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 (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.
- **Trascrizione *(4)*:** Disponibile solo al Trascrittore ed è la funzione che permette il passaggio da immagine a testo dell'opera.
- **Validazione *(4)*:** Permette al Revisore Trascrizioni di validare una trascrizione inserita.
- **Digitalizzazione *(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.
- **Pubblicazione *(5)*:** Consente all'amministratore del sistema di rendere disponibile agli Utenti Avanzati una o più opere digitalizzate ed eventualmente trascritte.
- **Gestione Generale del Sistema *(2/3)*:** Permette all'Amministratore di modificare/cancellare/inserire opere nella raccolta ed eventualmente modificare il grado di uno o più utenti

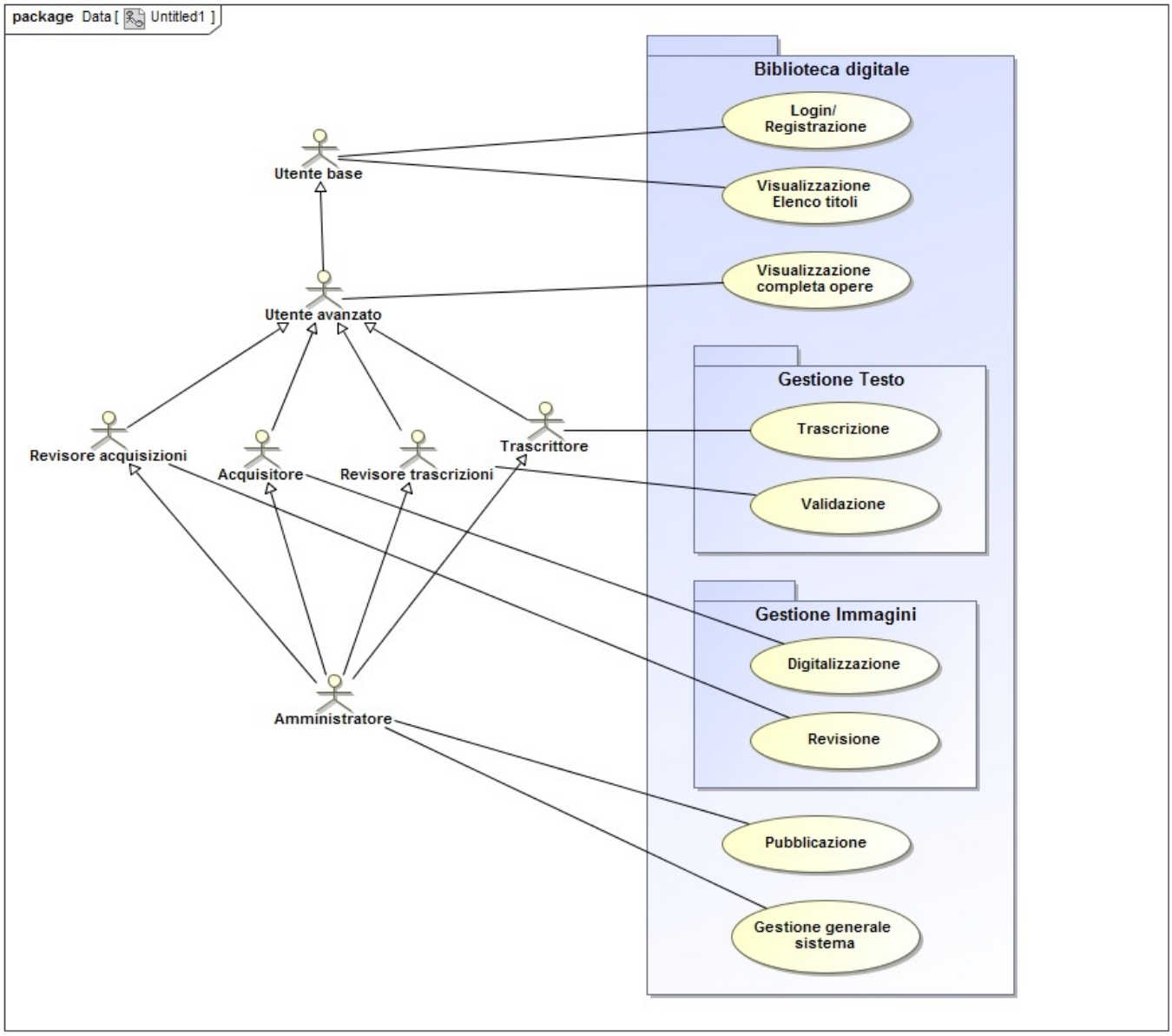
Requisiti Non Funzionali (NFR)

- **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.

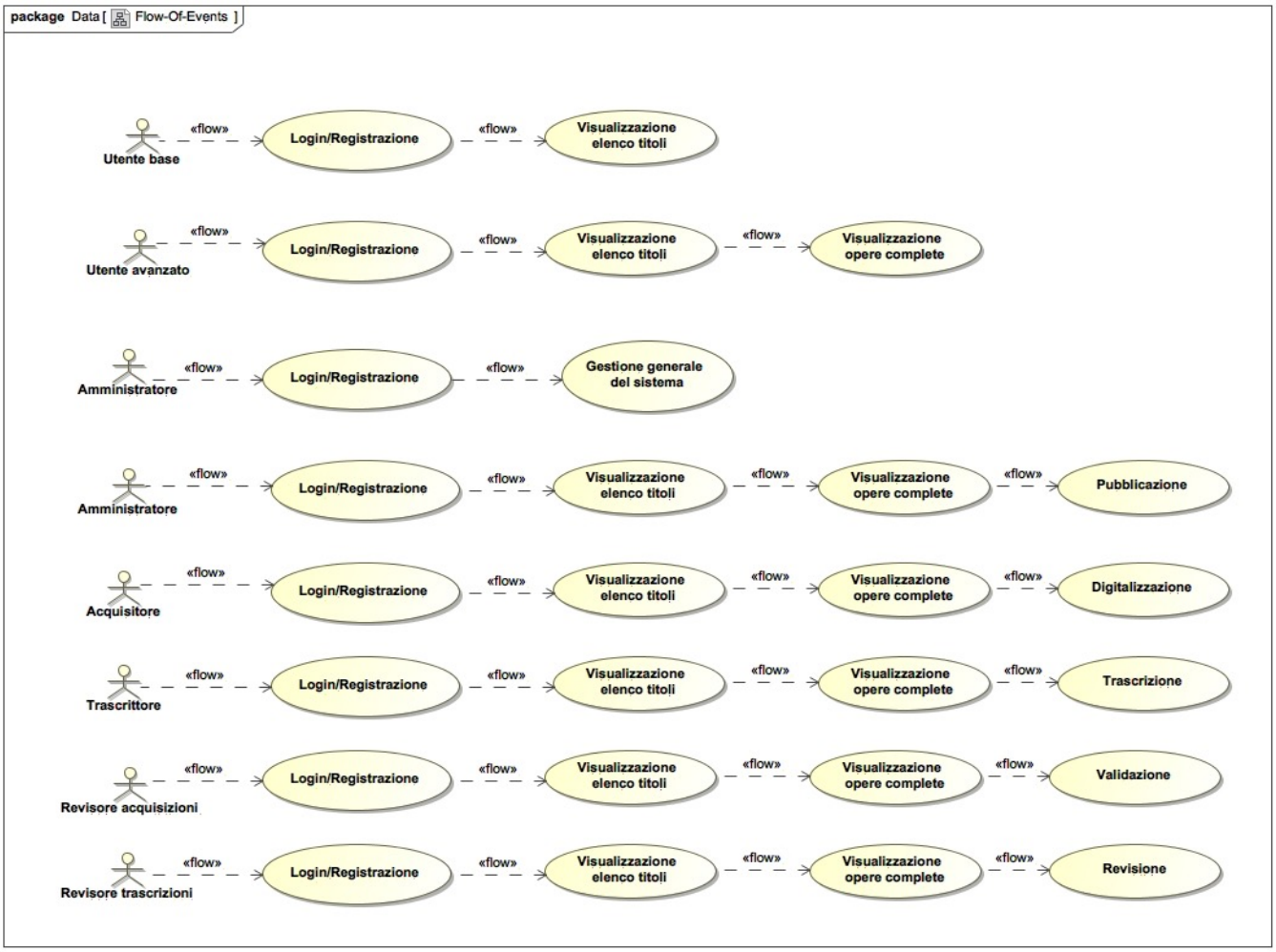
Use Case Diagram



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	**Login / Registrazione**
Partecipating actors	Utente Base
Descrizione	Permette ad un utente registrato di accedere al sistema o ad un utente no registrato di registrarsi.
Evento Scatenante	Click da parte dell'utente sul relativo pulsante
Uses	Login/Registrazione effettuati con successo o errore

Use Case name	**Trascrizione**
Partecipating actors	Trascrittore
Descrizione	Permette all'attore l'inserimento del testo relativo ad una pagine 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	**Validazione**
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

