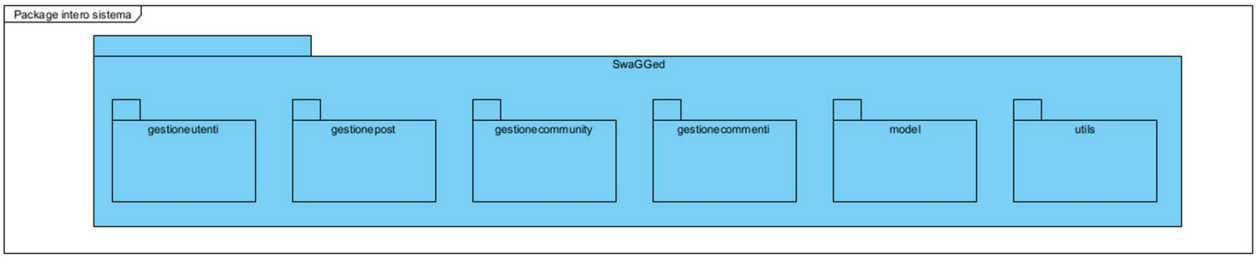
2. PACKAGES

In questa sezione viene illustrata la struttura del package principale di Swagged.

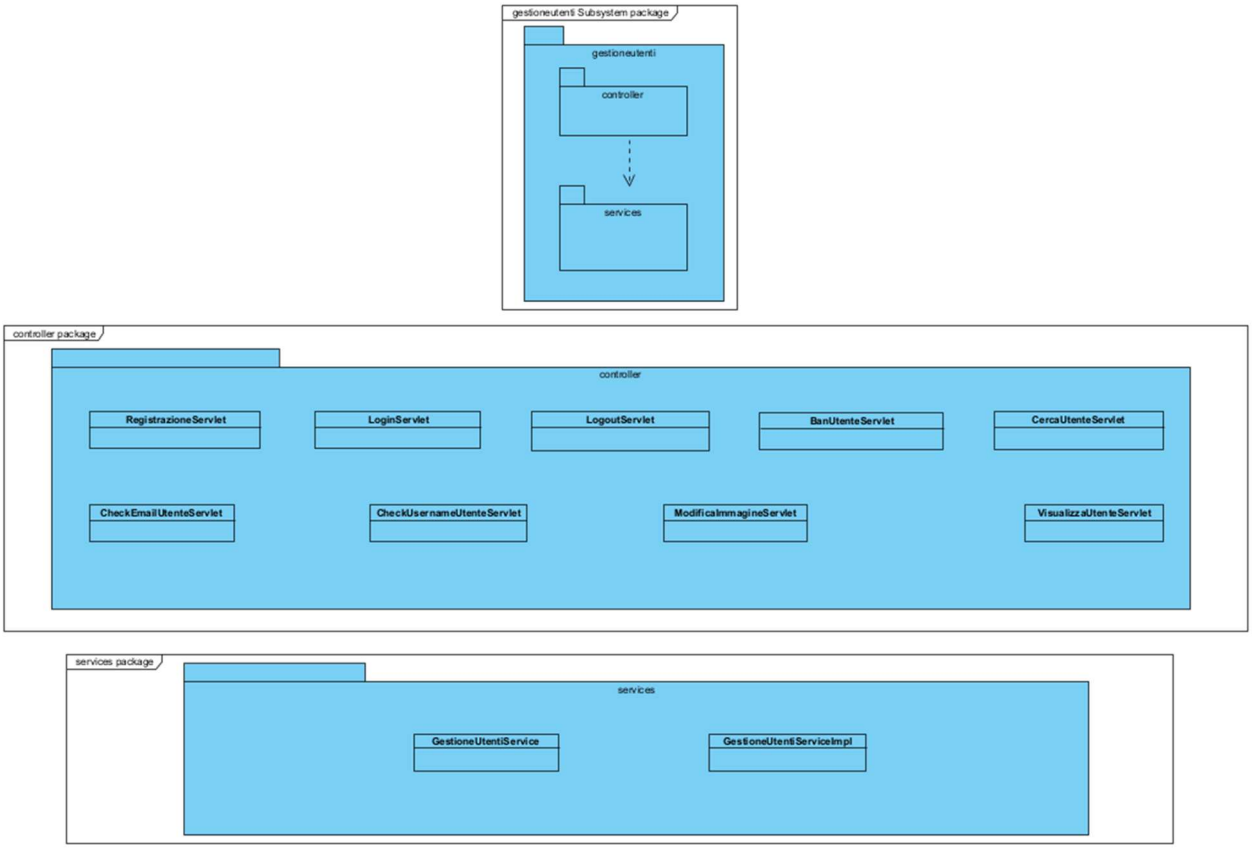
La progettazione generale si basa sulla suddivisione in partizioni e sottosistemi individuati durante la fase di System Design, come documentato nel **System Design Document (SDD)**.

La struttura adotta un'architettura a tre livelli, in cui ogni classe è assegnata a uno dei seguenti ruoli principali:

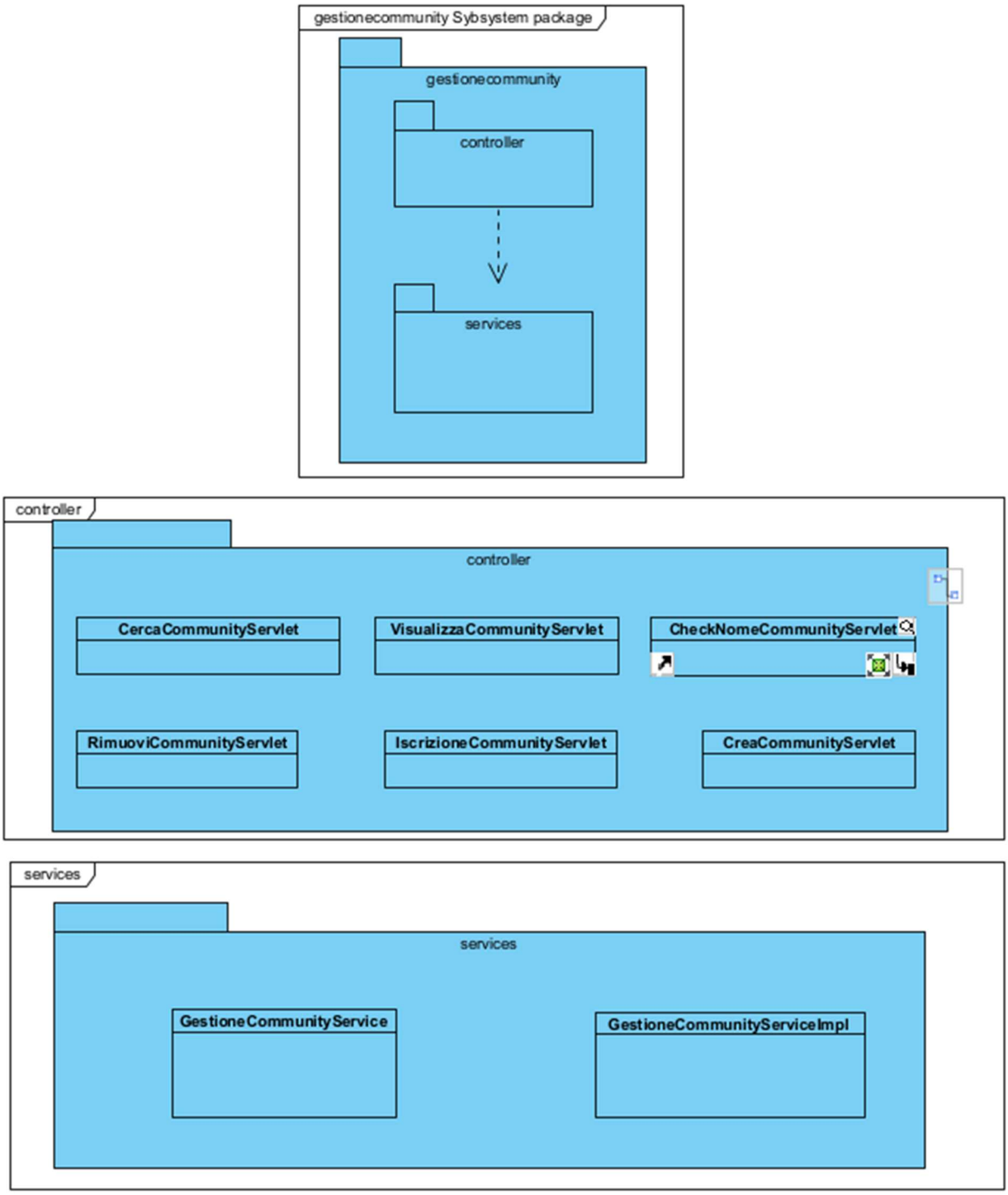
- **Presentazione**: gestione della visualizzazione e rappresentazione dei dati del modello.
- **Controllo**: supervisione delle attività di processamento interno del sistema.
- **Manipolazione dati**: gestione e trasformazione dei dati relativi all'Application Domain.



