



ODD Object Design Document

iLike

| | |
|----------------------|---|
| Riferimento | |
| Versione | 5.6 |
| Data | 14/01/2023 |
| Destinatario | Prof. Carmine Gravino e tutor IS a.a. 2022/23 |
| Presentato da | Costante Luigina, Giorgione Francesco, Lo Conte Simona, Napolillo Marta |
| Approvato da | |
| | |



Revision History

| Data | Versione | Descrizione | Autori |
|------------|----------|---|---------------------|
| 22/12/2022 | 0.1 | Aggiunta <i>Object Trade-off</i> e package <i>Recensioni</i> | Luigina Costante |
| 23/12/2022 | 0.2 | Aggiunta introduzione e package <i>liste e contenuti</i> | Francesco Giorgione |
| 23/12/2022 | 0.3 | Modifiche package <i>recensioni</i> | Luigina Costante |
| 23/12/2022 | 0.4 | Aggiunta package <i>Profilo</i> e <i>Account</i> | Marta Napolillo |
| 23/12/2022 | 0.5 | Modifica package <i>lista</i> e <i>contenuti</i> | Francesco Giorgione |
| 24/12/2022 | 0.6 | Aggiunta introduzione della sezione <i>Packages</i> e package <i>segnalazioni</i> | Simona Lo Conte |
| 27/12/2022 | 0.7 | Revisione | Tutto il team |
| 27/12/2022 | 1.0 | Aggiunta class interfaces package <i>liste e contenuti</i> | Francesco Giorgione |
| 27/12/2022 | 1.1 | Aggiunta class interface package <i>recensioni</i> | Luigina Costante |
| 28/12/2022 | 1.2 | Aggiunta class interfaces package <i>profili e account</i> | Marta Napolillo |
| 28/12/2022 | 1.3 | Aggiunta class interfaces package <i>segnalazioni</i> | Simona Lo Conte |
| 29/12/2022 | 1.4 | Modifica class interfaces package <i>segnalazioni</i> | Simona Lo Conte |
| 29/12/2022 | 1.5 | Modifica package e class interfaces <i>liste e contenuti</i> | Francesco Giorgione |
| 29/12/2022 | 1.6 | Modifica package e class interface <i>recensioni</i> . | Luigina Costante |
| 30/12/2022 | 2.0 | Aggiunta sezione <i>design pattern</i> | Francesco Giorgione |
| 30/12/2022 | 2.1 | Modifica class interfaces package <i>segnalazioni</i> | Simona Lo Conte |
| 30/12/2022 | 2.2 | Modifica class interface <i>segnalazioni, contenuti</i> e | Francesco Giorgione |



Laurea Triennale in informatica - Università di Salerno
Corso di *Ingegneria del Software* – Prof. C. Gravino

| | | | |
|------------|------|--|---------------------|
| | | <i>recensioni; aggiunta sezione definizioni e abbreviazioni</i> | |
| 30/12/2022 | 2.3 | Modifica package <i>recensioni</i> | Luigina Costante |
| 31/12/2022 | 2.4 | Modifica package segnalazioni | Simona Lo Conte |
| 31/12/2022 | 2.5 | Modifica package <i>account</i> e <i>profili</i> , modifica della descrizione del Design Pattern Proxy | Marta Napolillo |
| 02/01/2023 | 2.6 | Revisione package | Francesco Giorgione |
| 02/01/2023 | 2.7 | Aggiunta class diagram | Luigina Costante |
| 03/01/2023 | 2.8 | Aggiunta package <i>utils</i> | Marta Napolillo |
| 03/01/2023 | 2.9 | Modifiche package | Simona Lo Conte |
| 03/01/2023 | 2.10 | Modifiche Class Diagram | Luigina Costante |
| 04/01/2023 | 2.11 | Modifiche packages <i>account (storage)</i> , <i>profili (storage)</i> e <i>recensioni</i> (aggiunta eccezione <i>InvalidTipoException</i>) | Luigina Costante |
| 04/01/2023 | 2.12 | Revisione | Marta Napolillo |
| 04/01/2023 | 2.13 | Modifiche package <i>liste</i> | Luigina Costante |
| 05/01/2023 | 2.14 | Modifica livello presentazione dei package | Francesco Giorgione |
| 05/01/2023 | 2.15 | Modifica class interface segnalazioni | Simona Lo Conte |
| 05/01/2023 | 2.16 | Modica package <i>account</i> | Marta Napolillo |
| 06/01/2023 | 2.17 | Modifica package <i>contenuti</i> | Francesco Giorgione |
| 07/01/2023 | 2.18 | Modifiche varie | Marta Napolillo |
| 07/01/2023 | 2.19 | Modifica package segnalazioni e modifiche class interfaces | Simona Lo Conte |
| 07/01/2023 | 3.1 | Modifica package <i>account</i> | Marta Napolillo |
| 07/01/2023 | 3.2 | Modifica object design goal | Francesco Giorgione |



Laurea Triennale in informatica - Università di Salerno
Corso di *Ingegneria del Software* – Prof. C. Gravino

| | | | |
|------------|-----|--|---------------------|
| 08/01/2023 | 3.3 | Modifica interfaccia e package <i>contenuti</i> | Francesco Giorgione |
| 08/01/2023 | 3.4 | Modifiche varie | Marta Napolillo |
| 08/01/2023 | 3.5 | Modifica package <i>liste</i> | Francesco Giorgione |
| 09/01/2023 | 3.6 | Modifica <i>ContenutiService</i> | Francesco Giorgione |
| 09/01/2023 | 3.7 | Modifica package <i>utils</i> e revisione | Francesco Giorgione |
| 09/01/2023 | 3.8 | Modifica <i>RecensioneService</i> | Francesco Giorgione |
| 09/01/2023 | 4.0 | Modifica sezione <i>package</i> | Francesco Giorgione |
| 10/01/2023 | 4.1 | Modifica package <i>account</i> | Francesco Giorgione |
| 10/01/2023 | 4.2 | Modifica <i>ContenutoService</i> | Francesco Giorgione |
| 11/01/2023 | 5.0 | Rimozione package <i>profili</i> in seguito a modifica della suddivisione in sottosistemi; modifica package <i>account</i> | Francesco Giorgione |
| 11/01/2023 | 5.1 | Modifica sezione <i>design pattern</i> + modifiche varie | Francesco Giorgione |
| 12/01/2023 | 5.2 | Modifica package <i>account</i> e <i>recensioni</i> | Francesco Giorgione |
| 12/01/2023 | 5.3 | Modifiche varie | Francesco Giorgione |
| 13/01/2023 | 5.4 | Modifica package <i>account</i> | Francesco Giorgione |
| 14/01/2023 | 5.5 | Modifiche varie class interfaces | Luigina Costante |
| 14/01/2023 | 5.6 | Modifiche varie class interfaces | Marta Napolillo |



Sommario

| | |
|---|----|
| Revision History | 2 |
| 1 Introduzione | 6 |
| 1.1 Object Trade-off | 6 |
| 1.2 Linee guida per la scrittura del codice | 7 |
| 1.3 Definizioni, acronimi, e abbreviazioni | 7 |
| 1.4 Riferimenti | 7 |
| 2 Package | 7 |
| Package liste | 10 |
| Package contenuti | 11 |
| Package recensioni | 12 |
| Package account | 13 |
| Package segnalazioni | 14 |
| Package utils | 14 |
| 3 Class Interfaces | 15 |
| Package <i>it.unisa.iLike.liste</i> | 15 |
| Package <i>it.unisa.iLike.contenuti</i> | 16 |
| Package <i>it.unisa.iLike.recensioni</i> | 18 |
| Package <i>it.unisa.iLike.account</i> | 20 |
| Package <i>it.unisa.iLike.segnalazioni</i> | 22 |
| 4 Class Diagram | 25 |
| 5 Design Patterns | 25 |
| 6 Glossario | 28 |



1 Introduzione

iLike è un'applicazione Android che consente ai suoi utenti di recensire libri, album musicali, film e serie TV. Considerata la costante crescita del numero di fruitori di tali contenuti, la recensione rappresenta una fondamentale forma di interazione tra le persone. A tale scopo, *iLike* si propone come luogo di incontro di persone che condividono gli stessi interessi e garantisce un'interconnessione – seppur indiretta – tra il pubblico e i content creator.

