Object Design Document
Sorting Hat

Riferimento	
Versione	1.0
Data	14/02/2022
Destinatario	Docente Ingegneria del Software 2021/22
Presentato da	Ascione Josef, Di Gregorio Emanuele, Di Sarno Davide, Sacco Matteo
Approvato da	

Data	Versione	Descrizione	Autori
17/12/2021	0.1	Prima stesura	Ascione J., Di Sarno D.
08/01/2022	0.2	Revisione capitolo 1	Di Sarno D.
08/02/2022	0.3	Modifiche punti 1,1.1,1.2,1.4,1.5	Di Sarno D., Di Gregorio E.
12/02/2022	0.4	Modifiche punti 2,4 Stesura punto 2,5	Di Sarno D., Sacco M.
13/02/2022	0.5	Stesura punti 2.1, 2.2, 2.3,3,3.1	Sacco M., Di Gregorio E.
14/02/2022	1.0	Revisione finale	Di Sarno D.

SOMMARIO

1.Introduzione 2
1.1 Object Design Goals 2
1.2 Object Trade-offs 2
1.3 Linee Guida per la Documentazione delle Interfacce 3
1.4. Definizioni, acronimi e abbreviazioni 3
1.5 Riferimenti 3
2.Packages 4
2.1 Interface 4
2.2 Application logic 5
2.3 Storage 6
3. Class interfaces 6
3.1 Classi nel Package Storage/ 7
3.2 Classi nel package Application Logic 13
4.Class Diagram 16
5.Design Pattern 17
5.1 DAO Pattern 17
5.2 SINGLETON Pattern 17
5.3 Façade pattern 17
6.Glossario 17

1.INTRODUZIONE

SortingHat si propone di venire in aiuto nell'interazione tra studenti universitari e studenti neodiplomati sullo scambio di informazioni sui percorsi accademici e sul materiale didattico.

In questa prima sezione del documento, verranno descritti i design goals, i trade-offs e le linee guida per la fase di implementazione, riguardanti la nomenclatura, la documentazione e le convenzioni sui formati.

1.1 OBJECT DESIGN GOALS

Riusabilità, Robustezza, Incapsulamento, ECC

1.2 OBJECT TRADE-OFFS

Spazio vs. Tempi di risposta= Fin quando possibile il sistema deve garantire una maggiore velocita di risposta (per l'elaborazione e la visualizzazione all'utente) a discapito della memoria utilizzata.

Tempi di risposta vs. Affidabilità=Essendo il sistema basato sull'interazione con l'utente, deve preferire una maggiore affidabilità nelle sue operazioni piuttosto che ai tempi di risposta per poter garantire dati di input e output validi.

Tempi di risposta vs. Sicurezza=Per garantire la sicurezza del sito ed evitare operazioni e accessi illegali può essere necessario sacrificare la velocità delle elaborazioni.

Usabilità vs Estensibilità: Se per rendere il sistema estensibile dovremo sacrificare del tempo per quanto riguarda l'usabilità allora potrà essere necessario dare la priorità alla facilità di utilizzo

Sicurezza vs Estensibilità: Se per garantire la sicurezza del sistema, in merito ad accessi a pagine web e l'esecuzione di operazioni soggette a limitazioni relative ai ruoli degli utenti, dovremmo ridurre la facilità di aggiunta di nuove funzionalità per impedire operazioni illegali.

1.3 LINEE GUIDA PER LA DOCUMENTAZIONE DELLE INTERFACCE

Gli sviluppatori dovrebbero rispettare queste linee guida durante la progettazione.

La convenzione java utilizzata fa riferimento al **Sun Java Coding Conventions** redatto da Sun nel 2009.

