

# DESIGN DOC

## Object Oriented Programming

### **Progetto Biblioteca Virtuale**

Patrizio Pezzilli 227636

Stefano Cortellessa 227630

Luca Ferrari 229666

Repository GitHub: [Clicca Qui](#)

Per dubbi o domande relative alla documentazione: [Clicca Qui](#)

# INDICE

## - Requirements

- Requisiti Funzionali ..... (pag.3)
- Requisiti Non Funzionali ..... (pag.4)
- Attori e UseCase del sistema ..... (pag.5)
- Use Case Diagram ..... (pag.6)
- Scenari ..... (pag.9)
- Descrizione UseCase ad alta priorità ..... (pag.10)

## - System Design

- Modello dell'Architettura del sistema ..... (pag.12)
- Descrizione dell'Architettura ..... (pag.13)
- Descrizione delle scelte ..... (pag.14)
- Design Pattern ..... (pag.16)
- Librerie Utilizzate ..... (pag.17)
- Modello completo dell'Architettura del Sistema ..... (pag.18)
- Sequence Diagram ..... (pag.19)
- Modello ER ..... (pag.25)
- Osservazione ..... (pag.26)

## - Software Design

- Modello Object Diagram ..... (pag.27)
- Descrizione Object Diagram ..... (pag.28)
- Modello e Descrizione Class Diagram ..... (pag.29)
- Class Diagram completo ..... (pag.34)

## - Template Explanation

- Sitemap ..... (pag.35)
- Rappresentazione Sito Web ..... (pag.36)

# Requirements



## Requisiti Funzionali (FR)

Le priorità ai requisiti funzionali saranno assegnate su scala da 1 a 5 (**1: importanza nulla, 5: massima importanza**).

Quelli con priorità alta verranno descritti nel dettaglio nella pagina 7 sotto forma tabellare.

- **Login/Registrazione (5):** Funzione senza la quale un utente non potrà accedere al sistema.
- **Visualizzazione Elenco Titoli (3):** Funzionalità che mostra l'elenco di tutti i testi disponibili nella biblioteca.
- **Visualizzazione Completa Opere (3):** Disponibile solo per l'Utente Avanzato e permette la visualizzazione delle immagini e le trascrizioni di una specifica opera.
- **Visualizzazione Profilo Personale (1):** Disponibile per tutti gli utenti, permette la visualizzazione dei dati personali e nell'eventualità in cui tu sia amministratore le opere inserite.
- **Inserimento su Database (4):** Disponibile solo al Trascrittore ed è la funzione che permette il passaggio su database del testo relativo ad un'immagine di una determinata opera.
- **Validazione (4):** Permette al Revisore Trascrizioni di validare una trascrizione inserita.
- **Eliminazione (Gestione Testo) (4):** Permette al Revisore Trascrizioni di eliminare una trascrizione inserita, in base alle proprie conoscenze sulla sintassi TEI o in base a contenuti particolarmente errati.
- **Visualizzazione Editor TEI esterno (4):** Funzione disponibile al Revisore Trascrizione che consente di aprire la trascrizione attuale in una pagina esterna all'applicazione in cui avrà a disposizione l'editor TEI completo di trascrizione e sintassi del linguaggio. Da tale pagina sarà possibile approvarlo o eliminarlo.

- **Upload file Immagine (4):** Funzione di inserimento dell'immagine di una o più pagine di un testo da parte dell'acquisitore.
- **Revisione (4):** Permette al Revisore Acquisizioni di validare una o più immagini inserite.
- **Eliminazione (Gestione Immagini) (4):** Permette al Revisore Acquisizioni di eliminare una o più immagini inserite, in base alla qualità di quest'ultima.
- **Pubblicazione (5):** Consente all'amministratore del sistema di rendere disponibile agli Utenti Avanzati una o più opere digitalizzate ed eventualmente trascritte.
- **Inserimento (Gestione Opere) (5):** Funzionalità che permette l'aggiunta di una nuova opera, formata da immagini e trascrizioni e resa disponibile al proprio completamento.
- **Eliminazione (Gestione Opere) (3):** Funzionalità che permette l'eliminazione di un'opera.
- **Promozione Ruolo (Gestione Utenza) (4):** Funzionalità che permette all'amministratore di promuovere un'utente ad un ruolo superiore o inferiore. Sarà quindi possibile crearsi il proprio team di revisori e di acquisitori, nonché di trascrittori e utenti avanzati.
- **Inserimento (Gestione Utenza) (2/3):** Funzionalità che permette l'aggiunta di un nuovo utente.

## Requisiti Non Funzionali (NFR)

- **Usability:** Il sistema dovrà essere pulito e di facile utilizzo.
- **Reliability:** Il portale dovrà garantire all'utente le funzioni messe a disposizione (vedi Use Case Diagram) senza errori.
- **Availability:** Il portale dovrà essere sempre disponibile e deve poter garantire in qualsiasi momento tutte le funzioni desiderate.

## Attori e UseCase del sistema

Il primo attore identificato nel nostro sistema è l'**Amministratore**, con le funzioni di gestione generale del sistema, quali inserimento, modifica e cancellazione titoli, nonché la modifica del grado di un utente selezionato. Successivamente troviamo l'**Acquisitore** e il **Trascrittore**, con i relativi compiti di acquisizione e digitalizzazione dell'immagine per il primo e trascrizione del testo TEI per il secondo.

In parallelo a questi ultimi vi saranno di conseguenza il **Revisore acquisizioni** e il **Revisore trascrizioni**, che si occuperanno della verifica della correttezza degli inserimenti avvenuti.

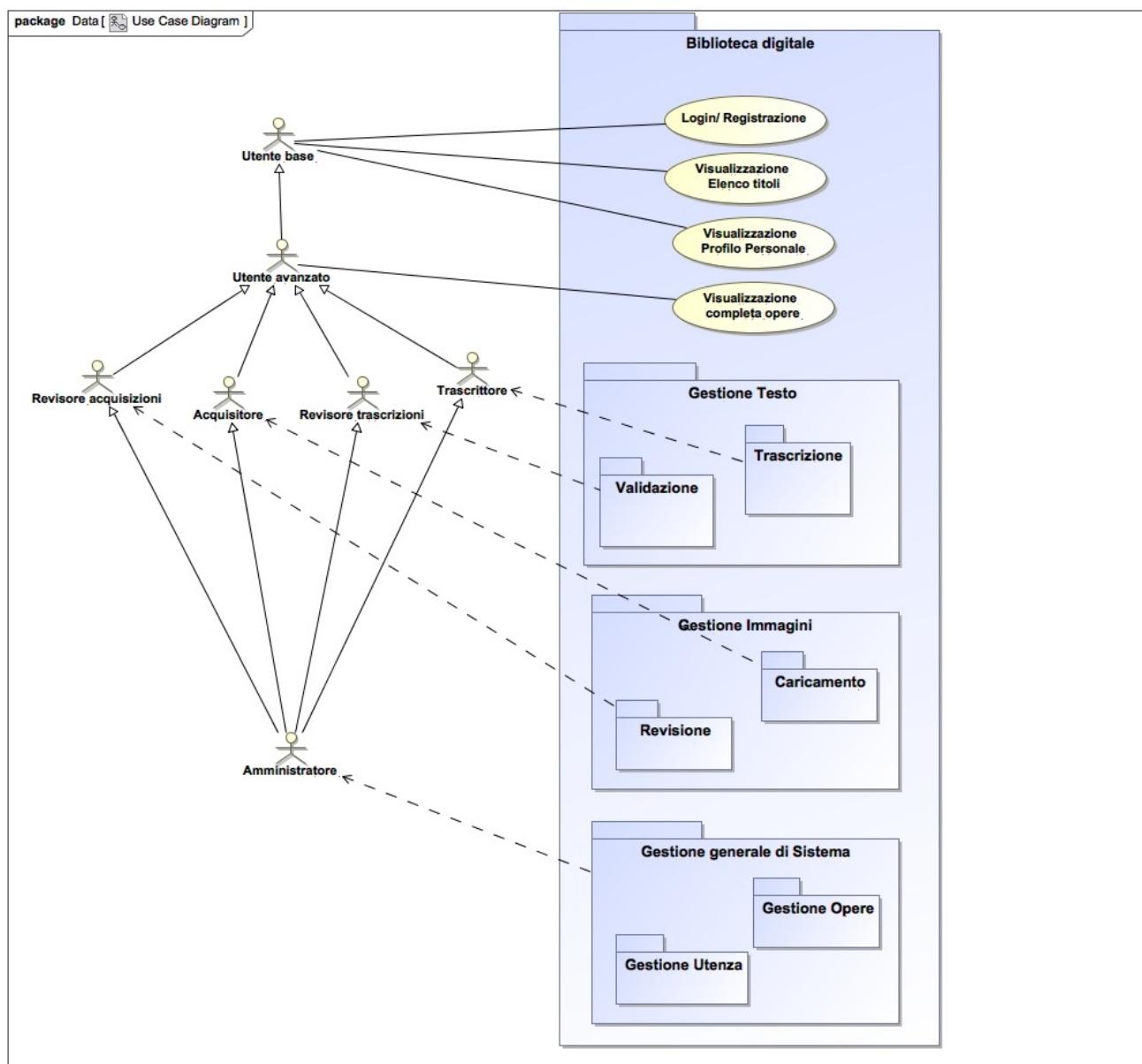
Ad alto livello infine troviamo l'**Utente Base** con compiti di pura consultazione dell'elenco dei libri, e l'**Utente Avanzato** con il privilegio di poter visualizzare completamente un'opera.

## Use Case Diagram

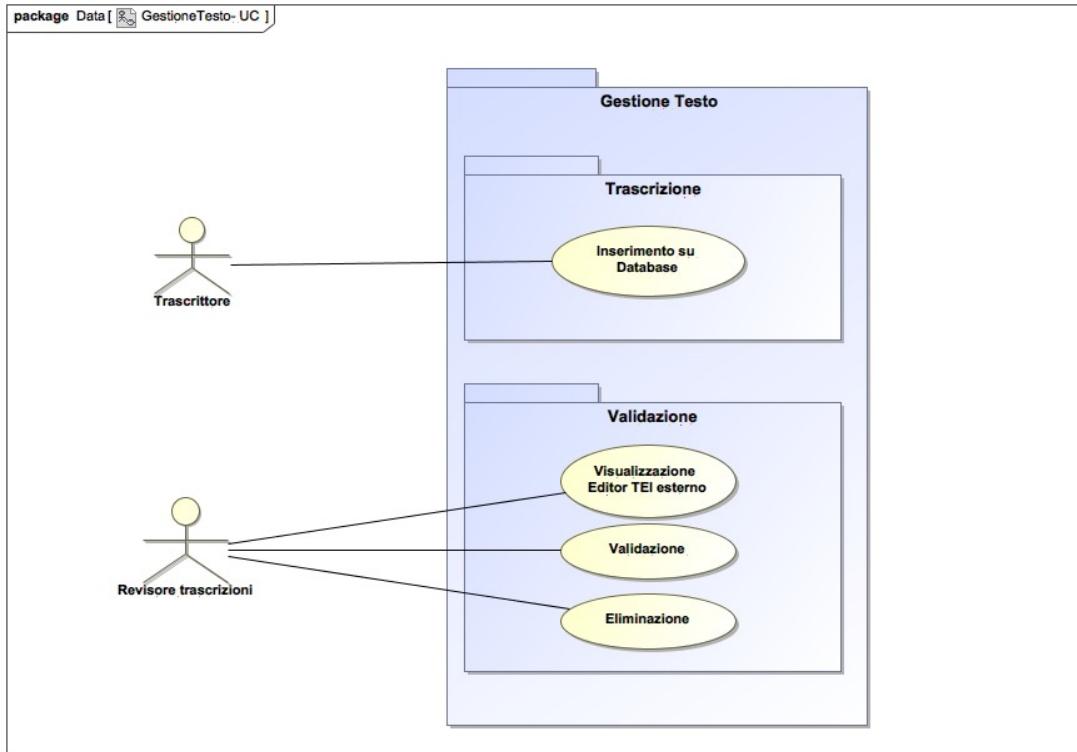
Si è preferito suddividere il diagramma principale in più parti, per una migliore leggibilità e comprensibilità del nostro sistema.

Di seguito viene riportato il diagramma principale e successivamente quelli specifici relativi alla gestione delle trascrizioni e della gestione delle immagini. Infine sarà riportato il diagramma relativo alla gestione generale del sistema.

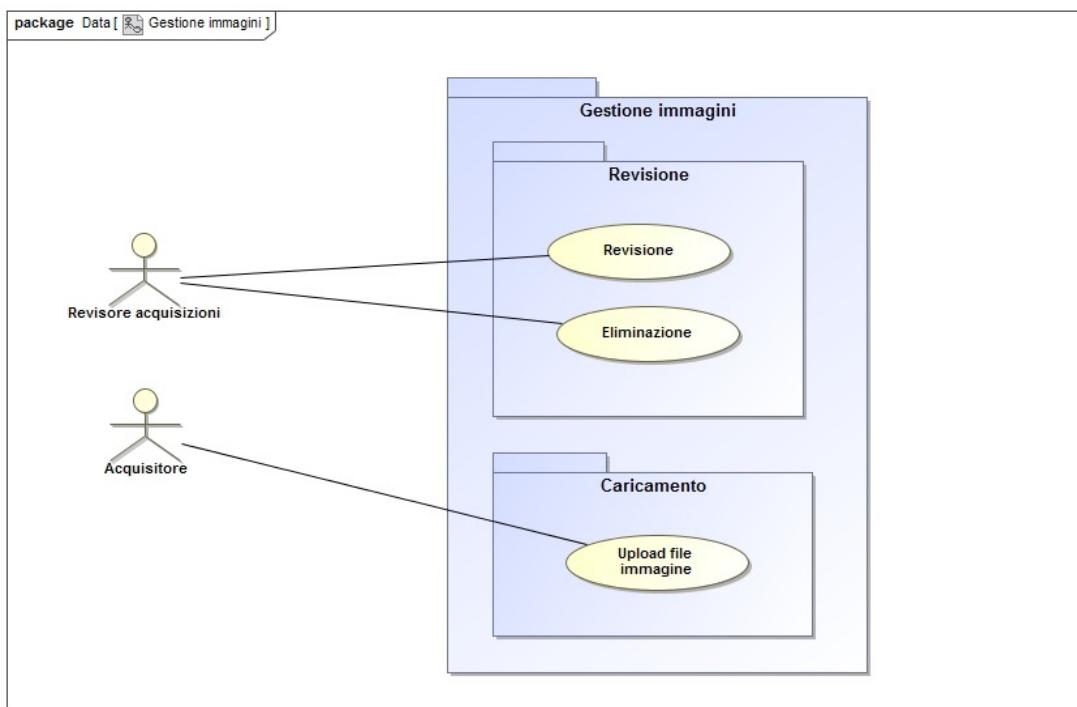
### Use Case Diagram principale:



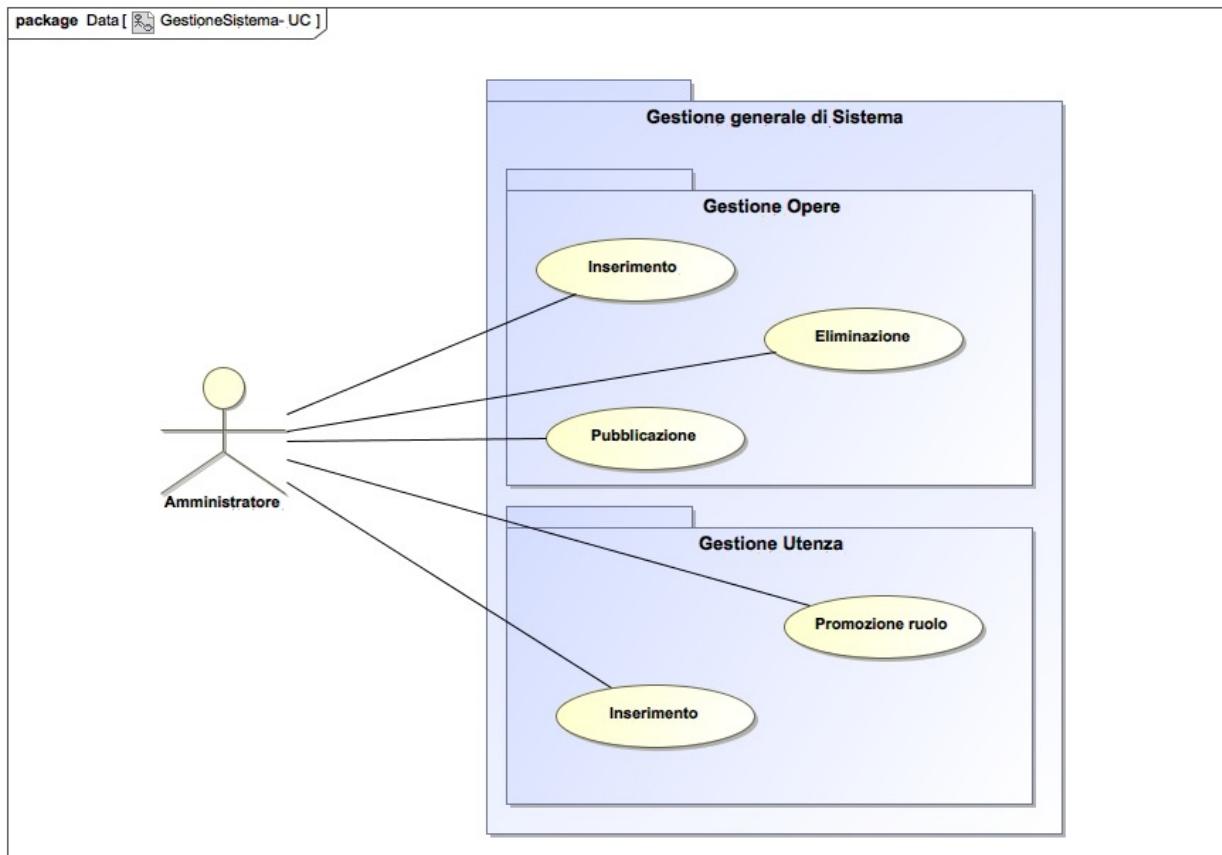
**Use Case Diagram “Gestione Testo”:**



**Use Case Diagram “Gestione Immagini”:**



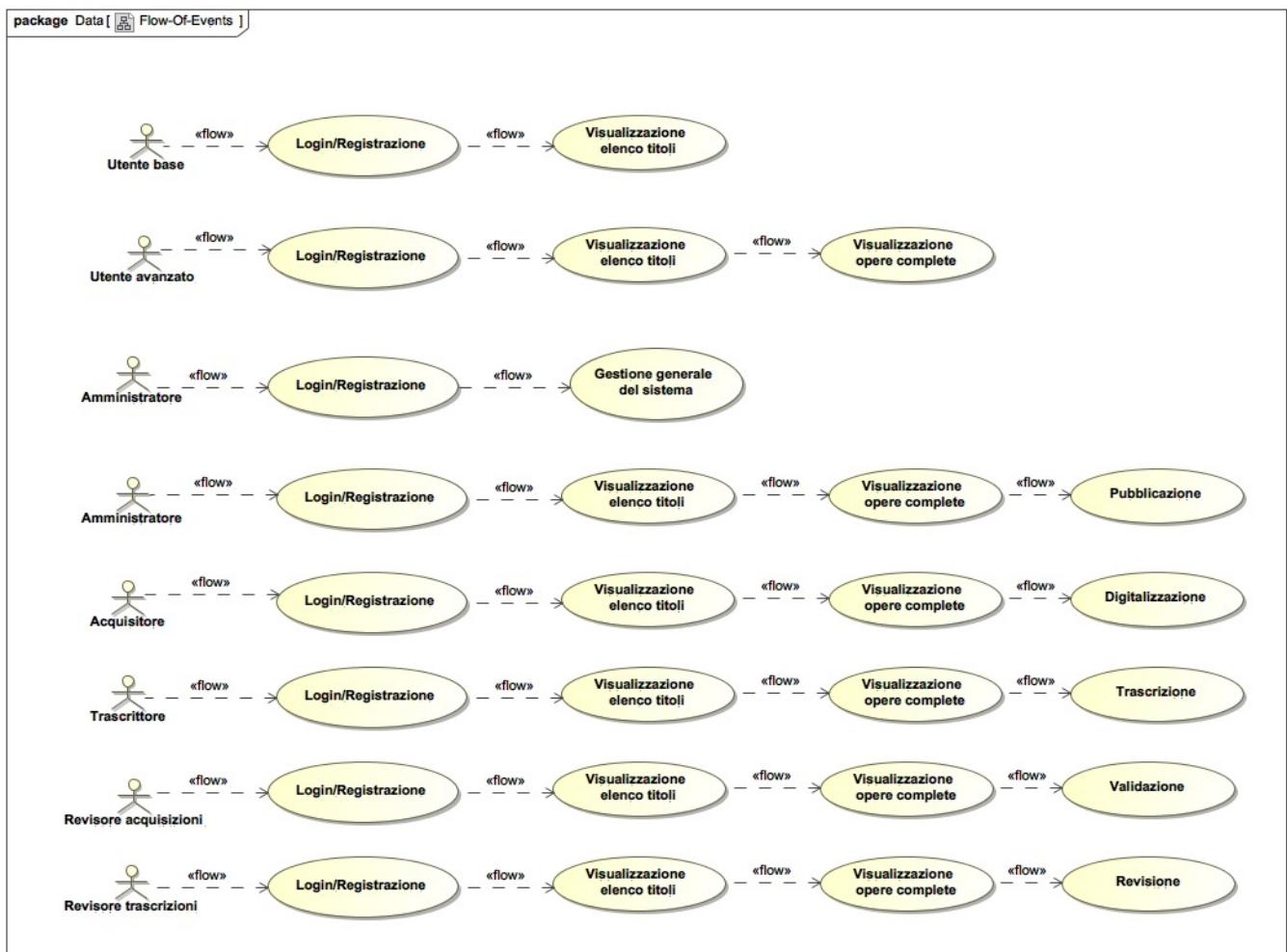
**Use Case Diagram “Gestione generale di Sistema”:**



## Scenari

Nei seguenti flussi di eventi verranno rappresentati in sequenza l'utilizzo degli UC (Use-case) da parte di ogni tipologia di attore del nostro sistema.

Questi comporranno gli scenari che ci serviranno successivamente nell'analisi.



## Descrizione UseCase ad alta priorità

Use Case name	<b>Login / Registrazione</b>
Partecipating actors	Utente Base
Descrizione	Permette ad un utente registrato di accedere al sistema o ad un utente non registrato di registrarsi.
Evento Scatenante	Click da parte dell'utente sul relativo pulsante
Uses	Login/Registrazione effettuati con successo o errore

Use Case name	<b>Trascrizione</b>
Partecipating actors	Trascrittore
Descrizione	Permette all'attore l'inserimento del testo relativo ad una pagina di una pubblicazione
Evento Scatenante	Inserimento testo da parte dell'utente nel form di input
Uses	Inserimento effettuato correttamente o meno, in attesa di validazione

Use Case name	<b>Validazione</b>
Partecipating actors	Revisore Trascrizioni
Descrizione	Permette all'attore di verificare un testo ed eventualmente validarlo, giudicandolo corretto
Evento Scatenante	Click da parte del Revisore sul pulsante "valida"
Uses	Testo validato correttamente, o errore in caso di problema nel sistema. In attesa di pubblicazione

Use Case name	Digitalizzazione
Partecipating actors	Acquisitore
Descrizione	Funzione che permette l'acquisizione dell'immagine, una volta scannerizzata, in attesa di revisione
Evento Scatenante	Upload dell'immagine nel relativo form
Uses	Immagine caricata correttamente o meno, in attesa di revisione.

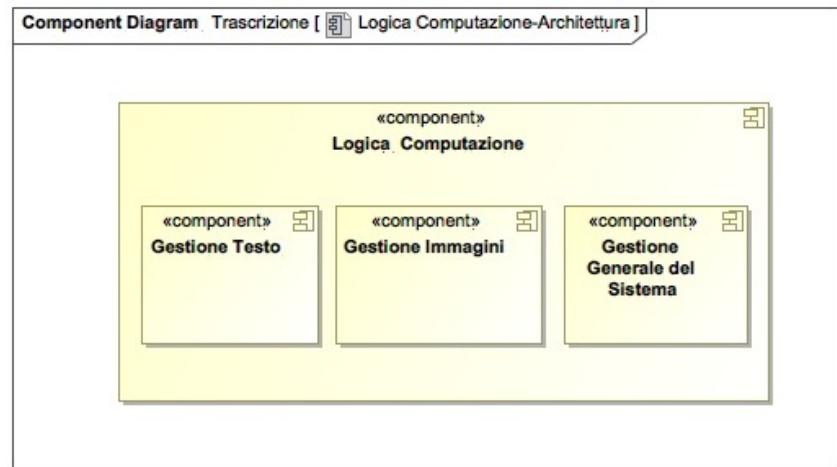
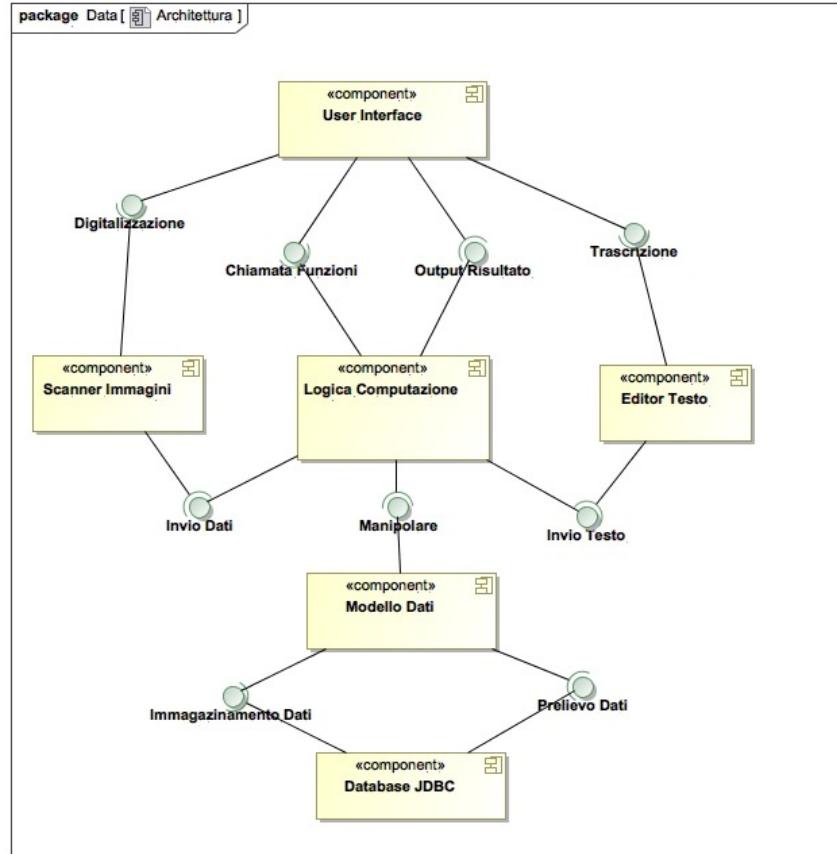
Use Case name	Revisione
Partecipating actors	Revisore Acquisizioni
Descrizione	Permette all'attore il controllo della qualità dell'immagine, e il successivo giudizio
Evento Scatenante	Click sul pulsante "revisiona"
Uses	Immagine revisionata correttamente, o errore in caso di problema nel sistema. In attesa di pubblicazione

Use Case name	Pubblicazione
Partecipating actors	Amministratore
Descrizione	Ultimo passo, consiste nell'approvazione finale da parte dell'amministratore riguardo una specifica opera non ancora pubblicata
Evento Scatenante	Click sul pulsante "pubblica"
Uses	Opera pubblicata correttamente

Use Case name	Inserimento (Gestione Opere)
Partecipating actors	Amministratore
Descrizione	Permette di aggiungere alla collezione una nuova opera, che verrà poi popolata da immagini e trascrizioni
Evento Scatenante	Click sul pulsante "insersci"
Uses	Opera inserita correttamente

# System Design

## Modello dell'Architettura del sistema



## Descrizione dell'Architettura

Dopo un confronto tra le componenti del gruppo si è deciso di optare per un implementazione web della Biblioteca Digitale.

Il sistema architetturale è stato decomposto in sette componenti principali: la **Logica Computazione**, il **Database JDBC**, la **User Interface**, il **Modello Dati**, lo **Scanner Immagini** e infine l'**Editor Testo**.

La User Interface rappresenta l'interazione tra il sistema e l'utente, in qualità di View, dove l'utente potrà immettere dati in input ed averne in output grazie alla logica che vi è dietro. Questa corrisponderà al template associato e configurato tramite motore di templating essendo l'applicativo sviluppato su piattaforma web.

La nostra View sarà quindi principalmente composta dalle classi del nostro motore di templating, nello specifico **FreeMarker**. In particolare il gruppo si è costruito una classe Java nel package “DataUtil” che non farà altro che utilizzare la libreria FreeMarker e aiutare nell’implementazione del portale.



The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The left pane shows a tree view of the project structure under "src". It includes packages like (default package), biblioteca.Model, biblioteca.Servlet, biblioteca.Util, and DataUtil. A file named "FreeMarker.java" is selected and highlighted in blue.
- Code Editor:** The right pane displays the content of "FreeMarker.java". The code is as follows:

```

17 * @author Patrizio
18 *
19 */
20 public class FreeMarker {
21 /**
22 *
23 * @param data           dati da inserire nel template
24 * @param path_template  pathname del template da caricare
25 * @param response
26 * @param servlet_context
27 * @throws IOException
28 */
29 public static void process(String path_template, Map<String, Object> data, HttpServletResponse response, ServletContext servletContext) throws IOException {
30     response.setContentType("text/html;charset=ISO-8859-1");
31     // Configurazione freemarker
32     Configuration cfg = new Configuration();
33     cfg.setDefaultEncoding("UTF-8");
34     cfg.setServletContextForTemplateLoading(servletContext, "/OOP_Template");
35 }
36 }
37 
```

Tale classe sarà quindi il nostro motore che girerà dietro la View e permetterà il passaggio dei dati dalla logica computazionale all'html (front-end) e viceversa, modellandoli e passandoli alle componenti opportune.

