



PROGETTO OOP

A.A. 2015-2016



19 GIUGNO 2016

24CNL

Riccardo Fortini – Francesco Paltera

REQUIREMENTS COLLECTION:

Requisiti funzionali

(Verrà assegnata priorità attraverso tre classificazioni: HP(High priority), MP(Medium priority), LP(Low priority)

- **Login (HP):** Funzione necessaria per l'accesso al sistema, senza di essa l'utente non sarà in grado di interfacciarsi con esso.
- **Gestione sistema (MP):** Consente all'amministratore di sistema di gestire i dati relativi ad opere e ad utenti, cancellarli, modificarli oppure convalidarli in vista di una pubblicazione.
- **Visualizzazione elenco opere (HP):** Funzione di browsing base, consente all'utente di visualizzare l'elenco dei titoli delle opere nel sistema. A questo ed ad alcuni dei prossimi requisiti verrà assegnata alta priorità, dato che rappresentano il cuore del sistema, ed inglobano le funzioni base di esso.
- **Visualizzazione intere opere (HP):** Funzione per utenti "premium", consente di consultare ambo immagini e testo di tutte le opere pubblicate.
- **Acquisizione (HP):** Consente agli acquirenti di caricare scansioni dei manoscritti nel sistema.
- **Validazione Scansione (MP):** Consente ai revisori delle acquisizioni di decretare se una scansione è idonea all'upload.
- **Trascrizione (HP):** Una delle operazioni più importanti del sistema, accedervi deve essere semplice per gli utenti abilitati. Essa consiste

nel trascrivere, appunto, tramite un editor di testo le scansioni relative ad un'opera selezionata.

- **ValidazioneTrascrizione (MP):** Consente ai revisori della trascrizione di decidere se accettare o meno la trascrizione di un manoscritto.
- **Pubblicazione (HP):** Consente (dopo aver superato la fase di validazione della scansione) ai revisori delle acquisizioni, oppure all'amministratore di sistema, di pubblicare il manoscritto selezionato.

Requisiti non funzionali:

- **Safety:** Il committente non ha richiesto vincoli di safety, pertanto il sistema viene considerato non safety-critical. La safety sarà pertanto misurata solamente in base alla sicurezza dell'account di ogni utente.
- **Usability:** In base all'utenza che potrebbe usufruire del sistema, abbiamo deciso di propendere per un'interfaccia grafica estremamente user-friendly.
- **Performance:** Il sistema maneggia dati (soprattutto le immagini) molto pesanti, pertanto la performance potrebbe variare a seconda delle prestazioni degli end-system e dei collegamenti con le risorse.
- **Operational:** Andrà garantita coerenza nella sequenzialità dei dati relativi ad un'opera, nonché la corrispondenza tra opere ed indicizzazione.

Assunzioni:

- Abbiamo assunto che i collaboratori (Trascrittori, revisori e acquisitori) abbiano accesso ai privilegi di un utente "premium" oltre che a quelli specifici del loro settore.
- Abbiamo assunto che la funzione di upload delle immagini nel web server si potrà effettuare dalla GUI tramite un apposito pulsante che permetterà di scegliere il file dal proprio terminale.
- Abbiamo assunto, a seguito dell'analisi della specifica, che i manoscritti devono essere stati digitalizzati per intero prima della messa in pubblicazione.

Attori e use case per il nostro sistema:

Verranno elencati gli attori in ordine gerarchico (quelli più in basso ereditano le funzioni di quello più in alto, oltre ad aggiungere le proprie)

1. **Utente base:** Partecipa a use case Login e visualizzaElencoOpere, che consentono, rispettivamente, di accedere al sistema e di visualizzare l'elenco delle opere pubblicate.
2. **Utente Premium:** Può, in aggiunta alle funzioni dell'utente base, visualizzare ogni opera per intero.
3. **Acquisitore:** Può, in aggiunta alle funzioni dell'utente premium, utilizzare la funzione upload.

Trascrittore: Può, in aggiunta alle funzioni dell'utente premium, utilizzare la funzione trascrizione con il relativo editor di testo.

RevisoreAcquisizioni: Può, in addizione alle funzioni dell'utente premium, eseguire il check sulle immagini scannerizzate (use case Convalida).

RevisoreTrascrizioni: Può, in addizione alle funzioni dell'utente premium, eseguire il check sulle trascrizioni dei manoscritti (use case Revisione)

4. **Admin:** Può, in addizione alle funzioni di tutti gli utenti sopra citati, accedere alle funzioni di gestione utenti e gestione sistema.

Specifica use case ad alta priorità

Nome Use Case	Login
Attori coinvolti	Utente Base
Evento scatenante	Click sul pulsante nella main form della GUI
Funzione	Permette ad un utente non registrato di procedere con il signup, mentre ad un utente già registrato, di entrare nel sistema con i propri dati.

Nome Use Case	visualizzaElencoOpere
Attori coinvolti	Utente Base
Evento scatenante	Click sul relativo pulsante nella GUI
Funzione	Funzione base che consente di visualizzare la lista dei titoli dei testi presenti nel sistema.

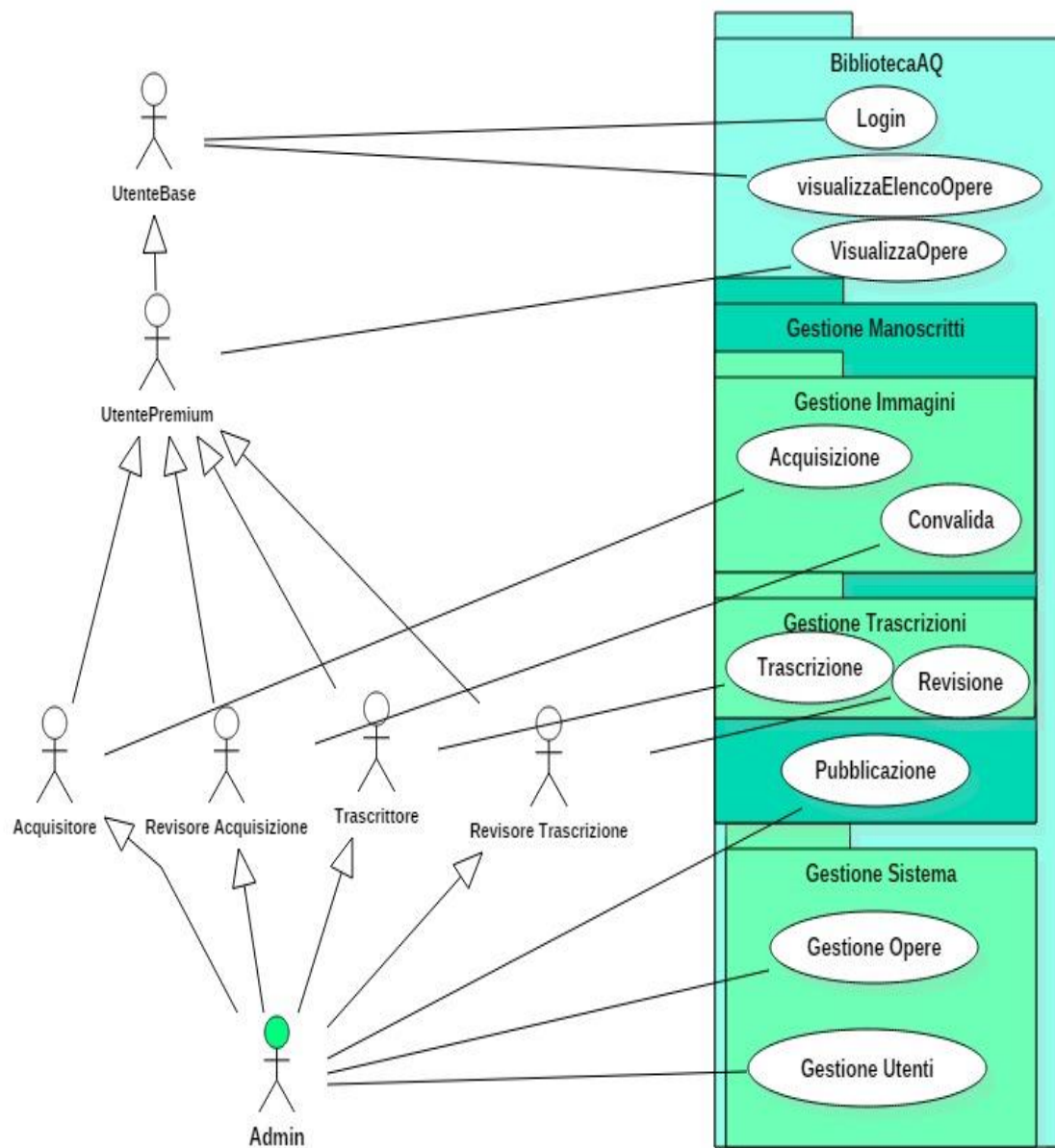
Nome Use Case	VisualizzaOpere
Attori coinvolti	Utente Premium
Evento scatenante	Click sul titolo di una specifica opera
Funzione	Consente di visualizzare testo e immagini presenti nel sistema relativi a quello specifico manoscritto.

Nome Use Case	Acquisizione
Attori coinvolti	Acquisitore
Evento scatenante	Upload di un file immagine nel sistema
Funzione	Consente agli addetti all'upload delle scansioni di caricare una pagina nel sistema.

Nome Use Case	Trascrizione
Attori coinvolti	Trascrittore
Evento scatenante	Inserimento di testo nell'editor TEI aperto tramite la shortcut nella GUI
Funzione	Consente agli utenti idonei di accedere alle funzioni di scrittura.

