Object Oriented Programming

Biblioteca Digitale

Team Members		
Name	Student Number	E-mail address
Marco Amadio	***230406***	***mark_ap@hotmail.it***
Marco Angelini	***228613***	***marcogila94@gmail.com***

Requirements

Requisiti funzionali

- **Visualizzazione elenco titoli**
  Funzionalità che permette agli utenti non registrati di visualizzare l'elenco dei titoli delle opere.

- **Login**
  Funzionalità tramite la quale un utente può accedere al sistema.

- **Registrazione**
  Funzione che permette a un utente non iscritto al sistema di registrarsi.

- **Visualizzazione opere**
  Permette agli utenti registrati di vedere in dettaglio le opere presenti.

- **Pagina personale**
  Racchiude le informazioni anagrafiche degli utenti registrati.

- **Inserimento utente**
  Funzione riservata all'*Amministratore*; permette di inserire un nuovo utente all'interno del portale.

- **Inserimento opera**
  Funzione riservata all'*Amministratore*; permette di inserire una nuova opera all'interno del portale.

- **Pubblicazione opera**
  Funzione riservata all'*Amministratore*; permette di pubblicare un' opera completamente revisionata.

- **Eliminazione opera**
  Funzione riservata all'*Amministratore*; permette di eliminare un' opera all'interno del portale.

- **Upload immagine**
  Funzione che permette all'*Acquisitore* di caricare le immagini nel sistema per sottoporle poi a revisione.

- **Eliminazione immagine**
  Funzione che permette al *Revisore Acquisizioni* di eliminare un immagine precedentemente caricata.

- **Revisione immagine**
  Funzione che permette al *Revisore Acquisizioni* di verificare gli standard di qualità dell'immagine.

- **Caricamento trascrizione**
  Funzione che permette al *Trascrittore* di caricare sul database il testo trascritto, rendendolo disponibile a revisione.

- **Eliminazione trascrizione**
  Funzione che permette al *Revisore Trascrizioni* di eliminare una trascrizione precedentemente caricata.

- **Revisione trascrizione**
  Funzione che permette al *Revisore Trascrizioni* di verificare la correttezza del testo caricato.

- **Visualizzazione editor**
  Funzione riservata al *Trascrittore* e *Revisore Trascrizioni;* essi avranno a disposizione un editor dove poter trascrivere o revisionare un'opera.

- **Promozione ruolo**
  Funzione riservata all'*Amministratore*; permette di promuovere un utente a un determinato ruolo.

Requisiti non-funzionali

- **Availability**
  Il sistema dovrà essere sempre disponibile e deve poter garantire in qualsiasi momenti tutte le funzionalità.

- **Usability**
  Il portale dovrà risultare pulito e di facile utilizzo.

- **Reliability**
  Il sistema dovrà garantire all'utente le funzioni messe a disposizione senza incorrere a errori.

Attori del sistema

- **Utente base :** Utente non ancora registrato; ha la possibilità di visualizzare solamente l'elenco dei titoli

- **Utente avanzato :** Utente registrato; può visualizzare le opere in maniera completa.

- **Acquisitore :** Effettua l'upload delle immagini.

- **Revisore acquisizioni :** Si occupa di revisionare le immagini caricate.

- **Trascrittore :** Ha il compito di trascrivere i testi in formato digitale.

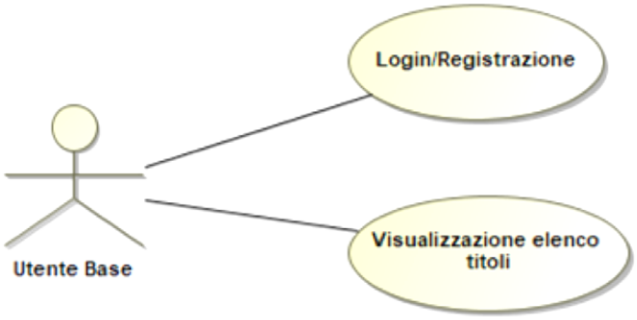
- **Revisore trascrizioni :** Si occupa di verificare la correttezza delle trascrizioni

- **Amministratore :** Ha una gestione completa del sistema, può inserire ed eliminare opere e utenti, ma anche fungersi da trascrittore, acquisitore ed altri attori del portale; ha tutti i privilegi. Può inoltre promuovere gli utenti al ruolo che desidera.

Use Case Diagram

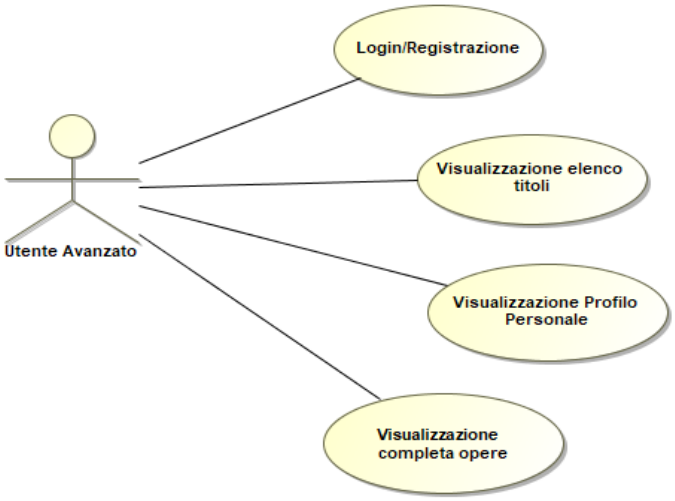
Abbiamo deciso di suddividere il nostri Use Case in base ai singoli attori del sistema, per poi ricomporlo alla fine di questa sezione.

Utente base



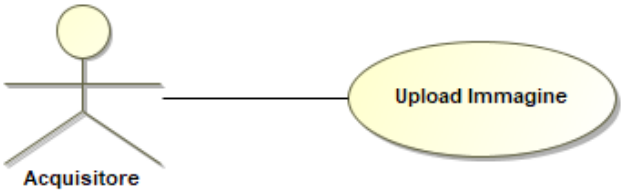
Non essendo ancora registrato l'utente base può effettuare la Registrazione e visualizzare l'elenco titoli.

Utente avanzato



L'utente avanzato è un attore già registrato nel portale, tra le funzionalità riservate possiamo vedere la visualizzazione completa delle opere e la possibilità di accedere alla propria pagina personale.

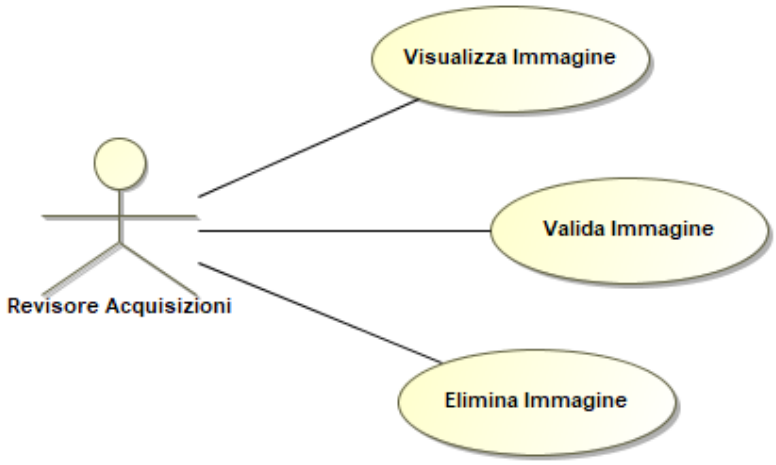
Acquisitore



Partiamo dal presupposto che qualsiasi ruolo superiore all'utente avanzato erediti da lui tutte funzionalità, quindi l'acquisitore, il trascrittore, i revisori e l'amministratore avranno la possibilità di loggarsi, accedere alla propria pagina personale e visualizzare le opere in tutti i contenuti (testo e immagini).

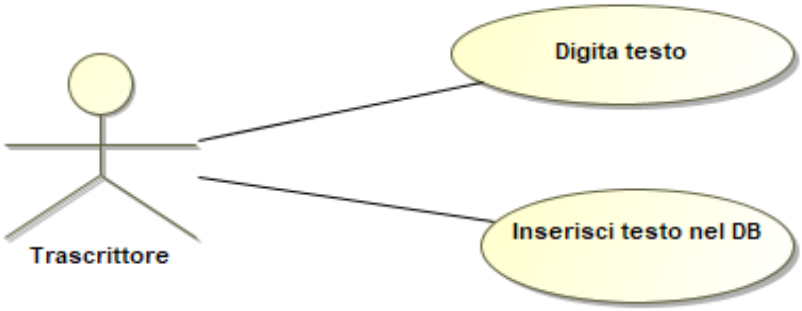
L'acquisitore ha il compito di effettuare l'upload delle immagini.

Revisore acquisizioni



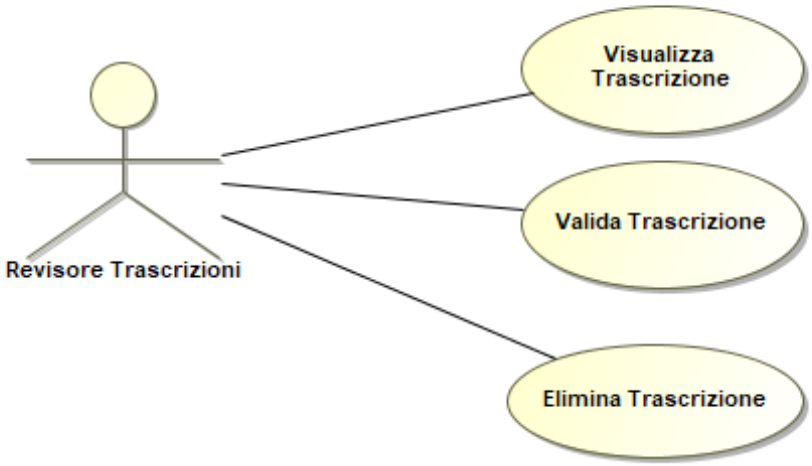
Il revisore delle acquisizioni ha il compito di controllare la correttezza dell'immagine caricata, con la possibilità di validarla o eliminarla.

