

PROGETTO DIGITAL LIBRARY
CORSO DI LAUREA IN INFORMATICA
UNIVERSITÀ DEGLI STUDI DELL'AQUILA
A.A. 2015/2016

| STUDENTE | | MATRICOLA | |
|-------------------|--|---------------|--|
| Gabriele Di Rocco | | 195037 | |
| Valerio Piccioni | | 223000 | |

SPECIFICA

Il progetto si propone di realizzare una biblioteca digitale di testi e studi che contribuiscono alla formazione della cultura all'interno dell'Università degli Studi dell'Aquila.

Una biblioteca digitale è uno spazio in cui mettere insieme collezioni, servizi e persone a supporto dell'intero ciclo di vita di creazione, uso, preservazione di dati, informazione e conoscenza. Lo scopo di questo progetto è la digitalizzazione di manoscritti, che costituiscono un patrimonio bibliografico antico per un totale di 60.000 carte (ms. sec. XV-XIX) contenenti memorie storiche della città dell'Aquila.

Il processo di digitalizzazione dei manoscritti si suddivide in diverse fasi:

- *Digitalizzazione*

Il manoscritto è acquisito dal sistema sotto forma di immagini digitali ad alta risoluzione attraverso scanner planetari. Ogni manoscritto è formato da più acquisizioni (ogni immagine rappresenta una singola pagina). La digitalizzazione viene controllata da supervisori all'acquisizione per assicurarne la correttezza (ad esempio, in accordo con standard richiesti) e la qualità. L'immagine acquisita viene memorizzata all'interno del sistema ed assegnata all'opera di riferimento, corredandola di opportuni metadati.

- *Trascrizione*

Il manoscritto così acquisito deve essere trasformato in un testo digitale; ciò avviene attraverso operazioni di trascrizioni in formato TEI (Text Encoding Initiative). Le trascrizioni sono digitate manualmente (la natura del testo rende inutilizzabili strumenti di acquisizione automatica) attraverso un text editor TEI integrato.

Le trascrizioni sono oggetto di revisione da parte di revisori alle trascrizioni.

- *Pubblicazione*

I manoscritti, una volta digitalizzati e superata la fase di revisione delle immagini, vengono pubblicati sul sistema e resi accessibili agli utenti del sistema. Le corrispondenti trascrizioni sono pubblicate successivamente, dopo la validazione della stessa.

Attori del sistema:

- *Amministratore: gestione generale del sistema*
- *Acquisitore: acquisizione/digitalizzazione immagine*
- *Revisore acquisizioni: revisione e verifica correttezza dell'acquisizione*
- *Trascrittore: trascrizione del testo TEI*
- *Revisore trascrizioni: revisione e validazione della trascrizione*
- *Utente base: visualizzazione elenco titoli opere*
- *Utente avanzato: visualizzazione completa delle opere*

REQUISITI

Il sistema Digital Library è finalizzato alla conservazione di testi in formato digitale.

Ogni manoscritto preso in analisi costituirà un artefatto chiamato Opera.

L'opera sarà costituita dall'insieme di una raccolta di immagini raffiguranti le pagine, realizzata dagli **acquisitori**.

In aggiunta ad ogni raccolta potrà essere associata una trascrizione del manoscritto in formato TEI. Tale compito è delegato ai **trascrittori**.

Dunque il sistema sarà predisposto ad accogliere dati testuali relativi alla trascrizione dei manoscritti in tale formato, inseriti tramite text editor di supporto.

Saranno messi a disposizione dei supervisor strumenti per il controllo della corretta acquisizione delle immagini e per l'associazione tra queste e l'opera relativa, con annessi metadati. Tali supervisor sono i **revisori delle acquisizioni**.

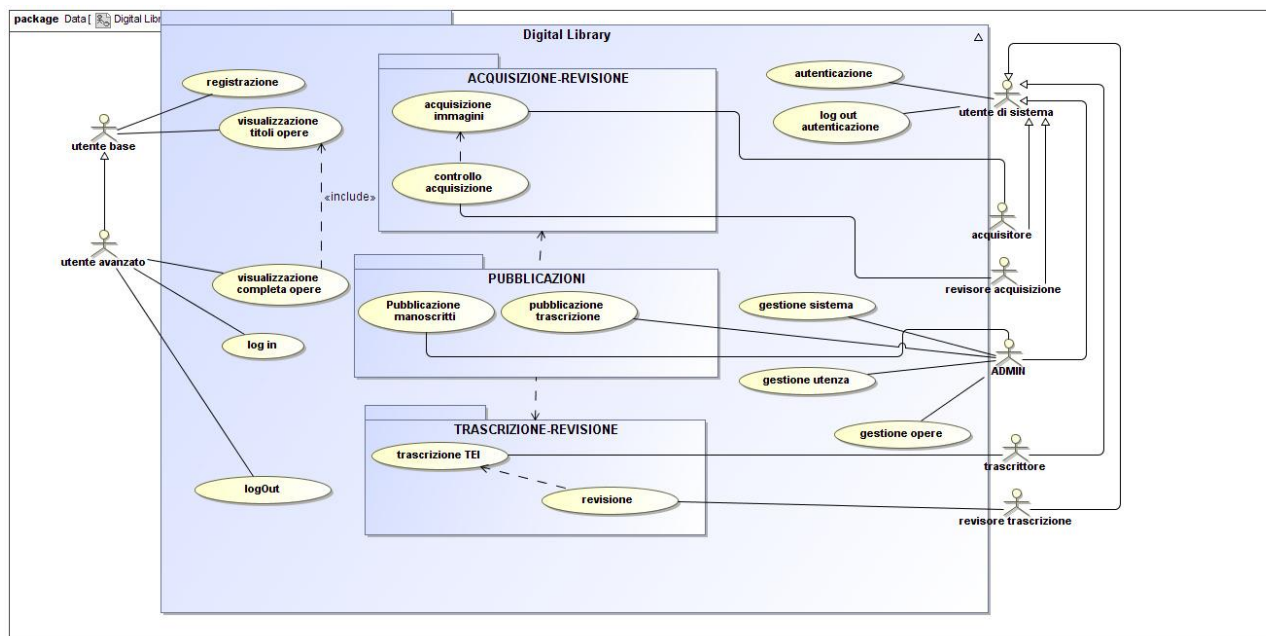
I **revisori delle trascrizioni** avranno la possibilità di avere accesso alle trascrizioni, validarle e associarle alle immagini corrispondenti già presenti nel sistema.

L'**utente base** avrà accesso parziale alle opere, ossia al solo elenco dei titoli, mentre l'**utente avanzato**, registrato opportunamente nel sistema, avrà accesso alla totalità dell'opera da lui cercata.

All'**amministratore** del sistema è permesso di pubblicare le opere, precedentemente revisionate, siano esse costituite dalle sole immagini o arricchite con la relativa trascrizione. Inoltre potrà gestire opportunamente gli utenti del sistema (trascrittori, revisori, acquisitori).

USE CASE DIAGRAM

Il seguente diagramma mostra in maniera concisa l'insieme delle funzionalità del sistema:

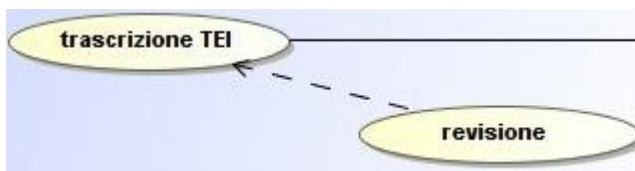


Riassumendo, il sistema mette a disposizione le seguenti attività:

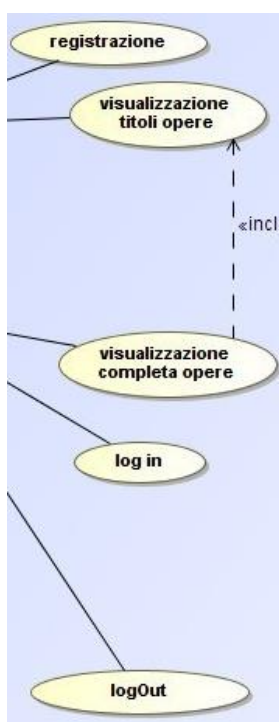
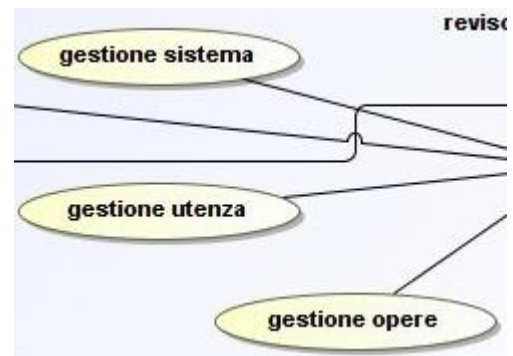
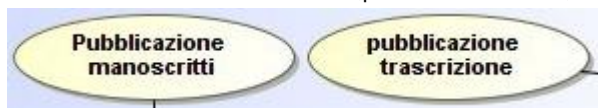
- Acquisizione immagini
- Revisione immagini



- Trascrizione manoscritti
- Revisione trascrizioni



- Pubblicazione manoscritti
- Pubblicazione trascrizione
- Associazione immagini-opere
- Gestione dell'utenza di sistema (promozione ecc....)

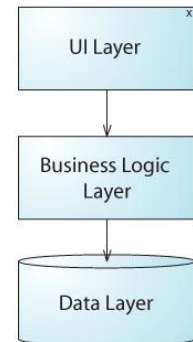


- Visualizzazione parziale e totale delle opere
- Assegnazione opere-staff
- Operazioni di registrazione e di login

SYSTEM DESIGN

Il sistema è stato opportunamente decomposto secondo il principio della separation of concerns, al fine di individuarne le componenti autonome.