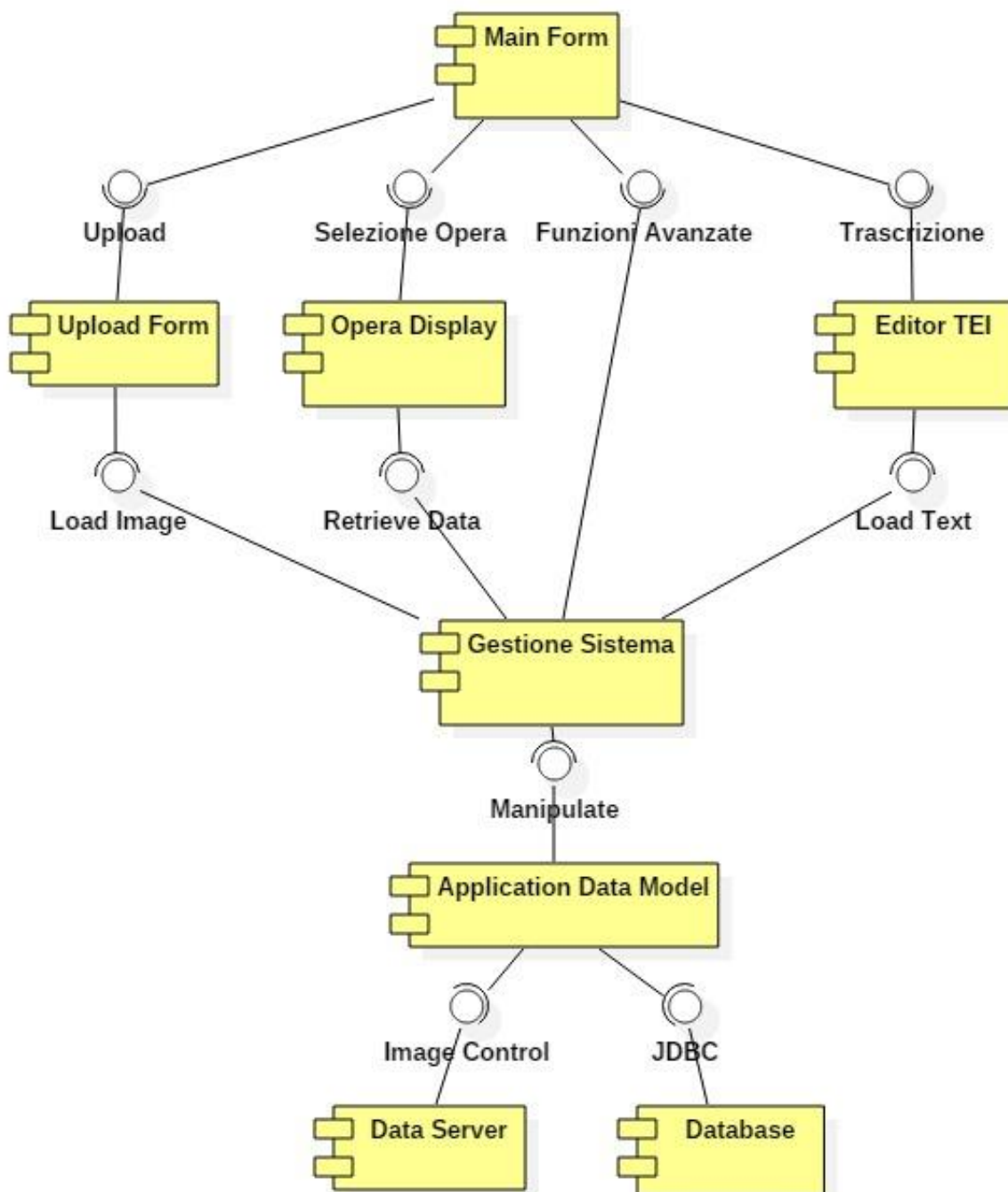
Nome Use Case	Pubblicazione
Attori coinvolti	Admin
Evento scatenante	Utilizzo del pannello di pubblicazione
Funzione	Consente di rendere un'opera visibile all'utenza

Use case diagram per il nostro sistema:



SYSTEM DESIGN

Modello architetturale del sistema.



Descrizione del nostro modello architetturale:

Abbiamo optato per un'applicazione in locale che interagisce con un server ed un database. La scelta è stata effettuata riflettendo sugli scenari di utilizzo del sistema, nello specifico, al tipo di utenza che l'applicazione potrebbe avere. Per questo la nostra GUI sarà user-friendly e veloce da utilizzare, permettendo all'utente di potersi concentrare solamente su quello che è lo scopo per cui sta usando il software.

Abbiamo illustrato, nel component diagram sopra, le parti che, per questo livello di astrazione, riteniamo importanti nel sistema.

Le prime quattro componenti fanno riferimento alla “view” (spiegheremo in seguito in dettaglio a cosa facciamo riferimento) del nostro sistema. Queste sono: la “main form”, la “upload form”, l'editor TEI ed “opera display”. Fanno tutte riferimento ad aree della GUI, ed ognuna consente di accedere ad una delle funzioni critiche indicate prima nella descrizione degli use case.

La componente “Gestione Sistema” è quella che si identifica con i “controller” (ne discuteremo sempre più avanti parlando dei Design Pattern) e consente di interfacciarsi con il livello di “application data model”, ossia il database ed il server.

Scelte implementative per il nostro sistema.

Editor TEI

Tra le scelte nelle quali il team ha riscontrato più difficoltà nel risolvere, c'è sicuramente quella relativa all'editor di testo. Gli editor TEI sono sicuramente una risorsa costosa (se si vuole arrivare a prodotti di un certo livello, ad esempio Oxygen) o rara, nel caso in cui si volesse trovare un software Open Source che riconosca e "convalidi" il giusto numero di tag (inoltre, risulta difficile trovare software open source con visual feedback oltre a text editing). Ci siamo imbattuti, dopo svariati tentativi tra editor più o meno recenti, in JEdit. Interamente java-based ed in grado di riconoscere e validare TEI attraverso l'utilizzo dei Plug-In XML , ErrorList ed XLST per trasformare documenti XML usando le stylesheets:

Per schemi diversi da RNG va aggiunto un file schemas.xml nella directory locale oppure installarlo globalmente nella cartella dei plugin XML aggiungendo queste 4 righe di codice (scelta consigliata):

```
<?xml version="1.0" ?>
<locatingRules xmlns="http://thaiopensource.com/ns/locating-rules/1.0">
  <namespace ns="http://www.tei-c.org/ns/1.0" uri="tei_all.rng"/>
</locatingRules>
```

Attraverso il plugin XML si possono anche generare DTD.

Visualizzazione TEI

Per visualizzare correttamente il TEI in un JFrame abbiamo dovuto affrontare non poche problematiche.

Il primo problema riguardava la memorizzazione del file nel DB poiché come scelta implementativa abbiamo deciso di utilizzare il web storage esclusivamente per le immagini e non per i file XML. Per evitare errori nelle query di inserimento e lettura del file a causa dei tag, abbiamo optato per codificare in base 64 il contenuto della trascrizione:

Scrittura) File XML -> FileReader -> Encode base64 -> PreparedStatement

Letture) Statement -> Decode base64

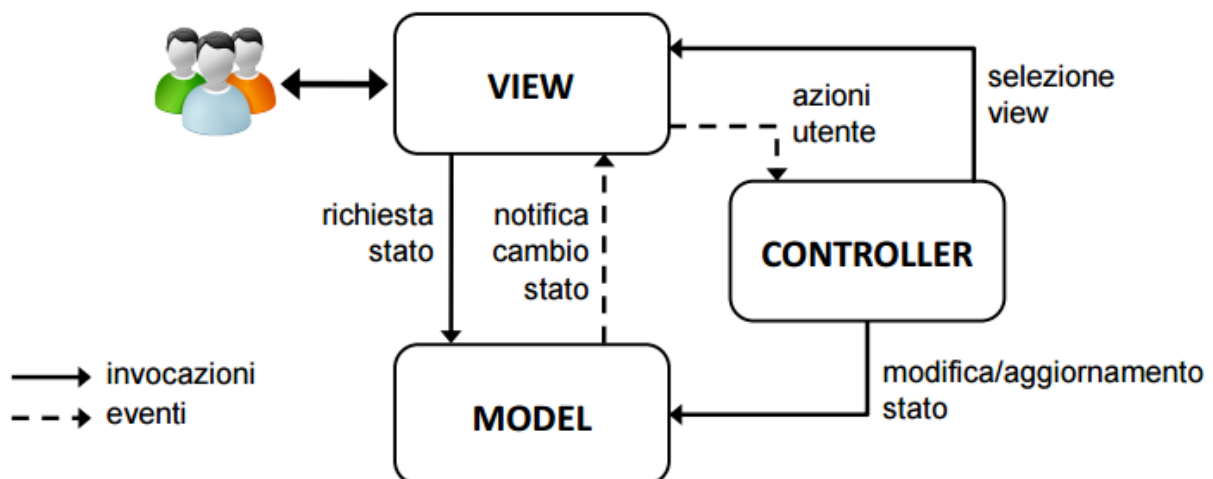
Il secondo problema, sicuramente il più complesso, riguardava il convertire il TEI in un formato ideale per la sua corretta visualizzazione. Siccome il plain text a nostro parere non era adatto per un'applicazione in cui la visualizzazione delle trascrizioni è fondamentale, abbiamo scelto l'HTML. Per renderlo possibile abbiamo utilizzato le classi Transformer base con l'aggiunta delle librerie Saxon in combinazione con gli StyleSheet XSLT per l'HTML.

L'ultimo problema, ma non meno importante, era la visualizzazione dell'HTML generato dal file TEI. JFrame supporta l'html base ma a nostro parere non era sufficiente per una visualizzazione completa. Per questo motivo abbiamo utilizzato il JFXframe richiamato in un thread separato, adatto a questo scopo.

Design Patterns

Il pattern usato per gestire la “separation of concerns”, ossia la distinzione tra i vari livelli del nostro sistema è il Model View Controller.

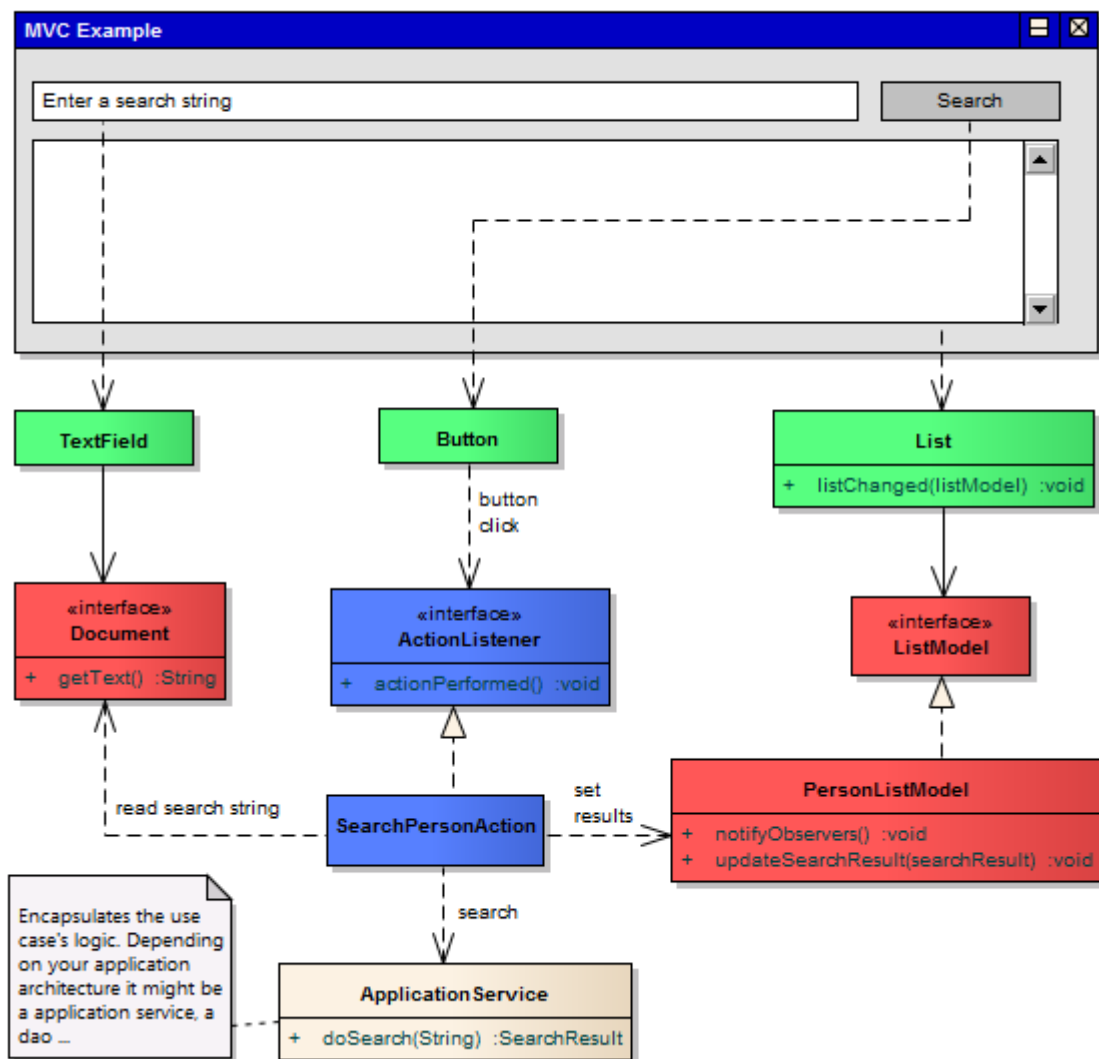
MVC è un pattern architetturale, descrive il più alto livello di astrazione di un sistema software. Questo, nello specifico, consente di separare e disaccoppiare il modello dei dati (model) e la logica applicativa (controller) dalle modalità di visualizzazione e interazione con l'utente (view).