Gli sviluppatori terranno traccia dei seguenti link:

- **HTML**: https://www.w3schools.com/html/html5_syntax.asp
- **Java**: https://www.w3schools.com/java/default.asp
- **CSS:** https://www.w3schools.com/css/default.asp

1.4. DEFINIZIONI, ACRONIMI E ABBREVIAZIONI

- **JSP**: JavaServer Pages è una tecnologia di programmazione web in Java per lo sviluppo della logica di presentazione di applicazioni web, fornendo contenuti dinamici in formato HTML o XML
- **Off-the-shelf**: Strumenti e servizi esterni usati dal sistema
- **HTML**: un linguaggio di programmazione per lo sviluppo web
- **CSS**: è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML
- **Javascript**: JavaScript è un linguaggio di programmazione orientato agli oggetti e agli eventi
- **TomCat**: un server web open source
- **JQuery**: libreria Javascript per applicazioni web
- **Javadoc**: sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile

1.5 RIFERIMENTI

- Dispense del docente
- Libro: Object-Oriented Software Engineering (Using UML, Patterns and Java) Third Edition Autori: Bernd Bruegge & Allen H. Dutoit

2.PACKAGES

In questa sezione viene mostrata la suddivisione in package del sistema.

- **.idea**
- **lib**, contiene tutte le librerie utilizzate
- **src**, contiene tutti i file sorgente
  - **main**
    - **java**, contiene tutte le classi relativi alle componenti *controller* e *model*
    - **webapp**
      - **css**, contiene tutti i fogli di stile del sistema
      - **js**, contiene tutti i file javascript del sistema
      - **WEB-INF**, contiene tutte le views del sistema
      - **homepage.jsp**, landing page del sistema
  - **test**, contiene tutto il necessario per il testing
    - **java**, contiene tutte le classi java necessarie per il testing
- **target**, contiene tutti i file prodotti dal sistema di build di Maven

Package SortingHat

In questa sezione viene mostrata la struttura del package del Sistema, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e evidenzia la struttura di directory standard definita da Maven.

2.1 INTERFACE

- Package **AutenticazioneGUI**
  - *adminDashboard* - mostra la dashboard principale dell'utente
  - *categorie* – mostra le categorie del forum
  - *discussione* – mostra tutte le discussioni di una determinata categoria
  - *login* – mostra un form per permettere il login alla piattaforma
  - *post* – mostra le info di una determinata discussione
  - *profilo* – mostra il profilo utente
  - *questionarioUniversitario* – mostra il form dedicato agli universitari
  - *questionarioUtente* – mostra il form dedicato agli utenti
  - *registrazione* – mostra un form per permettere la registrazione alla piattaforma
  - *utenteProfiloForm* – mostra il form per modificare il profilo
- Package **RispostaGUI**
  - *RispostaCreate* - mostra il form per creare una risposta ad una discussione
  - *RispostaFormCreate* - form per creare una risposta ad una discussione
  - *RispostaFormUpdate* - form per modificare una risposta ad una discussione
  - *RispostaUpdate* - mostra il form per modificare una risposta ad una discussione
- Package **CategoriaGUI**
  - *CategoriaCreate* - mostra il form per creare una categoria
  - *CategoriaFormCreate* - form per creare una categoria
  - *CategoriaUpdate* - form per modificare una categoria
  - *CategoriaUpdateForm* - mostra il form per modificare una categoria
- Package **DiscussioneGUI**

    - *DiscussioneCreate* - mostra il form admin per creare una discussione
    - *DiscussioneCreateUtente* - mostra il form per creare una discussione
    - *DiscussioneFormCreate* – form per creare una discussione
    - *DiscussioneUpdate* – mostra il form per modificare una discussione
    - *DiscussionUpdateForm* – form per modificare una discussione
- Package **AdminGUI**
    - *categoriaList* – contiene la lista delle categorie
    - *categoriaTable* – tabella contenente le categorie
    - *discussioneList* – contiene la lista delle discussioni
    - *discussioneTable* – tabella contenente le discussioni
    - *rispostaList* – contiene la lista delle risposte
    - *rispostaTable* – tabella contenente le risposte
    - *utenteList* – contiene la lista degli utenti
    - *utenteTable* – tabella contenente gli utenti
    - *UtenteUpdate* – permette la modifica dell'utente
    - *UtenteUpdateForm* – contiene il form per modificare l'utente
- Package **documenti**
    - *aboutUs* – mostra le informazioni riguardo la piattaforma
    - *statistiche* – mostra le statistiche della piattaforma
- Package **errors**
    - *internalError* – viene visualizzato in caso di errore 500
    - *notFound* – viene visualizzato in caso di errore 404
    - *unauthorized* – viene visualizzato in caso di errore 401
- Package **partials**
    - adminBar – menu dell'admin mostrato nella dashboard
    - alert – alert utilizzato in caso di errore/successo
    - background – sfondo di ogni pagina della piattaforma
    - footer – footer di ogni pagina della piattaforma
    - head – head di ogni pagina della piattaforma
    - headS – head di ogni pagina della piattaforma
    - menuDesktop – menu mostrato ad utenti desktop
    - menuMobile – menu mostrato ad utenti mobile
    - paginator - paginatore