La Logica Computazione rappresenta le funzioni del sistema (Controller). In particolare abbiamo deciso di scendere in profondità proprio su questa componente in quanto fondamentale al fine della modularità richiesta dall'Object Oriented Programming. Sono state individuate tre sotto componenti quali Gestione Testo, Gestione Immagini e la Gestione generale di Sistema.

Il Database JDBC è il “local storage” del nostro portale, in cui vi saranno salvate informazioni o dati utili a quest’ultimo. Questi verranno prelevati e immagazzinati dal modello dati (Model) che rappresenterà nel nostro sistema i dati persistenti.

Il Modello Dati raggrupperà le classi dei nostri oggetti in modo da poter essere usati dal package Controller e immagazzinati ed estratti dal DBMS passando per le classi DAO (Data Access Object). Nella vista ad alto livello del Component Diagram il package relativo al Model e il package relativo al DAO sono entrambi accorpati nella componente Modello Dati. Nel diagramma a pagina 30 invece si potrà osservare meglio quanto descritto.

L'editor testo è la nostra componente (implementata internamente, come verrà descritto nelle pagine successive) che rappresenta l'input tramite la quale l'utente immetterà il testo che comporrà le trascrizioni. In parallelo lo scanner immagini è la componente (esterna) tramite la quale vi sarà l'acquisizione dell'immagine che andrà a partecipare alla digitalizzazione.

A tal proposito sono stati identificati i seguenti editor testuali per il formato TEI:

- Ace: editor testuale Web-Based
- EditTEI: editor specifico per il formato TEI
- Wed: editor testuale Web-Based

## Descrizione delle scelte

Le scelte effettuate dal gruppo sono rivolte particolarmente alla componente Logica Computazione, che rifletterà il nostro **Controller**. Nello specifico si è deciso di scendere in profondità su questa componente in modo da sfruttare la modularità dell'OOP, ovvero la possibilità di aggiornare e modificare un modulo senza toccare e rischiare di compromettere un altro. Nel nostro caso abbiamo diviso il modulo principale in tre sotto moduli quali Gestione Testo, Immagini e gestione di sistema. Tuttavia in Logica Computazione non troveremo solo queste funzioni poiché nel nostro controller ci saranno anche tutte le funzioni che tuttavia non sono modulari e quindi aggiornabili separatamente, nello specifico : Login/ Registrazione, Visualizzazione elenco titoli, Visualizzazione Completa dell'opera e Pubblicazione. Abbiamo deciso di effettuare tale scelta poiché riteniamo che queste funzioni non necessitino di essere modulari e incrementabili costantemente nel tempo.

Un'altra scelta, soggetta di varie discussioni all'interno del gruppo è stata quella relativa alla scelta dell'**Editor TEI**. Vista la carenza di editor TEI nel web, e nello specifico editor completi che riconoscessero tutta la sintassi TEI (oltre 1000 tag) il gruppo ha virato su un Editor OpenSource sviluppato da "Matteo Abrate" basandosi su CodeMirror; un editor testuale basato su JavaScript.

La particolarità dell'editor scelto è soprattutto la possibilità di controllare in modo istantaneo le modifiche in sintassi TEI e quindi di rilevare eventuali errori (previa conoscenza della sintassi TEI).

<pre> 1 This is a sample text written in MTSS (Manuscript 2 Transcription Simple Syntax), a simple language that 3 can be automatically translated into TEI.  Sentences can 4 be 5 terminated with double pipes.  Line breaks are simply 6 defined by inserting 7 newline characters.  A word that's splitted by a line 8 break 9 can be marked by using two curly braces, as in this 10 {{exam 11 ple}}.  Abbreviated words can be annotated with the 12 corresponding expansion by using a combination of square 13 and round brackets: [abbr.]({expansion}).    12 Play with this code to see how the TEI is updated. </pre>	<pre> &lt;s class="sentence" n="s_01"&gt;   &lt;lb n="01"/&gt;This is a sample text written in MTSS   (Manuscript     &lt;lb n="02"/&gt;Transcription Simple Syntax), a simple     language that       &lt;lb n="03"/&gt;can be automatically translated into TEI.   &lt;/s&gt; &lt;s class="sentence" n="s_02"&gt;Sentences can be   &lt;lb n="04"/&gt;terminated with double pipes. &lt;/s&gt; &lt;s class="sentence" n="s_03"&gt;Line breaks are simply   &lt;lb n="05"/&gt;defined by inserting     &lt;lb n="06"/&gt;newline characters. &lt;/s&gt; &lt;s class="sentence" n="s_04"&gt;A word that's splitted by a   line break     &lt;lb n="07"/&gt;can be marked by using two curly braces,   as in this &lt;w&gt;exam     &lt;lb n="08"/&gt;ple&lt;/w&gt;. &lt;/s&gt; &lt;s class="sentence" n="s_05"&gt;Abbreviated words can be   annotated with the     &lt;lb n="09"/&gt;corresponding expansion by using a     combination of square       &lt;lb n="10"/&gt;and round brackets: &lt;choice&gt;&lt;abbr&gt;abbr.     &lt;/abbr&gt;&lt;expans&gt;expansion&lt;/expans&gt;&lt;/choice&gt;. &lt;/s&gt; &lt;s class="sentence" n="s_06"&gt;   &lt;lb n="11"/&gt;   &lt;lb n="12"/&gt;Play with this code to see how the TEI is updated. &lt;/s&gt; </pre>
---	---

Tale visualizzazione completa tuttavia sarà resa disponibile all'interno del nostro portale solamente a colui che si occuperà del controllo del testo. L'utente trascrittore, dovendo solo riportare in modo testuale il contenuto dell'immagine avrà a disposizione solo il lato sinistro della figura soprastante, mentre tramite un piccolo bottone potrà visualizzare ciò che ha scritto in formato TEI completo di opportuni tag. Tali funzioni saranno mostrate nel dettaglio nella sezione di questo documento relativa alla descrizione accurata del portale.

## Design Pattern

Il design pattern adottato principalmente è il

### Model View Controller (MVC).

Il pattern è basato sulla separazione dei compiti fra le componenti del sistema che interpretano tre ruoli principali:

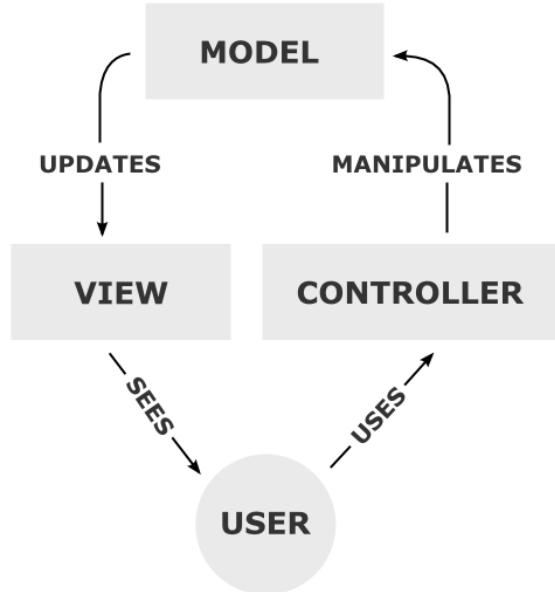
- **Model**: fornisce i metodi per accedere ai dati utili all'applicazione
- **View**: visualizza i dati contenuti nel Model e si occupa dell'interazione con gli utenti.
- **Controller**: riceve i comandi dell'utente e li attua modificando gli stati delle altre due componenti.

Perché è stato scelto il seguente pattern?

Si è scelto di utilizzare tale pattern in modo da tenere i tre strati (Model, View e Controller) quanto più separati e renderli il più indipendenti possibile. In tal modo ( noi o chi metterà mano al sistema ) si avrà vita più facile e agevole nella manutenzione del codice.

Se per esempio si decidesse di apportare delle modifiche al portale Biblioteca in modo da alterare i dati e visualizzarli in modo differente da quello già implementato, sarà sufficiente modificare solo il "Presentation Layer" senza minimamente intaccare le altre due componenti del MVC, ovvero il controller e il model. Allo stesso tempo sarà possibile incrementare il sistema aggiungendo funzioni e oggetti, senza dover re-implementare il tutto. Quest'ultima funzione risulta infatti una delle basi della programmazione ad oggetti.

In definitiva possiamo affermare di aver scelto l'**MVC** per facilitare la scalabilità e la manutenzione dell'applicazione.

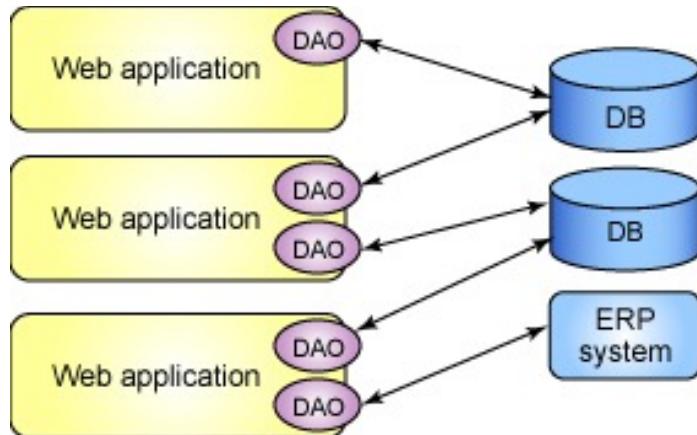


Un altro pattern risultato utile al team in fase di scelte architetturali è stato il sopracitato DAO.

## DAO - Data Access Object

Tale pattern è stato scelto ed adottato per isolare l'accesso ad una tabella tramite query.

Separare e quindi togliere il compito alla parte Model di interagire con il DBMS. Le nostre classi DAO gestiranno quindi l'accesso ai dati, incapsulando e comunicando con il DB. Si faranno quindi carico di gestire il codice SQL (con aggiunte, modifiche o eliminazioni), facendo trasparire il tutto ai package di Model e di Controller.



## Librerie utilizzate



**Apache Tomcat:** application server utilizzato per far girare il nostro portale in locale. Implementa le specifiche JSP e Servlet, quindi rientra pienamente nella nostra architettura.



**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.

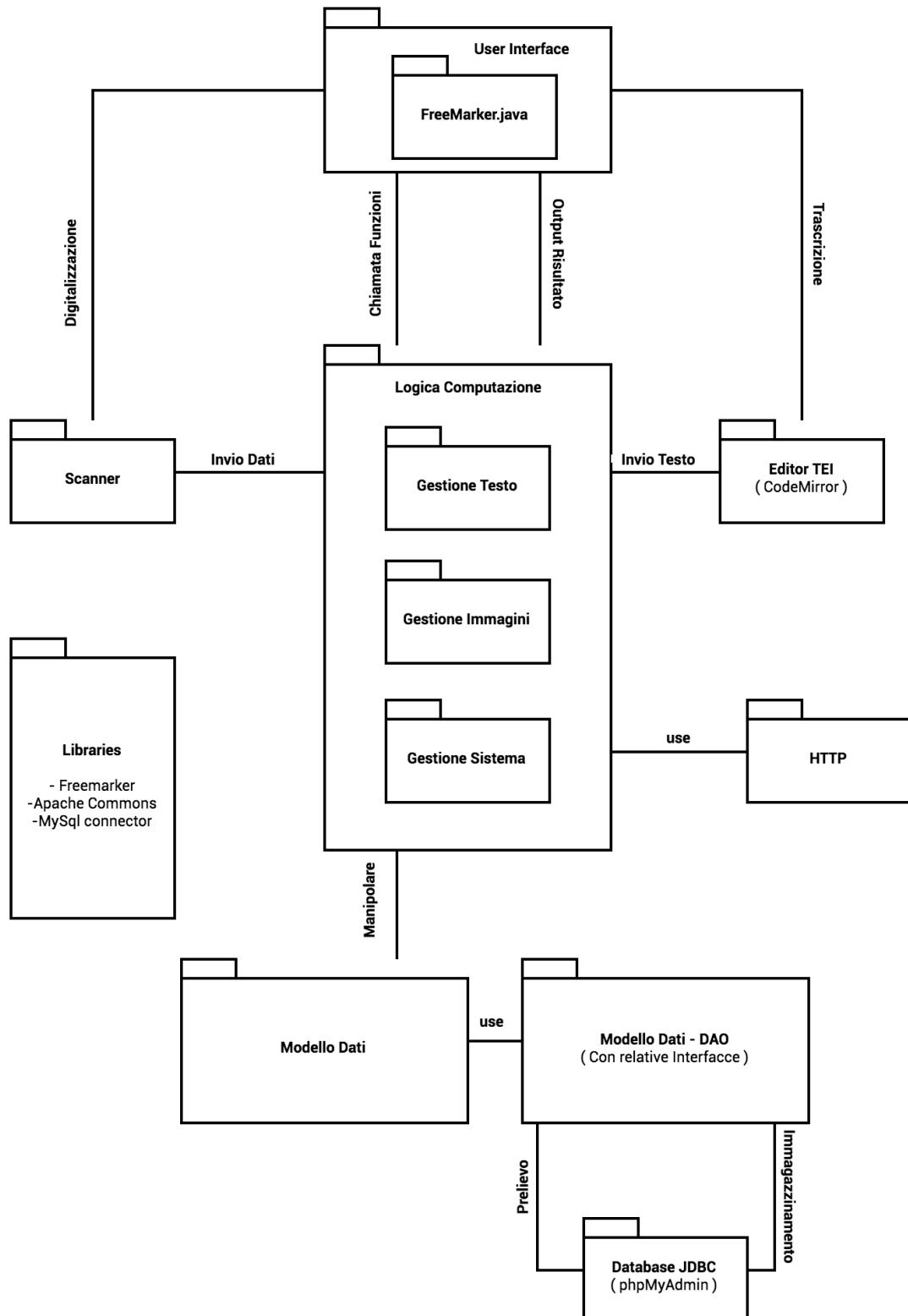


**FreeMarker:** motore di template utilizzato per fare da mediatore tra la logica computazione in Java e l'HTML.



**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 MAMP.

## Modello completo dell'Architettura del Sistema



## Sequence Diagram

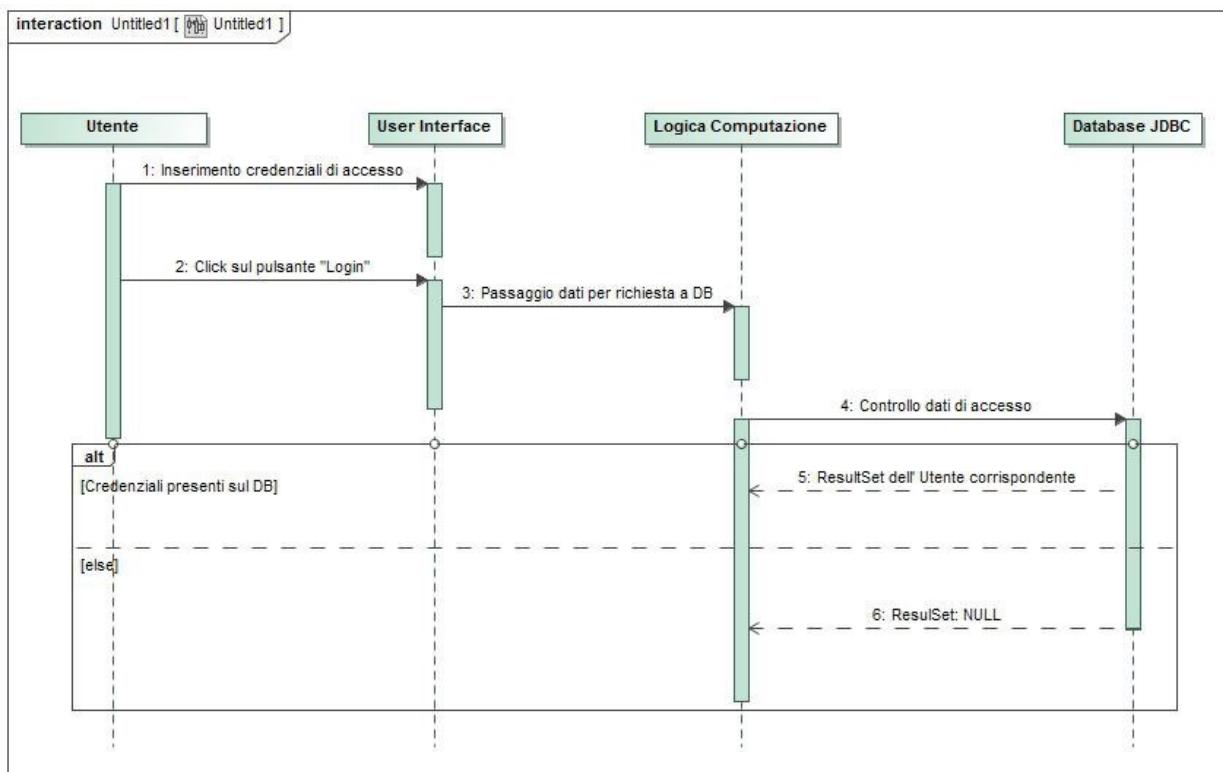
I seguenti Sequence Diagram permettono, una volta sviluppata la suddivisione in componenti, di capire l'interazione tra esse.

Va fatta l'assunzione che per le funzioni avanzate bisogni passare necessariamente per quelle basilari, quali Login o Registrazione, seguendo la logica espressa nel flusso di eventi rappresentato precedentemente tramite Use Case.

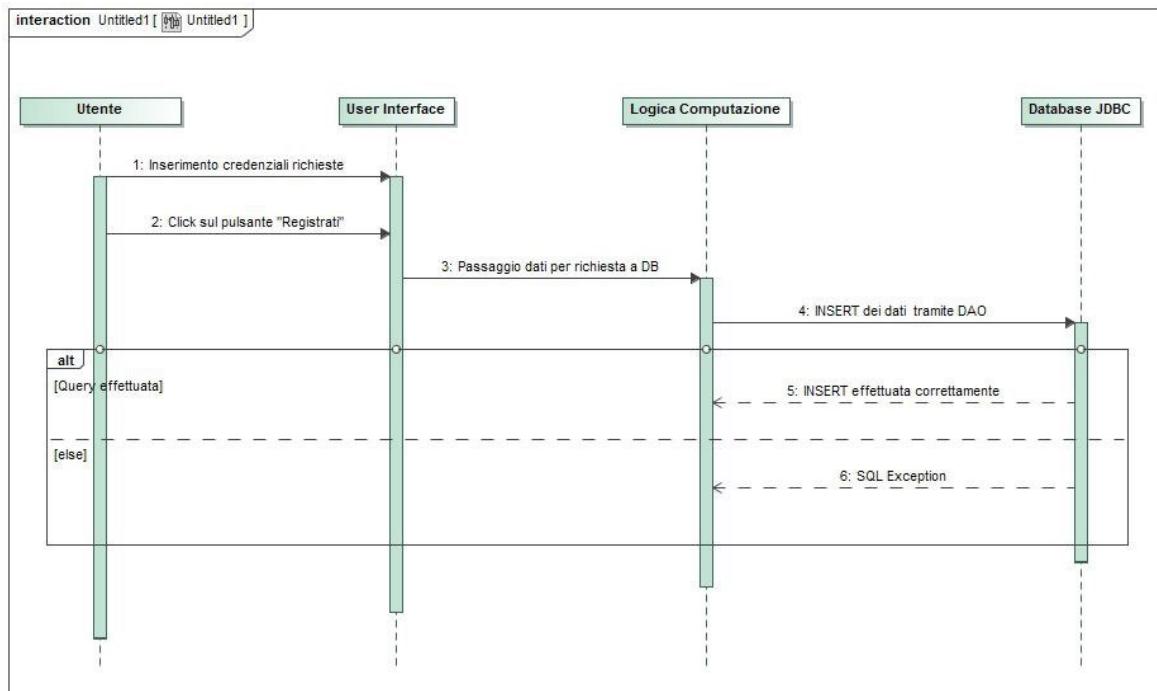
Verranno utilizzati i "Combined Fragment" per evidenziare i casi di successo o errore.

**NOTA:** l'eventuale risposta da parte della "User Interface" verso l'utente va intesa come messaggio a video di un messaggio passato dalla Logica Computazione.

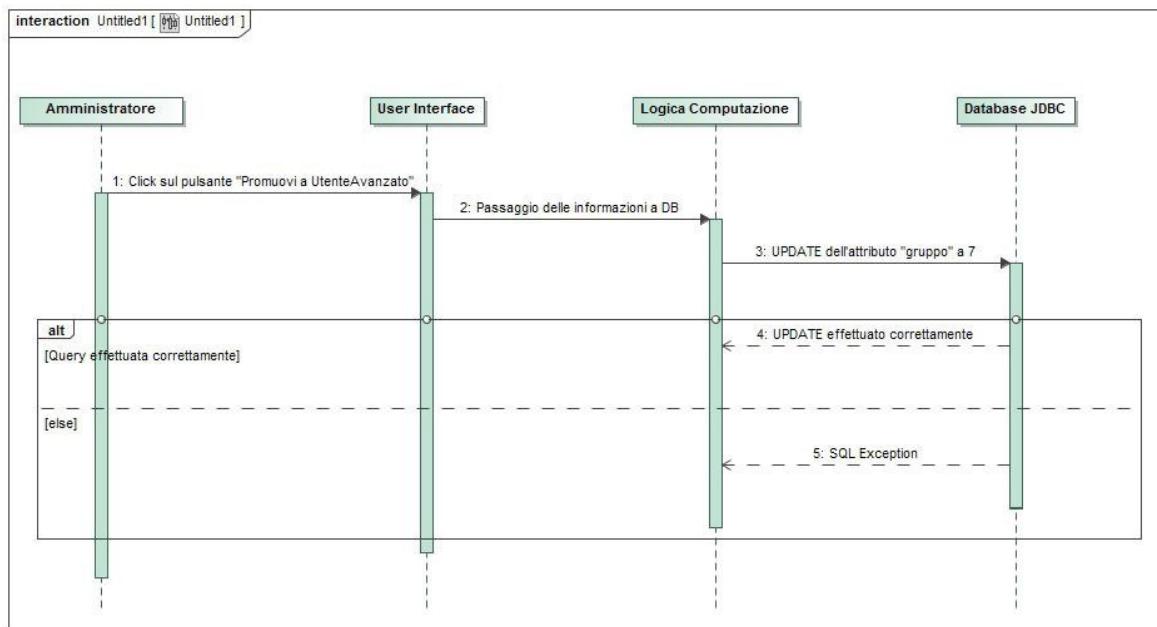
### Login



## Registrazione

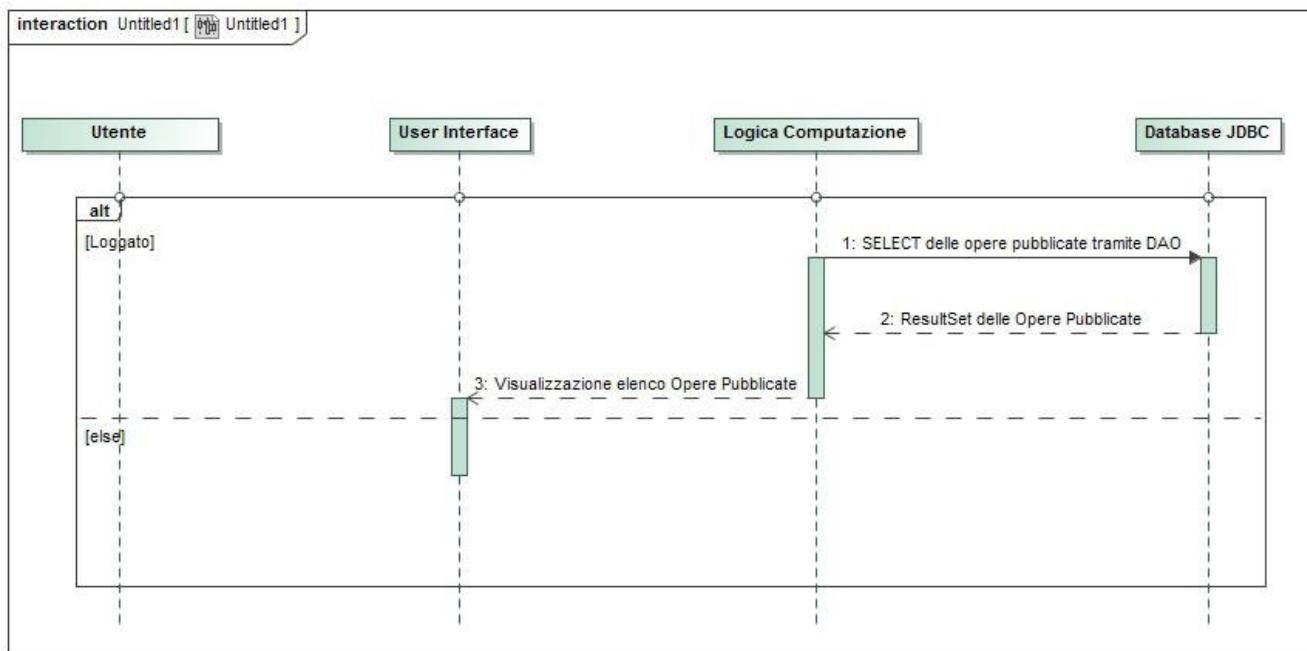


## Promozione Utente

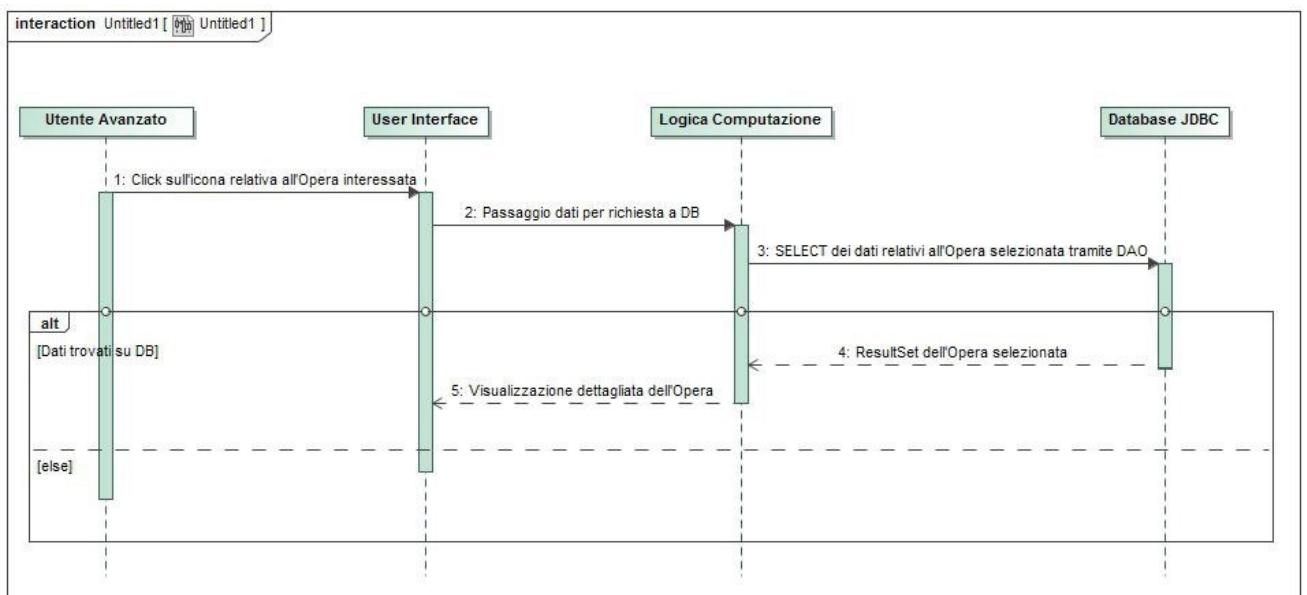


In questo Sequence Diagram è riportato il flusso per una qualsiasi promozione.

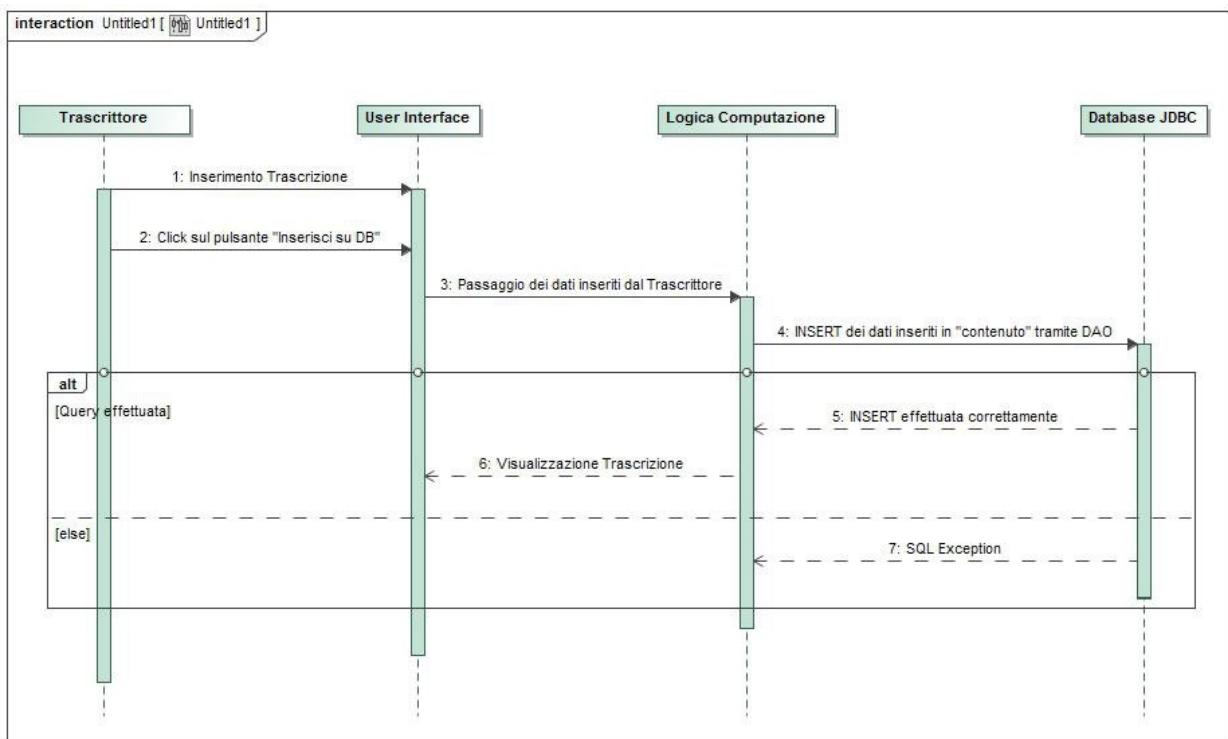
### Visualizzazione Elenco Titoli



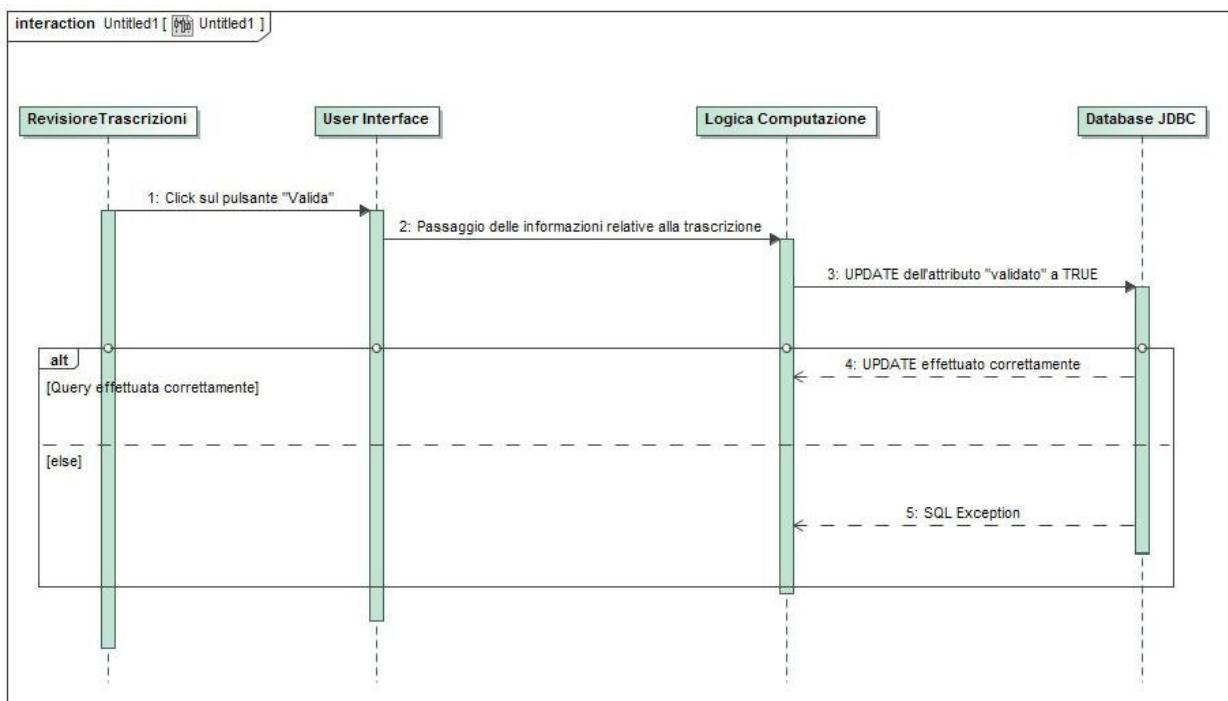
### Visualizzazione Opera Completa



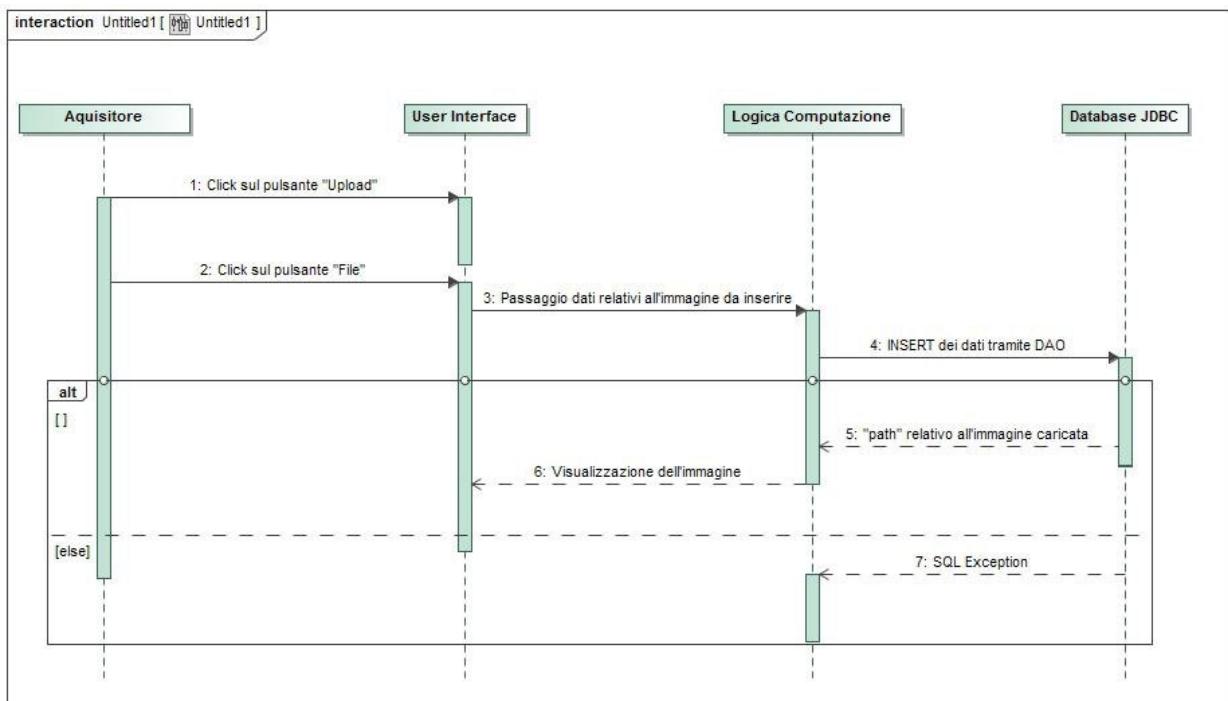
## Trascrizione



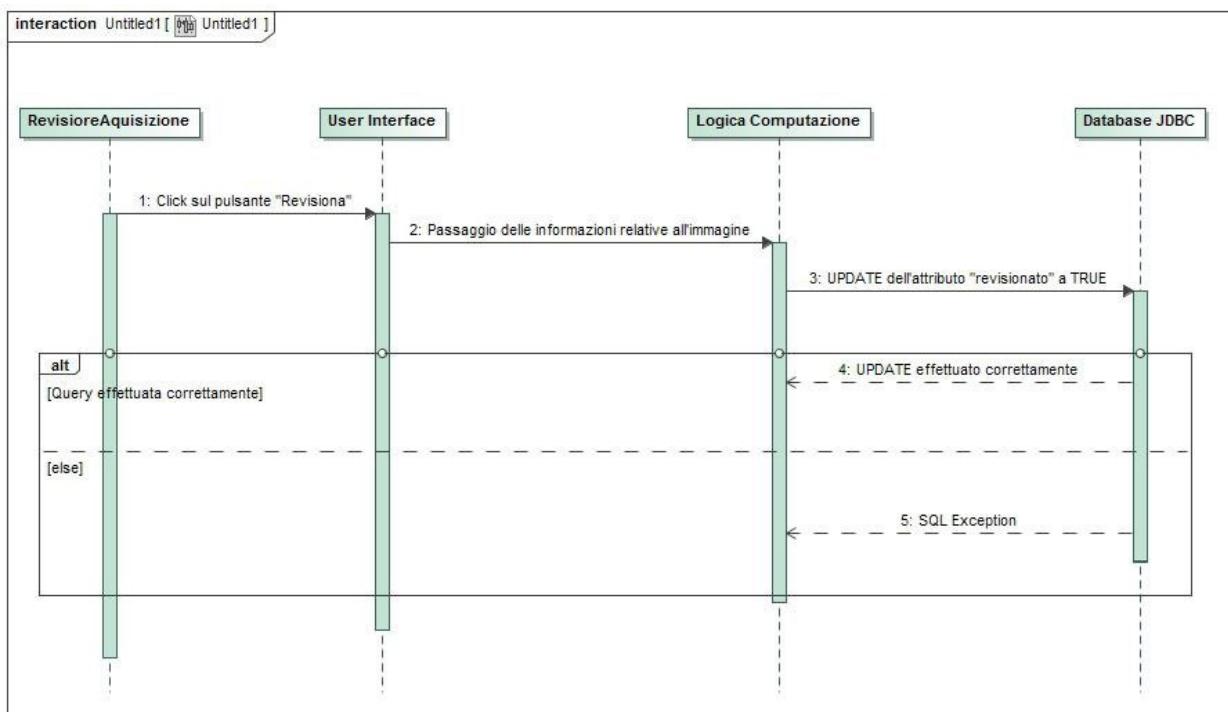
## Validazione



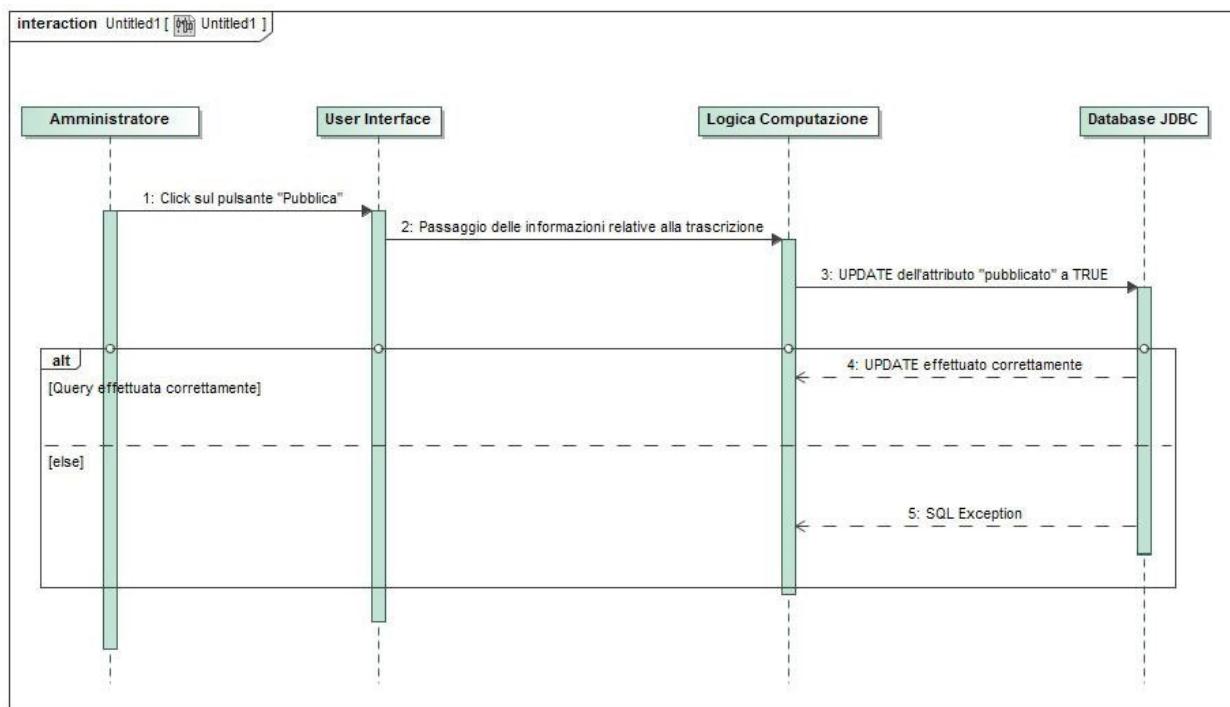
## Digitalizzazione



## Revisione



## Pubblicazione

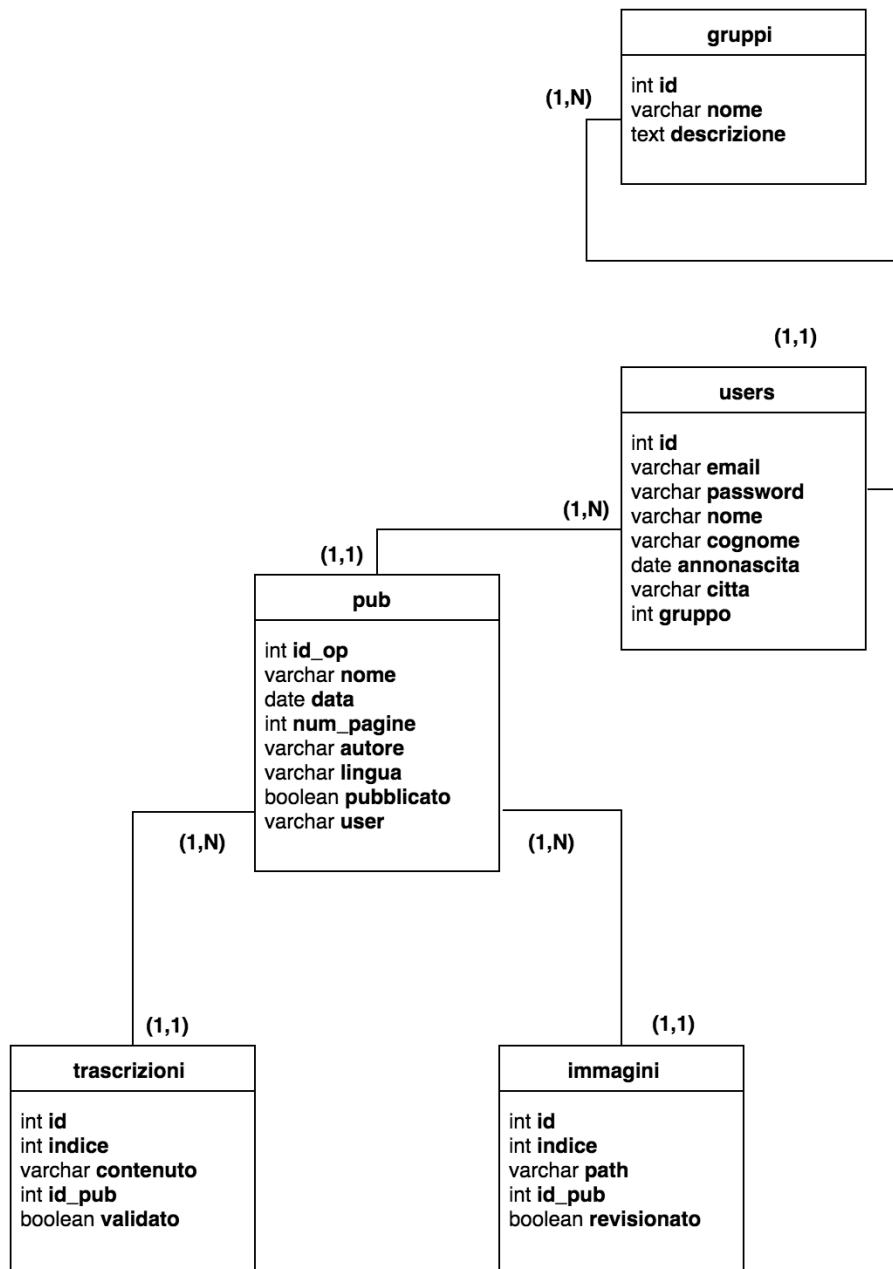


## Modello ER

Il Database è stato oggetto di varie discussioni e scelte nel team. Quella principale ha portato quest'ultimo ad effettuare la separazione logica tra Utenti - Gruppi in modo da agevolare ed organizzare bene il tipo di utenza e i servizi a disposizione. Questa scelta è stata preferita ad esempio all'eventuale creazione di più sottoclassi di utenti che avrebbero portato più codice e probabilmente meno efficienza.

