EAGLELIBRARY

Progetto OOSD
Università deli Studi dell'Aquila

Sviluppo di un'applicazione per la gestione di una libreria digitale

D'Agostino Giovanni
Sodano Sabato

1. REQUISITI

1.1 REQUISITI FUNZIONALI

1. Il sistema deve permettere la login e la registrazione di un nuovo utente.

2. Il sistema deve consentire la ricerca opere digitali mediante una serie di parametri. I parametri sono i seguenti :
   - Titolo;
   - Anno di pubblicazione;
   - Autore;
   - Genere;
   - Parte del testo di una trascrizione.

3. Il sistema deve fornire una classificazione delle opere attraverso il loro genere (Storia, Horror, …).

4. Quando viene completata la ricerca di un'opera, il sistema deve consentire la visualizzazione dell'opera attraverso un widget di paginazione per facilitarne la consultazione.

5. Il sistema deve consentire il download dell'opera se l'utente è munito di permesso speciale.

6. Se l'utente non è munito di opportuni permessi, l'icona che consente di fare download non apparirà sulla GUI.

7. Il sistema deve consentire ad ogni utente che fa uso dell'applicazione di poter diventare trascrittore per mezzo di un modulo opportunamente compilato.

8. Il modulo deve essere controllato da un manager al fine di essere validato o rigettato, qualora quest'ultimo dovesse essere rigettato, l'utente potrà riprovare l'operazione l'anno successivo.

9. Se l'utente prova ad inviare un modulo che era stato precedentemente rigettato senza che un anno sia già passato il sistema emetterà un messaggio di errore per informare l'utente dell'errata procedura.

10. Per ogni utente di EagleLibrary, il sistema inizializzerà un profilo inserendo le informazioni anagrafiche di base, da quel momento esso sarà modificabile in ogni momento.

11. Il sistema deve consentire l'upload delle opere, l'opera sarà caratterizzata da una testata formata dalle informazioni di base quali :
    - Autore;
    - Titolo;
    - Anno di pubblicazione;
    - Genere;
    -

    e da un insieme di pagine, in formato immagine, che saranno caricate una per volta.
    Anche la procedura di caricamento deve essere agevolata da un paginatore per avere un formato più compatto delle immagini che si stanno caricando.

12. Per ogni opera presente nel sistema bisognerà effettuarne la trasposizione in digitale, ad occuparsi di questo compito potranno essere soltanto i trascrittori, diventati tali per mezzo del modulo di cui sopra.

13. Il sistema deve consentire di effettuare assegnazioni utente ←→ pagina affinché l'utente possa lavorare alla trascrizione in digitale della pagina/pagine ad esso associata/e.

14. Quando viene completata la fase di trascrizione di una pagina, quest'ultima dovrà essere demandata a dei controlli quali quelli di sintassi e semantica. In virtù di questo passaggio la trascrizione potrà essere accettata senza problemi oppure potrà essere riassegnata all'utente perché non conforme alla pagina originale dell'opera. Seguiranno quindi correzioni e l'iter descritto in precedenza andrà avanti fintantoché non si arriva alla perfetta uguaglianza tra pagina e trascrizione.

15. Quando la trascrizione di una determinata pagina supera tutti i controlli il manager abilita la pubblicazione di quest'ultima. In particolare, quando l'opera verrà visualizzata in corrispondenza della pagina trascritta sarà possibile visualizzare anche la versione digitale.

16. Il sistema consente ai manager di attribuire delle valutazioni in scala ai trascrittori che ne valorizza l'abilità di trascrizione. In particolare, su una scala da 1-5 sarà possibile, ogni utente sarà caratterizzato da un valore appartenente a quel range.

17. In fase di upload di un'opera da parte dell'utente vengono effettuati una serie di controlli al fine di mandare a buon fine l'upload stesso. I controlli saranno automatici e riguarderanno la mandatorietà di :
    - Autore;
    - Titolo;
    - Anno di pubblicazione;
    - Genere.

1.2 REQUISITI NON FUNZIONALI

1. L'implementazione della parte backend dell'applicazione dovrà essere scritta utilizzando il linguaggio Java, in particolare Java 1.8.

2. Al push di qualsiasi pulsante sulla UI, il sistema deve rispondere in tempi sempre < 2sec.

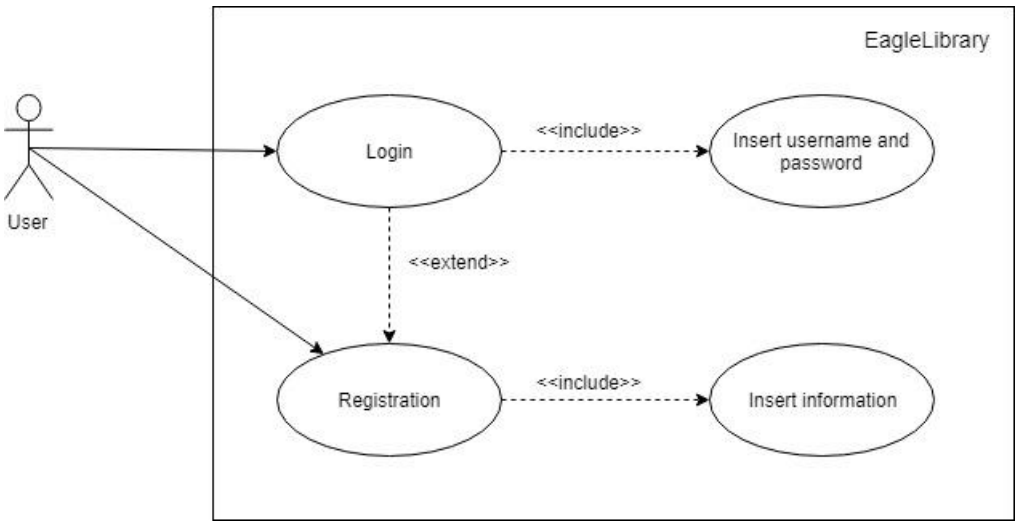
3. Il database utilizzato per immagazzinare i dati sarà un database di tipo relazionale e per interagire con quest'ultimo utilizzeremo MySQL.

4. Per il meccanismo di sincronizzazione delle modifiche degli utenti su di un'unica trascrizione si vuole utilizzare un pattern a semaforo, ossia gestire con lock pessimistico le modifiche sulla trascrizione in modo tale da non mandare in overlap le varie versioni.

