UNIVERSITÀ DEGLI STUDI
DI SALERNO

UNISA

Object Design Document
HAMPLANET.BLOG

Partecipanti:
Danese Vincenzo 0512107959
Nappi Nicola 0512109534
Russo Giovanni 0512118684
Vitiello Catello 0512110338

Presentato a:
Prof. Andrea De Lucia
Dipartimento di informatica

versione 1.1 | 4 aprile 2024

Indice

1 INTRODUZIONE

1.1 OBJECT DESIGN TRADE-OFFS

Il principale trade-off individuato in fase di System Design è Space vs Speed, ovvero garantire la fruizione di una grande quantità di contenuti in tempi rapidi. Ovviamente un sistema distribuito scalabile deve garantire all'utenza tempi di risposta ottimali.

1.2 DESIGN PATTERNS

Elenchiamo di seguito i Design Pattern utilizzati per realizzare l'applicazione. In molti casi si è deciso di non attenersi all'implementazione canonica ma piuttosto di seguire la filosofia dietro il design pattern scelto.

- **COMMAND** Per permettere all'utente la navigazione delle pagine già visitate tramite Navigator.java, una struttura dati basata su due stack che conservano oggetti Page.java, wrapper che memorizza tipo e id del contenuto visitato.
- **Observer** Per notificare l'utente di alcuni eventi, quali l'aggiunta di un commento ad un post, tramite un pop up Javascript che può essere attivato da qualunque altro script passando come parametro il messaggio di notifica.

1.3 INTERFACE DOCUMENTATION GUIDELINES

1.3.1 NAMING CONVENTIONS

Forniamo delle linee guida per la stesura del codice durante la fase di implementazione così da poter essere consistenti e renderne più semplice la lettura sia agli sviluppatori interni al team che a terzi.

- CLASSES
  - I nomi delle classi devono rispettare la notazione UpperCamelCase
  - Le classi Entity/DTO devono chiamarsi NameEntity.
  - Le classi che interagiscono con il layer della persistenza devono chiamarsi NameDAO.
  - Le classi che modellano gli oggetti Control, ossia le nostre Servlet, devono essere denominate con un nome che ne esplicita la funzionalità offerta con una 'S' finale ad esempio Servlet con nome x sarà 'xS.java'.
- METHODS
  - I nomi dei metodi devono rispettare la notazione lowerCamelCase
- PACKAGE ORGANIZATION
  - Deve essere creato un package per ogni sottosistema
  - Le classi che realizzano il sottosistema, principalmente Servlet, DAO e Entity, devono essere contenute nello stesso package

1.4 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

- OCL: Object Constraint Language
- DAO: Data Access Object
- DTO: Data Transfer Object

1.5 REFERENCES

Nel corso del documento facciamo riferimento a artefatti precedenti quali il **Requirements Analysis Document** (RAD) e al **System Design Document** (SDD).