Si è scelto quindi di adottare una architettura a 3 layer, ossia un modello di organizzazione del codice applicativo basato sulla separazione delle funzionalità logiche. La suddivisione avviene come segue:



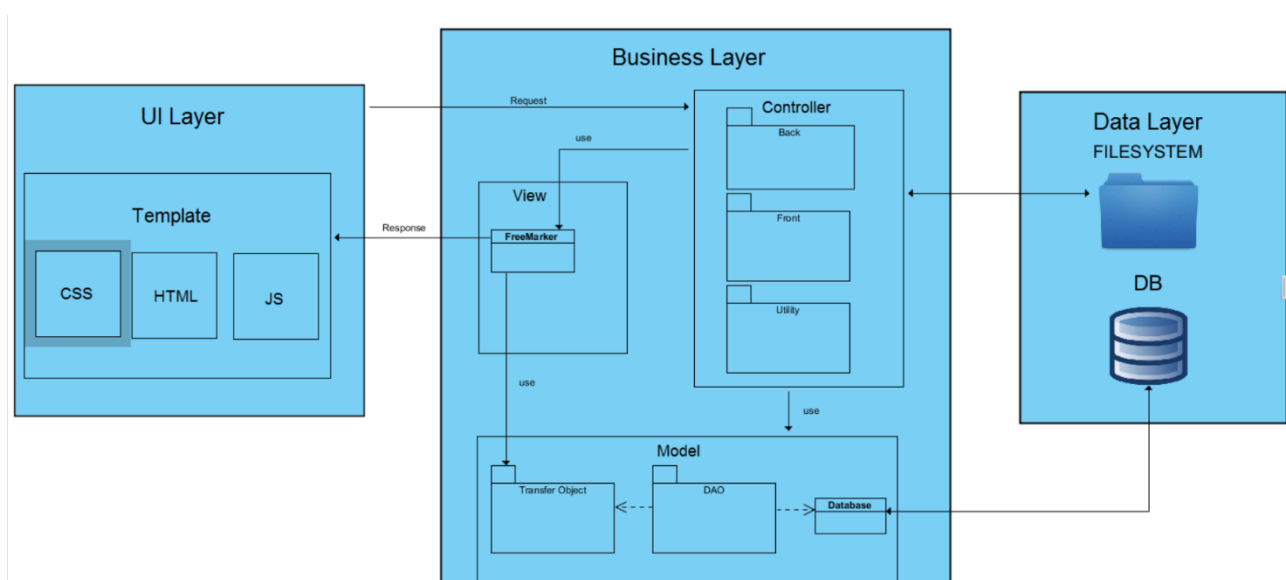
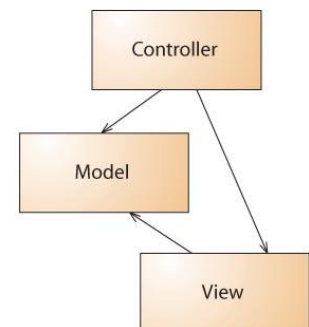
Presentation (PL): vale a dire UI Layer, la visualizzazione dei dati e, più in generale, la rappresentazione dei controlli (forms, controlli di input, labels, ecc.) necessari per l'interfaccia utente. Queste nel nostro sistema sono fornite dal template utilizzato.

Business Logic (BLL): rappresenta la parte principale dell'applicazione, definendo il **domain model** dell'applicazione, ovvero l'insieme delle entità, le loro relazioni e le logiche applicative.

Data Access (DAL): contiene tutto quello che concerne la persistenza dei dati (database, file system, ecc.)

Questo tipo di architettura ha principalmente il vantaggio di fornire una **separazione logica** così da aumentarne la manutenibilità, la scalabilità ed il riutilizzo. A seguito della valutazione di tali vantaggi, abbiamo ritenuto ottimale adottare questo tipo di design architetturale.

Nello specifico, per il Business Layer è stato pensato un pattern MVC, essendo il più indicato per applicazioni di tale fattura, ove è opportuno separare la logica di presentazione (**diversa** dalla semplice presentazione) dei dati dalla logica di business.



Il classico pattern MVC prevede la suddivisione concettuale dell'implementazione in **Model, View e Controller**

Model: è l'insieme dei componenti che mantengono lo "stato", i dati e i metodi per accedervi. Nella fattispecie comprende i transfer Object e i Dao che permettono la comunicazione con il database.

View: deputata alla visualizzazione vera e propria dell'interfaccia utente per la presentazione dei dati, rappresentata in questo caso dalla libreria di Freemarker. È un template engine basato su Java utilizzabile sia in modalità stand-alone che in un ambiente Web. Esso prevede un Template file che contiene l'output da visualizzare associato a dei place-holder riguardanti eventuali parti che vogliamo fornire dinamicamente.

<FreeMarker>

<http://freemarker.org/>

Controller: gestisce le interazioni dell'utente con l'applicazione (tipicamente intercettando e gestendo gli eventi e gli input dell'utente), mediante accesso al Model e definendo la View corrispondente da presentare.

L'insieme di queste caratteristiche rendono il pattern MVC ideale per l'implementazione della Digital Library.

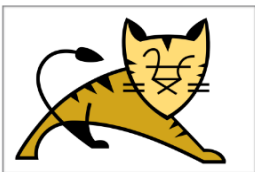
L'UI Layer effettua le request al controller della Business Layer. Il controller elabora la risposta e la formalizza mettendola a disposizione della View. I dati così elaborati vengono restituiti alla View, (freemarker), che si occupa del corretto completamento del template utilizzato.

Le componenti di business logic si appoggiano al modello, costituito dai **transfer object**, che realizzano l'astrazione necessaria al trattamento dei dati.

Le informazioni sono incapsulate tramite oggetti **DAO**, in modo da garantire l'opportuna persistenza dei dati tramite la connessione a database. Una volta ottenuto il dato richiesto, la business logic come detto restituisce tali informazioni alla View: anche quest'ultima si appoggia ai transfer object messi a disposizione dal model.



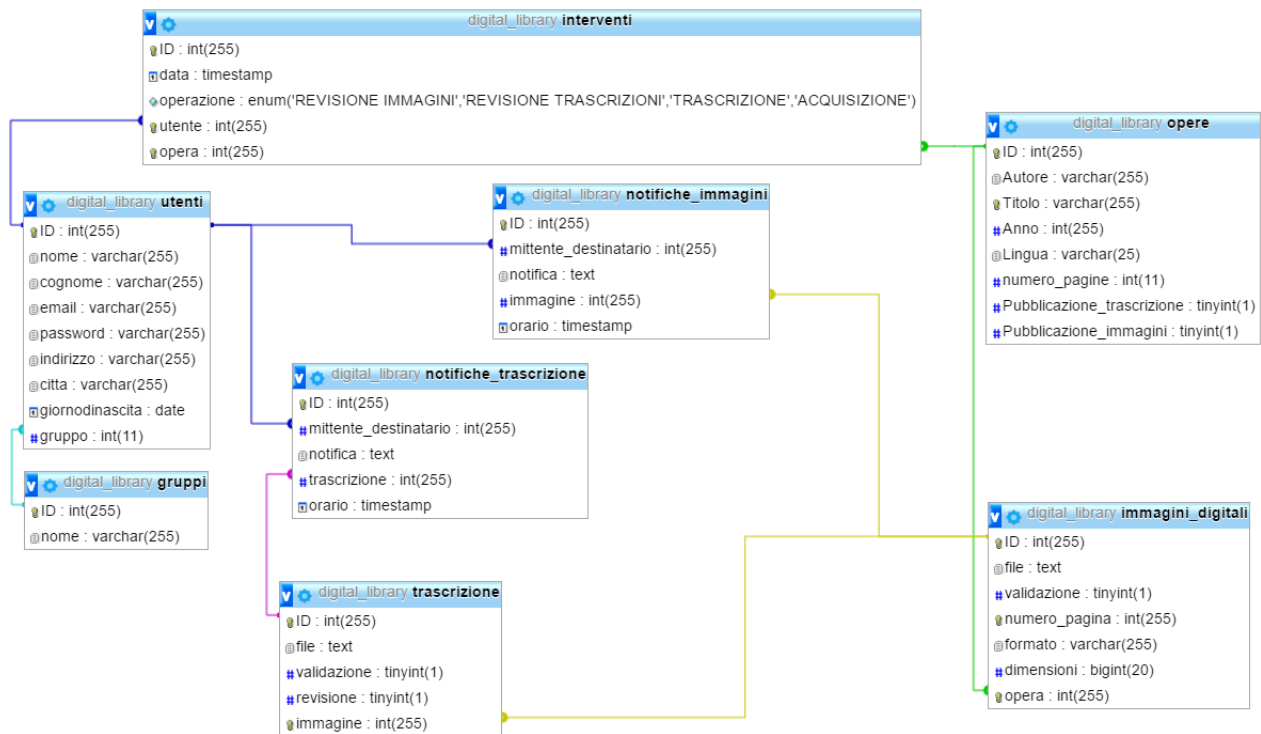
L'implementazione della **business logic** sfrutta oltre alla già citata libreria di Freemarker le seguenti tecnologie: **Commons**. Ovvero un insieme di librerie che contengono software Java riusabile e open source. **Apache Tomcat** (o semplicemente **Tomcat**). Un application server nella forma di contenitore servlet open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e Servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache.



DIGITAL LIBRARY DATABASE



A seguito dell'analisi dei requisiti e della stesura degli use case relativi, il database di sistema è stato organizzato come segue, e sviluppato nel linguaggio MySQL:

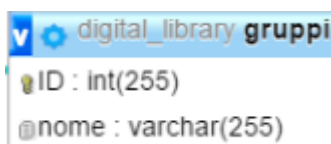


| digital_library utenti | |
|------------------------|--------------|
| ID | int(255) |
| @nome | varchar(255) |
| @cognome | varchar(255) |
| @email | varchar(255) |
| @password | varchar(255) |
| @indirizzo | varchar(255) |
| @citta | varchar(255) |
| @giornodinasita | date |
| #gruppo | int(11) |

Gli Utenti:

Un gruppo può contenere molteplici utenti, mentre un utente di sistema può loggarsi o autenticarsi sia come utente avanzato sia accedendo alla funzione di autenticazione, per accedere al ruolo assegnato all'interno del sistema (revisore, acquirente, ecc...). La decisione di mantenere un unico profilo utente per ruoli diversi, nel caso di utente di sistema, è stata presa guardando alla possibilità di verificare la corretta visualizzazione del contenuto della Digital Library in modalità front office.

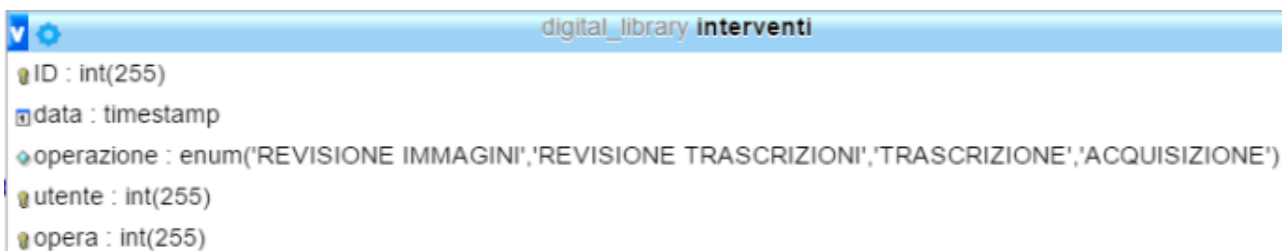
Gruppi: Il numero di record di questa tabella è fissato a 6 nel seguente ordine:



| digital_library gruppi |
|------------------------|
| ID : int(255) |
| nome : varchar(255) |

1. Utente avanzato
2. Acquisitore
3. Revisore Acquisitore
4. Trascrittore
5. Revisore Trascrizioni
6. Amministratore

La tabella **Interventi**:



| digital_library interventi |
|--|
| ID : int(255) |
| data : timestamp |
| operazione : enum('REVISIONE IMMAGINI','REVISIONE TRASCRIZIONI','TRASCRIZIONE','ACQUISIZIONE') |
| utente : int(255) |
| opera : int(255) |

Con questi attributi si esprime il ruolo dato da un utente di sistema nel lavorare su una determinata opera, (ovverosia revisionare, pubblicare, trascrivere ecc....).