Nella sezione introduttiva del presente documento, si descrivono gli Object design goal, i trade-off e le linee guida riguardanti la fase di implementazione.

1.1 Object Trade-off

- **Memoria vs tempo di risposta:**

Si è disposti ad aumentare lo spazio di memoria a disposizione dell'applicazione per rispettare il massimo tempo di risposta indicato. Ad esempio, come specificato nel dizionario dei dati dell'SDD, nelle tabelle del database si mantengono alcuni attributi calcolabili.

- **Tempo di risposta vs connessione ad Internet:**

Il sistema dovrà in ogni caso rispettare il massimo tempo di risposta indicato. In caso di connessione assente/scadente, verrà visualizzato un messaggio di errore entro i tempi stabiliti.

- **Effort vs interfaccia utente:**

Al fine di rispettare i vincoli di tempo e risorse stabiliti, si è disposti a realizzare un'interfaccia utente estremamente basilare. Ad esempio, come già specificato nel RAD, la prima versione dell'applicazione funzionerà soltanto in *portrait* (orientamento verticale). Ci riserviamo di aggiungere il supporto al *landscape* (orientamento orizzontale) in successive versioni dell'applicazione.

- **Tempo di risposta vs manutenibilità:**

Al fine di garantire un'alta manutenibilità del software, si è disposti ad accettare un aumento dei tempi di risposta.



1.2 Linee guida per la scrittura del codice

La scrittura del codice Java dovrà attenersi alle linee guida definite da Oracle e riportate [in questa pagina](#).

1.3 Definizioni, acronimi, e abbreviazioni

- **Bean:** classe Java serializzabile che possiede un costruttore nullo e delle proprietà accessibili tramite metodi getter e setter. È utilizzata per incapsulare più oggetti in un singolo oggetto (il bean), in modo da evitare di trattarli come oggetti distinti.
- **DAO:** Data Access Object
- **DB:** Database
- **Design pattern:** template di soluzioni – che gli sviluppatori hanno raffinato nel tempo – utilizzabili per risolvere un insieme di problemi ricorrenti.
- **Package:** raggruppamento di classi e interfacce logicamente collegate.

1.4 Riferimenti

- Bern Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering – Using UML, Patterns and Java, 3rd edition
- Materiale fornito dal docente e dai tutor sulla piattaforma e-learning

2 Package

Questa sezione riguarda la suddivisione del sistema in package, sulla base delle scelte fatte riguardo l'architettura del sistema (*Three-tier*) e i sottosistemi nella fase di System Design.

La struttura generale dei package prevede la creazione di un package separato per ogni sottosistema e di un package *utils*, contenente metodi di utilità che saranno definiti in seguito, sulla base delle necessità, e saranno utilizzabili da più classi del sistema.

Inoltre, ogni package relativo ad un sottosistema contiene i seguenti package:



- *application* per la logica applicativa del sottosistema (contiene l'interfaccia dei servizi del sottosistema e la corrispondente implementazione concreta);
- *storage* per l'interazione con il DB e la gestione dei dati persistenti.

Si precisa che le componenti *presentation* sono tutte memorizzate nel package *main.res.layout*: si tratta di un vincolo imposto da Android che non consente di posizionare le componenti *presentation* nei corrispondenti package. La tabella seguente evidenzia a quali sottosistemi fanno riferimento le componenti *presentation* presenti in *main.res.layout*.

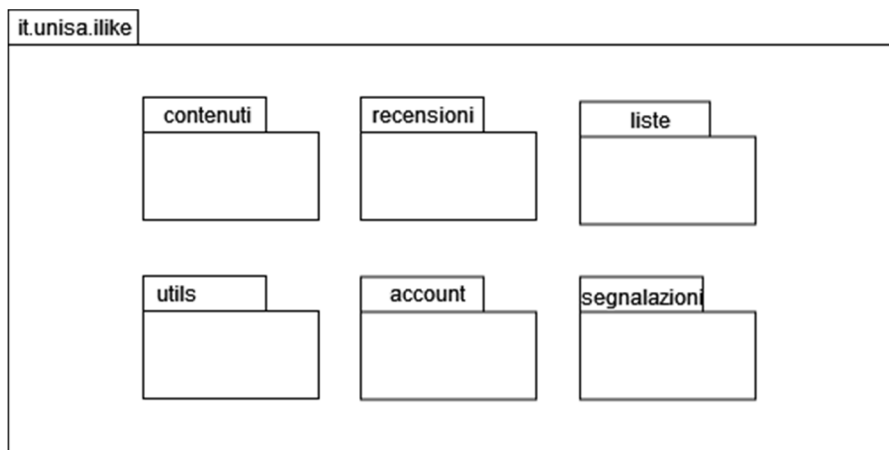
| Componente <i>presentation</i> | Sottosistema di riferimento |
|---|-----------------------------|
| activity_aggiunta_contenuto_lista.xml | Gestione liste |
| activity_aggiunta_segnalazione_recensione.xml | Gestione recensioni |
| activity_creazione_lista.xml | Gestione liste |
| activity_gestione_segnalazione.xml | Gestione segnalazione |
| activity_list_element_aggiunta_contenuto_lista.xml | Gestione liste |
| activity_list_element_ricerca_contenuto.xml | Gestione contenuti |
| activity_list_element_visualizzazione_dettagliata_contenuto.xml | Gestione contenuti |
| activity_list_element_visualizzazione_profilo_personale.xml | Gestione account |
| activity_list_element_visualizzazione_segnalazioni.xml | Gestione segnalazioni |
| activity_login.xml | Gestione account |
| activity_pubblicazione_recensione.xml | Gestione recensioni |
| activity_registrazione_iscritto.xml | Gestione account |
| activity_ricerca_contenuto.xml | Gestione contenuti |
| activity_visualizzazione_contenuti_lista_personale.xml | Gestione liste |
| activity_visualizzazione_dettagliata_contenuto.xml | Gestione contenuti |



| | |
|--|-----------------------|
| activity_visualizzazione_homepage.xml | Gestione contenuti |
| activity_visualizzazione_profilo_personale.xml | Gestione account |
| activity_visualizzazione_segnalazioni.xml | Gestione segnalazioni |