2 PACKAGES

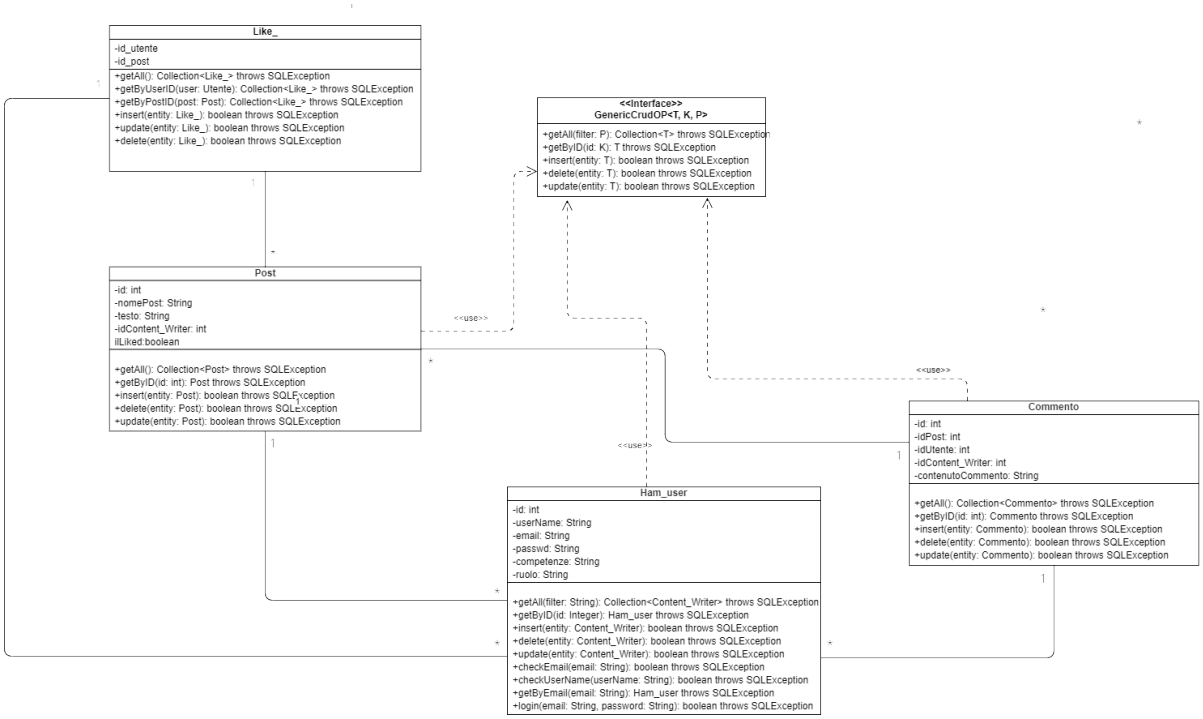
Con riferimento alla divisione in sottosistemi riportata nel SDD elenchiamo le componenti che compongono ciascun package.

- **Profile**
  - DAO
    - ProfileDAO.java
  - Servlet
    - CheckCWValidS
    - GetProfileS.java
    - ToUpdateProfileS.java
    - GetPostHistoryS.java
    - LoginS.java
    - LogoutS.java
    - SignUpS.java
    - UpdateProfileS.java
    - DeleteProfileS.java
  - Entity
    - ProfileEntity.java
    - UtenteEntity.java
    - ContentWriterEntity.java
    - SupervisorEntity.java
- **Navigation**
  - Servlet
    - NextPageS.java
    - prevPageS.java
  - Page.java
  - Navigator.java
- **Post**
  - DAO
    - getPostsHistoryS
    - PostDAO.java
  - Servlet
    - AddPostS.java
    - DeletePostS.java
    - PostS.java
    - LikePostS.java
    - UnlikePostS.java
  - Entity
    - PostEntity.java
- **Commento**
  - DAO
    - CommentoDAO.java
  - Servlet
    - AddCommentS.java
    - RemoveCommentS.java
  - Entity
    - CommentoEntity.java

3 CLASS INTERFACES GLOSSARY

3.1 CLASS DIAGRAM TO EER DIAGRAM

Partendo dal Class Diagram mostrato in precedenza nel RAD e nel SDD, effettuiamo le dovute modifiche per poter mappare le classi su delle entità da rendere persistenti su database relazionale SQL.

