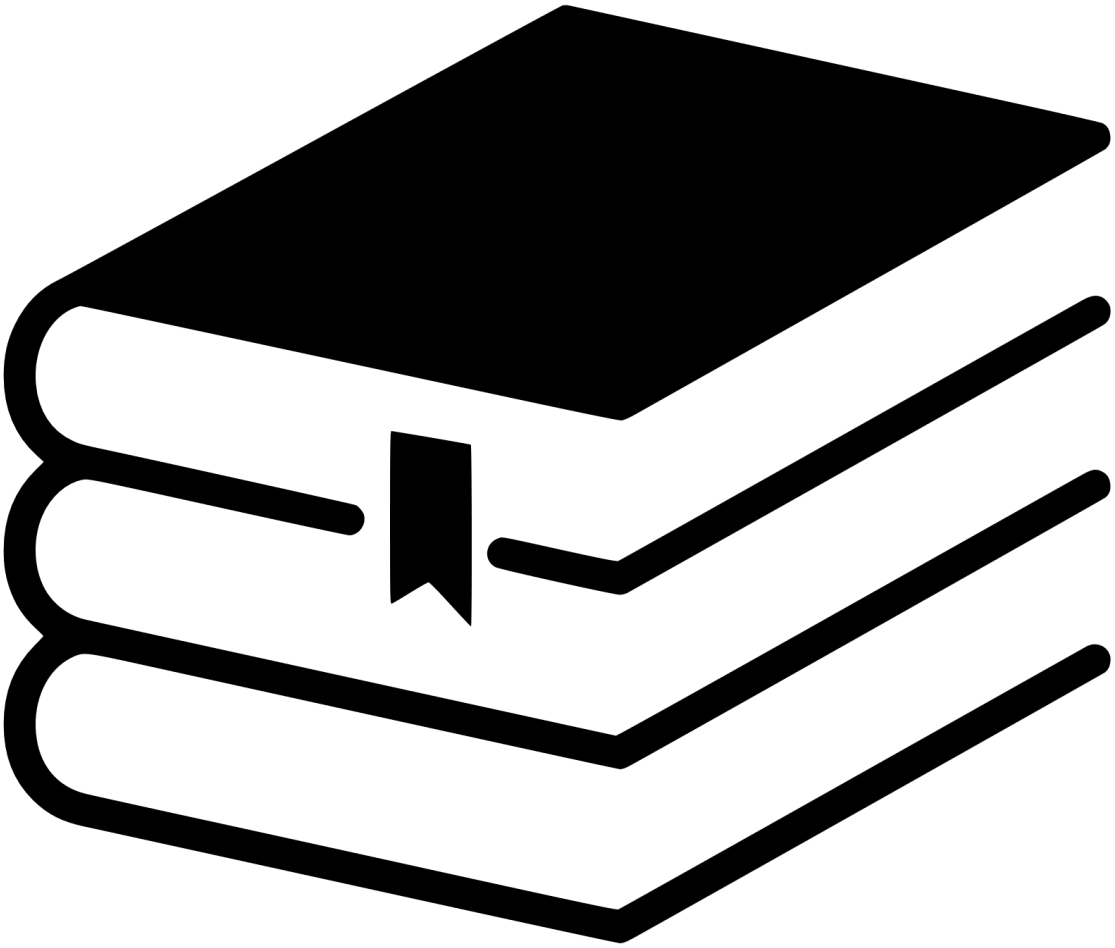
REQUIREMENTS ANALYSIS DOCUMENT

OBJECT ORIENTED SOFTWARE DESIGN

UNIVERSITA' DEGLI STUDI DELL'AQUILA

BIBLIOTECA DIGITALE



ANNO 2017/2018

Index

1. Introduzione

Questo progetto è stato realizzato da:

Antokhi Stanislav, Mat. 244295

D'Angelo Andrea, Mat. 248817

Zappacosta Francesco, Mat. 249168

Per il corso 2017/2018 di Object Oriented Software Design all'Università degli studi dell'Aquila.

L'intera documentazione e codice si può trovare nel repository di GitHub: https://github.com/dangeloandrea14/Project.

Come punto di riferimento, sia per la costruzione di questo documento che per la restante documentazione, abbiamo preso il libro *"Object-Oriented Software Engineering. Using UML, Patterns and Java"* di Bernd Bruegge e Allen H. Dutoit.

Questo progetto è stato realizzato come esame finale del corso di Object Oriented Software Design, e la sua funzione è quella di fornire un servizio di biblioteca digitale per la visualizzazione online di manoscritti del patrimonio bibliografico antico della citta dell'Aquila.

1.1 Scopo del Sistema

Lo scopo del progetto è la realizzazione di una biblioteca digitale formata da manoscritti per un totale di 60.000 carte (ms. sec. XV-XIV) contenenti memorie storiche della città dell'Aquila, consentendo la consultazione dei manoscritti, che devono essere digitalizzati. Il sistema deve quindi sia permettere l'upload e la digitalizzazione delle varie opere, che la loro consultazione da parte degli utenti.

1.2 Definizioni, acronimi, abbreviazioni

Con "manoscritto" intendiamo qualsiasi pubblicazione voglia essere caricata o letta sul sistema.

1.3 Fonti

Come fonte per tutti i requisiti abbiamo utilizzato la specifica del progetto pubblicata. Come riferimento per tutta la strutturazione del progetto e della documentazione, invece, abbiamo utilizzato il libro di testo consigliato.

2. Sistema corrente

Non c'è alcun sistema correntemente progettato per questo dominio di applicazione, e non è dunque presente alcun punto di riferimento a priori per le scelte di design che sono state prese. Queste scelte sono accuratamente descritte nella documentazione e sono state prese in gruppo dopo accurate riflessioni e costruzione di scenari che hanno portato alla struttura del modello Use case.

La costruzione degli scenari è stata fatta in modo tale da essere il più coerente possibile con il dominio dell'applicazione descritto dalla specifica.

3. Sistema proposto

3.1 Panoramica

Il sistema proposto garantisce un ruolo specifico ad ogni utente con dei relativi permessi gestibili dall'amministratore, che è unico. I ruoli e i permessi sono gestiti da un database interno al sistema, dove vengono memorizzati anche gli utenti registrati, le pubblicazioni e gli scan.

Solo gli utenti registrati come Uploader possono accedere al sistema di caricamento dei manoscritti, che è invisibile ai normali utenti. Il sistema gerarchico di ruoli, quindi, garantisce ad ogni utente solo una parte delle funzionalità del sistema, per mantenere il sistema sicuro e funzionante.

3.2 Requisiti funzionali

Il sistema deve:

➢ Permettere agli utenti di registrarsi

➢ Permettere agli utenti di fare il login

➢ Permettere agli utenti di ricercare manoscritti tramite metadati

➢ Permettere la visualizzazione dei manoscritti, con annessa trascrizione

➢ Consentire la visualizzazione del proprio profilo personale

➢ Permettere di scaricare le opere a determinati utenti

➢ Permettere di candidarsi come trascrittori

➢ Gestire l'upload delle scansioni delle pagine

➢ Permettere la revisione degli upload --

➢ Avere un editor di testo integrato per la trascrizione degli scan

➢ Consentire la gestione in back-end del sistema all'amministratore

➢ Assegnare scan da trascrivere ai trascrittori --

➢ Revisionare le trascrizioni concluse --

➢ Pubblicare trascrizioni e opere

➢ Gestire i livelli dei trascrittori

Tutti questi sono i requisiti funzionali descritti dalla specifica del progetto.

3.3 Requisiti non funzionali

3.3.1 Usabilità

Il sistema è indirizzato al più ampio spettro di utenti finali possibile, allo scopo di condividere la conoscenza dei manoscritti. Funzionalmente, quindi, il sistema deve essere il più semplice possibile da utilizzare, con label chiare ed efficaci, per ingaggiare anche il pubblico meno esperto.

3.3.2 Affidabilità

E' importante che utenti non autorizzati non possano avere accesso a risorse riservate o al management in back-end del sistema. Un accesso non autorizzato potrebbe portare alla perdita di informazioni fondamentali.

3.3.3 Implementazione

L'implementazione deve essere fatta tramite Java, JavaFX e il software Scene Builder.

3.3.4 Packaging

Il packaging deve essere realizzato seguendo il modello MVC (Model – View – Controller). E' importante che il codice sia ben diviso in sottosistemi autonomi che comunicano tra di loro. I dettagli esatti del packaging sono esplicitati nella specifica.

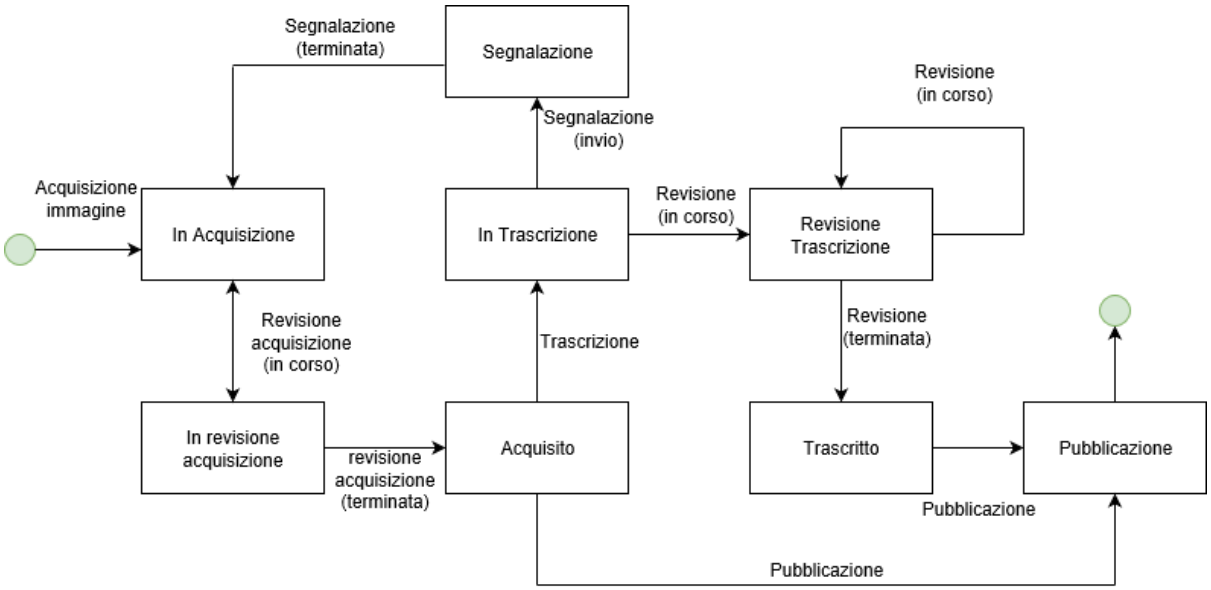
3.3.5 Cenni legali

I diritti legali di manutenzione e gestione dei manoscritti sono del tutto riservati all'azienda che gestisce. Il nostro software è stato sviluppato sotto loro autorizzazione.

3.4 Modelli del Sistema

3.4.1 Scenari

Per la costruzione degli scenari ci siamo basati su questo diagramma, che è una versione modificata e più completa di quello fornito con le specifiche.



Gli scenari basati su questo diagramma sono quelli relativi alla pubblicazione e trascrizione dei manoscritti. Andiamo ad illustrare in questa documentazione solo alcuni degli scenari costruiti:

Primo scenario:

Nome scenario: Pubblicazione immagine
Attori partecipanti: Berta: Uploader, Mario: RevisoreUpload

Flusso di eventi: 1. Berta ha un manoscritto e decide di fare la scansione della pagina 33 del manoscritto. Avvia allora la funzionalità di Upload a cui ha accesso in quanto uploader.

2. Berta carica il file attraverso la funzione di upload. Inserisce i dati relativi al manoscritto di cui sta facendo la scansione e il numero della pagina.
3. Mario vede il file. Dato che la qualità è accettabile e il testo è leggibile, decide di accettare la scansione, che va quindi ad arricchire la collezione di scansioni per quel manoscritto.

Secondo scenario:

Nome scenario: Iscrizione come Trascrittore

Attori partecipanti: Bob: Utente, John: Amministratore

Flusso di eventi: 1. Bob, utente utilizzatore del servizio da tempo, decide di voler aiutare la causa di

Condividere conoscenza della citta dell'Aquila e aiutare con la trascrizione degli scan. Per fare questo, attiva la funzionalità "Proponiti come trascrittore" dal pulsante che si trova nel suo profilo.

4. John, l'amministratore, vede che Bob si è proposto come trascrittore. Attraverso i dati in suo possesso, decide che Bob è un ottimo utente e di conseguenza il suo aiuto sarebbe utile, e lo accetta come trascrittore.
5. Bob ora può accedere alle funzionalità esclusive per i trascrittori.

3.4.2 Modello use case

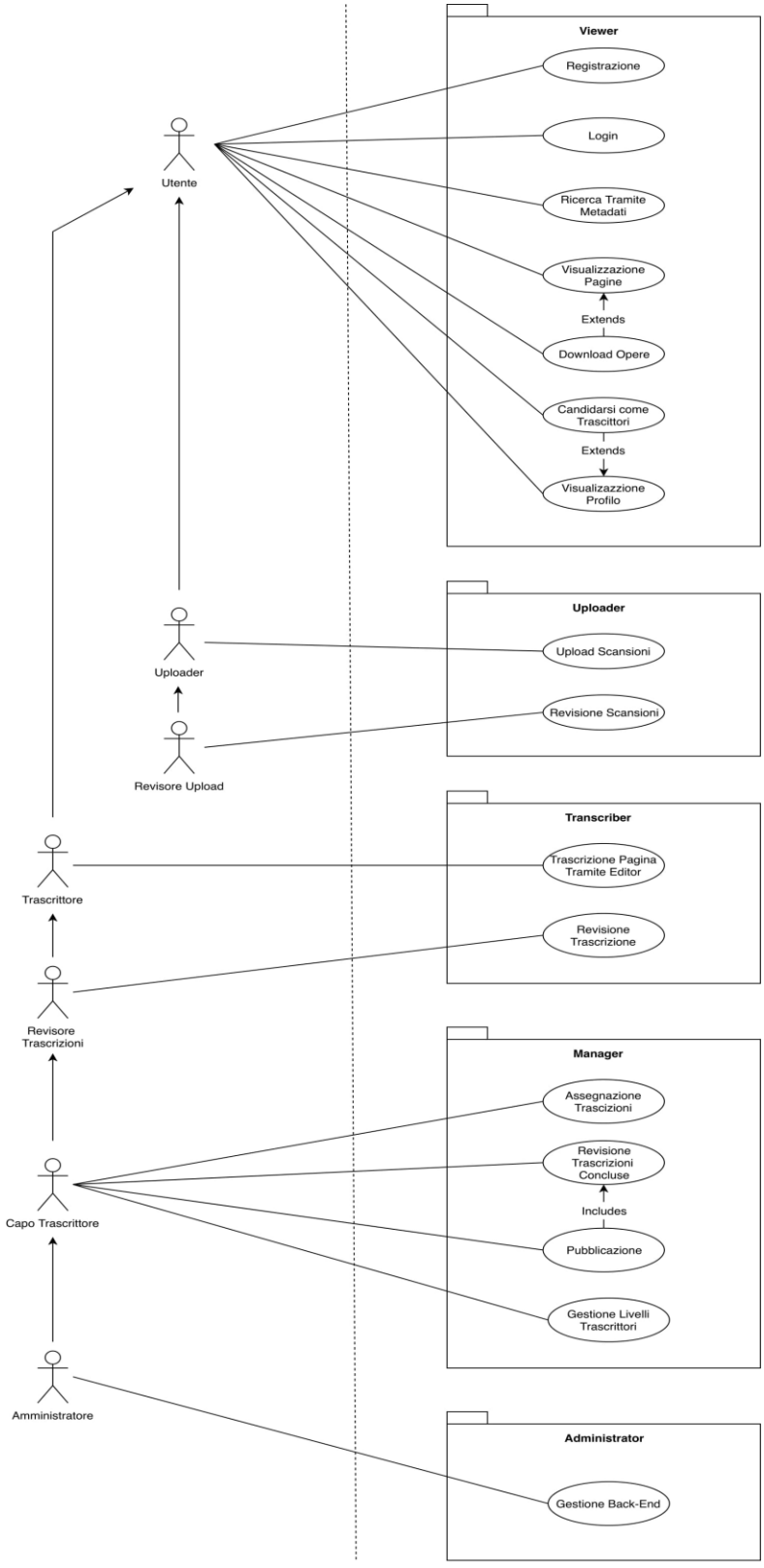
Il modello use case è stato costruito a partire dalla generalizzazione degli scenari che abbiamo costruito. E' anche specificato nello schema la limitazione ai confini del sistema. Alcune note sul diagramma use case UML:

1) Il database utilizzato nel sistema è stato considerato come interno e, di conseguenza, non un attore. Questa è una scelta di design: se

avessimo considerato un server o database proprietario dell'organizzazione, lo avremmo trattato come attore.

2) Volendo si potrebbero aggiungere diversi use case per estendere use case esistenti in caso di eccezioni o comportamenti imprevisti.
   Dato che lo scopo del modello use case è quello di essere il più chiaro possibile anche per i clienti, si è voluto evitare di introdurre ulteriore complessità se l'use case estendeva un solo altro use case.

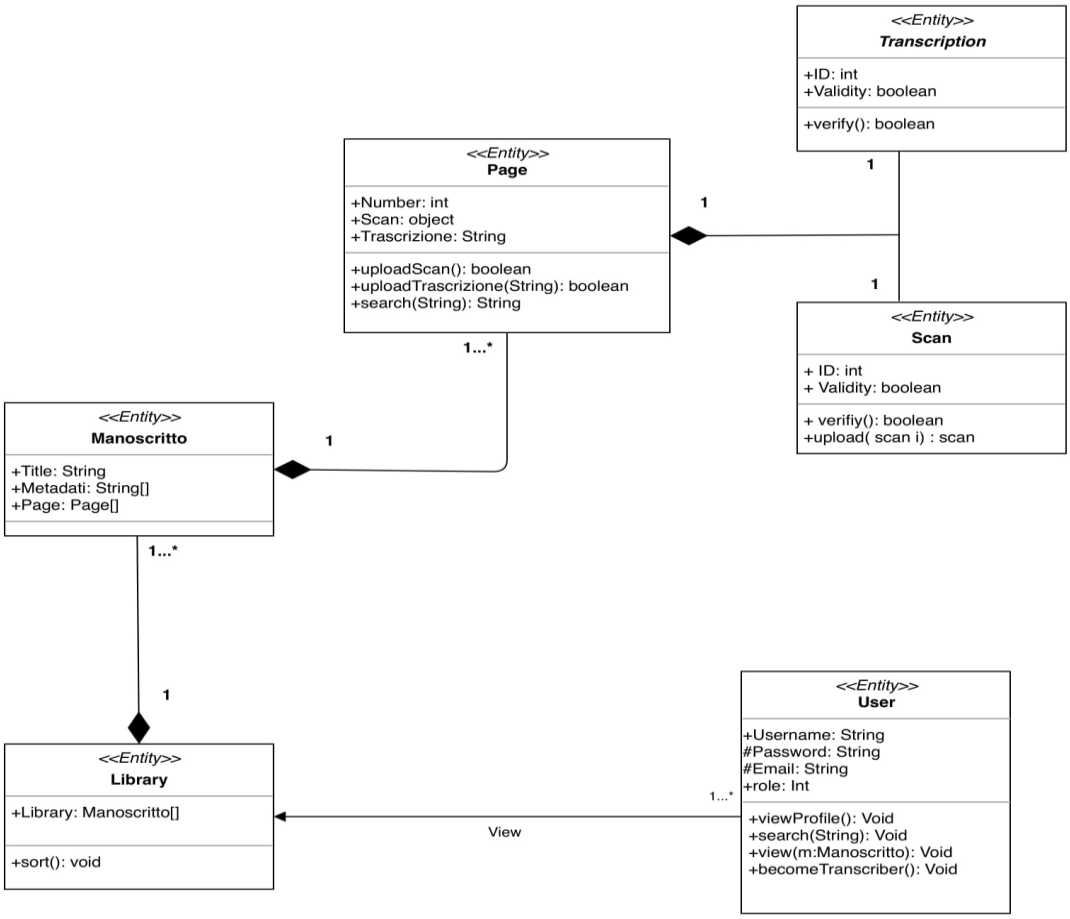
Il risultato del nostro lavoro di modellazione e Requirements Elicitation è il seguente:



Il file originale può essere trovato nella documentazione originale.

3.4.3 Object Model

Il modello a oggetti è un modello per trattare un problema attraverso la tipica rappresentazione Object-Oriented.



Quello in figura è il modello di Dominio con le classi entity richiesto dalla specifica 1.3 della specifica. L'analisi delle relative classi entity, boundary e controller può essere trovata nelle specifiche allegate.

3.4.4 Analisi finalizzata all'individuazione di classi Entity, Boundary e Controller

Per l'analisi finalizzata all'individuazione di Classi Entity, Boundary e Controller, ci siamo basati sulle euristiche di Abbott e quelle descritte nel libro "Object Oriented Software Engineering, Using UML, Patterns and Java" di Bernd Bruegge e Allen H. Dutoit.

Gli oggetti partecipanti sono le basi del modello di analisi e vengono trovati analizzando in dettaglio ogni use case. Il linguaggio naturale è comodo per farsi capire dai clienti ma molto impreciso. Questo modello di analisi serve ad avere un riferimento formale per gli sviluppatori. I Boundary sono il collegamento tra gli attori e i Controller. In generale, si potrebbe dire che gli Oggetti Boundary rappresentano l'interfaccia di sistema per gli attori. Non è necessario descriverli in dettaglio, ma abbiamo dato un'idea della loro struttura. Infine, gli oggetti Controller sono quelli responsabili di coordinare i boundary e le entity. Sono responsabili per collezionare le informazioni dagli oggetti boundary e fornirli agli oggetti entity. In generale, si può trovare un oggetto Controller per ogni use case.

Andiamo quindi ad analizzare ogni Use case definito nello schema 1.2 e trovare le classi entity, boundary e controller relativi.

Registrazione

Entity: Utente

Boundary: Form di Registrazione, Bottone Registrati

Control: RegistrationController

Login

Entity: Utente, Uploader, RevisoreUpload, Trascrittore, RevisoreTrascrizioni, CapoTrascrittore, Amministratore

Boundary: Form di Login, Bottone Login

Control: LoginController

Ricerca Tramite Metadati

Entity: Utente, Uploader, RevisoreUpload, Trascrittore, RevisoreTrascrizioni, CapoTrascrittore, Amministratore, Manoscritto

Boundary: Barra di Ricerca, Bottone Cerca

Control: SearchByAuthorController, SearchByNameController, ResultsfromSearchController

Visualizzazione Pagine

Entity: Utente, Uploader, RevisoreUpload, Trascrittore, RevisoreTrascrizioni, CapoTrascrittore, Amministratore, Manoscritto, Page

Boundary: Elenco manoscritti, Tasti avanti e indietro

Control: ListaOpereController

Download Opere

Entity: Utente

Boundary: Bottone Download

Control: ListaOpereController

Visualizzazione profilo

Entity: Utente, Uploader, RevisoreUpload, Trascrittore, RevisoreTrascrizioni, CapoTrascrittore, Amministratore

Boundary: Bottone Profilo

Control: ProfileController

Candidarsi come Trascrittori

Entity: Utente

Boundary: Bottone “Candidati come trascrittore”

Control: TranscriberCandidatureController

Upload Scansioni

Entity: Uploader

Boundary: Bottone Upload

Control: UploadController

Revisione Scansioni

Entity: RevisoreUpload, Page

Boundary: Lista pagine da controllare, Tasto Accetta, Tasto Rifiuta

Control: ReviseUploadController

Trascrizione pagine tramite Editor

Entity: Transcriber

Boundary: Lista opere da trascrivere per quell'utente, TEI, Bottone Submit

Control: TranscriberController

Revisione Trascrizioni

Entity: RevisoreTrascrizioni

Boundary: ListaTrascrizioni, tasto Accetta, tasto Rifiuta

Control: ReviseController

Assegnazione trascrizioni

Entity: Admin,CapoTrascrittore

Boundary: Lista Pages senza trascrizione, Lista Trascrittori

Control: AdminController

Gestione livelli trascrittori

Entity: Capotrascrittore, Admin

Boundary: Form utente, form Livello, tasto Cambia Livello

Control: AdminController

Gestione Back End

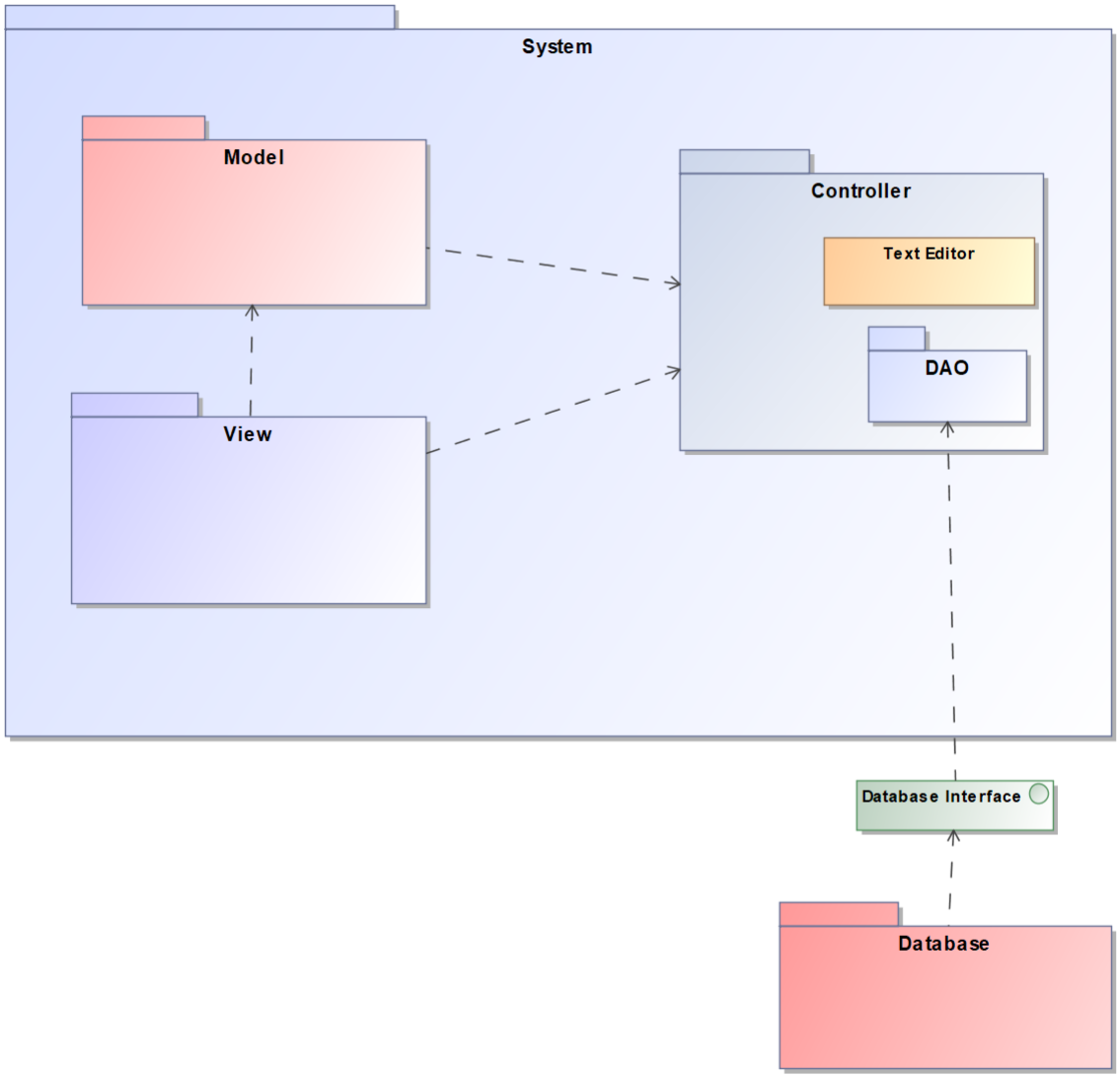
Entity: Admin

Boundary: Form funzioni amministratori

Control: Admin Controller

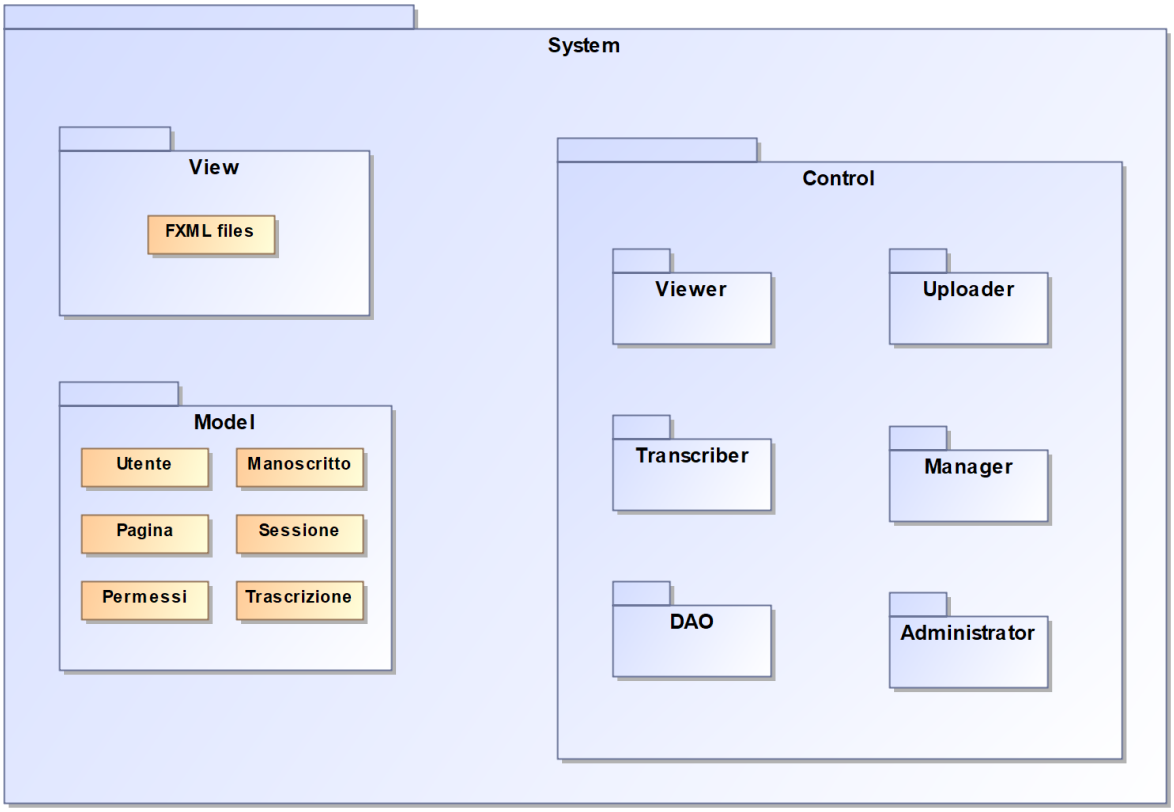
3.5 System Design

Il primo modello di System Design è il seguente, ad alto livello:



L'architettura scelta è di tipo MVC, con l'utilizzo di un DAO per la connessione al database MySQL (si veda anche il paragrafo sui Design Patterns).

Il seguente diagramma mostra una visione più dettagliata e concentrata sul nostro sistema:



Il sistema è stato progettato secondo i paradigmi della modellazione Object Oriented, con la priorità di rendere il sistema modulare per futuri utilizzi e per una migliore manutenzione: per questo abbiamo deciso di esplicitare la divisione del Control nei cinque sottosistemi logici.

3.6 Design Patterns

MVC

Per la progettazione del nostro sistema abbiamo utilizzato l'architettura Model-View-Controller. In questa architettura, L'utente comunica con la

View che invia updates al Controller, che a sua volta modifica il Model, da cui la View stessa dipende. Questo modello è triangolare e divide la logica di business dalle interfacce con il quale l'utente va a comunicare.

(Reference: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm)

Data Access Object

Abbiamo inoltre utilizzato un DAO per tutte le comunicazioni con il Database. Il DAO implementa diverse classi, in particolare una per ogni oggetto nel model, che si occupa di prendere e modificare i dati relativi a quella entità nel nostro database.

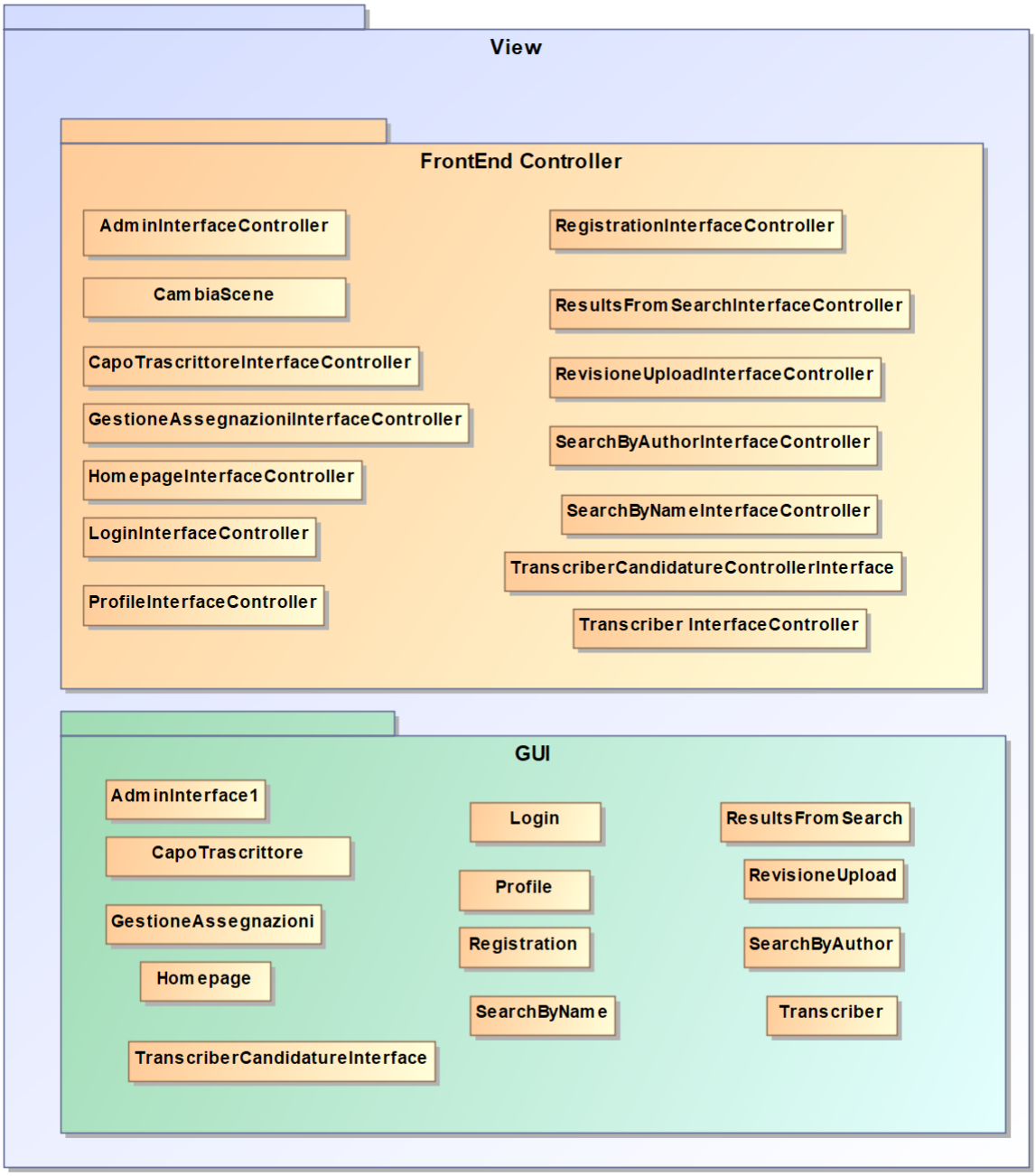
(Reference: https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm)

Front Controller

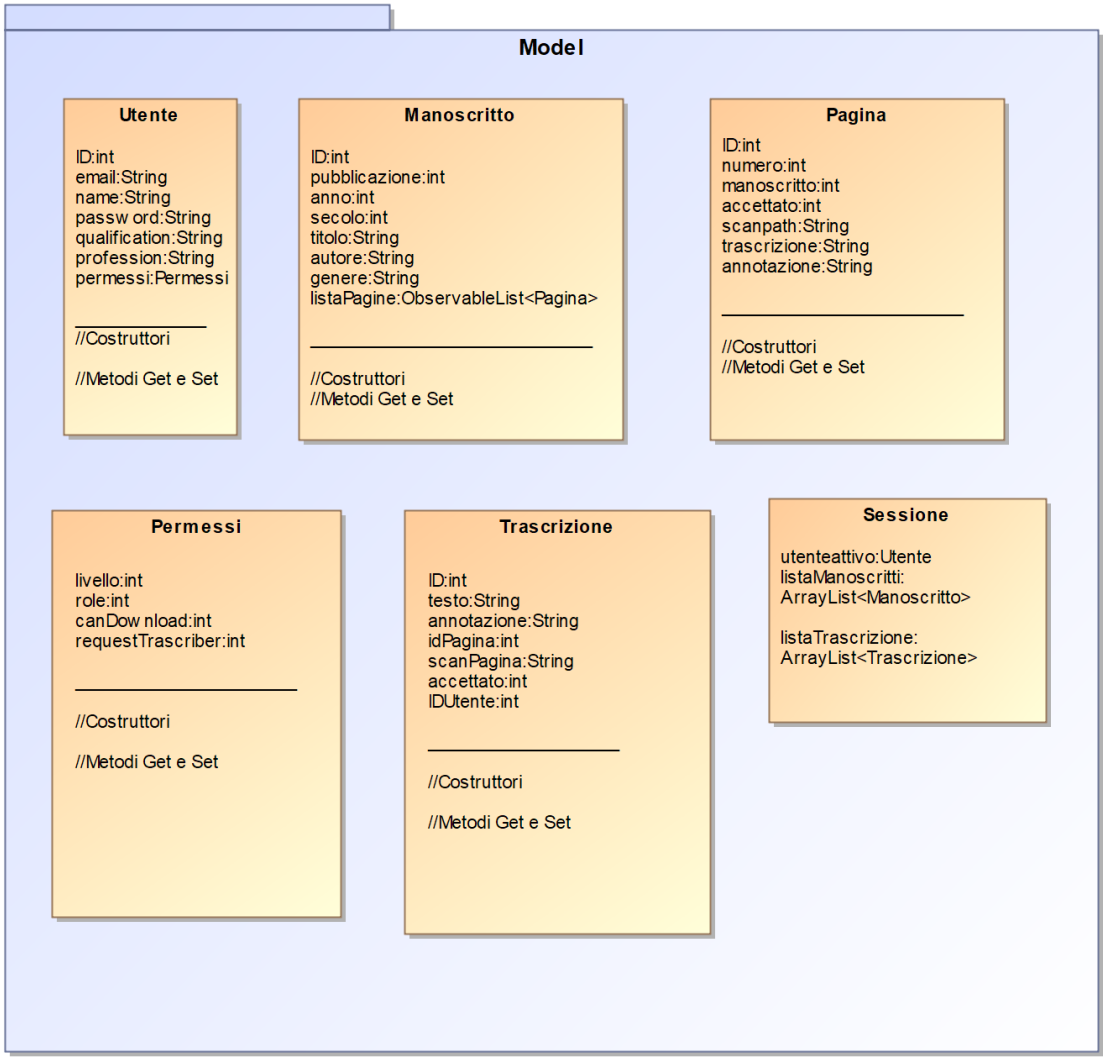
Il Front controller si occupa di gestire TUTTE le interazioni con l'utente e il suo package si trova nella view. Quando il Front Controller riceve un input (ActionEvent), quello che fa è modificare eventualmente la scena e chiamare il rispettivo metodo nel Back-End Controller.

(Reference: https://www.tutorialspoint.com/design_pattern/front_controller_pattern.htm)

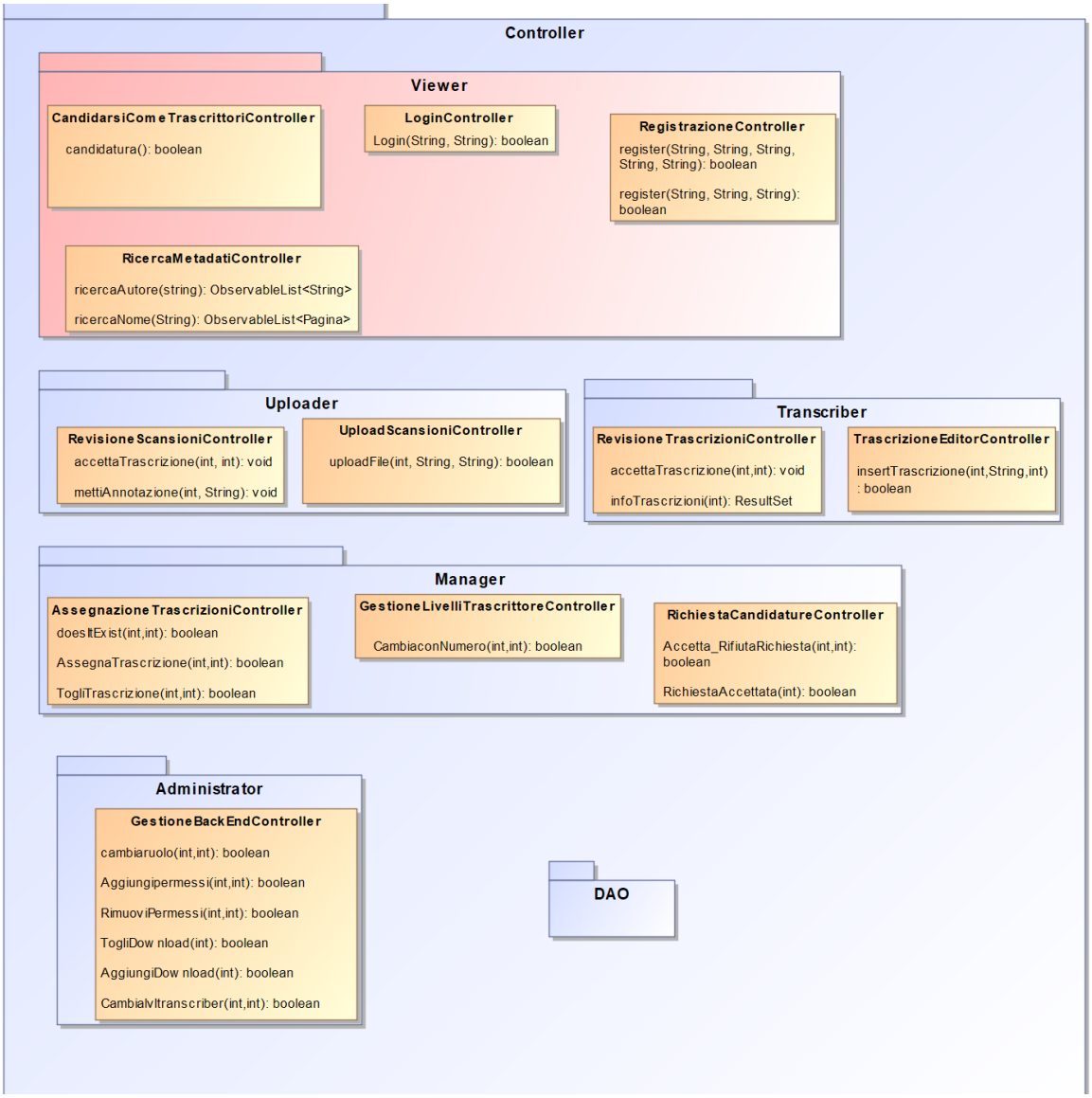
3.7 Class Diagram



Questo è il class diagram del nostro package View. Nella GUI ci sono tutti i file FXML relativi a ogni interfaccia utente del sistema, mentre nel package FrontEnd Controller c'è una classe per ogni interfaccia, che si occupa della gestione degli ActionEvent generati dall'utente in quella particolare interfaccia.

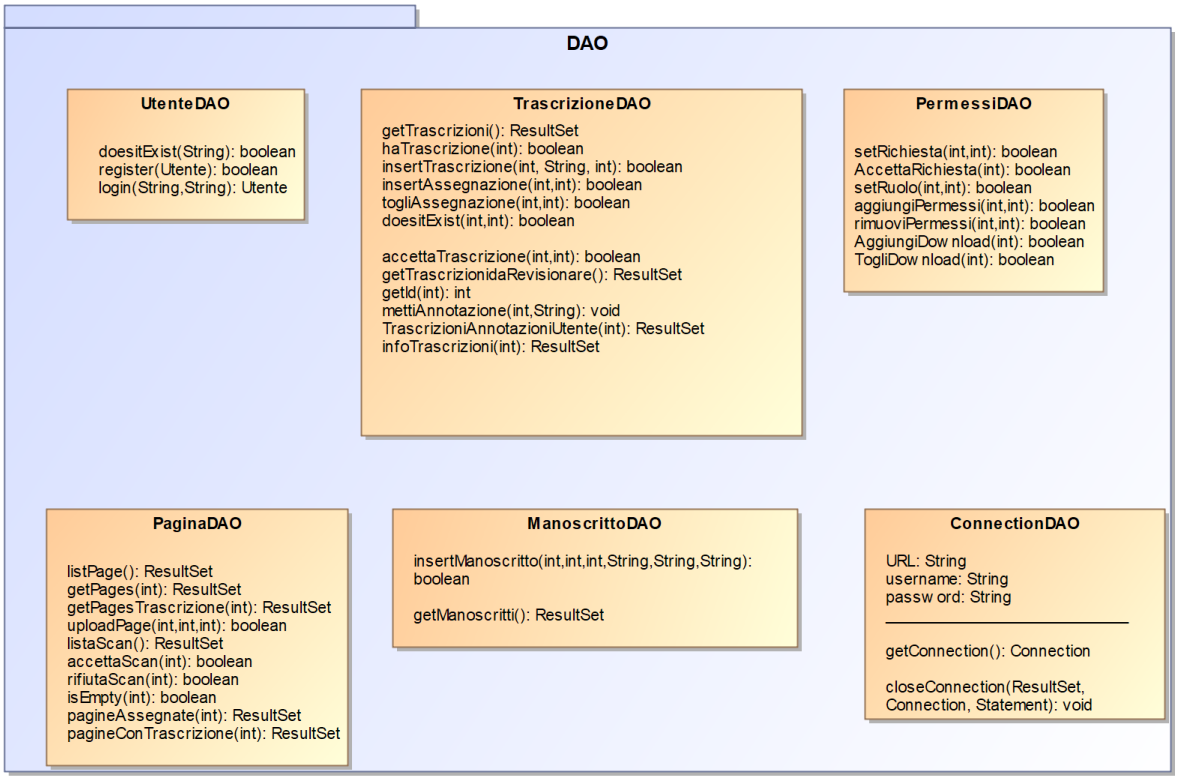


Per la modellazione del package Model ci siamo basati sul Modello di Dominio descritto in precedenza. Fondamentalmente le entità sono le stesse.



Per la parte Control del nostro sistema, ci siamo basati sulla suddivisione in sottosistemi presenti sia nella specifica che nel diagramma Use-Case. E' importante notare come i sottosistemi siano logici e non legati ai ruoli degli utenti (che sono di più). Per le classi interne, abbiamo cercato di collegare una classe ad ogni use case seguendo l'analisi delle classi Entity, Boundary e Controller descritta prima.

L'architettura e le classi presenti nel Controller.DAO vengono mostrate qui sotto:



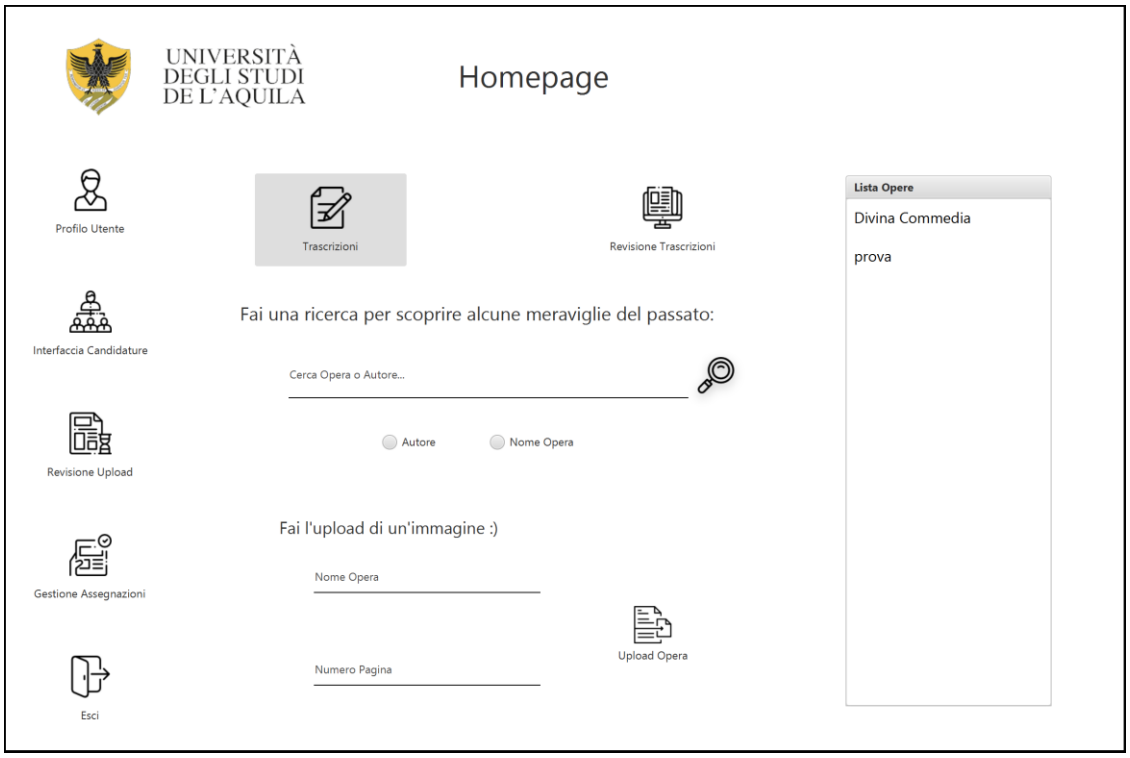
C'è sostanzialmente una classe DAO per ogni classe nel Model, che si occupa di gestirle nel Database. In questo modo ogni accesso al Database, e quindi ogni stringa di codice SQL, si trova in questo package. Questa soluzione ci ha dato modo di identificare più rapidamente eventuali errori dovuti a errato codice SQL.

Il System Design e i Class Diagram sono stati realizzati grazie al tool MagicDraw.

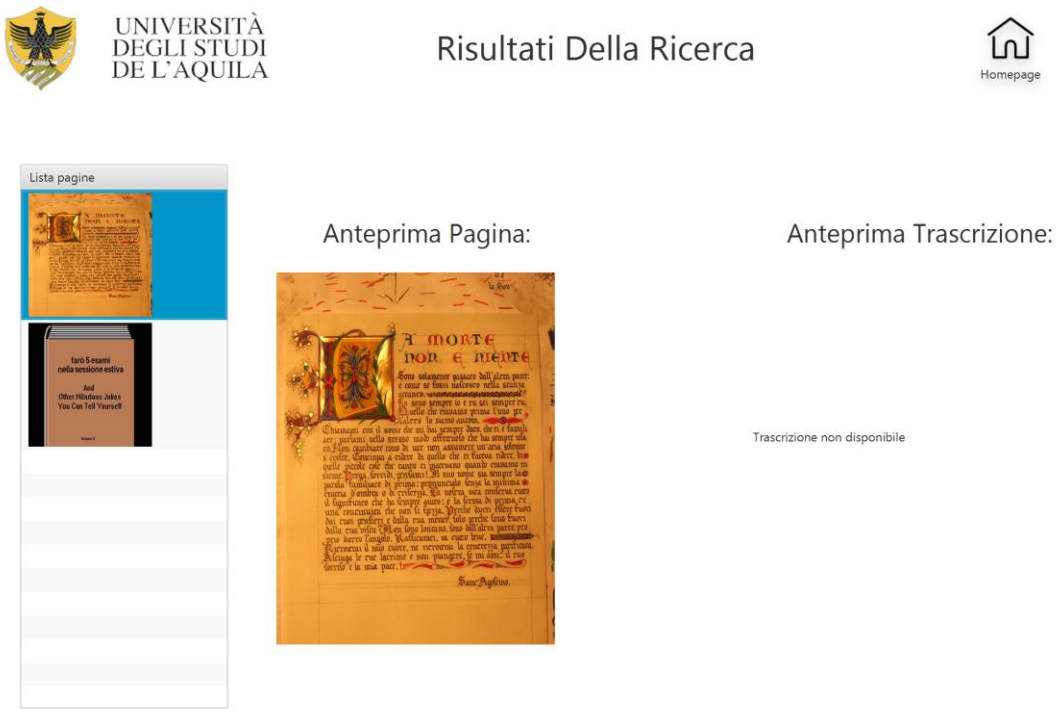
3.8 Interfacce

Infine, ecco una collezione delle interfacce implementate grazie a JavaFX e CSS.


UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA
Registrati
Pagina d'accesso
Username
Password
Accedi
Esci


UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA
Homepage
Profilo Utente
Trascrizioni
Revisione Trascrizioni
Interfaccia Candidature
Fai una ricerca per scoprire alcune meraviglie del passato:
Cerca Opera o Autore...
Autore
Nome Opera
Revisione Upload
Fai l'upload di un'immagine :)
Gestione Assegnazioni
Nome Opera
Upload Opera
Numero Pagina
Esci
Lista Opere
Divina Commedia
prova

Queste sono le schermate di accesso e di homepage. Nella homepage sono disponibili tutti i pulsanti, quindi l'accesso è stato eseguito come Admin. In generale, l'applicazione rimane sugli stessi colori e con lo stesso stile minimal in bianco e nero. Come da requisiti non funzionali, abbiamo cercato di rendere l'interfaccia il più semplice e immediata possibile, grazie ai CSS e all'utilizzo di icone esplicative.



Infine, quella di sopra è l'interfaccia per la visualizzazione dei manoscritti. A lato c'è una scrollbar dove si può selezionare la pagina, che apparirà a lato sotto l'anteprima, con a fianco la trascrizione, come richiesto dalle specifiche.