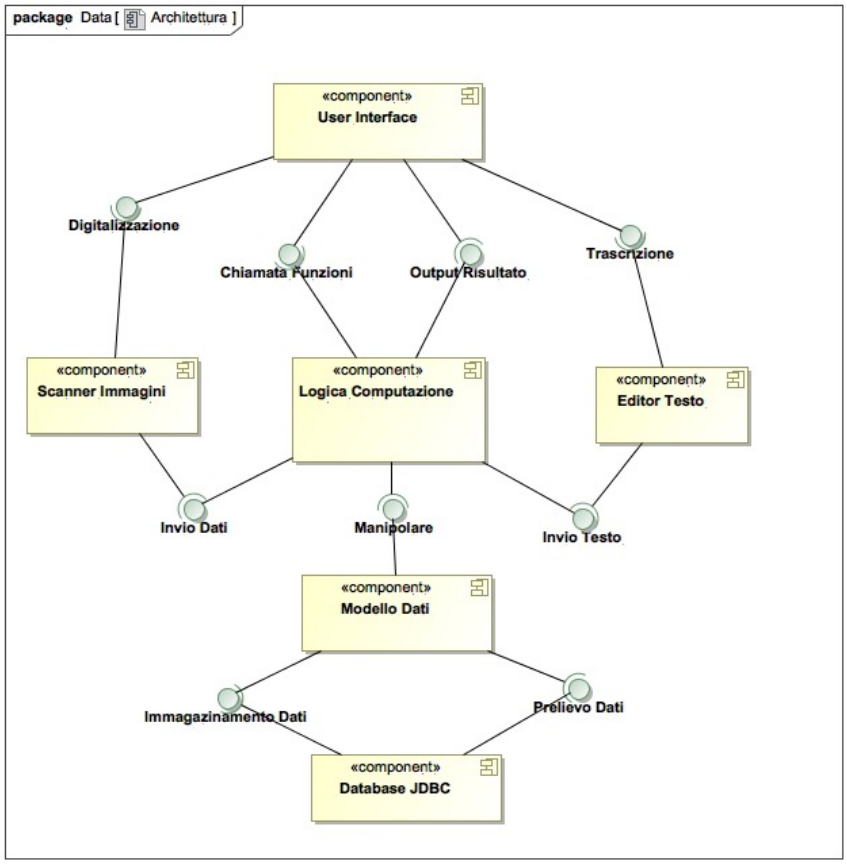
Use Case name	**Digitalizzazione**
Partecipating actors	Acquisitore
Descrizione	Funzione che permette l'acquisizione dell'immagine, una volta scannerizzata, in attesa di revisione
Evento Scatenante	Upload dell'immagine nel relativo form
Uses	Immagine caricata correttamente o meno, in attesa di revisione.

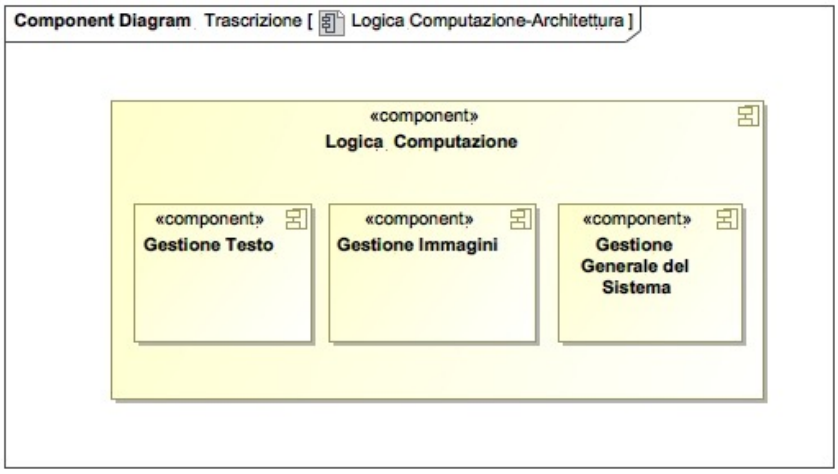
Use Case name	**Revisione**
Partecipating actors	Revisore Acquisizioni
Descrizione	Permette all'attore il controllo della qualità dell'immagine, e il successivo giudizio
Evento Scatenante	Click sul pulsante "revisiona"
Uses	Immagine revisionata correttamente, o errore in caso di problema nel sistema.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

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 ad template associato e configurato tramite motore di templating essendo l'applicativo sviluppato su piattaforma web.

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 del 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). In questa vista ad alto livello del nostro Component Diagram il package relativo al Model e il package relativo al DAO sono entrambi accorpati nella componente Modello Dati. Nel diagramma a pag.13 invece si potrà osservare meglio quanto descritto.

L'editor testo è la nostra componente (implementabile internamente o richiamabile esternamente) che rappresenta l'input tramite la quale l'utente immetterà il testo che comporrà le trascrizioni. In parallelo lo scanner immagini è la componente (quasi sicuramente 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 comprometterne un altro. Nel nostro caso abbiamo diviso il modulo principale in tre sotto moduli quali Gestione Testo, Immagini e gestione del 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.

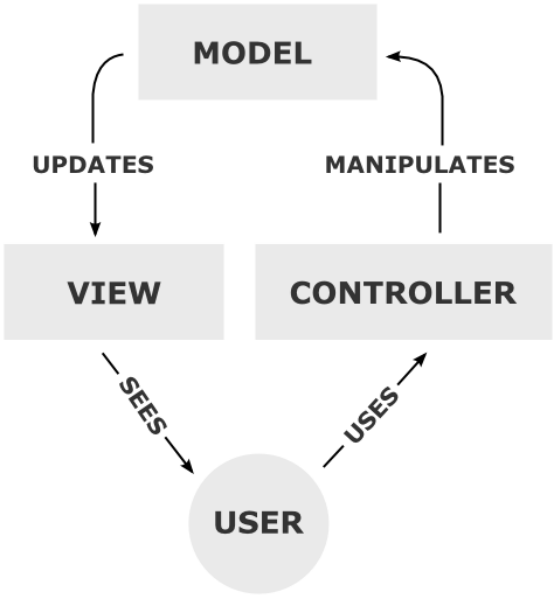
Design Pattern

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.

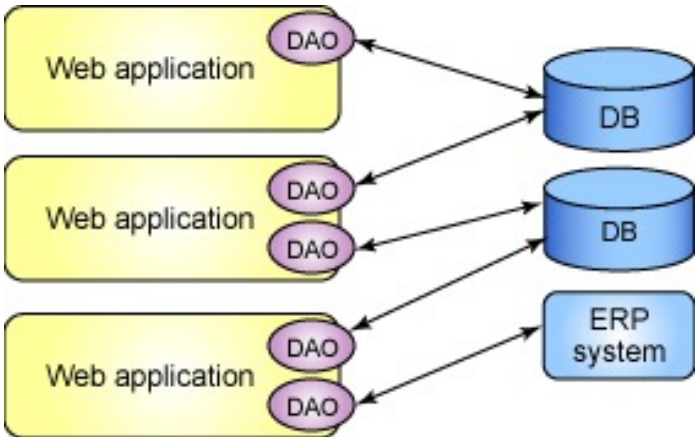
In definitiva possiamo affermare di aver scelto l'**MVC** per facilitare la scalabilità e la manutenzione dell'applicazione.

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

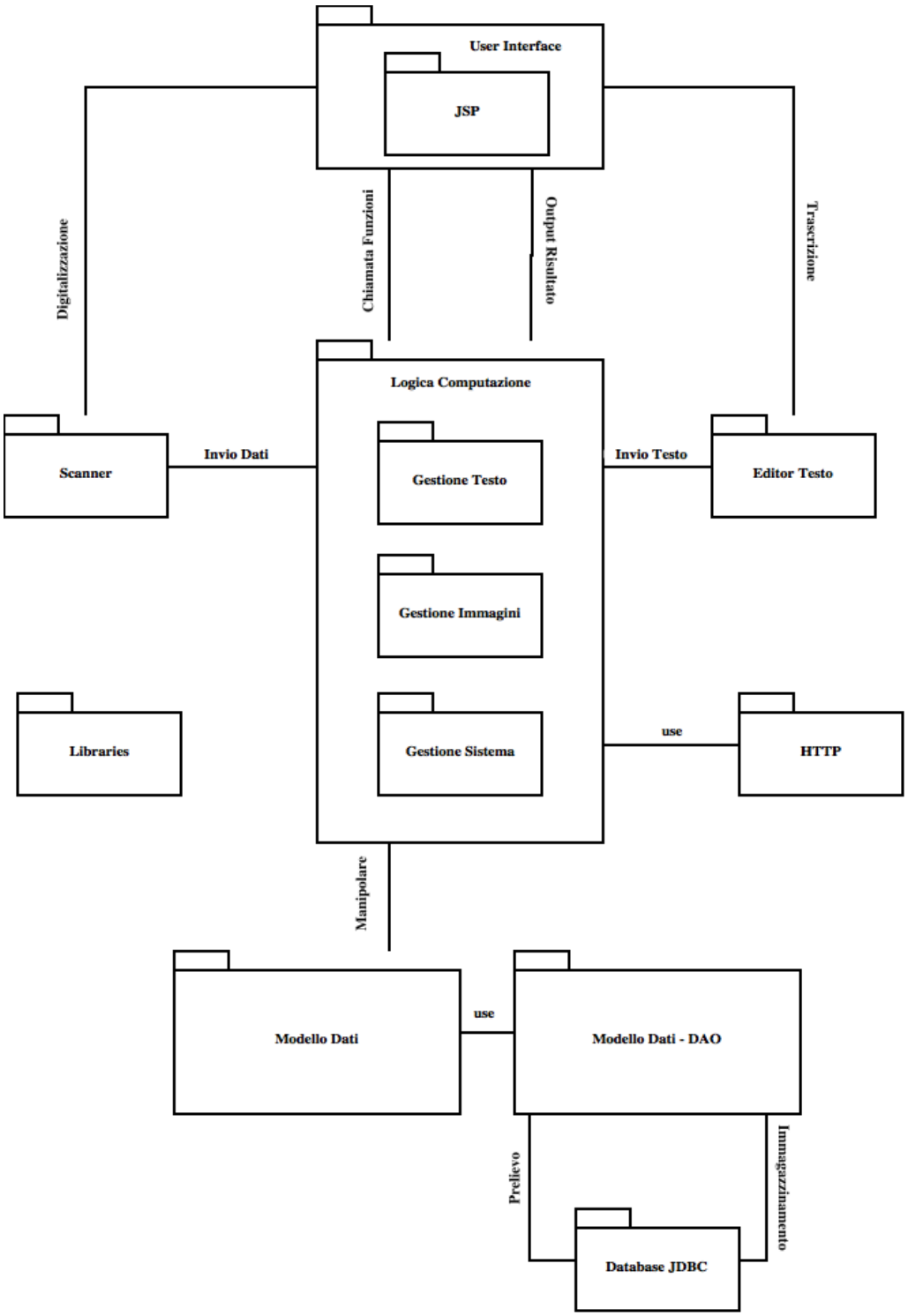
DAO - Data Access Object

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.



