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

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.

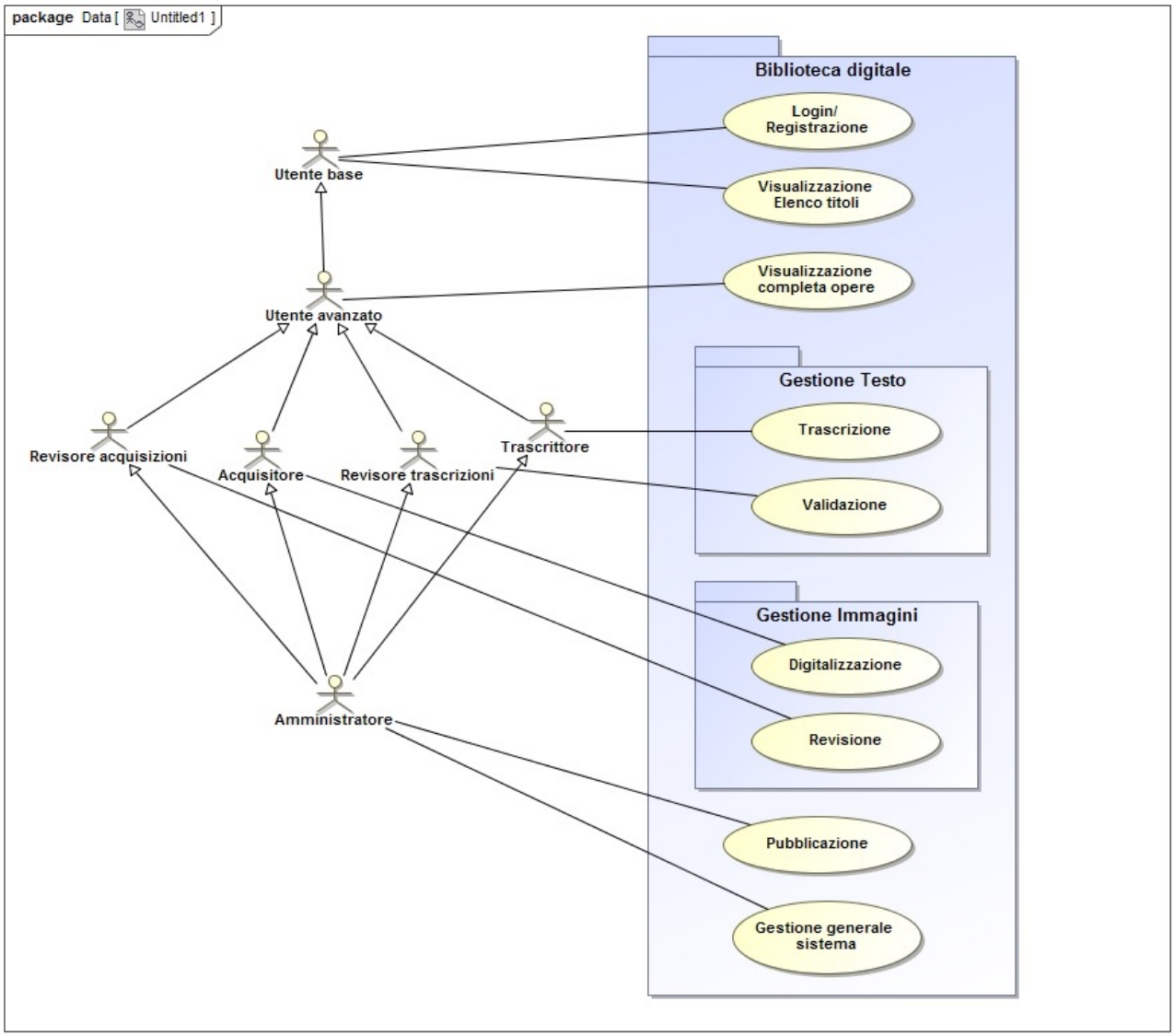
Requisiti Funzionali (FR)