Essendo la nostra implementazione basata su Java Swing, la documentazione relativa all'MVC è stata reperita dal testo “Head First Design Patterns” di Freeman e Freeman.

In Java si ha che – l'interazione tra view e controller avviene in base al meccanismo di propagazione e gestione eventi Swing/AWT • i componenti controller sono `EventListener` (es. `ActionListener`, `MouseListener`...) associati ai componenti grafici view (es. `JButton`)

Esempio della struttura adottata nel nostro progetto Swing.



Le nostre componenti del sistema andranno ad identificarsi in tre ruoli principali: Model, View e Controller. Riassumeremo, di seguito, le funzioni che ciascun ruolo assume.

L'utente interagisce con la view. Il controller riceve la sua azione e la interpreta.

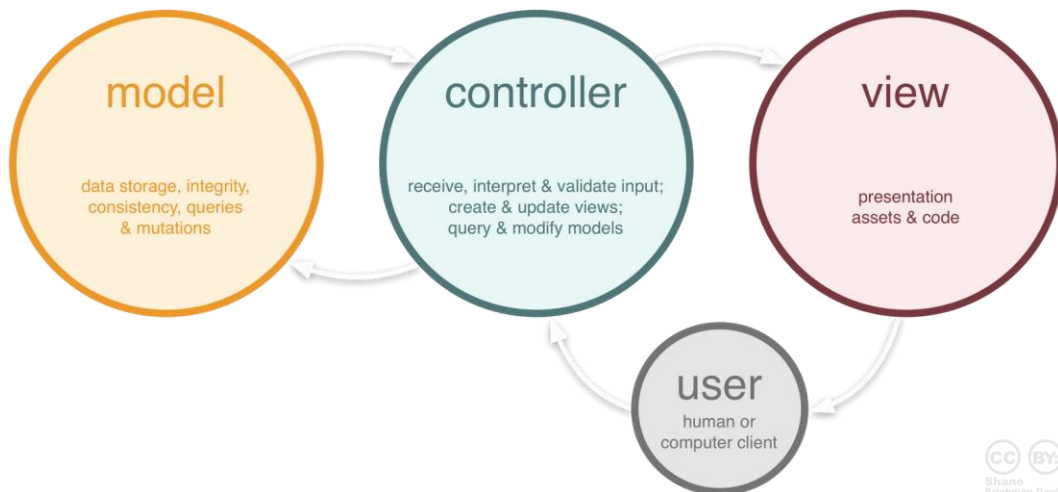
Il controller comunica al model di cambiare il suo stato.

(Facoltativo) Il controller fa cambiare stato anche alla view.

Il model segnala alla view che il suo stato è cambiato.

La view richiede e riceve le informazioni sul nuovo stato del model.

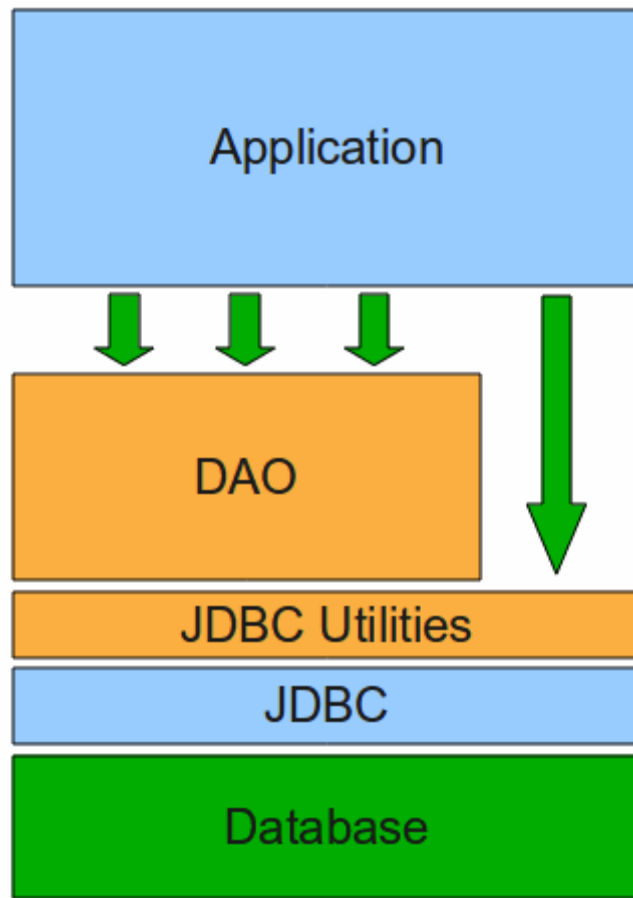
Perché abbiamo scelto MVC?



Volevamo un sistema che fosse loose coupled e highly cohesive. Questo perché in caso di future modifiche al software, il programmatore sarebbe notevolmente facilitato, risultando il codice più comprensibile e le componenti principali facili da individuare.

Fondamentalmente, le proprietà che ci fornisce MVC, sono: estendibilità, riusabilità, interoperabilità. Inoltre si possono anche notare delle qualità interne: strutturazione, modularità (come già specificato in precedenza), comprensibilità e manutenibilità.

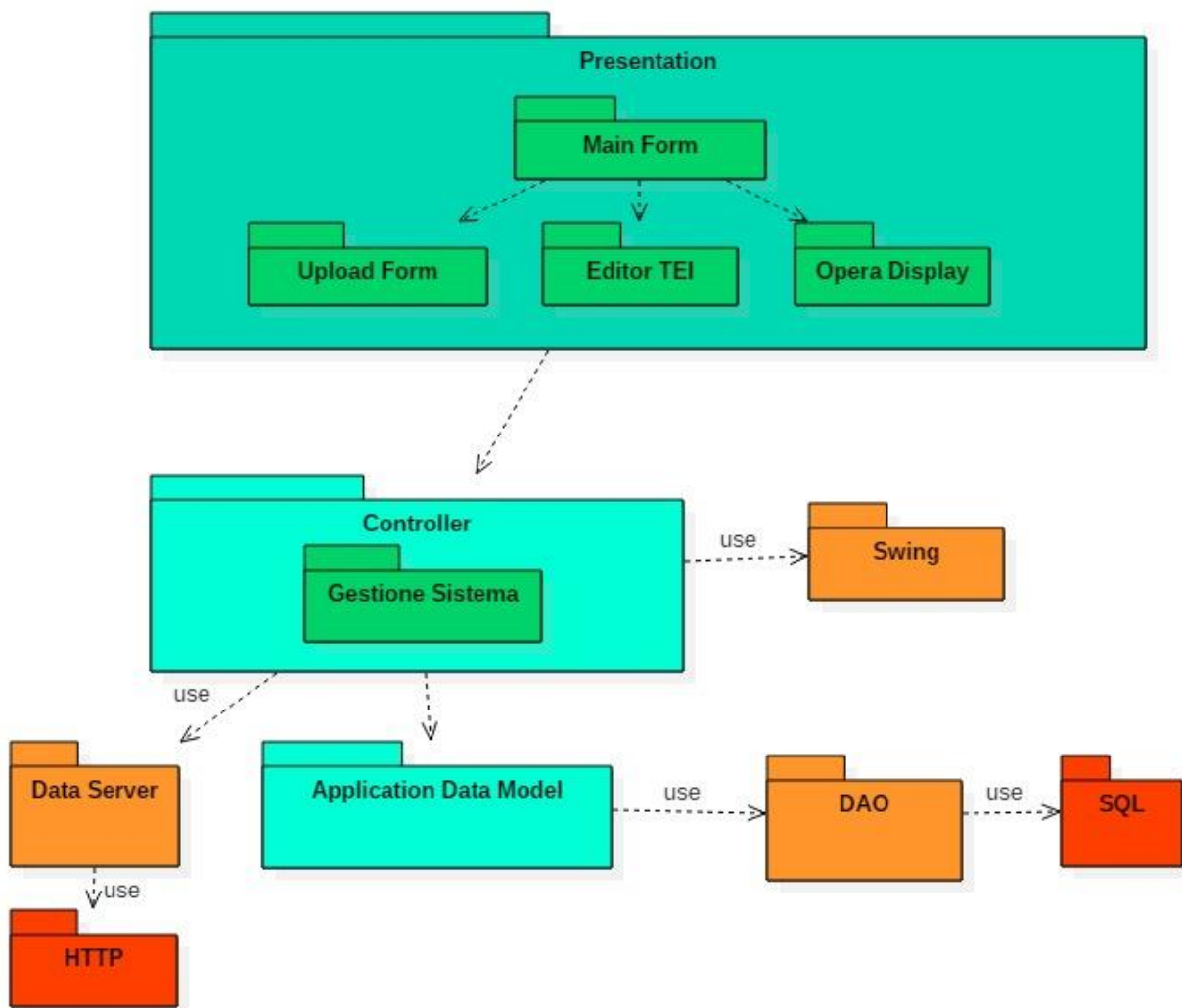
Design pattern adottato: DAO.



Abbiamo sfruttato questo pattern di cui si è discusso nel corso, in quanto ci permette, isolando l'accesso al "data layer" da parte della "business logic" (ossia isolare l'accesso ad una tabella tramite query poste all'interno dei metodi della classe), di avere facilitazioni nella manutenzione ed un maggiore livello di astrazione.

La scelta è sempre correlata a quella fatta per MVC. Vogliamo favorire scalabilità e manutenibilità del software.