Modello completo dell'Architettura del Sistema



Sequence Diagram

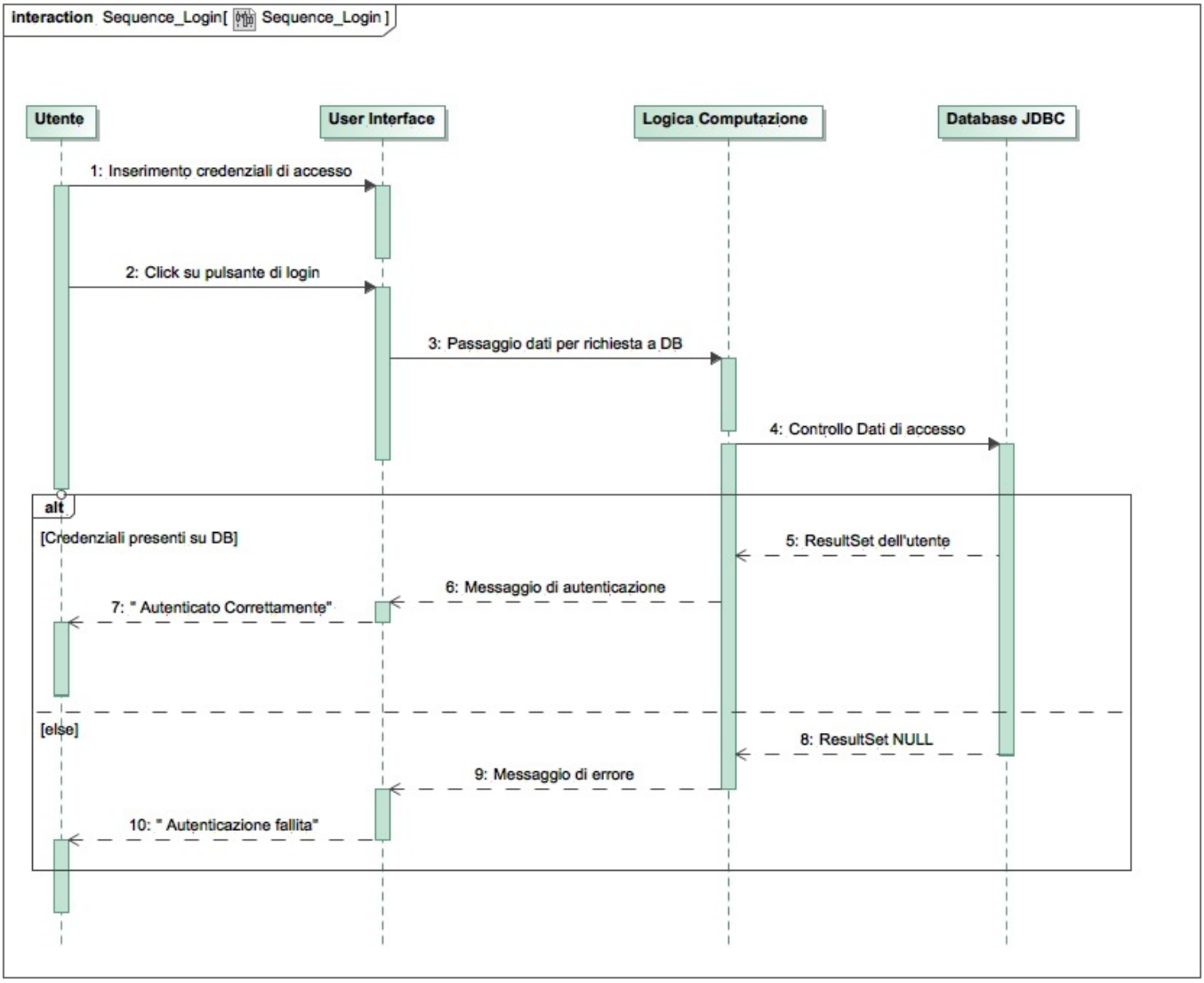
I seguenti Sequence Diagram permettono, una volta sviluppata la suddivisione in componenti, di capire l'interazione tra esse.

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.

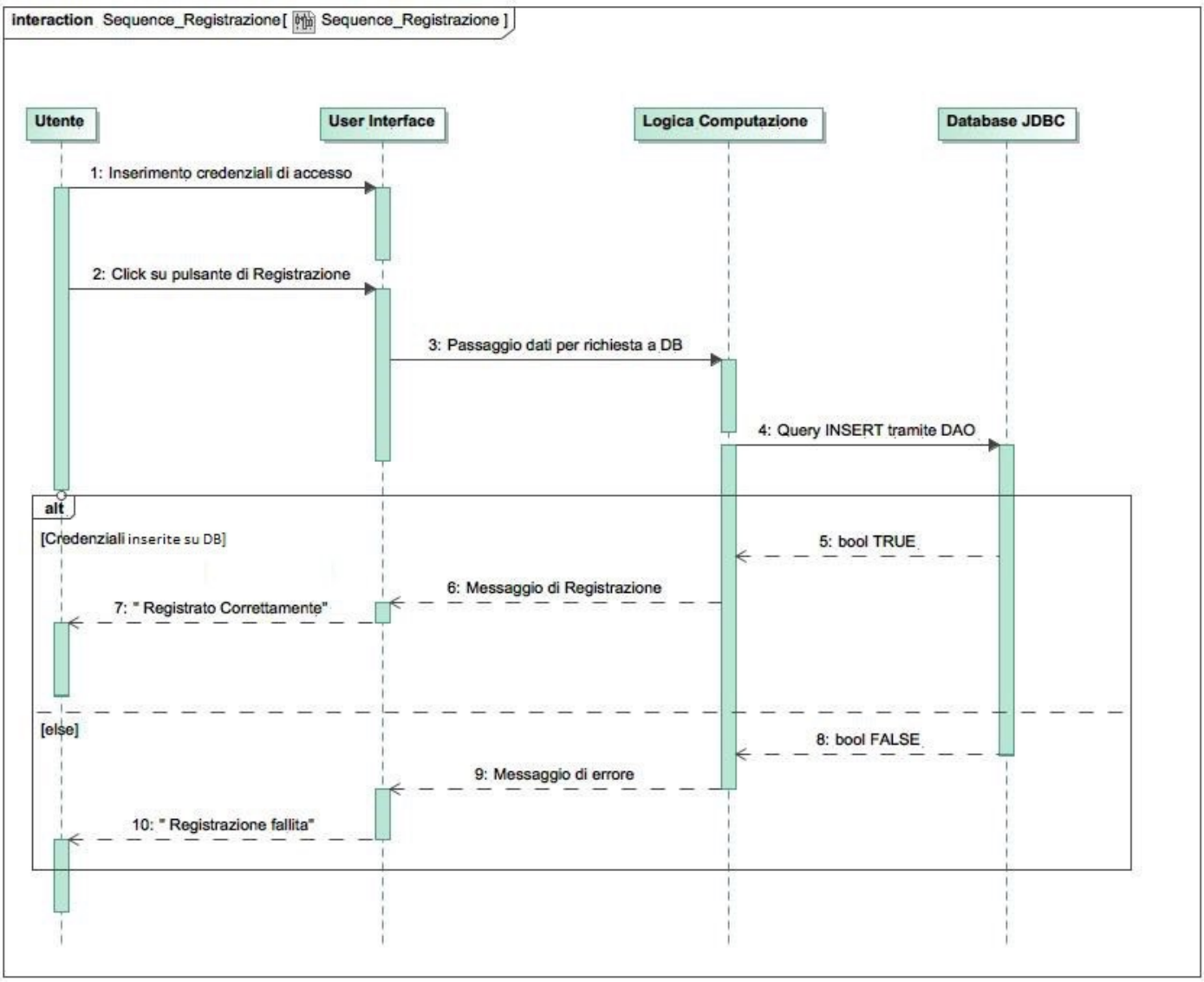
Verranno utilizzati i "Combined Fragment" per evidenziare i casi di successo o errore.