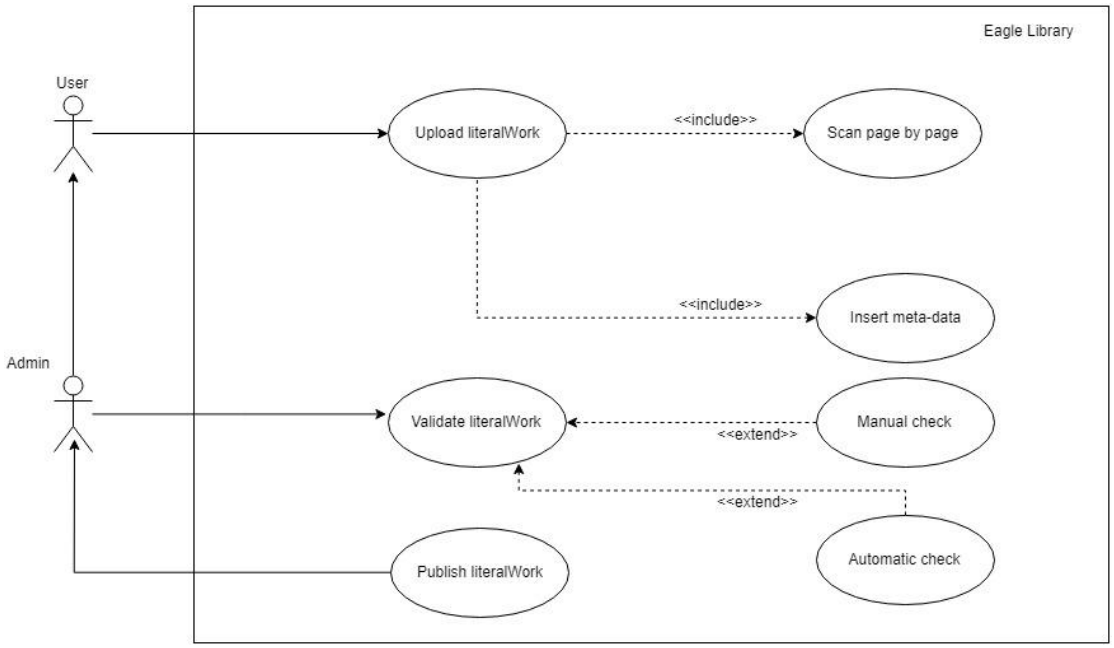
5. Le immagini di cui viene fatto l'upload possono avere formato in (JPEG,PNG,PDF).

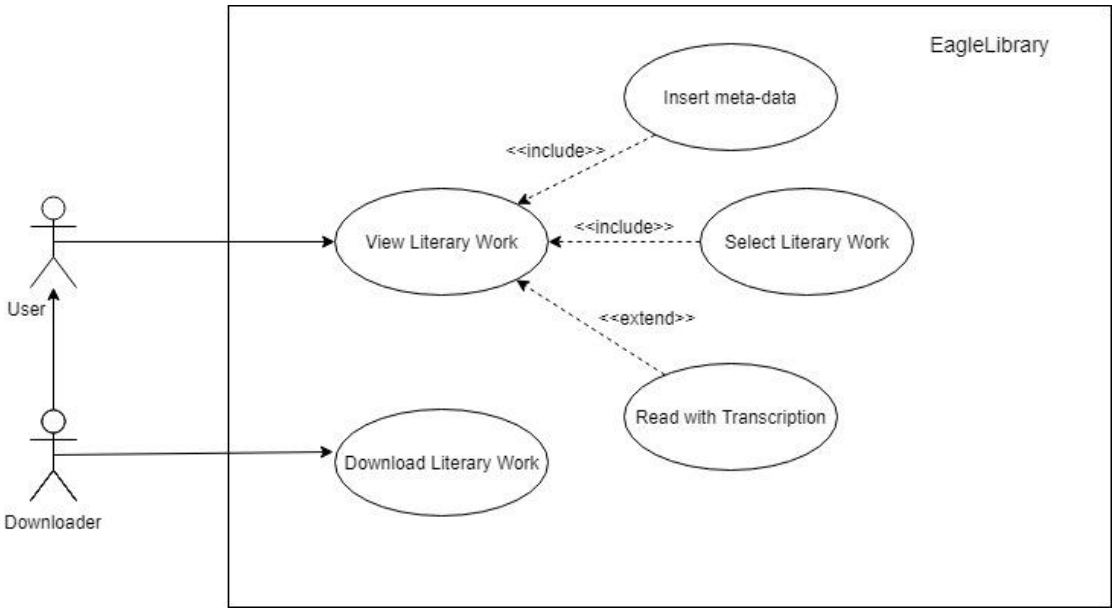
1.3 USE CASES

Il primo caso d'uso che andiamo ad analizzare è la login e la registrazione di un utente nel nostro sistema, che consiste nella prima funzionalità di cui un utente ha bisogno.