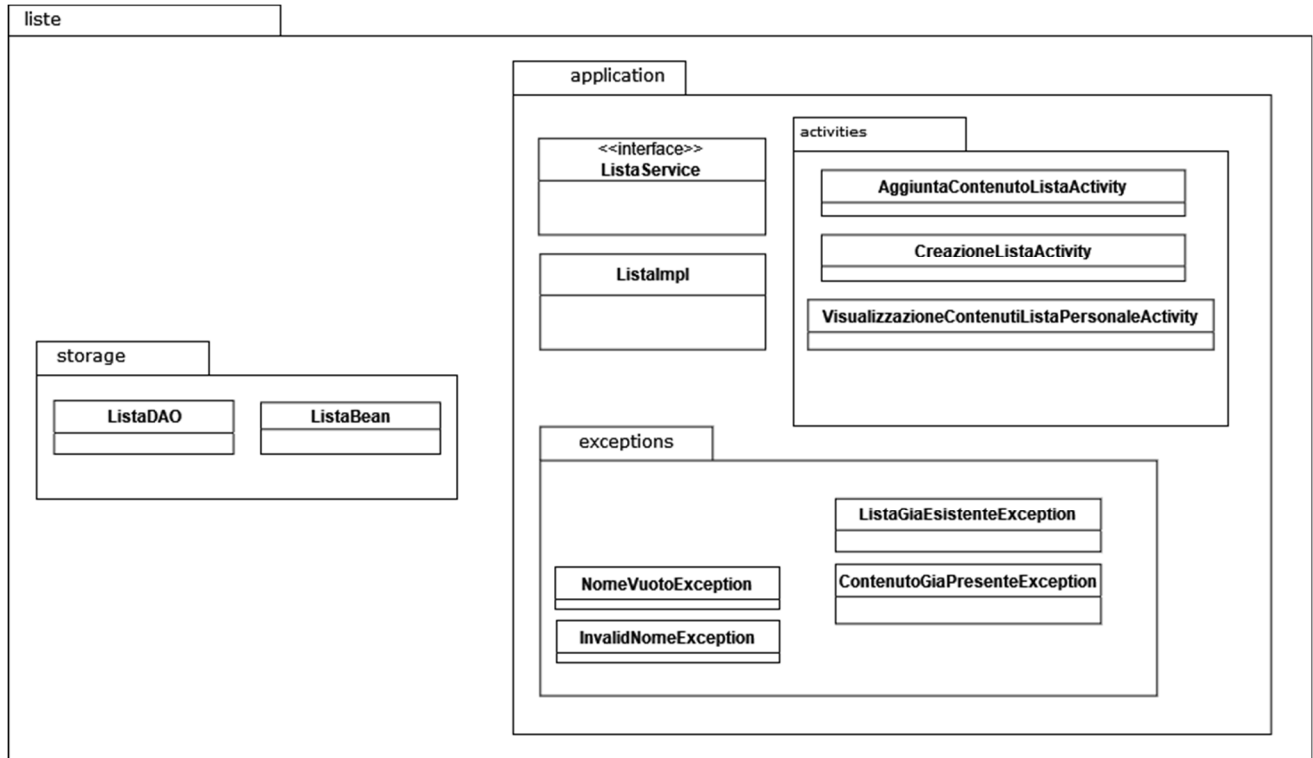
Si evidenzia inoltre che in ciascuno dei sottopackage *application* è presente un ulteriore package *activities*, in cui sono memorizzate

- le classi Java *Activity*: esse modellano gli oggetti *control* che gestiscono il flusso delle informazioni tra gli oggetti *boundary* e gli oggetti che forniscono i servizi (dettagli sul mapping oggetti control – classi Activity nella matrice di tracciabilità);
- le classi Java *Adapter*, utilizzate da alcune delle classi Activity per una migliore gestione del flusso delle informazioni (non essendo di fondamentale importanza, nella rappresentazione dei package vengono omesse).