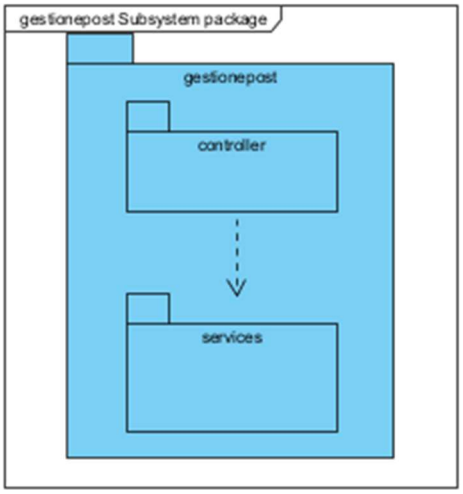
2.1 Partizione Gestione Utente

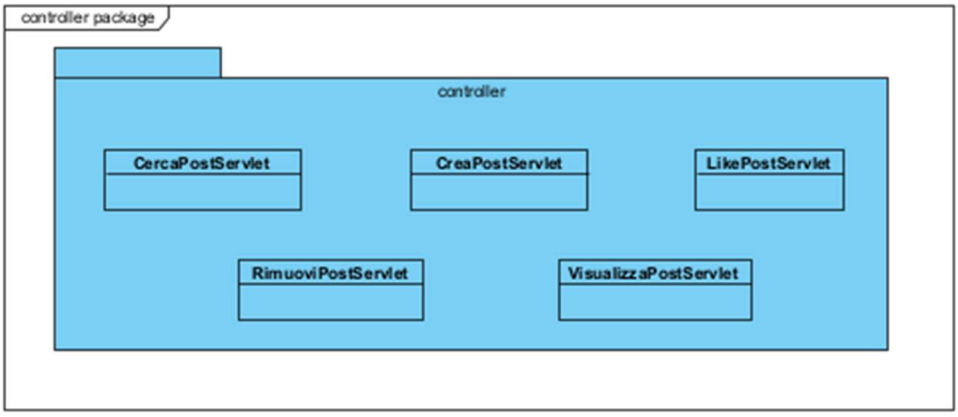


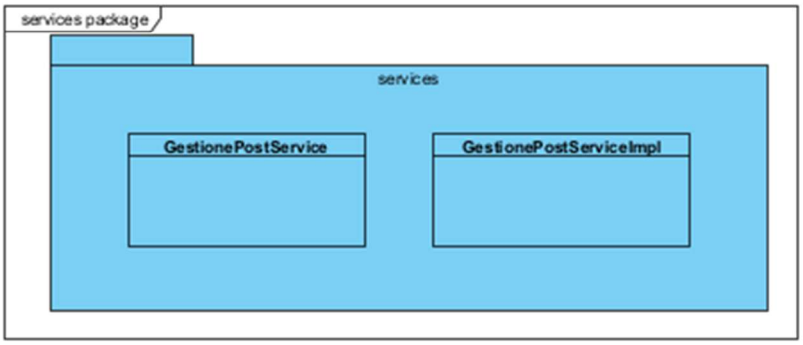
2.2 Partizione Gestione Community



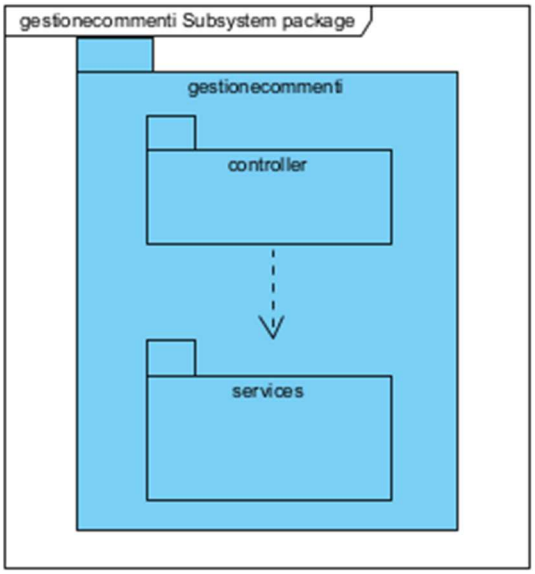
2.3 Partizione Gestione Post

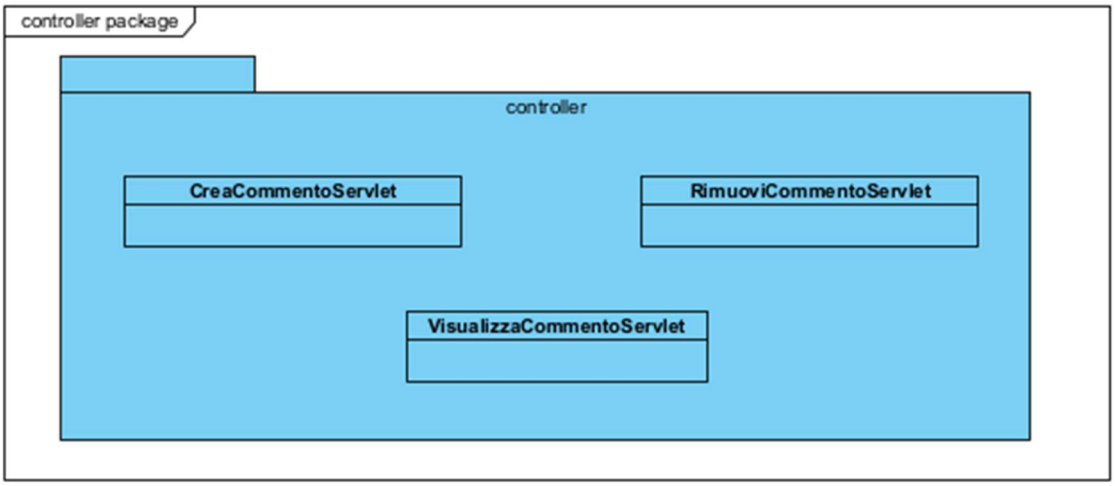


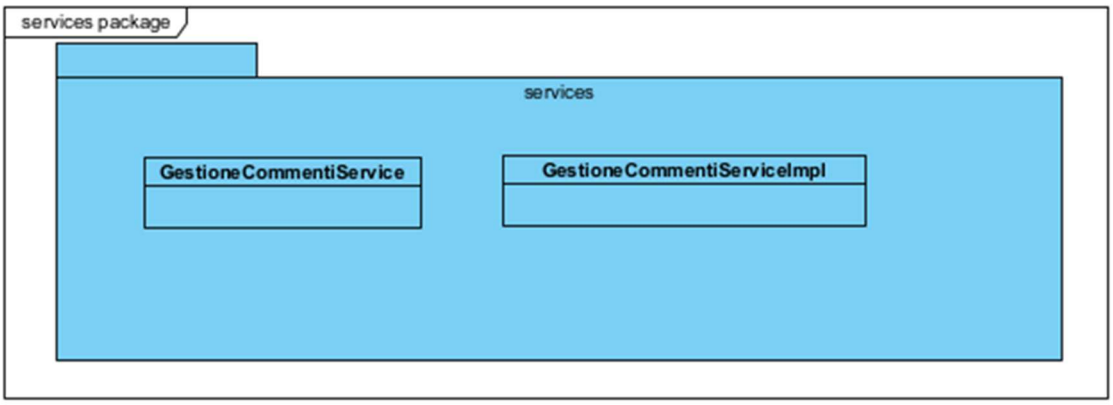




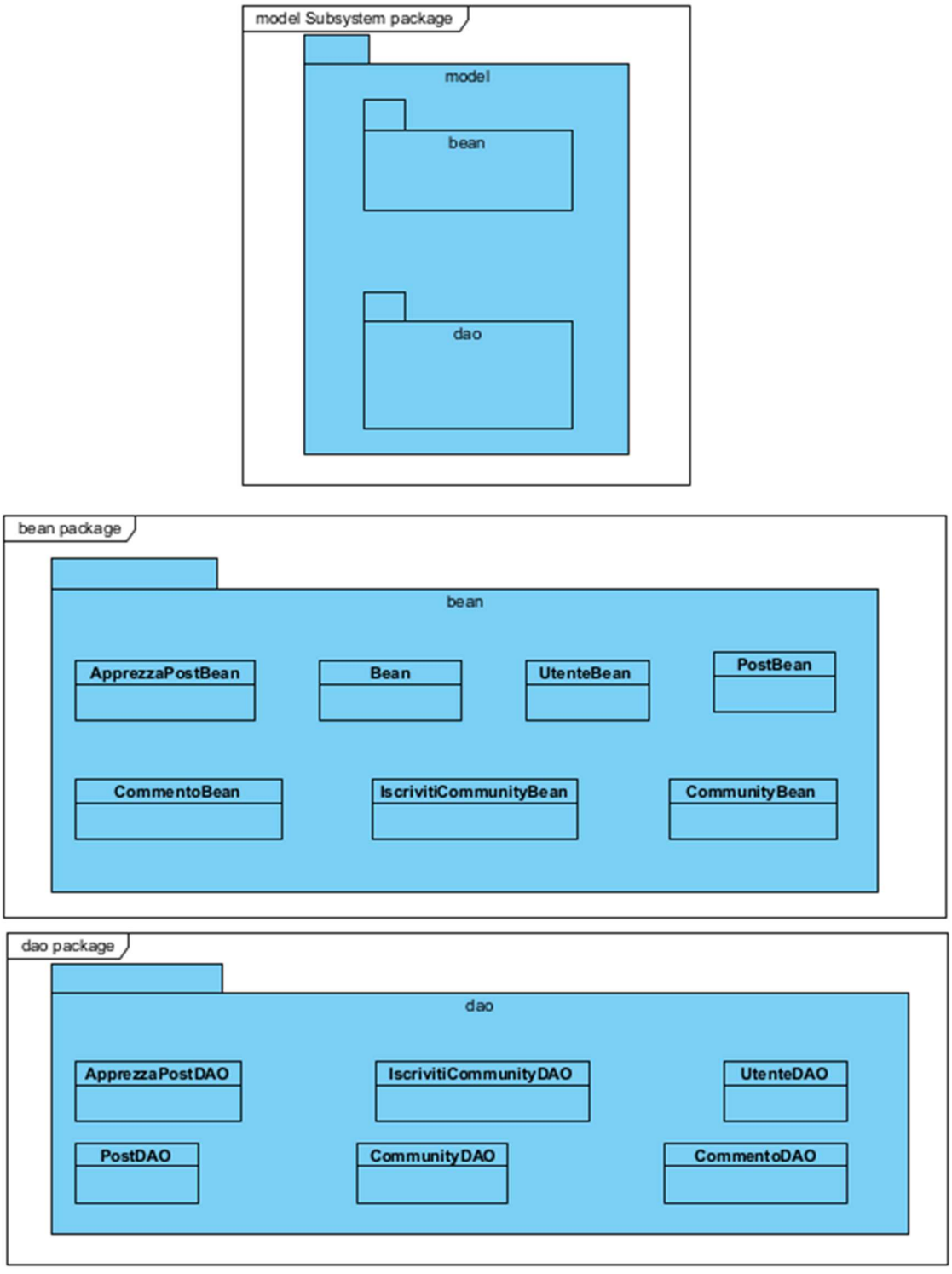
2.4 Partizione Gestione Commenti



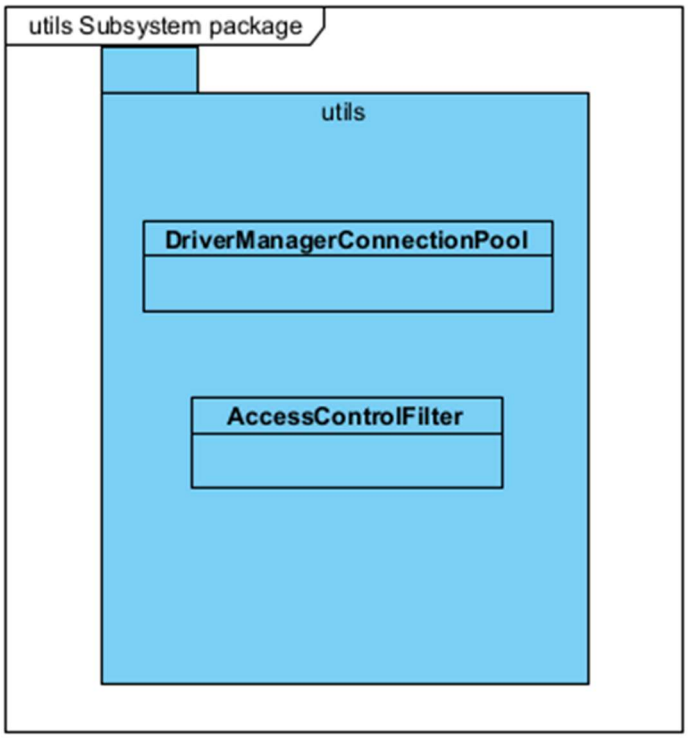




2.5 Partizione model



2.6 Partizione utils



3.1 CLASS INTERFACES

3.1 Gestione Utente

Nome Classe	GestioneUtentiService
Descrizione	Questa classe consente di amministrare le operazioni relative agli utenti
Metodi	+ban(utente: UtenteBean, email: String): boolean +cerca(substring: String):List<UtenteBean> +modificaImmagine(utente: UtenteBean, filePart: Part, servlet: GenericServlet): boolean +checkEmail(email: String): boolean +checkUsername(username: String): boolean +visualizza(username: String): UtenteBean +login(username: String, password: String): UtenteBean +registrazione(email: String, username: String, password: String, passowrdCheck: String): UtenteBean
Invariante di classe	

Nome Metodo	+ ban(utente: UtenteBean, email: String): Boolean
Descrizione	Questo metodo permette a un moderatore di inibire un utente
Pre-condizione	**Context**: GestioneUtentiService::ban(email: String): boolean
	Pre: email! = null AND email != "" AND UtenteDAO.getByEmail(email)!= null AND utente.isAdmin()
Post-condizione	**Context**: GestioneUtenti:: ban(email: String): boolean
	Post: UtenteDAO.getByEmail(email).isBandito = true

Nome Metodo	+cerca(substring: String): List<UtenteBean>
Descrizione	Questo metodo permette di cercare un utente che nell'username una data sottostringa
Pre-condizione	**Context**: GestioneUtentiService:: cerca(substring: String): List<UtenteBean>
	Pre: substring != null AND substring != ""
Post-condizione	**Context**: GestioneUtenti:: cerca(substring: String): List<UtenteBean>
	Post: result = UtenteDAO.getByUsernameSubstring(substring)

Nome Metodo	+modificaImmagine(utente: UtenteBean, filePart: Part, servlet: GenericServlet): boolean
Descrizione	Questo metodo permette di cercare un utente che nell'username una data sottostringa
