Object Design Document

Versione	Data	Descrizione	Autori
0.1	15/1/2020	Prima stesura	Team
0.2	19/1/2020	Descrizioni packages	Umberto Loria
0.3	20/1/2020	Diagrammi dei packages	Umberto Loria
0.4	21/1/2020	Introduzione OCL	Umberto Loria

Indice

Introduzione

Questo documento presenta le decisioni implementative del sistema **Moovie**. Forti di una progettazione documentata nell'SDD, e di un analisi descritta nel RAD, possiamo finalmente procedere ai dettagli implementativi. Il sistema verrà sviluppato tramite una **Web Application**.

Compromessi

Qui di seguito sono motivate le principali scelte di progettazione/implementazione del sistema riguardo i compromessi che spesso si presentano nello sviluppo di sistemi software.

Sicurezza vs efficienza

La sicurezza è un aspetto fondamentale per una Web Application. Per questa motivazione, la sicurezza viene garantita **ripetendo i controlli di autorizzazione** in tutti i punti di accesso al sistema. Per esempio, non solo una determinata funzionalità è "nascosta" all'utente non autorizzato a livello di presentazione, ma questa funzionalità ripeterà i controlli di autorizzazione anche nei punti di accesso a queste funzionalità lato server (vedi "controller" in basso).

Rapido sviluppo vs funzionalità

Tutte le funzionalità previste dai documenti precedenti sono state sviluppate con robustezza come obiettivo. Si è deciso, quindi, di attribuire robustezza a tutte le funzionalità sviluppate, sacrificando la rapidità dei tempi di sviluppo.

Efficienza vs portabilità

La portabilità di un sistema software è un fattore sicuramente determinante. La "sopravvivenza" di architetture software è spesso impedita dalla poca lungimiranza dei progettisti. Per questa ragione, questo sistema predilige la portabilità a scapito all'efficienza.

Convenzioni del codice

La convenzione che è stata subito stabilita, prima ancora dello stile d'indentazione, è la seguente: il codice avrebbe dovuto essere leggibile e comprensibile senza l'uso dei commenti. Questa affermazione è forte: in effetti ci sono parti di codice in cui è stato necessario introdurre commenti per stabilire una giusta comprensibilità. Comunque, è stato sempre presente l'impegno, da parte del team, di rendere il codice stesso chiaro ed esplicativo.

Le convenzioni applicate al codice sono:

- i nomi delle variabili devono essere intuitivi;
- i nomi delle variabili devono essere brevi;
- i nomi delle variabili devono essere significativi per il loro contesto di utilizzo;
- le parole che comporranno i nomi delle variabili saranno separate dal trattino basso;
- i nomi dei metodi dovranno suggerire le funzionalità che implementano;
- le parole che compongono i nomi delle classi avranno la prima lettera maiuscola;
- i nomi delle classi dovranno essere esplicativi di quello che andranno a realizzare.

Divisione dei packages

Questa è la gerarchia dei packages dell'applicazione.

```
moovie
├──controllers
│   ├──account
│   ├──amicizia
│   ├──film
│   ├──gestione
│   └──ricerca
├──dao
│   ├──factories
│   ├──implementation
│   └──interfaces
├──models
│       ...
└──views
    │   ...
    ├──account
```

```
    ├──amicizia
    ├──film
    ├──gestione
    └──ricerca
```

I package **controllers**, **dao**, **models** e **views** verranno singolarmente descritti nei paragrafi che seguono.

Package Models

Il package **models** contiene tutte le classi contenenti informazioni del dominio applicativo.

Classe	Descrizione
Amicizia	Descrive la relazione di amicizia tra due utenti. Può essere accettata oppure non ancora (in questo caso si parla di richiesta di amicizia).
Artista	Rappresenta le informazioni di un artista.
Film	Rappresentazione delle informazioni di un film.
Genere	Descrive le informazioni di un genere.
Giudizio	Rappresenta il giudizio di un utente su un film.
Promemoria	Descrive un promemoria creato da un utente per un certo film.
Recitazione	Contiene le informazioni sul personaggio interpretato da un attore in un film.
Utente	Rappresenta le informazioni di un utente registrato sul sito.

Package DAO

Questa è la struttura completa del package **dao**. Come recita il nome del package, questo conterrà classi che realizzano il pattern architetturale DAO (per ulteriori informazioni, vedi la sezione "Design pattern" in basso).

```
dao
├──factories
├──implementation
└──interfaces
```

Le interfacce, contenute nel package **dao/interfaces**, impongono i servizi che ogni DAO deve offrire. Le implementazioni reali dei DAO, che interagiscono con la base di dati, si trovato nel package **dao/implementations**. Tuttavia, queste classi non vengono istanziate direttamente dall'utilizzatore, bensì dalle classi nel package **dao/factories**. L'utilizzatore quindi utilizza le suddette classi factory per ottenere istanze delle interfacce DAO.

Interfaces

Il package **dao/interfaces** contiene delle interfacce che definiscono le modalità di fruizione dei dati persistenti del sistema. Tramite le factories (descritte sotto) è possibile fornire delle implementazioni di queste interfacce nascondendo all'utilizzatore il vero luogo in cui questi dati vengono salvati.

Queste interfacce sono:

- IAccountDAO;
- IAmiciziaDAO;
- IArtistaDAO;
- IFilmDAO;
- IGenereDAO;
- IGiudizioDAO;
- IPromemoriaDAO;
- IRecitazioneDAO;
- IRegiaDAO.

IAccountDAO

Metodo	Descrizione
bool exists(string email)	Indica se esiste un utente associato l'indirizzo e-mail fornito.
Utente create(Utente utente)	Aggiunge un nuovo utente con le informazioni fornite.
Utente findByID(int id)	Restituisce le informazioni dell'utente con l'ID fornito.
Utente update(Utente utente)	Aggiorna le informazioni di un utente esistente.
Utente authenticate(string email, string password)	Restituisce le informazioni dell'utente con EMAIL e PASSWORD fornite.
Utente[] search(string fulltext)	Ricerca gli utenti con campi NOME e COGNOME correlati al parametro FULLTEXT fornito.
bool delete(int id)	Rimuove l'utente con l'ID fornito.

```
context IAccountDAO::exists(email:string) pre:
    email <> null
context IAccountDAO::exists(email:string) post:
    result = (esiste utente u nel DB : utente.email = email)

context IAccountDAO::create(utente:Utente) pre:
    utente <> null and
    utente.nome <> null and
    utente.cognome <> null and
    utente.email <> null and
    not self.exists(utente.email) and
    utente.password <> null
context IAccountDAO::create(utente:Utente) post:
    result <> null and
    result.nome = utente.nome and
    result.cognome = utente.cognome and
    result.email = utente.email and
    result.password = utente.password and
    result.isGestore = utente.isGestore and
    self.findByID(result.id) = result
```

```
context IAccountDAO::findByID(id:int) pre:
    esiste utente u nel DB : utente.id = id
context IAccountDAO::findByID(id:int) post:
    result = u


context IAccountDAO::update(utente:Utente) pre:
    utente <> null and
    self.findByID(utente.id) <> null and
    utente.password <> null
context IAccountDAO::update(utente:Utente) post:
    result <> utente and self.findByID(result.id) = result


context IAccountDAO::authenticate(email:string, password:string) pre:
    esiste utente u nel DB : utente.email = email and utente.password = password
context IAccountDAO::authenticate(email:string, password:string) post:
    result = u


context IAccountDAO::search(fulltext:string) pre:
    fulltext <> null
context IAccountDAO::search(fulltext:string) post:
    result = (tutti gli utenti nel DB che hanno similitudini col parametro fulltext)


context IAccountDAO::delete(id:int) pre:
    self.findByID(id) <> null
context IAccountDAO::delete(id:int) post:
    result = true and self.findByID(id) = null
```

IAmiciziaDAO

Metodo	Descrizione
Amicizia[] getFriendships(int user_id)	Restituisce tutte le amicizie accettate che coinvolgono un utente.
Amicizia[] getRequests(int user_id)	Restituisce tutte le richieste di amicizia che coinvolgono un utente.
bool existsSomethingBetween(int user1, int user2)	Indica se esiste una relazione (amicizia richiesta o accettata) tra due utenti.
bool requestFriendshipFromTo(int user_from, int user_to)	Invia una richiesta di amicizia da un utente verso un altro.
bool existsRequestFromTo(int user_from, int user_to)	Indica se esiste una richiesta di amicizia inviata da un utente verso un altro.
bool removeFriendshipRequestFromTo(int user_from, int user_to)	Cancella la richiesta di amicizia inviata da un utente verso un altro.