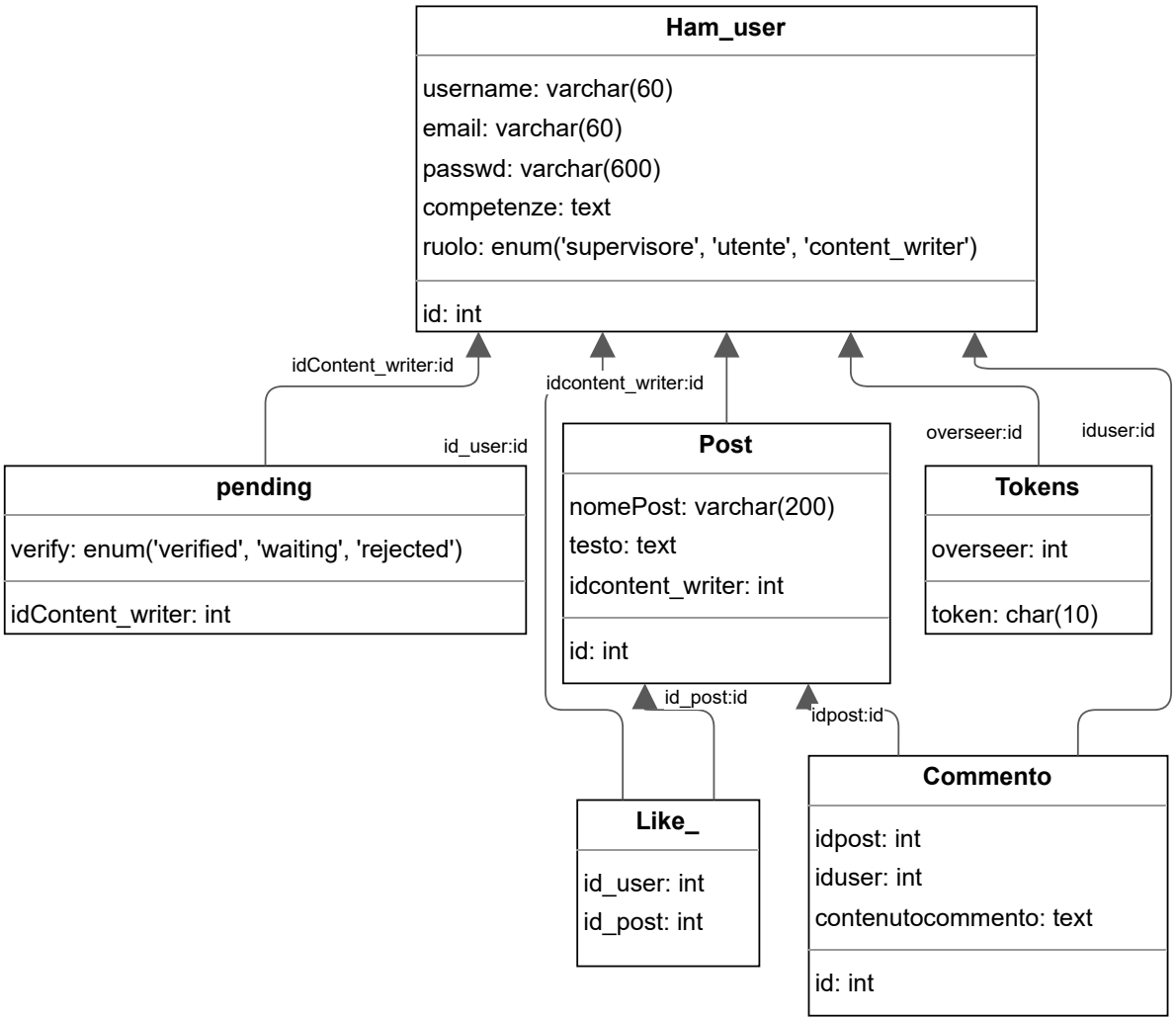


Rimuoviamo i seguenti attributi che NON andranno memorizzati su database bensì su file system. Il razionale dietro tale scelta è discusso in dettaglio nel SDD.

- profile_image della classe **Profile**
- post_image della classe **Post**

Modifichiamo inoltre la gerarchia che coinvolge le classi Profile, Artist e User includendo anche Overseer per facilitare la gestione delle credenziali e in seguito del login alla piattaforma.

Questo è il diagramma ER risultante generato dal tool integrato di IntelliJ:



3.2 INTERFACES SPECIFICATION

3.2.1 Entity

Nome Classe	PostEntity
Modulo	Post.Entity
Variabili di istanza	-id: int -nomePost: String -testo: String -idContentWriter: int -like: Boolean
Descrizione	Entità contenente le informazioni di un Post
Firme dei metodi	+ PostEntity(id: int, nomePost: String, testo: String, idCw: int) + getId(): int + getNomePost(): String + getTesto(): String + getIdContentWriter(): String + getLike(): Boolean + setId(id: int): void + setNomePost(nomePost: String): void + setTesto(testo: String): void + setLike(like: Boolean): void + setIdContentWriter(idCw: String): void
Pre e Post Condizioni	• context PostEntity::PostEntity(id, nomePost, testo, idCw) **pre:** true **post:** self.id = id **and** self.nomePost = nomePost **and** self.testo = testo **and** self.idContentWriter = idCw • context PostEntity::getId() **pre:** true **post:** id = self.id • context PostEntity::getNomePost() **pre:** true **post:** nomePost = self.nomePost • context PostEntity::getTesto() **pre:** true **post:** testo = self.testo • context PostEntity::getLike() **pre:** true **post**: like = self.like • context PostEntity::getIdContentWriter() **pre:** true **post:** idContentWriter = self.contentWriter • context PostEntity::setId(id) **pre:** true **post:** self.id = id • context PostEntity::setNomePost(nome) **pre:** true **post:** self.nomePost = nome • context PostEntity::setLike(like) **pre:** true **post:** self.like = like • context PostEntity::setTesto(testo) **pre:** true **post:** self.testo = testo • context PostEntity::setIdContentWriter(idCw) **pre:** true **post:** self.idContentWriter = idCw
Invarianti	

Nome Classe	UtenteEntity
Modulo	Profile.Entity
Variabili di Istanza	- id: int - username: String - password: String - email: String - competenze: String - role: Role
Descrizione	Entità che rappresenta un utente
Firma dei metodi	+ UtenteEntity(): UtenteEntity + getId(): int + getUsername(): String + getEmail(): String + getPassword(): String + getCompetenze(): String + getRole(): String + setId(id: int): void + setUsername(username: String): void + setEmail(email: String): void + setPassword(password: String): void + setCompetenze(comp: String): void + setRole(Role ruolo): void
Pre e Post condizioni	• context UtenteEntity::UtenteEntity() **pre:** true **post:** result = null • context UtenteEntity::getId() **pre:** true **post:** result = self.id • context UtenteEntity::getUsername() **pre:** true **post:** result = self.username

Pre e Post condizioni	• context UtenteEntity::getEmail() **pre:** true **post:** result = self.email • context UtenteEntity::getCompetenze() **pre:** true **post:** result = self.competenze • context UtenteEntity::getPassword() **pre:** true **post:** result = self.passowrd • context UtenteEntity::getRole() **pre:** true **post:** result = self.role • context UtenteEntity::setId(id: int) **pre:** true **post:** self.id = id • context UtenteEntity::setUsername(username: String) **pre:** true **post:** self.username = username • context UtenteEntity::setEmail(email: String) **pre:** true **post:** self.email = email • context UtenteEntity::setPassword(password: String) **pre:** true **post:** self.password = password • context UtenteEntity::setCompetenze(competenze: String) **pre:** true **post:** self.competenze = competenze • context UtenteEntity::setRole(Role: Role) **pre:** true **post:** self.role = role
Invarianti	

Nome Classe	Page
Modulo	Navigation.Entity
Variabili di Istanza	- id: int - type: Type
Descrizione	Entità che rappresenta una pagina visitata dall'utente
Firma dei metodi	+ Page(id: int, type: Type): Page + getId(): int + getType(): Type
Pre e Post condizioni	• context Page::Page(id: int, type: Type) **pre:** true **post:** self.id = id **and** self.type = type **and** result = self • context Page::getId() **pre:** true **post:** result = self.id • context Page::getType() **pre:** true **post:** result = self.type
Invarianti	

Nome Classe	Navigator
Modulo	Navigation.Entity
Variabili di Istanza	- prev: Stack<Page> - current: Page - next: Stack<Page>
Descrizione	Entità che rappresenta una pagina visitata dall'utente
Firma dei metodi	+ Navigator(): Navigator + getCurrent(): Page + setCurrent(page: Page): void + save(page: Page): void + prev(): Page + next(): Page + hashPrev(): boolean + hasNext(): boolean
Pre e Post condizioni	• context Navigator::Navigator **pre:** true **post:** self.prev = null **and** self.current = null **and** result = self • context Navigator::getCurrent() **pre:** true **post:** result = self.current • context Navigator::setCurrent(page) **pre:** true **post:** self.current = page • context Navigator::save(page) **pre:** true **post:** self.prev→top() = self.current **and** self.next.size = 0