Pre-condizione	**Context**: GestioneUtentiService:: modificaImmagine(filePart: Part): boolean
	Pre: filePart != null AND filePart!= "" AND utente != null
Post-condizione	**Context**: GestioneUtenti:: modificaImmagine(filePart: Part): boolean
	Post: utente.getImmagine() = filePart

Nome Metodo	+checkEmail(email: String): boolean
Descrizione	Questo metodo verifica se un email è già presente nel database
Pre-condizione	**Context**: GestioneUtentiService:: checkEmail(email: String): boolean
	Pre: email != null AND email != ""
Post-condizione	**Context**: GestioneUtenti:: checkEmail(email: String): boolean
	Post: result = UtenteDAO.getAll() -> forAll(u | u.email() != email)

Nome Metodo	+checkUsername(username: String): boolean
Descrizione	Questo metodo verifica se un username è già presente nel database
Pre-condizione	**Context**: GestioneUtentiService:: checkUsername(username: String): boolean
	Pre: username != null AND username != ""
Post-condizione	**Context**: GestioneUtenti:: checkUsername(username: String): boolean
	Post: result = UtenteDAO.getAll() -> forAll(u | u.getUsername() != username)

Nome Metodo	+visualizza(username: String): UtenteBean
Descrizione	Questo metodo restituisce bean dell'utente che si vuole visualizzare
Pre-condizione	**Context**: GestioneUtentiService:: visualizza(username: String): UtenteBean
	Pre: username != null AND username != ""
Post-condizione	**Context**: GestioneUtentiService:: visualizza(username: String): UtenteBean
	Post: result = UtenteDAO.getByUsername(username)

Nome Metodo	+login(username: String, password: String): UtenteBean
Descrizione	Questo metodo restituisce bean dell'utente che si autentica
Pre-condizione	**Context**: GestioneUtentiService:: login(username: String, password: String): UtenteBean
	Pre: username != null AND username != "" AND password != null AND password != “”
Post-condizione	**Context**: GestioneUtentiService:: login(username: String, password: String): UtenteBean
	Post: Utente = UtenteDAO.getByUsername(username)

Nome Metodo	+registrazione(email: String, username: String, password: String, passowrdCheck: String): UtenteBean
Descrizione	Questo metodo restituisce bean dell'utente che si vuole registra