Metodo	Descrizione
bool acceptFriendshipRequestFromTo(int user_from, int user_to)	Trasforma la richiesta di amicizia inviata da un utente verso un altro in una amicizia accettata.
bool refuseFriendshipRequestFromTo(int user_from, int user_to)	Rifiuta la richiesta di amicizia inviata da un utente verso un altro.
bool existsFriendshipBetween(int user1, int user2)	Indica se esiste un'amicizia accettata condivisa tra due utenti forniti.
bool removeFriendshipBetween(int user1, int user2)	Rimuove un'amicizia accettata condivisa tra due utenti forniti.

```
context IAmiciziaDAO::getFriendships(user_id:int) post:
    result = (tutte le amicizia a nel DB :
                (a.utente_from = user_id or a.utente_to = user_id)
                and a.timestamp_accettazione <> null)


context IAmiciziaDAO::getRequests(user_id:int) post:
    result = (tutte le amicizia a nel DB :
                (a.utente_from = user_id or a.utente_to = user_id)
                and a.timestamp_accettazione = null)


context IAmiciziaDAO::existsSomethingBetween(user1:int, user2:int) post:
    result = (esiste amicizia a nel DB :
                    (a.utente_from = user1 and a.utente_to = user2) or
                    (a.utente_from = user2 and a.utente_to = user1))


context IAmiciziaDAO::requestFriendshipFromTo(user_from:int, user_to:int) pre:
    not self.existsSomethingBetween(user_from, user_to)
context IAmiciziaDAO::requestFriendshipFromTo(user_from:int, user_to:int) post:
    result = true


context IAmiciziaDAO::existsRequestFromTo(user_from:int, user_to:int) post:
    result = (esiste amicizia a nel DB : a.utente_from = user_from and a.utente_to = user_to
                    and a.timestamp_accettazione = null)


context IAmiciziaDAO::removeFriendshipRequestFromTo(user_from:int, user_to:int) pre:
    self.existsRequestFromTo(user_from, user_to)
context IAmiciziaDAO::removeFriendshipRequestFromTo(user_from:int, user_to:int) post:
    result = true


context IAmiciziaDAO::acceptFriendshipRequestFromTo(user_from:int, user_to:int) pre:
    self.existsRequestFromTo(user_from, user_to)
context IAmiciziaDAO::acceptFriendshipRequestFromTo(user_from:int, user_to:int) post:
    result = true
```

```
context IAmiciziaDAO::refuseFriendshipRequestFromTo(user_from:int, user_to:int) pre:
    self.existsRequestFromTo(user_from, user_to)
context IAmiciziaDAO::refuseFriendshipRequestFromTo(user_from:int, user_to:int) post:
    result = true


context IAmiciziaDAO::existsFriendshipBetween(user1:int, user2:int) post:
    result = (esiste amicizia a nel DB :
                    ((a.utente_from = user1 and a.utente_to = user2) or
                    (a.utente_from = user2 and a.utente_to = user1)) and
                    a.timestamp_accettazione <> null)


context IAmiciziaDAO::removeFriendshipBetween(user1:int, user2:int) pre:
    self.existsFriendshipBetween(user1, user2)
context IAmiciziaDAO::removeFriendshipBetween(user1:int, user2:int) post:
    result = true
```

IArtistaDAO

Metodo	Descrizione
Artista findByID(int id)	Restituisce le informazioni dell'artista con l'ID fornito.
bin downloadFaccia(int id)	Preleva l'immagine dell'artista con l'ID fornito.
bool uploadFaccia(int id, bin faccia_bin)	Memorizza l'immagine dell'artista con l'ID fornito.
Artista[] search(string fulltext)	Ricerca gli artisti con campi NOME e DESCRIZIONE correlati al parametro FULLTEXT fornito.
Artista create(Artista artista, bin faccia_bin)	Aggiunge un nuovo artista con le informazioni fornite.
Artista[] getAll()	Preleva tutti gli artisti memorizzati.
Artista update(Artista artista)	Aggiorna le informazioni di un artista esistente.
bool delete(int id)	Rimuove l'artista con l'ID fornito.

```
context IArtistaDAO::findByID(id:int) pre:
    esiste artista a nel DB : artista.id = id
context IArtistaDAO::findByID(id:int) post:
    result = a


context IArtistaDAO::downloadFaccia(id:int) pre:
    self.findByID(id) <> null
context IArtistaDAO::downloadFaccia(id:int) post:
    result <> null


context IArtistaDAO::uploadFaccia(id:int, faccia_bin:bin) pre:
    faccia_bin <> null and self.findByID(id) <> null
context IArtistaDAO::uploadFaccia(id:int, faccia_bin:bin) post:
```

```
    result = true and self.downloadFaccia(id) = faccia_bin


context IArtistaDAO::search(fulltext:string) pre:
    fulltext <> null
context IArtistaDAO::search(fulltext:string) post:
    result = tutti gli artisti nel DB che hanno similitudini col parametro fulltext


context IArtistaDAO::create(artista:Artista, faccia_bin:bin) pre:
    artista <> null and
    artista.nome <> null and
    artista.nascita <> null and
    artista.descrizione <> null and
    faccia_bin <> null and
context IArtistaDAO::create(artista:Artista, faccia_bin:bin) post:
    result <> null and
    result.nome = artista.nome and
    result.nascita = artista.nascita and
    result.descrizione = artista.descrizione and
    self.findByID(result.id) = result and
    self.downloadFaccia(result.id) = faccia_bin


context IArtistaDAO::getAll() post:
    result = tutti gli artisti nel DB


context IArtistaDAO::update(artista:Artista) pre:
    artista <> null and
    self.findByID(artista.id) <> null and
    artista.nome <> null and
    artista.nascita <> null and
    artista.descrizione <> null
context IArtistaDAO::update(artista:Artista) post:
    result = artista and self.findByID(result.id) = result


context IArtistaDAO::delete(id:int) pre:
    self.findByID(id) <> null
context IArtistaDAO::delete(id:int) post:
    result = true and self.findByID(id) = null
```

IFilmDAO

Metodo	Descrizione
Film findByID(int id)	Restituisce le informazioni del film con l'ID fornito.
Film[] search(string fulltext)	Ricerca i film con campi NOME e DESCRIZIONE correlati al parametro FULLTEXT fornito.
Film[] suggestMe(int utente_id)	Preleva i film più in linea con le preferenze cinematografiche dell'utente con l'ID fornito.
bin downloadCopertina(int id)	Preleva l'immagine del film con l'ID fornito.

Metodo	Descrizione
bool uploadCopertina(int id, bin copertina_bin)	Memorizza l'immagine del film con l'ID fornito.
Film[] getClassifica()	Preleva i film meglio giudicati dalla community degli utenti.
Film create(Film film, bin copertina_bin)	Aggiunge un nuovo film con le informazioni fornite.
Film update(Film film)	Aggiorna le informazioni di un film esistente.
bool delete(int id)	Rimuove il film con l'ID fornito.

```
context IFilmDAO::findByID(id:int) pre:
    esiste film f nel DB : film.id = id
context IFilmDAO::findByID(id:int) post:
    result = f


context IFilmDAO::search(fulltext:string) pre:
    fulltext <> null
context IFilmDAO::search(fulltext:string) post:
    result = tutti i film nel DB che hanno similitudini col parametro fulltext


context IFilmDAO::suggestMe(utente_id:int) pre:
    utente_id > 0
context IFilmDAO::suggestMe(utente_id:int) post:
    result = 5 film in linea con le preferenze cinematografiche dell'utente con id = utente_id


context IFilmDAO::downloadCopertina(id:int) pre:
    self.findByID(id) <> null
context IFilmDAO::downloadCopertina(id:int) post:
    result <> null


context IFilmDAO::uploadCopertina(id:int, copertina_bin:bin) pre:
    copertina_bin <> null and self.findByID(id) <> null
context IFilmDAO::uploadCopertina(id:int, copertina_bin:bin) post:
    result = true and self.downloadCopertina(id) = copertina_bin


context IFilmDAO::getClassifica() post:
    result = i 50 film meglio votati dalla community degli utenti


context IFilmDAO::create(film:Film, copertina_bin:bin) pre:
    film <> null and
    film.titolo <> null and
    film.durata > 0 and
    film.anno >= 1900 and
    film.descrizione <> null and
    copertina_bin <> null
context IFilmDAO::create(film:Film, copertina_bin:bin) post:
```

```
    result <> null and
    result.titolo = film.titolo and
    result.durata = film.durata and
    result.anno = film.anno and
    result.descrizione = film.descrizione and
    self.findByID(result.id) = result and
    self.downloadCopertina(result.id) = copertina_bin


context IFilmDAO::update(film:Film) pre:
    film <> null and
    self.findByID(film.id) <> null and
    film.titolo <> null and
    film.durata > 0 and
    film.anno >= 1900 and
    film.descrizione <> null
context IFilmDAO::update(film:Film) post:
    result = film and self.findByID(result.id) = result


context IFilmDAO::delete(id:int) pre:
    self.findByID(id) <> null
context IFilmDAO::delete(id:int) post:
    result = true and self.findByID(id) = null
```