Trascrittore



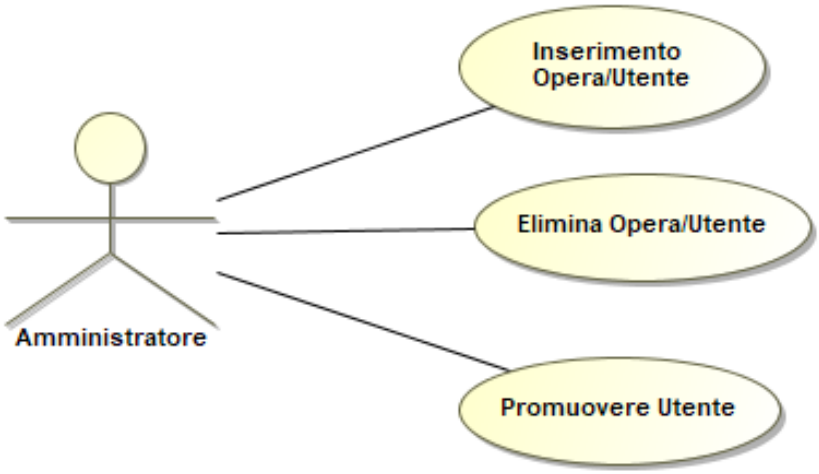
Il Trascrittore ha il compito di digitalizzare il testo e caricarlo sul database.

Revisore trascrizioni



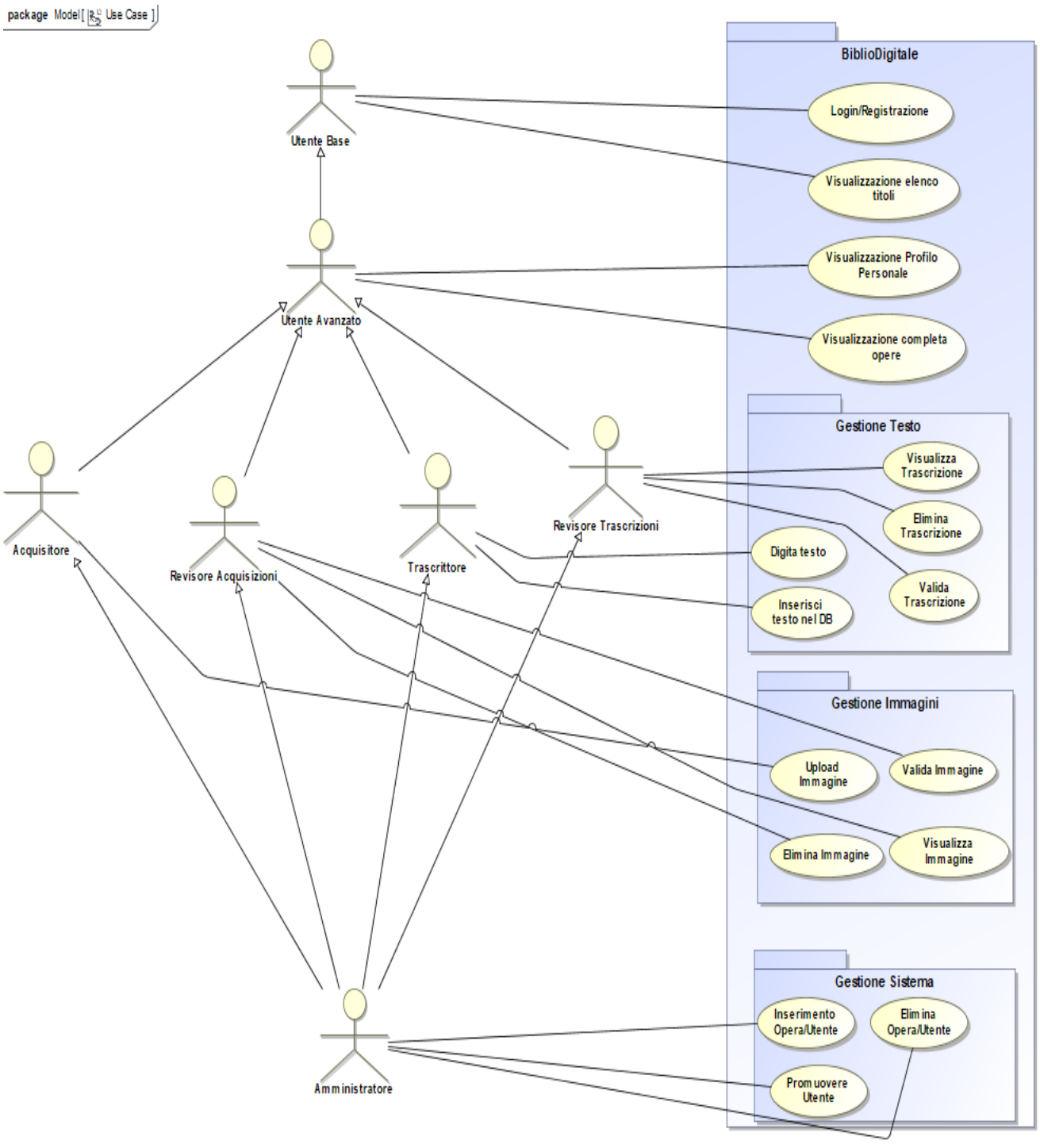
Il revisore delle trascrizioni ha il compito di controllare la correttezza della trascrizione caricata, con la possibilità di validarla o eliminarla.

Amministratore



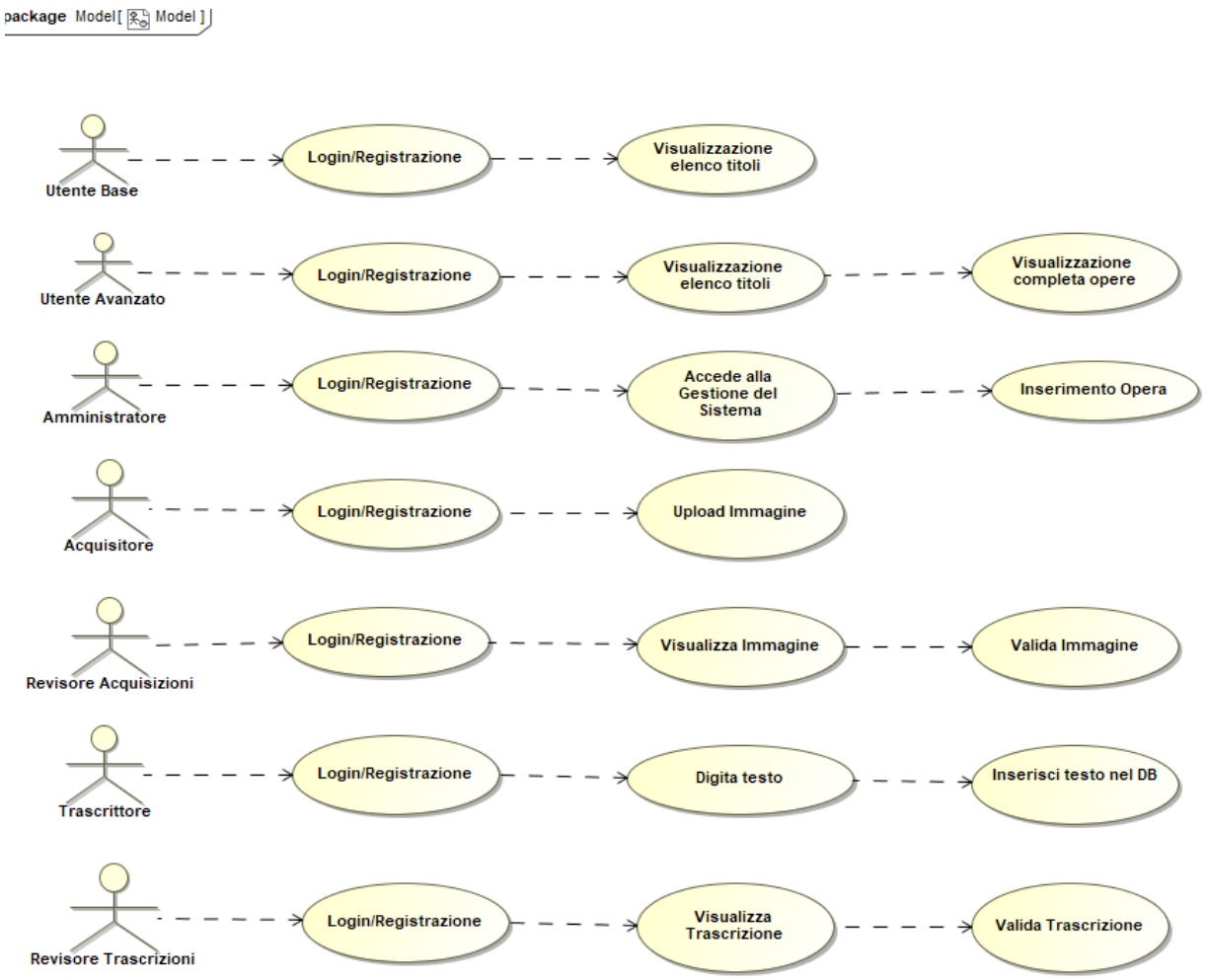
Ha una gestione completa del sistema, può inserire ed eliminare opere e utenti, ma anche fungersi da trascrittore, acquisitore ed altri attori del portale; ha tutti i privilegi. Può inoltre promuovere gli utenti al ruolo che desidera.