Pre e Post condizioni	• context Navigator::prev() **pre:** true **post:** self.next→top() = self.@pre.current **and** self.current = self.prev→pop() **and** result = self.current • context Navigator::next() **pre:** true **post:** self.next→top() = self.@pre.current **and** self.current = self.next→pop() **and** result = self.current • context Navigator::hasPrev() **pre:** true **post:** **if** self.prev.size() > 0 **then** result = true **else** result = false **endif** • context Navigator::hasNext() **pre:** true **post:** **if** self.next.size() > 0 **then** result = true **else** result = false **endif**
Invarianti	

3.2.2 DAO

Consideriamo il database come un insieme di set/collections con i quali i DAO interagiscono:

- User
- ContentWriter
- Supervisor
- Post
- Commento

La maggioranza dei metodi dei DAO restituiscono un booleano che funge da acknowledgement per comunicare agli oggetti Control, le Servlet, l'esito dell'operazione.

Nome Classe	ProfileDAO
Modulo	Profile.DAO
Variabili di Istanza	- ds: DataSource
Descrizione	Oggetto DAO che si occupa di gestire le operazioni di persistenza degli utenti base
Firma dei metodi	+ **ProfileDAO**(datasource: DataSource): ProfileDAO + **getAll(String: role)**: Collection⟨*UtenteEntity*⟩ + **getById**(utenteId: int): UtenteEntity + **checkEmail**(email: String): Boolean + **checkContentWriterProfile**(id: int): Boolean + **checkContentWriterProfile**(email: String): Boolean + **checkUsername**(username: String): Boolean + **checkToken**(token: String): Boolean + **login**(email: String, password: String): UtenteEntity + **insert**(Utente: UtenteEntity): Boolean + **update**(Utente: UtenteEntity): Boolean + **delete**(idUtente: int): Boolean

Pre e Post condizioni	• context UtenteDAO::ProfileDAO(datasource) **pre:** true **post:** self.ds = datasource • context ProfileDAO::getAll() **pre:** true **post:** result = utente • context ProfileDAO::getById(userId: int) **pre:** true **post:** result = user→select(u| u.id=userId) • context ProfileDAO::checkEmail(email: String) **pre:** true **post:** result = user→select(u| u.email=email) • context ProfileDAO::checkContentWriterProfile(id: int) **pre:** true **post:** result = false**if** user→select(u| u→exists(p|p.id = u.id) **else** false • context ProfileDAO::checkContentWriterProfile(email: String) **pre:** true **post:** result = false**if** user→select(u| u→exists(p|p.id = u.id) **else** false • context ProfileDAO::checkUsername(username: String) **pre:** true **post:** result = user→select(u| u.username=username) • context ProfileDAO::checkToken(token: String) **pre:** true **post:** result = profile→select(s| token→exists(t|t.token = token **and** t.superId = null)) • context ProfileDAO::login(email: String, password: String) **pre:** true **post:** result = user→select(u| u.email=email **and** u.password = password)

Pre e Post condizioni	• context ProfileDAO::insert(user: utenteEntity) **pre:** true **post:** utente→exists(u | u= user) • context ProfileDAO::update(user: utenteEntity) **pre:** true **post:** utente→exists(u | u.id = user.id **and** u=user) • context ProfileDAO::remove(id: int) **pre:** true **post:** **not** utente→exists(u| u.id = id)
Invarianti	

Nome Classe	PostDAO
Modulo	Post.DAO
Variabili di Istanza	- ds: DataSource
Descrizione	Oggetto DAO che si occupa di gestire le operazioni di persistenza dei Post
Firma dei metodi	+ **PostDAO**(datasource: DataSource): PostDAO + **getAll()**: Collection⟨*PostEntity*⟩ + **getAllByContentWriter**(contentWriterId): Collection⟨*PostEntity*⟩ + **getById**(postId: int): PostEntity + **like**(postId: int, userId: int): Boolean + **unlike**(postId: int, userId: int): Boolean + **getLike**(postId: int, userId: int): Boolean + **insert**(post: PostEntity): Boolean + **update**(post: PostEntity): Boolean + **delete**(post: PostEntity): Boolean