2.2 APPLICATION LOGIC

- Package **PageServlet**
    - Servlet che si occupa di svolgere e gestire tutte le operazioni di transizione dalla HomePage alle altre sezioni
- Package **RispostaServlet**
    - Servlet che si occupa di svolgere e gestire tutte le operazioni riguardanti le Risposte
- Package **CategoriaServlet**
    - Servlet che si occupa di svolgere e gestire tutte le operazioni riguardanti le Categorie
- Package **DiscussioneServlet**
    - Servlet che si occupa di svolgere e gestire tutte le operazioni riguardanti le Discussioni
- Package **UtenteServlet**
    - Servlet che si occupa di svolgere e gestire tutte le operazioni riguardanti gli Admin e gli Utenti

2.3 STORAGE

- Package **Risposta**
  - *Risposta*: Classe che modella una Risposta
  - *RispostaFormMapper*: Tiene conto unicamente degli attributi utili alle operazioni di SQL
  - *RispostaValidator*: Classe che effettua un controllo sui vincoli di creazione dei valori degli attributi
  - *SqlRispostaDAO*: Classe che contiene l'implementazione dei metodi CRUD per oggetti Risposta
- Package **Categoria**
  - *Categoria*: Classe che modella una Categoria
  - *CategoriaFormMapper*: Tiene conto unicamente degli attributi utili alle operazioni di SQL
  - *CategoriaValidator*: Classe che effettua un controllo sui vincoli di creazione dei valori degli attributi
  - *SqlCategoriaDAO*: Classe che contiene l'implementazione dei metodi CRUD per oggetti Categoria
- Package **Discussione**
  - *Discussione*: Classe che modella una Discussione
  - *DiscussioneFormMapper*: Tiene conto unicamente degli attributi utili alle operazioni di SQL
  - *DiscussioneValidator*: Classe che effettua un controllo sui vincoli di creazione dei valori degli attributi
  - *SqlDiscussioneDAO*: Classe che contiene l'implementazione dei metodi CRUD per oggetti Discussione
- Package **Utente**
  - *Utente*: Classe che modella una Utente
  - *UtenteFormMapper*: Tiene conto unicamente degli attributi utili alle operazioni di SQL
  - *UtenteValidator*: Classe che effettua un controllo sui vincoli di creazione dei valori degli attributi
  - *SqlUtenteDAO*: Classe che contiene l'implementazione dei metodi CRUD per oggetti Utente
- Package **Utente**
  - *Utente*: Classe che modella una Utente
  - *UtenteFormMapper*: Tiene conto unicamente degli attributi utili alle operazioni di SQL
  - *UtenteValidator*: Classe che effettua un controllo sui vincoli di creazione dei valori degli attributi
  - *SqlUtenteDAO*: Classe che contiene l'implementazione dei metodi CRUD per oggetti Utente

3. CLASS INTERFACES

In questo paragrafo vengono elencate le interfacce delle classi previste dal Sistema presenti all'interno del package "Storage" e "Application Logic".

Per rendere il documento meglio leggibile non vengono elencate le classi Mapper, Bean ed alcune classi verranno omesse.
L'intera documentazione è reperibile qui: https://ja3unisa.github.io/SortingHat/

3.1 CLASSI NEL PACKAGE STORAGE/

Nome Classe	CATEGORIADAO
Descrizione	Questa classe che modella gli oggetti categoria
Metodi	+countAll() : int +fetchCategories(paginatore : Paginator) : List<Categoria> +deleteCategoria(id : String) : boolean +fetchCategoriesById(id : int) : Optional<Categoria> +updateCategoria(categoriaAgg : Categoria) : boolean +createCategoria(categoria : Categoria) : boolean +fetchCategoriesAll() : List<Categoria>
Invariante di classe	/