IGenereDAO

Metodo	Descrizione
Genere findByID(int id)	Restituisce le informazioni del genere con l'ID fornito.
int[] findGeneriByFilm(int film_id)	Restituisce gli ID dei generi associati ad un film fornito.
int[] findFilmsByGenere(int id)	Restituisce gli ID dei film del genere fornito.
Genere[] getAll()	Preleva tutti i generi memorizzati.
bool setOnly(int film_id, int[] assign_genere_ids)	Associa ad un film solo i generi forniti, disassociandolo con tutti gli altri.
Genere create(Genere genere)	Aggiunge un nuovo genere con il nome fornito.
Genere update(Genere genere)	Aggiorna le informazioni di un genere esistente.
bool delete(int id)	Rimuove il genere con l'ID fornito.
bool exists(string nome)	Indica se esiste un genere chiamato con il nome fornito.

```
context IGenereDAO::findByID(id:int) pre:
    esiste genere g nel DB : genere.id = id
context IGenereDAO::findByID(id:int) post:
    result = g


context IGenereDAO::findGeneriByFilm(film_id:int) pre:
    film_id > 0
context IGenereDAO::findGeneriByFilm(film_id:int) post:
```

```
    result = (tutti i generi g nel DB : self.findFilmsByGenere(g.id).include(film_id))


context IGenereDAO::findFilmsByGenere(id:int) pre:
    id > 0
context IGenereDAO::findFilmsByGenere(id:int) post:
    result = (tutti i film f nel DB : self.findGeneriByFilm(f.id).include(id))


context IGenereDAO::getAll() post:
    result = tutti i generi nel DB


context IGenereDAO::setOnly(film_id:int, assign_genere_ids:int[]) pre:
    film_id > 0 and assign_genere_ids <> null
context IGenereDAO::setOnly(film_id:int, assign_genere_ids:int[]) post:
    self.findGeneriByFilm(film_id) = assign_genere_ids


context IGenereDAO::create(genere:Genere) pre:
    genere <> null and
    genere.nome <> null and
    not self.exists(genere.nome)
context IGenereDAO::create(genere:Genere) post:
    result <> null and
    result.nome = genere.nome and
    self.findByID(result.id) = result


context IGenereDAO::update(genere:Genere) pre:
    genere <> null and
    self.findByID(genere.id) <> null and
    genere.nome <> null
context IGenereDAO::update(genere:Genere) post:
    result = genere and self.findByID(result.id) = result


context IGenereDAO::delete(id:int) pre:
    self.findByID(id) <> null
context IGenereDAO::delete(id:int) post:
    result = true and self.findByID(id) = null


context IGenereDAO::exists(nome:string) pre:
    nome <> null
context IGenereDAO::exists(nome:string) post:
    result = (esiste genere g nel DB : genere.nome = nome)
```

IGiudizioDAO

Metodo	Descrizione
bool create(Giudizio giudizio)	Aggiunge il giudizio fornito.
bool update(Giudizio giudizio)	Aggiorna il giudizio fornito.

Metodo	Descrizione
bool delete(Giudizio giudizio)	Rimuove il giudizio fornito.
Giudizio[] findByUtenti(int[] utenti_ids)	Preleva tutti i giudizi espressi degli utenti forniti.
Giudizio findByUtenteAndFilm(int utente_id, int film_id)	Preleva (se esiste) il giudizio espresso dall'utente fornito verso il film fornito.
bool exists(int utente_id, int film_id)	Indica se esiste un giudizio espresso da un utente fornito verso un film fornito.

```
context IGiudizioDAO::create(giudizio:Giudizio) pre:
    giudizio <> null and
    1 <= giudizio.voto <= 10 and
    not self.exists(giudizio.utente, giudizio.film)
context IGiudizioDAO::create(giudizio:Giudizio) post:
    result = giudizio and
    self.findByUtenteAndFilm(result.utente, result.film) = result


context IGiudizioDAO::update(giudizio:Giudizio) pre:
    giudizio <> null and
    1 <= giudizio.voto <= 10 and
    self.exists(giudizio.utente, giudizio.film)
context IGiudizioDAO::update(giudizio:Giudizio) post:
    result = giudizio and
    self.findByUtenteAndFilm(result.utente, result.film) = result


context IGiudizioDAO::delete(giudizio:Giudizio) pre:
    giudizio <> null and self.exists(giudizio.utente, giudizio.film)
context IGiudizioDAO::delete(giudizio:Giudizio) post:
    result = true and not self.exists(giudizio.utente, giudizio.film)


context IGiudizioDAO::findByUtenti(utenti_ids:int[]) pre:
    utenti_ids <> null
context IGiudizioDAO::findByUtenti(utenti_ids:int[]) post:
    result = (tutti i giudizi g nel DB : utenti_ids.include(g.utente))


context IGiudizioDAO::findByUtenteAndFilm(utente_id:int, film_id:int) post:
    result = (giudizio g nel DB : g.utente = utente_id and g.film = film.id)


context IGiudizioDAO::exists(utente_id:int, film_id:int) post:
    result = (self.findByUtenteAndFilm(utente_id, film_id) <> null)
```

IPromemoriaDAO

Metodo	Descrizione
Promemoria[] findByUtente(int utente_id)	Preleva tutti i promemoria salvati da un utente fornito.

Metodo	Descrizione
bool exists(int utente_id, int film_id)	Indica se esiste un promemoria salvato da un utente fornito verso un film fornito.
bool create(Promemoria promemoria)	Aggiunge il promemoria fornito.
bool delete(Promemoria promemoria)	Rimuove un promemoria fornito.
Promemoria findByUtenteAndFilm(int utente_id, int film_id)	Preleva (se esiste) il promemoria salvato dall'utente fornito verso il film fornito.

```
context IPromemoriaDAO::findByUtente(utente_id:int) post:
    result = (tutti i promemoria p nel DB : p.utente = utente_id)


context IPromemoriaDAO::exists(utente_id:int, film_id:int) post:
    result = (self.findByUtenteAndFilm(utente_id, film_id) <> null)


context IPromemoriaDAO::create(promemoria:Promemoria) pre:
    promemoria <> null and not self.exists(promemoria.utente, promemoria.film)
context IPromemoriaDAO::create(promemoria:Promemoria) post:
    result = promemoria and self.findByUtenteAndFilm(result.utente, result.film) = result


context IPromemoriaDAO::delete(promemoria:Promemoria) pre:
    promemoria <> null and self.exists(promemoria.utente, promemoria.film)
context IPromemoriaDAO::delete(promemoria:Promemoria) post:
    result = true and not self.exists(promemoria.utente, promemoria.film)


context IPromemoriaDAO::findByUtenteAndFilm(utente_id:int, film_id:int) post:
    result = (promemoria p nel DB : p.utente = utente_id and p.film = film.id)
```

IRecitazioneDAO

Metodo	Descrizione
Recitazione[] findByArtista(int artista_id)	Preleva tutte le recitazioni espresse da un artista fornito.
Recitazione[] findByFilm(int film_id)	Preleva tutte le recitazioni nell'ambito di un film fornito.
bool setOnly(int film_id, Recitazione[] recitazioni)	Associa ad un film solamente le recitazioni fornite.

```
context IRecitazioneDAO::findByArtista(artista_id:int) post:
    result = (tutte le recitazioni r nel DB : r.attore = artista_id)


context IRecitazioneDAO::findByFilm(film_id:int) post:
    result = (tutte le recitazioni r nel DB : r.film = film_id)
```

```
context IRecitazioneDAO::setOnly(film_id:int, recitazioni:Recitazione[]) pre:
    recitazioni <> null
context IRecitazioneDAO::setOnly(film_id:int, recitazioni:Recitazione[]) post:
    result = true and self.findByFilm(film_id) = recitazioni
```