Pre e Post condizioni	• context PostDAO::PostDAO(datasource) **pre:** true **post:** self.ds = datasource • context PostDAO::getAll() **pre:** true **post:** result = post • context PostDAO::getAllByContentWriter(contentWriterId: int) **pre:** true **post:** result = post→select(p| p.idContentWriter = contentWriterId) • context PostDAO::getById(postId: int) **pre:** true **post:** result = post→select(p| p.id=postId) • context PostDAO::like(postId: int, userID: int) **pre:** true **post:** post→exists(p| like→exists(l|l.postId = postId **and** l.userId =userID) • context PostDAO::unlike(postId: int, userID: int) **pre:** true **post:** **not** post→exists(p| like→exists(l|l.postId = postId **and** l.userId =userID) • context PostDAO::unlike(postId: int, userID: int) **pre:** true **post:** result = post→select(p| like→select(l|l.postId = postId **and** l.userId =userID) • context PostDAO::insert(post: PostEntity) **pre:** true **post:** post→exists(p | p= post) • context PostDAO::update(post: PostEntity) **pre:** true **post:** post→exists(p | p.id = post.id **and** p. • context PostDAO::remove(id: int) **pre:** true **post:** **not** post→exists(p| p.id = id)
Invarianti	

Nome Classe	CommentoDAO
Modulo	Commento.DAO
Variabili di Istanza	- ds: DataSource
Descrizione	Oggetto DAO che si occupa di gestire le operazioni di persistenza dei commenti
Firma dei metodi	+ **CommentoDAO**(datasource: DataSource): CommentoDAO + **getAll()**: Collection⟨*CommentoEntity*⟩ + **getById**(commentoId: int): CommentoEntity + **insert**(commento: CommentoEntity): Boolean + **update**(commento: CommentoEntity): Boolean + **delete**(commento: CommentoEntity): Boolean
Pre e Post condizioni	• context CommentoDAO::CommentoDAO(datasource) **pre:** true **post:** self.ds = datasource • context CommentoDAO::getAll() **pre:** true **post:** result = commento • context CommentoDAO::getById(commentoId: int) **pre:** true **post:** result = commento→select(c| c.id=commentoId) • context CommentoDAO::insert(commento: CommentoEntity) **pre:** true **post:** commento→exists(c | c= commento) • context CommentoDAO::update(commento: CommentoEntity) **pre:** true **post:** commento→exists(c | c.id = commento.id **and**) • context CommentoDAO::remove(id: int) **pre:** true **post:** **not** commento→exists(c| c.id = id)
Invarianti	

3.2.3 CONTROL

il metodo **init()** di ogni Servlet inizializza i DAO singleton necessari allo svolgimento dell'operazione recuperandoli dal ServletContext. La logica di business è contenuta interamente nel metodo **doPost(request: HttpServletRequest, response: HttpServletResponse).**

Fatta questa premessa, per una maggiore chiarezza e semplicità, riportiamo per ogni oggetto Control solo i DAO utilizzati e le condizioni del metodo doPost(. . .). In particolare, facciamo riferimento ai principali metodi dei DAO invocati e alle informazioni salvate nell'oggetto session.
Indichiamo con bean: TipoBean = request-¿get() le varie operazioni di recupero di parametri per comporre il bean.