Nome Metodo	+countAll() : int
Descrizione	Questo metodo permette di contare il numero di categorie
Pre-condizione	/
Post-condizione	/

Nome Metodo	+fetchCategories(paginatore : Paginator) : List<Categoria>
Descrizione	Questo metodo permette di ottenere tutte le categorie
Pre-condizione	**Context**: List<Categoria> :: fetchCategories(paginatore) **Pre**: paginatore!=null
Post-condizione	/

Nome Metodo	+deleteCategoria(id : String) : boolean
Descrizione	Questo metodo permette di eliminare una categoria
Pre-condizione	**Context**: boolean :: deleteCategoria(id) **Pre**: id!=null
Post-condizione	/

Nome Metodo	+fetchCategoriesById(id : int) : Optional<Categoria>
Descrizione	Questo metodo permette di ottenere una categoria in base all'id
Pre-condizione	**Context**: Optional<Categoria> :: fetchCategoriesById(id) **Pre**: id!=null
Post-condizione	/

Nome Metodo	+updateCategoria(categoriaAgg : Categoria) : boolean
Descrizione	Questo metodo permette di aggiornare una categoria
Pre-condizione	**Context**: boolean :: updateCategoria(categoriaAgg) **Pre**: categoriaAgg!=null
Post-condizione	/

Nome Metodo	+createCategoria(categoria : Categoria) : boolean
Descrizione	Questo metodo permette di creare una categoria
Pre-condizione	**Context**: boolean :: createCategoria(categoria) **Pre**: categoria!=null
Post-condizione	/

Nome Metodo	+fetchCategoriesAll() : List<Categoria>
Descrizione	Questo metodo permette di ottenere tutte le categorie
Pre-condizione	/
Post-condizione	/

Nome Classe	DISCUSSIONEDAO
Descrizione	Questa classe modella gli oggetti discususione
Metodi	+countAll() : int +fetchDiscussioni(paginatore : Paginator) : List<Discussione> +fetchDiscussioniById(id : int) : Optional<Discussione> +deleteDiscussione(id: String) : boolean +updateDiscussione(discussioneAgg : Discussione) : boolean +createDiscussione(discussione : Discussione) : boolean +fetchDiscussioniByCategoria(categoria : Categoria, paginatorore : Paginator) : List<Discussione> +fetchDiscussioniAll() : List<Discussione> +createDiscussioneUtente(Discussione discussione) : int
Invariante di classe	/

Nome Metodo	+countAll() : int
Descrizione	Questo metodo permette di contare tutte le discussioni
Pre-condizione	/
Post-condizione	/

Nome Metodo	+fetchDiscussioni(paginatore : Paginator) : List<Discussione>
Descrizione	Questo metodo permette di ottenere tutte le discussioni grazie ad un paginatore
Pre-condizione	**Context**: List<Discussione> :: fetchDiscussioni(paginatore) **Pre**: paginatore!=null
Post-condizione	/

Nome Metodo	+fetchDiscussioniById(id : int) : Optional<Discussione>
Descrizione	Questo metodo permette di ottenere una discussione in base al suo id
Pre-condizione	**Context**: Optional<Discussione> :: fetchDiscussioniById(id) **Pre**: id!=null
Post-condizione	/

Nome Metodo	+deleteDiscussione(id: String) : boolean
Descrizione	Questo metodo permette eliminare una discussione
Pre-condizione	**Context**: boolean :: deleteDiscussione(id) **Pre**: id!=null
Post-condizione	/

Nome Metodo	+updateDiscussione(discussioneAgg : Discussione) : boolean
Descrizione	Questo metodo permette di aggiornare una discussione
Pre-condizione	**Context**: boolean :: updateDiscussione(discussioneAgg) **Pre**: discussioneAgg!=null
Post-condizione	/

Nome Metodo	+createDiscussione(discussione : Discussione) : boolean
Descrizione	Questo metodo permette di creare una discussione
Pre-condizione	**Context**: boolean :: createDiscussione(discussione) **Pre**: discussione!=null
Post-condizione	/