IRegiaDAO

Metodo	Descrizione
int[] findFilmsByArtista(int id)	Preleva i film di cui un artista fornito ha partecipato alla regia.
int[] findArtistiByFilm(int id)	Preleva gli artisti che hanno partecipato alla regia di un film fornito.
bool setOnly(int film_id, int[] registi_id)	Associa ad un film solamente i registi forniti.

```
context IRegiaDAO::findFilmsByArtista(id:int) post:
    result = (tutte le regie r.film nel DB : r.regista = id)


context IRegiaDAO::findArtistiByFilm(id:int) post:
    result = (tutte le regie r.regista nel DB : r.film = id)


context IRegiaDAO::setOnly(film_id:int, registi_id:int[]) pre:
    registi_id <> null
context IRegiaDAO::setOnly(film_id:int, registi_id:int[]) post:
    result = true and self.findArtistiByFilm(film_id) = registi_id
```

Implementations

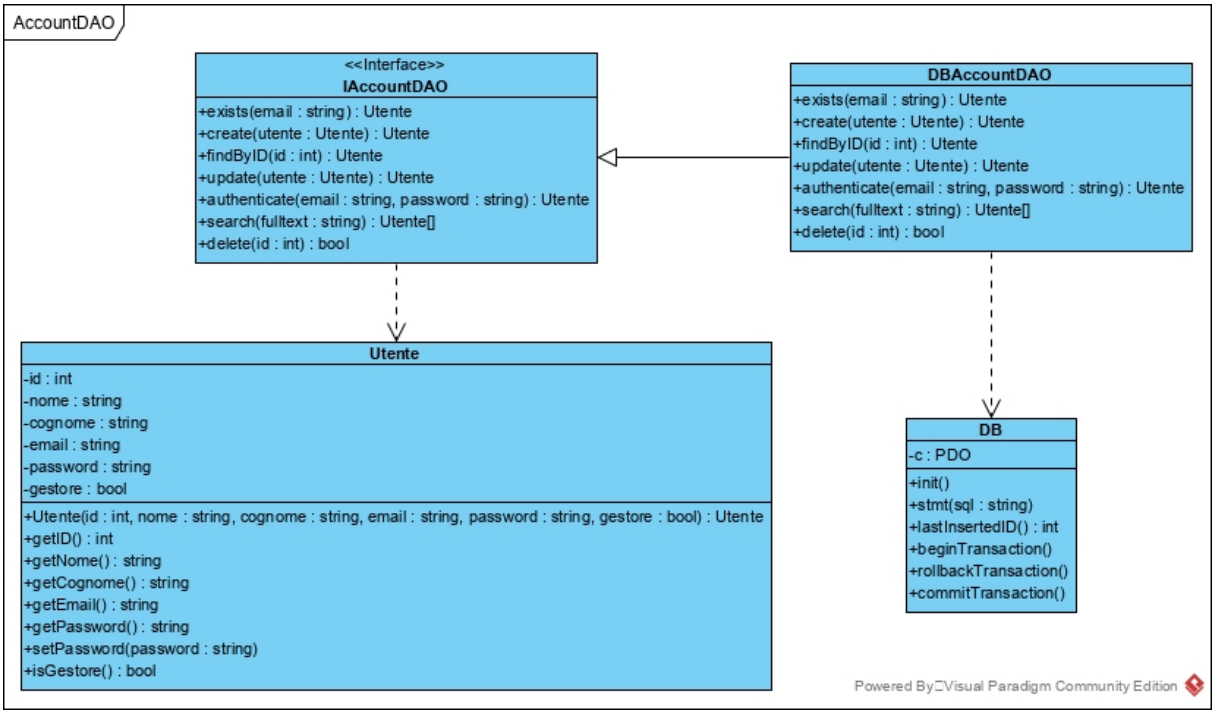
Il package **dao/implementations** contiene classi che implementano le interfacce DAO e realizzano le operazioni imposte di gestione della persistenza utilizzando la base di dati.

Queste classi sono:

- DBAccountDAO;
- DBAmiciziaDAO;
- DBArtistaDAO;
- DBFilmDAO;
- DBGenereDAO;
- DBGiudizioDAO;
- DBPromemoriaDAO;
- DBRecitazioneDAO;
- DBRegiaDAO.

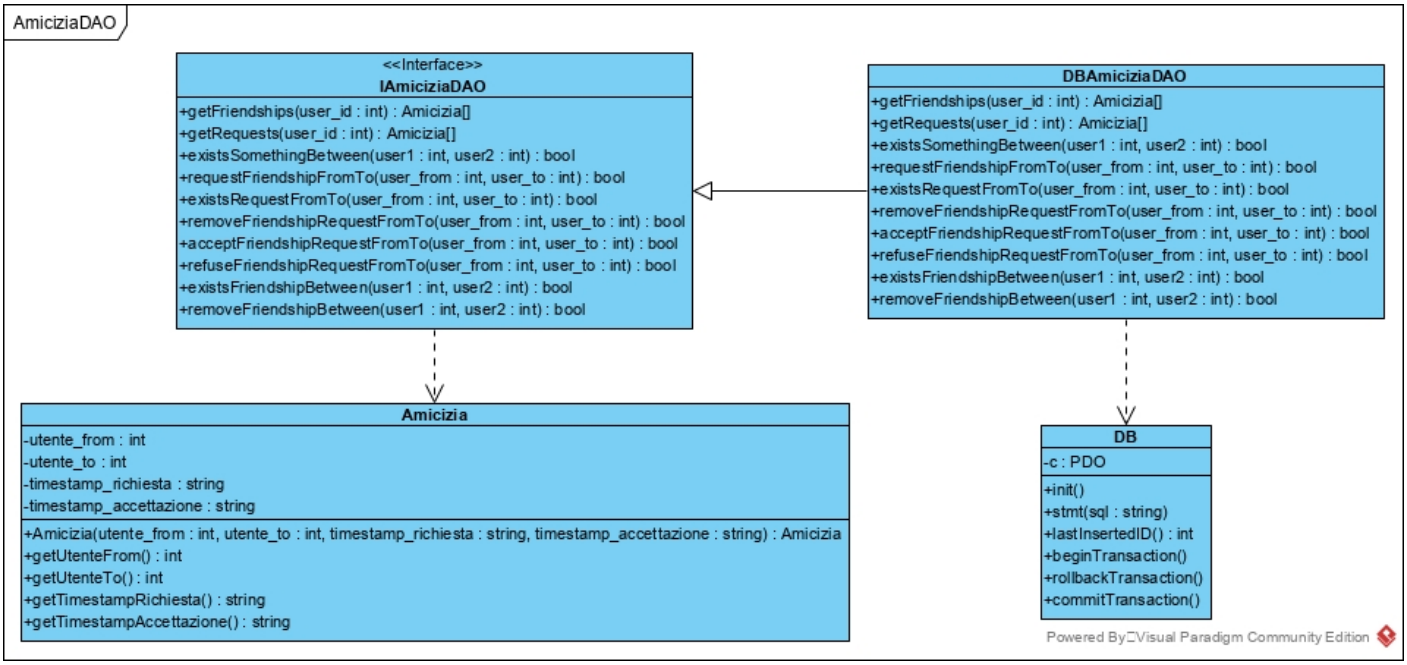
Seguono i class diagram che mostrano le relazioni tra le implementazioni di queste interfacce DAO e i relativi modelli.