Pre-condizione	**Context**: GestioneUtentiService:: registrazione(email: String, username: String, password: String, passowrdCheck: String): UtenteBean
	Pre: username != null AND username != "" AND email != null AND email != “” AND password != null AND password != “” AND passwordCheck != “” AND passwordChecl != null
Post-condizione	**Context**: GestioneUtentiService:: registrazione(email: String, username: String, password: String, passowrdCheck: String): UtenteBean
	Post: result = new UtenteBean

Nome Classe	UtenteDAO
Descrizione	Questa classe consente di amministrare le operazioni relative agli utenti
Metodi	+save(utente: UtenteBean): boolean +delete(email: String): boolean +update(utente: UtenteBean, email: String): boolean +getAll(): List<UtenteBean> +getByEmail(email: String): UtenteBean +getByUsername(username: String): UtenteBean +getByUsernameSubstring(substring: String): List<UtenteBean>
Invariante di classe	

Nome Metodo	+save(utente: UtenteBean): boolean
Descrizione	Questo metodo consente di inserire un nuovo utente nel database
Pre-condizione	**Context**: UtenteDAO::create(utente: UtenteBean): boolean
	Pre: utente != null AND utente.email != null AND utente.email != "" AND utente.username != null AND utente.username != "" AND utente.password != null AND utente.password != ""
Post-condizione	**Context**: UtenteDAO::create(utente: UtenteBean): boolean
	Post: self.getAll() -> include(utente)

Nome Metodo	+delete(email: String): boolean
Descrizione	Questo metodo consente di rimuovere un utente dal database
Pre-condizione	**Context**: UtenteDAO::delete(utente: UtenteBean): boolean
	Pre: email != null AND email != "" AND self.getByEmail(email) = utente
Post-condizione	**Context**: UtenteDAO::delete(utente: UtenteBean): boolean
	Post: self.getByEmail(email) -> null

Nome Metodo	+update(utente: UtenteBean, email: String): boolean
Descrizione	Questo metodo consente di aggiornare i dati di un utente esistente nel database
Pre-condizione	**Context**: UtenteDAO::update(utente: UtenteBean, email: String): boolean
	Pre: email != null AND email != "" AND self.getByEmail(email) = utente
Post-condizione	**Context**: UtenteDAO::update(utente: UtenteBean, email: String): boolean
	Post: self.getByEmail(email) = utente

