UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Laurea in Informatica

Progetto Ingegneria del Software

OBJECT DESIGN DOCUMENT

VERSIONE 1.4

BookPad

Partecipanti

Nome	Matricola
Marica D'Alfonso	0512106258
Marianna Farina	0512109126

Sommario

1. Introduzione

Il documento di Requirement Analysis e il System Design Document hanno fornito una panoramica generale del sistema e degli obiettivi, senza entrare nei dettagli implementativi. Il presente documento ha lo scopo di creare un modello che possa integrare in modo coerente e preciso tutte le funzionalità identificate in precedenza.

1.1 Object design trade-off

- **Interfaccia vs Usabilità**
  La progettazione dell'interfaccia grafica si è concentrata sull'obiettivo di rendere il sistema il più facile e intuitivo possibile per gli utenti. A tal fine, sono sati utilizzati colori e icone riconoscibili, e sono state semplificate le interazioni utente-sistema. Grazie a queste scelte, si è ottenuta un'interfaccia grafica chiara, concisa e facilmente utilizzabile da un vasto pubblico di utenti.

- **Sicurezza vs Efficienza**
  La sicurezza è un aspetto molto importante, ma il limitato tempo a disposizione per lo sviluppo ha impedito la progettazione di un sistema totalmente sicuro. Quindi ci si è concentrati unicamente sulla protezione delle password degli utenti, che vengono criptate per impedire accessi non autorizzati.

- **Prestazioni vs Costi**
  Nel progetto di creazione del sito, si è deciso di non utilizzare librerie e template a pagamento, a causa di un budget insufficiente. Si è, quindi, privilegiato l'utilizzo di soluzioni open source e di componenti hardware propri.

1.2 Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno seguire le seguenti convenzioni per la scrittura del codice:

1.2.1 Naming Convention

La nomenclatura deve rispettare le seguenti caratteristiche:

- descrittivi;
- pronunciabili;
- di lunghezza medio-corta;
- di uso comune;
- utilizzando solo caratteri consentiti (a-z, A-Z, 0-9).

1.2.2 Variabili

- I nomi delle variabili devono cominciare con una lettera minuscola.
- Se il nome della variabile è costituito da più parole, si può utilizzare la Lower Camel Notation (es: nomeVariabile). In alternativa è possibile usare l'underscore (_) come separatore.
- In ogni riga dovrà esserci un'unica variabile dichiarata.

1.2.3 Metodi

- I nomi dei metodi dovranno iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola, secondo la Lower Camel Notation.
- Il nome del metodo sarà costituito da un verbo che ne identifica l'azione seguito da un sostantivo.
- Il nome dei metodi accessori e modificatori seguirà, rispettivamente, i pattern getNomeVariabile e setNomeVariabile.

1.2.4 Classi

- I nomi delle classi e dovranno utilizzare la Pascal Case Notation; quindi, dovranno iniziare con la lettera maiuscola, così come le parole successive all'interno del nome (es: ProdottoMagazzino).
- I nomi delle classi dovranno corrispondere alle informazioni e le funzioni fornite da quest'ultime.
- Le classi saranno strutturate prevedendo rispettivamente:
    1. dichiarazione della classe pubblica;
    2. dichiarazione di costanti;
    3. dichiarazioni di variabili di classe;
    4. dichiarazioni di variabili di istanza;
    5. costruttore;
    6. metodi.

1.2.5 Packages

- I nomi dei packages dovranno essere scritti in minuscolo concatenando insieme diversi sostantivi o sigle, separate dal carattere ".".

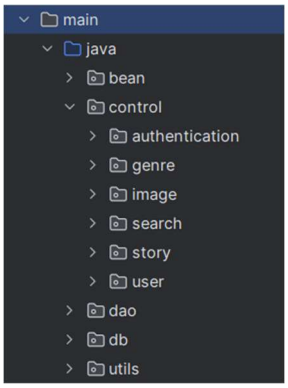
1.3 Definizione, acronimi e abbreviazioni

RAD sta per Requirement Analysis Document.
SDD sta per System Design Document.
ODD sta per Object Design Document.
JSP sta per Java Servlet Page.

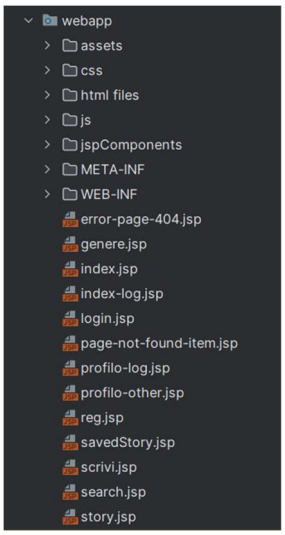
1.4 Riferimenti

Riferimento al RAD ed SDD.