Account DAO



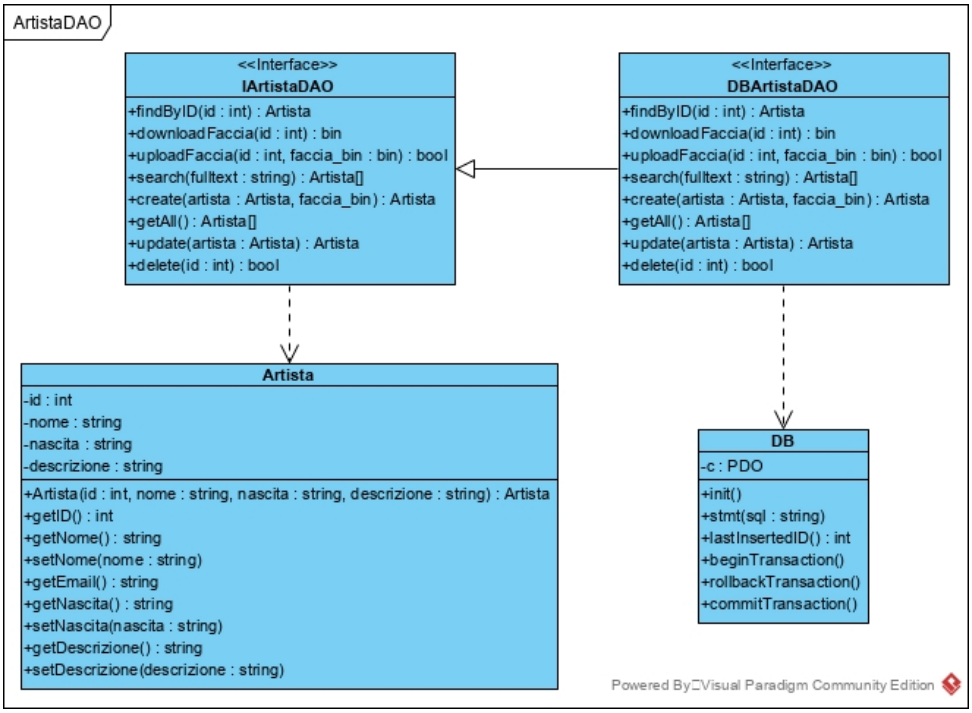
Queste sono le relazioni tra **IAccountDAO**, **DBAccountDAO** e **Utente**.

Amicizia DAO



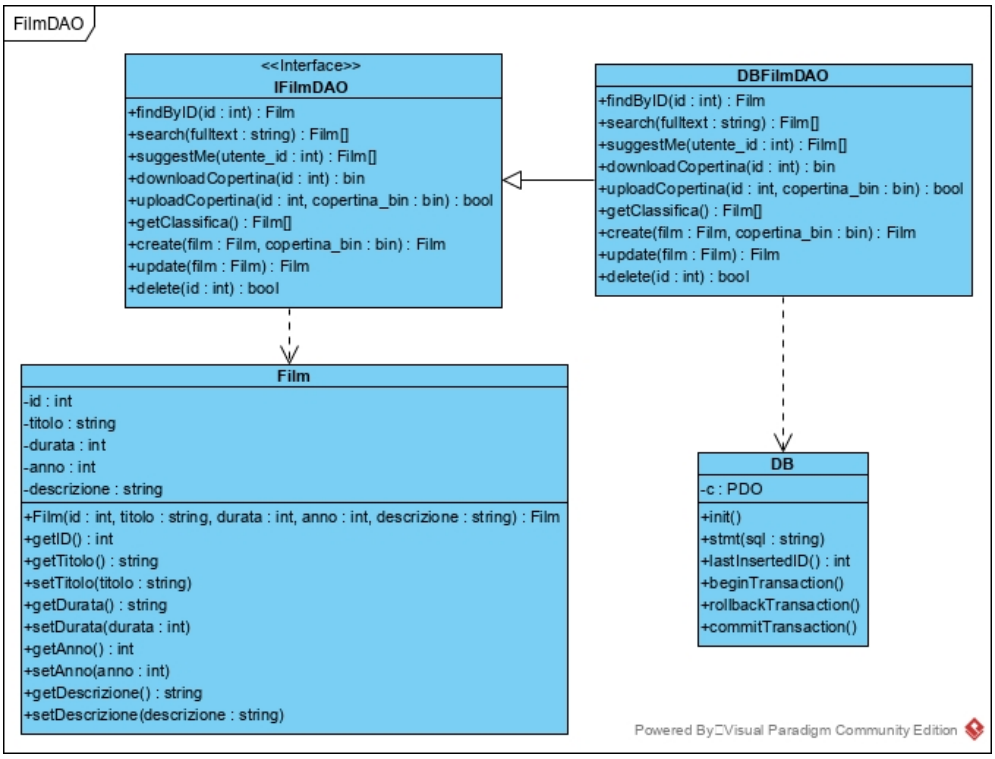
Queste sono le relazioni tra **IAmiciziaDAO**, **DBAmiciziaDAO** e **Amicizia**.

Artista DAO



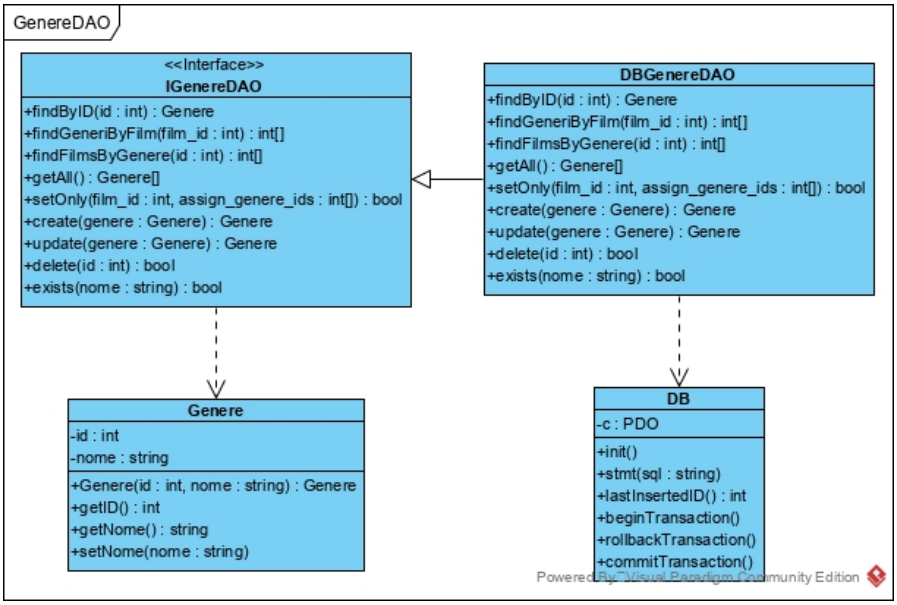
Queste sono le relazioni tra **IArtistaDAO**, **DBArtistaDAO** e **Artista**.

Film DAO



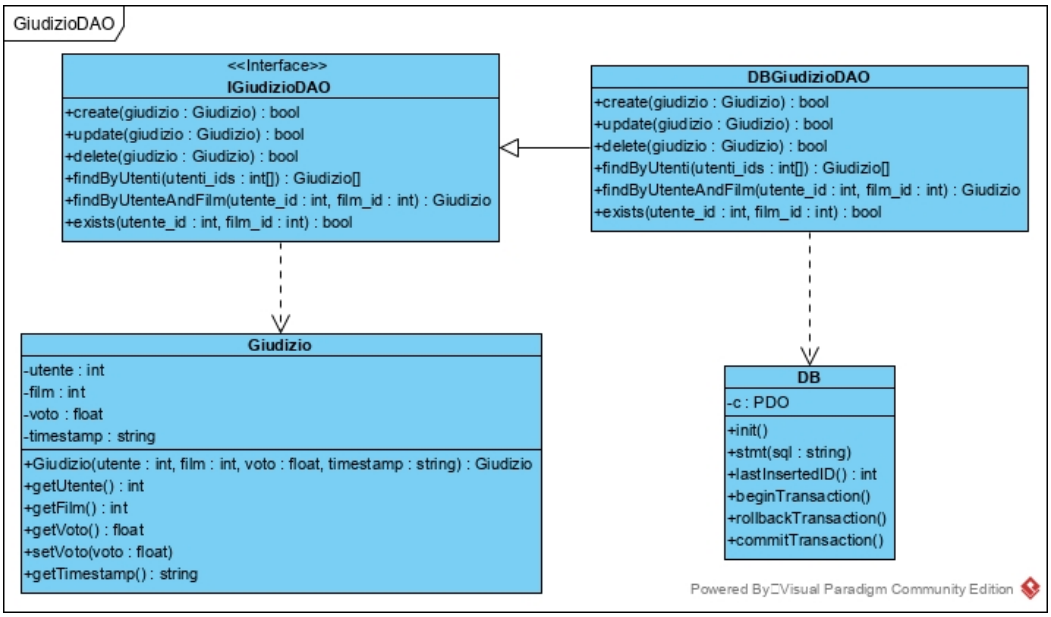
Queste sono le relazioni tra **IFilmDAO**, **DBFilmDAO** e **Film**.