Lo Use Case diagram risultante è il seguente:



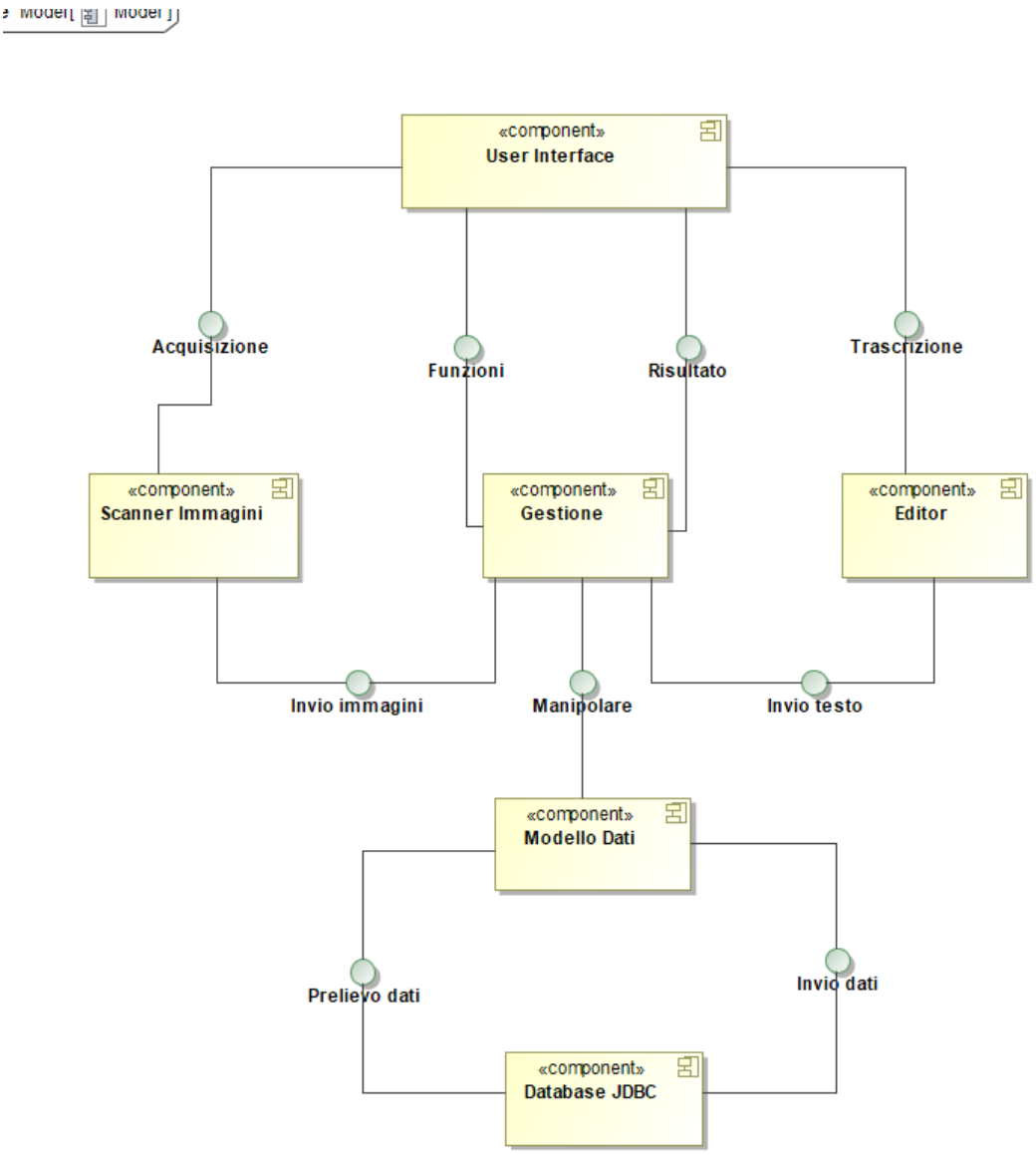
Scenari

Mostriamo di seguito, tramite l'utilizzo degli Use Case, una serie di scenari in cui gli attori del sistema possono essere coinvolti.



System Design

Modello dell'architettura del sistema



Descrizione dell'architettura

Dopo un confronto sulla possibile architettura del nostro sistema abbiamo optato per un implementazione web della biblioteca digitale.

Le componenti principali della nostra architettura sono:

- User Interface
- Scanner Immagini
- Modello Dati
- Editor
- Gestione
- Database JDBC

La User Interface è la componente utilizzata dall'utente per interagire con il sistema. Egli immetterà dati in input e visualizzerà i risultati forniti dal sistema. Corrisponde alla nostra componente View.

La componente View sarà composta dalle classi del nostro motore di templating, in particolare FreeMarker.
Questa classe costituirà il nostro motore che girerà dietro la View e permetterà il passaggio di dati verso l'html.

La componente di Gestione corrisponde al nostro Controller e include tutte le funzionalità del sistema, come la gestione delle immagini, delle trascrizioni e delle opere.

Il Database JDBC corrisponde al Local Storage del sistema, in cui verranno salvati tutti i dati relativi a immagini, testi, opere e utenti registrati.

Il Modello Dati permetterà l'utilizzo dei nostri oggetti alla componente Controller.

L'Editor è la componente che consentirà di trascrivere i testi in formato digitale.

Infine c'è lo Scanner Immagini che permetterà agli utenti addetti di caricare le immagini nel sistema. (componente esterna al sistema)

Scelte e strategie adottate

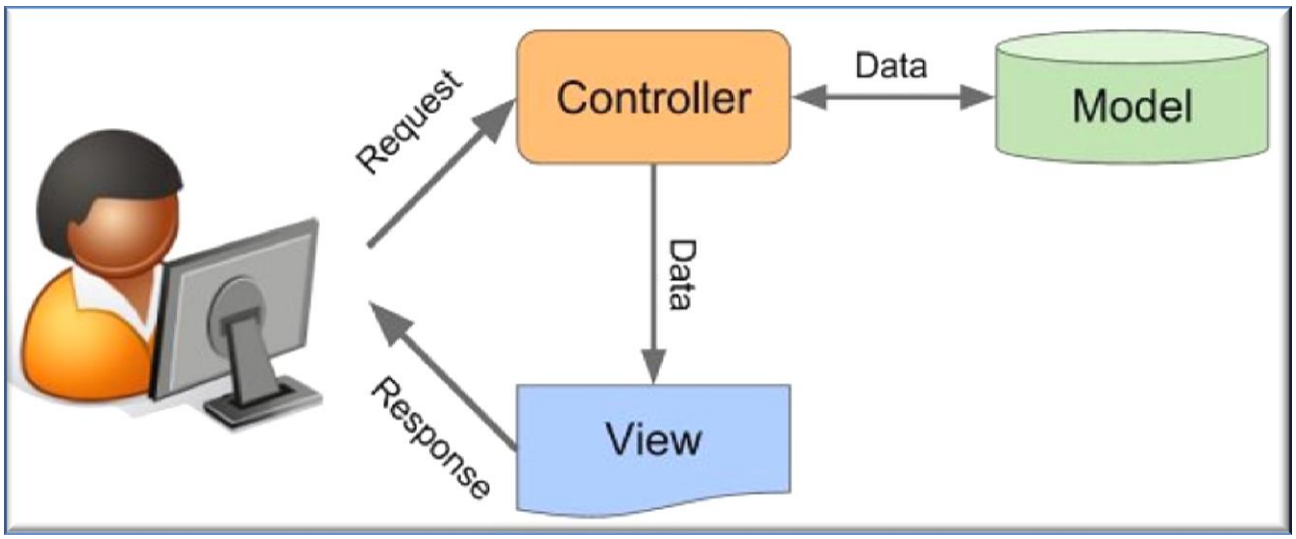
Una scelta a nostro avviso molto importante è stata quella di suddividere il nostro Controller in tre sotto moduli riguardanti la gestione delle immagini, la gestione dei testi e quella delle opere.
Questa scelta è legata al concetto di modularità, ossia la possibilità di modificare e aggiornare un modulo senza rischiare di incidere e compromettere gli altri.
Questo concetto è stato applicato in determinate funzioni, ossia quelle che rischiavano maggiormente di poter essere modificate e aggiornate nel tempo.
Al contrario è stato tralasciato in funzioni meno complesse o perlomeno abbastanza statiche e non mutuabili nel tempo come ad esempio il Login, o la Registrazione.

Un'altra scelta riguarda l'Editor; abbiamo cercato di includere nella nostra applicazione un Editor TEI, tuttavia abbiamo trovato difficoltà sia nella ricerca vera e propria di esso, sia nell'implementazione all'interno della nostra applicazione web.
Abbiamo optato infine per l'Editor OpenSource **TinyMCE** , un editor testuale scritto in Javascript e rilasciato dalla Ephox.
TinyMCE ha la capacità di convertire i campi Textarea html o altri elementi html in istanze di editor.

Design Pattern

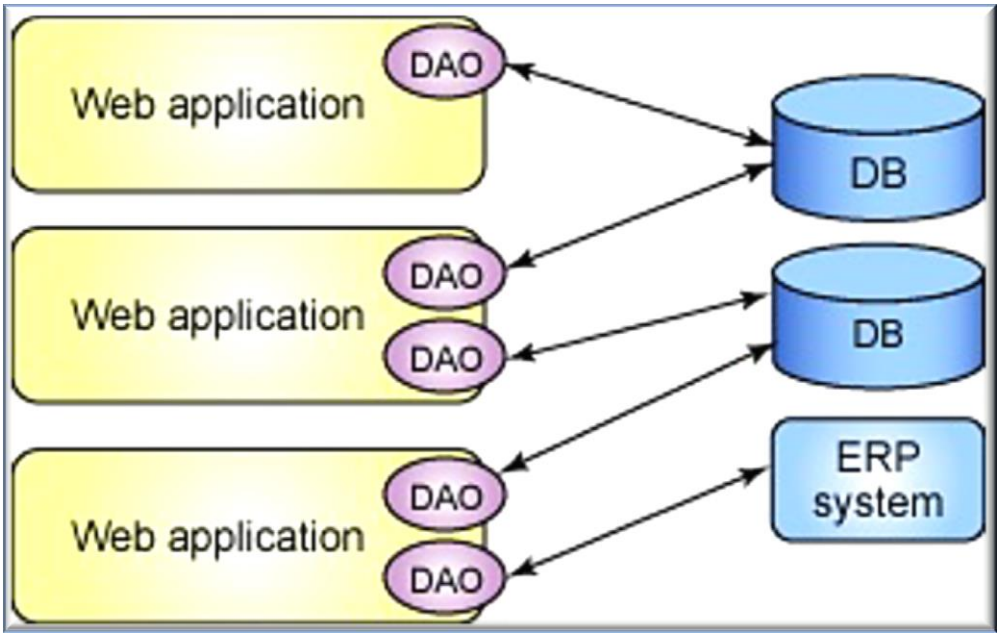
Il principale design pattern che abbiamo adottato è l' **MVC** (Model View Controller).
Questo pattern architetturale è molto usato nella programmazione orientata agli oggetti ed è basato sulla separazione delle tre principali componenti del sistema:

- **Model :** Fornisce i metodi per accedere ai dati utili all'applicazione.
- **View :** visualizza i dati contenuti nel Model e si occupa dell'interazione con gli utenti
- **Controller :** riceve i comandi dell'utente e li attua modificando gli stati delle altre due componenti.



Gestendo la nostra applicazione con questo pattern cerchiamo di rendere le tre componenti descritte quanto più indipendenti fra loro.
Così facendo, se ad esempio si decidesse di modificare la parte View della nostra applicazione, andremo ad agire quasi esclusivamente su di essa, senza apportare modifiche alle altre componenti.