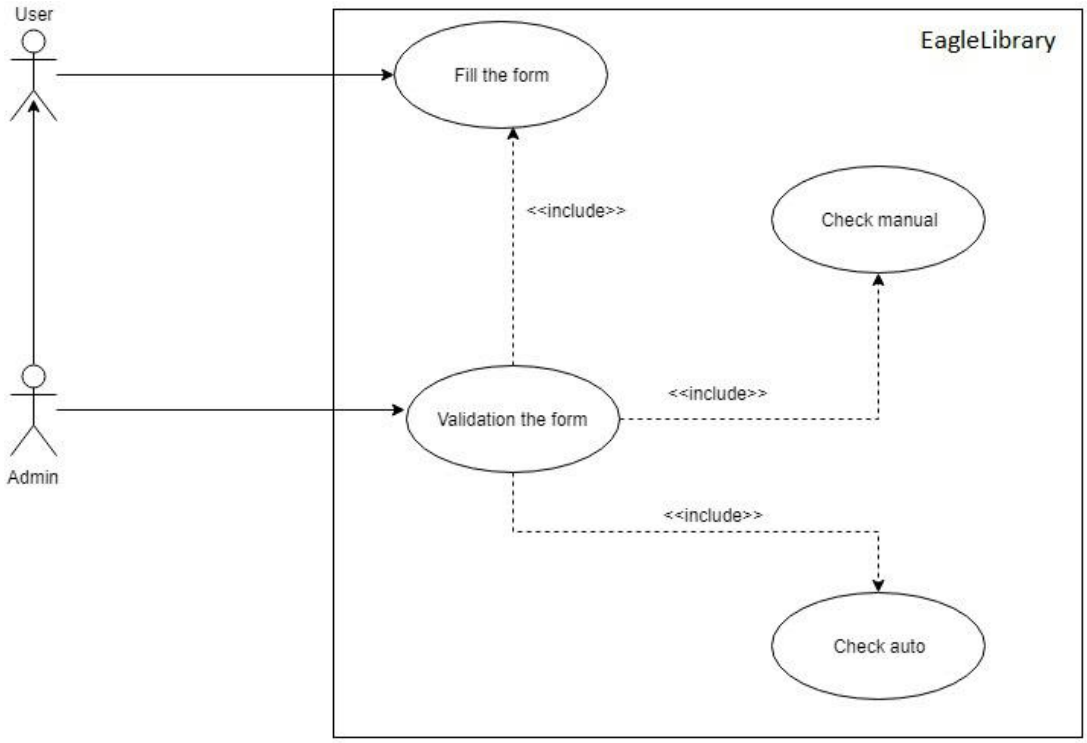


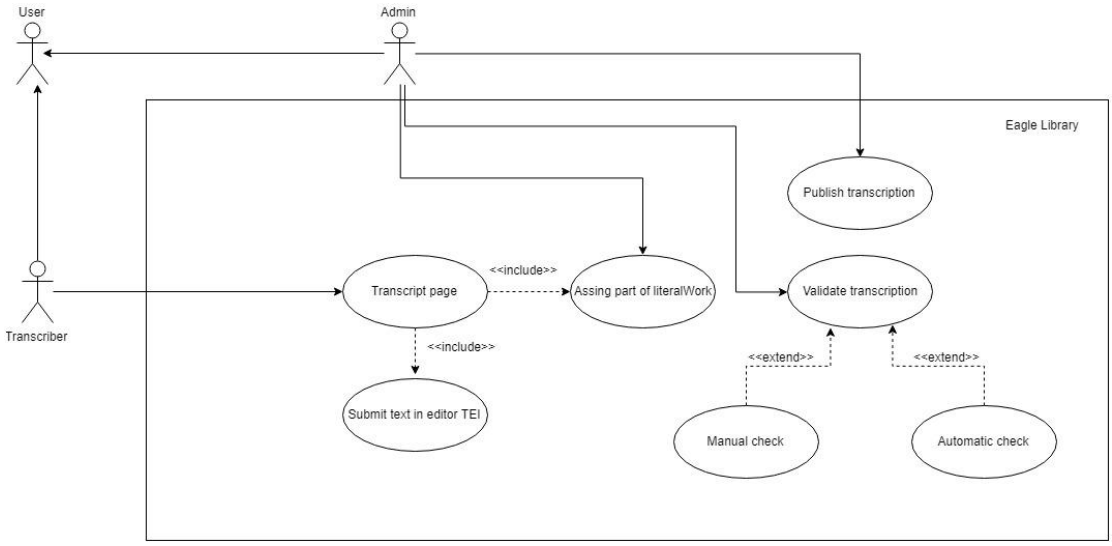
Successivamente l'utente potrà cercare, consultare, scaricare e caricare nel sistema le opere.





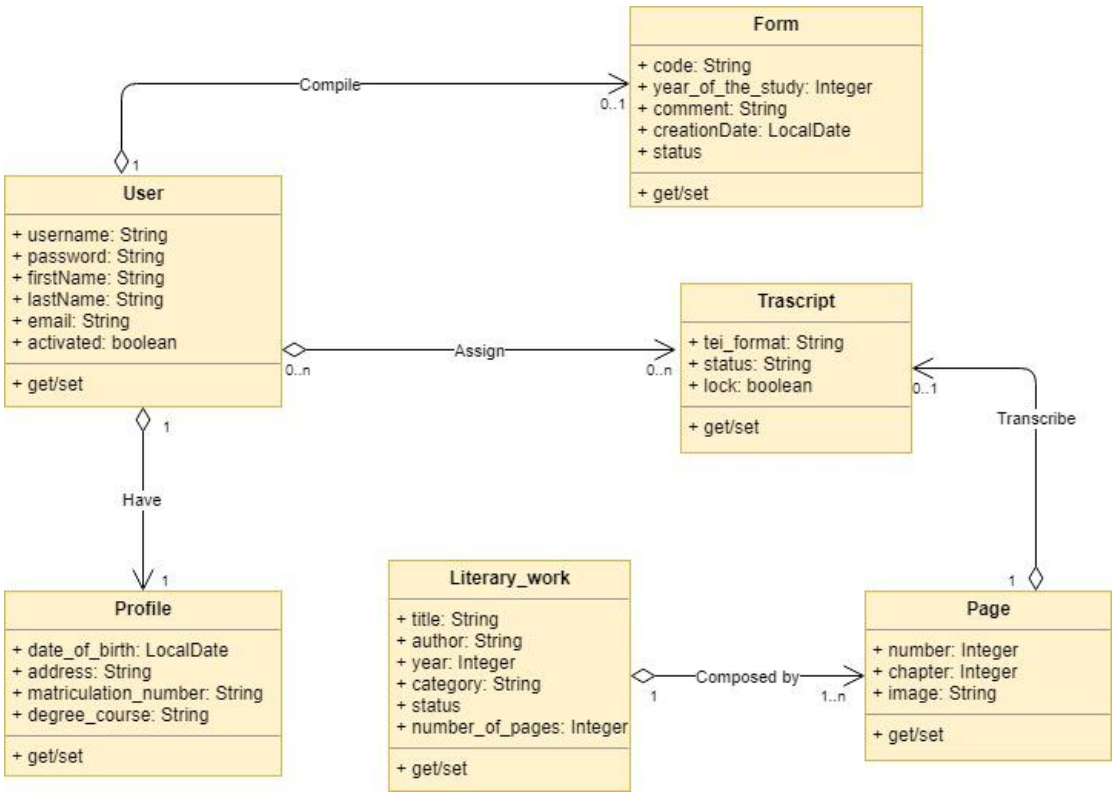
Un'altra funzionalità principale di EagleLibrary è la possibilità di gestire un sistema di trascrizione delle opere in formato digitale. Di seguito i casi d'uso per come diventare trascrittore e come eseguire una trascrittura.