Genere DAO



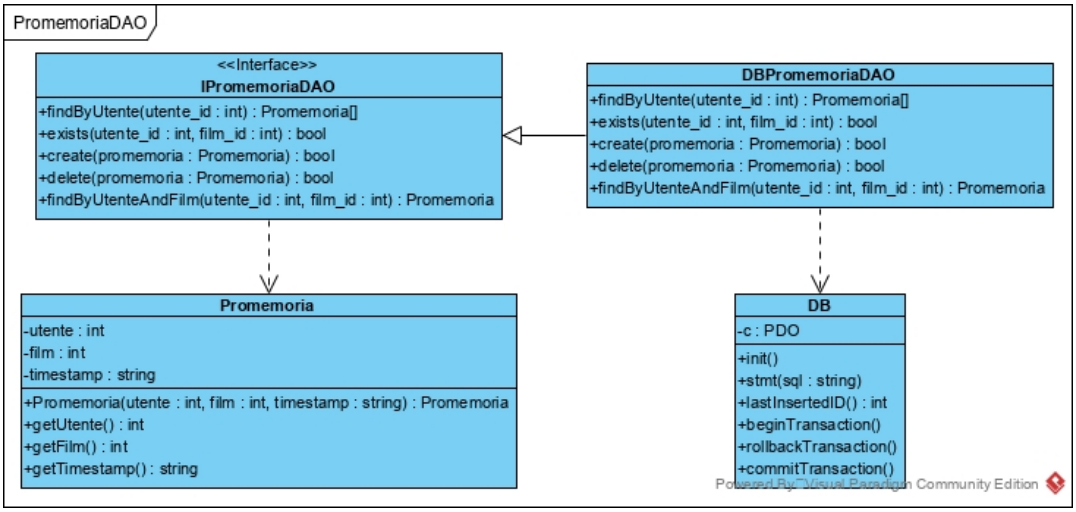
Queste sono le relazioni tra **IGenereDAO**, **DBGenereDAO** e **Genere**.

Giudizio DAO



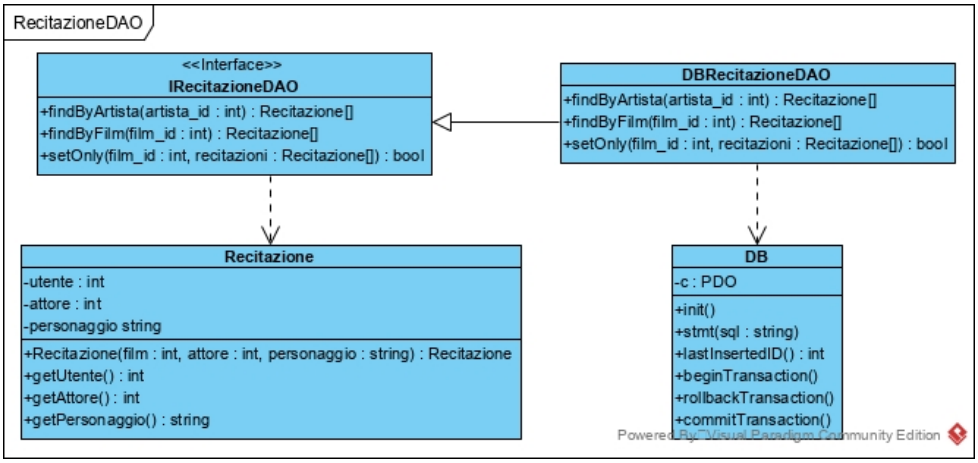
Queste sono le relazioni tra **IGiudizioDAO**, **DBGiudizioDAO** e **Giudizio**.

Promemoria DAO



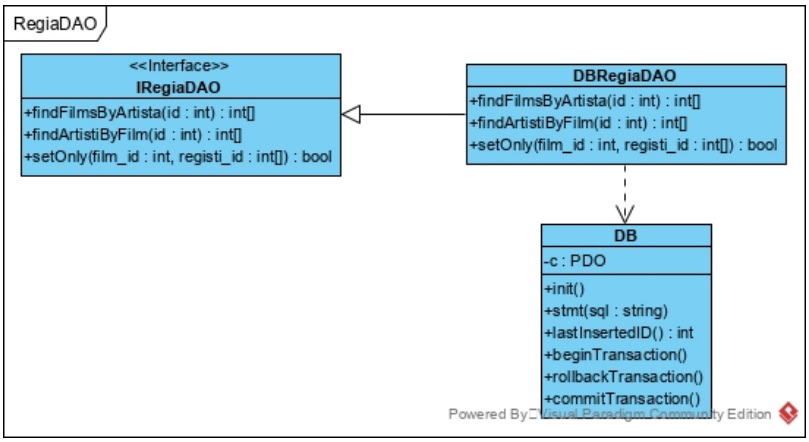
Queste sono le relazioni tra **IPromemoriaDAO**, **DBPromemoriaDAO** e **Promemoria**.

Recitazione DAO



Queste sono le relazioni tra **IRecitazioneDAO**, **DBRecitazioneDAO** e **Recitazione**.

Regia DAO



Queste sono le relazioni tra **IRegiaDAO** e **DBRegiaDAO**.

Factories

Il package **dao/factories** contiene delle classi factory che implementano le relative interfacce DAO.

Queste sono:

- AccountDAOFactory;
- AmiciziaDAOFactory;
- ArtistaDAOFactory;
- FilmDAOFactory;
- GenereDAOFactory;
- GiudizioDAOFactory;
- PromemoriaDAOFactory;
- RecitazioneDAOFactory;
- RegiaDAOFactory.

Ognuna di queste factory ha il compito di fornire una implementazione della relativa interfaccia DAO (che sono descritte sopra) nascondendo all'utilizzatore la reale locazione dei dati persistenti.

Attualmente, i dati persistenti sono memorizzati in una base di dati relazionale (MySQL). Se si avesse in futuro la necessità di migrare verso un altro tipo di DBMS, oppure di memorizzare direttamente su file secondo uno specifico formato (non strutturato oppure basato su XML, JSON...), oppure ancora utilizzando le sessioni o la RAM, si potrebbe far fronte a questa necessità con estrema facilità. Bisognerebbe banalmente implementare le interfacce DAO usando le modalità di persistenza candidate.

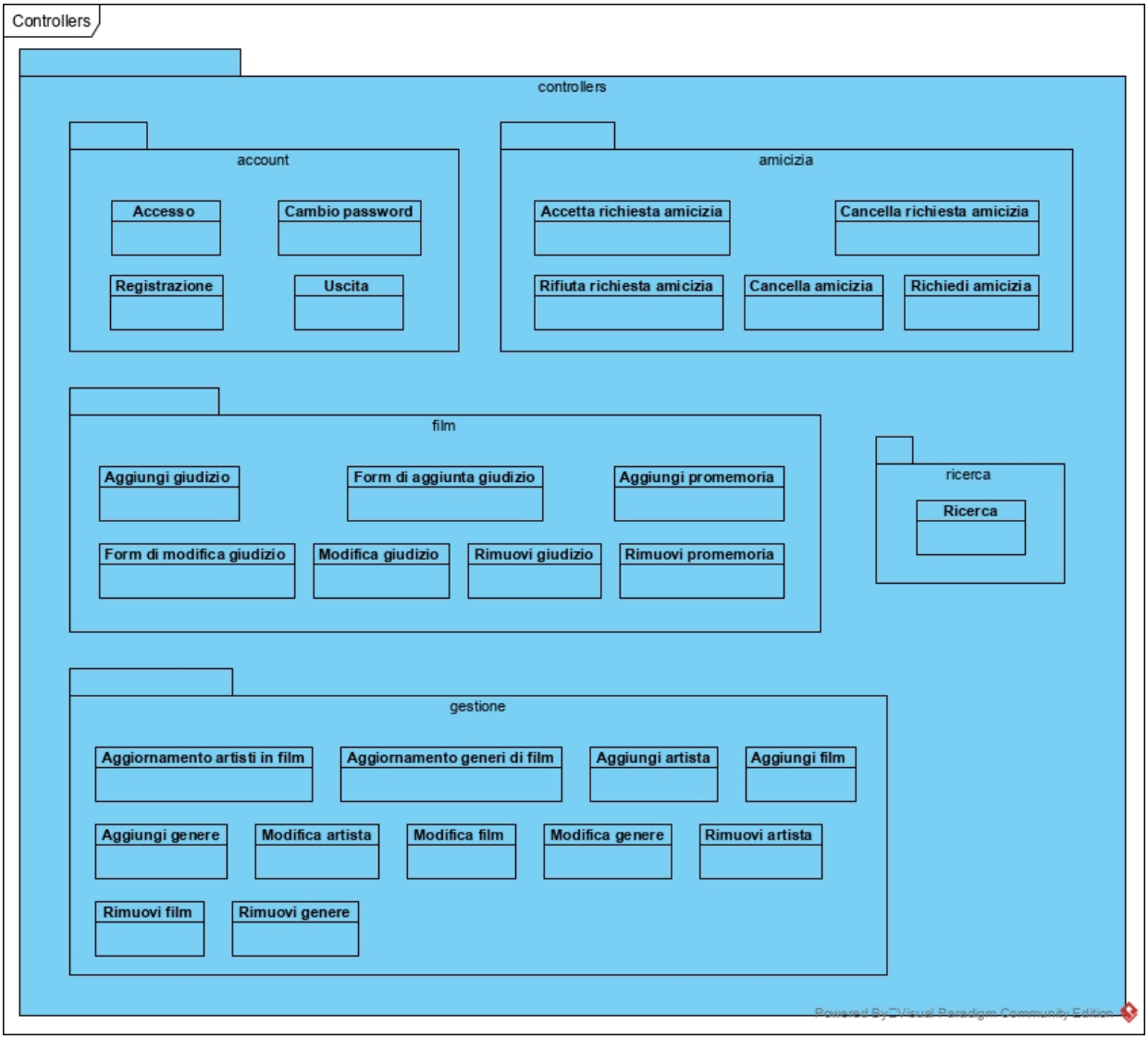
In realtà, la necessità di passare ad un altro tipo di memorizzazione (temporaneamente) è già sorta durante le fasi di sviluppo del progetto: in particolare, durante la fase di testing, le factory restituiscono delle implementazioni STUB delle interfacce DAO.