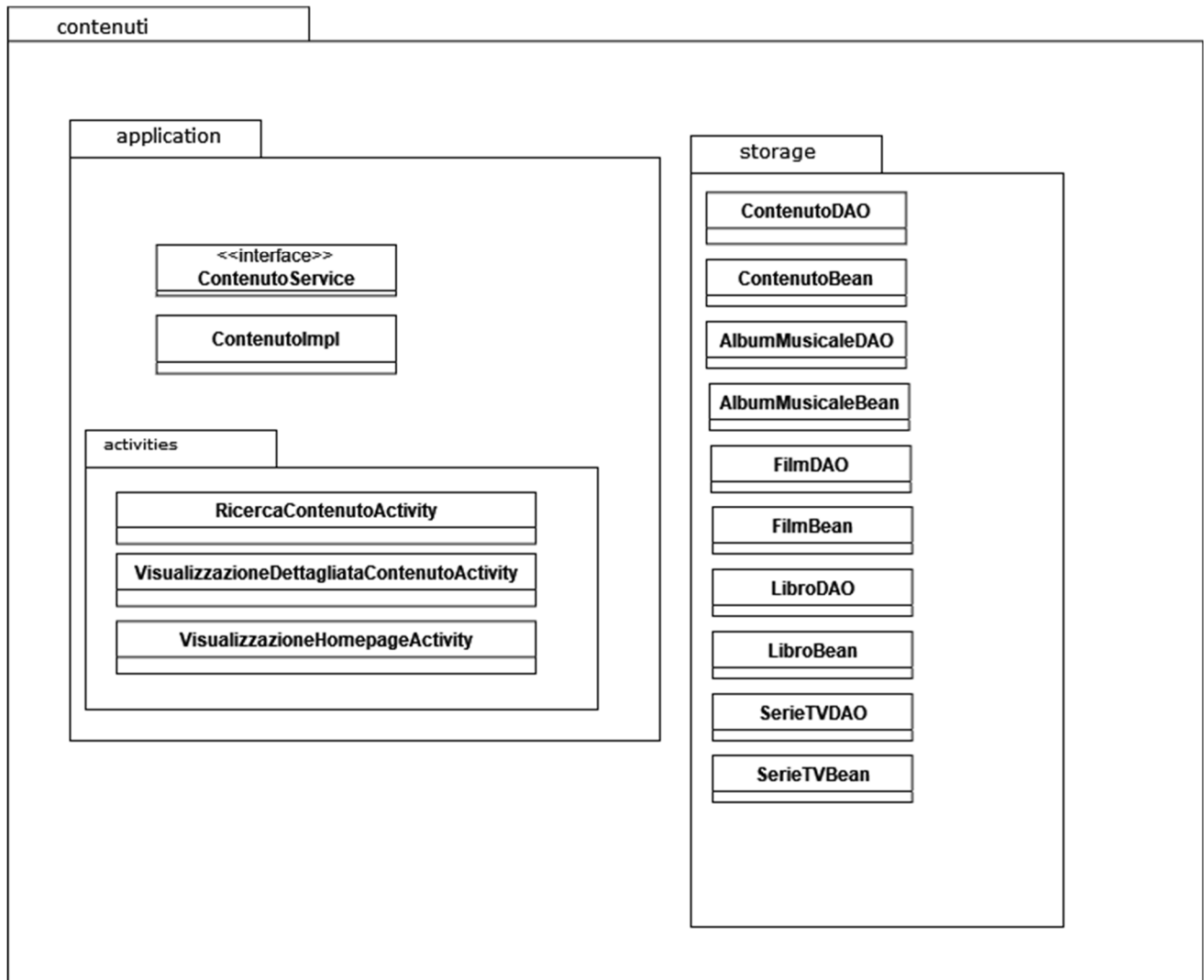




Package liste

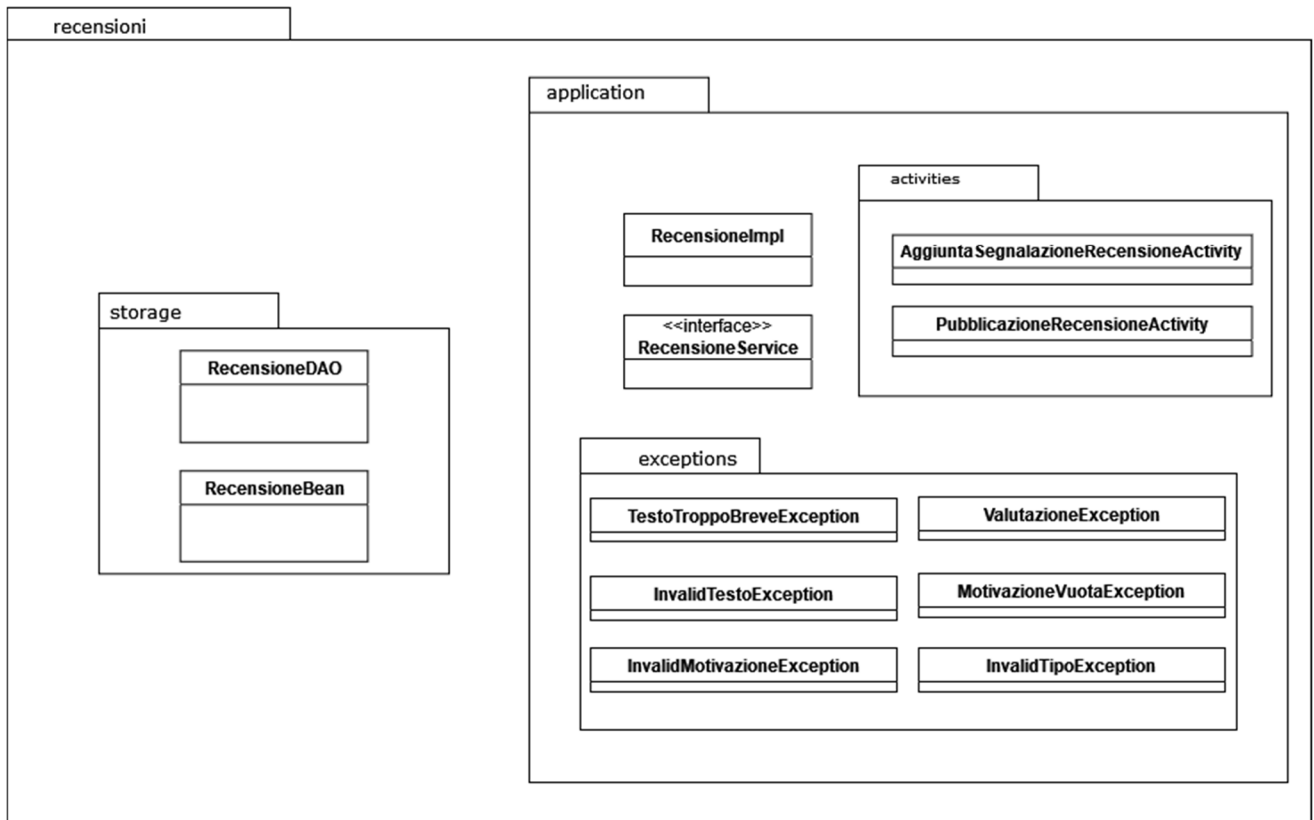


Package contenuti



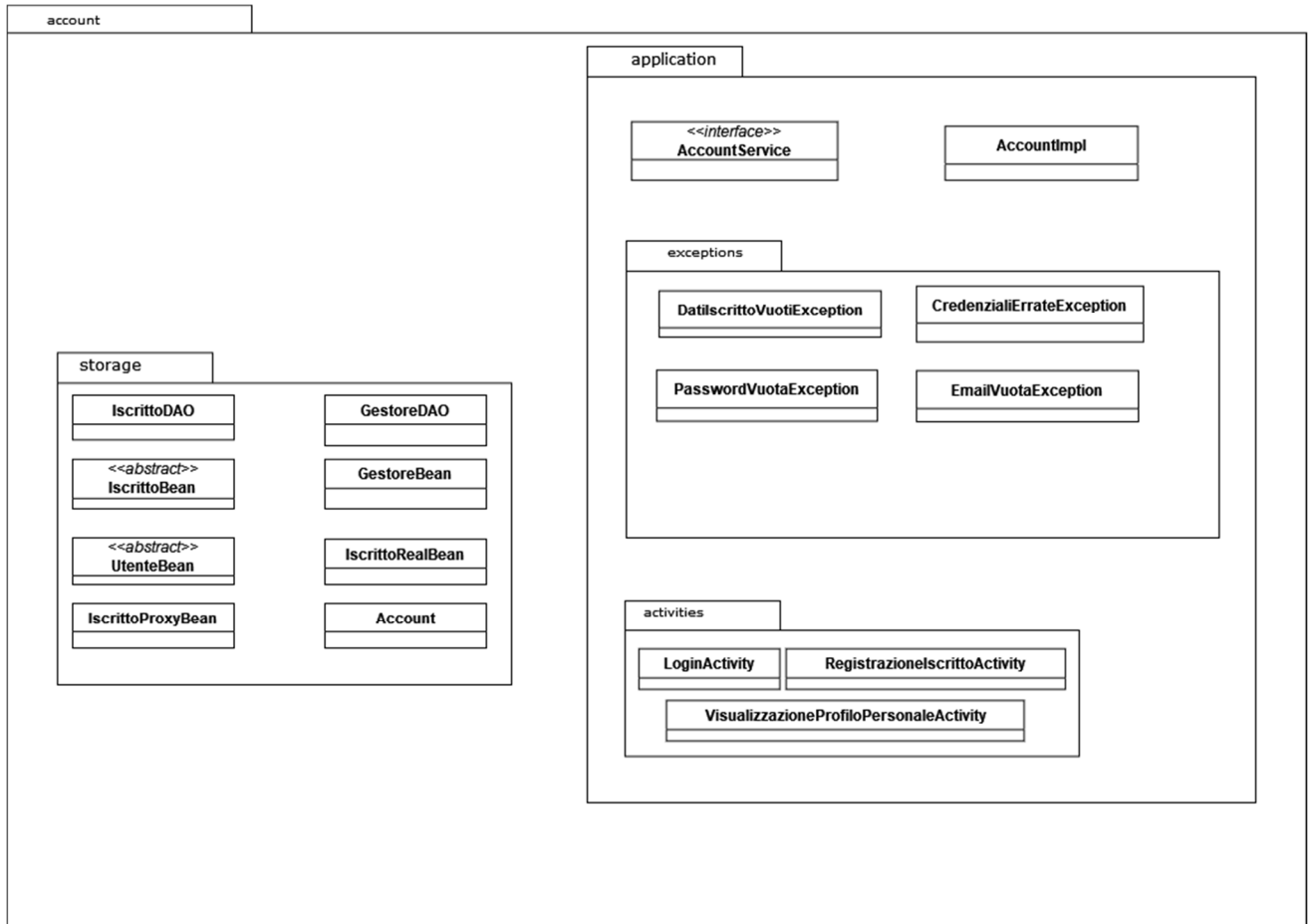


Package recensioni



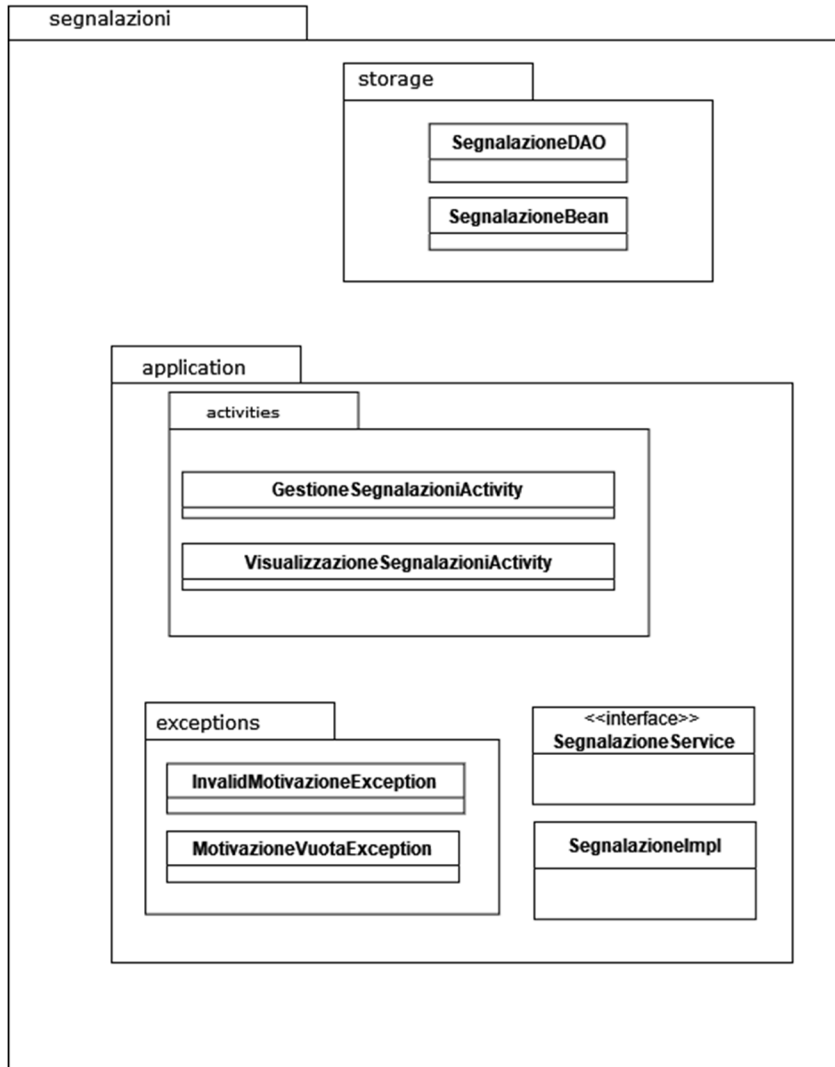


Package account

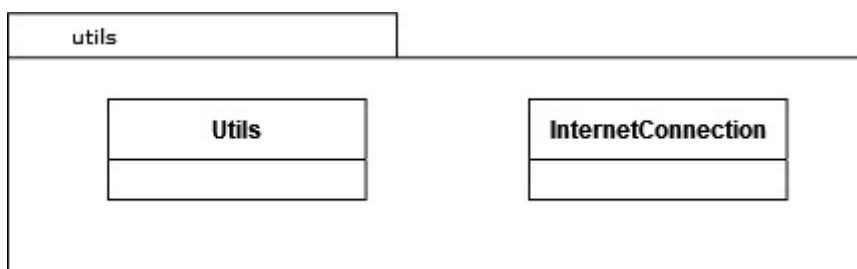




Package segnalazioni



Package utils





3 Class Interfaces

Di seguito sono presentate le interfacce delle classi di ciascun package. In particolare, sono riportate le interfacce *service*, relative cioè alla logica applicativa di ciascun package. Le classi per la realizzazione della GUI e la gestione dei dati persistenti, invece, non sono considerate.

Package *it.unisa.iLike.liste*

| | |
|----------------------|---|
| Nome classe | ListaService |
| Descrizione | Questa classe consente di gestire le operazioni relative alle liste. |
| Metodi | + creaLista(IscrittoBean i, String nome, boolean pubblica): <i>IscrittoBean</i> + aggiungiContenuto(ListaBean l, ContenutoBean c): <i>boolean</i> + getLista(String nome, String emailIscritto): <i>ListaBean</i> |
| Invariante di classe | / |

| | |
|----------------------------------|---|
| Firma metodo | + creaLista(IscrittoBean i, String nome, boolean pubblica): <i>IscrittoBean</i> |
| Descrizione | Questo metodo consente di aggiungere una nuova lista di nome <i>nome</i> all'insieme delle liste di un iscritto <i>i</i> . La lista aggiunta è pubblica o meno a seconda del valore dell'attributo booleano <i>pubblica</i> . Il metodo restituisce l' <i>IscrittoBean</i> aggiornato con la nuova lista. |
| Pre-condizione | context: <i>ListaService::creaLista(IscrittoBean i, String nome, boolean pubblica)</i> pre: $1 \leq \text{nome.length}() \leq 50$ AND NOT <i>hasLista(i, nome)</i> |