Un altro pattern architetturale utilizzato dal nostro team è stato il **DAO (Data Access Object).**
Questo pattern è stato scelto soprattutto per stratificare e isolare l'accesso ad una tabella tramite query.
Le classi DAO presenti nella nostra applicazione web avranno il compito di comunicare con il DB e gestire quindi il codice SQL utilizzato per le query; in questo modo quindi isoliamo la gestione dell'SQL senza l'interazione delle componenti Model e Controller.



Librerie utilizzate

Apache Tomcat: è un application server nella forma di contenitore servlet open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e Servlet, fornendo quindi una piattaforma softwareper l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

FreeMarker: motore di template utilizzato per fare da mediatore tra la logica computazione in Java e l'HTML.

Apache Commons : libreria necessaria (alternativa a tomcat) per il salvataggio locale dei file. E' stata utilizzata in particolare per il salvataggio dei file immagine relativi alle opere.

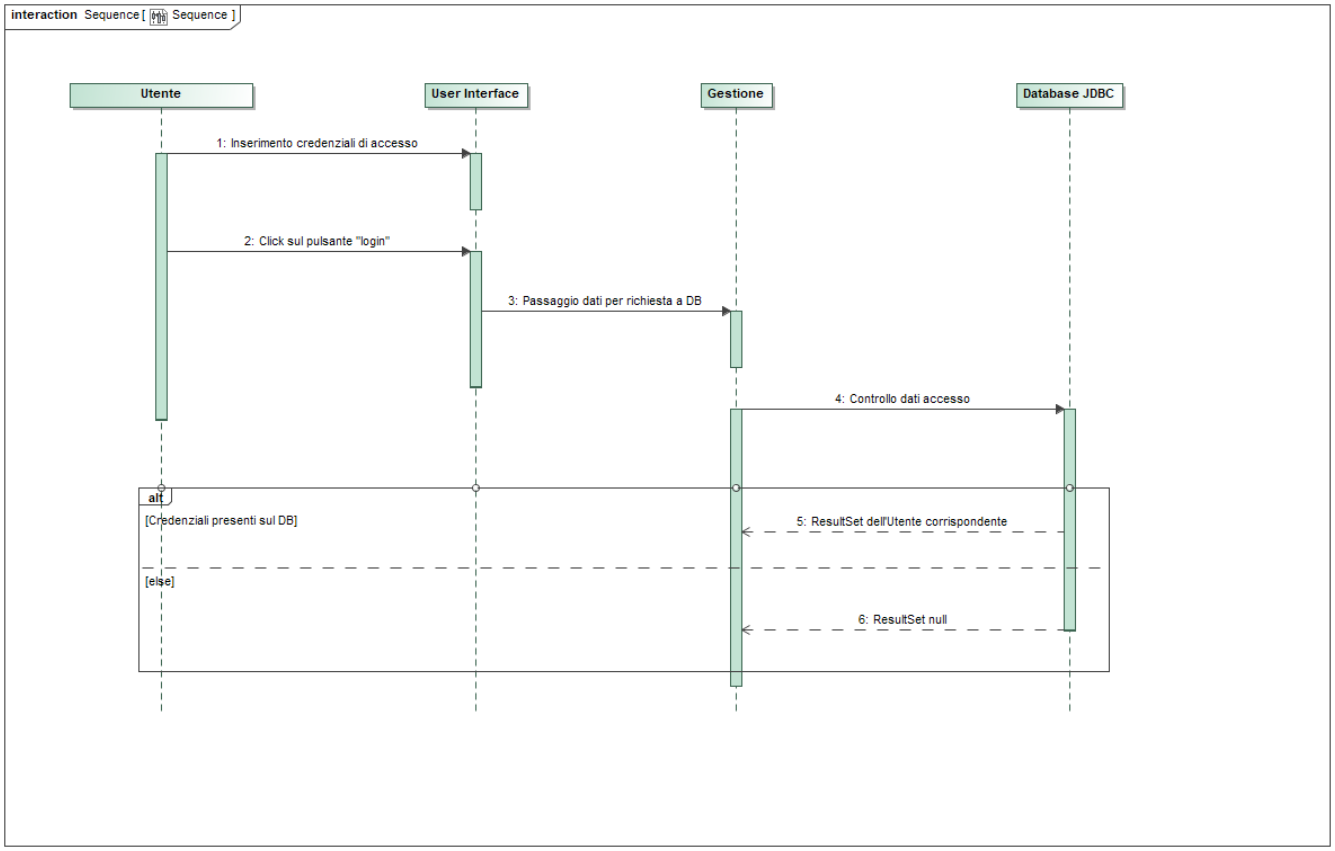
MySql Connector : utilizzato per interagire con il nostro JDBC, in particolare utilizzandone uno esterno al nostro ambiente di sviluppo (Eclipse). Nel nostro caso abbiamo utilizzato PhpMyAdmin integrato in WAMP.

Sequence Diagram

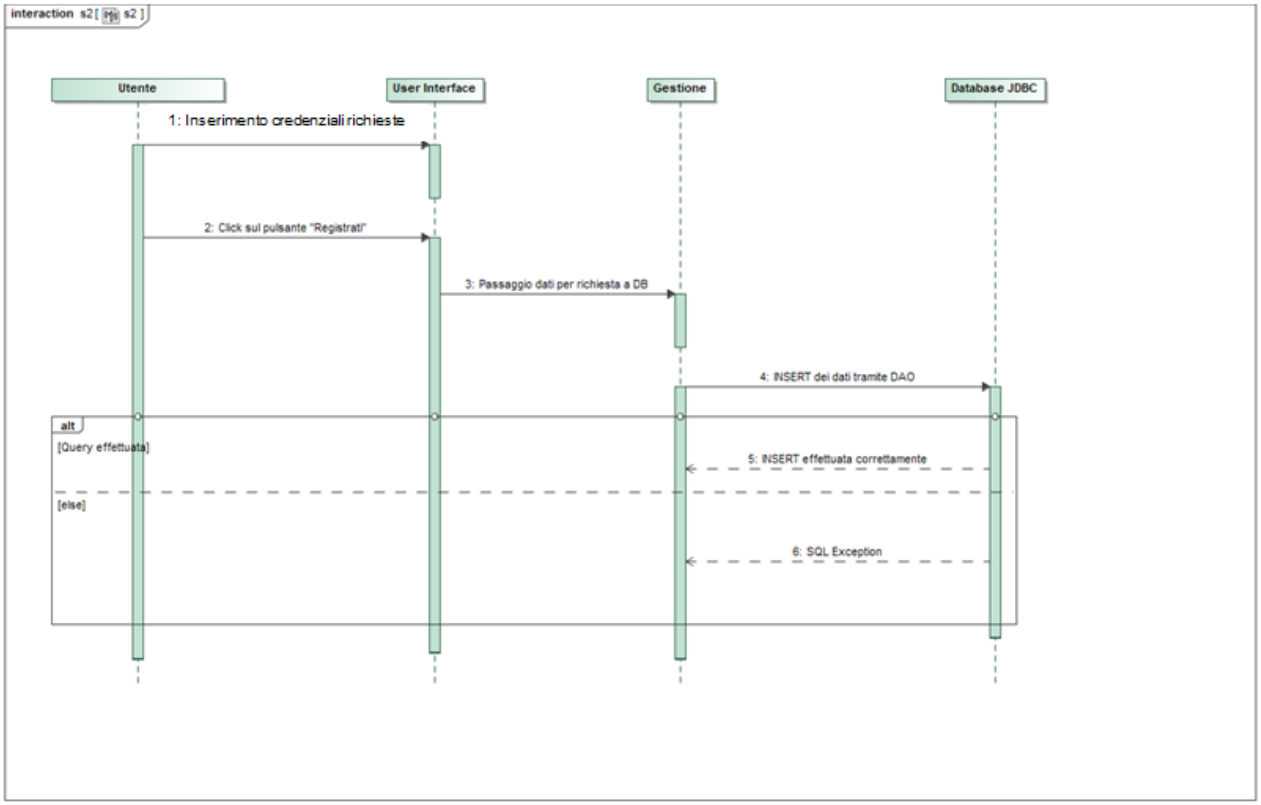
Il Sequence Diagram è un diagramma previsto dall'UML utilizzato per descrivere uno scenario, ossia una determinata sequenza di azioni compiute dagli attori del nostro sistema in cui tutte le scelte sono già state effettuate.

Mostriamo di seguito una serie di possibili scenari.