Nome Metodo	+fetchDiscussioniByCategoria(categoria : Categoria, paginatorore : Paginator) : List<Discussione>
Descrizione	Questo metodo permette di ottenere discussioni in base alla categoria ed il paginatore
Pre-condizione	**Context**: List<Discussione> :: fetchDiscussioniByCategoria(categoria, paginatore) **Pre**: categoria!=null&&paginatore!=null

Post-condizione	/

Nome Metodo	**+fetchDiscussioniAll() : List<Discussione>**
Descrizione	Questo metodo permette di ottenere tutte le discussioni
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+createDiscussioneUtente(Discussione discussione) : int**
Descrizione	Questo metodo permette di creare una discussione da parte dell'utente
Pre-condizione	**Context**: int :: createDiscussioneUtente (discussione) **Pre**: discussione!=null
Post-condizione	/

Nome Classe	**RISPOSTA DAO**
Descrizione	Questa è una classe che modella gli oggetti Risposta
Metodi	+createRisposta(Risposta risposta) +updateRisposta(Risposta risposta) +deleteRisposta(Risposta risposta) +countAll() +List<Risposta> fetchRispostaByIdDiscussione(iDDiscussione,Paginator paginator) + List<Risposta>fetchRisposta(Paginator paginatore) +Optional<Risposta> fetchRisposta(idUpd)
Invariante di classe	/

Nome Metodo	**+createRisposta(Risposta risposta):boolean**
Descrizione	Questo metodo consente di inserire una nuova risposta nel Database

Pre-Condizione	**Context:** boolean :: createRisposta(categoria) **Pre:** Risposta !=null
Post-Condizione	/

Nome Metodo	**+deleteRisposta(id : String) : boolean**
Descrizione	Questo metodo permette di eliminare una Risposta
Pre-condizione	**Context**: boolean :: deleteRisposta(id) **Pre**: id!=null
Post-condizione	/

Nome Metodo	**+updateRisposte(Risposta :risposta) : boolean**
Descrizione	Questo metodo permette di aggiornare una risposta
Pre-condizione	**Context**: boolean :: updateRisposta(categoriaAgg) **Pre**: risposta!=null
Post-condizione	/

Nome Metodo	**+countAll() : int**
Descrizione	Questo metodo permette di contare il numero di risposte
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+fetchRispostaByIdDiscussione(idDiscussione : int,paginator Paginatore) : List<Risposta>**
Descrizione	Questo metodo permette di ottenere una categoria in base all'id
Pre-condizione	**Context**: List<Risposta>:: fetchRispostaByIdDiscussione (id) **Pre**: id!=null
Post-condizione	/

Nome Metodo	**+fetchRisposta(paginatore : Paginator) : List<Risposta>**
Descrizione	Questo metodo permette di ottenere tutte le risposte
Pre-condizione	**Context**: List<Risposta> :: fetchRisposta(paginatore) **Pre**: paginatore!=null

Post-condizione	/

Nome Metodo	**+fetchRisposte(idUpd : int) : Optional <Risposta>**
Descrizione	Questo metodo permette di ottenere tutte le Risposte con un dato Id
Pre-condizione	**Context**: List<Risposta> :: fetchRisposte(inUpd) **Pre**: inUpd!=null
Post-condizione	/

Nome Classe	**UTENTE DAO**
Descrizione	Questa è una classe che modella gli oggetti Utente
Metodi	+createUtente(Utente utente) +updateUtente(Utente utente) +deleteUtente(Utente utente) +countAll() +List<Utente> fetchAccaunts(Paginator paginatore) + Optional<Risposta>findUtente(String email,String password) + Optional<Risposta>findUtentebyID(int idUpd) +updateUser(Utente utente)
Invariante di classe	/

Nome Metodo	**+ createUtente(Utente utente):boolean**
Descrizione	Questo metodo consente di inserire un nuovo Utente nel Database
Pre-Condizione	**Context:** boolean :: createUtente(utente) **Pre:** utente !=null
Post-Condizione	/

3.2 CLASSI NEL PACKAGE APPLICATION LOGIC

Nome Metodo	**+updateUtente(Utente utente):boolean**
Descrizione	Questo metodo permette di aggiornare un Admin
Pre-condizione	**Context**: boolean :: updateUtente(utente) **Pre**: utente!=null

Post-condizione	/

Nome Metodo	**+deleteUtente(id : int) : boolean**
Descrizione	Questo metodo permette di eliminare un Utente
Pre-condizione	**Context**: boolean :: deleteUtente(id) **Pre**: id!=null
Post-condizione	/

Nome Metodo	**+countAllUtente() : int**
Descrizione	Questo metodo permette di contare il numero di utenti
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+fetchAccounts (paginatore : Paginator) : List<Utente>**
Descrizione	Questo metodo permette di ottenere tutti gli Utenti
Pre-condizione	**Context**: List<Utenti> :: fetchAccounts (paginatore) **Pre**: paginatore!=null
Post-condizione	/

Nome Metodo	**+updateUser(Utente utente):boolean**
Descrizione	Questo metodo permette di aggiornare un Utente
Pre-condizione	**Context**: boolean :: updateUser(utente) **Pre**: utente!=null
Post-condizione	/

Nome Metodo	**+findUtente(String email,String password):Option<Utente>**
Descrizione	Questo metodo permette di trovare un utente in base all'email e la password
Pre-condizione	**Context**: Option<Utente>:: findUtente(email,password) **Pre**: email!=null &&password!=null
Post-condizione	/

Nome Metodo	**+findUtenteById(int idCl):Option<Utente>**
Descrizione	Questo metodo permette di trovare un utente in base all'ID

Pre-condizione	**Context**: Option<Utente>:: findUtente(idCl) **Pre**: idCl !=null
Post-condizione	/

Nome Classe	**ErrorHandler**
Descrizione	Questa classe permette di gestire gli errori
Metodi	+authenticated(session : HttpSession) : void +authorize(session : HttpSession) : void +internalError() : void +notFound() : void +notAllowed() : void
Invariante di classe	/

Nome Metodo	**+authenticated(session : HttpSession) : void**
Descrizione	Controlla se è presente una sessione e quindi se l'utente è loggato
Pre-condizione	**Context**: void :: authenticated(session) **Pre**: session!=null
Post-condizione	/

Nome Metodo	**+ authorize(session : HttpSession) : void**
Descrizione	Controlla se è l'utente in sessione è admin
Pre-condizione	**Context**: void :: authenticated(session) **Pre**: session!=null
Post-condizione	/

Nome Metodo	**+ internalError() : void**
Descrizione	Gestisce errori di sistema
Pre-condizione	/
Post-condizione	/

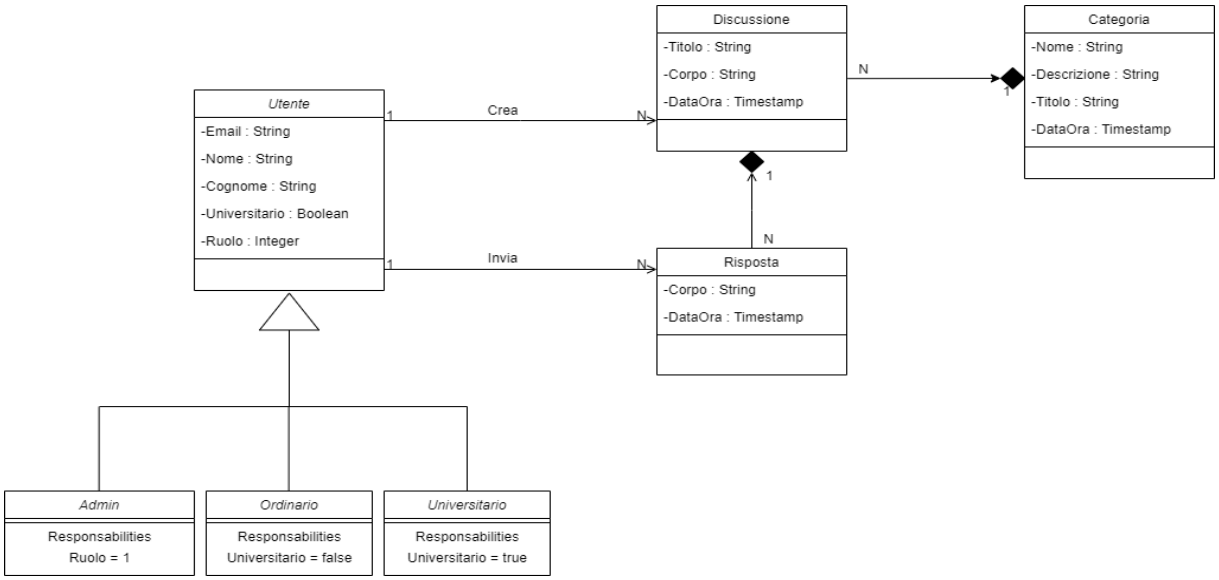
Nome Metodo	**+ notFound() : void**
Descrizione	Gestisce errori 404
Pre-condizione	/
Post-condizione	/

Nome Metodo	+ notAllowed() : void
Descrizione	Gestisce errori di autorizzazione
Pre-condizione	/
Post-condizione	/

Nome Classe	FormMapper
Descrizione	Gestisce la formattazione dei Bean prima di un'operazione
Metodi	+map(request : HttpServletRequest, update : Boolean) : T
Invariante di classe	/

Nome Metodo	+map(request : HttpServletRequest, update : Boolean) : T
Descrizione	Permette di mappare un Bean prima di un'operazione
Pre-condizione	**Context**: T :: map(request, update) **Pre**: request!=null&&update!=null
Post-condizione	/

4.CLASS DIAGRAM



5.DESIGN PATTERN

In questa sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo di SortingHat.

5.1 DAO PATTERN

Un DAO (Data Access Object) è un pattern architetturale che permette di utilizzare un'API astratta per poter interagire con alcuni tipi di database. Attraverso l'API è possibile effettuare operazioni CRUD(create,read,update e delete) e operazioni specifiche come, ad esempio, le query per selezionare gli elementi relativi ai dati persistenti.

Essendo SortingHat una web appliaction risulta chiaro la necessità di utilizzare numero query per interagire con un database per poter inserire, visualizzare, modificare e rimuovere entità quali le discussioni, le domande, le categorie e gli utenti. Inoltre, attraverso il database, vengono ottenute le credenziali 'ruolo' e 'universitario' necessarie per poter separare gli utenti ordinari, universitari e gli admin, suddividendo le azioni permesse. Nel sistema, infine, la sezione 'contribuisci', l'area del sito web relativa alla compilazione dei moduli form per migliorare l'accuratezza dell'algoritmo di apprendimento, termina con il salvataggio delle risposte in una tabella nel database da cui poi il modulo IA esterno attinge per poter consigliare il dipartimento più adatto nella sezione 'consiglio'. Pertanto, abbiamo deciso di utilizzare il design pattern DAO.

5.2 SINGLETON PATTERN

Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga strutturata una sola istanza, fornendo un punto di accesso globale a tale istanza.

SortingHat necessita, per i motivi citati nel punto 5.1, di potersi connettere al database e svolgere le operazioni relative ai dati persistenti. La connessione avviene utilizzando una classe denominata 'ConPool',la quale è un Singleton poiché deve essere istanziata una sola volta.

5.3 FAÇADE PATTERN

Questo design pattern permette di utilizzare un'interfaccia semplificata per poter definire e accedere ai servizi dei sottosistemi, realizzando, inoltre, un'architettura chiusa.

La scelta di utilizzare queto pattern ha permesso di semplificare l'utilizzo della logica di business in diverse aree di implementazione:

- Le servlet sono collegate ad un'interfaccia ErrorHandler per l'autenticazione (verifica che l'utente sia loggato), l'autorizzazione (verifica che la sessione sia di un admin) e per la gestione di errori interni e tentativi di accessi illegali.
- Tutte le classi in cui vengono effettate query al database implementano metodi specificati da delle interfacceDAO che definisce anche le eccezioni relative ai metodi.
- L'interfaccia FormMapper(per l'utilizzo di oggetti specifici per le operazioni CRUD) permette di implementare le classi relative agli oggetti specificando anche se si richiede un'operazione di modifica oppure no.

6.GLOSSARIO

- **SortingHat**: Nome del software proposto
- API: Application programming interface, un insieme di definizioni e protocolli per la creazione e l'integrazione di software applicativi