Le **Opere**:



| digital_library opere |
|---|
| ID : int(255) |
| Autore : varchar(255) |
| Titolo : varchar(255) |
| Anno : int(255) |
| Lingua : varchar(25) |
| numero_pagine : int(11) |
| Pubblicazione_trascrizione : tinyint(1) |
| Pubblicazione_immagini : tinyint(1) |

Gli ultimi due attributi specificano se l'opera presenta o meno immagini e trascrizioni validate, necessarie per la pubblicazione (ricordiamo che un'opera può essere pubblicata anche con le sole immagini scannerizzate e validate).

La relazione che intercorre tra Opere e Immagini Digitali, denota il fatto che ad un'opera possano essere associate l'insieme di immagini che costituiscono il manoscritto originale, ma soprattutto il fatto che un'immagine è parte di una singola opera.

Analizzando nel dettaglio le Immagini Digitali, notiamo i seguenti attributi:



| digital_library immagini_digitali |
|-----------------------------------|
| ID : int(255) |
| file : text |
| validazione : tinyint(1) |
| numero_pagina : int(255) |
| formato : varchar(255) |
| dimensioni : bigint(20) |
| opera : int(255) |

Anzitutto l'attributo File contiene la stringa atta a completare il path del file, nella View.

L'attributo validazione esprime la correttezza dell'immagine in questione. Qualora le immagini di un'opera siano tutte validate, sarà possibile pubblicarla.

Ad ogni Immagine può essere associato un file contenente la trascrizione della pagina corrispondente del manoscritto originale, in formato TEI.

Con la relazione che intercorre tra Immagini e Trascrizioni si denota la possibilità di un'immagine di non avere nessuna trascrizione associata o al massimo una, tramite la cardinalità (0, 1).

| digital_library trascrizione | |
|------------------------------|--------------|
| ID | : int(255) |
| file | : text |
| validazione | : tinyint(1) |
| revisione | : tinyint(1) |
| immagine | : int(255) |

L'entità **Trascrizione** è rappresentata da questi attributi i cui significati sono analoghi agli attributi dell'entità Immagini ad eccezione del campo file che conterrà la stringa TEI xml.

Per favorire il corretto svolgimento delle operazioni di back office si ritiene essenziale una comunicazione coordinata tra le varie figure in gioco.

Per questo si rendono necessarie le entità **Notifiche_immagini** e **Notifiche_trascrizione**.

| digital_library notifiche_immagini | |
|------------------------------------|-------------|
| ID | : int(255) |
| mittente_destinatario | : int(255) |
| notifica | : text |
| immagine | : int(255) |
| orario | : timestamp |

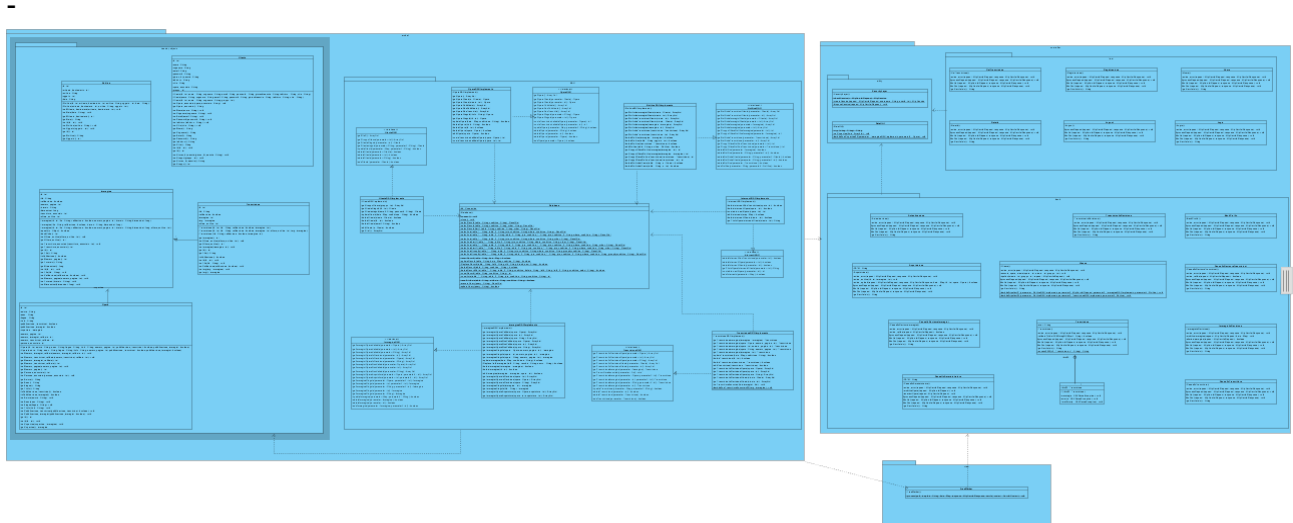
| digital_library notifiche_trascrizione | |
|--|-------------|
| ID | : int(255) |
| mittente_destinatario | : int(255) |
| notifica | : text |
| trascrizione | : int(255) |
| orario | : timestamp |

Il tipico record di queste tabelle indica una notifica relativa ad un'immagine/trascrizione di interesse, e l'utenza di sistema che invia tale notifica e che si aspetterà una probabile risposta (utente indicato con il campo mittente_destinatario: egli infatti indicherà il mittente in caso di invio di notifica, il destinatario in caso di ricezione).

SOFTWARE OBJECT DESIGN

Il Class Diagram contenente le reali classi, interfacce e i relativi membri e metodi utilizzati nella Digital Library.

Ovviamente la struttura di base dovrà rispecchiare le caratteristiche dell'architettura, dunque si ripropone la suddivisione tra i package Model, View e Controller, sempre secondo il pattern MVC.



Ogni package ad eccezione della View è suddiviso in ulteriori sotto-package, così costituiti:

Model

- Transfer Object
- DAO Objects

View

- Contenente la sola classe Freemarker che richiama le librerie apposite.

Controller

- Front (classi che implementano il front office)
- Back (classi che implementano il back office)
- Utility (classi che implementano la gestione delle sessioni e utility per i dati)

Package Model

Transfer Object con all'interno le classi:

Utente

| Utente |
|--|
| <pre>-id : int -nome : String -cognome : String -email : String -password : String -giorno_di_nascita : String -indirizzo : String -citta : String -opera_associata : String -gruppo : int +Utente(id : int, nome : String, cognome : String, email : String, password : String, giornonascita : String, indirizzo : String, citta : String) +Utente(nome : String, cognome : String, email : String, password : String, giornonascita : String, indirizzo : String, citta : String) +Utente(id : int, nome : String, cognome : String, gruppo : int) +setOpera_associata(opera_associata : String) : void +getOpera_associata() : String +setName(nome : String) : void +setCognome(cognome : String) : void +setEmail(email : String) : void +setPassword(password : String) : void +setIndirizzo(indirizzo : String) : void +setCitta(citta : String) : void +getName() : String +getCognome() : String +getEmail() : String +getPassword() : String +getIndirizzo() : String +getCitta() : String +setId(id : int) : void +getId() : int +setGiorno_di_nascita(giorno_di_nascita : String) : void +setGruppo(gruppo : int) : void +getGiorno_di_nascita() : String +getGruppo() : int</pre> |

Opera

| Opera |
|--|
| <pre>-id : int -autore : String -anno : String -lingua : String -titolo : String -pubblicazione_trascrizioni : boolean -pubblicazione_immagini : boolean -copertina : Immagine -numero_pagine : int -numero_immagini_validate : int -numero_trascrizioni_validate : int -persone_associate : int +Opera(id : int, autore : String, anno : String, lingua : String, titolo : String, numero_pagine : int, pubblicazione_trascrizioni : boolean, pubblicazione_immagini : boolean) +Opera(autore : String, anno : String, lingua : String, titolo : String, numero_pagine : int, pubblicazione_trascrizioni : boolean, pubblicazione_immagini : boolean) +setNumero_immagini_validate(numero_immagini_validate : int) : void +setNumero_trascrizioni_validate(numero_trascrizioni_validate : int) : void +getNumero_immagini_validate() : int +getNumero_trascrizioni_validate() : int +setNumero_pagine(numero_pagine : int) : void +getNumero_pagine() : int +getPersone_associate() : int +setPersone_associate(persone_associate : int) : void +getAutore() : String +getAnno() : String +getLingua() : String +getTitolo() : String +isPubblicazione_trascrizioni() : boolean +isPubblicazione_immagini() : boolean +setAutore(autore : String) : void +setAnno(anno : String) : void +setLingua(lingua : String) : void +setTitolo(titolo : String) : void +setPubblicazione_trascrizioni(pubblicazione_trascrizioni : boolean) : void +setPubblicazione_immagini(pubblicazione_immagini : boolean) : void +getId() : int +setId(id : int) : void +setCopertina(copertina : Immagine) : void +getCopertina() : Immagine</pre> |

Immagine

| Immagine |
|--|
| <div><div>-id : int -file : String -validazione : boolean -numero_pagina : int -formato : String -dimensioni : long -trascrittore_associato : int -ultima_notifica : int</div></div> |
| <div><div>+Immagine(id : int, file : String, validazione : boolean, numero_pagina : int, formato : String, dimensioni : long) +Immagine(file : String, validazione : boolean, formato : String, dimensioni : long) +Immagine(id : int, file : String, validazione : boolean, numero_pagina : int, formato : String, dimensioni : long, ultima_notifica : int) +equals(o : Object) : boolean +hashCode() : int +setUltima_notifica(ultima_notifica : int) : void +getUltima_notifica() : int +setTrascrittore_associato(trascrittore_associato : int) : void +getTrascrittore_associato() : int +getId() : int +getFile() : String +isValidazione() : boolean +getNumero_pagina() : int +getFormato() : String +getDimensioni() : long +setId(id : int) : void +setFile(file : String) : void +setValidazione(validazione : boolean) : void +setNumero_pagina(numero_pagina : int) : void +setFormato(formato : String) : void +setDimensioni(dimensioni : long) : void</div></div> |

Trascrizione

| Trascrizione |
|--|
| <div><div>-id : int -file : String -validazione : boolean -immagine : int -img : Immagine -ultima_notifica : int</div></div> |
| <div><div>+Trascrizione(id : int, file : String, validazione : boolean, immagine : int) +Trascrizione(id : int, file : String, validazione : boolean, immagine : int, ultima_notifica : int, img : Immagine) +Trascrizione(file : String, validazione : boolean, immagine : int) +getImmagine() : int +setUltima_notifica(ultima_notifica : int) : void +getUltima_notifica() : int +setImmagine(immagine : int) : void +getId() : int +getFile() : String +isValidazione() : boolean +setId(id : int) : void +setFile(file : String) : void +setValidazione(validazione : boolean) : void +setImg(img : Immagine) : void +getImg() : Immagine</div></div> |

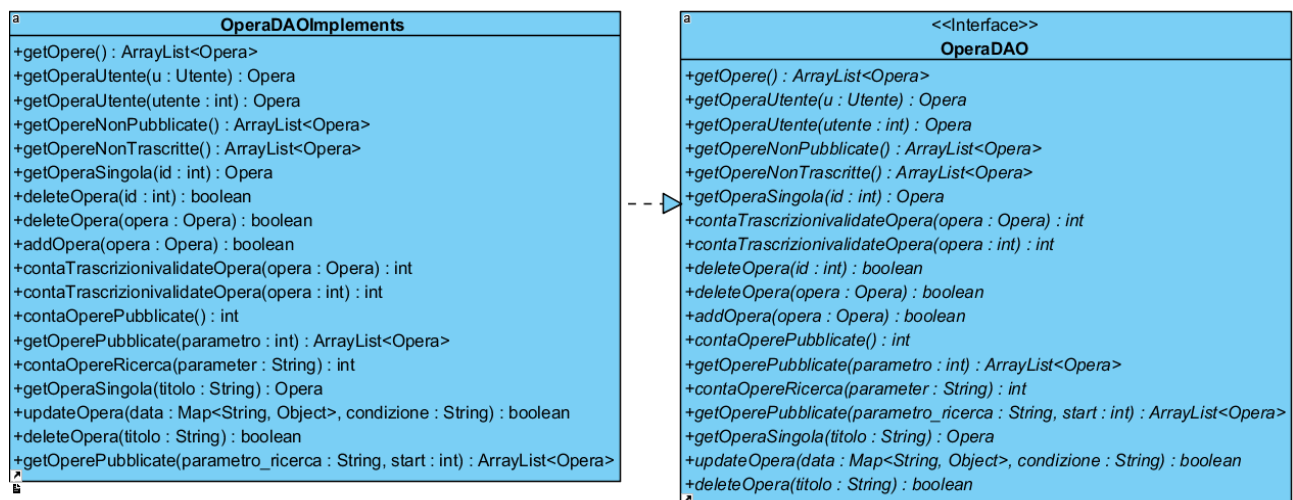
Notifica

| Notifica |
|--|
| -id : int -mittente_destinatario : int -notifica : String -oggetto : int -date : String |
| +Notifica(id : int, mittente_destinatario : int, notifica : String, oggetto : int, date : String) +Notifica(mittente_destinatario : int, notifica : String, oggetto : int) +setMittente_destinatario(mittente_destinatario : int) : void +setDate(date : String) : void +getMittente_destinatario() : int +getDate() : String +setId(id : int) : void +setNotifica(notifica : String) : void +setOggetto(oggetto : int) : void +getId() : int +getNotifica() : String +getOggetto() : int |

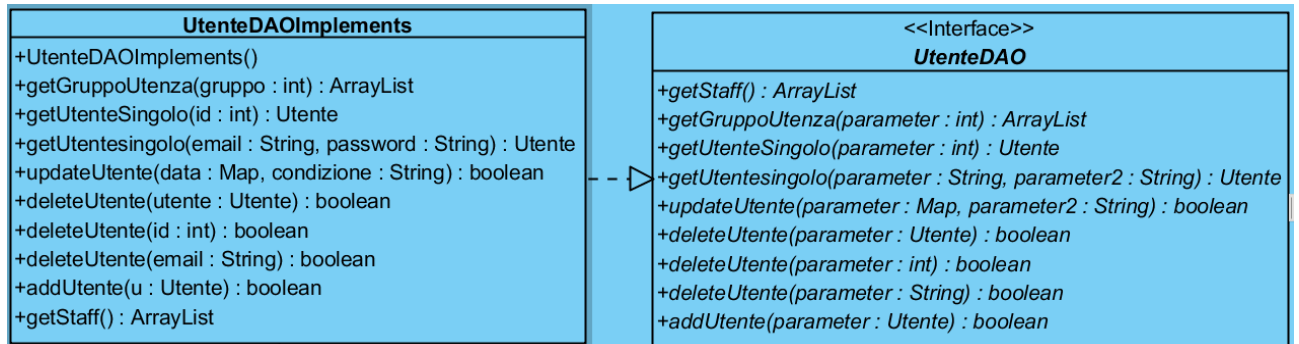
Esse rappresentano i contenitori dei dati di interesse, i cui membri e metodi rispecchiano l'organizzazione presente nel database del sistema, precedentemente trattato nella documentazione.

DAO Objects contenente le interfacceDAO e le relative implementazioni:

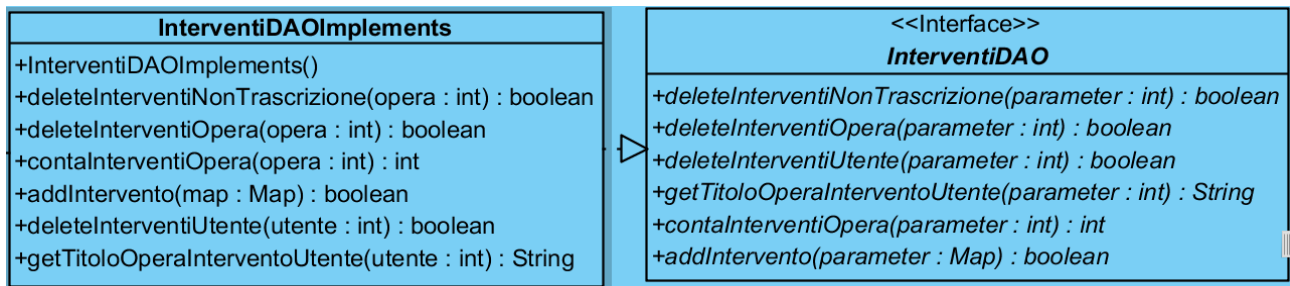
- OperaDAO
- OperaDAOImplements



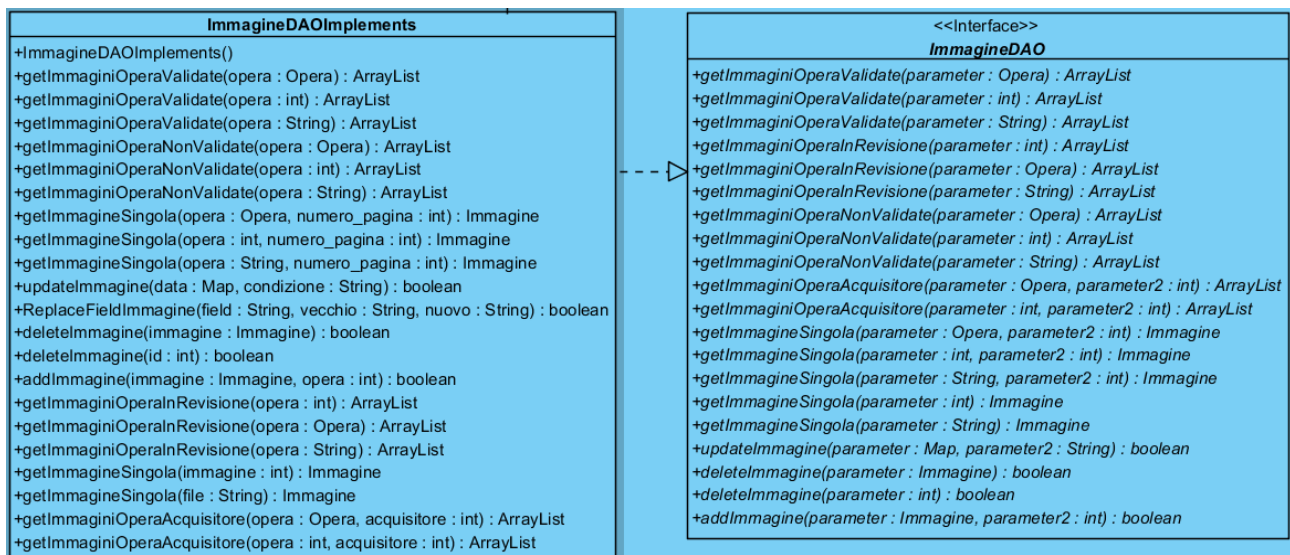
- UtenteDAO
- UtenteDAOImplements



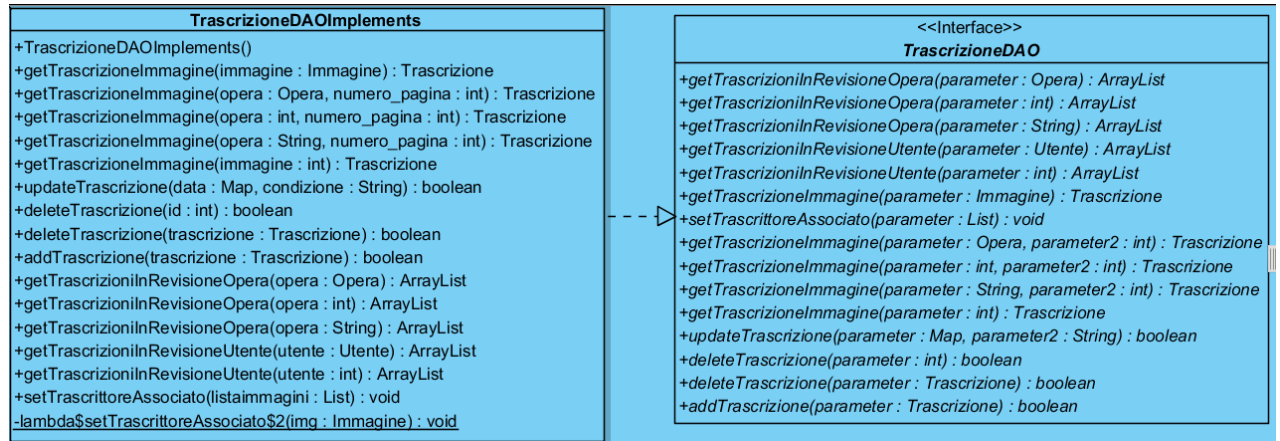
- InterventiDAO
- InterventiDAOImplements



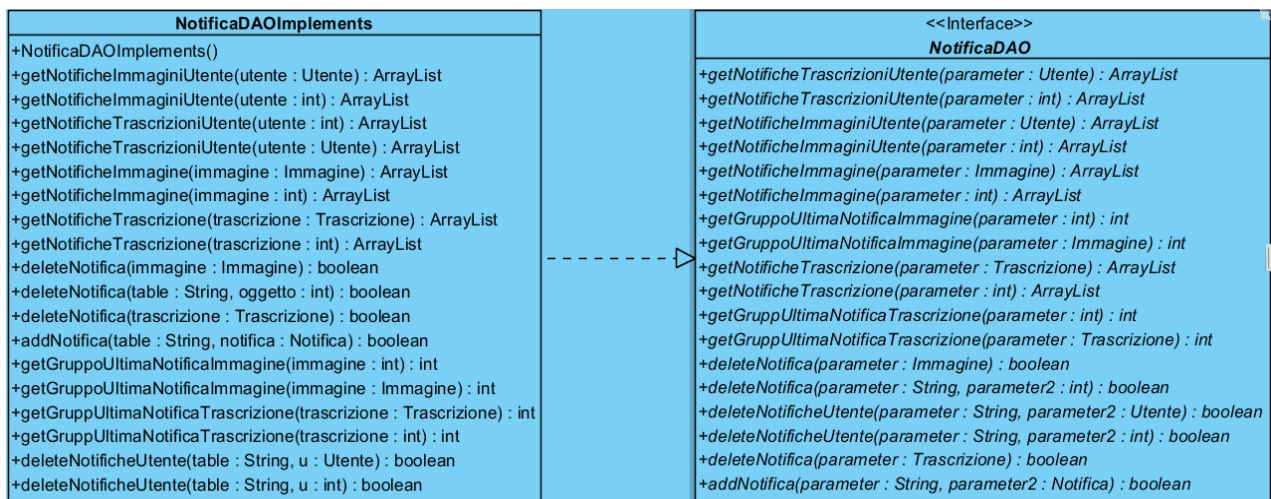
- ImmagineDAO
- ImmagineDAOImplements



- TrascrizioneDAO
- TrascrizioneDAOImplements



- NotificaDAO
- NotificaDAOImplements



Tutto il package si appoggia alla classe database, deputata alle CRUDE operations. Utilizza i driver JDBC, e dunque nello specifico la libreria MySQLconnector.jar.



Package View

- Freemarker

È la classe deputata alla configurazione della libreria di Freemarker.

http://freemarker.org/docs/pgui_config.html

| FreeMarker |
|---|
| +FreeMarker() <u>+process(path template : String, data : Map, response : HttpServletResponse, servlet_context : ServletContext) : void</u> |

Package Controller

Il package **Utility** contiene le classi:

- SecurityLayer
- DataUtil

| SecurityLayer |
|---|
| +SecurityLayer() <u>+checkSession(r : HttpServletRequest) : HttpSession</u> <u>+createSession(request : HttpServletRequest, username : String, userid : int) : HttpSession</u> <u>+disposeSession(request : HttpServletRequest) : void</u> |
| DataUtil |
| +DataUtil() <u>+crypt(string : String) : String</u> <u>+setCopertine(lista : ArrayList) : void</u> <u>-lambda\$setCopertine\$2(parameter : ImmagineDAOImplements, parameter2 : Opera) : void</u> |

Deputate alla gestione della sessione ed altre operazioni di interesse per il buon funzionamento del sistema.

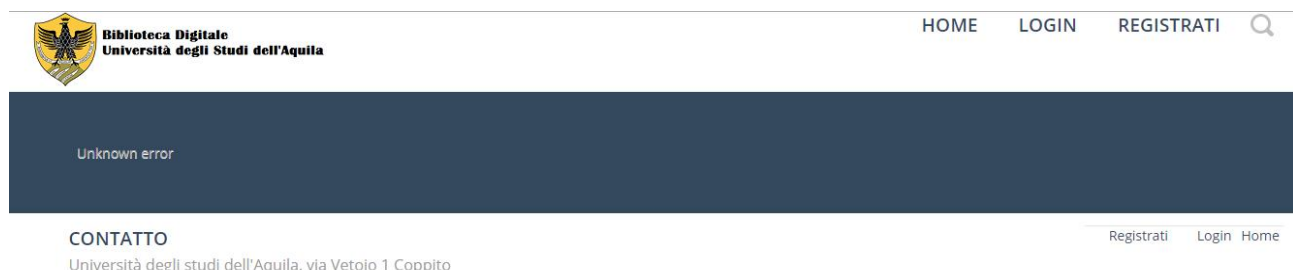
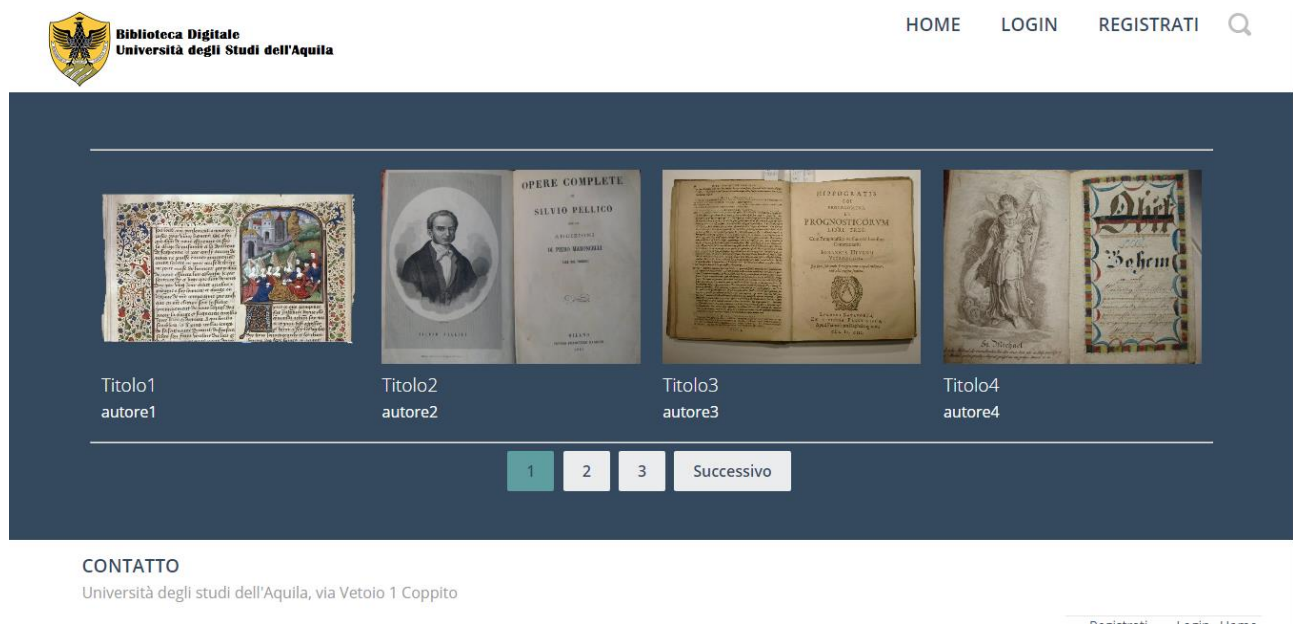
I package **front e back** contengono le servlets della nostra applicazione, in generale esse hanno tutte un metodo per processare la richiesta (chiamato indipendentemente se quest'ultima sia GET o POST) e un metodo per richiamare una pagina di errore se questo avviene. Alcuni esempi:

- Home

```
Home

+Home()
-action_error(request : HttpServletRequest, response : HttpServletResponse) : void
#processRequest(request : HttpServletRequest, response : HttpServletResponse) : void
#doGet(request : HttpServletRequest, response : HttpServletResponse) : void
#doPost(request : HttpServletRequest, response : HttpServletResponse) : void
+getServletInfo() : String
```

Screenshots(con pagina di errore):



- Registrazione

| Registrazione |
|---|
| <pre> +Registrazione() -action_error(request : HttpServletRequest, response : HttpServletResponse) : void #processRequest(request : HttpServletRequest, response : HttpServletResponse) : void #doGet(request : HttpServletRequest, response : HttpServletResponse) : void #doPost(request : HttpServletRequest, response : HttpServletResponse) : void +getServletInfo() : String </pre> |

Screenshot:

Biblioteca Digitale
Università degli Studi dell'Aquila

HOME LOGIN REGISTRATI 🔍

Email
Password
Conferma Password
Nome
Cognome
Città
Indirizzo
Data di nascita
gg/mm/aaaa

Registrati

- Login

| Login |
|---|
| <pre> +Login() -action_error(request : HttpServletRequest, response : HttpServletResponse) : void #processRequest(request : HttpServletRequest, response : HttpServletResponse) : void #doGet(request : HttpServletRequest, response : HttpServletResponse) : void #doPost(request : HttpServletRequest, response : HttpServletResponse) : void +getServletInfo() : String </pre> |

Screenshot:

Biblioteca Digitale
Università degli Studi dell'Aquila

HOME LOGIN REGISTRATI Q

Email *

Password *

Login

CONTATTO

Università degli studi dell'Aquila, via Vetoio 1 Coppito

Registrati Login Home

- Logout

| Logout |
|---|
| <pre>+Logout() #processRequest(request : HttpServletRequest, response : HttpServletResponse) : void #doGet(request : HttpServletRequest, response : HttpServletResponse) : void #doPost(request : HttpServletRequest, response : HttpServletResponse) : void +getServletInfo() : String</pre> |

Il package **back** in particolare contiene le servlets chiave per l'applicazione. Vedremo nel dettaglio alcune di queste con i sequence diagrams(dettagli se questi ultimi troppo grandi per entrare nella documentazione) di alcuni metodi.

- Acquisizione

È una servlet specifica per l'acquirente, permette l'upload delle immagini e la loro associazione alla pagina

Caricamento Immagini per Divina commedia

Selezione immagine

Scegli file | 2 file

numero pagina file divina_commedia copertina.jpg:

0

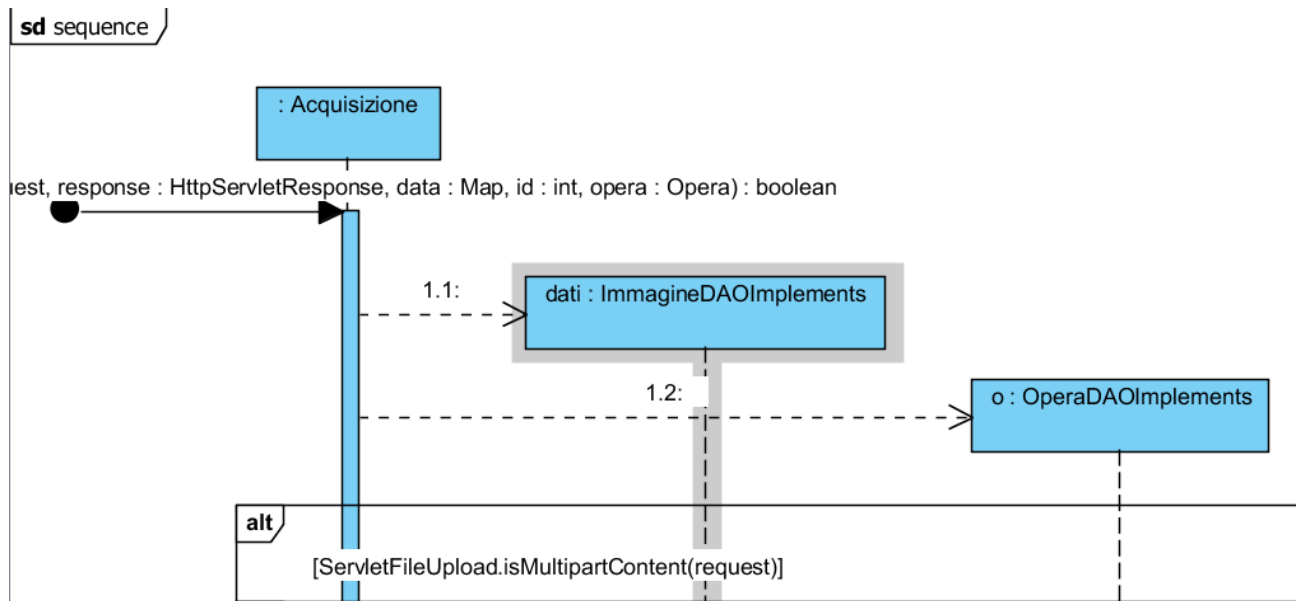
numero pagina file pag1.jpeg:

1

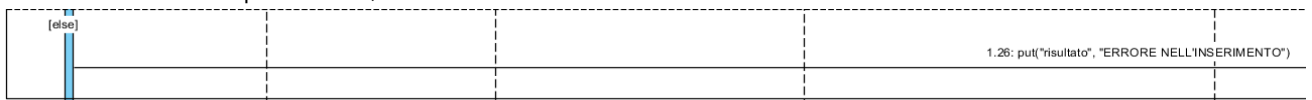
esegui annulla

| Acquisizione |
|---|
| <pre>-PATH : String +Acquisizione() -action_error(request : HttpServletRequest, response : HttpServletResponse) : void -action_notifica(id : int, immagine : int) : void -action_upload(request : HttpServletRequest, response : HttpServletResponse, data : Map, id : int, opera : Opera) : boolean #processRequest(request : HttpServletRequest, response : HttpServletResponse) : void #doGet(request : HttpServletRequest, response : HttpServletResponse) : void #doPost(request : HttpServletRequest, response : HttpServletResponse) : void +getServletInfo() : String</pre> |

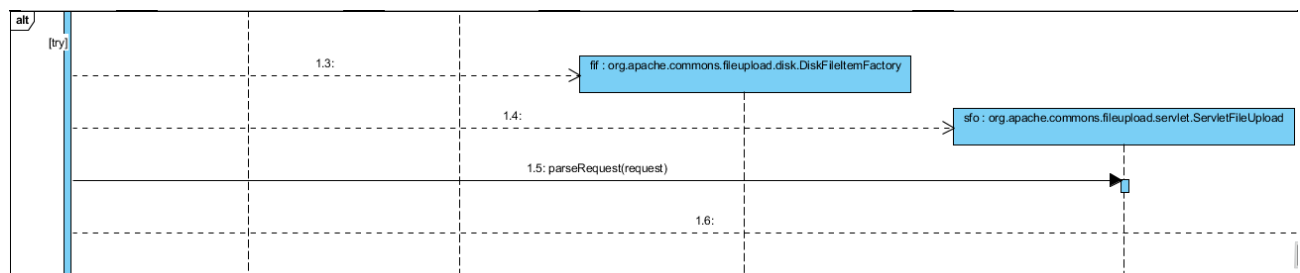
Dettagli sequence diagram metodo action_upload:



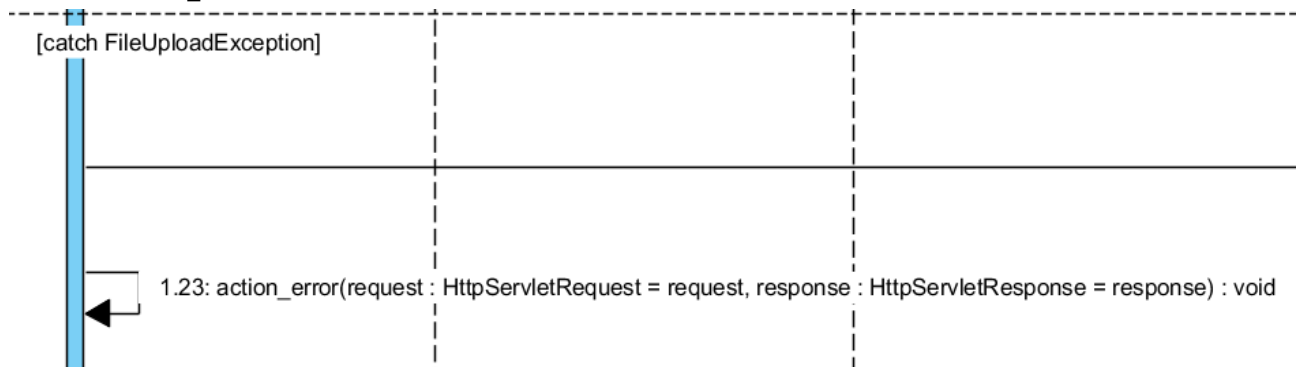
Come possiamo notare sopra il metodo action upload tornerà un booleano che rappresenterà il successo o meno dell'upload. Vengono inizializzati i DAOImplements e si effettua il controllo se la request è multipartContent(ovvero si sta effettuando un upload di file). Altrimenti è un insuccesso.



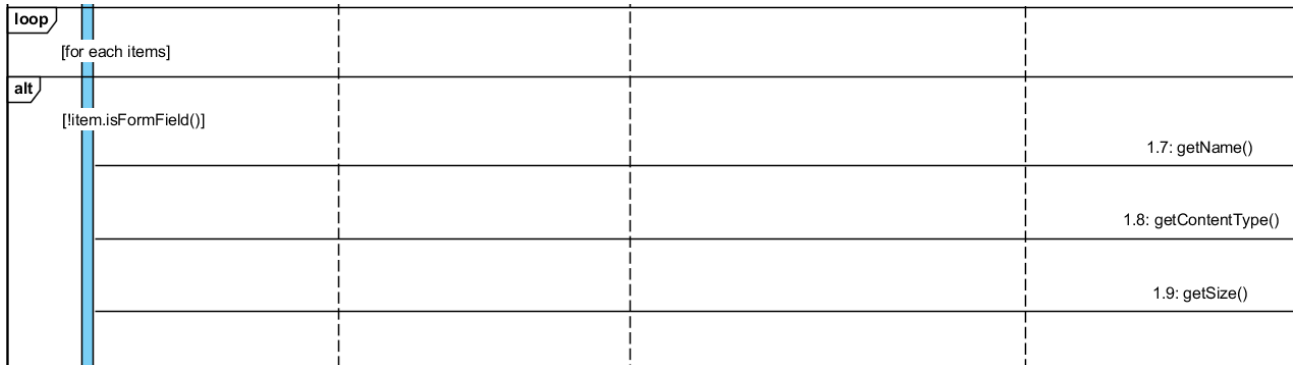
Successivamente si inizializzano la DiskFileItemFactory e la ServletFileUpload della libreria Apache commons.



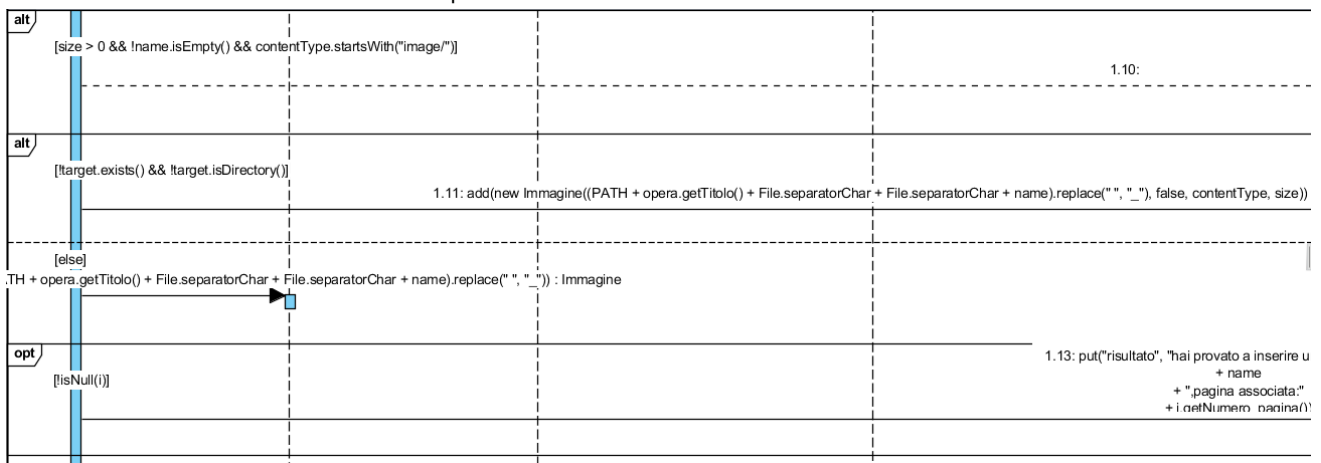
Queste librerie possono lanciare un'eccezione di tipo FileUploadException, in tal caso sarà un errore e richiameremo il metodo action_error



Si parse la request e per ogni oggetto inviato si controlla se è un campo form oppure un file. Noi siamo interessati a quest'ultimo caso dunque se l'item non è un campo form (e quindi un file) prendiamo il suo nome, il tipo del contenuto e la sua grandezza (in byte)

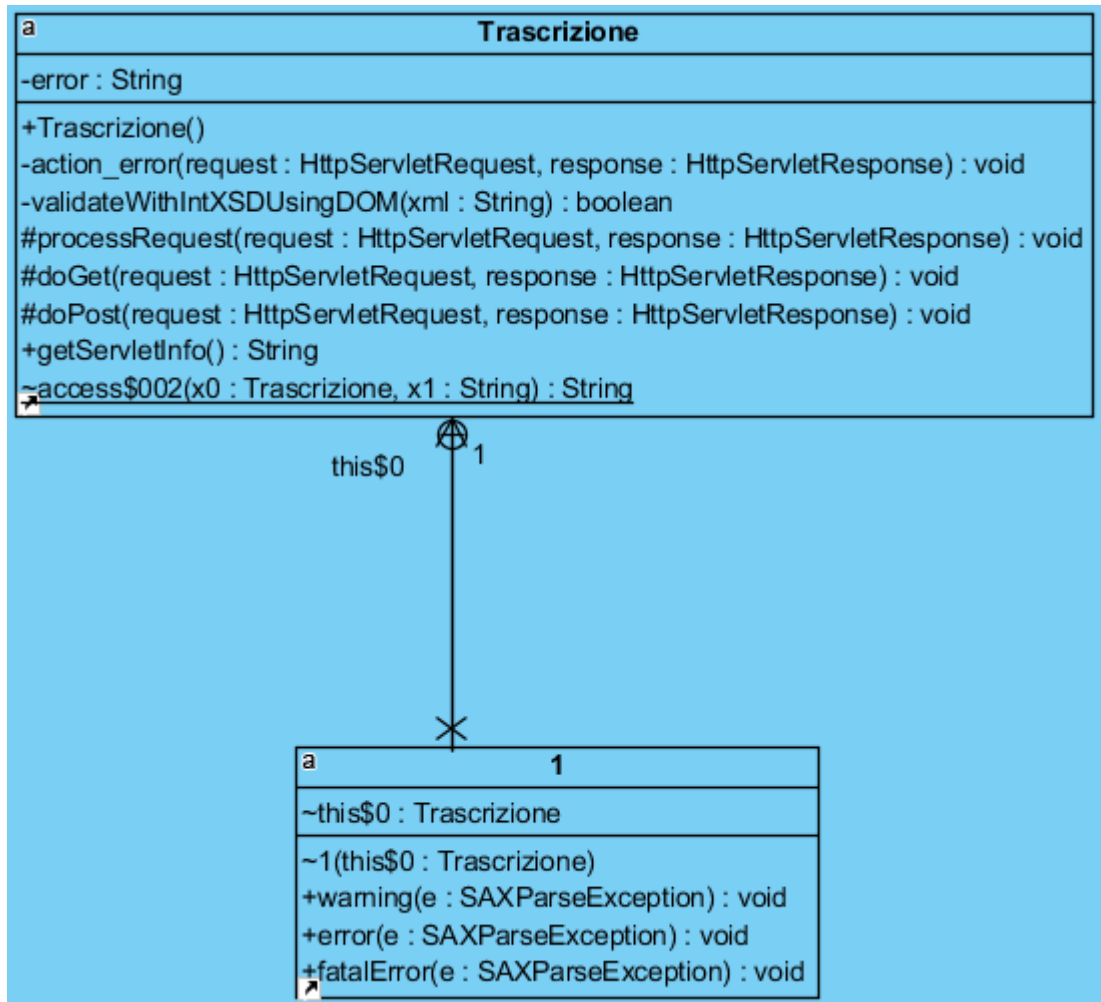


Non ci resta da controllare che il contenuto sia effettivamente un immagine, che abbia un nome non vuoto. L'ultimo controllo prima di salvare il file è controllare che questo non esista già o che il suo nome sia uguale a quello di una cartella. Se tutto viene confermato il file può essere salvato.

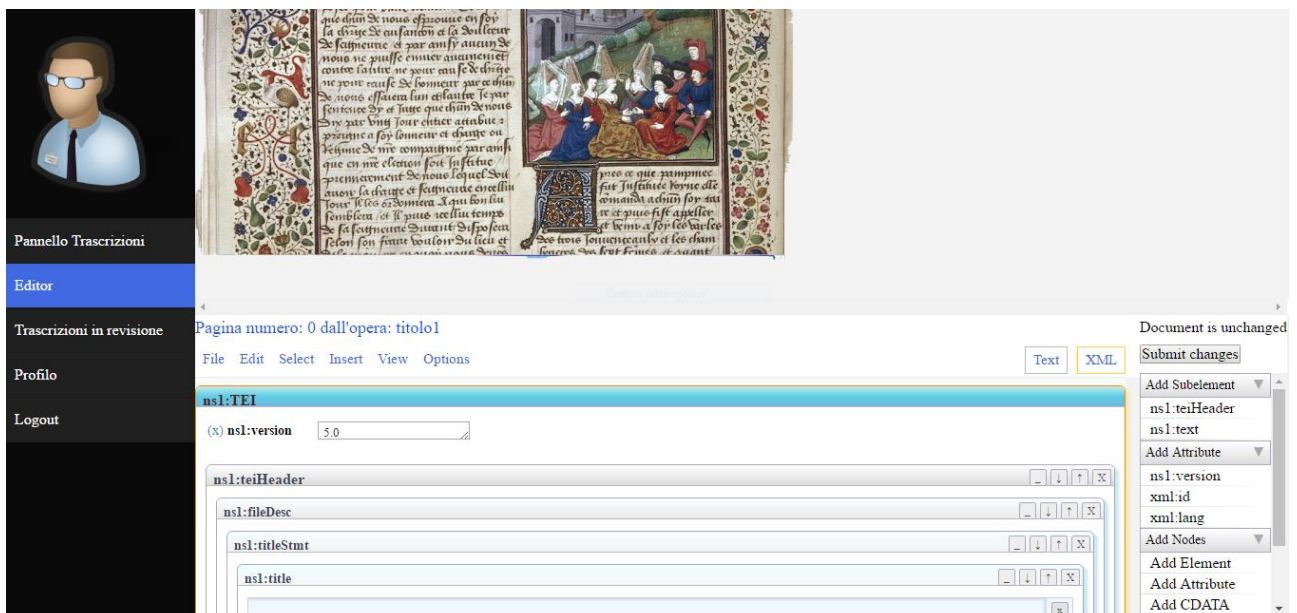


- Trascrizione

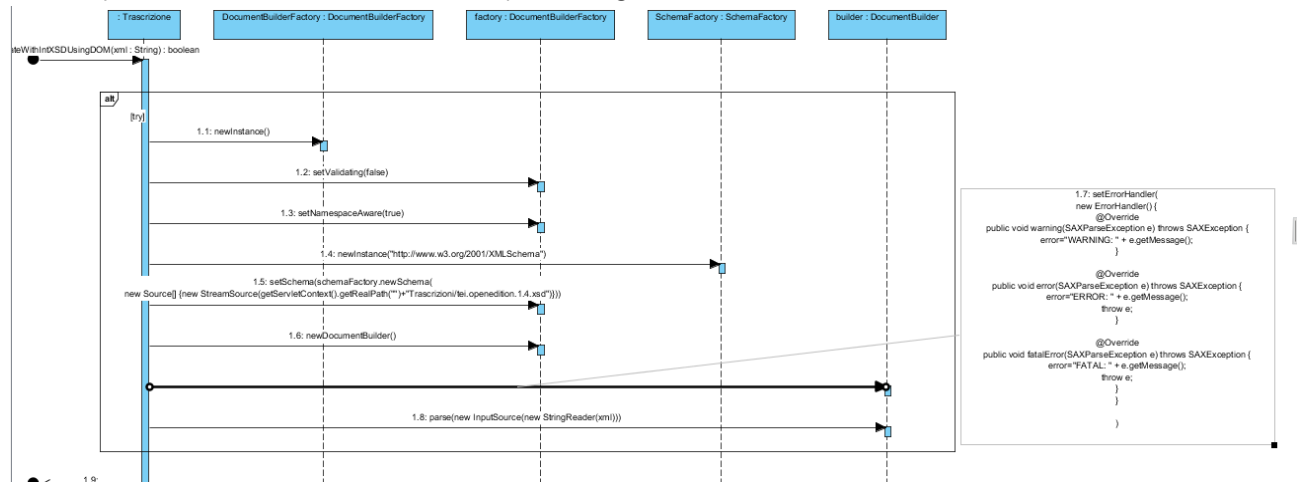
È una servlet specifica per il trascrittore e gli consente di inserire una trascrizione nel formato xml/tei nel database.



Screenshot:

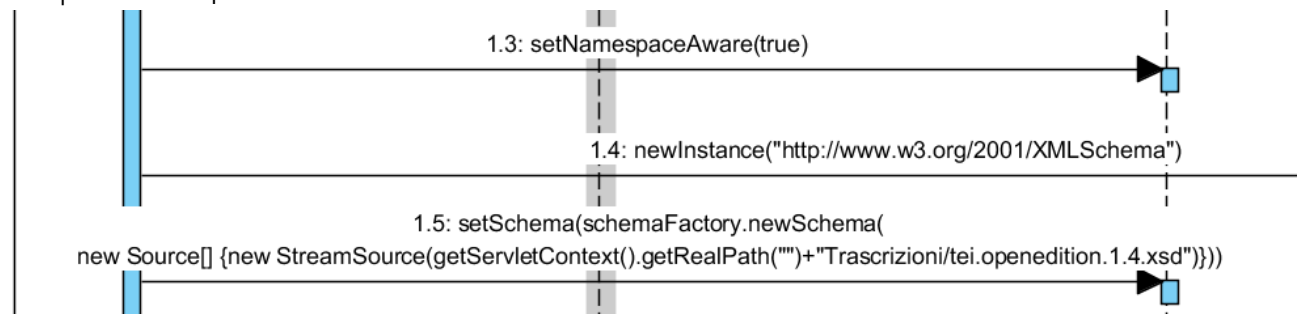


Come si può intuire, il trascrittore necessita di avere feedback da parte dell'applicazione, questi arrivano dal metodo per la validazione. Questo è il suo sequence diagram:

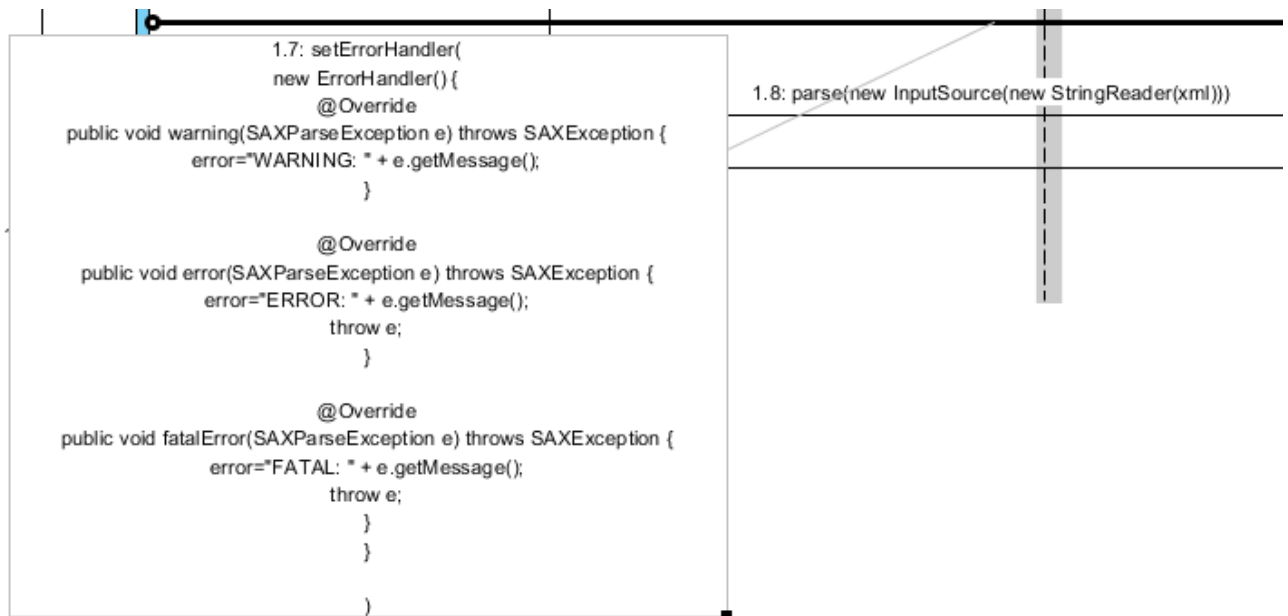


Nel dettaglio:

Inizializzo una DocumentBuildDactory e chiamo il metodo setNamespaceAware con true (fondamentale per l'XML TEI) e il controllo di sintassi tramite l'url allo schema standard del W3. Per quanto riguarda invece la semantica è necessario fornire uno schema (il path per lo schema tei che si vuole utilizzare). Nel mio caso tei.openedition.1.4.xsd



Una volta configurato il validatore è importante definire un error_handler nel caso la stringa tei non sia valida (Successivamente la risposta dell'error handler sarà mostrato all'utente per guidarlo alla correzione). Definito l'error handler si può finalmente parsare la stringa.



- PannelloRevisoreImmagini

È una servlet specifica per il revisore di immagini (non si discosta molto da quella per il revisore trascrizione) ed è presa ad esempio come servlet di un revisore.

PannelloRevisoreImmagini

+PannelloRevisoreImmagini()

-action_error(request : HttpServletRequest, response : HttpServletResponse) : void

-action_valida(request : HttpServletRequest) : boolean


#processRequest(request : HttpServletRequest, response : HttpServletResponse) : void

#doGet(request : HttpServletRequest, response : HttpServletResponse) : void

#doPost(request : HttpServletRequest, response : HttpServletResponse) : void

+getServletInfo() : String

Screenshot:



Pannello Revisione
Acquisizioni

Profilo

Logout

Immagini per Opera: Divina commedia

10 righe per pagina

| Anteprima | Numero Pagina | Formato | Dimensioni | Notifiche | Operazione |
|-------------------------------|---------------|------------|------------|--|--|
| divina_commedia_copertina.jpg | 0 | image/jpeg | 180,4x8 | Rispondi | Valida |
| img?jimg | 1 | image/jpeg | 627,4x8 | Rispondi | Valida |


In mostra 1 fino a 2 di 2 totali

Precedente 1 Successivo


Notifiche per Trascrizione pagina: 0

18/06/2016 16:41:28

Immagine Uploadata



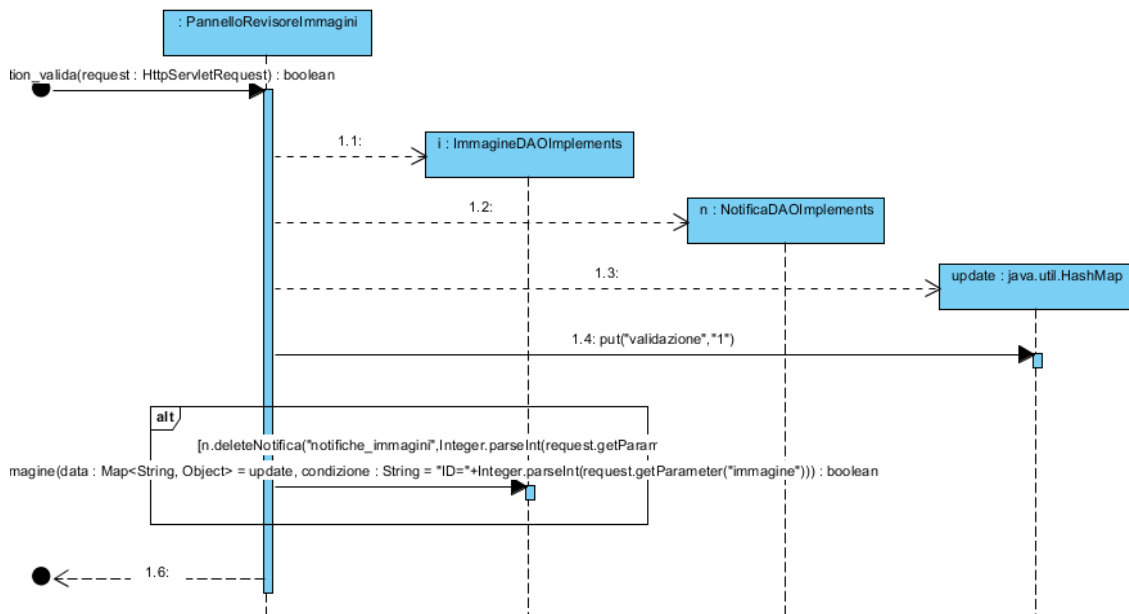
Revisore
ben fatto

Acquisitore 

18/06/2016 16:43:31

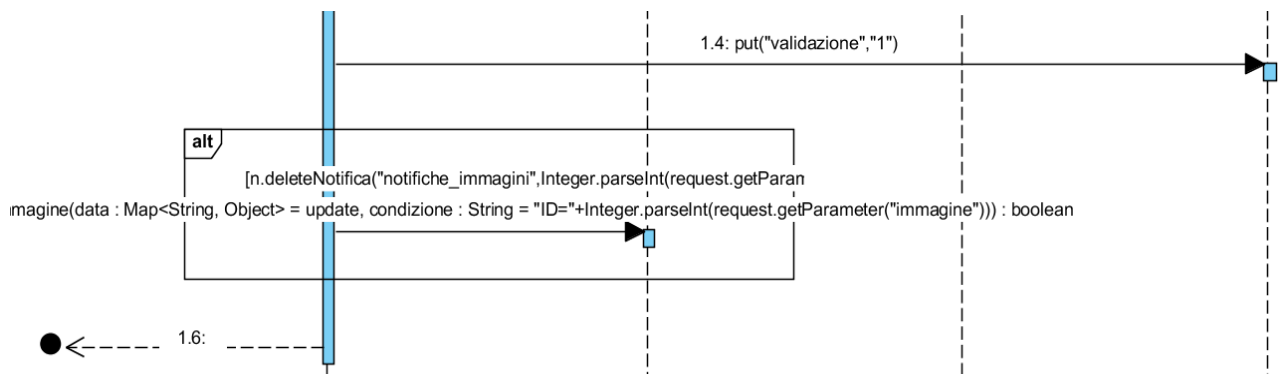
Inserisci il messaggio da inviare...

Il metodo del quale vedremo il sequence diagram è action_valida(request), necessario affinché un revisore possa validare un'immagine/trascrizione.



Nel dettaglio:

Bisogna assicurarsi che se la validazione va a buon fine, le notifiche relative siano cancellate.




- Utenze

L'ultima servlet che vedremo è specifica dell'amministratore e permette la promozione di utenti avanzati a utenti di sistema. Nonché l'associazione di quest'ultimi alle opere.

| Utenze |
|---|
| <pre> +Utenze() -action_error(request : HttpServletRequest, response : HttpServletResponse) : void -associa_opera_utente(opera : int, utente : int, gruppo : int) : void -ripulisci(utente : int, gruppo : int, request : HttpServletRequest) : void #processRequest(request : HttpServletRequest, response : HttpServletResponse) : void #doGet(request : HttpServletRequest, response : HttpServletResponse) : void #doPost(request : HttpServletRequest, response : HttpServletResponse) : void +getServletInfo() : String -lambda\$ripulisci\$1(parameter : NotificaDAOImplements, parameter2 : HttpServletRequest, parameter3 : ImmagineDAOImplements, parameter4 : Notifica) : void -lambda\$ripulisci\$0(parameter : NotificaDAOImplements, parameter2 : TrascrizioneDAOImplements, parameter3 : Notifica) : void </pre> |

Screenshot:



- Pannello Amministrazione
- Utenze**
- Profilo
- Logout

Staff

10 righe per pagina

Ricerca:

| Ruolo | Nome | Cognome | Opera Associata | Operazione |
|------------------------|----------|----------|-----------------|---------------|
| Acquisitore | utente1 | utente1 | Divina commedia | Degrada |
| Revisione Acquisizione | Gabriele | Giggino | Divina commedia | Degrada |
| Revisore Trascrizione | po | po | Divina commedia | Degrada |
| Revisore Trascrizione | utente2 | utente2 | Nessuna | Associa Opera |
| | | | | Degrada |
| Trascrizione | ciro | immobile | Divina commedia | Degrada |









In mostra 1 fino a 5 di 5 totali

Precedente 1 Successivo

Utenza

10 righe per pagina

Ricerca:

| Nome | Cognome | Email | Promozione |
|---------|---------|--------------------|---|
| val | pic | t@t.it |     |
| utente4 | utente4 | utente4@utente4.it |     |

Il metodo del quale vedremo il sequence diagram è associa_opera_utente (opera,utente,gruppo)