- **Login/Registrazione:** Funzione senza la quale un utente non potrà accedere alle funzionalità del sistema.
- **Visualizzazione Elenco Titoli:** Funzionalità che mostra l'elenco di tutti i testi disponibili nella biblioteca.
- **Visualizzazione Completa Opere:** Disponibile solo per l'Utente Avanzato e permette la visualizzazione delle immagini e le trascrizioni di una specifica opera.
- **Trascrizione:** Disponibile solo al Trascrittore ed è la funzione che permette il passaggio da immagine a testo dell'opera.
- **Validazione:** Permette al Revisore Trascrizioni di validare una trascrizione inserita.
- **Digitalizzazione:** Funzione di inserimento dell'immagine di una o più pagine di un testo da parte dell'acquisitore.
- **Revisione**: Permette al Revisore Acquisizioni di validare una o più immagini inserite.
- **Pubblicazione:** Consente all'amministratore del sistema di rendere disponibile agli Utenti Avanzati una o più opere digitalizzate ed eventualmente trascritte.
- **Gestione Generale del Sistema:** 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.

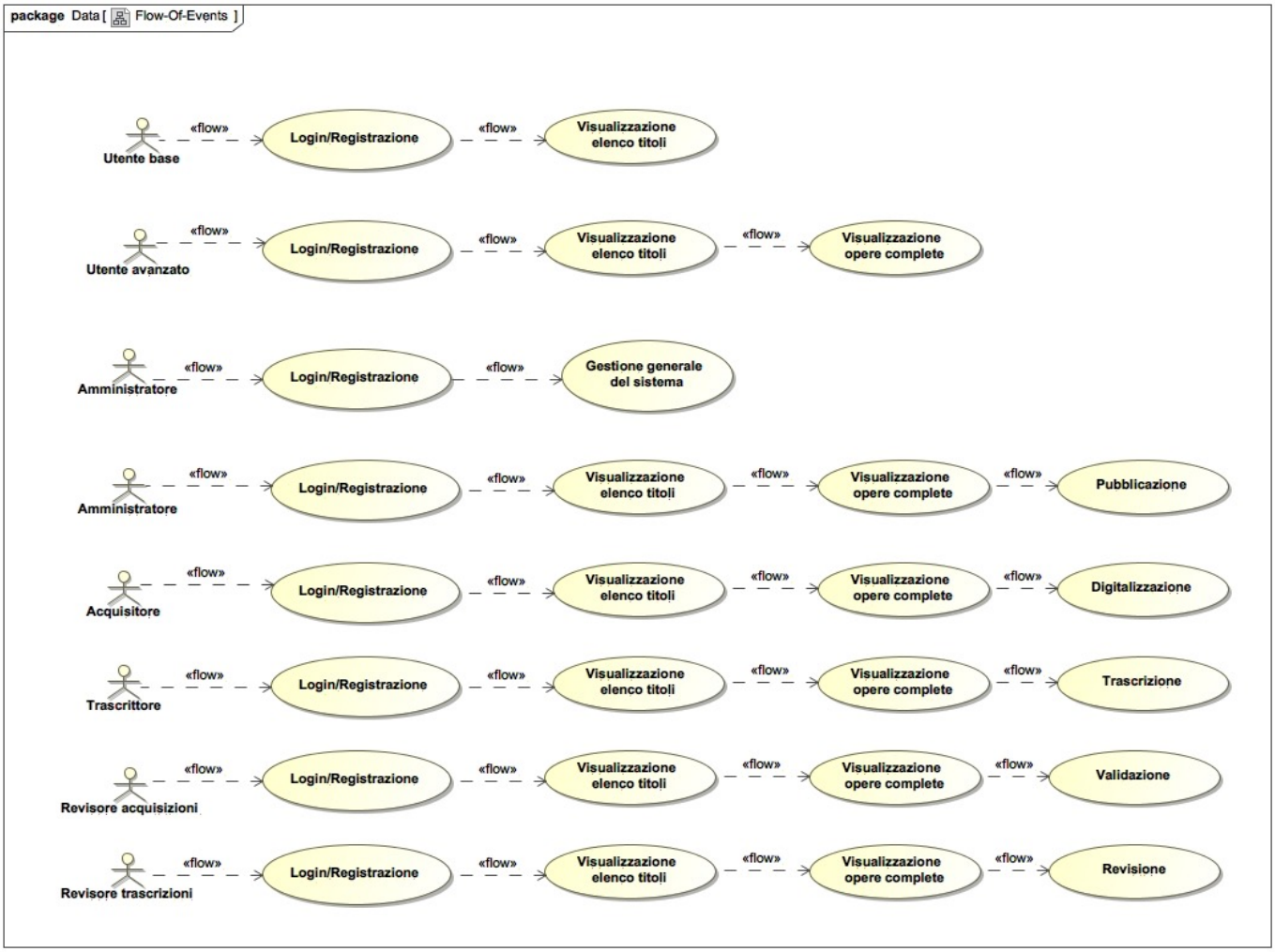
Use Case Diagram



Scenari

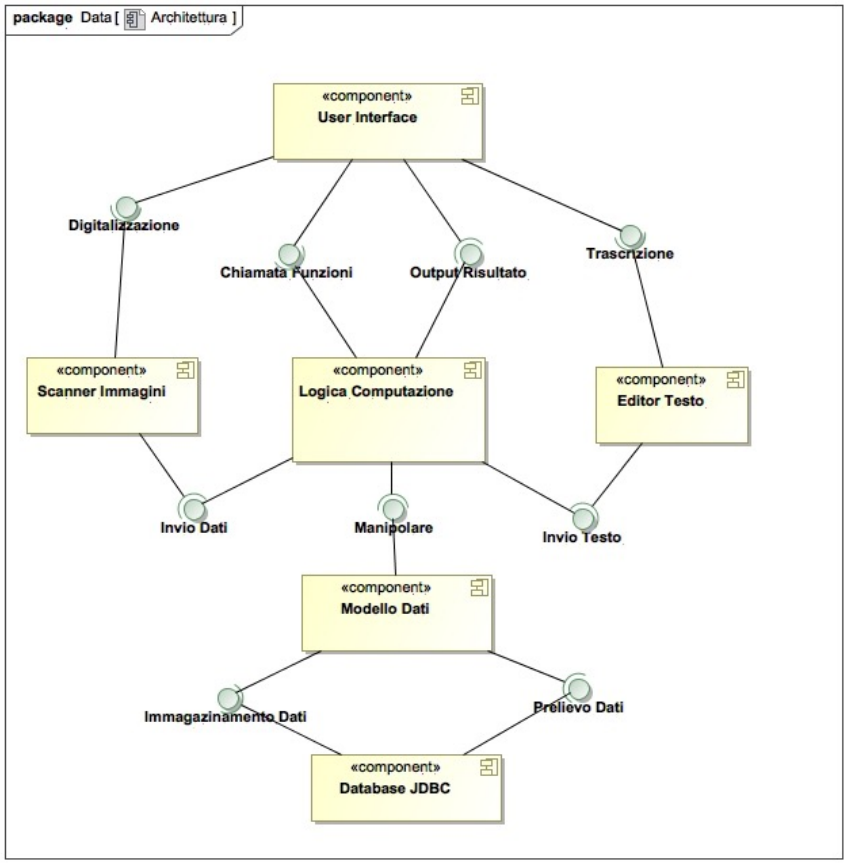
Nei seguenti flussi di eventi verranno rappresentati in sequenza l'utilizzo degli UC (Use-case) da parte di ogni tipologia di attore del nostro sistema.

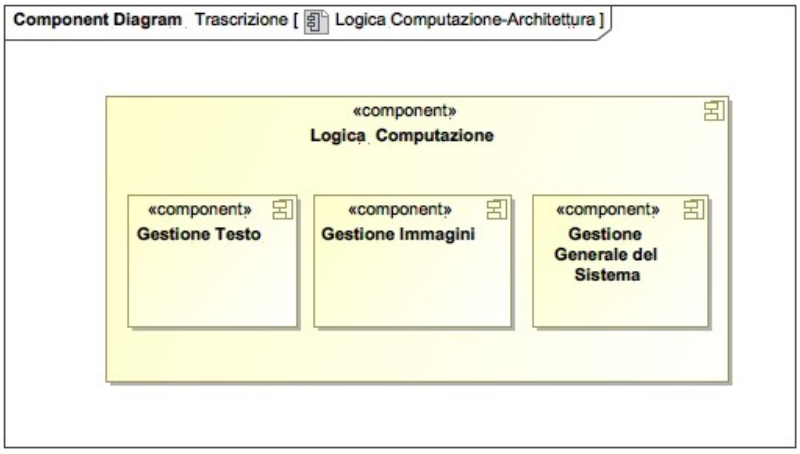
Questi comporranno gli scenari che ci serviranno successivamente nell'analisi.