Nome Metodo	+getAll(): List<UtenteBean>
Descrizione	Questo metodo restituisce tutti gli utenti presenti nel database
Pre-condizione	**Context**: UtenteDAO:: getAll(): List<UtenteBean> **Pre:**
Post-condizione	**Context**: UtenteDAO:: getAll(): List<UtenteBean> **Post**:

Nome Metodo	+getByEmail(email: String): UtenteBean
Descrizione	Questo metodo restituisce l'utente a cui è associata l'email
Pre-condizione	**Context**: UtenteDAO::getByEmail(email: String): UtenteBean
	Pre: email != null AND email != ""
Post-condizione	**Context**: UtenteDAO::getByEmail(email: String): UtenteBean
	Post: self.getAll() -> exist(u | u.email = email)

Nome Metodo	+getByUsername(username: String): UtenteBean
Descrizione	Questo metodo restituisce l'utente a cui è associato l'username
Pre-condizione	**Context**: UtenteDAO::getByUsername(username: String): UtenteBean
	Pre: username != null AND username != ""
Post-condizione	**Context**: UtenteDAO::getByUsername(username: String): UtenteBean **Post**: self.getAll() -> exist(u | u.username = username)

Nome Metodo	+getByUsernameSubstring(substring: String): List<UtenteBean>
Descrizione	Questo metodo restituisce gli utenti che hanno nell'username la sottostringa passata
Pre-condizione	**Context**: UtenteDAO::getByUsernameSubstring(substring: String): List<UtenteBean>
	Pre: substring != null AND substring != ""
Post-condizione	**Context**: UtenteDAO::getByUsernameSubstring(substring: String): List<UtenteBean>
	Post: result = self.getAll() -> forAll(u | u.username.contains(substring))

3.2 Gestione Post

Nome Classe	GestionePostService
Descrizione	Questa classe consente di amministrare le operazioni relative ai post
Metodi	+create(titolo: String, corpo: String, immagine: Part, utente: UtenteBean, comunityNome: String, servlet: GenericServlet): PostBean +remove(id: Integer, utente: UtenteBean): boolean +cerca(substring: String): List<PostBean> +visualizza(id: Integer): PostBean +like(utente: UtenteBean, postId: Integer): boolean
Invariante di classe	

Nome Metodo	+create(titolo: String, corpo: String, immagine: Part, utente: UtenteBean, comunityNome: String, servlet: GenericServlet): PostBean
Descrizione	Questo metodo permette di creare un post
Pre-condizione	**Context**: GestionePostService:: create(titolo: String, corpo: String, immagine: Part, utente: UtenteBean, comunityNome: String, servlet: GenericServlet): PostBean
	Pre: titolo != null AND titolo!= "" AND email != null AND email != “” AND UtenteDAO.getByEmail(email) != null AND communityNome != null AND communityNome != “” AND CommunityDAO.getByNome(communityNome) != null AND utente != null
Post-condizione	**Context**: GestionePost:: create(titolo: String, corpo: String, immagine: Part, utente: UtenteBean, comunityNome: String, servlet: GenericServlet): PostBean
	Post: PostDAO.getAll() -> include (post) AND utente.get(“postCreati”) -> include post

Nome Metodo	+ remove(id: Integer, utente: UtenteBean): Boolean
Descrizione	Questo metodo permette di rimuovere un post
Pre-condizione	**Context**: GestionePostService:: remove(id: Integer, utente: UtenteBean): boolean
	Pre: id != null AND id != “” AND PostDAO.getById(id) != null AND utente != null
Post-condizione	**Context**: GestionePost:: remove(id: Integer, utente: UtenteBean): boolean
	Post: PostDAO.getById(id) = null AND utente.get(“postCreati”) -> !include(post)

Nome Metodo	+ cerca(substring: String): List<PostBean>
Descrizione	Questo metodo permette di cercare un post
Pre-condizione	**Context**: GestionePostService:: cerca(substring: String): List<PostBean>
	Pre: substring != null AND substring != “”
Post-condizione	**Context**: cerca(substring: String): List<PostBean>
	Post: result = PostDAO.getByTitleSubstring(substring)

Nome Metodo	+ visualizza(int id): PostBean
Descrizione	Questo metodo permette di caricare la homePage con dei post
Pre-condizione	**Context**: GestionePost:: visualizza(int id): PostBean
	Pre: Id != null
Post-condizione	**Context**: GestionePost:: visualizza(int id): PostBean
	Post: result = PostDAO.getById(id)

Nome Metodo	+ like(utente: UtenteBean, postId: Integer): boolean
Descrizione	Questo metodo permette di apprezzare post
Pre-condizione	**Context**: GestionePostService:: like(utenteEmail: String, postId: Integer): boolean
	Pre: utente != null AND postId != null AND postId != “” AND PostDAO:getByPostId(id) != null
Post-condizione	**Context**: GestionePostService:: like(utenteEmail: String, postId: Integer): boolean
	Post: post.likes = @pre.likes ± 1 AND utente.get(“postApprezzati”) -> include(post)

Nome Classe	PostDAO
Descrizione	Questa classe consente di amministrare le operazioni relative ai post
Metodi	+save(post: PostBean): boolean +delete(post: PostBean, id: Integer): boolean +update(post: PostBean): boolean +getById(id: Integer): PostBean +getAll(): List<PostBean> +getByEmail(email: String): List<PostBean> +getByTitleSubstring(substring: String): List<PostBean> +getByCommunityNomw(communityNome: String): List<PostBean> +getByDate(): List<PostBean>
Invariante di classe	

Nome Metodo	+save(post: PostBean): boolean
Descrizione	Questo metodo consente di inserire un nuovo post nel database
Pre-condizione	**Context**: PostDAO::create(post: PostBean): boolean
	Pre: post.titolo != null AND post.titolo != “”
Post-condizione	**Context**: PostDAO::create(post: PostBean): boolean
	Post: self.getAll() -> include(post)

Nome Metodo	+delete(id: Integer): boolean
Descrizione	Questo metodo consente di rimuovere un post dal database
Pre-condizione	**Context**: PostDAO:: delete(id: Integer): boolean
	Pre: id != null AND self.getById(id) != null
Post-condizione	**Context**: PostDAO:: delete(id: Integer): boolean
	Post: self.getById(id) = null

Nome Metodo	+update(post: PostBean): boolean
Descrizione	Questo metodo consente di aggiornare i dati di un post esistente nel database
Pre-condizione	**Context**: PostDAO:: update(post: PostBean): boolean
	Pre: self.getById(post.id) = post
Post-condizione	**Context**: PostDAO:: update(post: PostBean): boolean **Post**: self.getByIdl(post.id) = post

Nome Metodo	+getAll(): List<PostBean>
Descrizione	Questo metodo restituisce tutti i post presenti nel database
Pre-condizione	**Context**: PostDAO:: getAll(): List<PostBean>
	Pre:
Post-condizione	**Context**: PostDAO:: getAll(): List<PostBean>
	Post:

Nome Metodo	+getById(id: Integer): List<PostBean>
Descrizione	Questo metodo restituisce tutti i post presenti nel database
Pre-condizione	**Context**: PostDAO:: getById(id: Integer): List<PostBean>
	Pre: id != null AND self.getAll() -> exist(p | p.id = id)
Post-condizione	**Context**: PostDAO:: getById(id: Integer): List<PostBean>
	Post: self.getAll() -> exist(p | p.id = id)