| Post-condizione | context: <i>ListaService::creaLista(IscrittoBean i, String nome, boolean pubblica)</i> post: $i.\text{hasLista}(\text{nome})$ AND $i.\text{getLista}(\text{nome}) = [\text{lista vuota}]$ |
| Eccezioni controllate lanciabili | - <i>NomeVuotoException</i> se l'argomento <i>nome</i> ha un numero di caratteri minore di 1; |



| | |
|---|--|
| | <ul style="list-style-type: none"> - <i>InvalidNomeException</i> se l'argomento <i>nome</i> ha un numero di caratteri maggiore di 50; - <i>ListaGiaEsistenteException</i> se l'iscritto ha già una lista di nome <i>nome</i>; |
| Firma metodo | + aggiungiContenuto(ListaBean l, ContenutoBean c): boolean |
| Descrizione | <p>Questo metodo consente di aggiungere un contenuto c ad una lista già esistente l. Restituisce un valore booleano che descrive l'esito dell'operazione:</p> <ul style="list-style-type: none"> - <i>true</i> se l'operazione è stata eseguita con successo; - <i>false</i> altrimenti. |
| Pre-condizione | <p>context: ListaService::aggiungiContenuto(ListaBean l, ContenutoBean c)</p> <p>pre: NOT l.contains(c)</p> |
| Post-condizione | <p>context: ListaService::aggiungiContenuto(ListaBean l, ContenutoBean c)</p> <p>post: l' = l + [c]</p> |
| Eccezioni controllate lanciabili | <ul style="list-style-type: none"> - <i>ContenutoGiaPresenteException</i> se il contenuto c è già presente nella lista l. |
| Firma metodo | + getLista(String nome, String emailiscritto): ListaBean |
| Descrizione | Questo metodo permette di ottenere un oggetto Lista identificato con l'email dell'iscritto e il nome della lista. |
| Pre-condizione | / |
| Post-condizione | / |
| Eccezioni controllate lanciabili | / |