Package Controllers

La struttura completa del package **controllers** è questa.

```
controllers
│
├──account
│       Accesso.php
│       Cambio password.php
│       Registrazione.php
│       Uscita.php
│
├──amicizia
│       Accetta richiesta amicizia.php
│       Cancella amicizia.php
│       Cancella richiesta amicizia.php
│       Richiedi amicizia.php
│       Rifiuta richiesta amicizia.php
│
├──film
│       Aggiungi giudizio.php
│       Aggiungi promemoria.php
│       Form di aggiunta giudizio.php
│       Form di modifica giudizio.php
│       Modifica giudizio.php
│       Rimuovi giudizio.php
│       Rimuovi promemoria.php
│
├──gestione
│       Aggiornamento artisti in film.php
│       Aggiornamento generi di film.php
│       Aggiungi artista.php
│       Aggiungi film.php
│       Aggiungi genere.php
│       Modifica artista.php
│       Modifica film.php
│       Modifica genere.php
│       Rimuovi artista.php
│       Rimuovi film.php
│       Rimuovi genere.php
│
```

```
└──ricerca
       Ricerca.php
```



Account

Controller	Descrizione
Accesso	Permette la autenticazione.
Cambio password	Permette di cambiare password.
Registrazione	Permette di creare un nuovo account.
Uscita	Permette la deautenticazione.

Amicizia

Controller	Descrizione
Accetta richiesta amicizia	Permette di accettare una richiesta di amicizia inviata da un utente verso l'utilizzatore.
Cancella amicizia	Permette di cancellare una amicizia stabilita tra l'utilizzatore ed un altro utente.

Controller	Descrizione
Cancella richiesta amicizia	Permette di cancellare una richiesta di amicizia precedentemente inviata dall'utilizzatore verso un altro utente.
Richiedi amicizia	Permette di inviare una richiesta di amicizia a nome dell'utilizzatore verso un altro utente.
Rifiuta richiesta amicizia	Permette di rifiutare una richiesta di amicizia inviata da un altro utente verso l'utilizzatore.

Film

Controller	Descrizione
Aggiungi giudizio	Permette di aggiungere un giudizio dell'utilizzatore su un film.
Aggiungi promemoria	Permette di aggiungere un promemoria su un film.
Form di aggiunta giudizio	Permette di inserire il voto di un giudizio da aggiungere.
Form di modifica giudizio	Permette di modificare il voto di un giudizio esistente.
Modifica giudizio	Permette di modificare un giudizio creato dall'utilizzatore.
Rimuovi giudizio	Permette di rimuovere un giudizio creato dall'utilizzatore.
Rimuovi promemoria	Permette di rimuovere un promemoria di film dell'utilizzatore.

Gestione

Controller	Descrizione
Aggiornamento artisti in film	Permette al gestore di aggiornare le recitazioni e regie di un film.
Aggiornamento generi di film	Permette al gestore di aggiornare i generi di un film.
Aggiungi artista	Permette al gestore di aggiungere un artista.
Aggiungi film	Permette al gestore di aggiungere un film.
Aggiungi genere	Permette al gestore di aggiungere un genere.
Modifica artista	Permette al gestore di modificare un artista.
Modifica film	Permette al gestore di modificare un film.
Modifica genere	Permette al gestore di modificare un genere.
Rimuovi artista	Permette al gestore di rimuovere un artista.
Rimuovi film	Permette al gestore di rimuovere un film.
Rimuovi genere	Permette al gestore di rimuovere un genere.

Ricerca

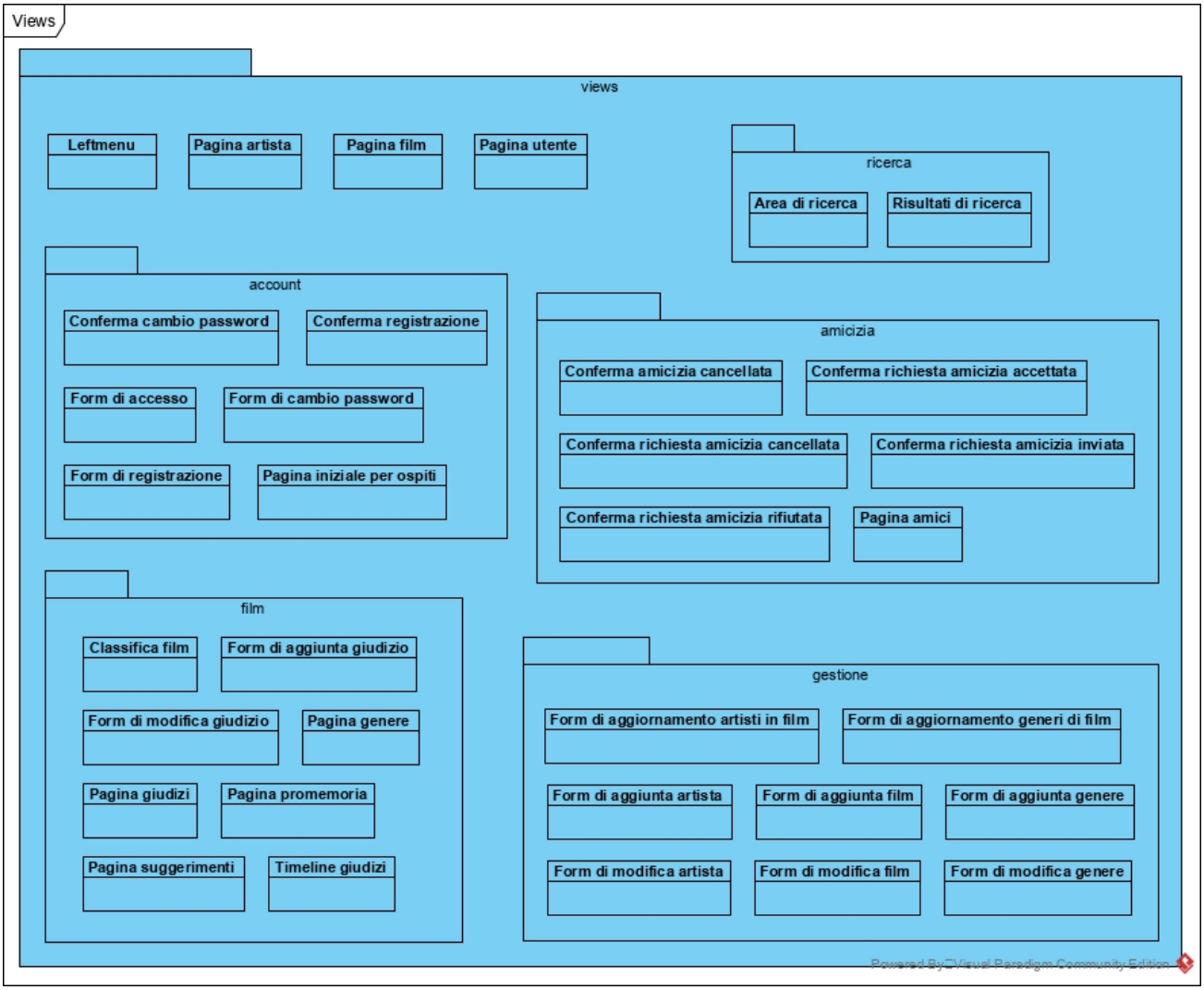
Controller	Descrizione

Controller	Descrizione
Ricerca	Permette di effettuare una ricerca di artisti, film e/o utenti.