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

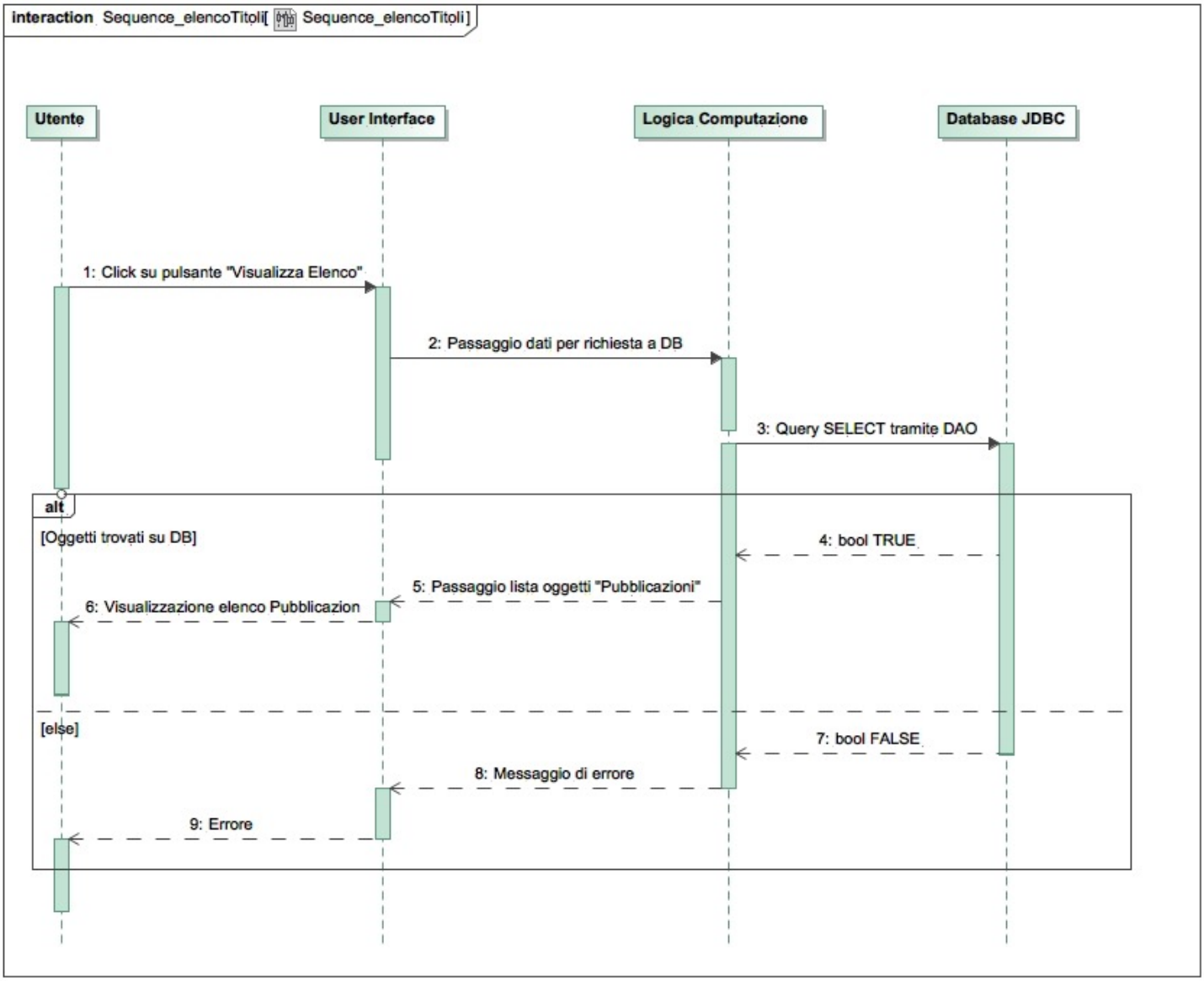
Login



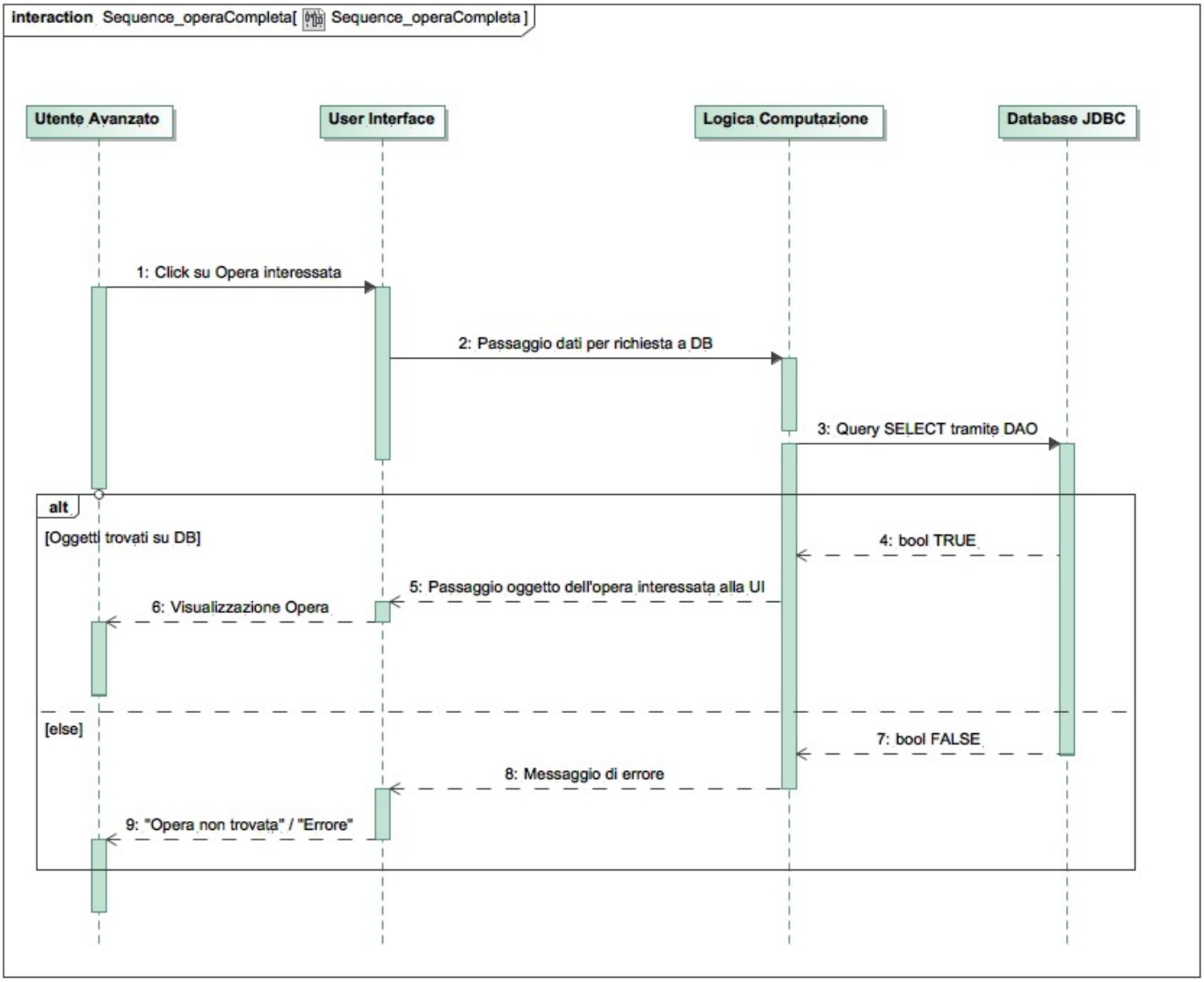
Registrazione



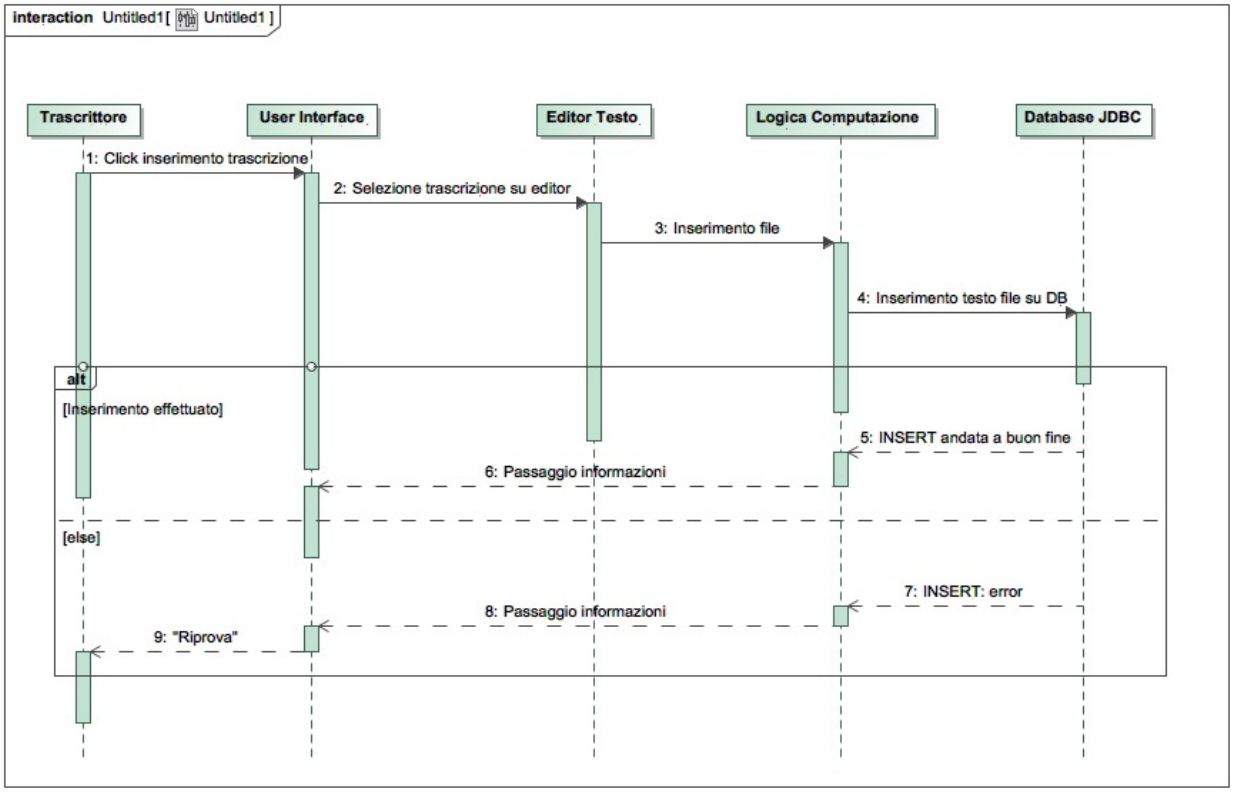
Visualizzazione Elenco Titoli



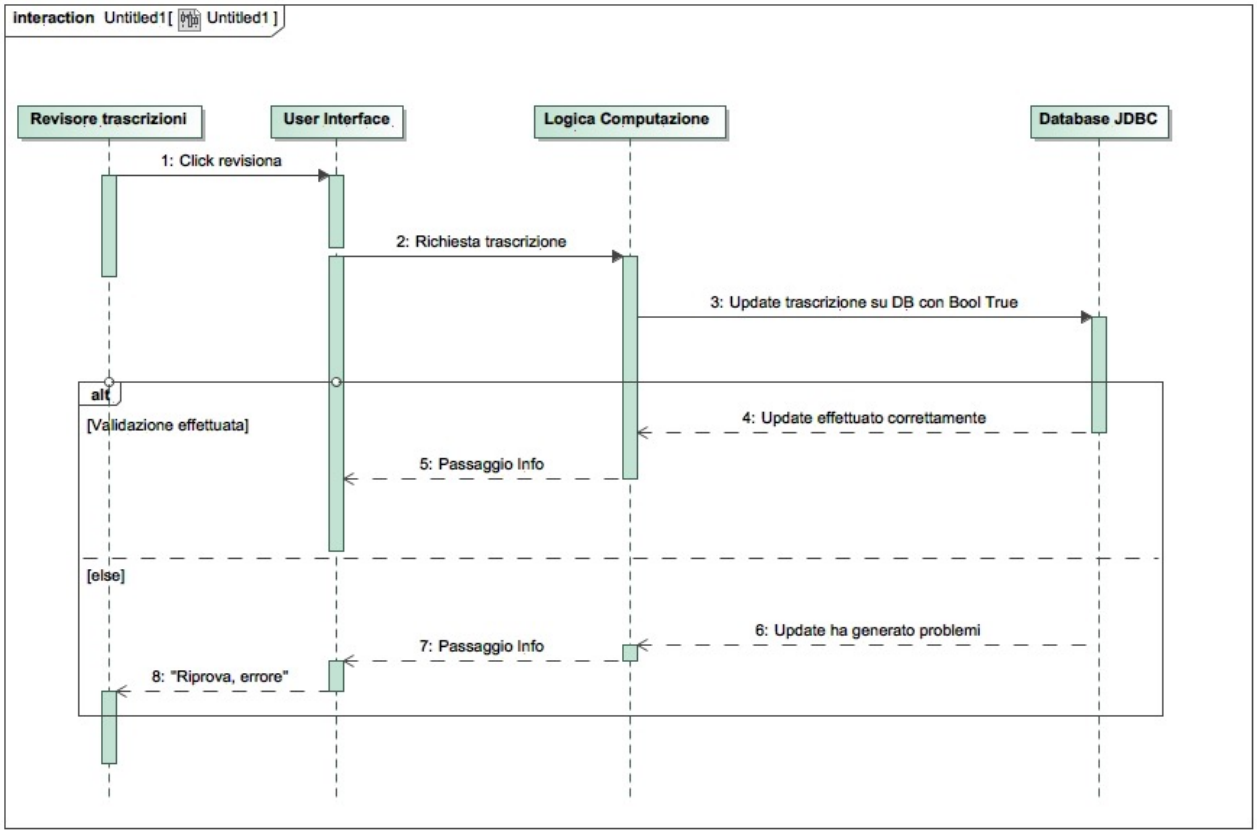
Visualizzazione Opera Completa



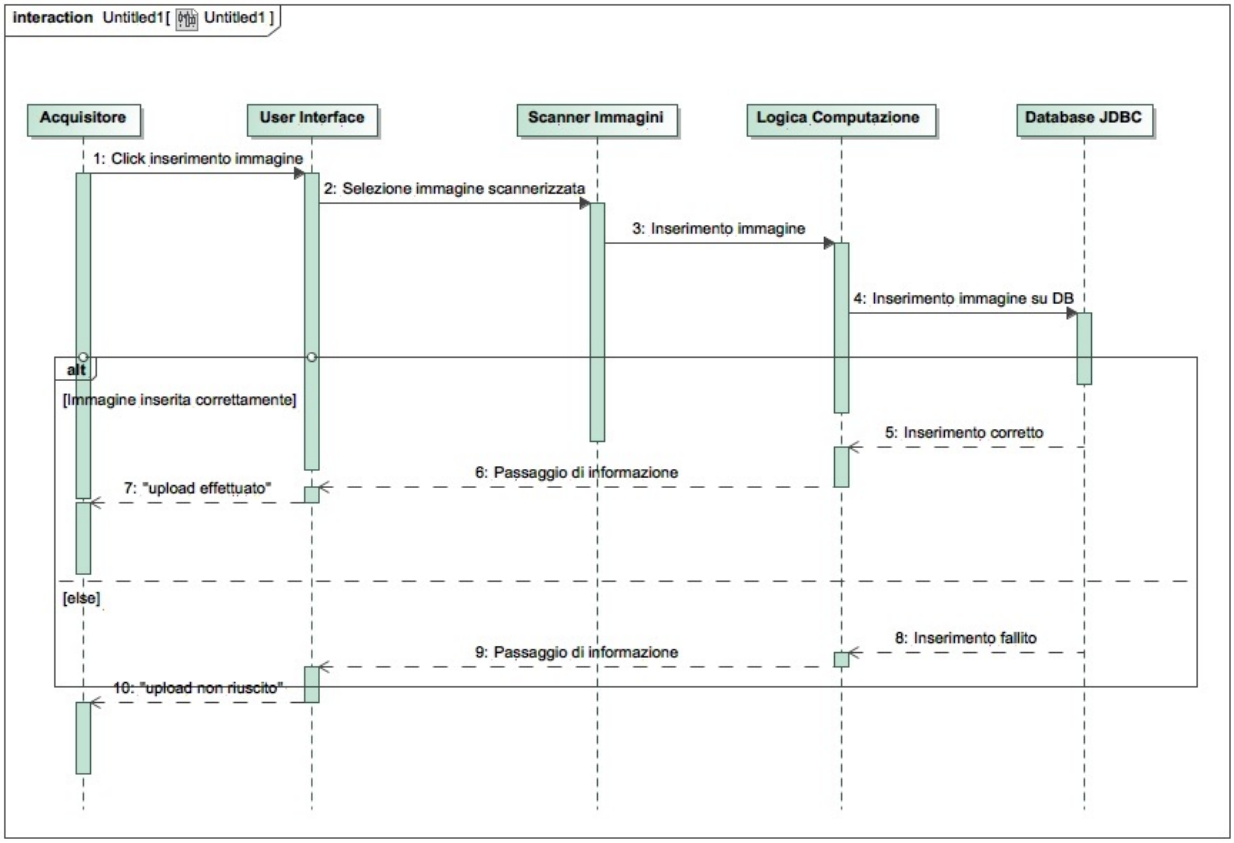
Trascrizione



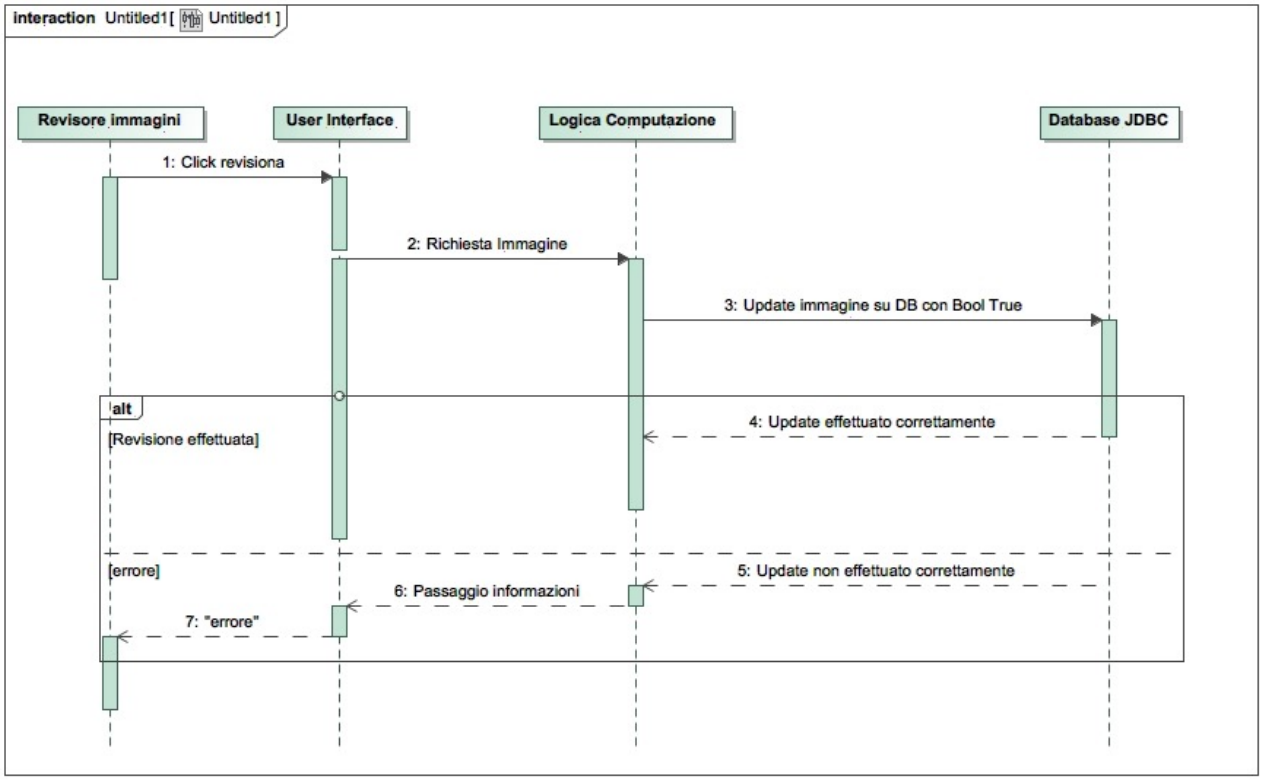
Validazione



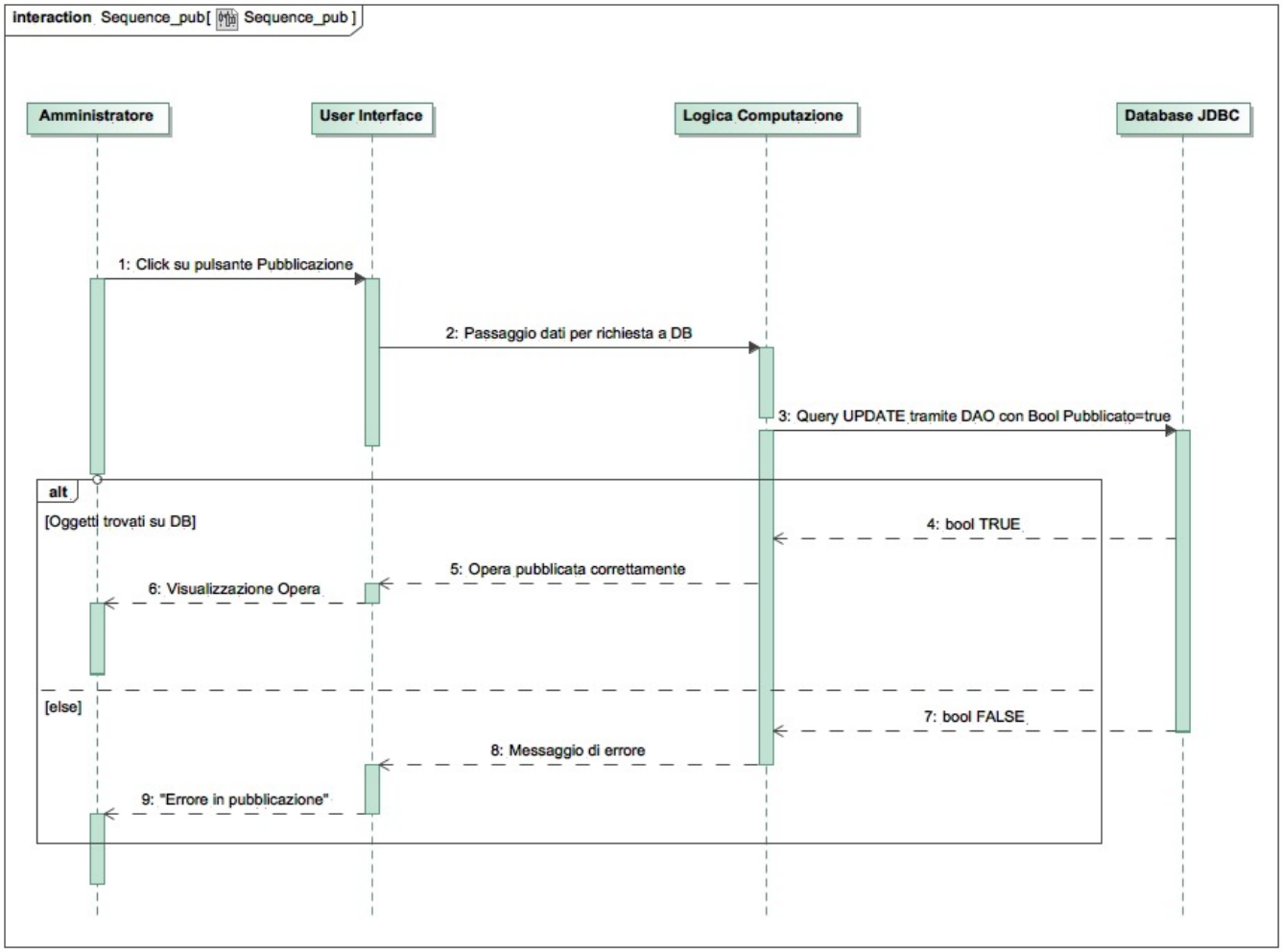
Digitalizzazione



Revisione

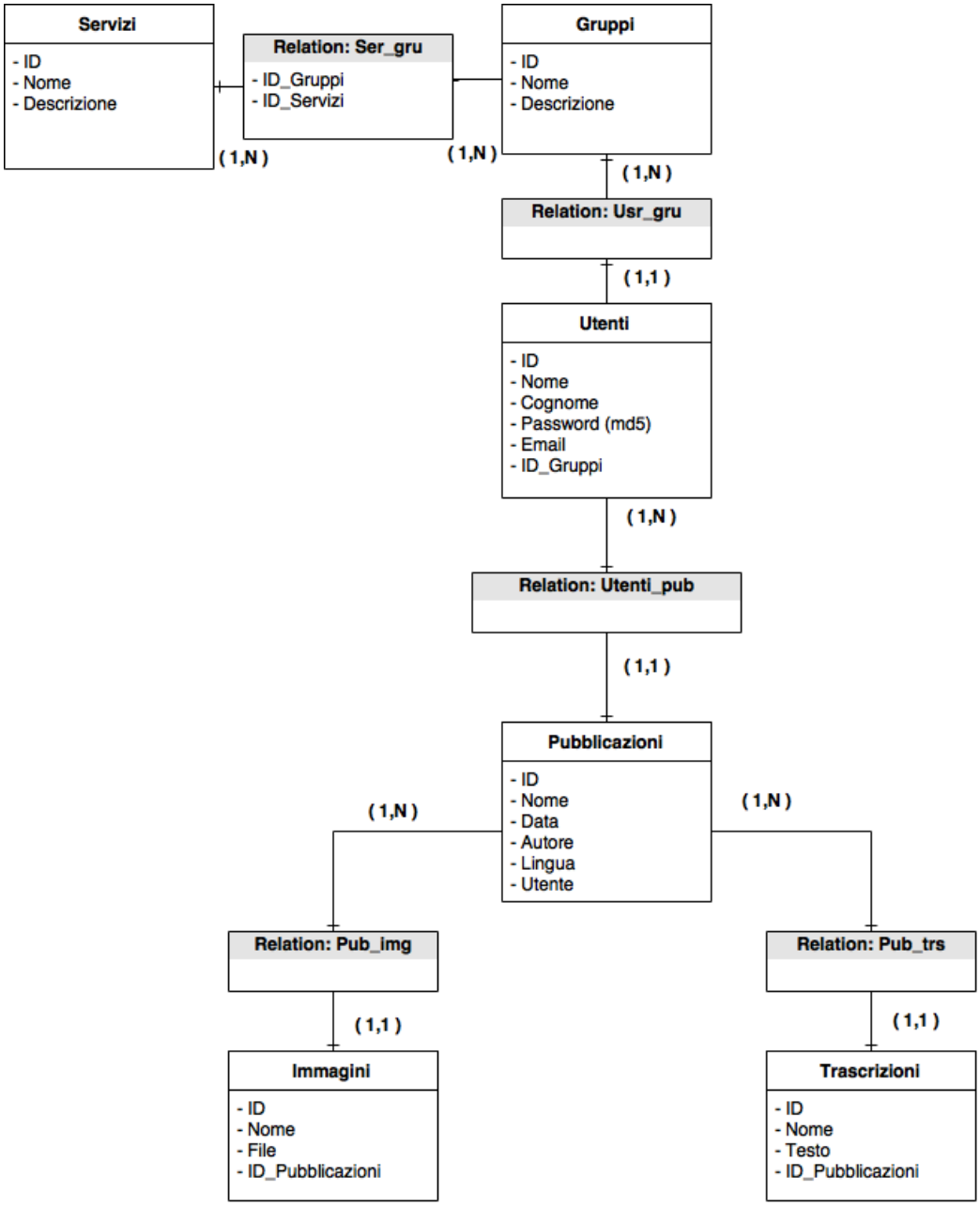


Pubblicazione



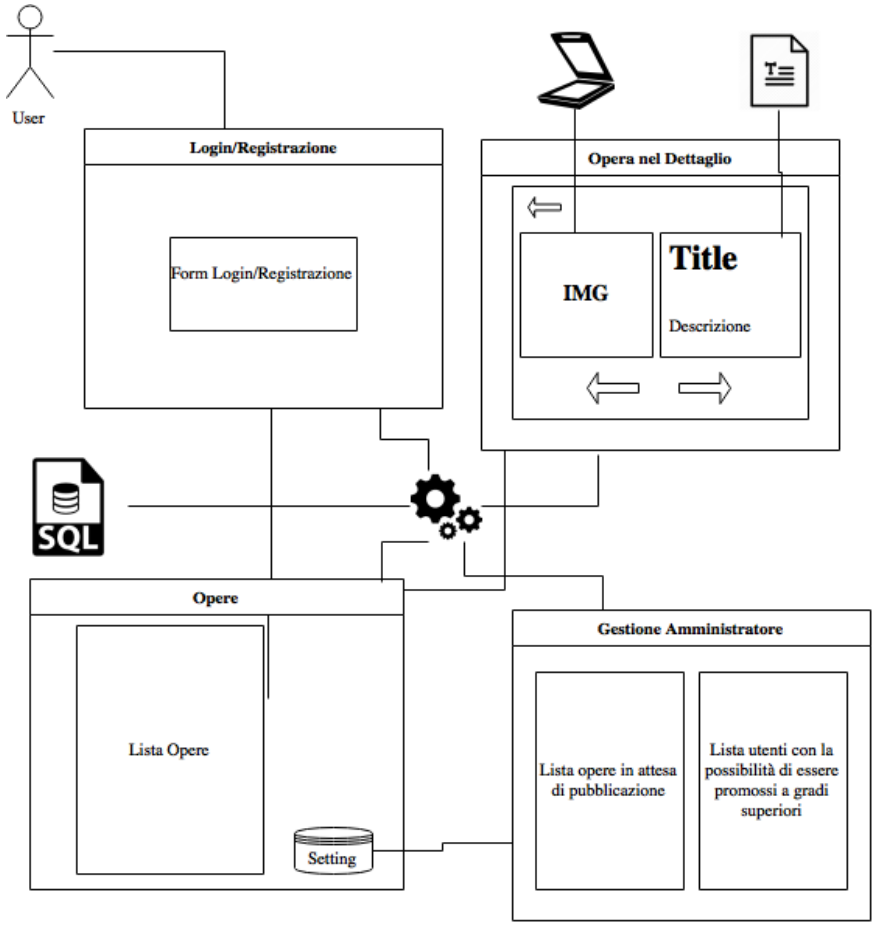
Modello ER

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 - Servizi 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.



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.



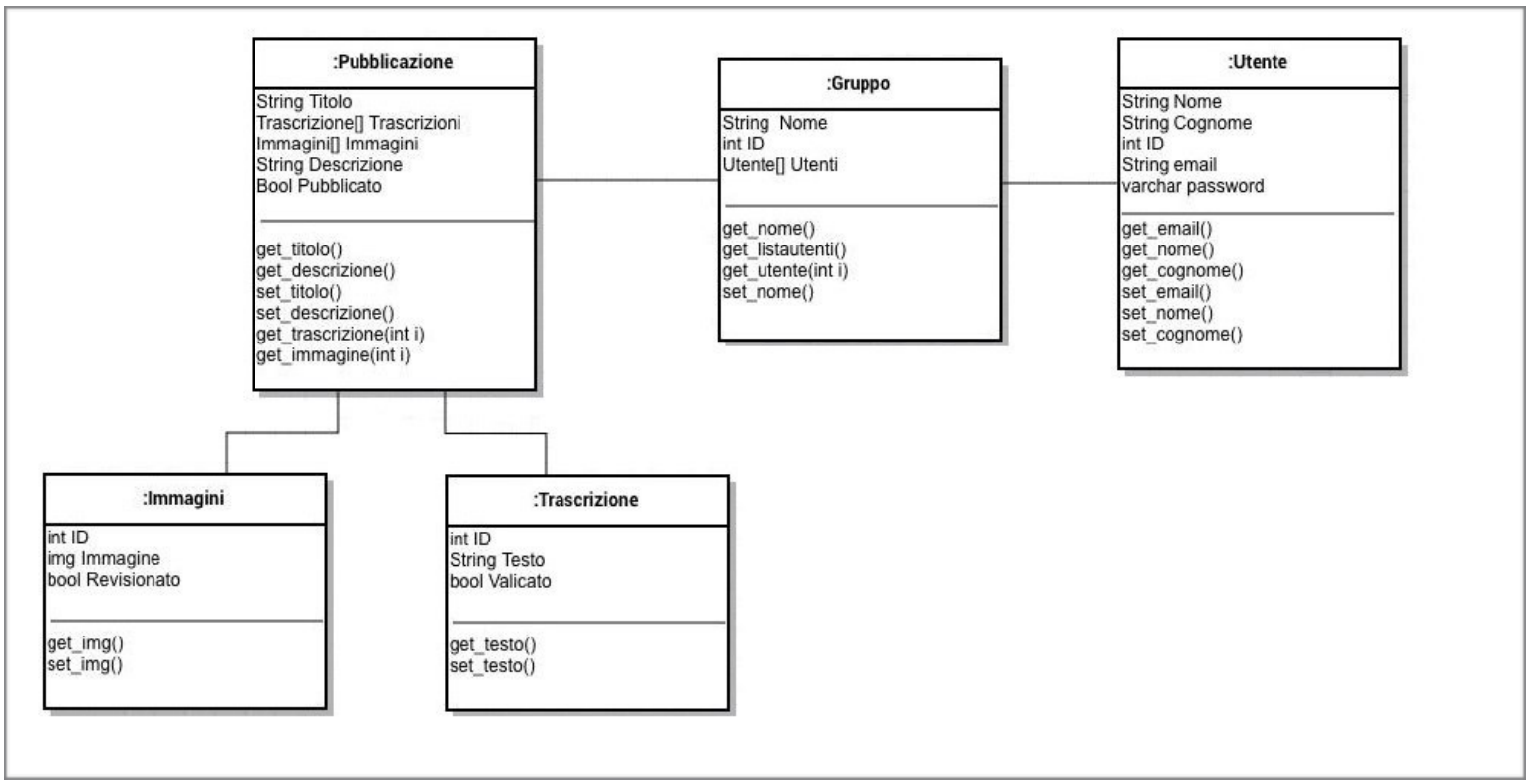
Si è immaginato quindi un portale (Web in questo caso) tramite la quale l'utente alla prima apertura non può vedere altro che la form di Login e Registrazione e solo dal momento che questa vada a buon fine egli potrà accedere alla lista delle opere disponibili, ma non alle opere nel dettaglio, poichè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

Nel caso in cui quindi l'utente sia un revisionatore 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 basso a destra un pulsante "setting" che gli permetterà di accedere alla pagina di gestione del portale nella quale vi sarà la lista dele opere in attesa di pubblicazione e la lista degli utenti con il relativo grado, permettendogli quindi di promuoverli ad uno più alto.

Software Design

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



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:

Pubblicazione: rappresenta l'opera nel dettaglio, contenente le variabili indicanti il Titolo, la descrizione e gli array relativi alle trascrizioni e alle immagini associate. Inoltre vi è una variabile booleana "Pubblicato" che nel caso in cui fosse true ci indicherebbe che l'opera è stata pubblicata dall'amministratore, false altrimenti.
I metodi inclusi in questo oggetto sono i get ed i set relativi alle variabili, con in particolare il get_immagine(int i) e get_trascrizione(int i) che ci permettono di ottenere un'immagine o una trascrizione specifica indicata tramite l'indice dell'array.