1.4 MODELLI DI DOMINIO

Le entità e le relazioni che abbiamo estrapolato dai requisiti e i casi d'uso sono le seguenti:

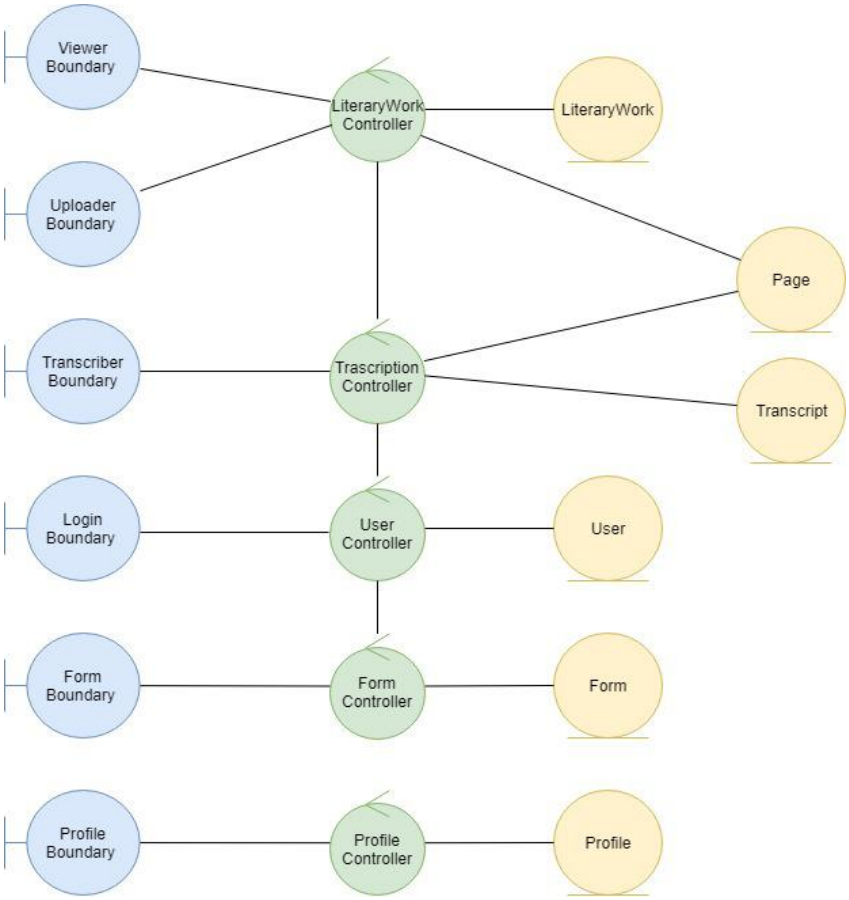


Durante la modellazione delle entità abbiamo dovuto fare delle assunzioni, per permetteci di fornire le giuste cardinalità e relazioni:

1. Supponiamo che un utente può inviare un singolo modulo che, in base alla revisione del manager, può rinviare oppure dovrà aspettare un tot di tempo (es. 1 anno) prima di poter rinviare ancora il modulo; caso eccezionale per alcuni utenti che non potranno più rinviare il modulo.
2. Una pagina può avere una sola trascrizione.
3. Assumiamo che la tabella di user_trans_link possa essere intesa come il concetto di Assegnazione.
4. Il modulo è un form online da compilare.
5. Supponiamo che l'immagine di una pagina è semplicemente un attributo della tabella Pagina.
6. Supponiamo che solo l'administrator possa effettuare upload delle varie opere, e che quindi non abbia bisogno di ulteriore validazione.
7. Assumiamo che il modulo da parte di un utente può essere accettato o rifiutato solo dal manager.
8. Assumiamo che quando un utente sta modificando una trascrizione viene applicato un lock su quest'ultima. Il lock viene applicato solo quando l'utente ci sta effettivamente lavorando.

1.5 ENTITY, BOUNDARY E CONTROLLER

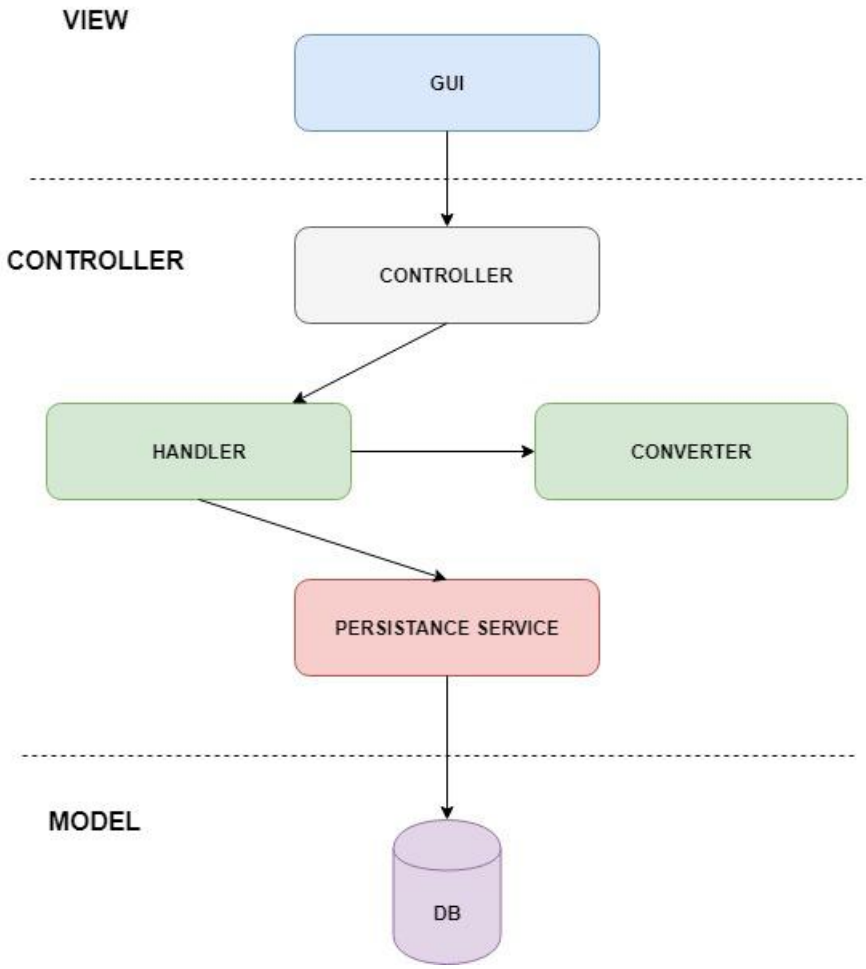
Dopo aver specificato le entità, abbiamo individuato boundary e controller del sistema.