System Design

Modello dell'Architettura del sistema





Descrizione dell'Architettura

Il team ha deciso di decomporre il sistema architetturale in tre 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.

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.

L'editor testo è la nostra componente (implementabile internamente o richiamabile esternamente) che rappresenta l'editor 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:

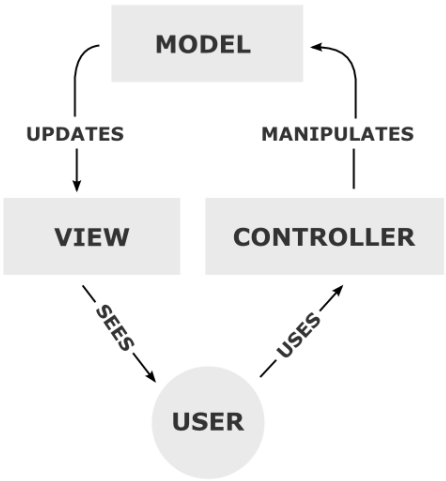
- Ace: editor testuale Web-Based
- EditTEI: editor specifico per il formato TEI
- Wed: editor testuale Web-Based

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 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.

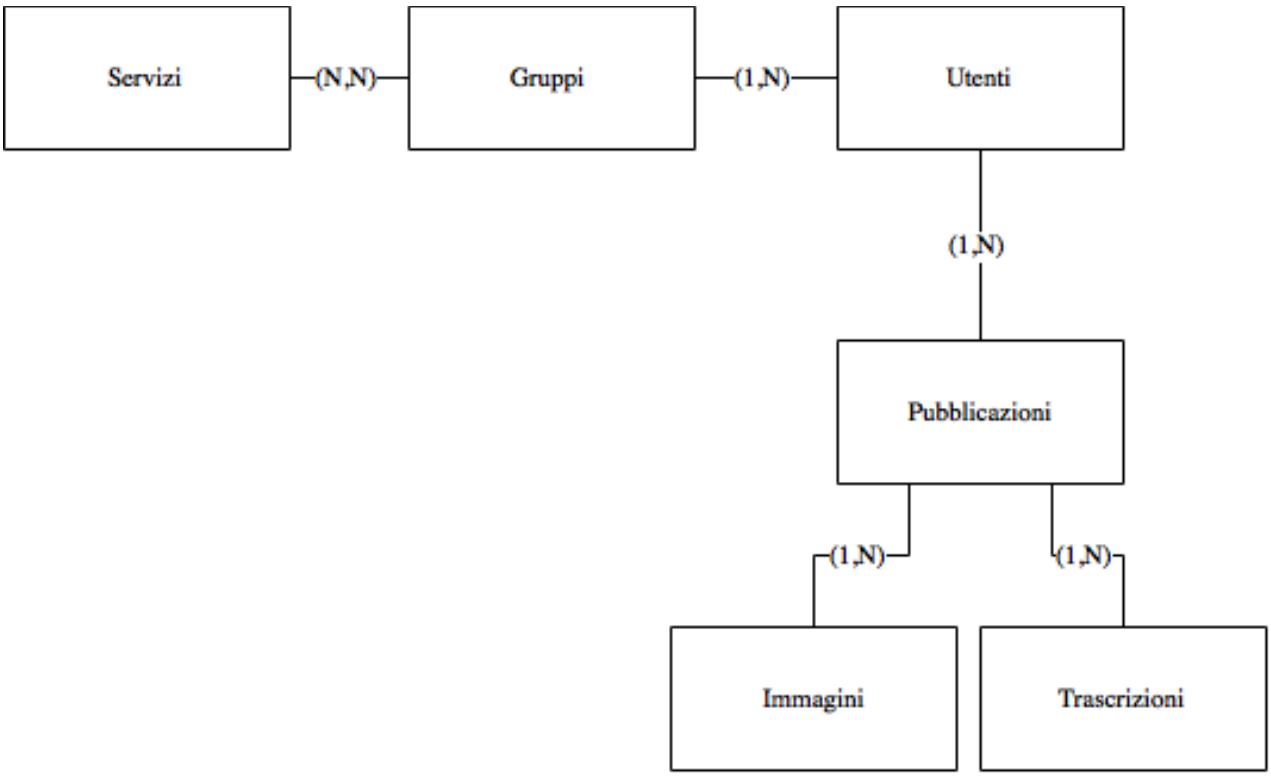
Perché è stato scelto il seguente pattern?