2. Packages



Il back-end è implementato in pacchetti presenti nella directory **/java/.**

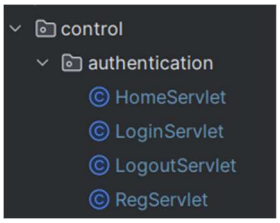


Il front-end è implementato in pacchetti presenti nella directory **/webapp/.**

2.1 Back-end Package

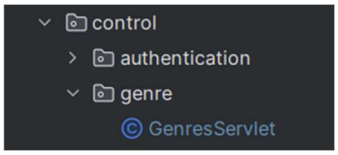
2.1.1 control.authentication

Classe	Descrizione
HomeServlet	Servlet che permette di visualizzare la homepage di un utente guest o la homepage di un utente loggato, in base a se l'utente è loggato o meno.
LoginServlet	Servlet che permette all'utente di accedere con le credenziali alla piattaforma.
LogoutServlet	Servlet che permette il logout dell'utente loggato dalla piattaforma.
RegServlet	Servlet che permette la registrazione dell'utente alla piattaforma.



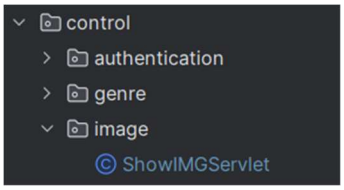
2.1.2 control.genre

Classe	Descrizione
GenresServlet	Servlet che permette di caricare i generi della piattaforma e di visualizzare la jsp riguardante i generi.



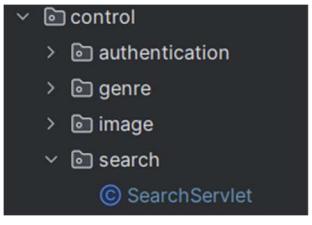
2.1.3 control.image

Classe	Descrizione
ShowIMGServlet	Servlet che permette di ottenere gli avatar degli utenti o le copertine delle storie.



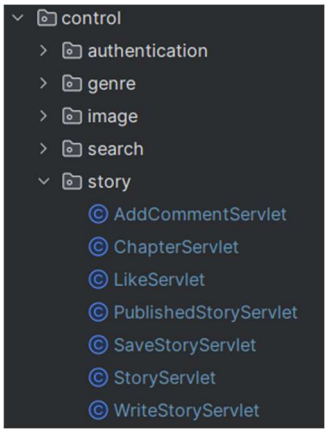
2.1.4 control.search

Classe	Descrizione
SearchServlet	Servlet che permette di ricercare le storie attraverso tag o titolo e di ricercare gli utenti tramite l'username. Inoltre, permette di visualizzare la pagina jsp relativa alla ricerca.



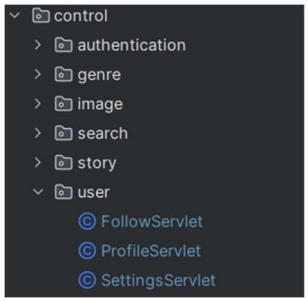
2.1.5 control.story

Classe	Descrizione
AddCommentServlet	Servlet che permette di aggiungere commenti alle storie della piattaforma.
ChapterServlet	Servlet che permette all'utente di visualizzare il capitolo che desidera leggere.
LikeServlet	Servlet che permette all'utente di aggiungere o rimuovere il like dalle storie.
PublishedStoryServlet	Servlet che permette di visualizzare la jsp di esito di pubblicazione della storia.
SaveStoryServlet	Servlet che permette all'utente di salvare le storie.
StoryServlet	Servlet che permette di visualizzare la pagina jsp di una specifica storia.
WriteStoryServlet	Servlet che permette all'utente di pubblicare le storie e di visualizzare la pagina jsp che permette all'utente di scrivere la storia.



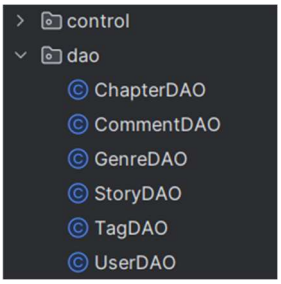
2.1.6 control.user

Classe	Descrizione
FollowServlet	Servlet che permette di aggiungere o rimuovere il follow ad un altro utente.
ProfileServlet	Servlet che permette di visualizzare la jsp del proprio profilo e del profilo di un altro utente.
SettingsServlet	Servlet che permette all'utente di cambiare la propria email, password, biografia o avatar.