2. SYSTEM DESIGN

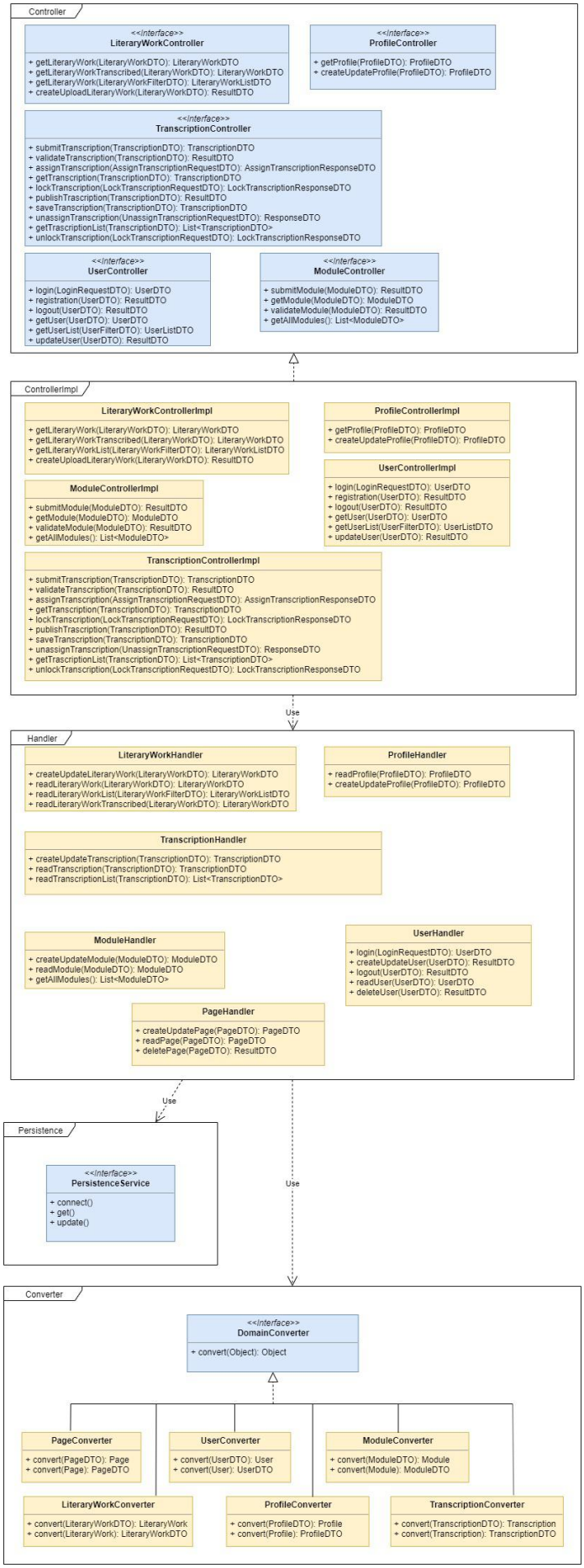
2.1 MODELLO ARCHITETTURA SOFTWARE

Il pattern seguito per definire la struttura del software è MVC. Abbiamo deciso di creare un'interfaccia di servizi (in realtà sono più di una, poi vedremo nel dettaglio), disponibili alla GUI, presente nel package CONTROLLER. La logica del sistema è lasciata all'HANDLER. Per il trasferimento di informazioni tra i diversi livelli del sistema abbiamo deciso di usare la composizione. IDTO sono i nostri Transfer Object che dovranno essere convertiti in modelli (CONVERTER) prima di essere persistiti nel DB.