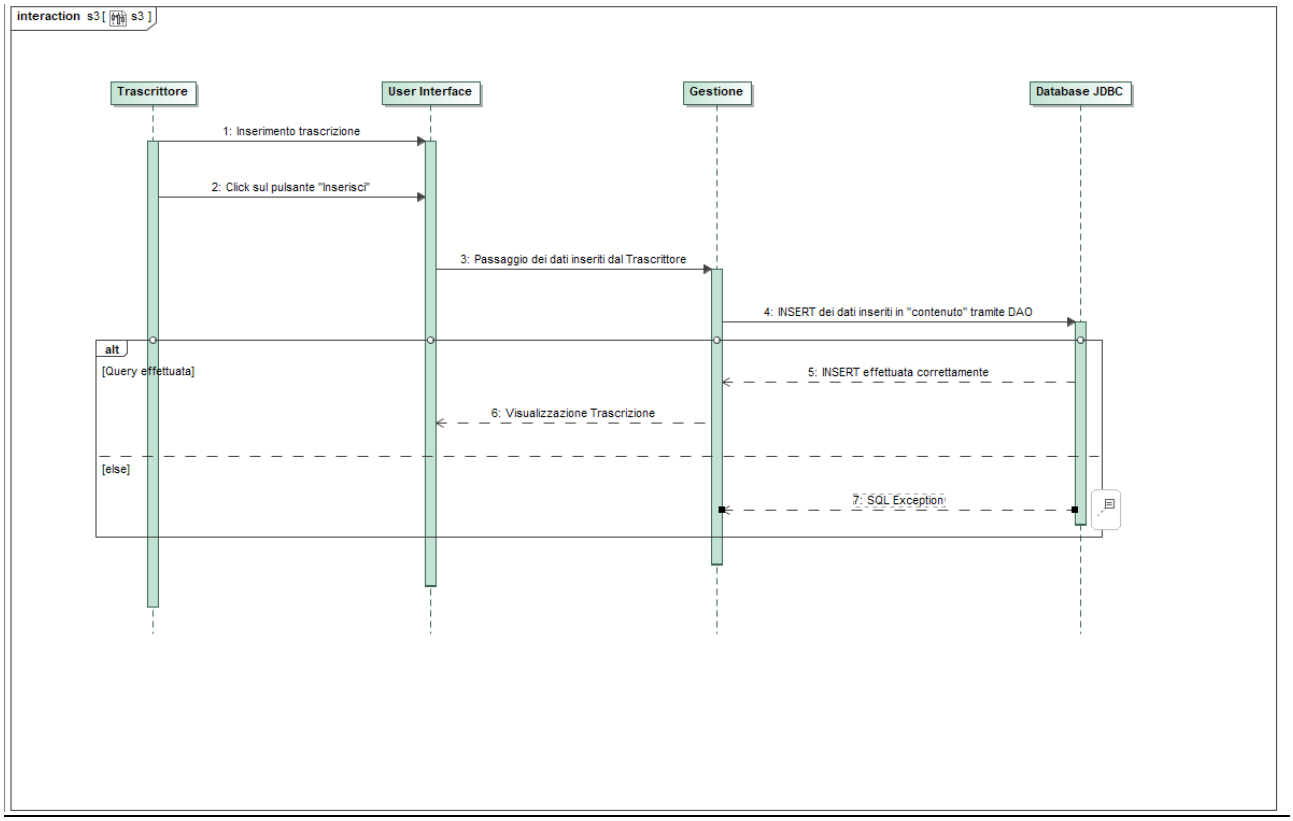
LOGIN



REGISTRAZIONE



TRASCRIZIONE

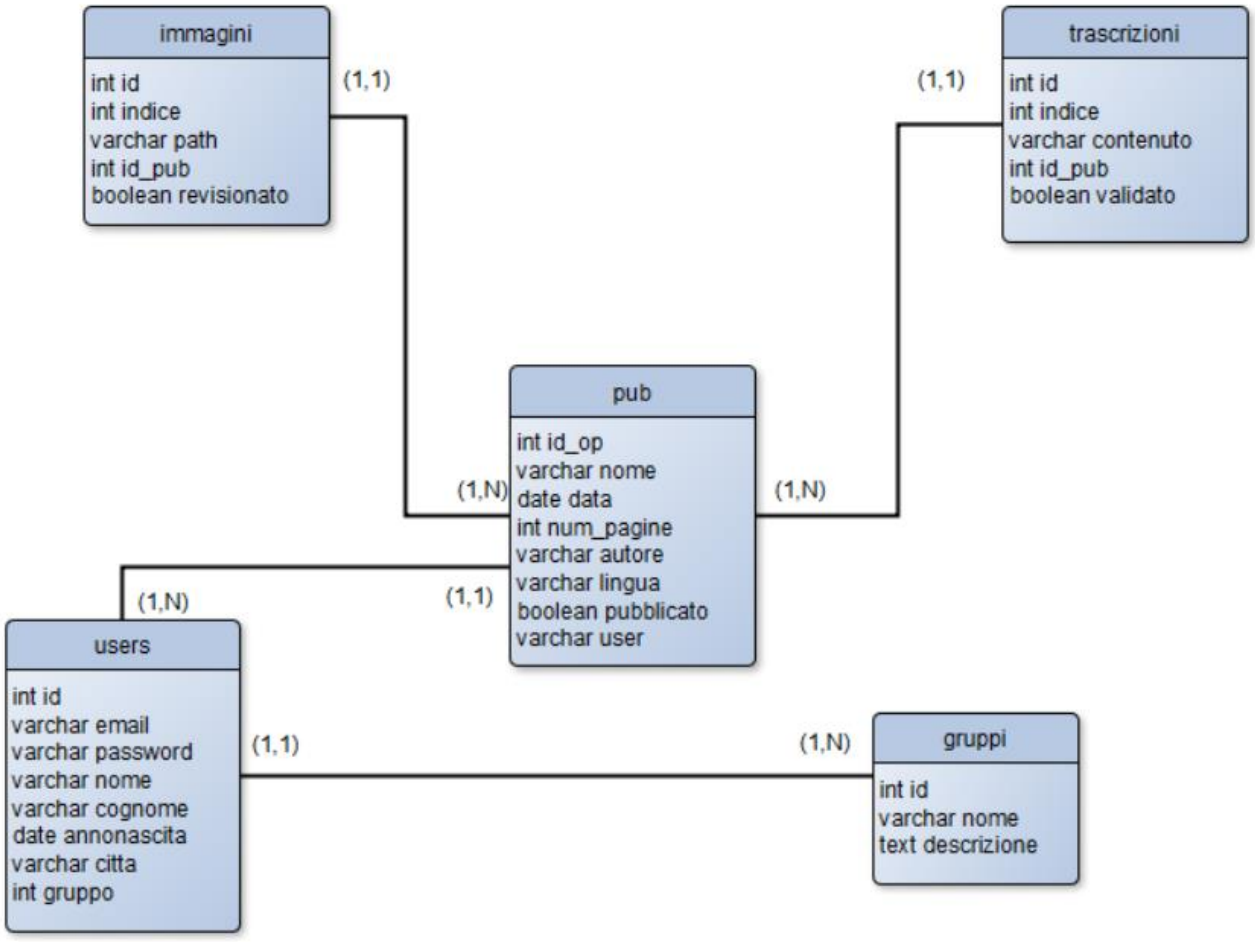


Modello ER

Una scelta effettuata dal nostro team riguardo la struttura del nostro database è stata quella di separare gli Utenti dai Gruppi, in modo che la struttura della nostra applicazione risulti più pulita.
Sono stati assegnati degli Id ai gruppi per distinguere appunto il gruppo di appartenenza.

I gruppi risultanti sono:

Id	Gruppo
1	Amministratore
2	Utente Base
3	Trascrittore
4	Revisore Trascrizioni
5	Acquisitore
6	Revisore Acquisizioni
7	Utente Avanzato



Descrizione del modello ER

Users
Rappresenta un generico utente del sistema; è caratterizzato da un identificatore, l'Id, da dati anagrafici quali Nome, Cognome, Data di nascita e città, dai campi Username e Password, utilizzati come credenziali di accesso e dal Gruppo di appartenenza.
E' in relazione con la tabella Gruppi perché in base al gruppo di appartenenza avrà a disposizione determinate funzionalità all'interno del sistema.

Gruppi
Rappresentano i gruppi di appartenenza del sistema, caratterizzati come spiegato in precedenza da un identificatore (da 1 a 7) per suddividere i gruppi, da un nome e da una descrizione.

Pub
Rappresenta l'opera nel dettaglio, identificata da un Id, da un nome, una data di pubblicazione, il numero delle pagine, l'autore, la lingua, una flag di pubblicazione e l'utente che l'ha inserita.

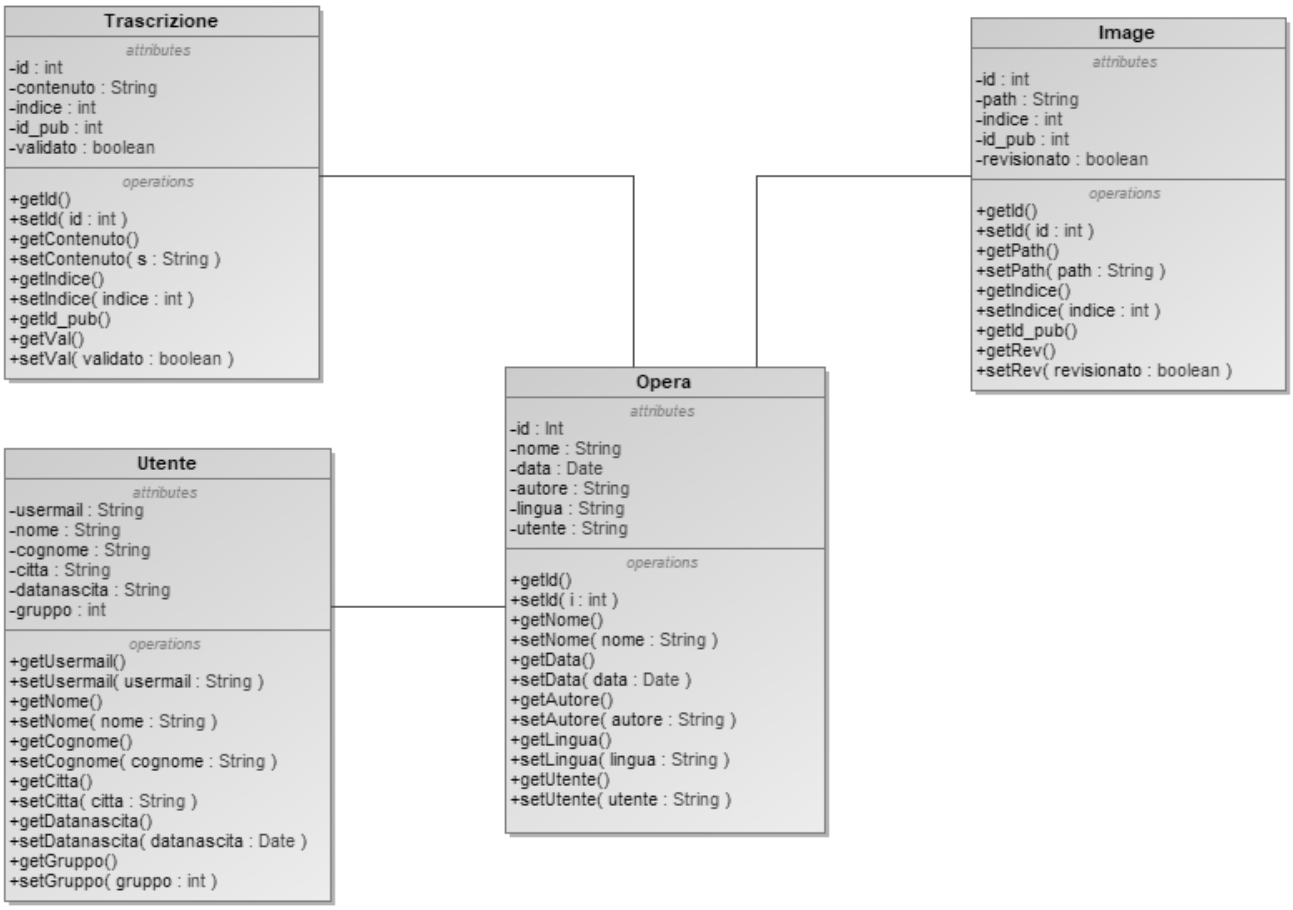
Immagini
Questa tabella rappresenta un istanza di tipo immagine; è caratterizzata da un attributo identificativo, Id, dal path dell'immagine, dall'indice della pagina, da un booleano per indicare lo stato di revisione e dall'id dell'opera a cui si riferisce.