Nome Classe	LikePostS
Modulo	Post.Servlet
Variabili di Istanza	- postDAO: PostDAO
Descrizione	Servlet che permette a un utente di mettere like ad un post
Definizioni	• context LikePostS **def** session: HttpSession = request→getSession() **def** userId: int = session→getAttribute("Profile") **def** postId: int = request→getParameter("idPOost")
Pre e Post condizioni	• context LikePostS::doPost(...) **pre:** idPost⟨⟩ null **post:** PostDAO.like(userId, postId)→getLike→exists(p|p.getId()
Invarianti	

Nome Classe	UnlikePostS
Modulo	Post.Servlet
Variabili di Istanza	- postDAO: PostDAO
Descrizione	Servlet che permette a un utente di togliere like ad un post
Definizioni	• context UnlikePostS **def** session: HttpSession = request→getSession() **def** userId: int = session→getAttribute("Profile") **def** postId: int = request→getParameter("idPost")
Pre e Post condizioni	• context UnlikePostS::doPost(...) **pre:** idPost⟨⟩ null **post:** PostDAO.unlike(userId, postId)→getLike→forAll(u|u.getId() ⟨⟩ id)
Invarianti	

Nome Classe	LoginS
Modulo	Profile.Servlet
Variabili di Istanza	- profileDAO: ProfileDAO
Descrizione	Servlet che permette a un utente di effettuare il login
Definizioni	• context LoginS **def** session: HttpSession = request→getSession() **def** email: String = session→getParameter("email") **def** password: String = request→getParameter("password")
Pre e Post condizioni	• context LoginS::doPost(...) **pre:** email⟨⟩ null **and** password ⟨⟩ null **post: let** profile: UtenteEntity = ProfileDAO→get(id) session→setAttribute("Profile") = profile
Invarianti	

Nome Classe	LogoutS
Modulo	Profile.Servlet
Variabili di Istanza	
Descrizione	Servlet che permette a un utente di effettuare il logout
Definizioni	• context LikeS **def** session: HttpSession = request→getSession()
Pre e Post condizioni	• context LikeS::doPost(...) **pre:** true **post:** session→invalidate()
Invarianti	

Nome Classe	SignUpS
Modulo	Profile.Servlet
Variabili di Istanza	- profileDAO: ProfileDAO
Descrizione	Servlet che permette a un utente di effettuare la sua registrazione
Definizioni	• context SignUPS **def** session: HttpSession = request→getSession() **def** email: String = request.getParameter(”email”) **def** password: String = request.getParameter(”password”) **def** role: String = request.getParameter(”role”) **def** competenze = request.getParameter(”competenze”)
Pre e Post condizioni	• context SignUpS::doPost(...) **pre:** email⟨⟩ null **and** password ⟨⟩ null **and** username ⟨⟩ null **post:** **if** role = ”user” **then** profileDAO→insert(user) **else** profileDAO→insert(contentWriter)
Invarianti	

Nome Classe	DeleteProfileS
Modulo	Profile.Servlet
Variabili di Istanza	- profileDAO: ProfileDAO
Descrizione	Servlet che permette a un utente di eliminare il proprio profilo
Definizioni	• context DeleteProfileS **def** session: HttpSession = request→getSession() **def** profile: UtenteEntity = request.getAttribute("profile")
Pre e Post condizioni	• context DeleteProfileS::doPost(...) **pre:** profile⟨⟩ null **post** profileDAO→delete(user)
Invarianti	

Nome Classe	AddPostS
Modulo	Post.Servlet
Variabili di Istanza	- postDAO: PostDAO
Descrizione	Servlet che permette ad un Content Writer di creare un nuovo post
Definizioni	• context AddPostS **def** session: HttpSession = request→getSession() **def** post: PostEntity = request.getAttribute("newPost") **def** role: Role = session→getAttribute("profile")
Pre e Post condizioni	• context AddPostS::doPost(...) **pre:** role = ContentWriter **post** postDao→add(post)
Invarianti	

Nome Classe	NextPageS
Modulo	Navigation.Servlet
Variabili di Istanza	
Descrizione	Servlet che permette all'utente di tornare a una pagina visitata in precedenza
Definizioni	• context NextPage **def** session: HttpSession = request→getSession() **def** navigator: Navigator = request.getAttribute("Navigator")
Pre e Post condizioni	• context NextPage::doPost(...) **pre:** true **post:** **let** page: Page = navigator→next() result = page
Invarianti	