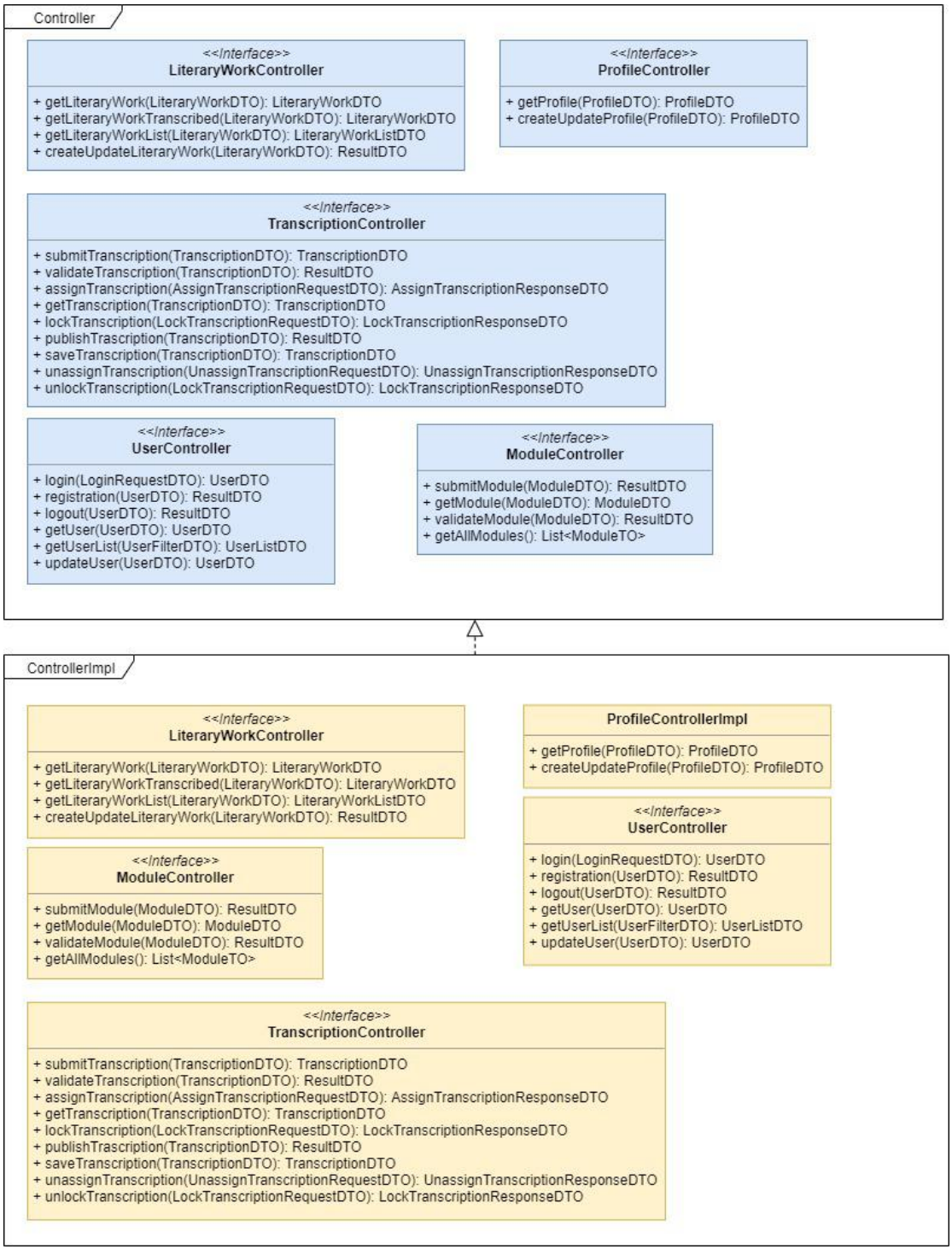


2.2 DESCRIZIONE DELL'ARCHITETTURA

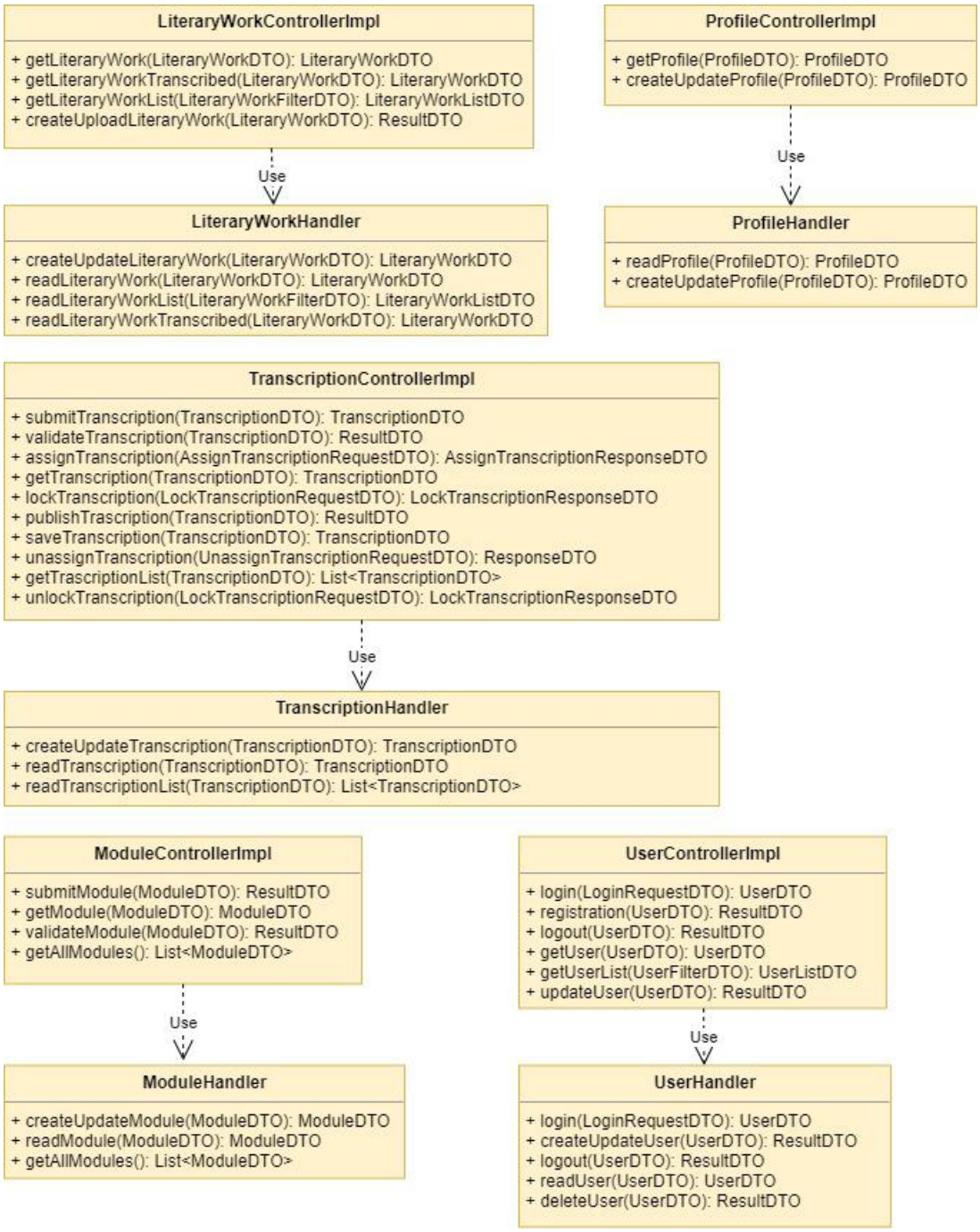
Nella pagine successive vi saranno diversi design con diverso grado di dettaglio per descrivere al meglio l'architettura adottata.