Nome Metodo	+getByEmail(email: String): List<PostBean>
Descrizione	Questo metodo restituisce tutti i post a cui è associata l'email
Pre-condizione	**Context**: PostDAO:: getByEmail(email: String): List<PostBean>
	Pre: email != null AND email != "" AND UtenteDAO.getByEmail(email) != null
Post-condizione	**Context**: PostDAO:: getByEmail(email: String): List(PostBean
	Post: result = self.getAll() -> forAll(p | p.utenteEmail = email)

Nome Metodo	+getByTitleSubstring(substring: String): List<PostBean>
Descrizione	Questo metodo restituisce i post che hanno nel titolo la sottostringa
Pre-condizione	**Context**: PostDAO:: getByTitleSubstring(substring: String): List<PostBean>
	Pre: substring != null AND substring != “”
Post-condizione	**Context**: PostDAO:: getByTitleSubstring(substring: String): List<PostBean>
	Post: result = self.getAll() -> forAll(p | p.titolo.contains(substring))

Nome Metodo	+getByCommunityNome(communityNome: String): List<PostBean>
Descrizione	Questo metodo restituisce tutti i post a cui è associato l’id della community
Pre-condizione	**Context**: PostDAO:: getByCommunityNome(communityNome: String): List<PostBean>
	Pre: communityId != null AND communityNome != “”
Post-condizione	**Context**: PostDAO:: getByCommunityNome(communityNome: String): List<PostBean>
	Post: result = self.getAll() -> forAll(p | p.communityNome = communityNome)

Nome Metodo	+getByDate(): List<PostBean>
Descrizione	Questo metodo restituisce tutti i post a cui è associato l'id della community
Pre-condizione	**Context**: PostDAO:: getByDate(): List<PostBean>
	Pre:
Post-condizione	**Context**: PostDAO:: getByDate(): List<PostBean>
	Post: result = self.getAll() -> forAll(p | p.sort(p.dataCreazione))

Nome Classe	ApprezzaPostDAO
Descrizione	Questa classe consente di amministrare le operazioni relative agli apprezzamenti (like) dei post.
Metodi	+save(apprezzaPost: ApprezzaPostBean): boolean +delete(email: String, postId: Integer): boolean +getByKey(email: String, postId: Integer): ApprezzaPostBean +getByEmail(email: String): List<ApprezzaPostBean>
Invariante di classe	

Nome Metodo	+ save(segueCommunity: SegueCommunityBean): boolean
Descrizione	Questo metodo consente di inserire un nuovo apprezzamento per un post nel database
Pre-condizione	**Context**: ApprezzaPostDAO::save(apprezzaPost: ApprezzaPostBean): boolean
	Pre: apprezzaPost.email != null AND apprezzaPost.postId != null
Post-condizione	**Context**: ApprezzaPostDAO::save(apprezzaPost: ApprezzaPostBean): boolean **Post**: self.getByKey(apprezzaPost.email, apprezzaPost.postId) != null

Nome Metodo	+ delete(email: String, nome: String): boolean
Descrizione	Questo metodo consente di rimuovere un apprezzamento (like) per un post dal database
Pre-condizione	**Context**: ApprezzaPostDAO::delete(email: String, postId: Integer): boolean
	Pre: email != null AND email != "" AND UtenteDAO.getByEmail(email) != null AND postId != null AND PostDAO.getById(postId) != null
Post-condizione	**Context**: ApprezzaPostDAO::delete(email: String, postId: Integer): boolean
	Post: self.getByKey(email, postId) = null

Nome Metodo	+getByKey(email: String, nome: String): SegueCommunityBean
Descrizione	Questo metodo restituisce l'apprezzamento (like) di un post se esiste.
Pre-condizione	**Context**: ApprezzaPostDAO::getByKey(email: String, postId: Integer): ApprezzaPostBean
	Pre: email != null AND email != "" AND postId != null
Post-condizione	**Context**: ApprezzaPostDAO::getByKey(email: String, postId: Integer): ApprezzaPostBean
	Post: self.getAll() -> exist(p | p.id= postId AND p.email = email)

Nome Metodo	+ getByEmail(email: String): List<SegueCommunityBean>
Descrizione	Questo metodo restituisce la relazione se esiste
Pre-condizione	**Context**: ApprezzaPostDAO:: getByEmail(email: String): List<ApprezzaPostBean>
	Pre: email != null AND email != “”
Post-condizione	**Context**: ApprezzaPostDAO:: getByEmail(email: String): List<ApprezzaPostBean>
	Post: self.getAll() -> exist(p | p.email = email)

3.3 Gestione Commenti

Nome Classe	GestioneCommentiService
Descrizione	Questa classe consente di amministrare le operazioni relative ai commenti
Metodi	+create(postId: Integer, corpo: String, utenteEmail: String): CommentoBean +remove(id: Integer, int postId, utente: UtenteBean): boolean +visualizza(postId: Integer): List<CommentoBean>
Invariante di classe	

Nome Metodo	+ create(postId: Integer, corpo: String, utenteEmail = String) +remove(id: Integer): CommentoBean
Descrizione	Questo metodo permette di creare un commento
Pre-condizione	**Context**: GestioneCommentiService:: create(postId: Integer, corpo: String, utenteEmail = String): CommentoBean
	Pre: corpo != null AND corpo!= "" AND utenteEmail != null AND utenteEmail != “” AND UtenteDAO.getByEmail(utenteEmail) != null AND postId != null AND PostDAO.getById(postId) != null
Post-condizione	**Context**: GestioneCommenti:: create(postId: Integer, corpo: String, utenteEmail = String): CommentoBean
	Post: CommentoDAO.getAll() -> include (commento) AND utente.get(“commentiCreati”) -> include(commento)

Nome Metodo	+ remove(id: Integer, int postId, utente: UtenteBean): boolean
Descrizione	Questo metodo permette di rimuovere un commento
Pre-condizione	**Context**: GestioneCommentiService:: remove(id: Integer, int postId, utente: UtenteBean): boolean
	Pre: id != null AND id != “” AND CommentoDAO.getById(id) != null AND utente != null AND postId != null
Post-condizione	**Context**: GestioneCommenti:: remove(id: Integer, int postId, utente: UtenteBean): boolean
	Post: CommentoDAO.getById(id) = null AND utente.get(“commentiCreati”) -> !include(commento)

Nome Metodo	+ visualizza(postId: Integer): List<CommentoBean>
Descrizione	Questo metodo permette di caricare i commenti di un post
Pre-condizione	**Context**: GestioneCommentiService:: visualizza(postId: Integer): List<CommentoBean>
	Pre: postId != null AND PostDAO.doGetById(postId) != null
Post-condizione	**Context**: GestioneCommenti:: visualizza(postId: Integer): List<CommentoBean> **Post**: result = CommentoDAO.getAll() -> forAll(c | c.postId = postId)

Nome Classe	CommentoDAO
Descrizione	Questa classe consente di amministrare le operazioni relative commenti
Metodi	+save(commento: CommentoBean): boolean +delete(id: Integer): boolean + update(commento: CommentoBean): boolean +getAll(): List<CommentoBean> +getById(id: Integer): CommentoBean +getByPostId(postId: Integer): List<CommentoBean> +getByUtenteEmail(email: String): List<CommentoBean>
Invariante di classe	

Nome Metodo	+save(bean: CommentoBean): boolean
Descrizione	Questo metodo consente di inserire un nuovo commento nel database
Pre-condizione	**Context**: CommentoDAO::save(commento: CommentoBean): boolean
	Pre: commento.corpo != null AND commento.corpo != ""
Post-condizione	**Context**: CommentoDAO::save(commento: CommentoBean): boolean
	Post: self.getAll() -> include(bean)

Nome Metodo	+delete(id: Integer): boolean
Descrizione	Questo metodo consente di rimuovere un commento dal database
Pre-condizione	**Context**: CommentoDAO::delete(id: Integer): boolean
	Pre: id != null AND self.getById(id) != null
Post-condizione	**Context**: CommentoDAO::delete(id: Integer): boolean
	Post: self.getById(id) = null

Nome Metodo	+update(commento: CommentoBean): boolean
Descrizione	Questo metodo aggiorna le informazioni di un commento nel database
Pre-condizione	**Context**: CommentoDAO::update(commento: CommentoBean): boolean **Pre**: self.getById(commento.id) != null
Post-condizione	**Context**: CommentoDAO::update(commento: CommentoBean): boolean
	Post: self.getById(commento.id) = commento

Nome Metodo	+getAll(): List<CommentoBean>
Descrizione	Questo metodo restituisce tutti i commenti presenti nel database
Pre-condizione	**Context**: CommentoDAO::getAll(): List<CommentoBean>
	Pre:
Post-condizione	**Context**: CommentoDAO::getAll(): List<CommentoBean>
	Post:

Nome Metodo	+getById(id: Integer): CommentoBean
Descrizione	Questo metodo restituisce il commento a cui è asscoiato l'id
Pre-condizione	**Context**: CommentoDAO::getById(id: Integer): CommentoBean
	Pre: id != null AND self.getAll() -> exist(c | c.id = id)
Post-condizione	**Context**: CommentoDAO::getById(id: Integer): CommentoBean
	Post: self.getAll() -> exist(c | c.id = id)

Nome Metodo	+getByPostId(postId: Integer): List<CommentoBean>
Descrizione	Questo metodo restituisce tutti i commenti a cui è associato l'id del post
Pre-condizione	**Context**: CommentoDAO::getByPostId(postId: Integer): List<CommentoBean>
	Pre: postId != null AND PostDAO.getById(postId) != null
Post-condizione	**Context**: CommentoDAO::getByPostId(postId: Integer): List<CommentoBean>
	Post: result = self.getAll() -> forAll(c | c.postId= postId)

Nome Metodo	+getByUtenteEmail(email: String): List<CommentoBean>
Descrizione	Questo metodo restituisce tutti i commenti a cui è associata l'email
Pre-condizione	**Context**: CommentoDAO::getByUtenteEmail(email: String): List<CommentoBean>
	Pre: email != null AND email != "" AND UtenteDAO:getByEmail(email) != null
Post-condizione	**Context**: CommentoDAO::getByUtenteEmail(email: String): List<CommentoBean>
	Post: result = self.getAll() -> forAll(c | c.utenteEmail = email)

3.4 Gestione Community

Nome Classe	GestioneCommunityService
Descrizione	Questa classe consente di amministrare le operazioni relative alle community
Metodi	+create(nome: String, descrizione: String, utente: UtenteBean): CommunityBean +remove(community: CommuntiyBean, utente: UtenteBean): boolean +visualizza(nome: String): CommunityBean +iscrizione(utente: UtenteBean, communityNome: String): boolean +cerca(substring: String): List<CommunityBean> +checkNome(nome: String): boolean
Invariante di classe	

Nome Metodo	+ create(nome: String, descrizione: String, utente: UtenteBean): CommunityBean
Descrizione	Questo metodo permette di creare una community
Pre-condizione	**Context**: GestioneCommunityService create(nome: String, descrizione: String, utente: UtenteBean): CommunityBean
	Pre: nome != null AND nome!= "" AND utente != null AND UtenteDAO.getByEmail(email) != null
Post-condizione	**Context**: GestioneCommunityService create(nome: String, descrizione: String, utente: UtenteBean): CommunityBean
	Post: CommunityDAO.getAll() -> include (community) AND utente.get("communityCreate") -> include(community)

Nome Metodo	+ remove(community: CommuntiyBean, utente: UtenteBean): boolean
Descrizione	Questo metodo permette di rimuovere una community
Pre-condizione	**Context**: GestioneCommunityService:: remove(community: CommuntiyBean, utente: UtenteBean): boolean
	Pre: community != null AND CommunityDAO.getByNome(nome) != null AND utente != null
Post-condizione	**Context**: GestioneCommunityService:: remove(community: CommuntiyBean, utente: UtenteBean): boolean
	Post: CommunityDAO.getByNome(nome) = null AND utente.get("communityCreate") -> !include(community)

Nome Metodo	+ visualizza(nome: String): CommunityBean
Descrizione	Questo metodo permette di visualizzare una community
Pre-condizione	**Context**: GestioneCommunityService:: visualizza(nome: String): CommunityBean
	Pre: nome != null AND nome != "" AND CommunityDAO.doGetByNome(nome) != null
Post-condizione	**Context**: GestioneCommunityService:: visualizza(nome: String): CommunityBean
	Post: result = CommunityDAO.getByNome(nome)

Nome Metodo	+ iscrizione(utente: UtenteBean, communityNome: String): boolean
Descrizione	Questo metodo permette di iscriversi a una community
Pre-condizione	**Context**: GestioneCommunityService:: iscrizione(utente: UtenteBean, communityNome: String): boolean
	Pre: utente != null AND communityNome != null AND CommunityDAO:getByNome(nome) != null
Post-condizione	**Context**: GestioneCommunityService:: iscrizione(utente: UtenteBean, communityNome: String): boolean
	Post: community.iscritti= @pre.iscritti ± 1

Nome Metodo	+ cerca(substring: String): List<CommunityBean>
Descrizione	Questo metodo permette di cercare una community
Pre-condizione	**Context**: GestioneCommunityService:: cerca(substring: String): List<CommunityBean>
	Pre: substring != null AND substring != “”
Post-condizione	**Context**: GestioneCommunityService:: cerca(substring: String): List<CommunityBean>
	Post: result = CommunityDAO.getByNameSubstring(substring)

Nome Metodo	+checkNome(nome: String): boolean
Descrizione	Questo metodo verifica se un nome è già presente nel database
Pre-condizione	**Context**: GestioneCommunityService:: checkNome(nome: String): boolean
	Pre: nome != null AND nome != ""
Post-condizione	**Context**: GestioneCommunityService:: checkNome(nome: String): boolean
	Post: result = CommunityDAO.getAll() -> forAll(c | c.getnome() != nome)