Package Views

Questa è la struttura del package **views** del sistema.

```
views
│   Leftmenu.php
│   Pagina artista.php
│   Pagina film.php
│   Pagina utente.php
│
├──account
│       Conferma cambio password.php
│       Conferma registrazione.php
│       Form di accesso.php
│       Form di cambio password.php
│       Form di registrazione.php
│       Pagina iniziale per ospiti.php
│
├──amicizia
│       Conferma amicizia cancellata.php
│       Conferma richiesta amicizia accettata.php
│       Conferma richiesta amicizia cancellata.php
│       Conferma richiesta amicizia inviata.php
│       Conferma richiesta amicizia rifiutata.php
│       Pagina amici.php
│
├──film
│       Classifica film.php
│       Form di aggiunta giudizio.php
│       Form di modifica giudizio.php
│       Pagina genere.php
│       Pagina giudizi.php
│       Pagina promemoria.php
│       Pagina suggerimenti.php
│       Timeline giudizi.php
│
├──gestione
│       Form di aggiornamento artisti in film.php
│       Form di aggiornamento generi di film.php
│       Form di aggiunta artista.php
│       Form di aggiunta film.php
│       Form di aggiunta genere.php
│       Form di modifica artista.php
│       Form di modifica film.php
│       Form di modifica genere.php
│
└──ricerca
        Area di ricerca.php
        Risultati di ricerca.php
```



View generiche

Queste viste non sono state ulteriormente raggruppate non fanno parte di un solo ambito particolare.

View	Descrizione
Leftmenu	Compone la barra laterale del sito web.
Pagina artista	Presenta le informazioni di un artista ed i film a cui ha partecipato.
Pagina film	Presenta le informazioni di un film, le opzioni di giudizi e promemoria, e gli artisti che vi hanno partecipato.
Pagina utente	Presenta le informazioni di un utente, le opzioni di amicizia e (se il visitatore è amico) i suoi giudizi.

View di account

Queste sono le view del package **views/account**.

View	Descrizione
Conferma cambio password	Conferma il cambio password.
Conferma registrazione	Conferma l'avvenuta registrazione.

View	Descrizione
Form di accesso	Mostra il form di accesso.
Form di cambio password	Mostra il form di cambio password.
Form di registrazione	Mostra il form di registrazione.
Pagina iniziale per ospiti	Invita l'utente ad effettuare l'accesso o la registrazione.

View di amicizia

Queste sono le view del package **views/amicizia**.

View	Descrizione
Conferma amicizia cancellata	Conferma la rimozione di un'amicizia.
Conferma richiesta amicizia accettata	Conferma l'accettazione di una richiesta di amicizia.
Conferma richiesta amicizia cancellata	Conferma la rimozione di una richiesta di amicizia.
Conferma richiesta amicizia inviata	Conferma l'invio di una richiesta di amicizia.
Conferma richiesta amicizia rifiutata	Conferma il rifiuto di una richiesta di amicizia.
Pagina amici	Mostra gli amici dell'utente autenticato e le richieste ancora in atto.

View di film

Queste sono le view del package **views/film**.

View	Descrizione
Classifica film	Mostra i 50 film meglio votati dalla community degli utenti.
Form di aggiunta giudizio	Permette di selezionare un voto da 1 a 10 da assegnare a un film creando un giudizio.
Form di modifica giudizio	Permette di selezionare un voto da 1 a 10 da assegnare a un film modificando un preesistente giudizio.
Pagina genere	Mostra tutti i film di un genere.
Pagina giudizi	Mostra tutti i giudizi espressi dall'utente autenticato e permette di modificarli e cancellarli.
Pagina promemoria	Mostra tutti i promemoria salvati dall'utente autenticato e permette di cancellarli.
Pagina suggerimenti	Mostra 5 film in linea con le preferenze cinematografiche dell'utente autenticato.
Timeline giudizi	Mostra tutti i giudizi in ordine cronologico che un determinato utente ha creato.

View di gestione

Queste sono le view del package **views/gestione**.

View	Descrizione
Form di aggiornamento artisti in film	Permette di riassegnare gli artisti che hanno partecipato (come registi o attori) in un film.
Form di aggiornamento generi di film	Permette di riassegnare i generi a un film.
Form di aggiunta artista	Permette di creare un nuovo artista.
Form di aggiunta film	Permette di creare un nuovo film.
Form di aggiunta genere	Permette di creare un nuovo genere.
Form di modifica artista	Permette di modificare un artista.
Form di modifica film	Permette di modificare un film.
Form di modifica genere	Permette di modificare un genere.

View di ricerca

Queste sono le view del package **views/ricerca**.

View	Descrizione
Area di ricerca	Permette di effettuare una ricerca di artisti, film e utenti.
Risultati di ricerca	Presenta i risultati di ricerca.

Design pattern

Factory

Il Design Pattern Factory fornisce l'interfaccia per la creazione di un oggetto e fa fronte al problema della creazione di oggetti senza specificarne l'esatta classe da istanziare.

Le classi factory del sistema forniscono implementazioni delle interfacce DAO per la gestione dei dati persistenti. Per esempio, **FilmDAOFactory** è una classe che, attraverso il metodo **getFilmDAO**, fornisce una implementazione della interfaccia **IFilmDAO**. L'implementazione maggiormente utilizzata dal sistema è **DBFilmDAO**.

Utilizzando le factory, ogni parte di codice in cui è richiesta un'implementazione di **IFilmDAO** eviterà di contenere il codice necessario alla creazione di implementazioni dell'interfaccia, riducendo al minimo le duplicazioni di codice.

Inoltre, se in futuro si volesse cambiare l'implementazione da utilizzare, passare per esempio da **DBFilmDAO** a **XMLFilmDAO**, bisognerebbe semplicemente modificare il suddetto metodo, evitando lo sforzo di un grande refactoring su tutto il codice che istanziava oggetti **DBFilmDAO**.

Singleton

Il Design Pattern Singleton viene usato per garantire che, di una determinata classe, venga usata una sola istanza, oltre a fornire il punto di accesso globale a tale istanza.

Nel sistema, in alcune circostanze, **ArtistaDAOFactory** si comporta come un singleton di **StubArtistaDAO**. Il metodo **getArtistaDAO** normalmente restituisce una nuova istanza della classe **DBArtistaDAO**. Se prima, però, viene invocato il metodo **useStub**, questo inizializzerà una istanza di **StubArtistaDAO**, la memorizzerà e la farà restituire ad ogni invocazione del metodo **getArtistaDAO**, in sostituzione delle istanze di **DBArtistaDAO**.

Model View Controller

Il pattern architetturale Model View Controller (MVC) viene usato per separare la logica di presentazione dei dati dalla logica di business.

Le componenti dell'MVC interpretano tre ruoli principali:

- I **model** forniscono metodi per accedere ai dati;
- Le **view** visualizzano i dati contenuti nei model e si occupano dell'interazione con utente e agenti;
- I **controller** ricevono i comandi dell'utente e li attuano modificando lo stato delle altre due componenti.

Un esempio di model è **Utente**, di view è **Form di registrazione**, e di controller è **Registrazione**.

Data Access Object

Il pattern architetturale Data Access Object (DAO) è un modo di gestione della persistenza. Viene usato per ottenere una stratificazione e rigida separazione tra le componenti che "usano" lo storage e quelle che realizzano le modalità di interazione con lo storage.

Per esempio, **IGenereDAO** realizza le operazioni CRUD della tabella 'generi' nella base di dati, e presenta questi dati tramite il modello **Genere**.