Trascrizioni
Rappresenta appunto un istanza della porzione di testo che identifica una pagina di un'opera.
Oltre ad essere caratterizzata da un Id, una trascrizione è composta anche da un contenuto di tipo String, dall'indice della pagina, da un booleano di validazione e dall'id dell'opera a cui si riferisce.

Software Design

Object Diagram

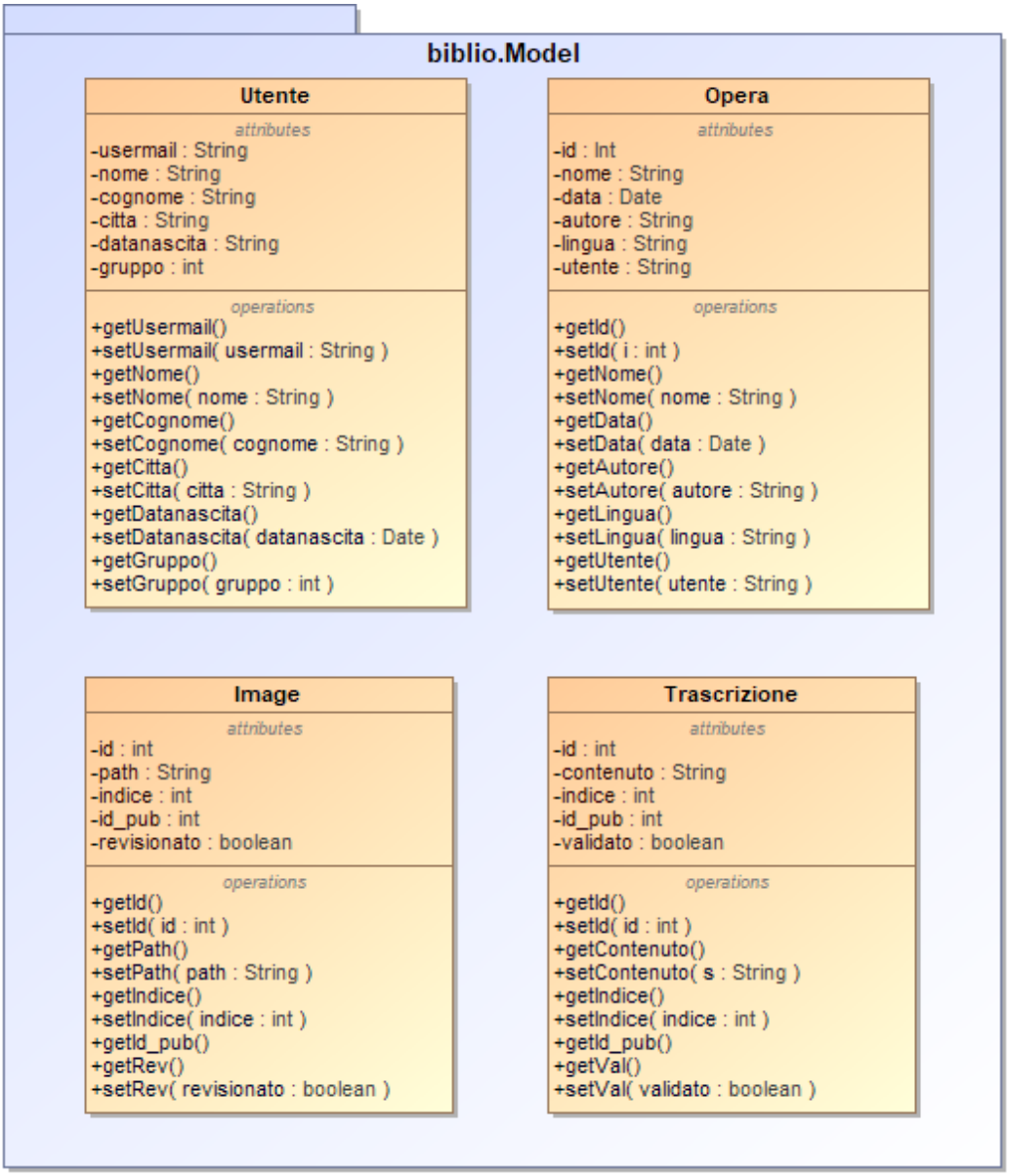
L'Object Diagram è un diagramma di tipo statico previsto dall'UML per descrivere un sistema in termini di oggetti e relative relazioni.
Il seguente diagramma rappresenta l' Object Diagram della nostra applicazione.



Class Diagram

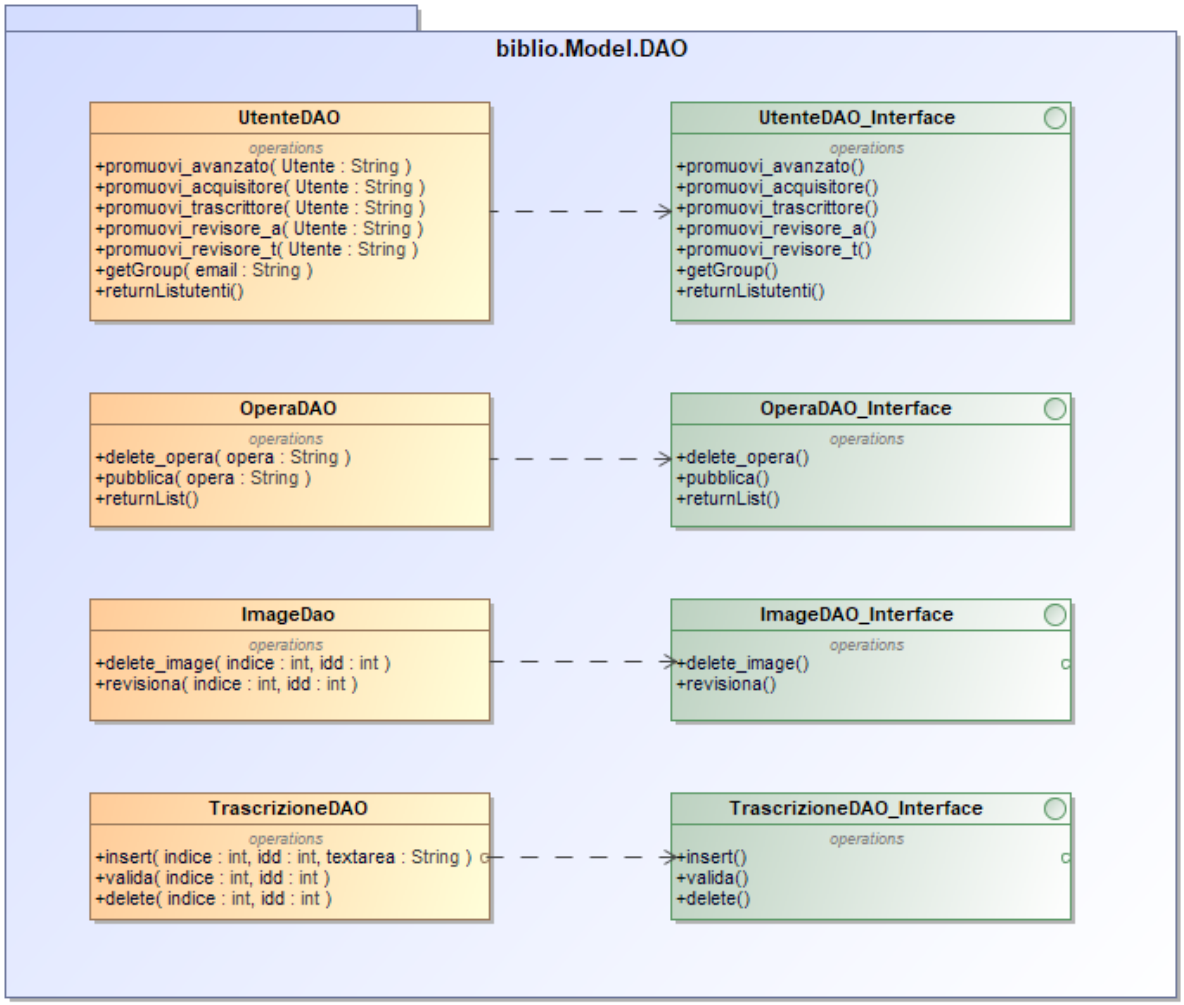
Il Class Diagram è un altro dei tipi di diagrammi che può comparire in un modello UML. In generale consentono di descrivere tipi di entità, quindi classi, con i propri attributi e metodi e le relazioni che intercorrono tra essi.
Il principale strumento concettuale utilizzato è appunto il concetto di classe del paradigma object oriented.

Class Diagram – Package Model



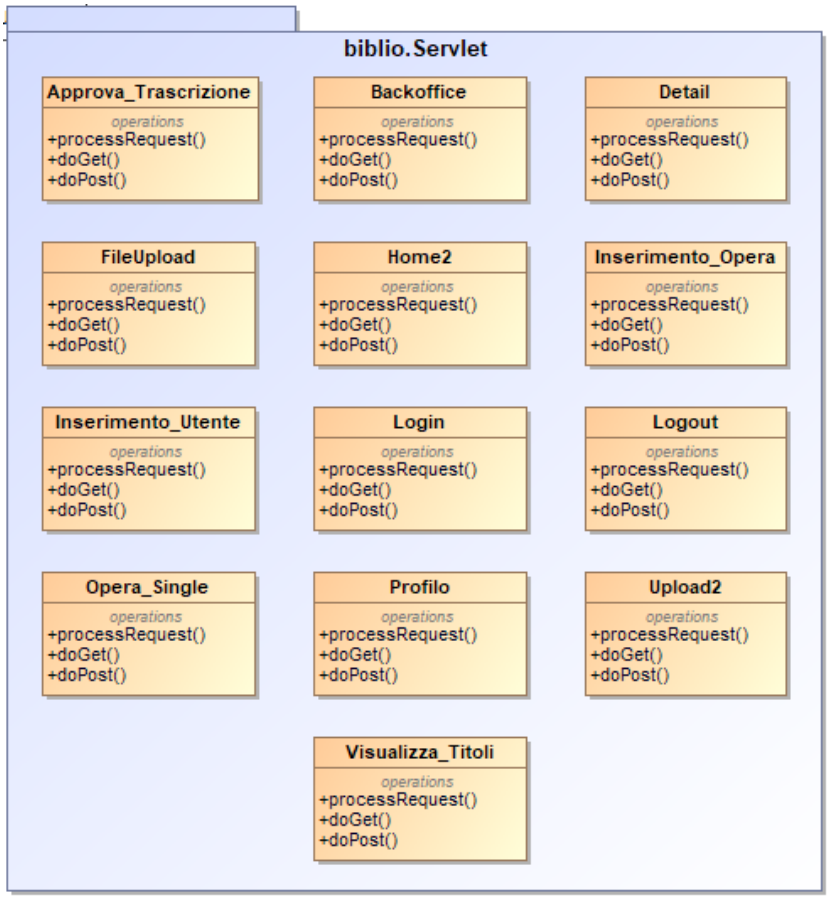
All'interno del package Model è presente un ulteriore package DAO, contenente i metodi per l'interazione con il DB, come descritto in precedenza.