Nome Classe	IscrivitiCommunityDAO
Descrizione	Questa classe consente di amministrare le operazioni relative alle iscrizioni alle community
Metodi	+save(segueCommunity: IscrivitiCommunityBean): boolean +delete(email: String, nome: String): boolean +getByKey(email: String, nome: String): IscrivitiCommunityBean +getByEmail(email: String): List< IscrivitiCommunityBean >
Invariante di classe	

Nome Metodo	+ save(segueCommunity: SegueCommunityBean): boolean
Descrizione	Questo metodo consente di inserire una nuova relazione nel database
Pre-condizione	**Context**: SegueCommunityDAO:: save(segueCommunity: SegueCommunityBean): boolean
	Pre: segueCommunity.nome != null AND segueCommunity.email != null
Post-condizione	**Context**: SegueCommunityDAO:: save(segueCommunity: SegueCommunityBean): boolean
	Post: self.getByKey(segueCommunity.email, segueCommunity.nome) != null

Nome Metodo	+ delete(email: String, nome: String): boolean
Descrizione	Questo metodo consente di rimuovere una relazione dal database
Pre-condizione	**Context**: SegueCommunityDAO:: delete(email: String, nome: String): boolean
	Pre: email != null AND email!= “” AND UtenteDAO.getByEmail(email) != null AND nome != null AND nome != “” AND CommunityDAO.getByNome(nome) != null
Post-condizione	**Context**: SegueCommunityDAO:: delete(email: String, nome: String): Boolean **Post**: self.getByKey(email, nome) = null

Nome Metodo	+getByKey(email: String, nome: String): SegueCommunityBean
Descrizione	Questo metodo restituisce la relazione se esiste
Pre-condizione	**Context**: SegueCommunityDAO:: getByKey(email: String, nome: String): SegueCommunityBean
	Pre: nome != null AND nome != “” AND email != null AND email != “”
Post-condizione	**Context**: SegueCommunityDAO:: getByKey(email: String, nome: String): SegueCommunityBean
	Post: self.getAll() -> exist(c | c.nome = nome AND c.email = email)

Nome Metodo	+ getByEmail(email: String): List<SegueCommunityBean>
Descrizione	Questo metodo restituisce la relazione se esiste
Pre-condizione	**Context**: SegueCommunityDAO:: getByEmail(email: String): List<SegueCommunityBean>
	Pre: email != null AND email != “”
Post-condizione	**Context**: SegueCommunityDAO:: getByEmail(email: String): List<SegueCommunityBean>
	Post: self.getAll() -> exist(c | c.email = email)

Nome Classe	CommunityDAO
Descrizione	Questa classe consente di amministrare le operazioni relative alle community
Metodi	+save(community: CommunityBean): boolean +delete(nome: String): boolean +update(community: CommunityBean): boolean +getByNome(nome: String): CommunityBean +getAll(): List<CommunityBean> +getByEmail(email: String): List<CommunityBean> +getByNameSubstring(substring: String): List<CommunityBean> +checkNome(nome: String): boolean
Invariante di classe	

Nome Metodo	+save(community: CommunityBean): boolean
Descrizione	Questo metodo consente di inserire una nuova community nel database
Pre-condizione	**Context**: CommunityDAO:: save(community: CommunityBean): boolean
	Pre: community.nome != null AND community.nome != “” AND self.getByNome = null
Post-condizione	**Context**: CommunityDAO:: save(community: CommunityBean): boolean **Post**: self.getAll() -> include(community)

Nome Metodo	+delete (nome: String): boolean
Descrizione	Questo metodo consente di rimuovere una community dal database
Pre-condizione	**Context**: CommunityDAO:: delete(nome: String): boolean
	Pre: nome != null AND nome != “” AND self.getByNome(nome) != null
Post-condizione	**Context**: CommunityDAO:: delete(nome: String): boolean
	Post: self.getByNome(nome) = null

Nome Metodo	+update(community: CommunityBean): boolean
Descrizione	Questo metodo aggiorna le informazioni di una community nel database
Pre-condizione	**Context**: CommunityDAO:: update(community: CommunityBean): boolean
	Pre: self.getByNome(nome) = community
Post-condizione	**Context**: CommunityDAO:: update(community: CommunityBean): boolean
	Post: self.getByNome(nome) = community

Nome Metodo	+getByNome(nome: String): CommunityBean
Descrizione	Questo metodo restituisce la community a cui è asscoiato il nome
Pre-condizione	**Context**: CommunityDAO:: getByNome(nome: String): CommunityBean
	Pre: nome != null AND nome != “”
Post-condizione	**Context**: CommunityDAO:: getByNome(nome: String): CommunityBean
	Post: self.getAll() -> exist(c | c.nome = nome)

Nome Metodo	+getAll(): List<CommunityBean>
Descrizione	Questo metodo restituisce tutte le community presenti nel database
Pre-condizione	**Context**: CommunityDAO:: getAll(): List<CommunityBean>
	Pre:
Post-condizione	**Context**: CommunityDAO:: getAll(): List<CommunityBean>
	Post:

Nome Metodo	+getByEmail(email: String): List<CommunityBean>
Descrizione	Questo metodo restituisce tutte le community a cui è associata l'e-mail
Pre-condizione	**Context**: CommunityDAO:: getByEmail(email: String): List<CommunityBean>
	Pre: email != null AND email != “” AND UtenteDAO.getByEmail(email) != null
Post-condizione	**Context**: CommunityDAO:: getByEmail(email: String): List<CommunityBean>
	Post: result = self.getAll() -> forAll(c | c.utenteEmail = email)

Nome Metodo	+getByNameSubstring(substring: String): List<CommunityBean>
Descrizione	Questo metodo restituisce le community che hanno nel nome la sottostringa
Pre-condizione	**Context**: CommunityDAO:: getByNameSubstring(substring: String): List<CommunityBean>
	Pre: substring != null AND substring != “”
Post-condizione	**Context**: CommunityDAO:: getByNameSubstring(substring: String): List<CommunityBean>
	Post: result = self.getAll() -> forAll(c | c.nome.contains(substring))

Nome Metodo	+checkNome(nome: String): Boolean
Descrizione	Questo metodo verifica l’esistenza di una community nel database a cui è associato il nome
Pre-condizione	**Context**: **Pre**: nome != null AND nome != ""
Post-condizione	**Context**: **Post**: result = self.getAll() -> exists(c | c.nome == nome)

4 DESIGN PATTERN

I seguenti **Design Patterns** sono stati adottati per l'implementazione:

- **DAO (Data Access Object)**
- **Façade**

DAO

Il pattern DAO è stato utilizzato per semplificare le operazioni di accesso e manipolazione dei dati nel database del sistema. Per ogni tipo di entità definita a livello applicativo, è stato sviluppato un DAO specifico, garantendo una gestione dei dati chiara e strutturata.

Façade

Il pattern Façade è stato impiegato per fornire un'interfaccia semplificata per interagire con le operazioni associate a un **Post**. In particolare, il Façade gestisce le operazioni di apprezzamento, commento, segnalazione e salvataggio di un post, evitando la necessità di invocazioni separate per ciascuna di esse. Questo approccio riduce la complessità e il rischio di utilizzi errati, migliorando la manutenibilità del sistema e l'accoppiamento tra le componenti.