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) avrà vita più facile e agevole nella manutenzione del codice.

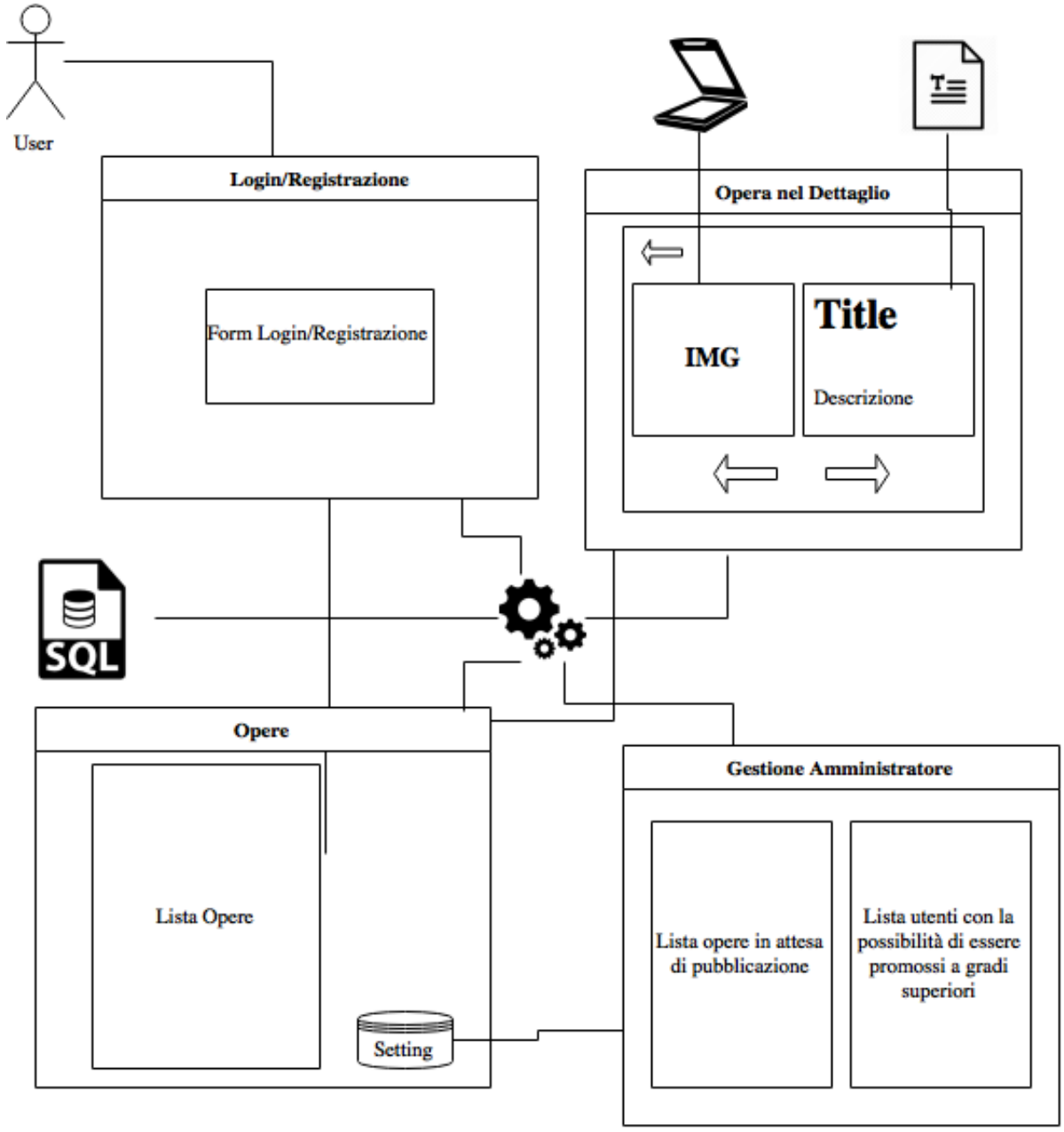
Se per esempio si decidesse di apportare delle modifiche al portale Biblioteca in modo da alterare i dati e visualizzarli in modo differente da quello già implementato, sarà sufficiente modificare solo il "Presentation Layer" senza minimamente intaccare le altre due componenti del MVC, ovvero il controller 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.

Modello ER

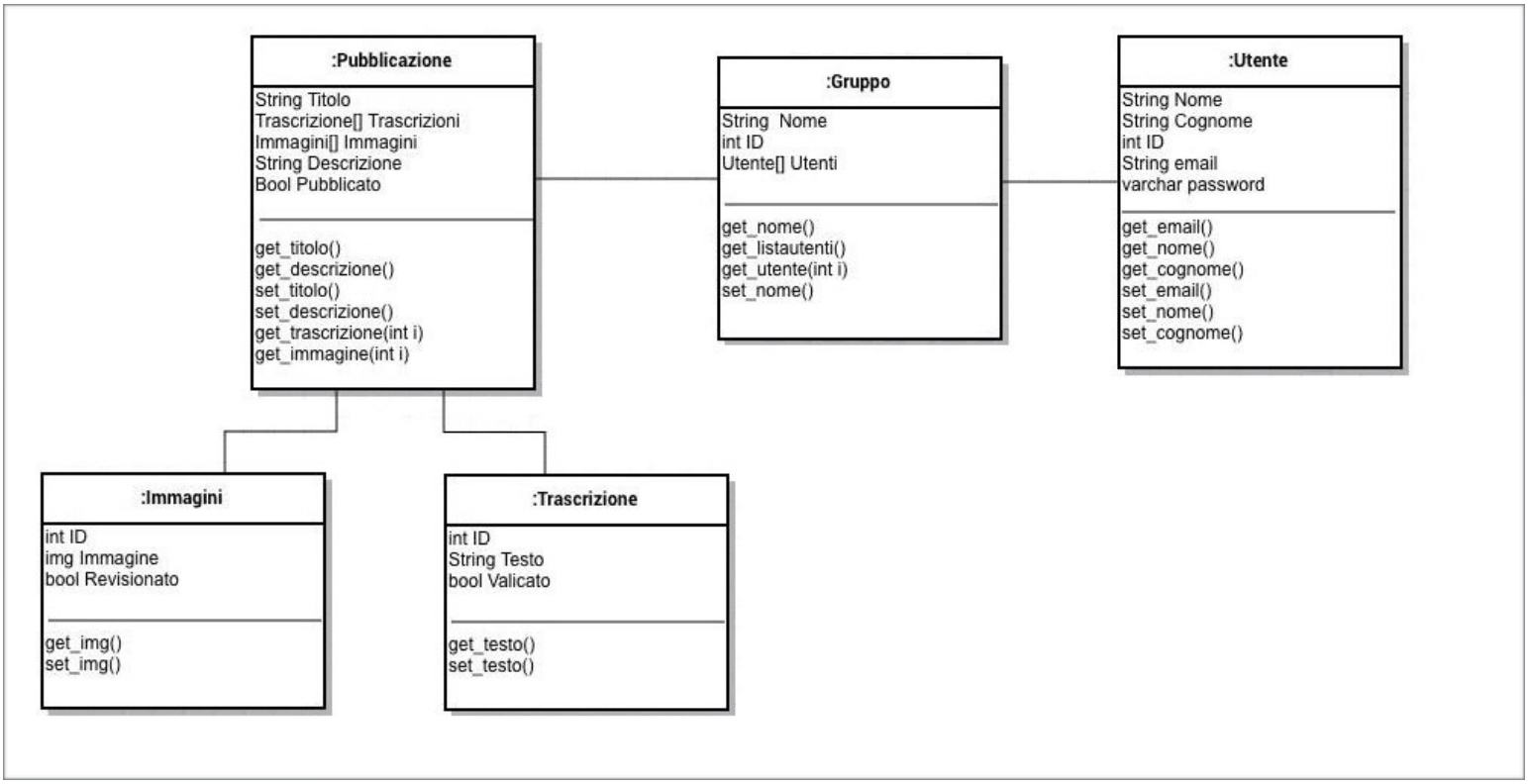

Servizi
(N,N)
Gruppi
(1,N)
Utenti
(1,N)
Pubblicazioni
(1,N)
(1,N)
Immagini
Trascrizioni

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



Object Design

Modello - Object Diagram



Descrizione Modello - Object Diagram

Gli oggetti identificati sono i seguenti e costituiranno la parte **Model** in base al riferimento al pattern adottato:

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.