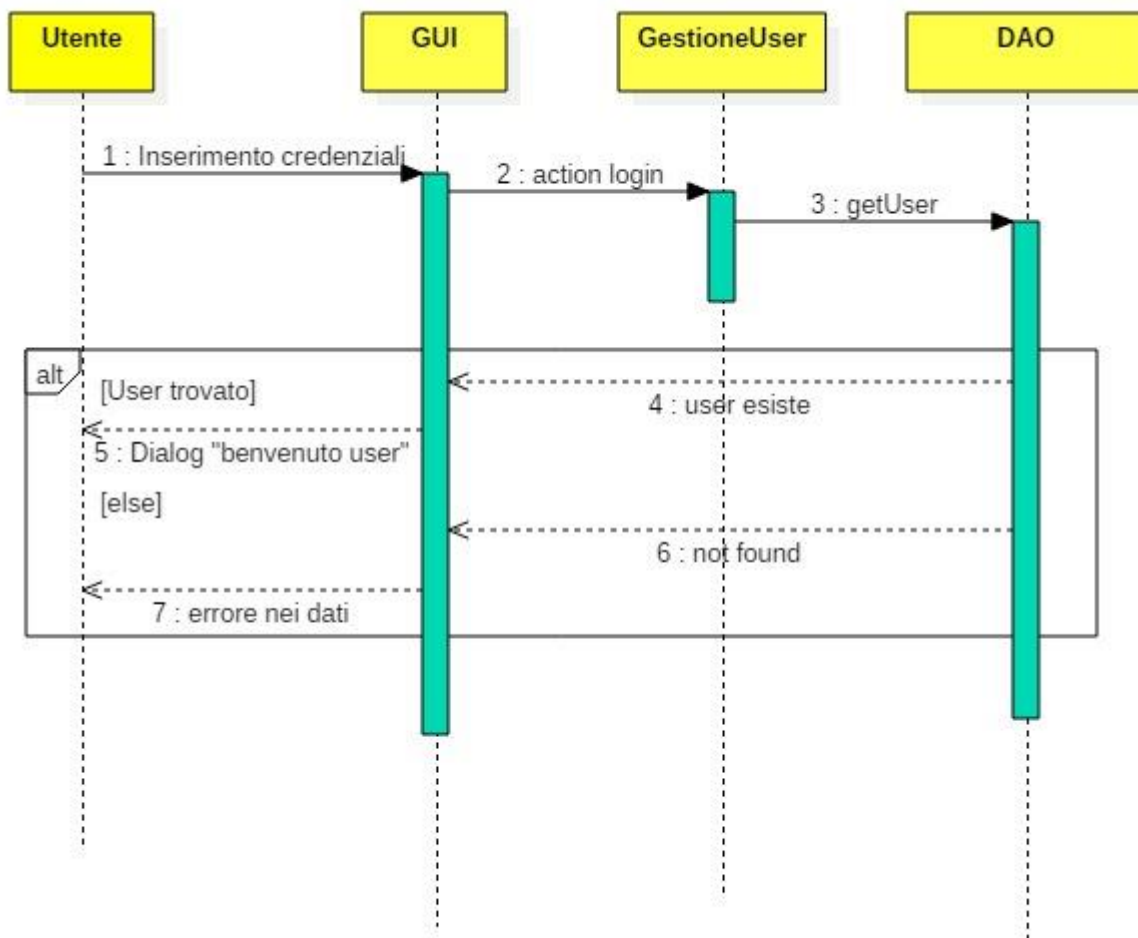
Package Diagram raffigurante il modello del sistema.



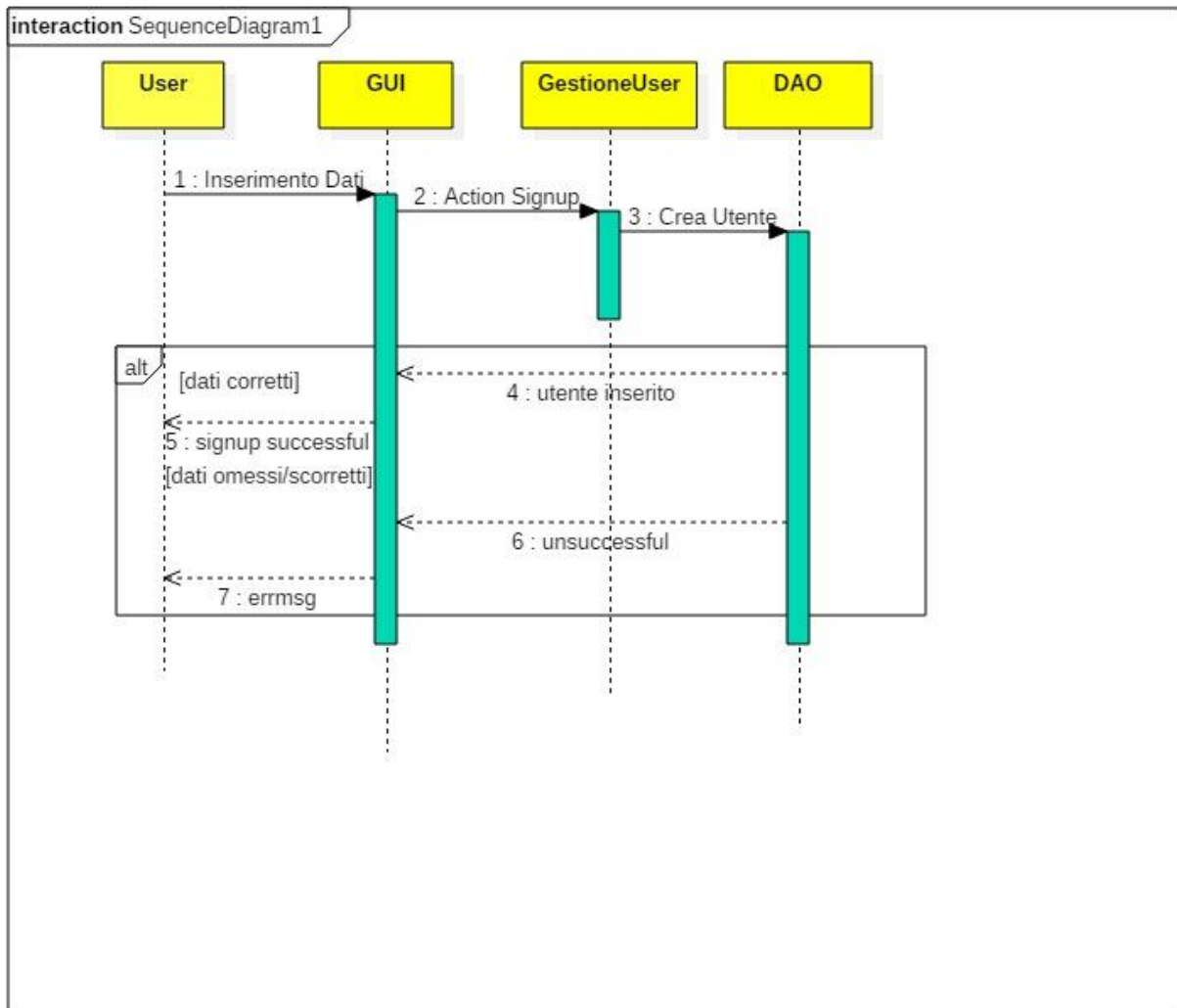
Sequence Diagrams

Sotto verranno illustrati alcuni sequence diagram che raffigurano alcuni scenari per il nostro sistema.

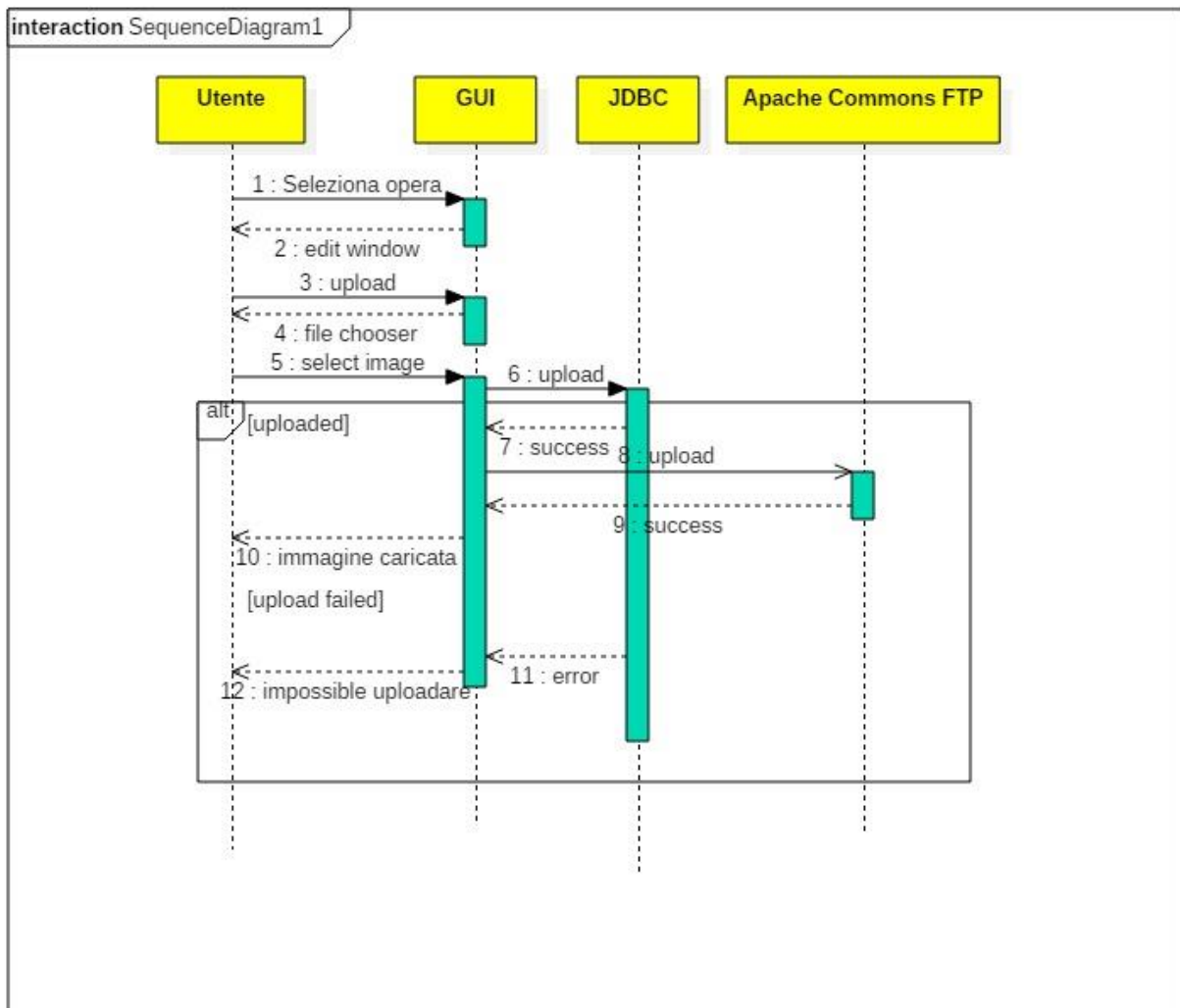
Login



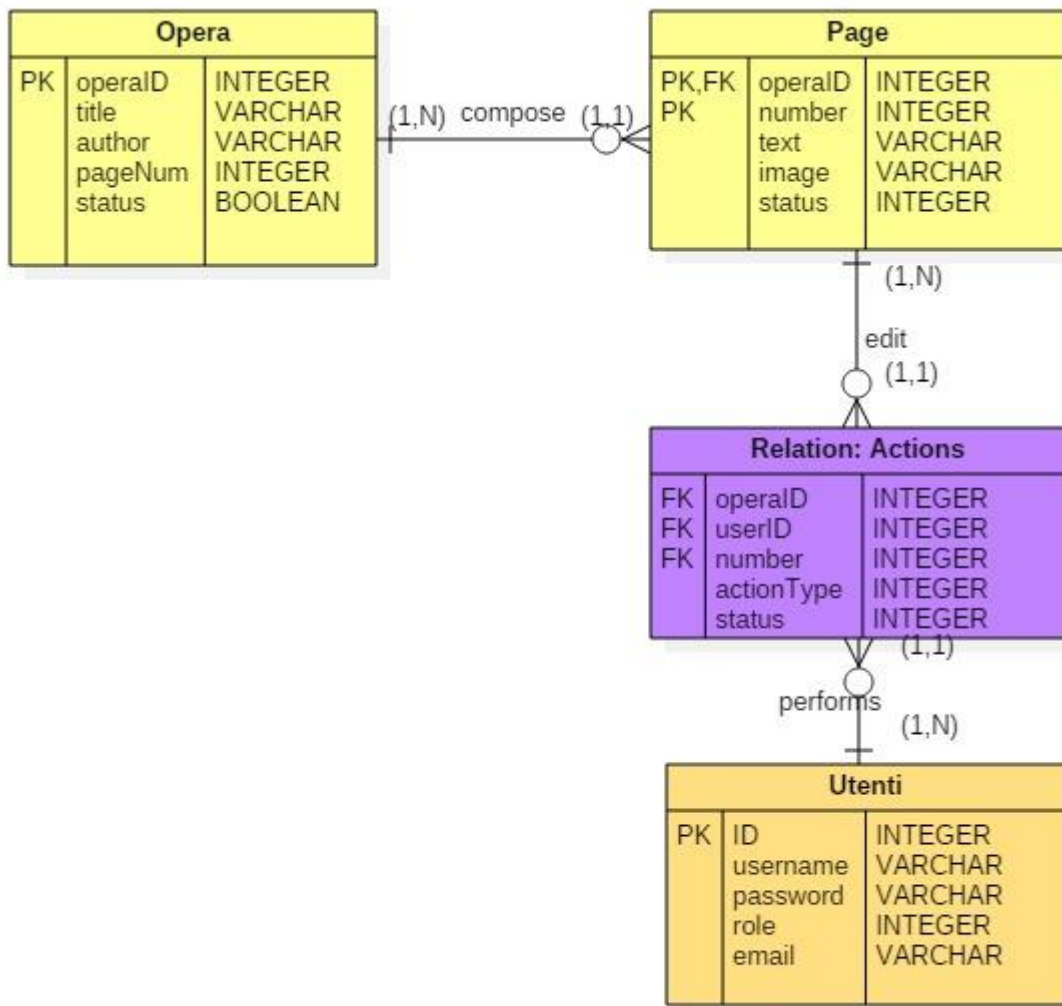
Registrazione



Upload



ER Diagram



Note: abbiamo inteso il ruolo dell'utente come un intero, ad esempio: admin=0, base =1 ecc.

Lo status della "Page" è un intero, dato che potrebbe trovarsi in più stati, come, ad esempio: (testo inserito ma non validato, immagine inserita e validata), (testo non presente, immagine validata) ecc.

Lo status dell'opera invece è un booleano, dato che si può semplicemente trovare nelle due situazioni: pubblicata o non pubblicata.

Nota: i seguenti diagrammi rappresentano la visione pre-implementazione che avevamo del software. Quelli definitivi saranno rappresentati di seguito.

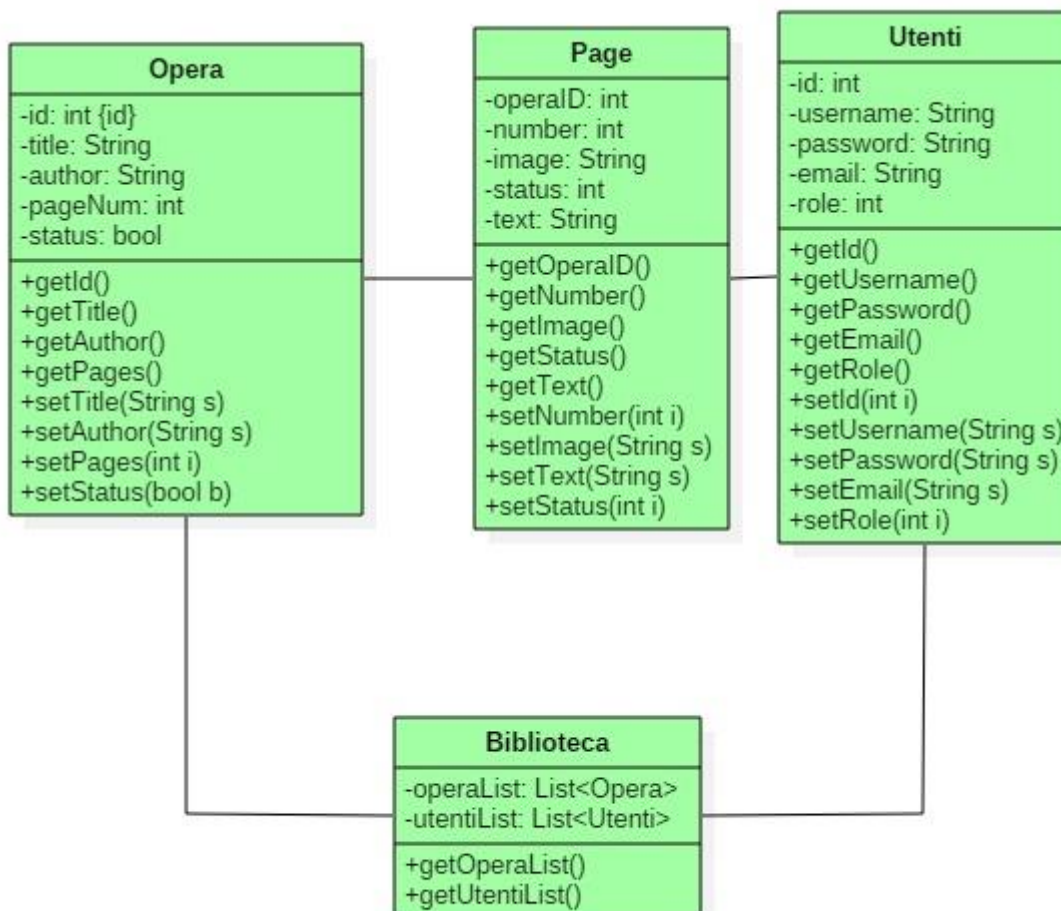
Modello: Object Diagram per il model

L'oggetto Opera è il manoscritto nella sua interezza. L'id la rappresenta univocamente, è composto poi dai dati relativi al manoscritto (titolo, autore e numero di pagine), mentre lo status è un booleano che indica l'avvenuta pubblicazione o meno.

L'oggetto Page indica la singola pagina del manoscritto. Fa riferimento all'id dell'opera a cui appartiene ed il numero è la sua collocazione all'interno di essa. Ogni Page è composta da un'immagine e l'eventuale testo trascritto. Lo status indica quali di queste due componenti è presente e convalidata.

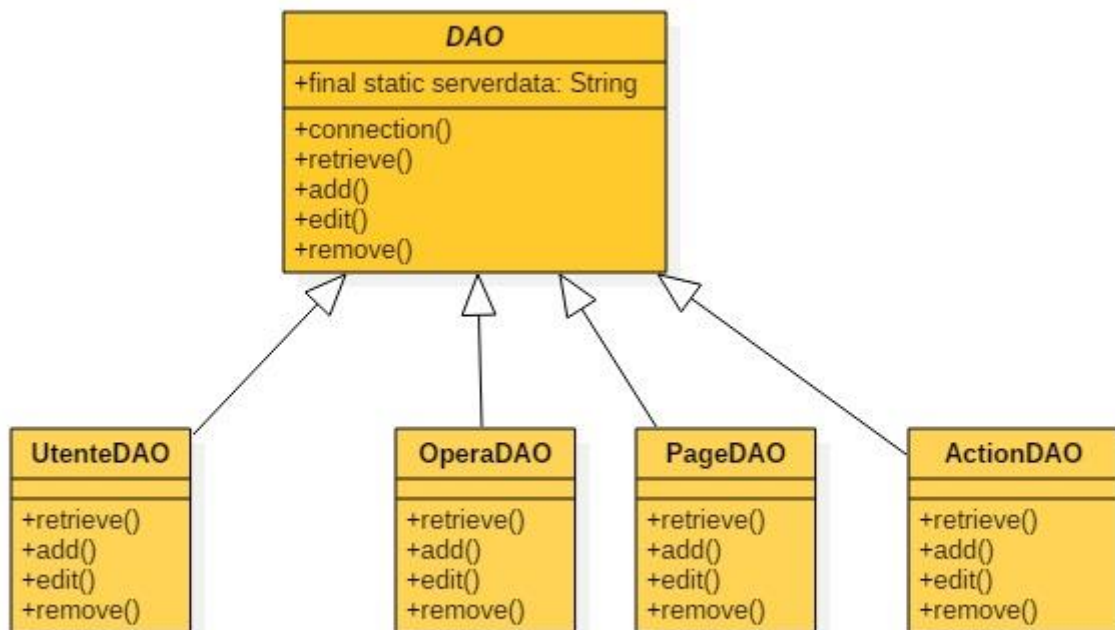
L'oggetto Utenti identifica coloro che sono registrati al sistema. Sono univocamente rappresentati da un id, mentre i restanti dati sono relativi al login (username, password ed email) ed alla loro funzione nel sistema (role).

L'oggetto Biblioteca ci permette di accedere alla totalità delle opere e degli utenti nel sistema.



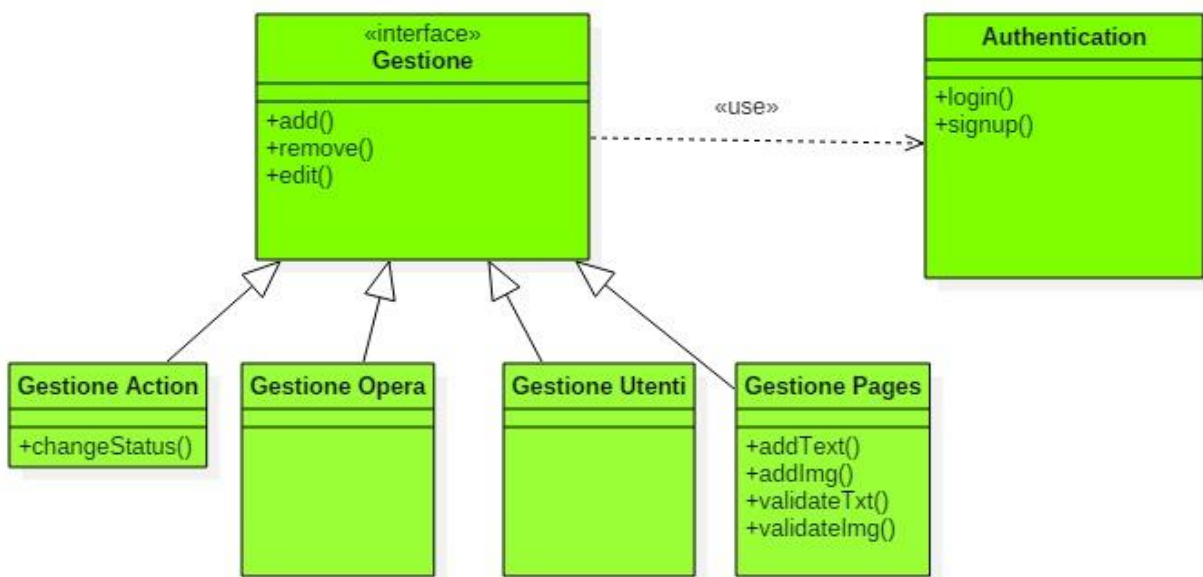
Class Diagram: DAO

Il seguente class diagram ci consente di visualizzare come le nostre classi DAO interagiranno col DB.



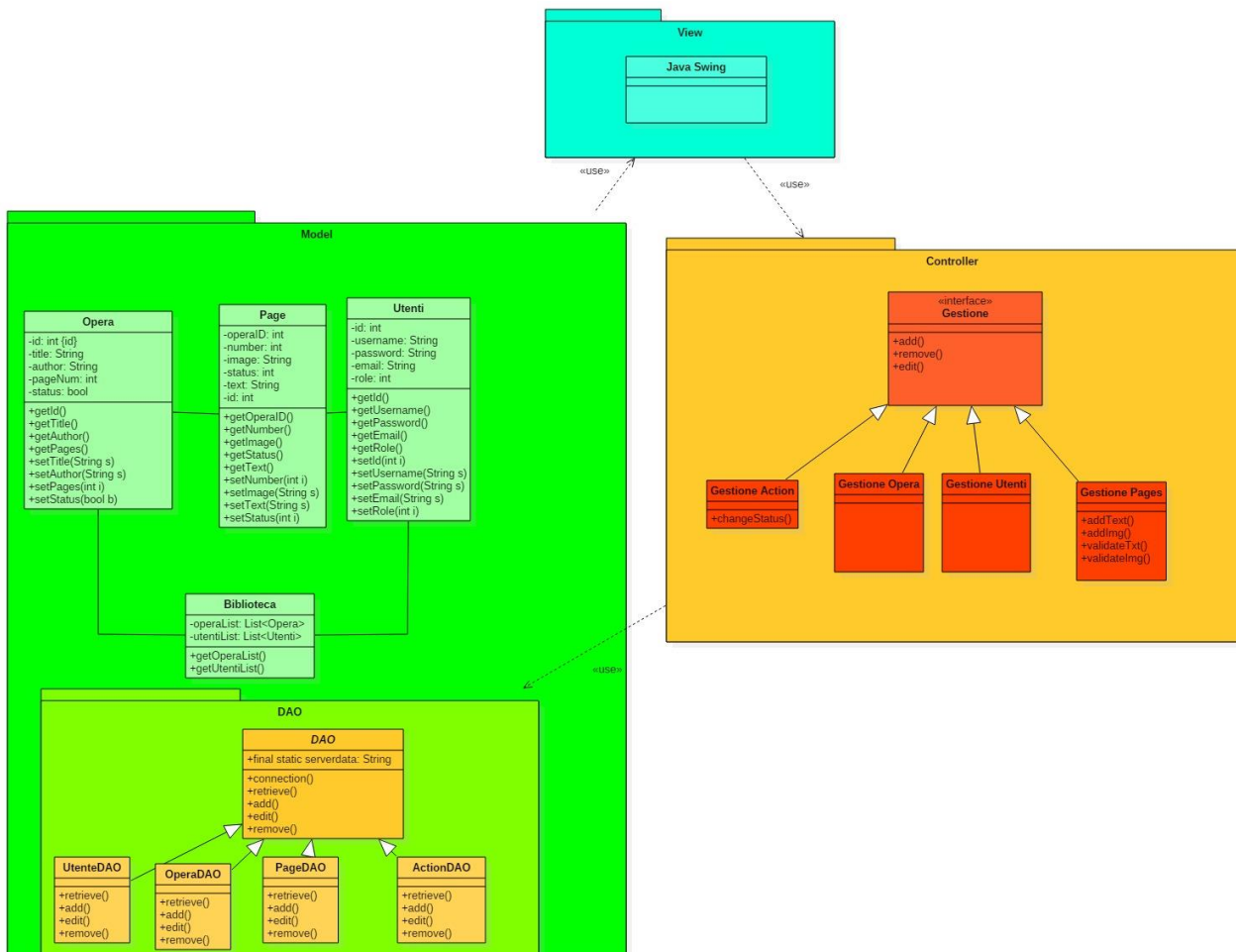
Class Diagram: Controller

Funzioni delle classi controller.

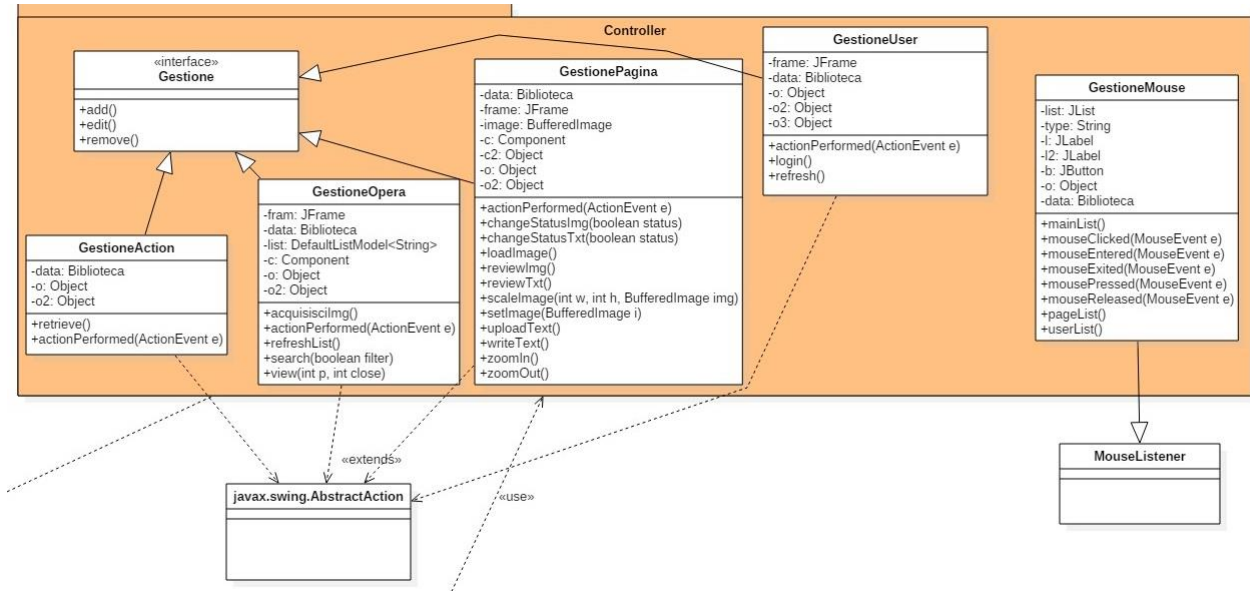


Class diagram totale

Nota: Questo class diagram è quello che abbiamo usato come riferimento per l'implementazione. Nelle prossime pagine, invece, inseriremo i class diagram sviluppati a posteriori (i quali rappresentano in maniera fedele il nostro software)



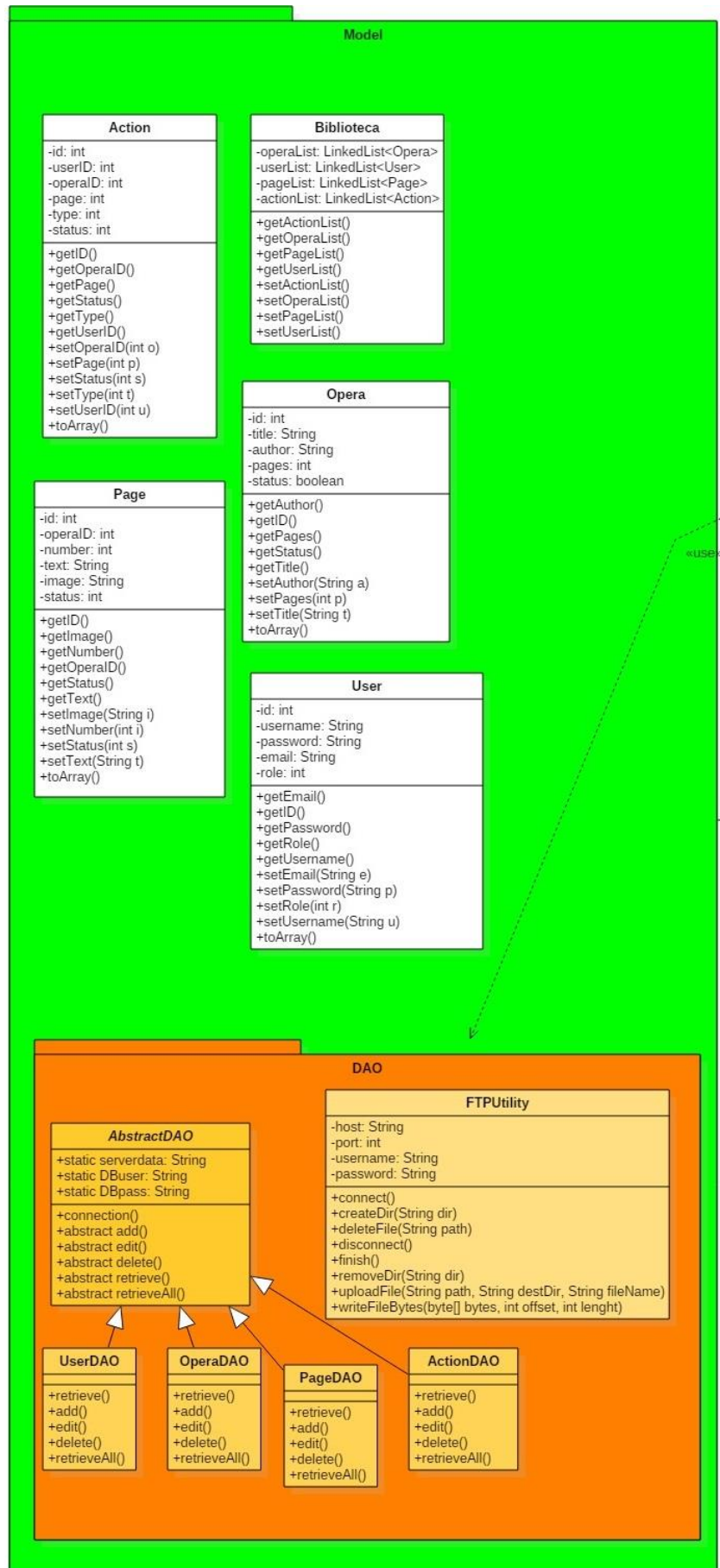
CLASS DIAGRAMS DEFINITIVI



Package Controller

Nel package controller gestiamo tutte le action che vengono intercettate nelle views. Swing di norma prevede che la gestione delle action vengano fatte direttamente nelle view ma in questo modo si andrebbe fuori dal pattern MVC. Per evitare ciò abbiamo creato le classi nel package estendendo la classe astratta `AbstractAction` cosicché abbiamo potuto trattarle come action listeners. Questo è anche il motivo per cui molte delle classi prendono come argomenti degli oggetti generici, essendo richiamate da action completamente differenti fra loro anche se applicate allo stesso model.

Inoltre tutte le classi implementano l'interfaccia `Gestione` e le relative funzioni, fatta eccezione per `GestioneMouse` che viene utilizzata esclusivamente per mostrare maggiori dettagli dopo la relativa selezione su `JList`.

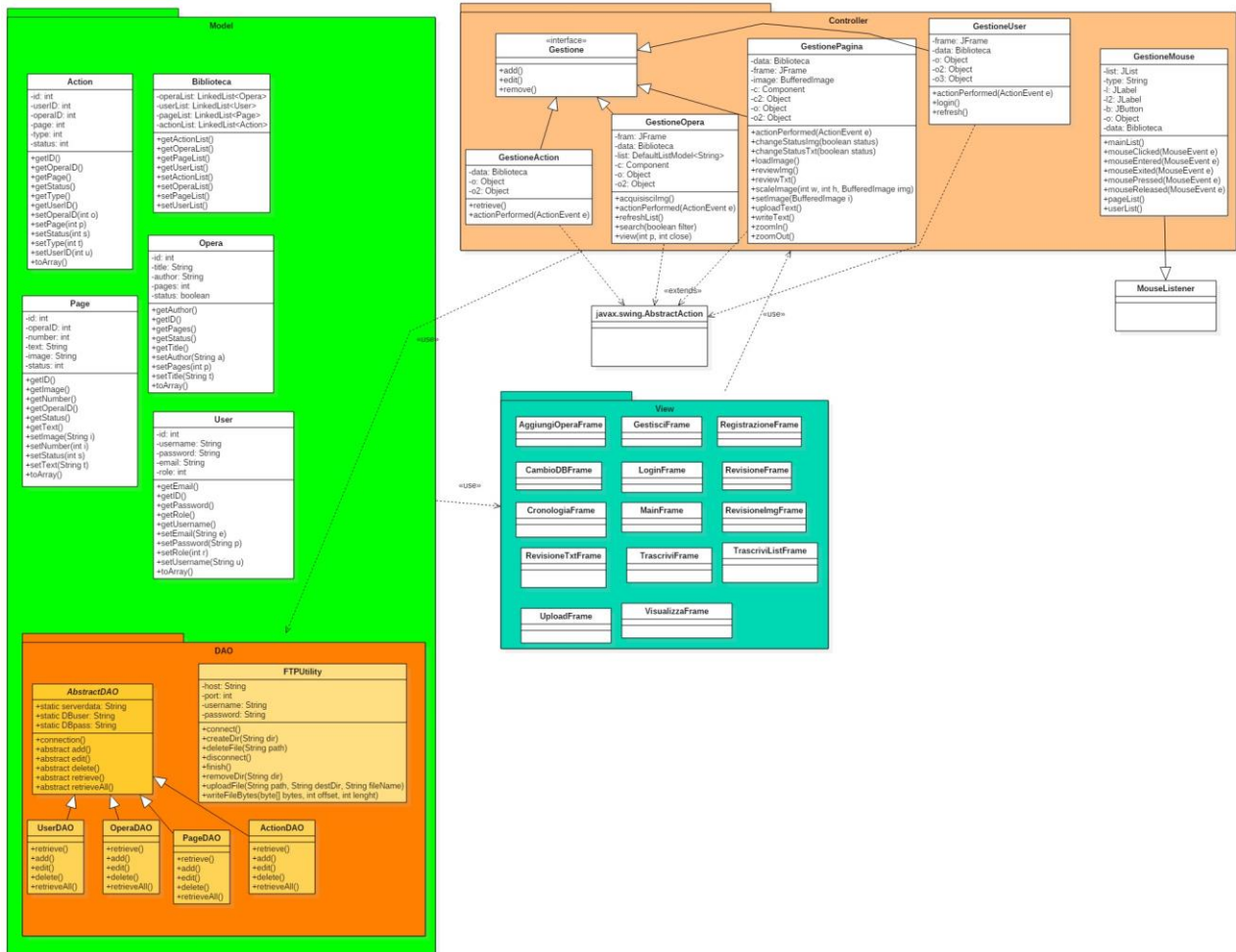


Package Model

Come specificato precedentemente, in questo package vengono dichiarate tutte le classi utili alla gestione da parte dei controller dei dati presenti sul DB. Rispetto alla versione precedente sono stati aggiunti dei costruttori che prendono il campo id come argomento in ogni model, così che possano costruirsi richiamando il relativo DAO.

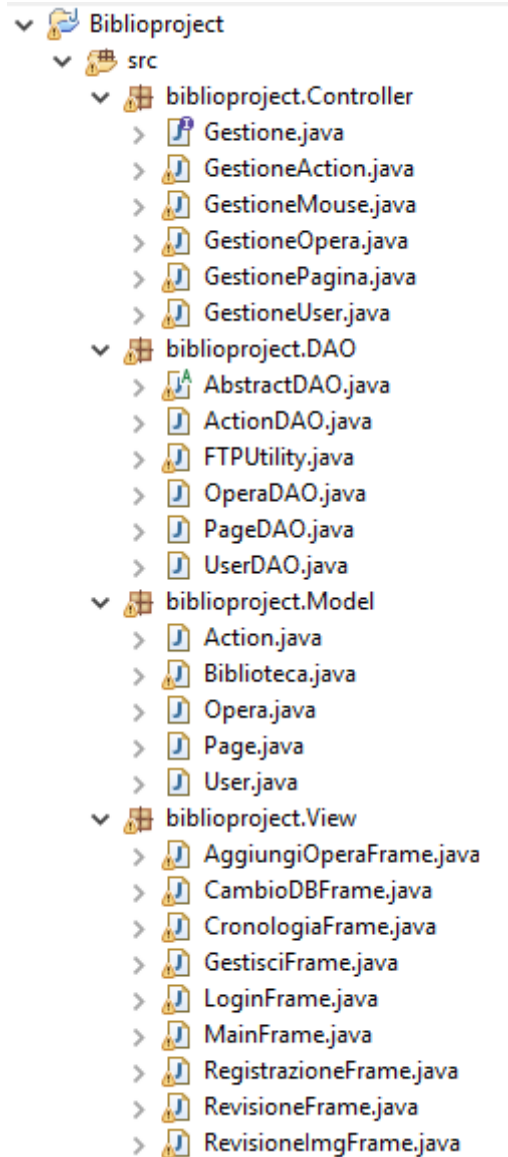
Le classi DAO si interfacciano con il DB manipolandone i dati. La classe che nello specifico si occupa della connessione al DB è AbstractDAO che alla prima connessione utilizza il file di configurazione locale presente nella stessa path del programma per ottenere le credenziali di connessione. Scelta dovuta al fatto che l'Admin del sistema può cambiare in qualsiasi momento DB mediante il pannello di gestione database. Inoltre questa classe contiene le definizioni delle funzioni che devono essere dichiarate anche nelle classi che le ereditano. Rispetto alla versione precedente del design viene dichiarata la funzione retrieveAll() che scarica dal DB tutti gli ID nella relativa tabella, utilizzata dal model Biblioteca.

Si distingue dalle altre la classe FTPUtility che utilizza la libreria Apache Commons per effettuare la manipolazione dei dati sul server online dove sono contenute le acquisizioni delle opere. Contiene funzioni utili al popolamento, modifica e cancellazione dei singoli file o intere cartelle.

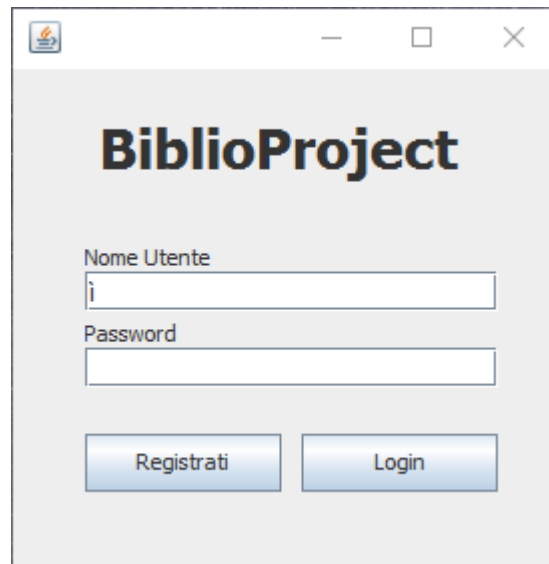


Class diagram completo

IMPLEMENTAZIONE (VISTE)



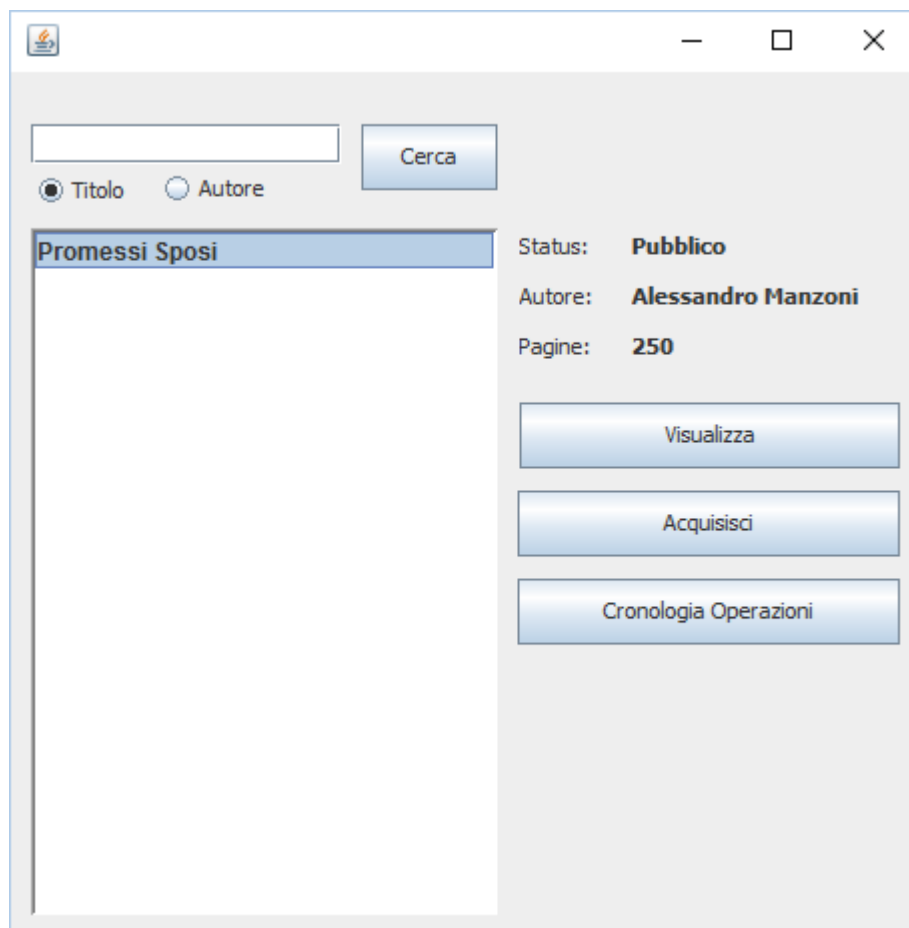
Overview dei package.



A screenshot of a web application window titled "BiblioProject". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. The main content area is light gray and contains the following elements:

- The title "BiblioProject" in a large, bold, black font.
- A label "Nome Utente" above a text input field containing the letter "i".
- A label "Password" above an empty password input field.
- Two buttons at the bottom: "Registrati" and "Login", both with a blue gradient and rounded corners.

Login screen



A screenshot of the "BiblioProject" main form. The window has a standard Windows-style title bar. The main content area is light gray and contains the following elements:

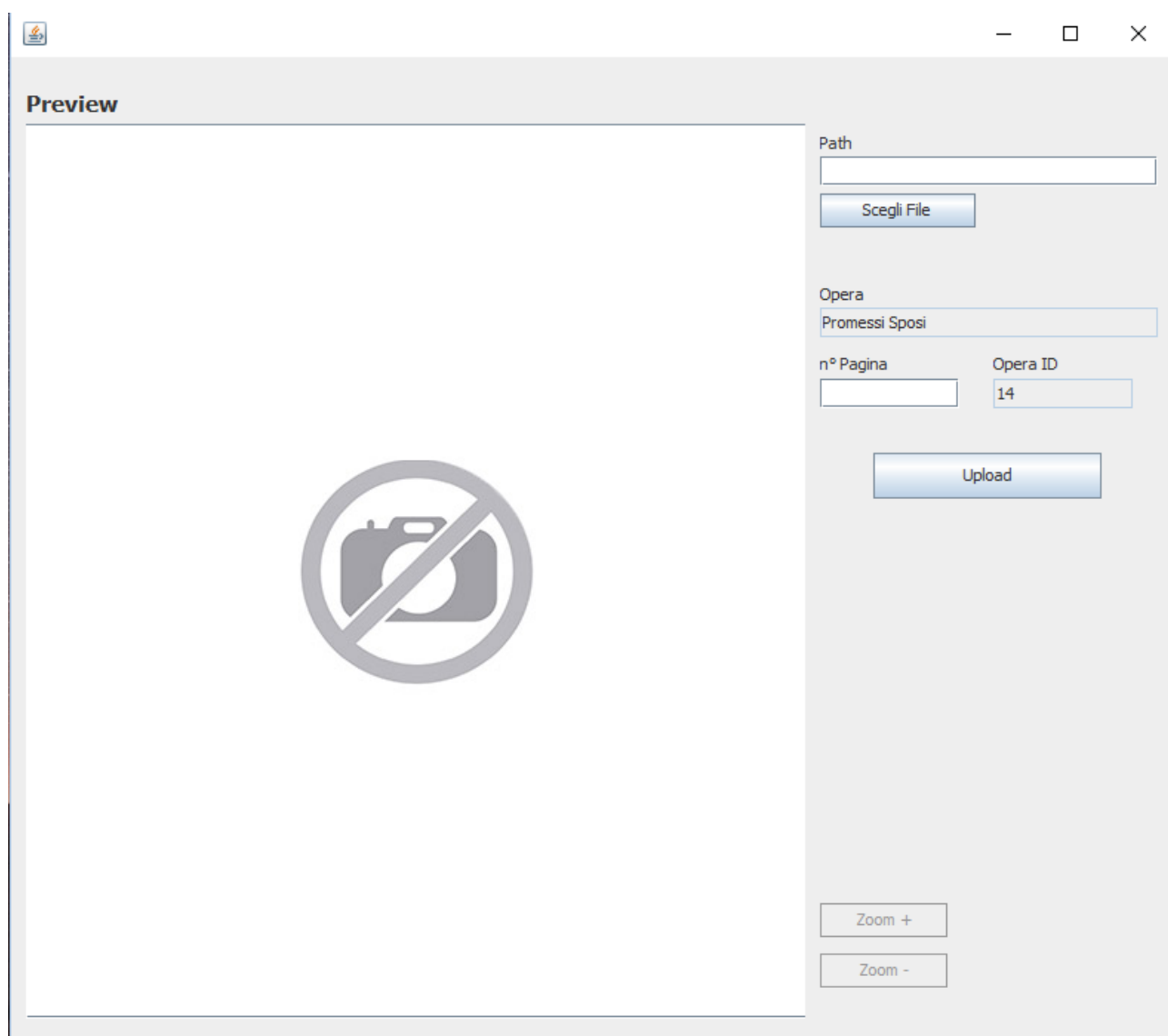
- A search bar at the top left with a text input field and a "Cerca" button.
- Below the search bar, two radio buttons: "Titolo" (selected) and "Autore".
- A list of search results. The first result, "Promessi Sposi", is highlighted with a blue background. Below it is a large empty rectangular area.
- To the right of the list, the following details are displayed:
 - Status: **Pubblico**
 - Autore: **Alessandro Manzoni**
 - Pagine: **250**
- Below the details, three buttons are stacked vertically: "Visualizza", "Acquisisci", and "Cronologia Operazioni", all with a blue gradient and rounded corners.

Main Form

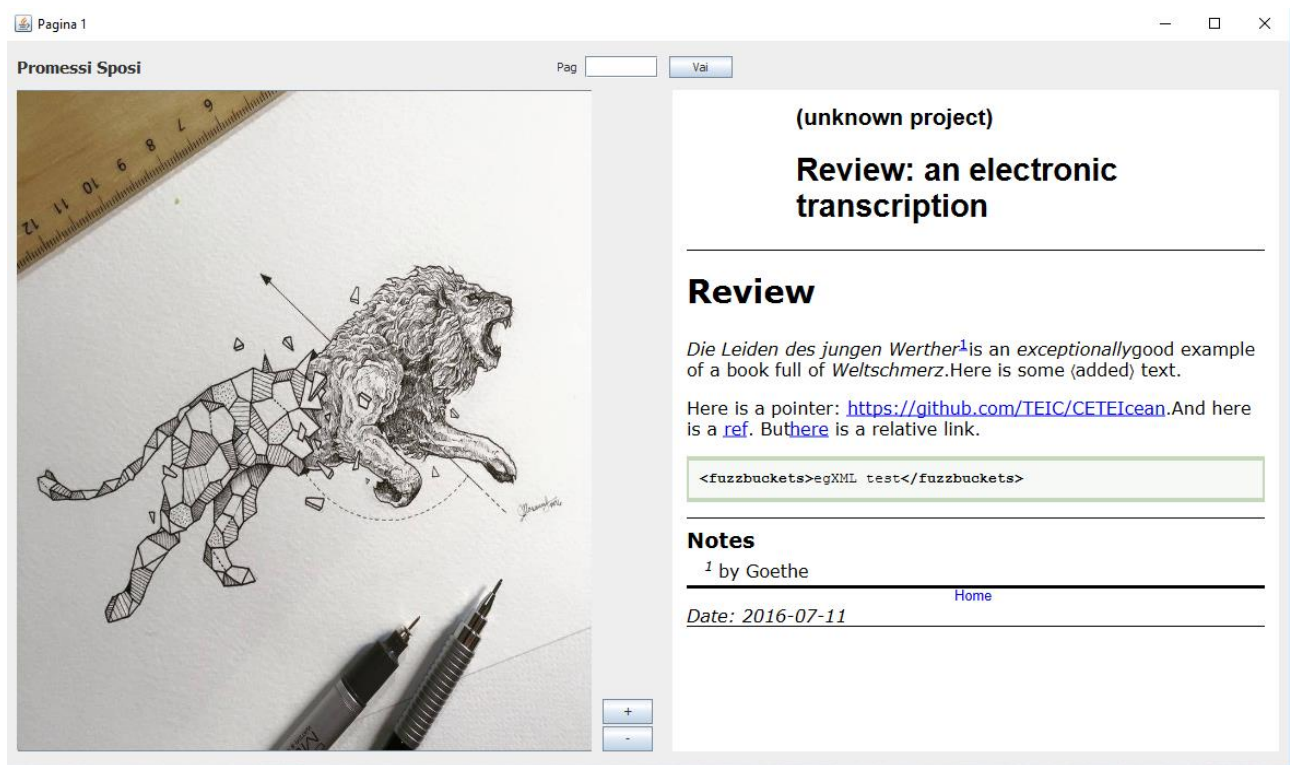
The screenshot shows a Windows-style application window titled "Main Form (Admin)". The interface is divided into several sections:

- Search Section:** Located at the top left, it includes a text input field, a "Cerca" (Search) button, and two radio buttons labeled "Titolo" (selected) and "Autore".
- Database Management Section:** A prominent red button labeled "Gestione DB" is located at the top right.
- Table Section:** A table with a single row containing the text "Promessi Sposi". The table has a blue header and a light blue body.
- Details Section:** To the right of the table, there are three lines of text: "Status: **Pubblico**", "Autore: **Alessandro Manzoni**", and "Pagine: **250**".
- Action Buttons:** A vertical stack of six buttons is located on the right side of the form: "Visualizza", "Pubblica/Privata", "Elimina Opera", "Gestisci Pagine", "Aggiungi Opera", and "Pannello Utenti".

Main Form (Admin)



Upload Immagine



Visualizza Opera

Tutti i file riguardanti il software, nonché il file mdj contenente tutti i diagrammi realizzati con StarUML è contenuto nella repository Git.