Nello specifico i gruppi individuati (tra i requirements) sono stati suddivisi in base agli ID:

- **1 :** Amministratore
- **2:** Utente Base
- **3:** Trascrittore
- **4:** Revisore Trascrizioni
- **5:** Acquisitore
- **6:** Revisore Acquisizioni
- **7:** Utente Avanzato



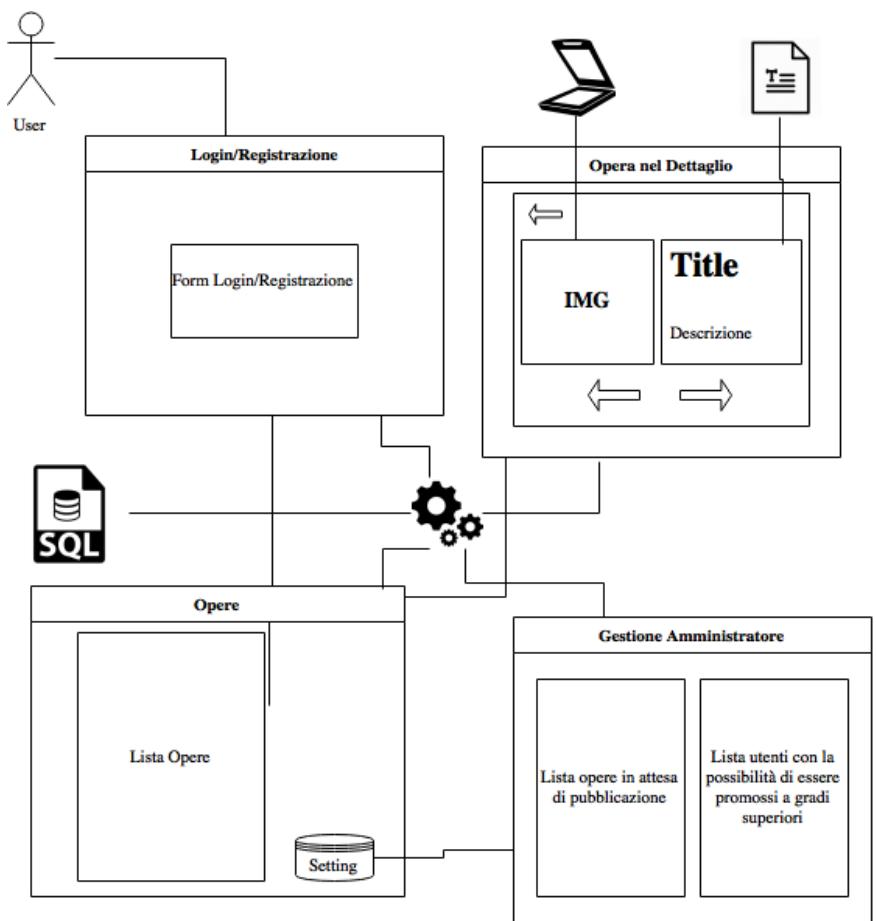
## Osservazione

Prima di procedere al Software Design il team ha voluto mostrare l'idea sviluppata circa la realizzazione del sistema, e come lo si è immaginato.

Si è immaginato quindi un portale (Web in questo caso) tramite la quale l'utente alla prima apertura non può vedere altro che la form di Login e Registrazione e solo dal momento che questa vada a buon fine egli potrà accedere alla lista delle opere pubblicate, ma non alle opere nel dettaglio, poiché tale funzionalità è riservata ad un'altra gerarchia di utente.

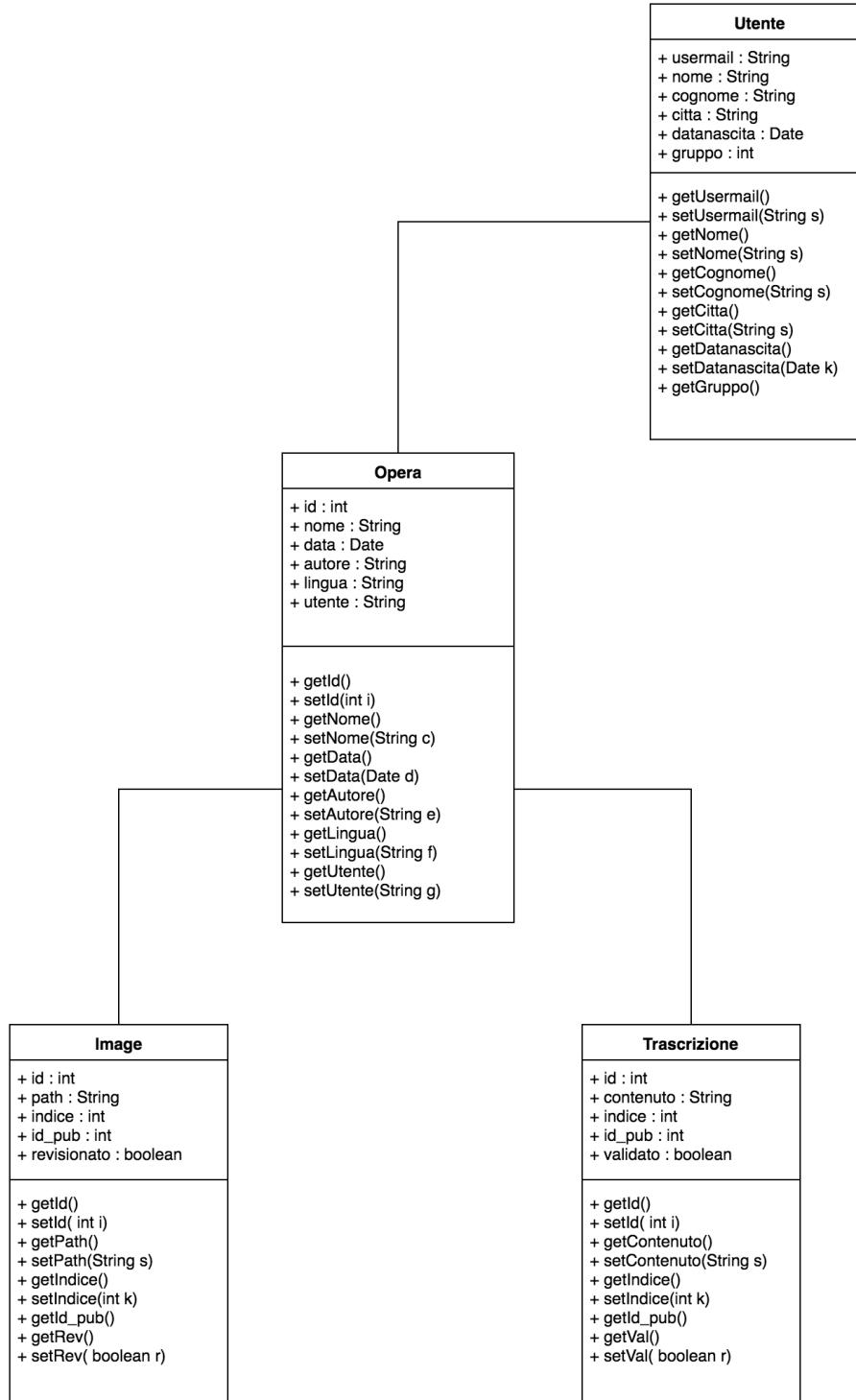
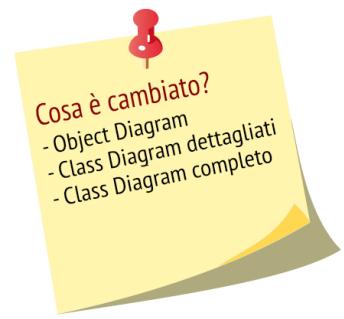
Nel caso in cui l'utente sia un revisore o validatore egli potrà accedere all'opera nel dettaglio, controllando e validando/revisionando le eventuali immagini/trascrizioni.

Nel caso in cui invece l'utente sia l'amministratore del sistema egli potrà osservare in alto a destra un pulsante "setting" che gli permetterà di accedere alla pagina di gestione del portale nella quale vi sarà la lista completa delle opere e la lista degli utenti, permettendogli quindi di promuoverli o declassarli ad un altro grado.



# Software Design

## Modello - Object Diagram



### Descrizione Modello - Object Diagram - *Package: bibliotecadigitale.Model*

Gli oggetti identificati sono i seguenti e costituiranno la parte **Model** in base al riferimento al pattern adottato, e ovviamente saranno poi implementati come classi nella fase successiva:

**Opera:** rappresenta l'opera nel dettaglio, identificata dall'ID e caratterizzata dai propri attributi quali: il nome, la data di pubblicazione, l'autore, la lingua e l'utente che l'ha inserita. I metodi disponibili per tale oggetto saranno i get e i set dei relativi attributi.

**Image:** rappresenta la prima componente di un'opera, identificata da una variabile ID, da un path del file immagine ottenuto tramite la scannerizzazione e da un booleano che ci identifica lo stato di revisione; se settato a true l'immagine è passata dalla revisione e quindi accettata, false altrimenti. Un'altra caratterizzazione di tale oggetto è l'id della pubblicazione di riferimento e l'indice che rappresenta la pagina di quest'ultima. L'indice inoltre sarà utilizzato dalla Logica per effettuare le corrispondenze tra immagini e trascrizioni. I metodi dell'oggetto fanno riferimento a i get/set della variabile immagine.

**Trascrizione:** indica la seconda componente di un'opera, identificata tramite un ID ed è composta da una variabile stringa contenente il testo ottenuto dall'edito esterno, e da una variabile booleana che ci indica lo stato di validazione; true nel caso sia stato validato, false nel caso in cui debba ancora passare la validazione oppure non l'abbia passata. Un'altra caratterizzazione di tale oggetto è l'id della pubblicazione di riferimento e l'indice che rappresenta la pagina di quest'ultima. L'indice inoltre sarà utilizzato dalla Logica per effettuare le corrispondenze tra immagini e trascrizioni. I metodi dell'oggetto fanno riferimento al get/set del testo della trascrizione.

**Utente:** rappresenta l'utente vero e proprio ed è caratterizzato da dati anagrafici quali nome e cognome, data di nascita, città di residenza e dati di accesso quali usermail e password e l'identificatore ID.

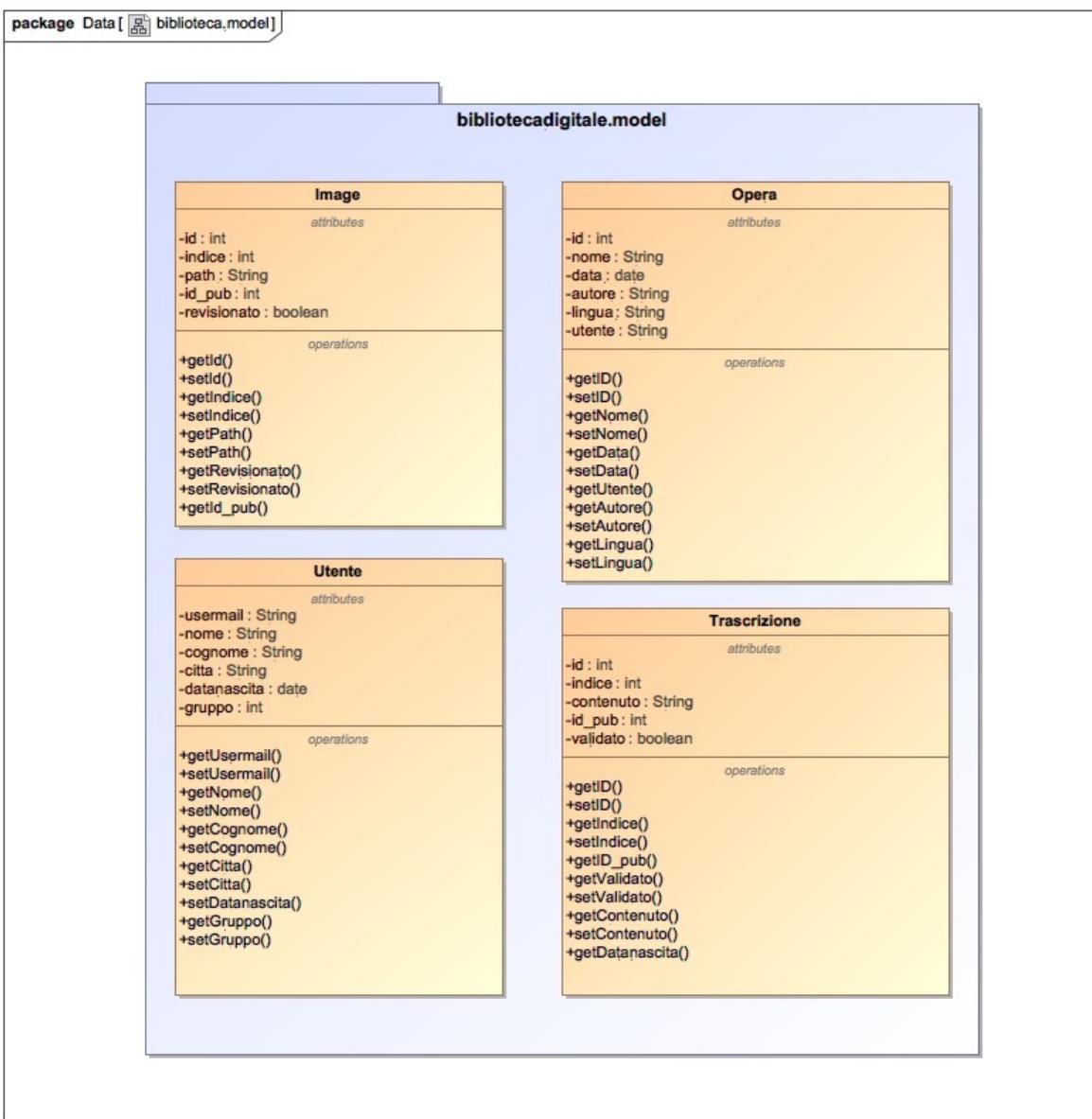
Inoltre come spiegato nel modello ER vi sarà il riferimento all'intero del gruppo di appartenenza. Tramite questo intero la logica Java e il motore di templating FreeMarker saranno in grado di verificare le funzionalità disponibili agli utenti e quindi i permessi sul portale.

I metodi disponibili in tale oggetto sono i get e set generici di tutte le variabili eccetto password ed ID.

## Modello - Class Diagram

Il nostro Object Diagram è convertibile a questo punto in un **Class Diagram** riguardante i nostri oggetti nello specifico, ovvero di come questi sono stati quindi sviluppati nel nostro ambiente.

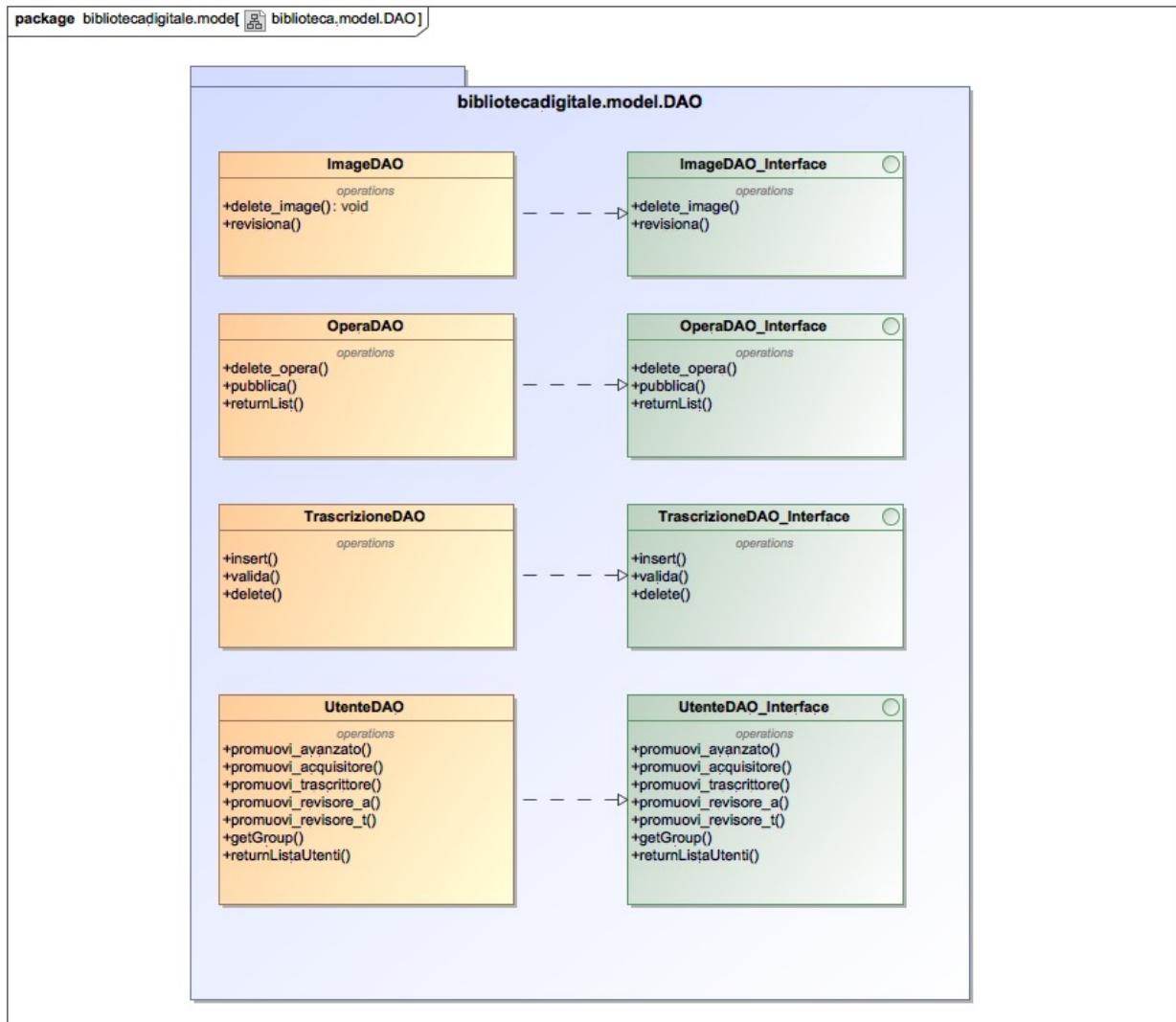
### Class Diagram - Model



A questi poi corrisponderanno, secondo il nostro design pattern adottato il package relativo alle classi DAO, che si occuperanno dell'interazione con il Database.

Tale class diagram lo osserveremo nella prossima pagina.

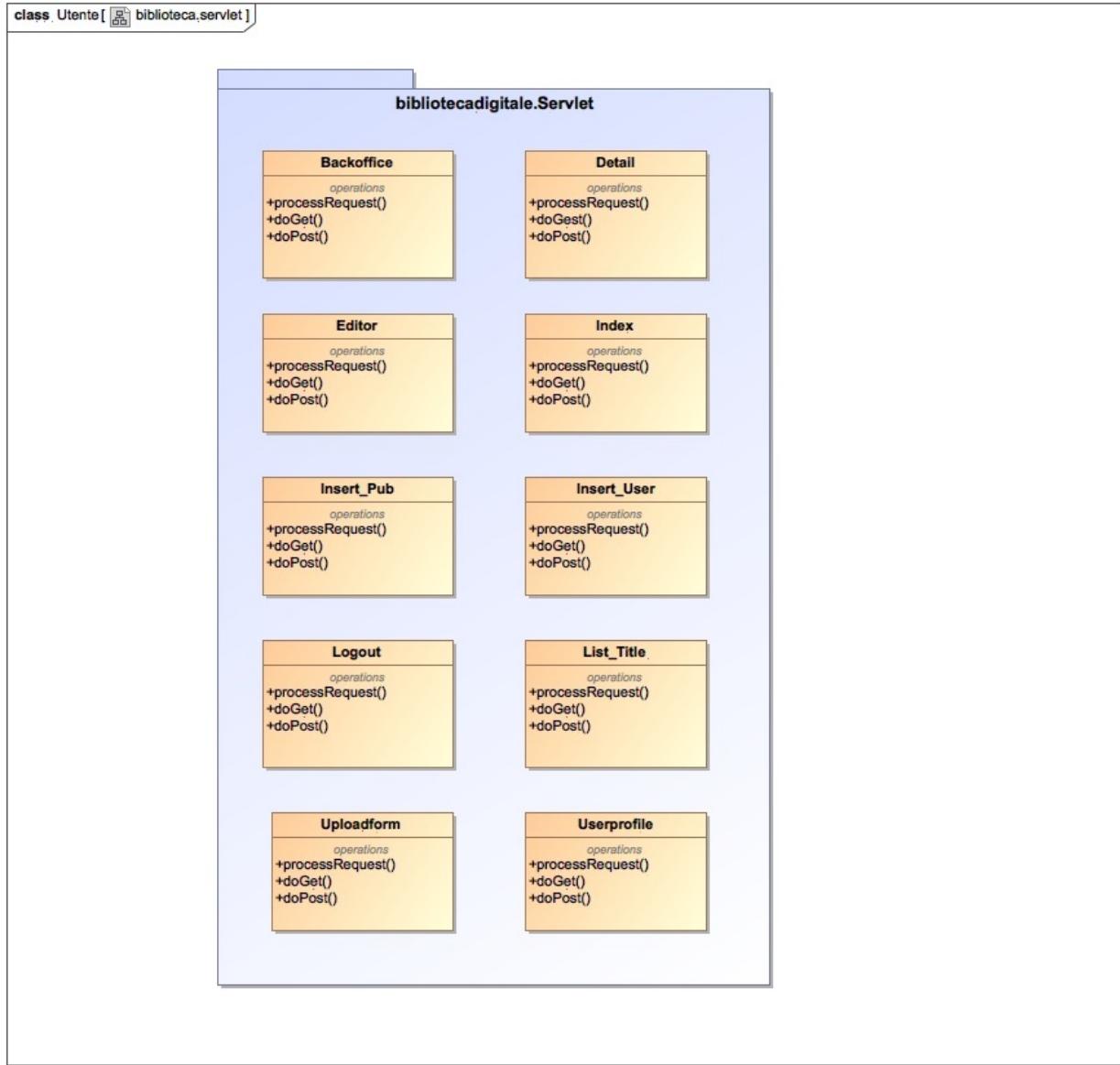
## Class Diagram - Model DAO



Tale diagramma raffigura il nostro package contenente le classi DAO, che si occuperanno come abbiamo detto in precedenza dell'interazione con il database e le interfacce che queste estenderanno.

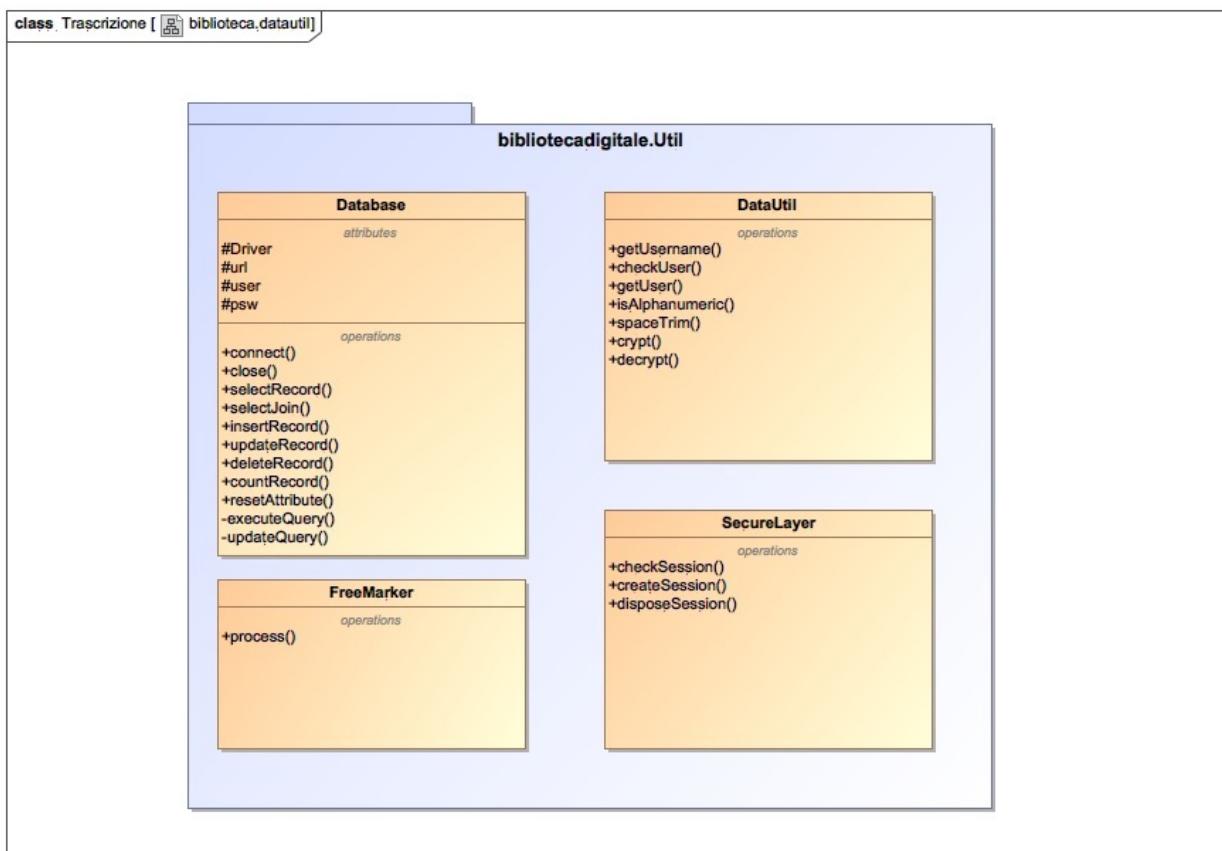
Le **Interfacce** conterranno come da definizione solo la dichiarazione dei metodi vuota che andrà poi riscritta dalla classe che le implementerà. Saranno quindi una sorta di “contratto” tra noi e il linguaggio, in modo da permettere a futuri sviluppatori di poter accedere in modo semplice e rapido alle funzioni e la loro semantica tramite le stesse senza dover andare a leggere ogni classe specifica.

## Class Diagram - Servlet (Controller)



Questo diagramma raffigura nel nostro sistema il package relativo alle classi che faranno da Servlet del nostro portale. I metodi raffigurati saranno sempre gli stessi come da definizione di "Servlet". Tuttavia al loro interno ci sarà tutta la nostra logica di business che farà girare il sistema, nello specifico vi saranno chiamate alle classi DAO per l'interazione con il database e ci saranno gli algoritmi che permetteranno la corretta esecuzione del portale.

## Class Diagram - Util (Controller)

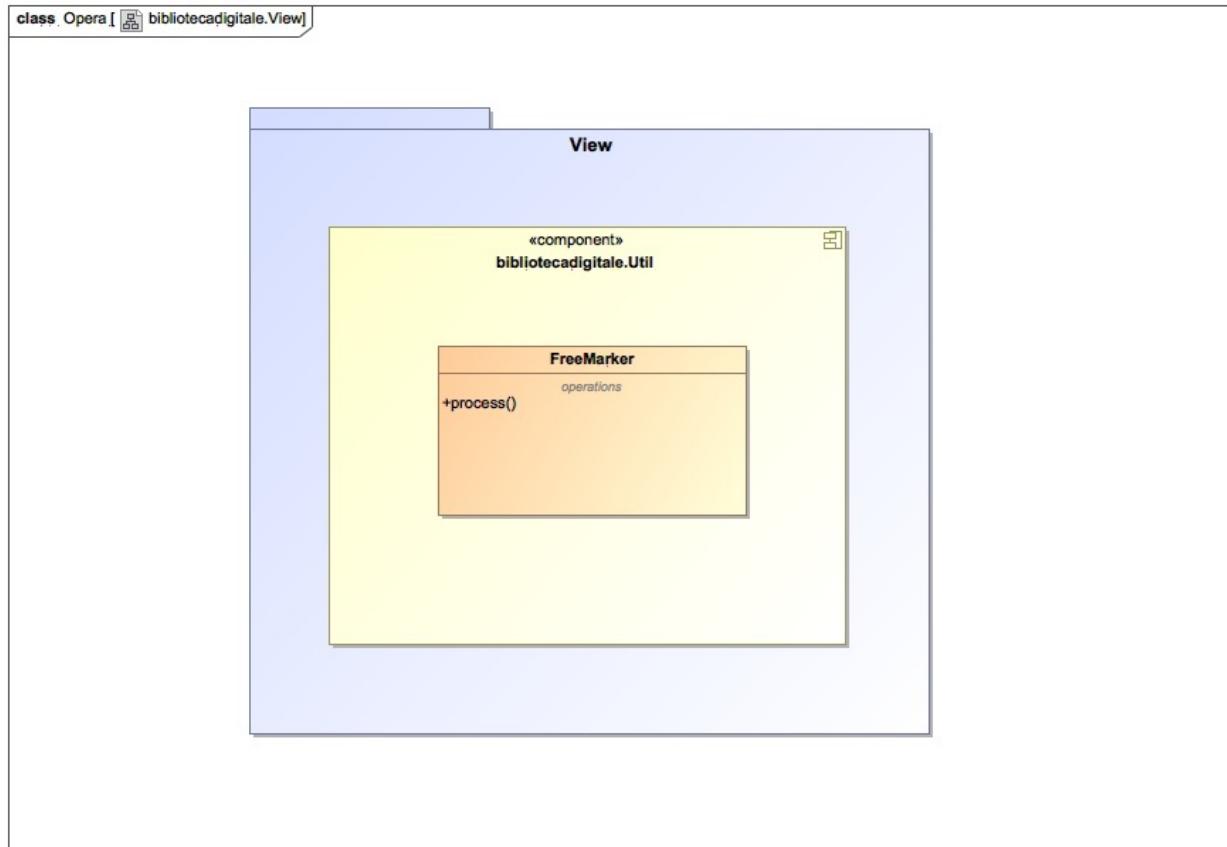


Il team per convenienza ha voluto creare questo package in modo da velocizzare e semplificare la futura modifica dei moduli sviluppati (Model, View, Controller).

Nello specifico tale package conterrà quattro classi:

- **Database.java:** classe contenente metodi di accesso semplificato al database, in modo da non dover riscrivere le stesse righe di codice ogni qual volta bisognava interrogare lo stesso.
- **FreeMarker.java:** classe che corrisponderà alla nostra View, poichè mediatrice della logica di business con il motore di template adottato. Basterà quindi richiamare il metodo “process” di questa classe senza dover richiamare tutti i metodi di FreeMarker relativi al montaggio di un template.
- **SecureLayer:** classe utilizzata per una corretta gestione delle sessioni. Basterà quindi un richiamo ad uno di questi metodi (check/create/dispose- session) per controllare l’eventuale esistenza, crearne una o terminare la sessione d’utenza.
- **DataUtil:** è una sorta di “magazzino” del nostro portale, in cui andiamo a sviluppare metodi che possono esserci utili in quasi tutto il sistema, senza dover ripeterne la dichiarazione.

## Class Diagram - View

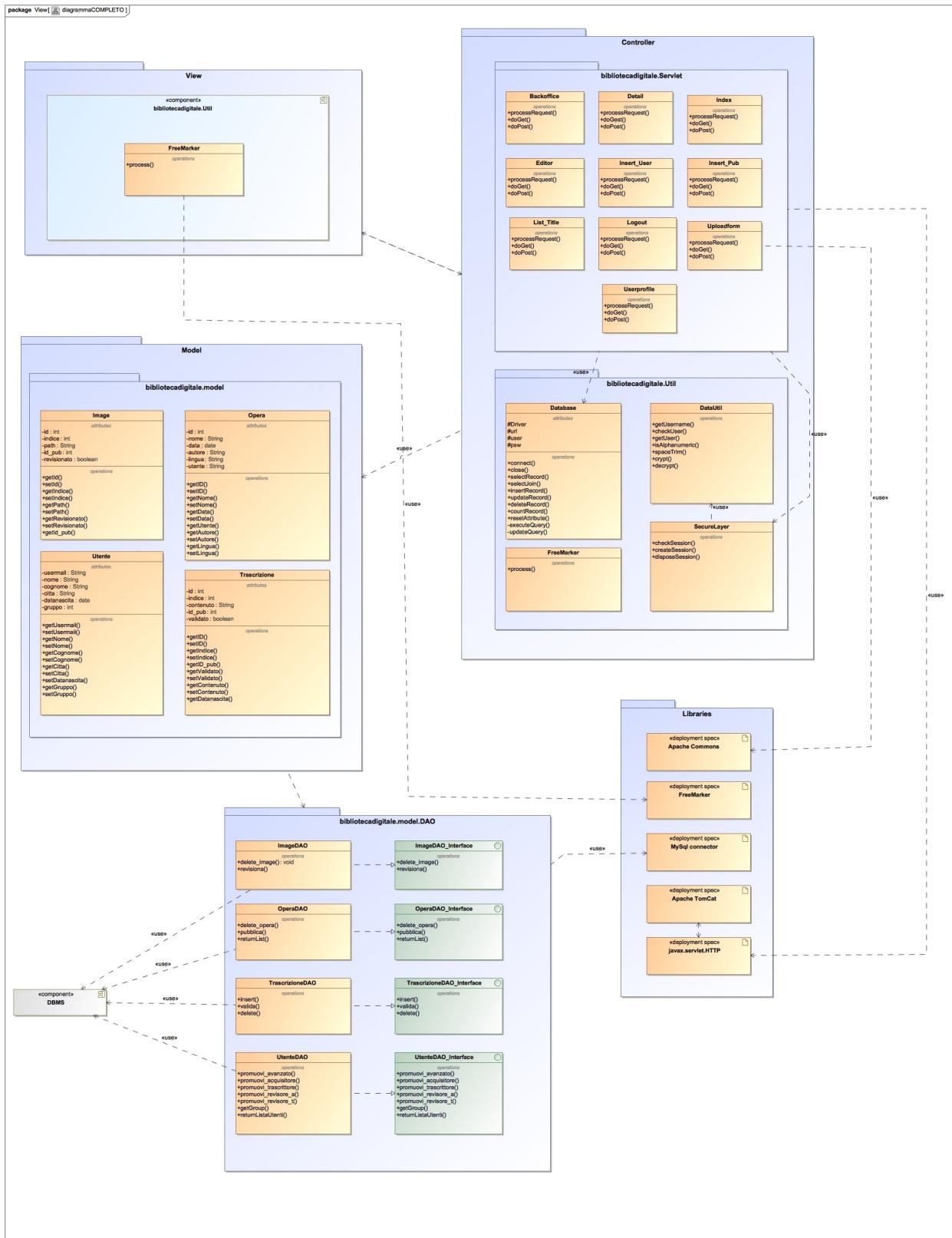


Classe che corrisponde alla nostra View del design pattern adottato (MVC). Farà quindi da mediatore tra il controller e l'utente. Prenderà i dati dalla logica, li modellerà e li manderà a video, sfruttando la libreria che fa riferimento al motore di template associato (FreeMarker).

*Esempio: nella Servlet relativa alla lista delle opere mi creo una lista tramite chiamata alla classe DAO delle opere e inserisco il risultato nella mappa che dovrò passare all'HTML per mostrarlo a video. Come avverrà tale passaggio?*

*Una volta creata la mappa da passare, chiameremo il metodo "freemarker.process("HTML", MAPPA)" che andrà a richiamare il metodo contenuto nella nostra classe View ovvero **FreeMarker.java**, che tramite il metodo "process" modellerà la mappa e la manderà a video.*

## Class Diagram Completo



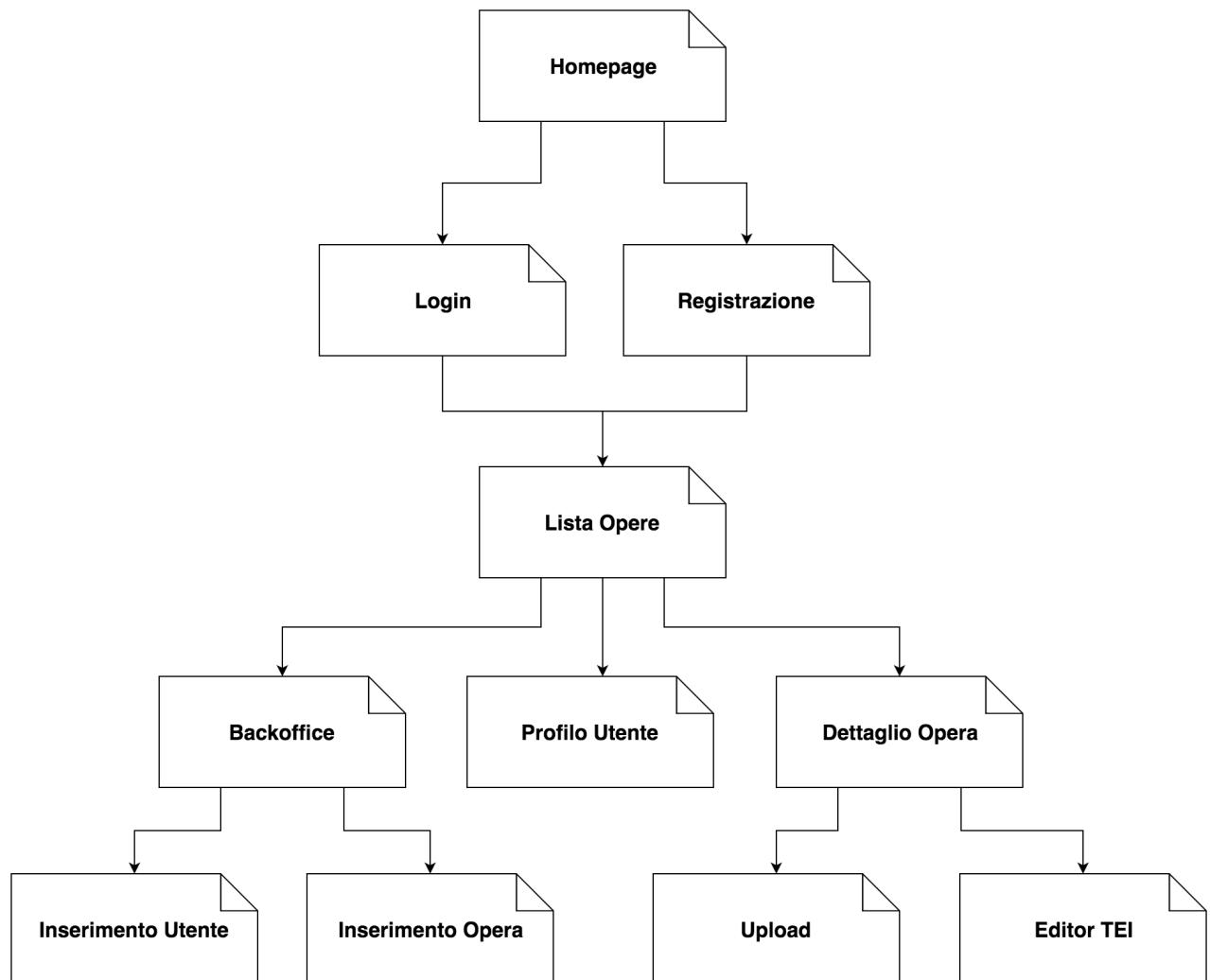
# Template Explanation



Trattandosi di un portale Web il team ha deciso di aggiungere una sezione nel documento relativa alla spiegazione del portale da un punto di vista web in modo da capirne bene il funzionamento.

Ciò sarà effettuato tramite raffigurazione del sitemap nella quale ogni “card” corrisponderà ad una pagina HTML e tramite raffigurazione del portale in screenshot con relativi commenti.

## Sitemap



## Rappresentazione Sito Web

The screenshot shows the homepage of a digital library. At the top center is the title "Biblioteca Digitale". Below it is a yellow shield logo featuring a bird of prey. A subtitle "Portale Web per progetto di Object Oriented Programming" is displayed. On the left, there is a "Login" form with fields for "Email" and "Password", and a blue "ACCEDI" button. To the right of the login form is a photograph of two people, a man and a woman, looking at a document together. Below the photo is the word "Registrazione".

The screenshot shows a page titled "Elenco Opere". At the top, a dark header bar displays the user's name "Benvenuto Patrizio" and navigation icons. The main title "Elenco Opere" is centered above a stylized icon of three books. Below the title, a subtitle reads "Lista delle Opere accessibili per progetto di **Object Oriented Programming**". A list of accessible works is shown in a table format:

Think in Java	>
Teoria calcolabilità e complessità	>
Autamata Tutor	>
UML in the heart	>

localhost

## Amministrazione

### Opere da pubblicare



- Think in Java
- Teoria calcolabilità e complessità
- Autamata Tutor
- UML in the heart

**PUBBLICA**   **ELIMINA**   **VAI ALL'OPERA**

**+**

### Gestione Utenza



Luca Ferrari - luca@ferrari.it

#### Promuovi a:

**UTENTE AVANZATO**   **AQUISITORE**   **TRASCRITTORE**

**REVISORE AQUISIZIONI**   **REVISORE TRASCRIZIONI**

Stefano Cortellessa - stefano@gmail.it  
Patrizio Pezzilli - test@test.it

**+**

localhost

## Pagina Personale



**Patrizio Pezzilli**

### Dati Personalni

<input type="checkbox"/>	test@test.it
<input type="checkbox"/>	12-dic-1993
<input type="checkbox"/>	Latina

localhost

## Inserimento Opera



**TITOLO** Titolo \_\_\_\_\_

**AUTORE** Autore \_\_\_\_\_

**NUMERO PAGINE** Numero Pagine \_\_\_\_\_

**LINGUA** Lingua \_\_\_\_\_

 **INSERISCI**

localhost

## Inserimento Utente



**EMAIL** Email \_\_\_\_\_

**PASSWORD** Password \_\_\_\_\_

**NOME** Nome \_\_\_\_\_

**COGNOME** Cognome \_\_\_\_\_

**DATA DI NASCITA** \_\_\_\_\_

**CITTÀ** Citta \_\_\_\_\_

 **AGGIUNGI**

The screenshot shows a web browser window with the URL "localhost". The title bar says "Think in Java" and "PAGINA ATTUALE : 0". The main content area has a large "UPLOAD" button. Below it is a placeholder image with a black arrow pointing up and the text "Immagine non disponibile, caricala qui!". At the bottom are buttons for "PREV", "REVISIONA", "ELIMINA", and "NEXT". To the right is a code editor window with the following text:

```
1 {{ !!!!! Testo non ancora inserito !!!!! }}
2
3 ||Vuoi essere il primo? ||
4
5 ||Modifica questo testo e clicca "Inserisci" in basso a destra
6 |Quello che inserisci sarà soggetto a validazione.||(Grazie...).||
```

Below the code editor are buttons for "View TEI Code", "OPEN IN EDITOR TEI", and "INSERISCI".

The screenshot shows a web browser window with the URL "localhost". The title bar says "Think in Java". The main content area has a "FILE" button and an "UPLOAD" button.

The screenshot shows a web browser window titled "Think in Java" with the URL "localhost". The page content is a text editor interface. On the left, there is a code editor with the following text:

```
1 {{ !!!!! Testo non ancora inserito !!!!! }}
2
3 ||Vuoi essere il primo?||
4
5 ||Modifica questo testo e clicca "Inserisci" in basso a destra
6 [Quello che inserisci sarà soggetto a validazione.] (Grazie...). ||
7
8
9
10 AGGIORNATO
```

On the right, there is a larger text area containing XML-like code:

```
<s class="sentence" n="s_01">
<lb n="01"/><w> !!!!! Testo non ancora inserito !!!!! </w>
<lb n="02"/>
<lb n="03"/>
</s>
<s class="sentence" n="s_02">Vuoi essere il primo?
</s>
<s class="sentence" n="s_03">
<lb n="04"/>
<lb n="05"/>
</s>
<s class="sentence" n="s_04">Modifica questo testo e clicca "Inserisci" in
basso a destra
<lb n="06"/><choice><abbr>Quello che inserisci sarà soggetto a
validazione.</abbr><expans>Grazie...</expans></choice>.
</s>
<s class="sentence" n="s_05">
<lb n="07"/>
<lb n="08"/>
<lb n="09"/>
<lb n="10"/>AGGIORNATO
```

At the bottom center of the page are two buttons: "VALIDA" and "ELIMINA".