Class Diagram – Package Model.DAO



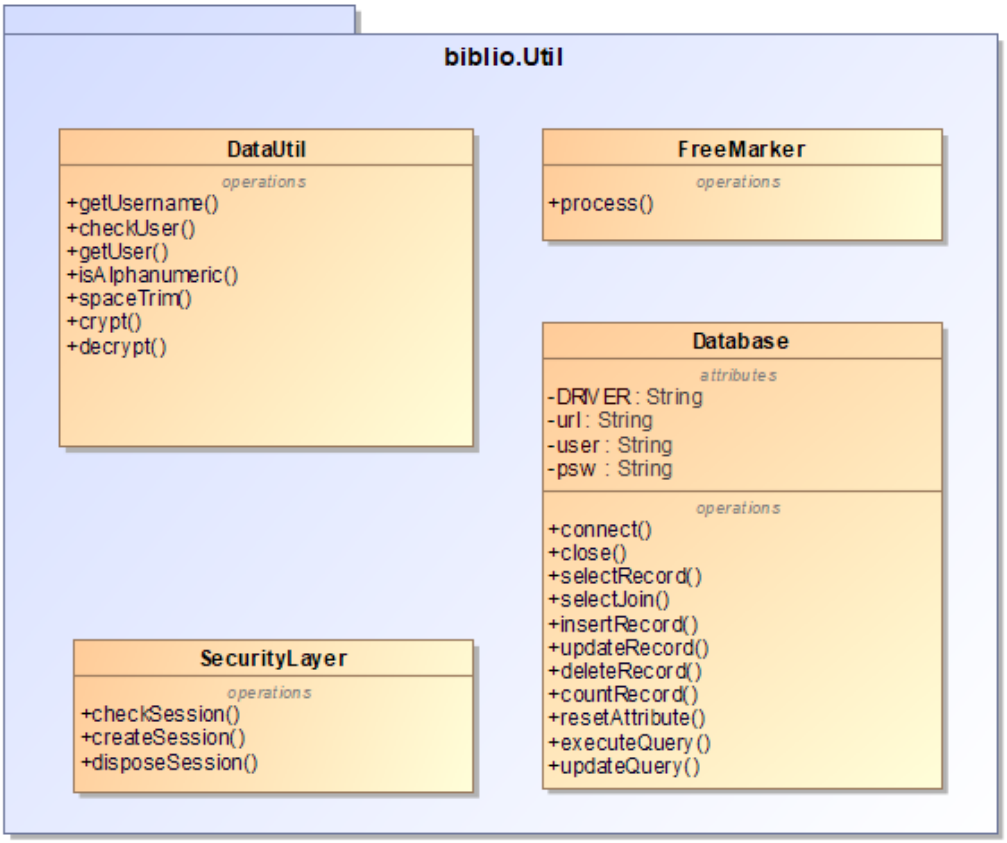
Questo Class Diagram rappresenta il package DAO, all'interno del Model.
In questo Package abbiamo fatto uso delle interfacce; esse conterranno, come da definizione, solo la dichiarazione dei metodi, implementati poi nelle rispettive classi.
Abbiamo optato per questa scelta implementativa sia per renderci il codice più chiaro e pulito in fase di costruzione del progetto, sia per permettere a futuri sviluppatori di avere una visione più strutturata del sistema.

Class Diagram – Package Servlet



Il diagramma sopra raffigurato rappresenta il package relativo alle Servlet della nostra applicazione.
I metodi di ciascuna classe sono il “processRequest”, il “doGet” e il “doPost”, cioè i metodi di una struttura classica di una Servlet.

Class Diagram – Package Util



Il diagramma rappresenta il package Utils, contenente le classi Database, FreeMarker, DataUtil e SecurityLayer.

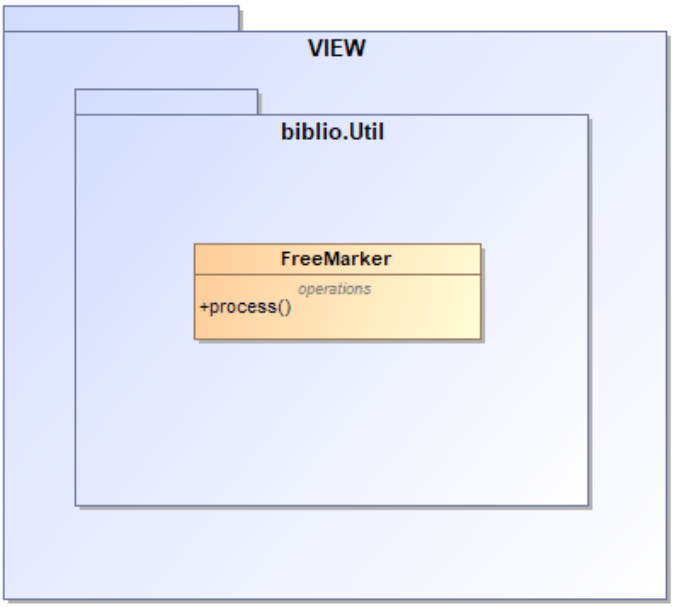
- **Database** contiene tutti i metodi di connessione e relazione con il DB; abbiamo creato questa classe in modo da non riscrivere i metodi ogni volta; così facendo abbiamo un codice meno ridondante.

- **FreeMarker** è la classe che corrisponderà alla nostra View; costiuisce infatti il nostro motore che girerà dietro la View e permetterà il passaggio di dati verso l'html.

- **DataUtil** contiene metodi più generali ma sempre utili al sistema. Costrire questa classe era comunque un modo migliore di in termini di struttura del sistema.

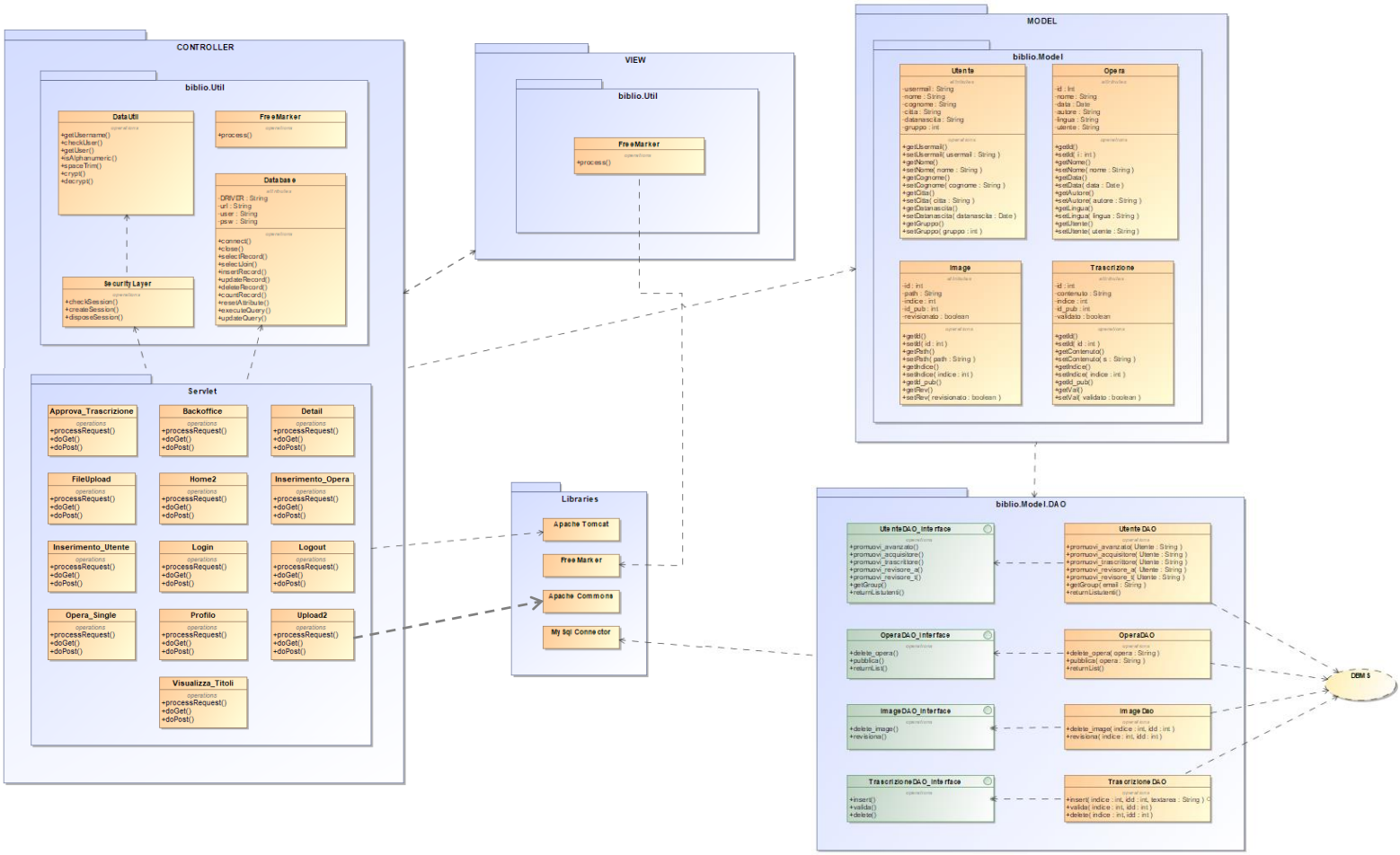
- **Security Layer** infine è la classe che utilizziamo per gestire le sessioni.

Class Diagram – View



Questo diagram non rappresenta altro che la nostra componente View del modello MVC adottato.
Lavora come mediatore tra l'utente e il controller; prende infatti dati in input e modellandoli riesce a mandarli a video sfruttando la classe FreeMarker, in particolare richiamando il metodo "process" contenuto in essa.

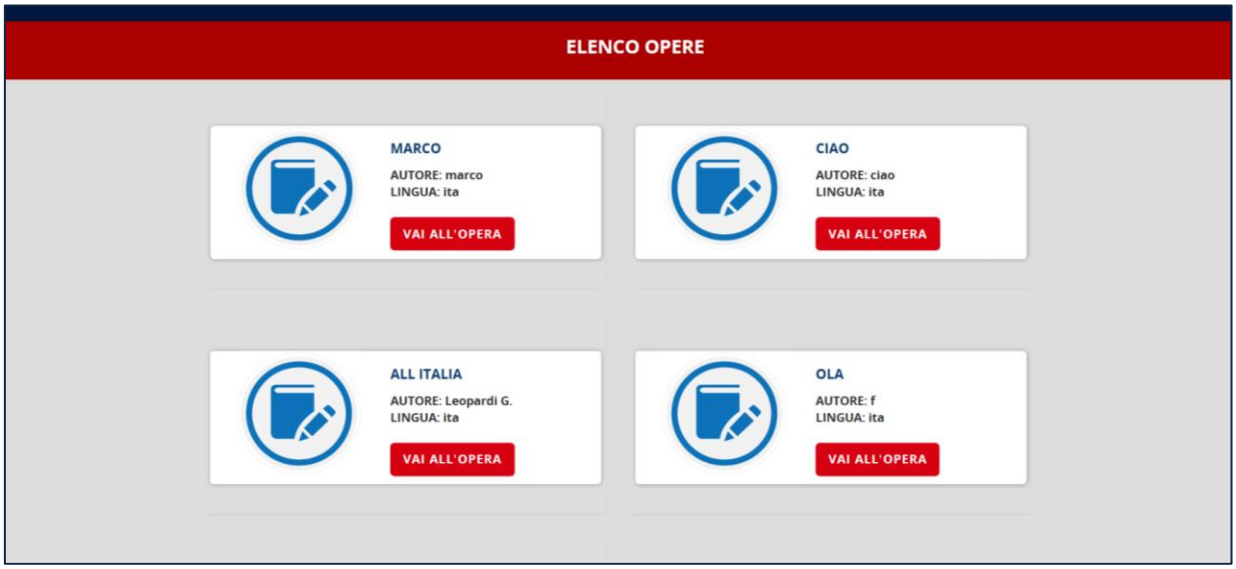
Class Diagram – SISTEMA COMPLETO



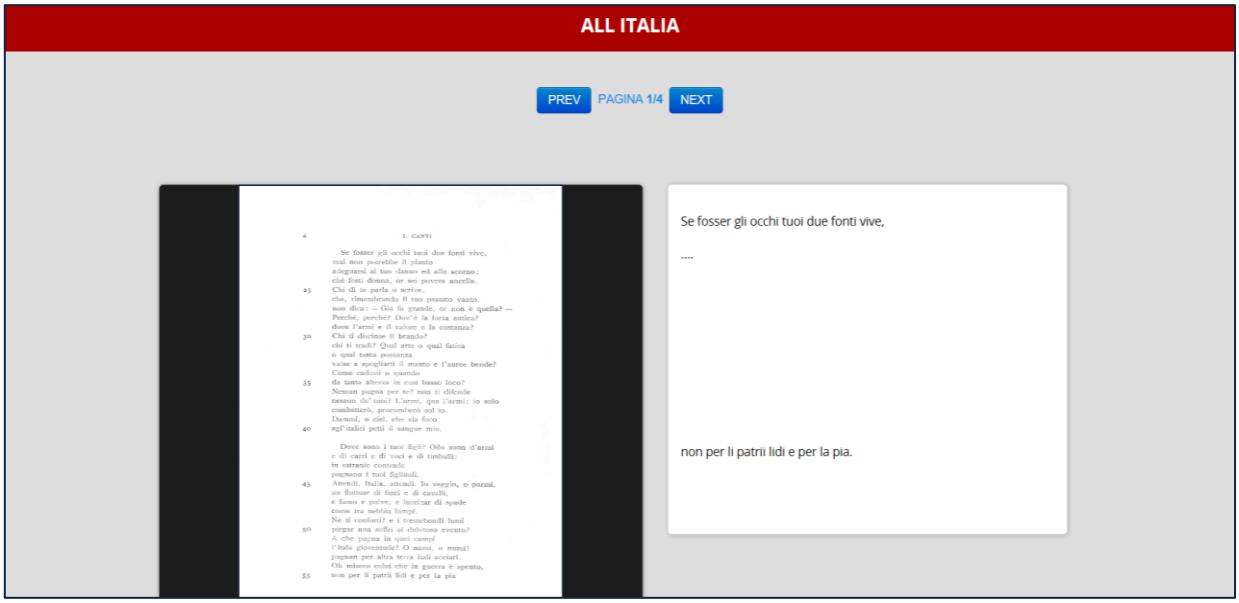
Le seguenti immagini rappresentano alcune view della nostra applicazione web.

Login – Registrazione

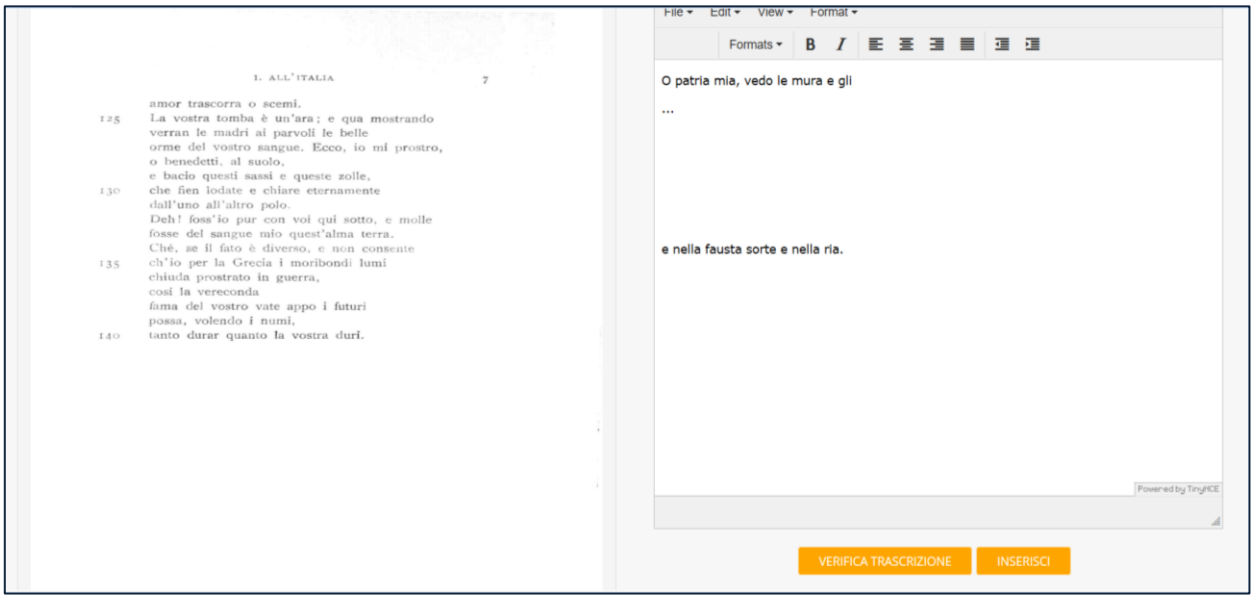
Pagina principale



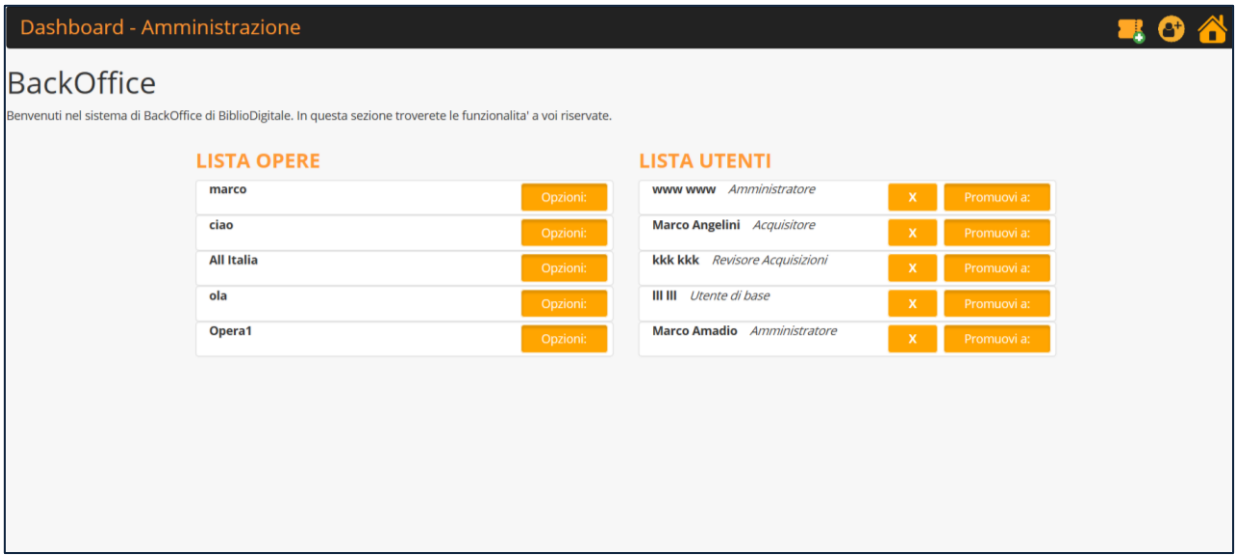
Visualizzazione opera



Visualizzazione opera (Backoffice)



Pagina principale Backoffice



Inserimento di un' opera (Backoffice)