Controller
<<Interface>>
LiteraryWorkController
+ getLiteraryWork(LiteraryWorkDTO): LiteraryWorkDTO
+ getLiteraryWorkTranscribed(LiteraryWorkDTO): LiteraryWorkDTO
+ getLiteraryWork(LiteraryWorkFilterDTO): LiteraryWorkListDTO
+ createUploadLiteraryWork(LiteraryWorkDTO): ResultDTO
<<Interface>>
ProfileController
+ getProfile(ProfileDTO): ProfileDTO
+ createUpdateProfile(ProfileDTO): ProfileDTO
<<Interface>>
TranscriptionController
+ submitTranscription(TranscriptionDTO): TranscriptionDTO
+ validateTranscription(TranscriptionDTO): ResultDTO
+ assignTranscription(AssignTranscriptionRequestDTO): AssignTranscriptionResponseDTO
+ getTranscription(TranscriptionDTO): TranscriptionDTO
+ lockTranscription(LockTranscriptionRequestDTO): LockTranscriptionResponseDTO
+ publishTrascription(TranscriptionDTO): ResultDTO
+ saveTranscription(TranscriptionDTO): TranscriptionDTO
+ unassignTranscription(UnassignTranscriptionRequestDTO): ResponseDTO
+ getTrascriptionList(TranscriptionDTO): List<TranscriptionDTO>
+ unlockTranscription(LockTranscriptionRequestDTO): LockTranscriptionResponseDTO
<<Interface>>
UserController
+ login(LoginRequestDTO): UserDTO
+ registration(UserDTO): ResultDTO
+ logout(UserDTO): ResultDTO
+ getUser(UserDTO): UserDTO
+ getUserList(UserFilterDTO): UserListDTO
+ updateUser(UserDTO): ResultDTO
<<Interface>>
ModuleController
+ submitModule(ModuleDTO): ResultDTO
+ getModule(ModuleDTO): ModuleDTO
+ validateModule(ModuleDTO): ResultDTO
+ getAllModules(): List<ModuleDTO>
ControllerImpl
LiteraryWorkControllerImpl
+ getLiteraryWork(LiteraryWorkDTO): LiteraryWorkDTO
+ getLiteraryWorkTranscribed(LiteraryWorkDTO): LiteraryWorkDTO
+ getLiteraryWorkList(LiteraryWorkFilterDTO): LiteraryWorkListDTO
+ createUploadLiteraryWork(LiteraryWorkDTO): ResultDTO
ProfileControllerImpl
+ getProfile(ProfileDTO): ProfileDTO
+ createUpdateProfile(ProfileDTO): ProfileDTO
ModuleControllerImpl
+ submitModule(ModuleDTO): ResultDTO
+ getModule(ModuleDTO): ModuleDTO
+ validateModule(ModuleDTO): ResultDTO
+ getAllModules(): List<ModuleDTO>
UserControllerImpl
+ login(LoginRequestDTO): UserDTO
+ registration(UserDTO): ResultDTO
+ logout(UserDTO): ResultDTO
+ getUser(UserDTO): UserDTO
+ getUserList(UserFilterDTO): UserListDTO
+ updateUser(UserDTO): ResultDTO
TranscriptionControllerImpl
+ submitTranscription(TranscriptionDTO): TranscriptionDTO
+ validateTranscription(TranscriptionDTO): ResultDTO
+ assignTranscription(AssignTranscriptionRequestDTO): AssignTranscriptionResponseDTO
+ getTranscription(TranscriptionDTO): TranscriptionDTO
+ lockTranscription(LockTranscriptionRequestDTO): LockTranscriptionResponseDTO
+ publishTrascription(TranscriptionDTO): ResultDTO
+ saveTranscription(TranscriptionDTO): TranscriptionDTO
+ unassignTranscription(UnassignTranscriptionRequestDTO): ResponseDTO
+ getTrascriptionList(TranscriptionDTO): List<TranscriptionDTO>
+ unlockTranscription(LockTranscriptionRequestDTO): LockTranscriptionResponseDTO
Use
Handler
LiteraryWorkHandler
+ createUpdateLiteraryWork(LiteraryWorkDTO): LiteraryWorkDTO
+ readLiteraryWork(LiteraryWorkDTO): LiteraryWorkDTO
+ readLiteraryWorkList(LiteraryWorkFilterDTO): LiteraryWorkListDTO
+ readLiteraryWorkTranscribed(LiteraryWorkDTO): LiteraryWorkDTO
ProfileHandler
+ readProfile(ProfileDTO): ProfileDTO
+ createUpdateProfile(ProfileDTO): ProfileDTO
TranscriptionHandler
+ createUpdateTranscription(TranscriptionDTO): TranscriptionDTO
+ readTranscription(TranscriptionDTO): TranscriptionDTO
+ readTranscriptionList(TranscriptionDTO): List<TranscriptionDTO>
ModuleHandler
+ createUpdateModule(ModuleDTO): ModuleDTO
+ readModule(ModuleDTO): ModuleDTO
+ getAllModules(): List<ModuleDTO>
UserHandler
+ login(LoginRequestDTO): UserDTO
+ createUpdateUser(UserDTO): ResultDTO
+ logout(UserDTO): ResultDTO
+ readUser(UserDTO): UserDTO
+ deleteUser(UserDTO): ResultDTO
PageHandler
+ createUpdatePage(PageDTO): PageDTO
+ readPage(PageDTO): PageDTO
+ deletePage(PageDTO): ResultDTO
Use
Use
Persistence
<<Interface>>
PersistenceService
+ connect()
+ get()
+ update()
Converter
<<Interface>>
DomainConverter
+ convert(Object): Object
PageConverter
+ convert(PageDTO): Page
+ convert(Page): PageDTO
UserConverter
+ convert(UserDTO): User
+ convert(User): UserDTO
ModuleConverter
+ convert(ModuleDTO): Module
+ convert(Module): ModuleDTO
LiteraryWorkConverter
+ convert(LiteraryWorkDTO): LiteraryWork
+ convert(LiteraryWork): LiteraryWorkDTO
ProfileConverter
+ convert(ProfileDTO): Profile
+ convert(Profile): ProfileDTO
TranscriptionConverter
+ convert(TranscriptionDTO): Transcription
+ convert(Transcription): TranscriptionDTO