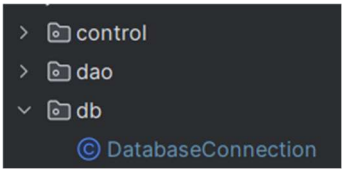
2.1.7 dao

Classe	Descrizione
ChapterDAO	Classe che implementa le operazioni con il database riguardanti i capitoli delle storie.
CommentDAO	Classe che implementa le operazioni con il database riguardanti i commenti delle storie.
GenreDAO	Classe che implementa le operazioni con il database riguardanti i generi della piattaforma.
StoryDAO	Classe che implementa le operazioni con il database riguardanti le storie della piattaforma.
TagDAO	Classe che implementa le operazioni con il database riguardanti i tag della piattaforma.
UserDAO	Classe che implementa le operazioni con il database riguardanti gli utenti della piattaforma.



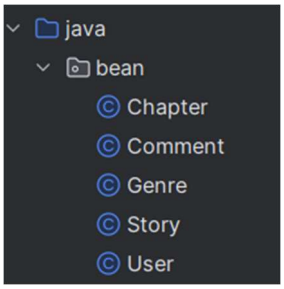
2.1.8 db

Classe	Descrizione
DatabaseConnection	Classe che permette l'accesso al database attraverso i driver JDBC.



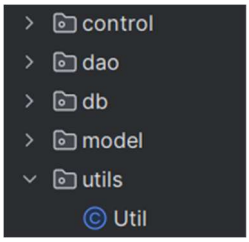
2.1.9 bean

Classe	Descrizione
Chapter	Classe che rappresenta le informazioni relative ad un capitolo di una storia.
Comment	Classe che rappresenta le informazioni relative ad un commento di una storia.
Genre	Classe che rappresenta le informazioni relative ad un genere della piattaforma.
Story	Classe che rappresenta le informazioni relative ad una storia della piattaforma.
User	Classe che rappresenta le informazioni relative ad un utente iscritto alla piattaforma.



2.1.10 utils

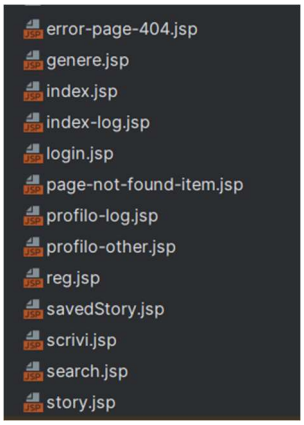
Classe	Descrizione
Util	Classe contenente metodi d'ausilio.



2.2 Front-end Package

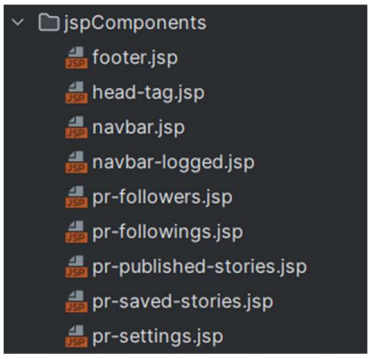
2.2.1 /webapp/

JSP	Descrizione
error-page-404.jsp	View che viene mostrata in caso di errore 404.
genere.jsp	View che mostra le informazioni di un genere e la lista delle storie appartenenti a quel genere.
index.jsp	View che mostra la homepage per un utente guest.
index-log.jsp	View che mostra la homepage per un utente loggato alla piattaforma.
login.jsp	View che mostra il form per accedere alla piattaforma.
page-not-found-item.jsp	View che mostra messaggio d'errore in caso un utente o una storia non è stato trovato.
profilo-log.jsp	View che mostra il profilo dell'utente loggato.
profilo-other.jsp	View che mostra il profilo di un altro utente registrato alla piattaforma.
reg.jsp	View che mostra il form per registrarsi alla piattaforma.
savedStory.jsp	View che mostra l'esito positivo di salvataggio di una storia.
scrivi.jsp	View che mostra il form per pubblicare una storia.
story.jsp	View che mostra le informazioni di una storia pubblicata sulla piattaforma.
search.jsp	View che mostra i risultati di ricerca.



2.2.2 /webapp/jspComponents/

JSP	Descrizione
footer.jsp	View che mostra il footer.
head-tag.jsp	View che contiene gli headers.
navbar.jsp	View che mostra la navbar per un utente guest.
navbar-logged.jsp	View che mostra la navbar per un utente loggato.
pr-followers.jsp	View che mostra la sezione contente la lista di followers di un utente.
pr-followings.jsp	View che mostra la sezione contente la lista di followings di un utente.
pr-published-stories.jsp	View che mostra la sezione contente le storie pubblicate da un utente.
pr-saved-stories.jsp	View che mostra la sezione contente le storie salvate da un utente.
pr-settings.jsp	View che mostra la sezione delle impostazioni.



3. Class Interface

3.1 ChapterDAO

Nome classe	**ChapterDAO**
	`context ChapterDAO::`**getChapter**(id:int, num:int): Chapter
precondizione	`context ChapterDAO::`**addChapter**(title:String, content:String, n_chap:int, story:Story): Chapter **pre**: title!=null, content!=null `context ChapterDAO::`**getAllChapters**(story:Story): List<Chapter> **pre**: story!=null
post-condizione	
invariante	

3.2 GenreDAO

Nome classe	**GenreDAO**
precondizione	`context GenreDAO::`**getByName**(name:String): Genre **pre**: name!=null
post-condizione	`context GenreDAO::`**getAllGenres**(): List<Genre> l **post**: l !=null
invariante	

3.3 CommentDAO

Nome classe	**CommentDAO**
	`context CommentDAO::`**updateComments**(id_story:int, num:int): void
precondizione	`context CommentDAO::`**addComment**(u:User, id_story:int, comment:String, date:LocalDate): Comment **pre**: u!=null, comment!=null, date!=”0-00-0000” `context CommentDAO::`**getAllComments**(story:Story): List<Comment> **pre**: story!=null
post-condizione	
invariante	

3.4 StoryDAO

Nome classe	**StoryDAO**
	`context StoryDAO::`**getById**(id:int): Story `context StoryDAO::`**getCover**(id:int): InputStream `context StoryDAO::`**updateLikes**(id_story:int, num:int): void `context StoryDAO::`**updateSavings**(id_story:int, num:int): void

precondizione	`context StoryDAO::`**getByGenre**(genre: Genre): List<Story> **pre**: genre!=null `context StoryDAO::`**getByTagSearch**(tag: String): List<Story> **pre**: tag!=null `context StoryDAO::`**getByTitleSearch**(title: String): List<Story> **pre**: title!=null; `context StoryDAO::`**addStory**(title:String, plot:String, author:User, genre:Genre, cover:InputStream): Story **pre**: title!=null, plot!=null, author!=null, genre!=null, cover !=null `context StoryDAO::`**getByTitle**(title:String): Story **pre**: title!=null `context StoryDAO::`**isLike**(u:User, id_story:int): boolean **pre**: u!=null `context StoryDAO::`**addLike**(u:User, id_story:int): void **pre**: u!=null `context StoryDAO::`**removeLike**(u:User, id_story:int): void **pre**: u!=null `context StoryDAO::`**getPublishedStories**(u:User): List<Story> **pre**: u!=null `context StoryDAO::`**getSavedStories**(u:User): List<Story> **pre**: u!=null `context StoryDAO::`**getFollowingsStories**(u:User): List<Story> **pre**: u!=null `context StoryDAO::`**saveStory**(u:User, id_story:int): void **pre**: u!=null `context StoryDAO::`**isSaved**(u:User, id_story:int): boolean **pre**: u!=null `context StoryDAO::`**removeSavedStory**(u:User, id_story:int): void **pre**: u!=null
post-condizione	
invariante	

3.5 TagDAO

Nome classe	**TagDAO**
precondizione	`context TagDAO::`**addTagToStory**(story: Story, tag: String): void **pre**: story!=null, tag!=null `context TagDAO::`**addTag**(tag: String): void **pre**: tag!=null

	`context TagDAO::`**checkTag**(tag:String): boolean **pre**: tag!=null `context TagDAO::`**getAllTags**(story:Story): List<String> **pre**: story!=null
post-condizione	
invariante	

3.6 UserDAO

Nome classe	**UserDAO**
precondizione	`context UserDAO::`**login**(email:String, password:String): User **pre**: email!=null, password!=null `context UserDAO::`**register**(email:String, password:String, username:String): User **pre**: email!=null, password!=null, username!=null `context UserDAO::`**getByUsername**(username:String): User **pre**: username!=null `context UserDAO::`**checkEmail**(email:String): boolean **pre**: email!=null `context UserDAO::`**checkUsername**(username:String): boolean **pre**: username!=null `context UserDAO::`**getAvatar**(username:String): InputStream **pre**: username!=null `context UserDAO::`**getFollowings**(u:User): List<User> **pre**: u!=null `context UserDAO::`**getFollowers**(u:User): List<User> **pre**: u!=null `context UserDAO::`**getByUsernameSearch**(u:String): List<User> **pre**: u!=null `context UserDAO::`**updateAvatar**(u:User, avatarStream:InputStream): void **pre**: u!=null, avatarStream!=null `context UserDAO::`**updateBio**(u:User, bio:String): void **pre**: u!=null, bio!=null `context UserDAO::`**updateEmail**(u:User, email:String): void **pre**: u!=null, email!=null `context UserDAO::`**updatePassword**(u:User, pass:String): void **pre**: u!=null, pass!=null `context UserDAO::`**isFollowing**(u:User, username:String): boolean **pre**: u!=null, username!=null

	`context UserDAO::`**unfollow**(u:User, u2:User): void **pre**: u!=null, u2!=null `context UserDAO::`**follow**(u:User, u2:User): void **pre**: u!=null, u2!=null
post-condizione	
invariante	