Nome Classe	PrevPageS
Modulo	Navigation.Servlet
Variabili di Istanza	
Descrizione	Servlet che permette all'utente di tornare a una pagina visitata in precedenza
Definizioni	• context PrevPage **def** session: HttpSession = request→getSession() **def** navigator: Navigator = request.getAttribute("Navigator")
Pre e Post condizioni	• context PrevPageS::doPost(...) **pre:** true **post:** **let** page: Page = navigator→prev()
Invarianti	

Nome Classe	AddCommentS
Modulo	Commento.Servlet
Variabili di Istanza	- commentoDAO: CommentoDAO
Descrizione	Servlet che permette all'utente di aggiungere un commento ad un post
Definizioni	• context AddCommentoS **def** session: HttpSession = request→getSession() **def** idUtente: int = session→getAttribute("Profile") **def** idPost: int = request→getParameter("idPost") **def** commento: String = request→getParameter("commento")
Pre e Post condizioni	• context AddCommentoS::doPost(...) **pre:** idPost <> null **and** commento <> null **post:** commentoDAO.insert(commento) →getCommento →exists (c|c.getid())
Invarianti	

Nome Classe	RemoveCommentS
Modulo	Commento.Servlet
Variabili di Istanza	- commentoDAO: CommentoDAO
Descrizione	Servlet che permette all'utente di rimuovere un commento ad un post
Definizioni	• context RemoveCommentoS **def** session: HttpSession = request→getSession() **def** idUtente: int = session→getAttribute("Profile") **def** idPost: int = request→getParameter("idPost")
Pre e Post condizioni	• context RemoveCommentoS::doPost(...) **pre:** idPost <> null **post:** commentoDAO.delete(id) →getCommento →exists (c|c.getid()<>id)
Invarianti	

Nome Classe	DeletePostS
Modulo	Post.Servlet
Variabili di Istanza	- postDAO: PostDAO
Descrizione	Servlet che permette ad un content Writer ed un supervisore di eliminare un post
Definizioni	• context DeletePostS **def** session: HttpSession = request→getSession() **def** utente: UtenteEntity = session→getAttribute("Profile") **def** idPost: int = request→getParameter("idPost")
Pre e Post condizioni	• context DeletePostS::doPost(...) **pre:** idPost <> null **and** utente.role <> Role.utente **post:** postDAO.delete(id) →getPost →exists (p|p.getid()<>id)
Invarianti	

Nome Classe	UpdateProfileS
Modulo	Profile.Servlet
Variabili di Istanza	- profileDAO: ProfileDAO
Descrizione	Servlet che permette ad un utente di aggiornare il proprio profilo
Definizioni	• context UpdateProfileS **def** session: HttpSession = request→getSession() **def** utente: UtenteEntity = session→getAttribute("Profile") **def** updateCredential: UtenteEntity = request→getParameter("updateProfile")
Pre e Post condizioni	• context UpdateProfileS::doPost(...) **pre:** utente.id <> null **post:** profileDAO.update(updateCredential)
Invarianti	

Nome Classe	getPostsHistoryS
Modulo	Post.Servlet
Variabili di Istanza	- postDAO: PostDAO
Descrizione	Servlet che permette di ottenere lo storico dei post di un Content Writer
Definizioni	• context getPostsHistoryS **def** session: HttpSession = request→getSession() **def** utente: UtenteEntity = session→getAttribute("Profile")
Pre e Post condizioni	• context UpdateProfileS::doPost(...) **pre:** utente.id <> null **and** utente.role = Role.contentWriter **post:** result = postDAO.getAllByContentWriter(utente.id)
Invarianti	

Nome Classe	CheckCWValidS
Modulo	Post.Servlet
Variabili di Istanza	- profileDAO: ProfileDAO
Descrizione	Servlet che permette di verificare la validicità di un account Content Writer
Definizioni	• context CheckCWValidS **def** email: String = request→getParameter("email")
Pre e Post condizioni	• context CheckCWValidS::doPost(...) **pre:** email <> null **post:** result = profileDAO.checkContentWriter(email)
Invarianti	