Package *it.unisa.iLike.contenuti*

| | |
|--------------------|--|
| Nome classe | ContenutoService |
| Descrizione | Questa classe consente di gestire le operazioni relative ai contenuti. |



| | |
|-----------------------------|--|
| Metodi | + getById(int id): ContenutoBean + cerca(String titolo): List<ContenutoBean> + cerca(String titolo, int tipo): List<ContenutoBean> |
| Invariante di classe | / |

| | |
|---|--|
| Firma metodo | + getById(int id): ContenutoBean |
| Descrizione | Questo metodo restituisce un oggetto ContenutoBean contenente le informazioni relative al contenuto avente un dato id. Il metodo restituisce null se non esiste un contenuto che abbia l'id specificato. |
| Pre-condizione | / |
| Post-condizione | Sia c l'oggetto restituito. Allora $c.id = id$. |
| Eccezioni controllate lanciabili | / |
| Firma metodo | + cerca(String titolo): List<ContenutoBean> |
| Descrizione | Questo metodo restituisce una lista dei contenuti il cui titolo matcha con <i>titolo</i> . |
| Pre-condizione | / |
| Post-condizione | Sia <i>conts</i> la lista restituita dal metodo. Per ogni <i>c</i> in <i>conts</i> , $c.titolo \text{ like } '%titolo\%'$. |
| Eccezioni controllate lanciabili | / |
| Firma metodo | + cerca(String titolo, int tipo): List<ContenutoBean> |
| Descrizione | Questo metodo restituisce una lista dei contenuti d un dato tipo il cui titolo matcha con <i>titolo</i> . Il parametro <i>tipo</i> consente di selezionare il tipo di contenuto (0 per film, 1 per serie tv, 2 per libri, 3 per album musicali). |



| | |
|---|---|
| Pre-condizione | tipo = 0 OR tipo = 1 OR tipo = 2 OR tipo = 3 |
| Post-condizione | Sia <i>conts</i> la lista restituita dal metodo. Tutti i contenuti di <i>conts</i> sono del tipo specificato. Inoltre, per ogni <i>c</i> in <i>conts</i> , <i>c.titolo</i> like '%titolo%'. |
| Eccezioni controllate lanciabili | / |

Package *it.unisa.iLike.recensioni*

| | |
|-----------------------------|--|
| Nome classe | RecensioneService |
| Descrizione | Questa classe consente di gestire le operazioni relative alle recensioni. |
| Metodi | + creaRecensione (String testo, int valutazione, IscrittoBean i, ContenutoBean c): RecensioneBean + aggiungiSegnalazione (int tipo, String motivazione, RecensioneBean r, IscrittoBean i): boolean + getRecensione(int id): RecensioneBean |
| Invariante di classe | / |

| | |
|---------------------|--|
| Firma metodo | + creaRecensione (String testo, int valutazione, IscrittoBean i, ContenutoBean c): RecensioneBean |
| Descrizione | <p>Questo metodo consente di aggiungere una nuova recensione contenente un testo, una valutazione, una data (inserita dinamicamente), l'iscritto che ha composto la recensione, il contenuto cui quest'ultima si riferisce, un attributo booleano cancellata posto a <i>false</i> ed una motivazioneCancellazione che viene posta a <i>null</i>.</p> <p>Restituisce</p> <ul style="list-style-type: none"> - l'oggetto <i>RecensioneBean</i> contenente le informazioni relative alla recensione creata, se l'operazione è andata a buon fine; - <i>null</i> altrimenti. |



| | |
|---|---|
| Pre-condizione | <p>context: RecensioneService::creaRecensione (String testo, int valutazione, IscrittoBean i, ContenutoBean c)</p> <p>pre: $3 \leq \text{testo.length} \leq 1000$ AND $1 \leq \text{valutazione} \leq 5$</p> |
| Post-condizione | <p>context: RecensioneService::creaRecensione (String testo, int valutazione, IscrittoBean i, ContenutoBean c)</p> <p>post: sia R l'insieme delle recensioni prima della chiamata di questo metodo e sia r la recensione che si vuole aggiungere ad R. Dopo la chiamata a tale metodo avremo il nuovo insieme $R' = R \cup \{r\}$.</p> |
| Eccezioni controllate lanciabili | <ul style="list-style-type: none"> - <i>TestoTropoBreveException</i> se l'argomento <i>testo</i> ha un numero di caratteri minore di 3; - <i>InvalidTestoException</i> se l'argomento <i>testo</i> ha un numero di caratteri maggiore di 1000; - <i>ValutazioneException</i> se la variabile <i>valutazione</i> ricevuta come argomento dal metodo non ha un valore compreso tra 1 e 5. |
| Firma metodo | + aggiungiSegnalazione (int tipo, String motivazione, RecensioneBean r, IscrittoBean i): boolean |
| Descrizione | <p>Questo metodo permette di aggiungere una segnalazione ad una recensione, specificandone il tipo (0 per <i>Spoiler Alert</i>, 1 per <i>Altre Segnalazioni</i>) ed una motivazione. Contiene inoltre un attributo booleano gestita che viene posto a <i>false</i>.</p> <p>Restituisce un valore booleano che descrive l'esito dell'operazione:</p> <ul style="list-style-type: none"> - <i>true</i> se l'operazione è andata a buon fine; - <i>false</i> altrimenti. |
| Pre-condizione | <p>context: RecensioneService::aggiungiSegnalazione (int tipo, String motivazione, RecensioneBean r, IscrittoBean i)</p> <p>pre: $1 \leq \text{motivazione.length}() \leq 500$ AND $(\text{tipo}==0 \text{ OR } \text{tipo}==1)$</p> |
| Post-condizione | <p>context: RecensioneService::aggiungiSegnalazione (int tipo, String motivazione, RecensioneBean r, IscrittoBean i)</p> <p>post: sia S l'insieme delle segnalazioni riferite alla recensione r prima della chiamata a questo metodo e sia s la nuova segnalazione riferita ad r che si desidera aggiungere ad S. Dopo la chiamata a tale metodo avremo il nuovo insieme $S' = S \cup \{s\}$.</p> |



| | |
|---|--|
| Eccezioni controllate lanciabili | <ul style="list-style-type: none">- <i>MotivazioneVuotaException</i>: l'argomento <i>motivazione</i> ha un numero di caratteri minore di 1;- <i>InvalidMotivazioneException</i> l'argomento <i>motivazione</i> ha un numero di caratteri maggiore di 500.- <i>InvalidTipoException</i> l'argomento <i>tipo</i> ha un valore non contenuto nell'insieme dei valori ammissibili {0,1}. |
| Firma metodo | + getRecensione(int id): RecensioneBean |
| Descrizione | Questo metodo permette di recuperare la recensione attraverso il suo identificativo. |
| Pre-condizione | / |
| Post-condizione | / |
| Eccezioni controllate lanciabili | / |

Package *it.unisa.iLike.account*

| | |
|-----------------------------|---|
| Nome classe | AccountService |
| Descrizione | Questa classe consente di gestire le operazioni riferite all'account. |
| Metodi | + login(String email, String password): Account + logout(UtenteBean u): Account + registrazioneiscritto(String email, String password, String nome, String cognome, String nickname, String bio): Account |
| Invariante di classe | / |

Di seguito si riporta una descrizione dei metodi di *AccountService*. In questa descrizione con **a** si fa riferimento all'oggetto *Account* restituito dai metodi. Tale oggetto avrà due variabili d'istanza per determinare se l'attore è loggato durante l'esecuzione. Nel caso in cui non è stato effettuato il login



(oppure è stato effettuato il logout), tutte le variabili d'istanza di **a** avranno valore Null. In più la classe Account avrà il metodo isiscritto() che restituirà: True se è loggato l'iscritto, False se è loggato il gestore, Null se non è stato effettuato il login.

| | |
|---|---|
| Firma metodo | + login(String email, String password): Account |
| Descrizione | Questo metodo consente di recuperare l'utente dal DB. In particolare, restituisce un oggetto Account conterrà l'attore che ha effettuato la login, altrimenti in caso di errore null. |
| Pre-condizione | context: AccountService::login(String email, String password): pre: (3 <= email.length() <= 100 AND 8 <= password.length() <= 25) or (8 <= password.length() <= 25 AND 3 <= nickname.length() <= 30) |
| Post-condizione | context: AccountService::login(String email, String password): post: se email & password corrispondono all'iscritto: a.iscritto = {obj corrispondente}, a.gestore = Null se email & password corrispondono al gestore: a.iscritto = Null , a.gestore = {obj corrispondente} |
| Eccezioni controllate lanciabili | - <i>CredenzialiErrateException</i> : se l'email e/o la password sono errate e non è possibile individuare l'attore. |
| Firma metodo | + logout(UtenteBean u): Account |
| Descrizione | Questo metodo consente di effettuare il logout dell'utente. In particolare, restituisce un oggetto Account dove tutte le variabili hanno valore null. |
| Pre-condizione | / |
| Post-condizione | context: ProfiloService::logout(UtenteBean u): post: a.iscritto = Null, a.gestore = Null |
| Eccezioni controllate lanciabili | / |
| Firma metodo | + registrazioneiscritto(String email, String password, String nome, String cognome, String nickname, String bio): Account |



| | |
|---|---|
| Descrizione | <p>Questo metodo consente di effettuare la registrazione di un iscritto. In particolare, restituisce:</p> <ul style="list-style-type: none"> - <i>L'oggetto account</i>, contenente le informazioni sull'iscritto, se la registrazione è andata a buon fine; - <i>Null</i>, se la registrazione non è andata a buon fine. |
| Pre-condizione | <p>context: AccountService::registrazionelscritto(String email, String password, String nome, String cognome, String nickname, String bio):</p> <p>pre: email.match('^([A-Z0-9\\._-]+@[A-Z0-9\\._-]+\\.([A-z]{2,6})\$') AND password.match('^([A-Za-z])(?=.*\\d)[A-Za-z\\d]{8,}\$') AND 3 <= nome.length() <= 50 AND 3 <= cognome.length() <= 50 AND 3 <= nickname.length() <= 30</p> |
| Post-condizione | <p>context: AccountService::registrazionelscritto(String email, String password, String nome, String cognome, String nickname, String bio):</p> <p>post: a.iscritto = {obj corrispondente} AND a.gestore = Null AND sia S l'insieme contenente tutti gli iscritti di iLike e sia i il nuovo iscritto, allora S' = S + {i} con S' lo l'insieme aggiornato.</p> |
| Eccezioni controllate lanciabili | <ul style="list-style-type: none"> - <i>EmailVuotaException</i>: se il campo email non rispetta il limite minimo o massimo; - <i>PasswordVuotaException</i>: se il campo password non rispetta il limite minimo o massimo; - <i>DatiscrittoVuotiException</i>: se i campi nome, cognome, nickname non rispettano il limite minimo o massimo. |

Package *it.unisa.iLike.segnalazioni*

| | |
|-----------------------------|---|
| Nome classe | SegnalazioneService |
| Descrizione | Questa classe consente di gestire le operazioni relative alle segnalazioni |
| Metodi | <p>+ getSegnalazioni(): List<SegnalazioneBean></p> <p>+ cancellaRecensione (SegnalazioneBean s, String motivazione, GestoreBean g): boolean</p> <p>+ rifiutaSegnalazione(SegnalazioneBean s, GestoreBean g): boolean</p> <p>+ getSegnalazione(int id): SegnalazioneBean</p> |
| Invariante di classe | / |



| | |
|---|--|
| Firma metodo | + getSegnalazioni(): List<SegnalazioneBean> |
| Descrizione | Questo metodo consente al gestore di visualizzare tutte le segnalazioni che ancora non sono state gestite. |
| Pre-condizione | / |
| Post-condizione | context: SegnalazioneService::getSegnalazioni () post: Per ogni segnalazione s nella lista restituita, s.gestita = false |
| Eccezioni controllate lanciabili | / |
| Firma metodo | + cancellaRecensione (SegnalazioneBean s, String motivazione, GestoreBean g): boolean |
| Descrizione | Questo metodo può essere chiamato all'interno dell'applicazione solo da chi dispone di un account gestore e consente di cancellare una recensione in seguito alla ricezione di una o più segnalazioni. Purché ciò avvenga occorre fornire al metodo la segnalazione relativa alla recensione da cancellare ed una stringa contenente la motivazione della cancellazione. Il metodo incrementa inoltre il numero di segnalazioni gestite dal gestore. Restituisce un valore booleano che descrive l'esito dell'operazione: <ul style="list-style-type: none"> - <i>true</i> se l'operazione è andata a buon fine; - <i>false</i> altrimenti. |
| Pre-condizione | context: SegnalazioneService::cancellaRecensione(SegnalazioneBean s, String motivazione, GestoreBean g) pre: 1 <= motivazione.length() <= 300 |
| Post-condizione | context: SegnalazioneService::cancellaRecensione(SegnalazioneBean s, String motivazione, GestoreBean g) post: s.gestita = true AND s.getRecensione().cancellata = true AND g.numSegnalazioniGestite += 1 |
| Eccezioni controllate lanciabili | <ul style="list-style-type: none"> - <i>MotivazioneVuotaException</i> se l'argomento <i>motivazione</i> ha un numero di caratteri minore di 1; - <i>InvalidMotivazioneException</i> se l'argomento <i>motivazione</i> ha un numero di caratteri maggiore di 300. |
| Firma metodo | + rifiutaSegnalazione(SegnalazioneBean s, GestoreBean g): boolean |



| | |
|---|---|
| Descrizione | Questo metodo permette al gestore di ignorare una segnalazione, rifiutandola in quanto non veritiera. Inoltre, il metodo incrementa il numero di segnalazioni gestite dal gestore. Restituisce un boolean che assume valore: <ul style="list-style-type: none"> - true, se il rifiuto della segnalazione passata come parametro va a buon fine; - false altrimenti. |
| Pre-condizione | / |
| Post-condizione | context: SegnalazioneService::rifiutaSegnalazione(SegnalazioneBeans, GestoreBean g) post: s.gestita = true and s.getRecensione().cancellata = false AND g.numSegnalazioniGestite += 1 |
| Eccezioni controllate lanciabili | / |
| Firma metodo | + getSegnalazione(int id): SegnalazioneBean |
| Descrizione | Questo metodo permette di ottenere un oggetto segnalazione, identificato con il suo id. |
| Pre-condizione | / |
| Post-condizione | / |
| Eccezioni controllate lanciabili | / |

Il metodo di utilità usato per la verifica delle precondizioni

- hasLista(Iscritto i, String nome): boolean

sarà implementato nella classe *Utils* del package *it.unisa.iLike.utils*.

Inoltre, tutte le classi utilizzate per il lancio delle eccezioni modellano eccezioni controllate, per cui estendono *java.lang.Exception*.



4 Class Diagram

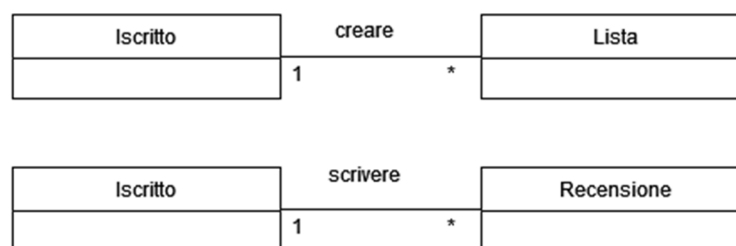
[Link Class Diagram.](#)

5 Design Patterns

Il sistema utilizzerà il design pattern *Proxy*, per il quale si descrivono il contesto di applicazione, la soluzione utilizzata e i risultati conseguiti.

DP Proxy

L'utilizzo di questo design pattern è finalizzato a rinviare la completa inizializzazione di un oggetto *IscrittoBean* al momento in cui tale operazione si rende effettivamente necessaria. Consideriamo innanzitutto le associazioni dell'entity *Iscritto* che motiveranno l'utilizzo del proxy.



Essendo di tipo 1:N, in fase di implementazione tali associazioni saranno mappate aggiungendo ai campi di *IscrittoBean*

- una collezione di riferimenti ad oggetti *Lista*, contenente le liste dell'iscritto;
- una collezione di riferimenti ad oggetti *Recensione*, contenente le recensioni pubblicate dall'iscritto.

Pertanto, per ciascun iscritto, il corrispondente bean conterrà le informazioni relative

- a) al nickname;
- b) al nome;
- c) al cognome;
- d) alla bio;
- e) alle liste e – indirettamente – ai contenuti di ciascuna lista;
- f) alle recensioni pubblicate.



Il caricamento dell'informazione **e**

- si rende necessario soltanto per la realizzazione della funzionalità *Visualizzazione profilo personale* (RF_GP_10) e *Aggiunta contenuti liste* (RF_GP_7);
- risulta particolarmente oneroso, considerata la necessità di eseguire query complesse tramite cui ottenere informazioni memorizzate in più tabelle.

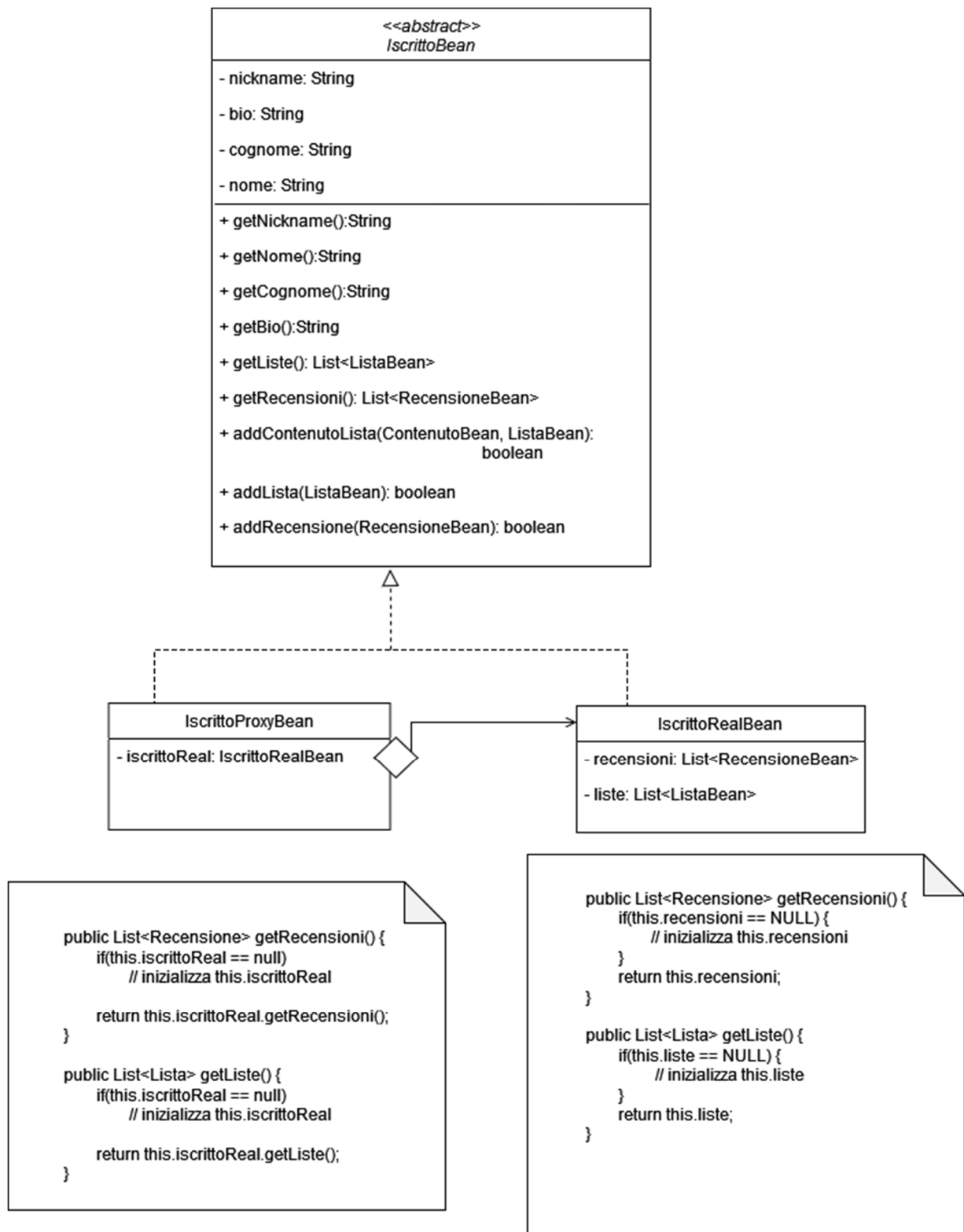
Il caricamento dell'informazione **f**

- si rende necessario soltanto per la realizzazione delle funzionalità *Visualizzazione profilo personale* (RF_GP_10)
- risulta particolarmente oneroso, considerata la necessità di eseguire query complesse tramite cui ottenere informazioni memorizzate in più tabelle.

Pertanto, allo scopo di caricare le informazioni **e** e **f** solo quando necessario, si ritiene opportuno utilizzare il design pattern Proxy.

Come rappresentato nel diagramma UML seguente, le classi *IscrittoProxy* e *IscrittoReal* estendono entrambe *Iscritto*, cioè una classe astratta che specifica l'interfaccia del servizio. Il suo scopo primario è nascondere la reale implementazione corrente. La classe *IscrittoProxy* mantiene, dunque, come variabile di istanza un oggetto *IscrittoReal*.

Tale oggetto è istanziato soltanto quando è effettivamente necessario, ossia durante l'esecuzione di *Visualizzazione profilo personale* e *Aggiunta contenuti liste*.





6 Glossario

| Sigla/Termine | Definizione |
|-----------------------|--|
| Categoria | Differenziazione di contenuti dello stesso tipo (es. horror e commedie per i film, jazz e pop per gli album musicali, ecc.). |
| Contenuto | Elemento appartenente all'insieme di film, serie TV, libri e album musicali. |
| Credenziali | Per il gestore sono l'indirizzo email e la password. Per l'iscritto sono l'indirizzo email (o in alternativa il nickname) e la password. |
| Star rating | Input type che consente di esprimere una valutazione indicata da stelle (un maggior numero di stelle indica una maggiore qualità). |
| Tipo | Film, serie TV, libro oppure album musicale. |
| Valutazione contenuto | Numero di stelle assegnate dal recensore al contenuto recensito. |