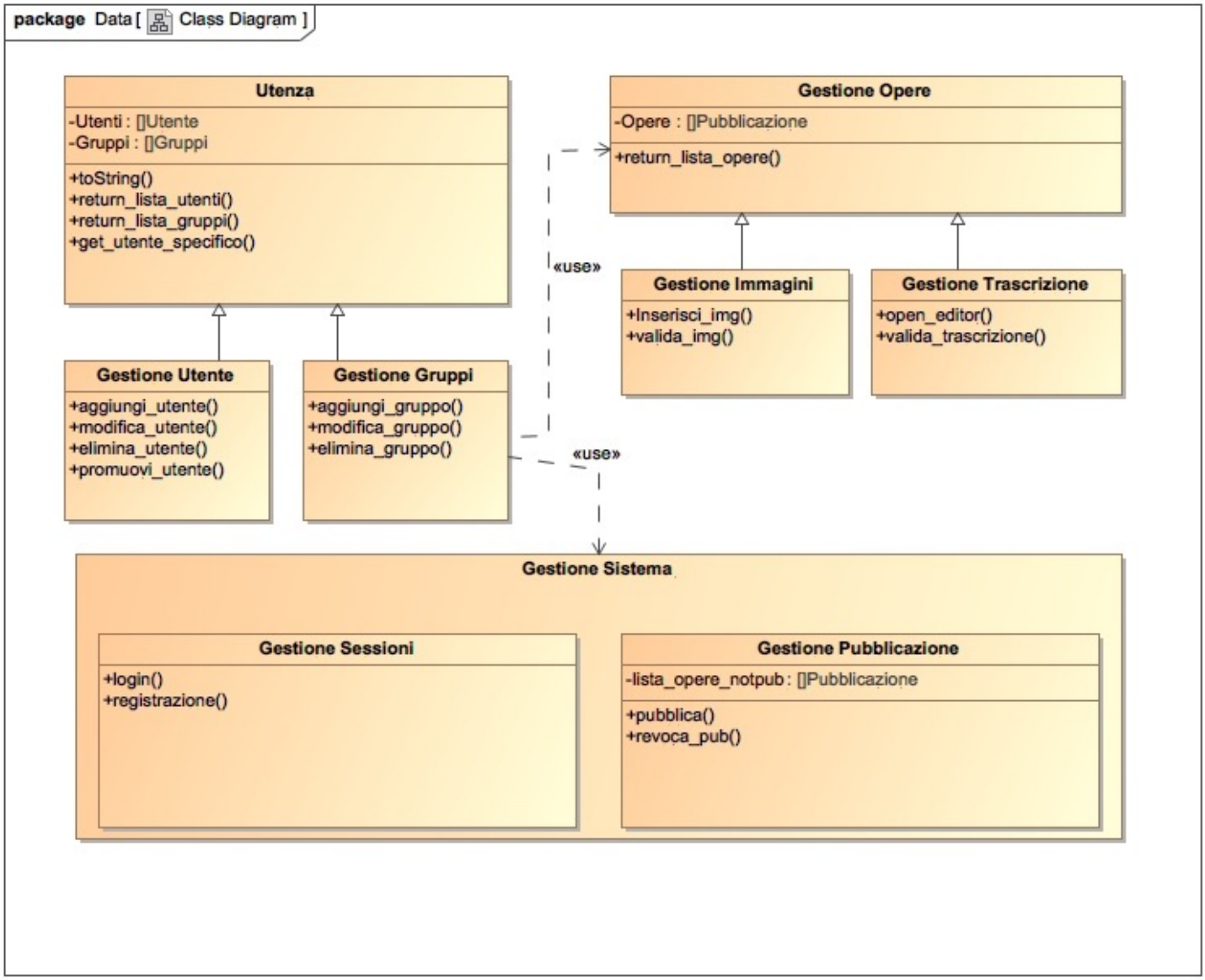
Immagini: rappresenta la prima componente di un'opera, identificata da una variabile ID, da un file immagine ottenuto tramite la scannerizzazione e da un booleano che ci identifica lo stato di revisione; se settato a true l'immagine è passata dalla revisione e quindi accettata, false altrimenti. I metodi 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'editor, esterno o meno, 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.
I metodi dell'oggetto fanno riferimento al get/set del testo della trascrizione.

Gruppo: oggetto rappresentante le categorie di utente, come ad esempio amministratore, trascrittori, revisionatori, etc.. ed è caratterizzato da un ID identificativo, un nome, e un'array contenente gli utenti appartenenti al gruppo. I metodi dell'oggetto sono i get/set, in particolare riferiti alla lista completa degli utenti ed ad un'utente specifico ottenuto tramite indice di array.

Utente: rappresenta l'utente vero e proprio ed è caratterizzato da dati anagrafici quali nome e cognome, dati di accesso quali email e password e l'identificatore ID.
I metodi disponibili in tale oggetto sono i get e set generici di tutte le variabili eccetto password ed ID.

Modello - Class Diagram - *Package: bibliotecadigitale.Controller*



[Le relazioni <<use>> indicano un comportamento nel class diagram relativo all'utilizzo astratto e non implementativo, poichè nella realtà il codice implementato relativo per esempio ai gruppi non andrà ad intaccare il codice relativo alla Gestione Opere o della Gestione del Sistema , rompendo quindi il concetto di "Modularità" .

Tali relazioni sono state utilizzate quindi solo per mostrare l'effettivo collegamento tra le classi a livello semantico]

Descrizione Modello - Class Diagram - *Package: bibliotecadigitale.Controller*

Dopo aver effettuato l'identificazione degli oggetti, si è proceduto all'identificazione più a basso livello delle classi del nostro sistema, raffinando in modo graduale quelle degli oggetti analizzati nella fase precedente e introducendo nuove classi riferite alla parte **Controller** e **Presentation** del nostro pattern adottato (MVC).

Utenza: contenente due array di oggetti, uno di utenti e uno di gruppi. I metodi della seguente classe sono l'override del toString, il return delle liste di utenti e gruppi ed il get di un utente specifico via indice.

Gestione Utente: sottoclasse di "Utenza" , che oltre ai metodi della superclasse aggiunge metodi per l'aggiunta, la modifica e l'eliminazione di un'utente nonché la relativa promozione dello stesso.

Gestione Gruppi: sottoclasse di "Utenza" , che oltre ai metodi della superclasse aggiunge metodi per l'aggiunta, la modifica e l'eliminazione di un gruppo.

Gestione Opere: classe contenente l'array di oggetti di tipo Pubblicazione ed il relativo metodo per stampare a video la lista di quest'ultimi (ci risulterà utile nella home del nostro portale).

Gestione Immagini: sottoclasse di "Gestione Opere" , che oltre ai metodi della superclasse aggiunge metodi per l'inserimento e la validazione di un'immagine.

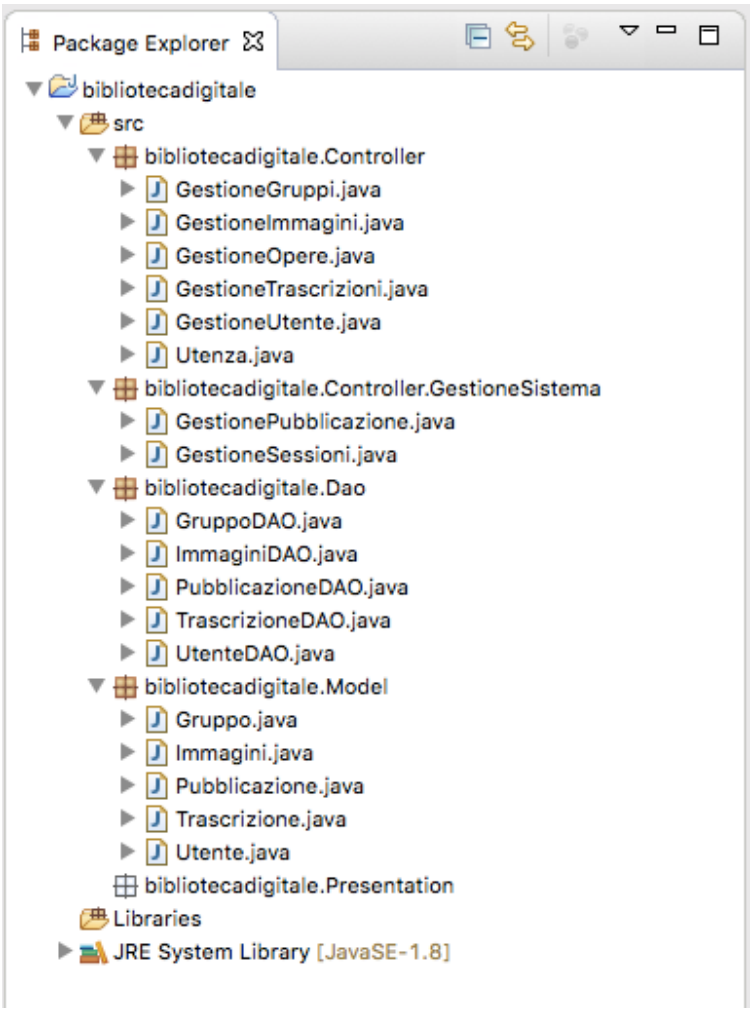
Gestione Trascrizioni: sottoclasse di "Gestione Opere" , che oltre ai metodi della superclasse aggiunge i metodi "open_editor" e "valida_trascrizioni"; il primo per aprire l'editor TEI (esterno o interno) e importare il testo prodotto, il secondo servirà all'utente che revisiona le trascrizioni per la validazione di quest'ultima.

Gestione Sistema: classe utilizzata per la modularità e per il raggruppamento di due sottoclassi principali, quali "Gestione Sessioni" e "Gestione Pubblicazione".

Gestione Sessioni: sottoclasse di "Gestione Sistema" , che implementa i metodi necessari per il Login e la Registrazione, che conterranno a loro volta eventuali controlli tramite query su DB.

Gestione Pubblicazione: sottoclasse di "Gestione Sistema" , contenente l'array delle Pubblicazioni non pubblicate e quindi con il parametro booleano settato a "false". I metodi implementati in questa sottoclasse saranno quelli relativi alla pubblicazione di un opera specifica dell'array precedente, o l'eventuale revoca di una pubblicazione avvenuta.

Sviluppato quindi l'object model che ci ha permesso di identificare gli oggetti principali del nostro sistema e sviluppato il class diagram che ci ha permesso di osservare ad alto livello la composizione delle nostre classi appartenenti al modulo Controller del nostro pattern, il team si è fatta quest'idea di come potrebbe essere sviluppata ed impostata eventualmente la WebApp della biblioteca digitale in un IDE come Eclipse .



Nella rappresentazione ovviamente non sono riportate le librerie JAR importate nel nostro progetto quali:

- freemarker.jar
- mysql-connector-java-5.1.37-bin.jar

E inoltre non sono riportate le varie cartelle in bibliotecadigitale.Presentation contenenti i relativi file di template (html/ js/ css) che verrano caricati tramite opportuna servlet indicate nel "Web.xml" tramite dichiarazione e mapping.

Class Diagram Completo