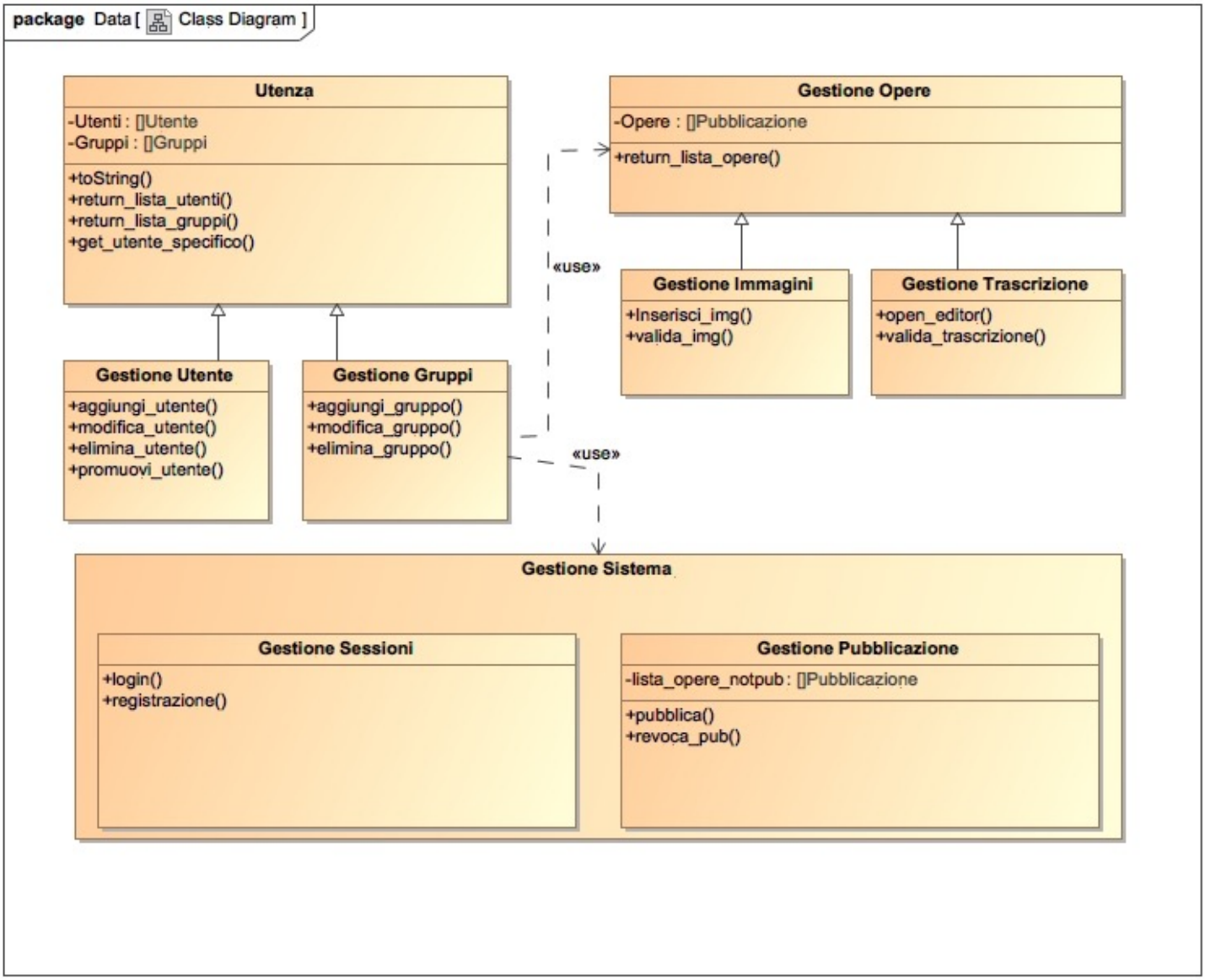
Immagini: rappresenta la prima componente di un'opera, identificata da una variabile ID, da un file immagine ottenuto tramite la scannerizzazione e da un booleano che ci identifica lo stato di revisione; se settato a true l'immagine è passata dalla revisione e quindi accettata, false altrimenti. I metodi 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



[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

Dopo aver effettuato l'identificazione degli oggetti, si è proceduto all'identificazione più ad alto livello delle suddette classi che costituiranno la parte **Controller** del nostro MVC, ovvero la logica di business che si cela dietro il nostro applicativo:

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.