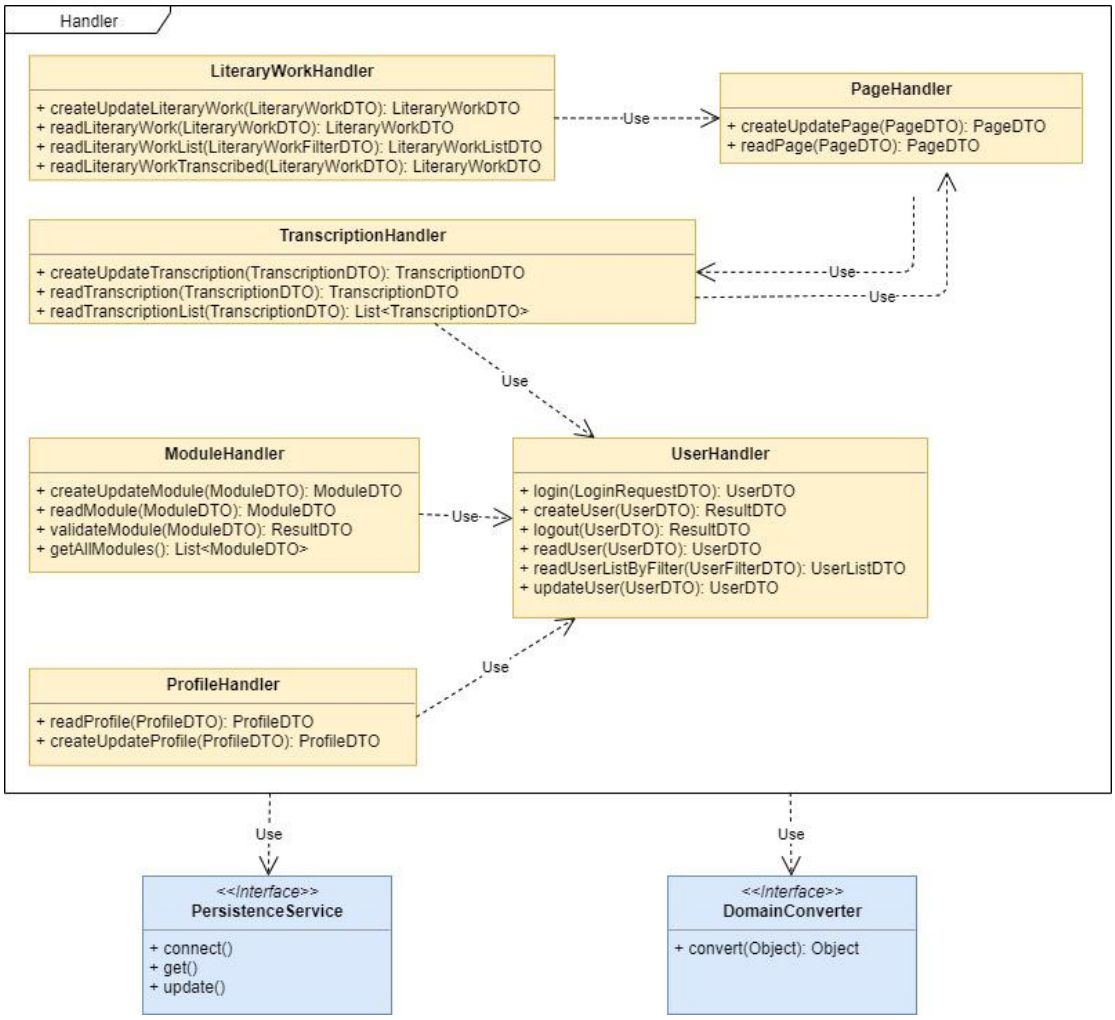
Dopo la prima overview andiamo ad analizzare ogni elemento della struttura e come sono collegati tra loro. Iniziamo dal Controller e ControllerImpl.



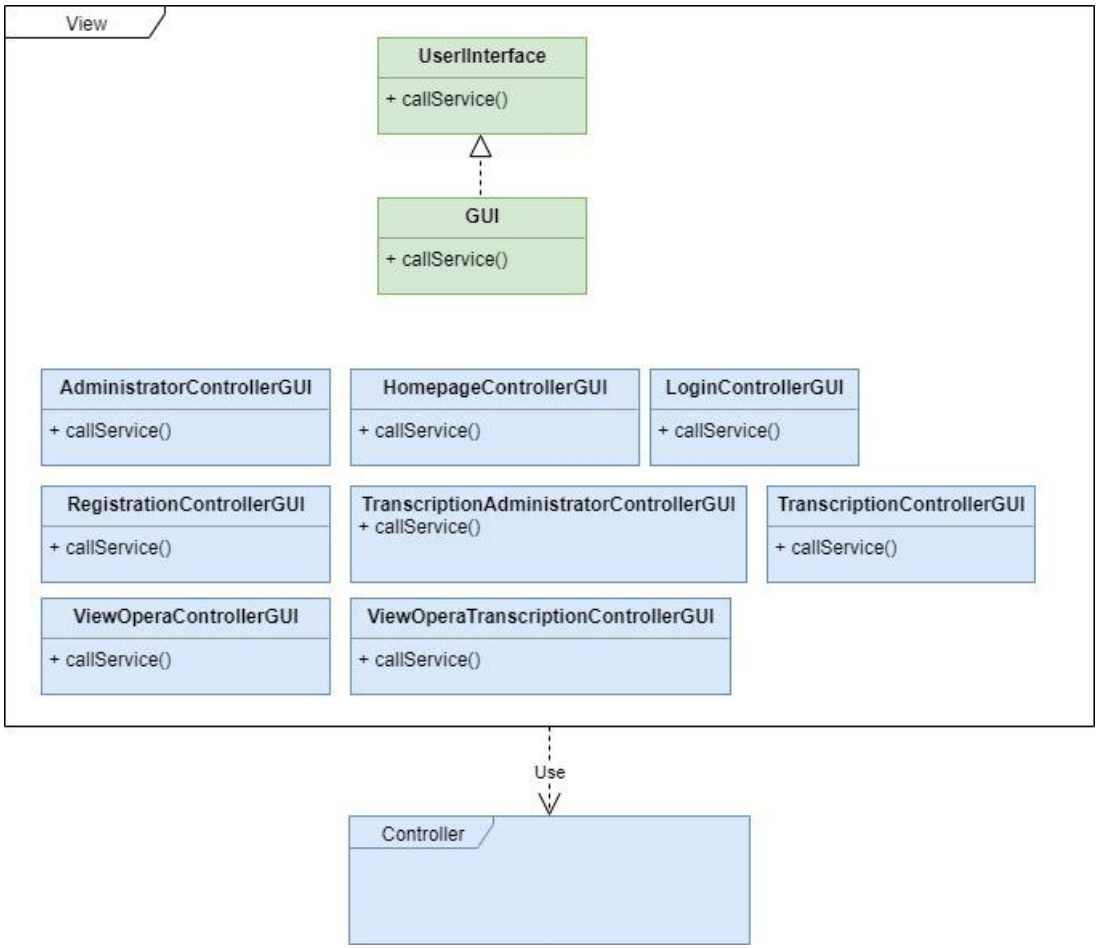
Di seguito il diagramma che descrive come il Controller comunica con l'handler.



Infine descriviamo come comunicano fra loro le classi dell'handler e come, prima di persistere (PersistanceService), dovranno chiamare i servizi dell'interfaccia DomainConverter.



Come abbiamo già detto la View potrà interagire solo con il Controller:



2.3 DESCRIZIONE SCELTE E STRATEGIE

Utilizzando questo tipo di struttura il nostro obiettivo è quello di delimitare in modo ottimale l'interfaccia utente e la logica di business. Inoltre abbiamo un altro grado di riusabilità del codice grazie all'uso della composizione, e quindi dei DTO, e all'uso delle varie interfacce e classi, che tramite pochi interventi è possibile modificare/aggiungere diverse funzionalità ed effettuare molte migliorie.

3. SOFTWARE/OBJECT DESIGN

Come detto in precedenza abbiamo favorito la composizione ripetto all'ereditarietà per la comunicazione tra i diversi "livelli" della nostra applicazione, questo grazie all'uso dei DTO (Data Transfer Object) che sono strutturati e collegati nel seguente modo: