

# DOCUMENTAZIONE

**Object Oriented Programming**

***Progetto biblioteca digitale***

Marco D’Ettorre 222737

Francesco Proietti 238047

BIBLIOTECA DIGITALE 4

1 Requisiti 5

1.1 Requisiti funzionali 5

1.2 Requisiti non funzionali 8

2 System design 9

2.1 Component diagram 9

2.2 Sequence diagram 10

2.2.1 Sequence trascrizione 10

2.2.2 Sequence verifica trascrizione 12

2.3 Deploymentdiagram 13

2.4 Schema entità relazioni 14

2.4.1 Entità 14

2.4.2 Relazioni 14

2.4.3 Diagramma ER 15

2.5 Design decisions 16

2.6 Problemi noti 18

2.7 Design patterns 18

2.7.1 Model-View-Controller 19

2.7.2 Data-Access-Object 20

3 Software design 21

3.1 Class diagram 21

3.1.1 Package Controller 21

3.1.2 Package Framework 22

3.1.2.1 Sottopackage result 23

3.1.2.2 Sottopackage util 23

3.1.2.3 Sottopackage data 23

3.1.3 Package data 24

3.2 Class diagram 26

3.3 Problemi noti 27

4 Screenshots 27

4.1 Homepage 27

4.2 Registrazione 28

4.3 Accesso acquisitore 29

4.4 Accesso revisore acquisizione 29

4.5 Trascrizione 31

4.6 Revisione trascrizione 32

4.7 Accesso admin 33

4.8 Accesso da dispositivo mobile 34

# BIBLIOTECA DIGITALE

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.

Il progetto è presente al seguente URL:

**https://github.com/marcodettorre/Biblioteca-Digitale**

ed è strutturato come segue:

* *src*: contiene il codice sorgente del progetto
* *doc*: contiene i documenti di design del progetto
* *javadoc*: contiene il codice javadoc del progetto

# Requisiti

## Requisiti funzionali

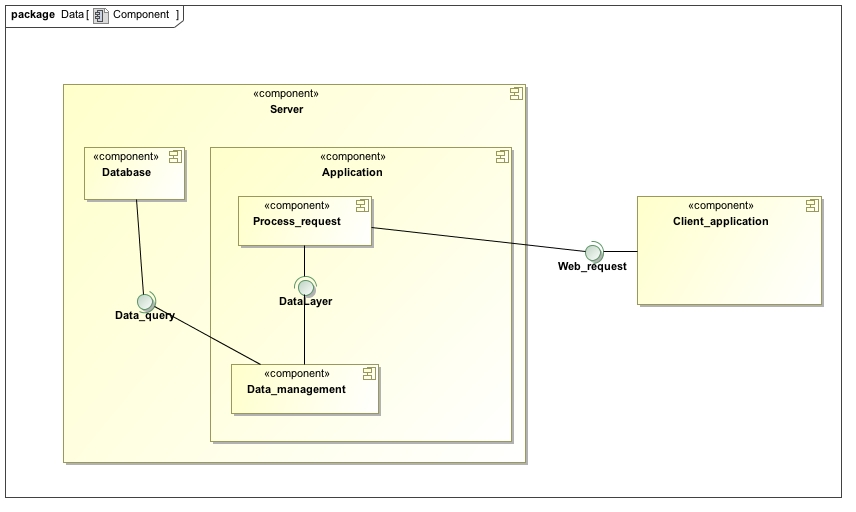
| CODICE | TIPO | DESCRIZIONE |
| --- | --- | --- |
|  | RUOLI | I ruoli identificati all’interno dell’applicazione sono: revisore acquisizione, revisore trascrizione, acquisitore e trascrittore agli utenti registrati. |
|  | ADMIN | L’amministratore può attribuire i vari ruoli agli utenti registrati. |
|  | ADMIN | L’amministratore può rimuovere gli utenti. |
|  | ADMIN | L’amministratore può inserire una nuova opera nel sistema. |
|  | UTENTE | Un utente non loggato al sistema viene identificato come utente base. |
| 1. R | UTENTE | L’utente base può visualizzare solo l’elenco delle opere con i loro attributi. |
| 1. R | UTENTE | L’utente base può registrarsi al sistema tramite una form indicando: nome, cognome, username, email e password, diventando utente avanzato. |
| 1. R | UTENTE | L’utente avanzato può consultare completamente le opere pubblicate con le immagini, seguite eventualmente, dalla trascrizione correlata. |
|  |  |  |
|  |  |  |
|  | UTENTE | Gli utenti possono cercare le opere in base ai seguenti criteri: titolo, autore, editore, anno di pubblicazione e lingua, da ricercare in AND. |
|  | ACQUISIZIONE | L’acquisitore inizia la procedura di acquisizione di una nuova opera, facendo l’upload di un immagine scannerizzata relativa ad una sua pagina e il cui processo di acquisizione non è stato iniziato da un altro. |
| 1. R | ACQUISIZIONE | L’immagine scannerizzata deve essere validata successivamente, da un revisore di acquisizione che verifica la correttezza dell’acquisizione. |
| 1. R | ACQUISIZIONE | L’immagine che passa il controllo del revisore diventerà immagine validata. |
|  | ACQUISIZIONE | Le opere che hanno tutte le immagini validate risultano “acquisite”. |
|  | ACQUISIZIONE | L’opera che non ha tutte le immagini validate è un’opera in “revisione acquisizione”. |
| 1. R | ACQUISIZIONE | La pubblicazione delle opere acquisite è riservata all’utente admin. |
|  | ACQUISIZIONE | Il “revisore acquisizione” avrà a disposizione un catalogo di opere che sono in “revisione acquisizione”. |
| 1. R | TRASCRIZIONE | Il trascrittore può trascrivere solo pagine di opere già acquisite e il cui processo di trascrizione non è stato iniziato da un altro. |
| 1. R | TRASCRIZIONE | Ad ogni pagina corrisponde la sua trascrizione. |
|  | TRASCRIZIONE | La trascrizione deve essere validata da un revisore di trascrizione. |
|  | TRASCRIZIONE | Un’opera che ha tutte le pagine trascritte e validate viene considerata “Trascritta”. |
|  | TRASCRIZIONE | L’opera acquisita che non ha tutte le trascrizioni validate, risulta in “revisione trascrizione”. |
| 1. R | TRASCRIZIONE | La pubblicazione delle opere trascritte è riservata all’utente admin. |
|  | TRASCRIZIONE | Il “revisore trascrizioni” avrà accesso ad un catalogo di opere in revisione trascrizione. |
| 1. R | REVISORI | Il “revisore acquisizione” e il “revisore trascrizione”, possono convalidare o rigettare l’immagine o la trascrizione che stanno revisionando. |
|  | OPERE | Ogni opera è caratterizzata da: titolo, autore, editore, e metadati. |
|  | OPERE | I metadati consistono di: num. pagine, lingua, data di pubblicazione e descrizione. |

## Requisiti non funzionali

| CODICE | DESCRIZIONE |
| --- | --- |
| RNF1 | La trascrizione deve essere in formato TEI |
| RNF2 | Il trascrittore e il revisore trascrizione conoscono lo standard TEI |
| RNF3 | il progetto deve essere scritto utilizzando il linguaggio Java |
| RNF4 | Il sistema deve garantire 50 connessioni al database in contemporanea |
| RNF5 | L'interfaccia del client deve essere responsive e quindi fruibile da device mobili e computer desktop |

# System design

## Component diagram



L’architettura è stata decomposta in 4 componenti principali che sono:

**Database**: unità che gestisce la memorizzazione dei dati.

**Data\_management**: componente dell’applicazione che interagisce con il database per la gestione dei dati.

**Process\_request**: componente dell’applicazione per la gestione delle richieste web.

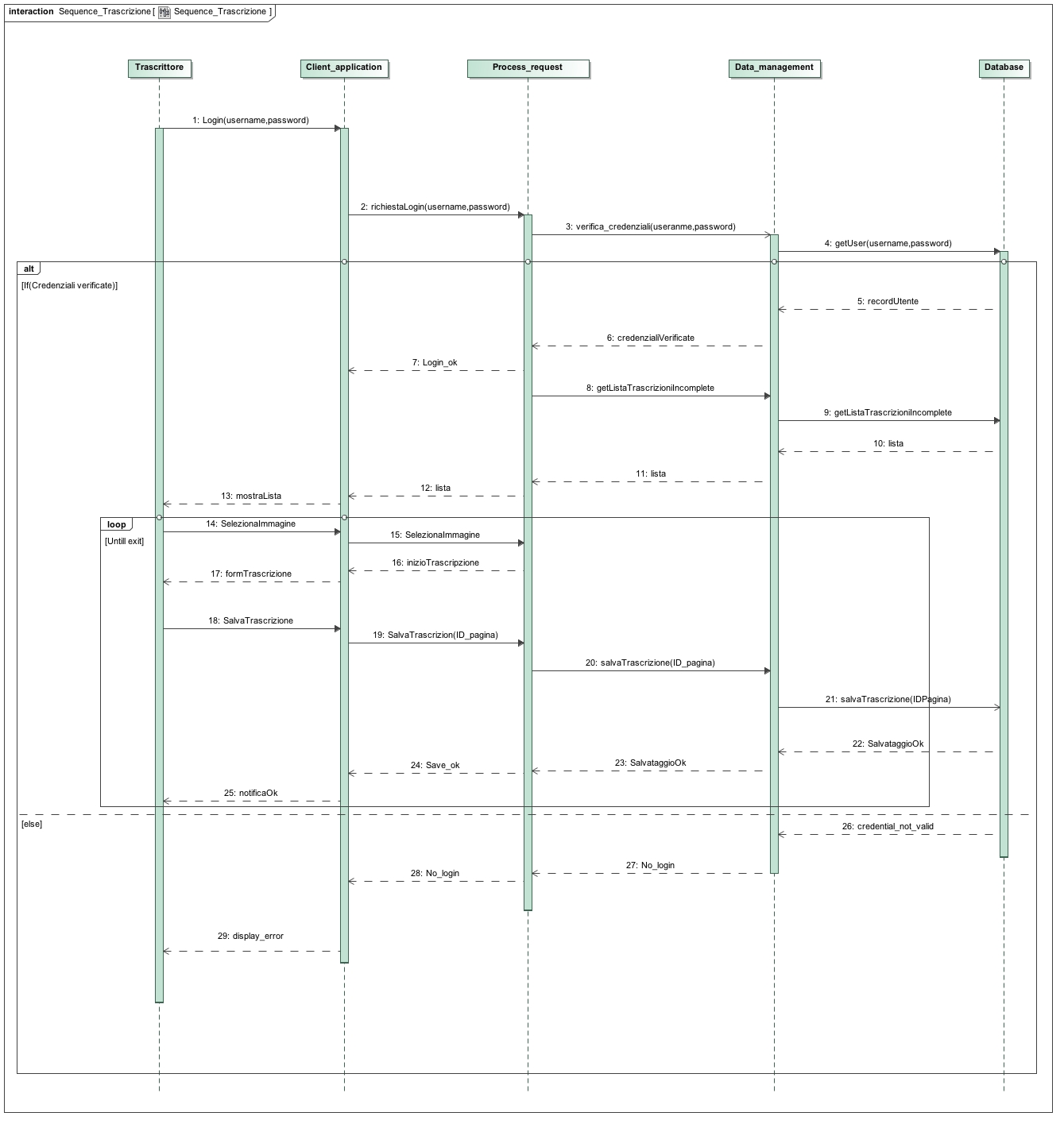
**Client\_application**: web browser per l’interazione con l’applicazione.

## Sequence diagram

### Sequence trascrizione

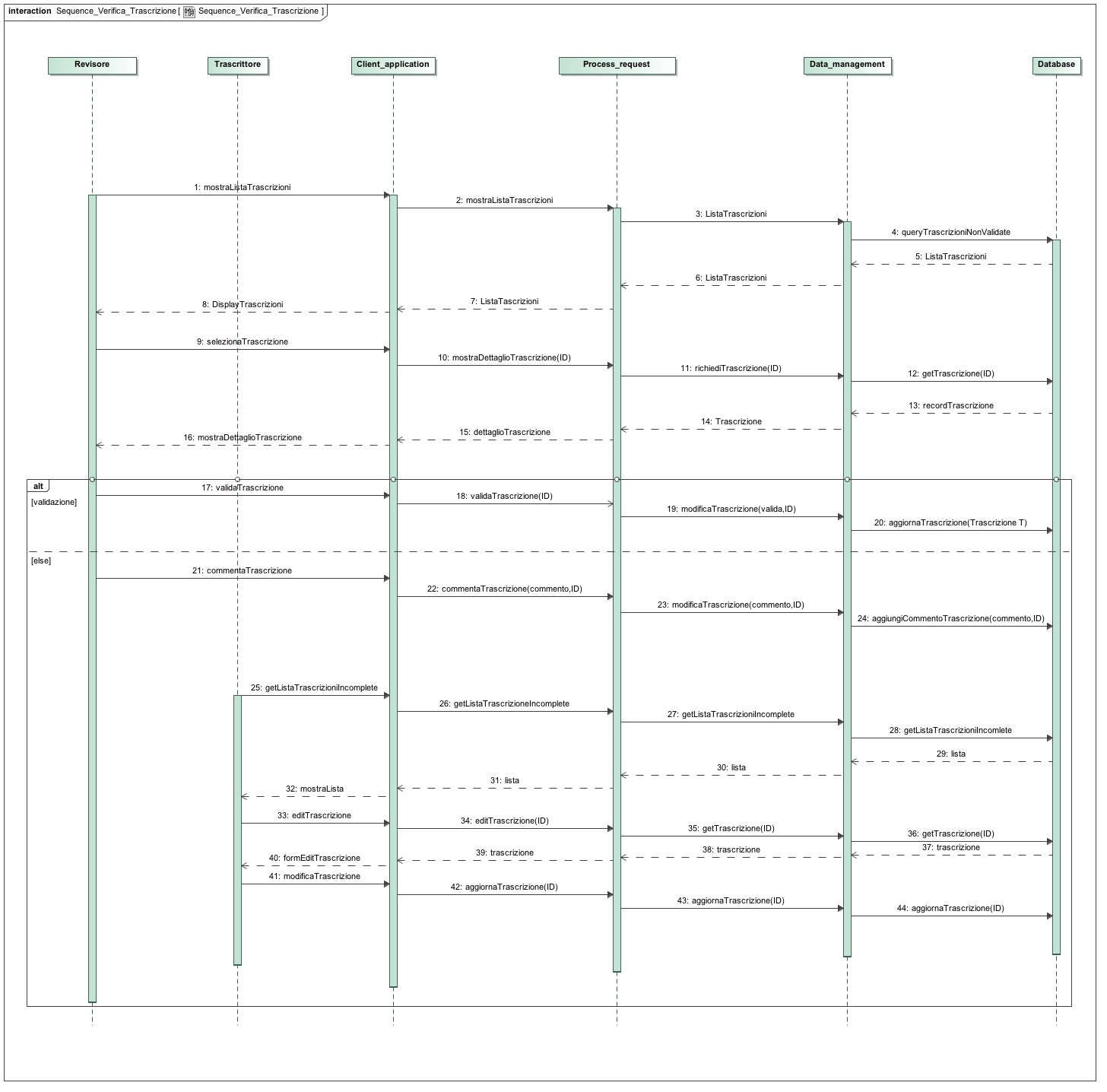
Nel seguente schema viene esplicitato il processo con cui un utente “trascrittore” si autentica nel sistema ed effettua una trascrizione.

Il sistema, una volta verificate le credenziali di accesso, visualizza direttamente all’utente la pagina contenente le opereche hanno immagini senza una trascrizione. L’utente, scegliendo un’opera, potrà scegliere l’immagine da trascrivere dall’elenco delle pagine.

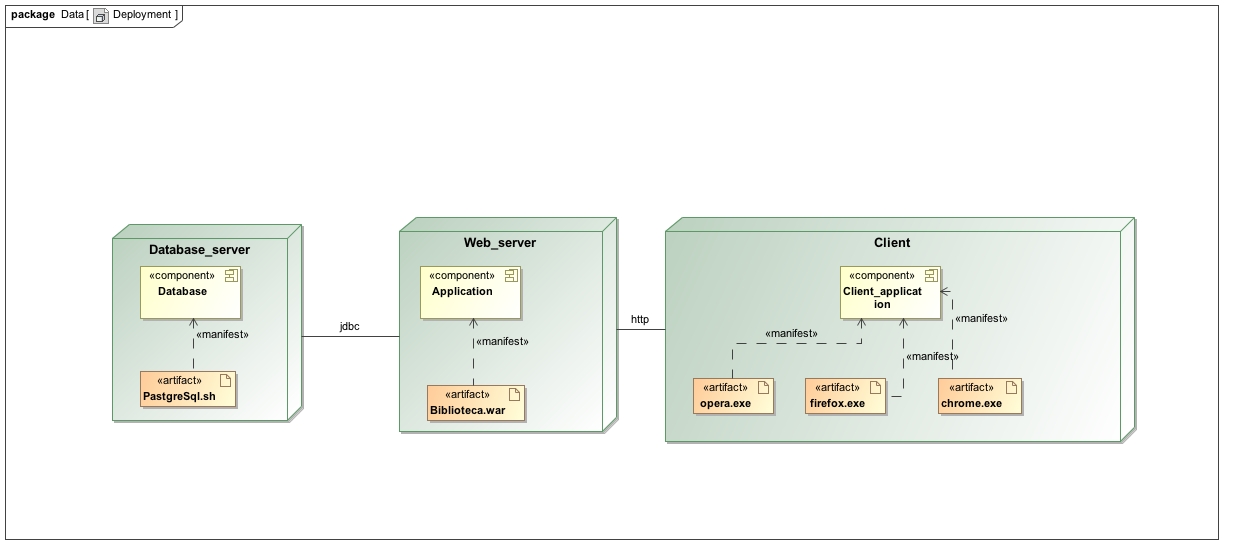


### Sequence verifica trascrizione

Nel seguente diagramma viene descritto il processo con cui una trascrizione viene validata da un “revisore trascrizione”.



## Deploymentdiagram



Le componenti logiche possono essere distribuite in 3 dispositivi diversi:

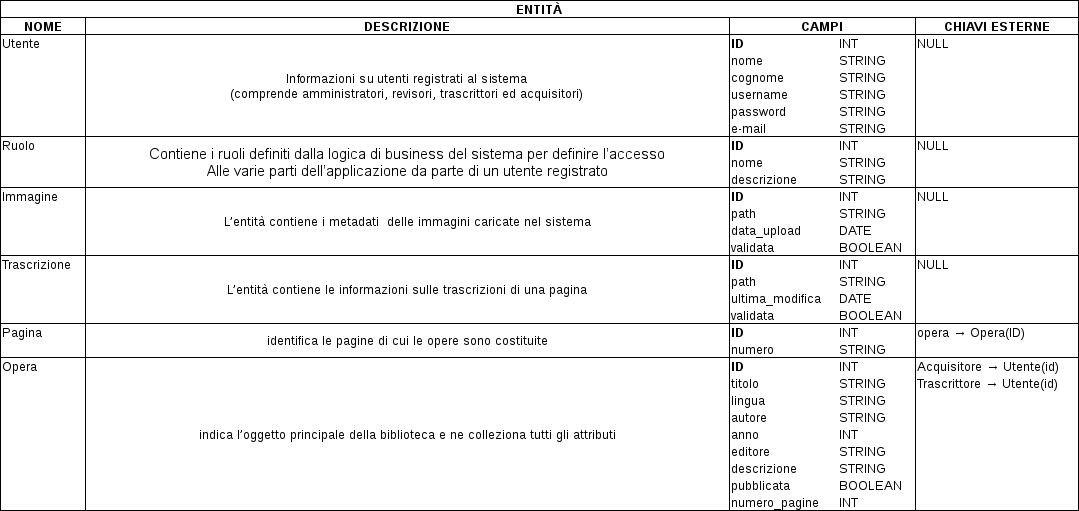
* **Database\_server**: sul quale è installato il DBMS PostgreSQL che implementa la componente Database. Si è scelto di mettere questa componente separata dal server, pensando di poterla utilizzare in maniera distribuita in modo da non sovraccaricare il workload dei vari componenti, garantendo così un servizio più fruibile.

In un lavoro futuro è possibile anche utilizzare un database di tipo cloud.

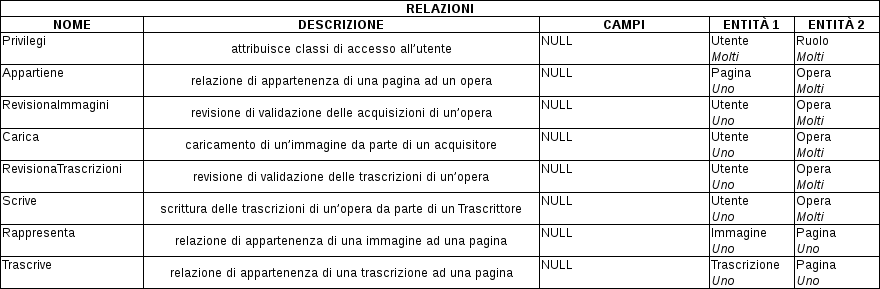
* **Web\_server**: la componente centrale sulla quale è deployata la parte logica dell’applicazione. Fornisce accesso al database e interfaccia per le richieste del client.
* **Client**: workstation o device mobile da cui poter accedere al servizio tramite browser che realizzano la clientapplication.

## Schema entità relazioni

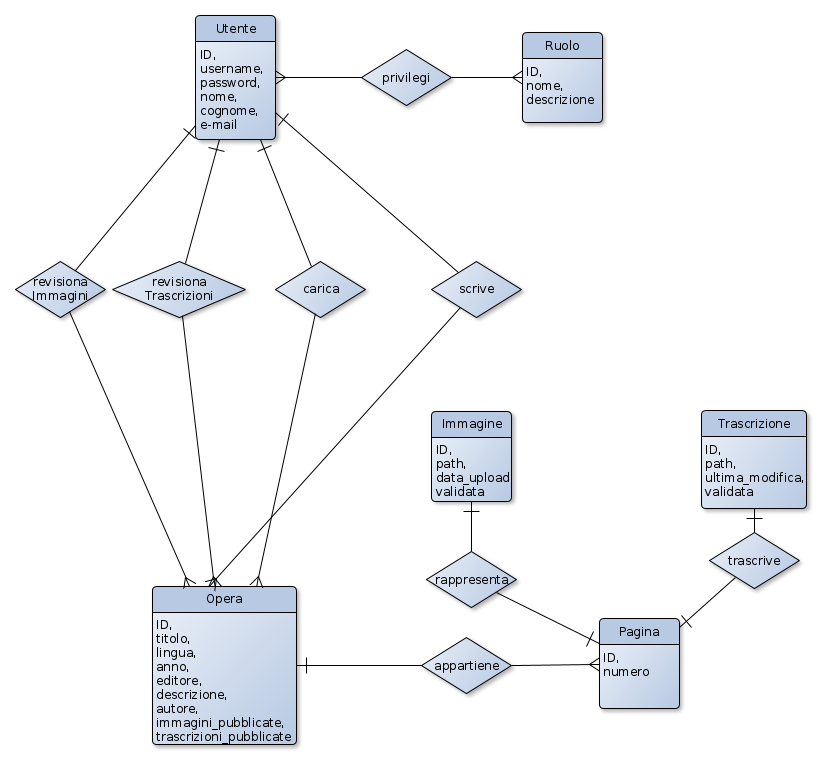
### Entità



### Relazioni



### Diagramma ER



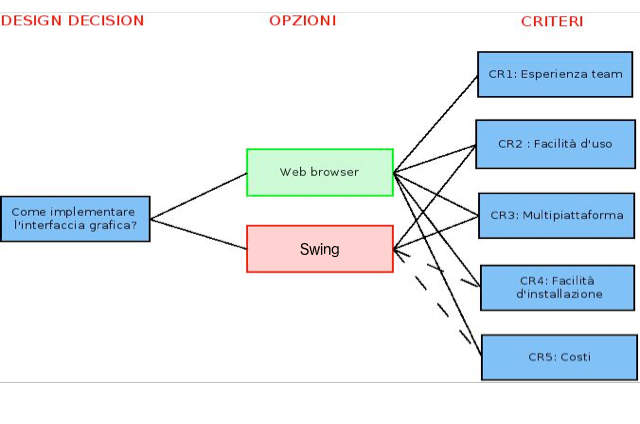
## Design decisions

Nell’ambito del progetto, tre sono state le scelte di design principali sulle quali il team si è soffermato: implementazione dell’interfaccia grafica, template engine ed editor TEI. Di seguito, una descrizione schematica del processo di decisione. Negli schemi, il box a sinistra rappresenta la domanda di design, i box centrali rappresentano le opzioni individuate ed i box a destra rappresentano i criteri di valutazione adottati. Con il colore verde viene evidenziato il box dell’opzione scelta.

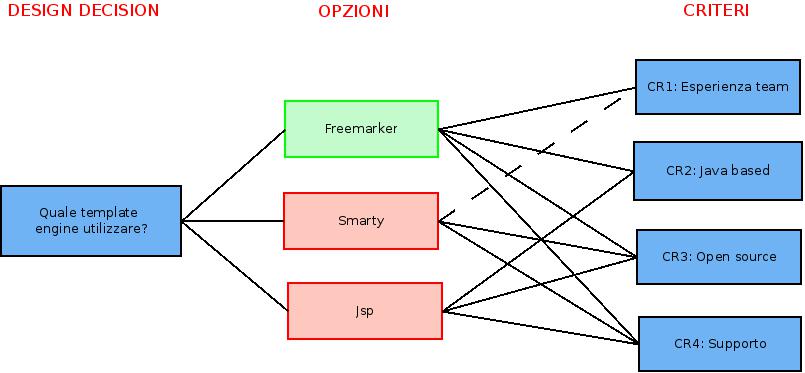
Inoltre, le frecce indicano come il criterio venga soddisfatto dall’opzione da cui la freccia parte: freccia piena -> soddisfatto pienamente, freccia tratteggiata -> soddisfatto parzialmente, nessuna freccia -> non soddisatto

1. Per quanto riguarda la scelta su “Come implementare l’interfaccia grafica”, è stato considerato il fatto che in un’applicazione web based non è necessario installare alcun software sul client, differentemente dall’interfaccia swing, rendendo l’utilizzo del sistema immediato a tutti.

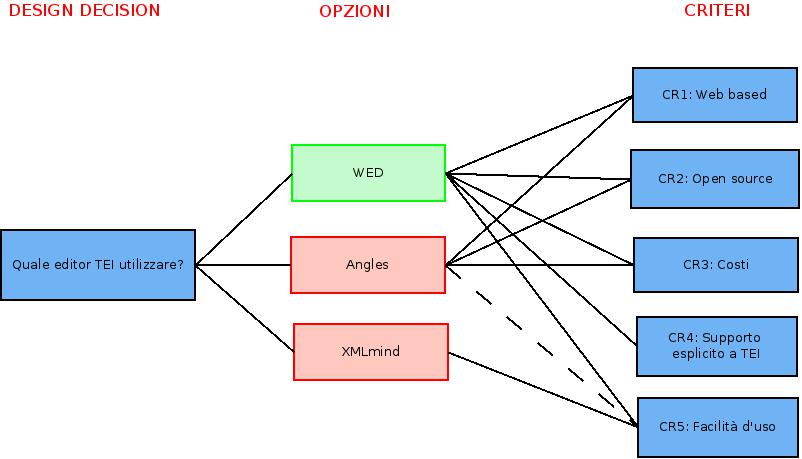
Inoltre, un’applicazione web rientra fra l’expertise del team, è multipiattaforma e ci permette di applicare pattern ben noti nelle applicazioni web.



1. Fra le varie opzioni per quanto riguarda “Quale templateengine utilizzare?”, la scelta ricadeva in particolare tra Freemarker e JSP. Il criterio che ha determinato la scelta difreemarker è il fatto che questo tool risulta già famigliare al team di sviluppo.



1. Per quanto riguarda la scelta su “quale editor TEI utilizzare”, si è preso in considerazione il fatto che possa essere facilmente integrabile all’interno del browser (importando una libreria javascript) e la completezza del supporto dell’editor rispetto allo standard TEI.



## Problemi noti

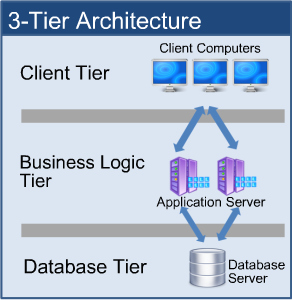
Nella fase implementativa si sono trovate grosse difficoltà nell’integrare gli editor TEI considerati nelle decisioni.

Il team ha dunque valutato un quarto editor (TEA), contattando personalmente anche gli sviluppatori per riuscire nell’intento di integrarlo nell’applicazione. Tuttavia, il progetto TEA è ancora in una fase di sviluppo e non ancora pronto per l’utilizzo all’interno di un’applicazione con TEI.

Si è quindi scelto di implementare un parser TEI in grado di riconoscere i tag dello standard e un’area di testo per l’immissione del contenuto.

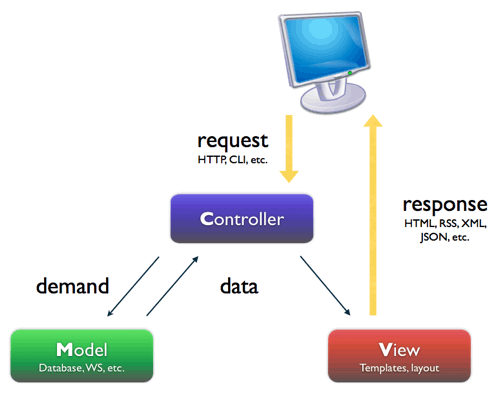
## Design patterns

L’architettura adottata è un’architettura 3-tier classica con il livello di presentazione, di logica applicativa e di accesso ai dati. Per realizzarla sono stati adottati i design pattern MVC e DAO.



### Model-View-Controller

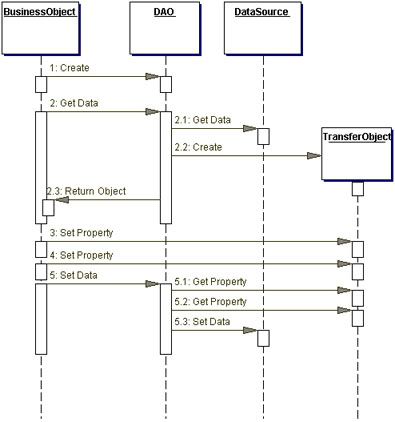
Il pattern MVC ci permette di organizzare in modo snello ed adattabile il livello di presentazione andando a definire una parte MODEL che si occupa solo della gestione dei dati del dominio, una parte VIEW che viene implementata dalle pagine web e definisce diverse rappresentazioni di output di informazioni ed infine una parte CONTROLLER che accetta input e li converte in comandi per il MODEL e/o per la VISTA.



### Data-Access-Object

Per la gestione della persistenza dei dati si è invece adottato il pattern Data-Access-Object che fa parte dei Core J2EE Patterns. Questo pattern fornisce operazioni sui dati senza esporre dettagli della base di dati mappando chiamate dell’applicazione al livello di persistenza. Questo ci permette di realizzare l’information hiding dei dettagli della memorizzazione dei dati. Infatti, con l’introduzione dell’interfaccia DAO, i cambiamenti alla logica di business potranno fare comunque affidamento sulla stessa interfaccia DAO, mentre cambiamenti alla logica di persistenza non coinvolgeranno i client dell’interfaccia se questa rimarrà correttamente implementata.

Nello schema seguente una sintesi del processo di accesso ai dati tramite DAO.



Il BusinessObject rappresenta il client che richiede accesso al database per memorizzare e ottenere dati. Il DataAccessObject astrae l’implementazione dell’accesso ai dati per garantire accesso trasparente dal BusinessObject. DataSource rappresenta il database o altra forma di sorgente dati (e.g. file system, XML repository). Il TransferObject rappresenta un oggetto usato come contenitore di dati. Esso viene usato dal DataAccessObject per ritornare dati al client o per aggiornare dati nel DataSource.

# Software design

## Class diagram

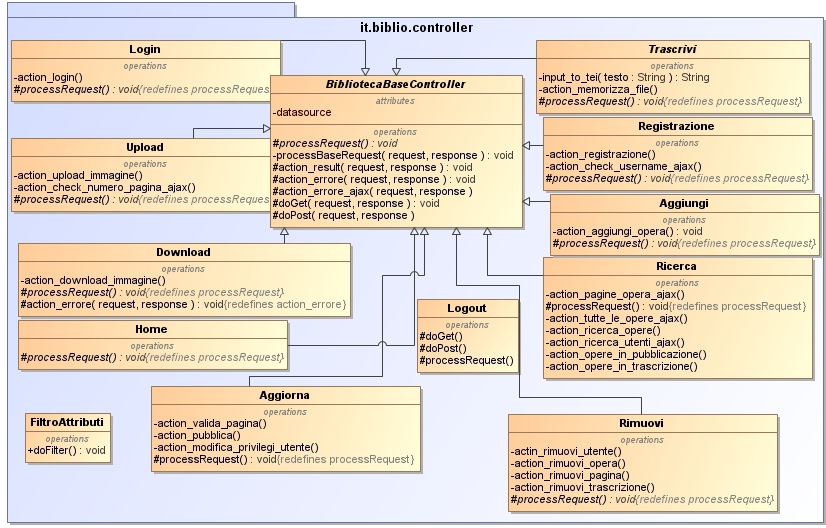
### Package Controller

Questo package rappresenta la componente Controller del pattern MVC. Esso contiene infatti tutte le Servlet, ovvero il codice che può essere direttamente chiamato dal browser, che eseguono le funzionalità di business dell’applicazione.

La classe principale è la classe astratta **BibliotecaBaseController**, che viene estesa da tutte le servlet che abbiano la necessità di accedere alla base di dati. Infatti, tale classe oltre ad estendere la classe HttpServlet del core J2EE ed implementarne alcuni metodi ereditati fondamentali (doGet, doPost), possiede la reference alla risorsa di tipo database definita nel deployment descriptor dell’applicazione ed effettua l’inizializzazione della connessione al database rendendola disponibile come attributo della HttpServletRequest alle classi che la estendono. Essa definisce inoltre un metodo standard per la renderizzazione di una pagina di errore ed anche un metodo per la generazione automatica della pagina di risposta. Il metodo *processRequest* è astratto e viene ridefinito dalle classi figlie come punto di accesso per iniziare la computazione specifica.

La classe **Logout** è l’unica a non estendere BibliotecaBaseController in quanto altro non fa che invalidare la sessione. La classe astratta si serve delle classi del package framework.result per la generazione delle pagine dinamiche e di conseguenza anche le classi figlie. Infine, è stata definita anche una classe filtro (**FiltroAttributi**) in ingresso che analizza tutte le richieste ed inizializza alcune variabili usate dalle servlet.

Le Servlet hanno un metodo privato per ogni azione che sono chiamate a fare dal client. La logica di accesso a tali operazioni viene implementanta al loro interno.



### Package Framework

Il package rappresenta la base dello sviluppo di un’applicazione WEB che utilizza il pattern MVC. Infatti, si potranno usare le classi del package result per la generazione della grafica (VIEW), le classi del package data come base per gli oggetti DAO (MODEL) e le classi del package util per operazioni comuni a tutta l’applicazione.



#### Sottopackage result

Tale package dispone di tutte le classi che si interfacciano direttamente con la libreria FreeMarker per la gestione dei template. Esse mettono a disposizione del controller, dunque, funzioni per la generazione delle pagine dinamiche.

La classe **TemplateResult** fornisce funzioni per la generazione delle pagine HTML.

La classe **FailureResult** fornisce la funzione per la generazione della pagine d’errore dinamica.

La classe **StreamResult** fornisce funzioni per l’invio di file e flussi di byte in generale.

La classe **SplitSlashesFmkExt** estende le funzionalità del template manager FreeMarker per poter richiamare la funzione di logica di rappresentazione nella quale vengono tolti i caratteri di slash dalle stringhe direttamente nel codice HTML del template.

Infine, è stata definita l’eccezione **TemplateManagerException** generata ogniqualvolta si riscontra un errore nella logica del template.

#### Sottopackage util

Tale package dispone delle classi di utilità che vengono utilizzate trasversalmente da tutta l’applicazione. Le seguenti sono state individuate come classi di utilità:

**SecurityLayer**: classe che fornisce funzioni per la sicurezza dell’applicazione come la gestione delle sessioni e prevenzione da SQL Injection.

**PaserTEI**: classe usata per analizzare testi TEI.

#### Sottopackage data

Tale package fornisce delle classi basilare da cui poter creare classi di tipo risorsa da usare per la gestione delle connessioni con il database.

L’interfaccia **DataLayer** definisce le operazioni base da effettuare per un oggetto di tipo AutoCloseable (come una risorsa).

La classe **DataLayerPgsqlImpl** realizza tale interfaccia.

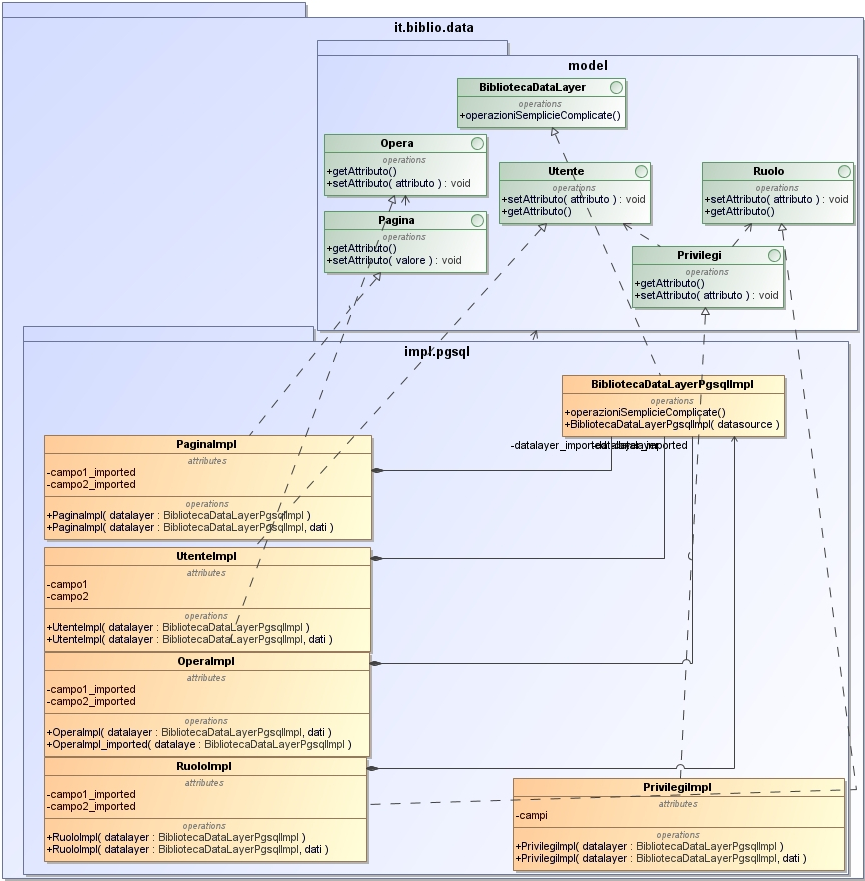
È stata inoltre definita l’eccezione DataLayerException da sollevare ogni volta che ci saranno errori nella gestione del livello dei dati.

### Package data

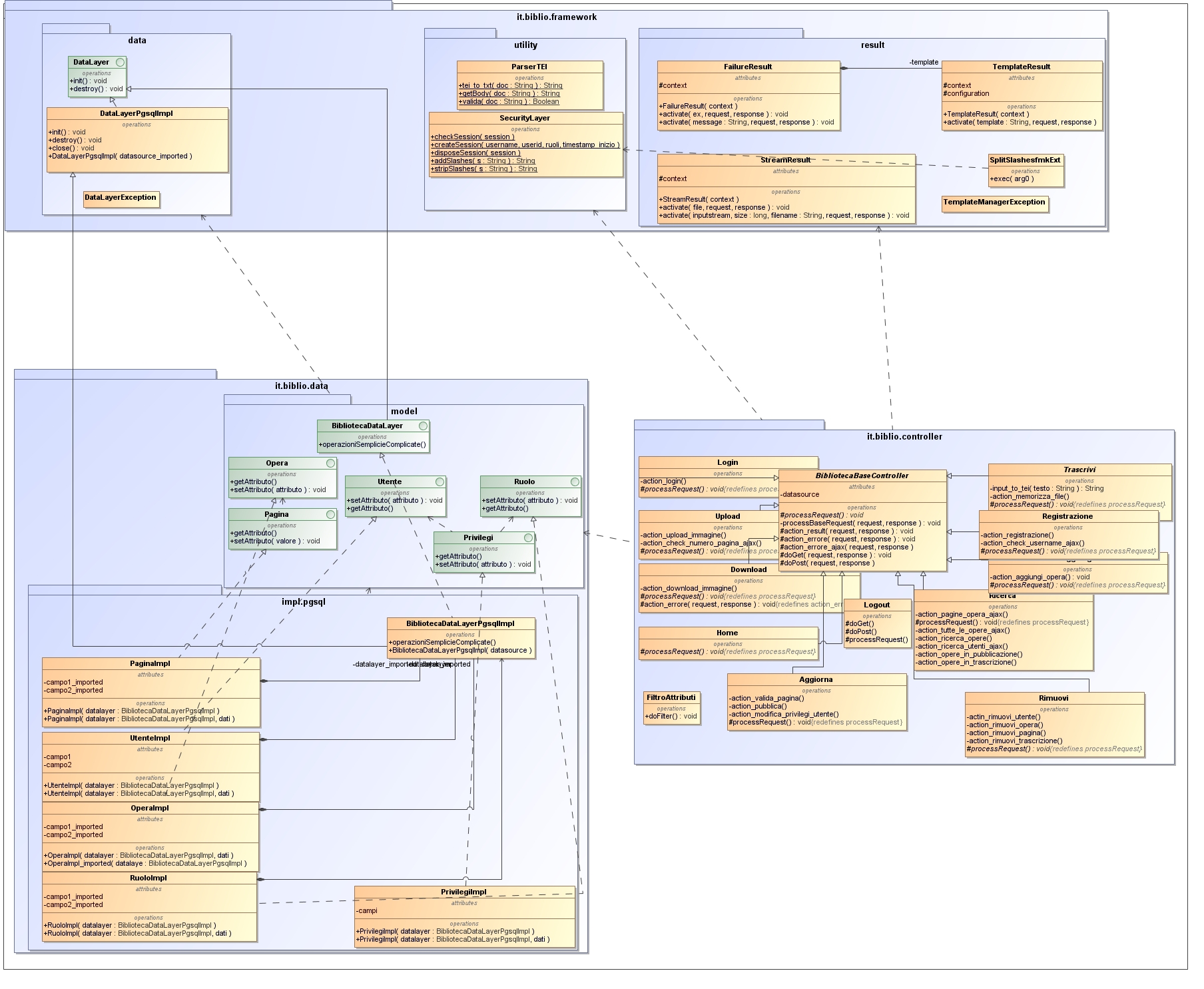
Tale package contiene tutte le interfacce e le classi che realizzano lo strato di persistenza dell’applicazione.

Le interfacce vengono posizionate nel sottopackage model mentre le realizzazioni sono in impl.pgsql. Infatti, le interfacce fungono da contratto verso i client (Controller) mentre l’implementazione può essere diversa per ogni tipo di database utilizzato. Infatti se volessimo un giorno utilizzare Mysql, basterà creare un sottopackage chiamato impl.mysql ed implementare le interfacce di model in maniera dedicata a quel DB senza dover cambiare le classi dell’applicazione che utilizzano il modello dei dati (o quantomeno cambiare solo l’invocazione del costruttore della classe che implementa BibliotecaDataLayer che nel nostro caso è una sola riga di codice posizionata di proposito in it.biblio.controller.BibliotecaBaseController).

Ad ogni interfaccia corrisponde un entità di business alla quale a sua volta corrisponde una tabella del database relazionale. I metodi get e set si riferiscono agli attributi delle Entità specificate nel diagramma Entità/Relazioni.



## Class diagram



Da questo diagramma si evince che le relazioni dei vari package rispettano il pattern MVC.

Il controller dipende solo dalle interfacce presenti in model per la gestione dei dati e non dipende dall’implementazione.

Le classi all’interno del package controller sfruttano inoltre quelle di framework.result e framework.util.

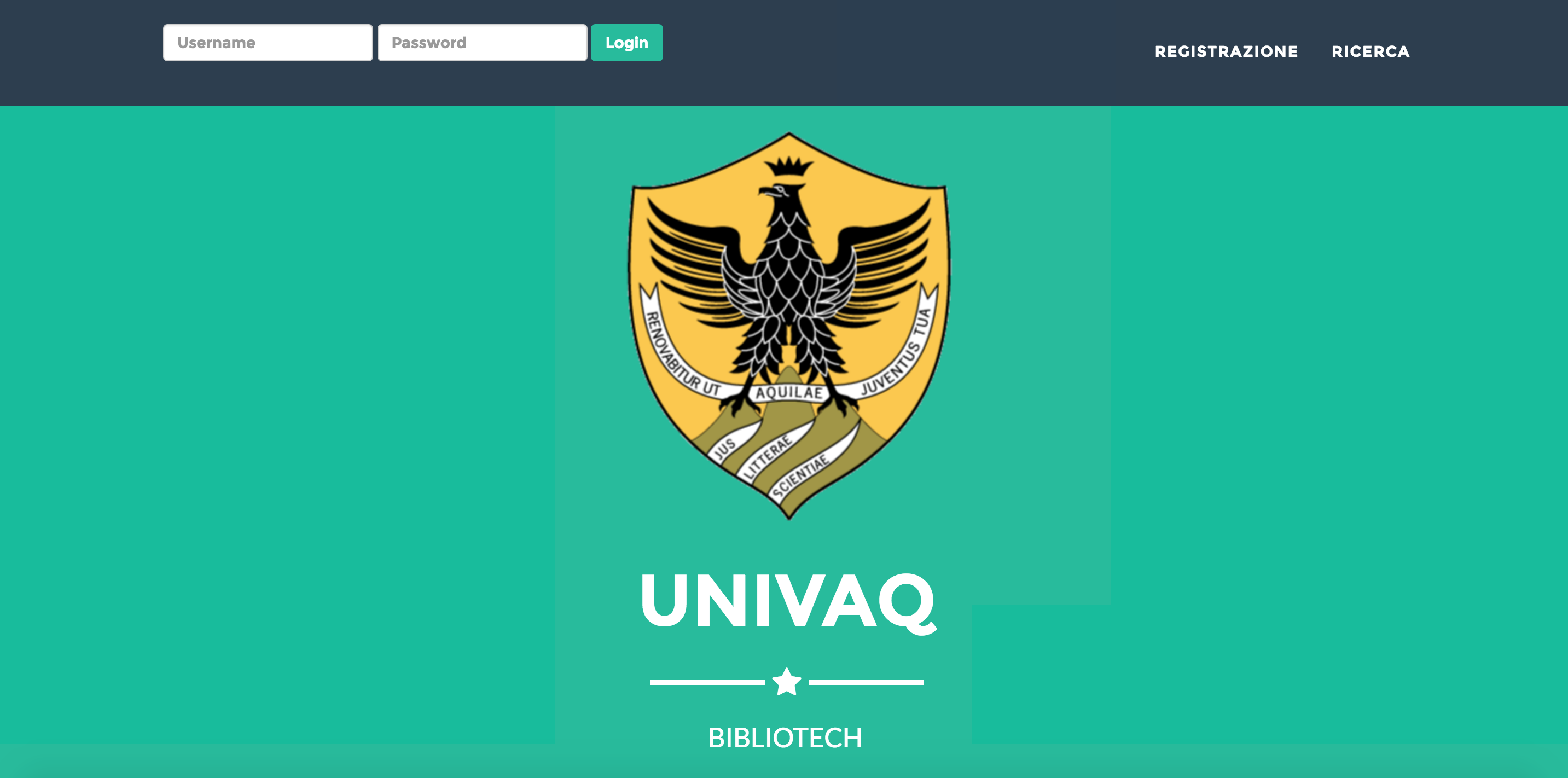
Il package it.biblio.data.model fa uso del framework.data per costruire il datalayer.

## Problemi noti

Poiché trattasi di oggetti POJO che mappano direttamente le entità del database e poiché si è limitati da una licenza demo di MagicDraw (il tool utilizzato per la modellazione) non sono stati esplicitati i campi e le operazioni get e set. Lo stesso vale per la classe BibliotecaDataLayerPgsqlImpl. Per questo può essere usata la documentazione generata da Javadoc nell’apposita cartella.

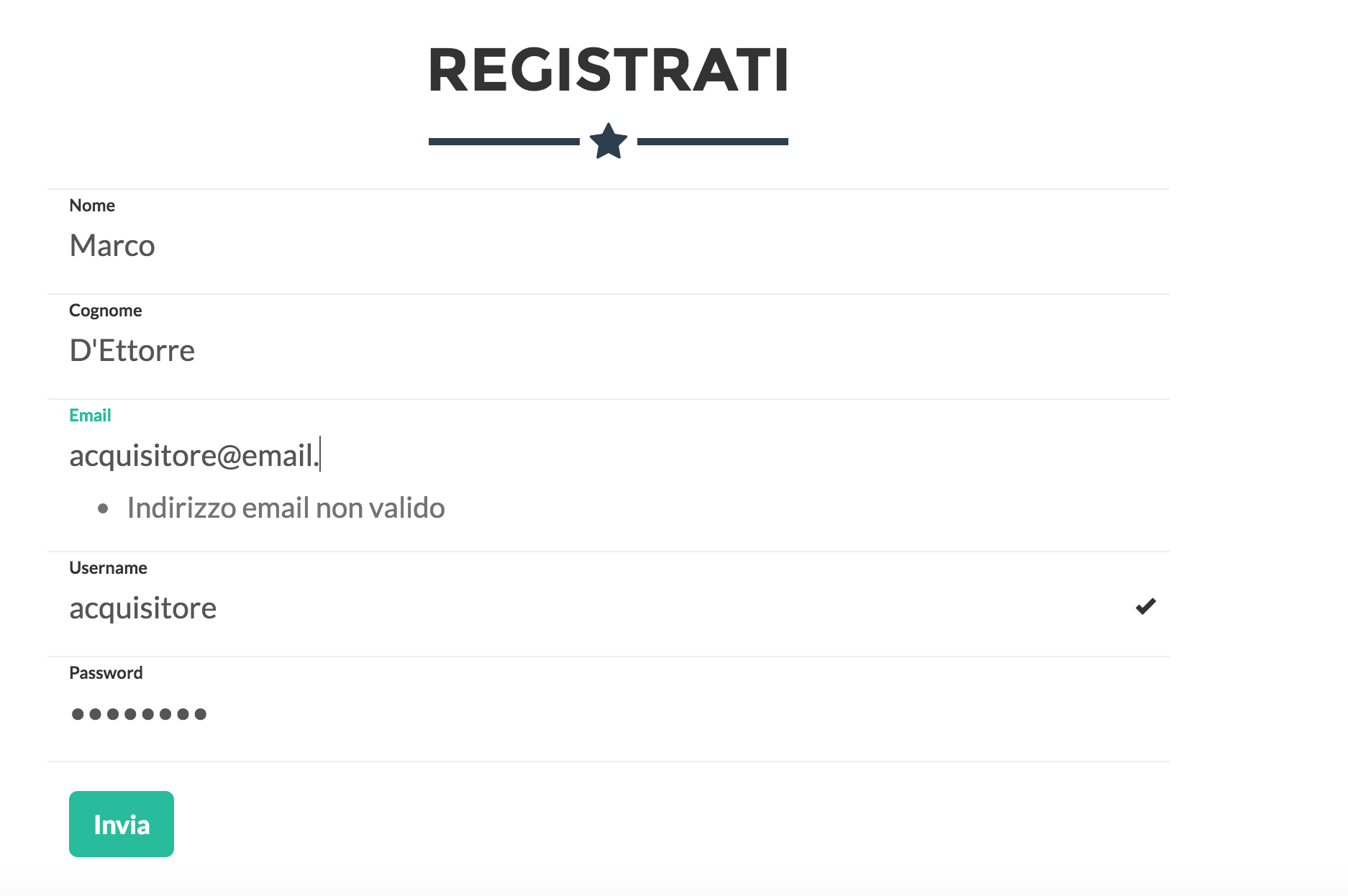
# Screenshots

## Homepage



Nella schermata homepage si trova la form per il login e scorrendo in basso o cliccando i link nella navbar, è possibile vedere la form per la registrazione e per la ricerca.

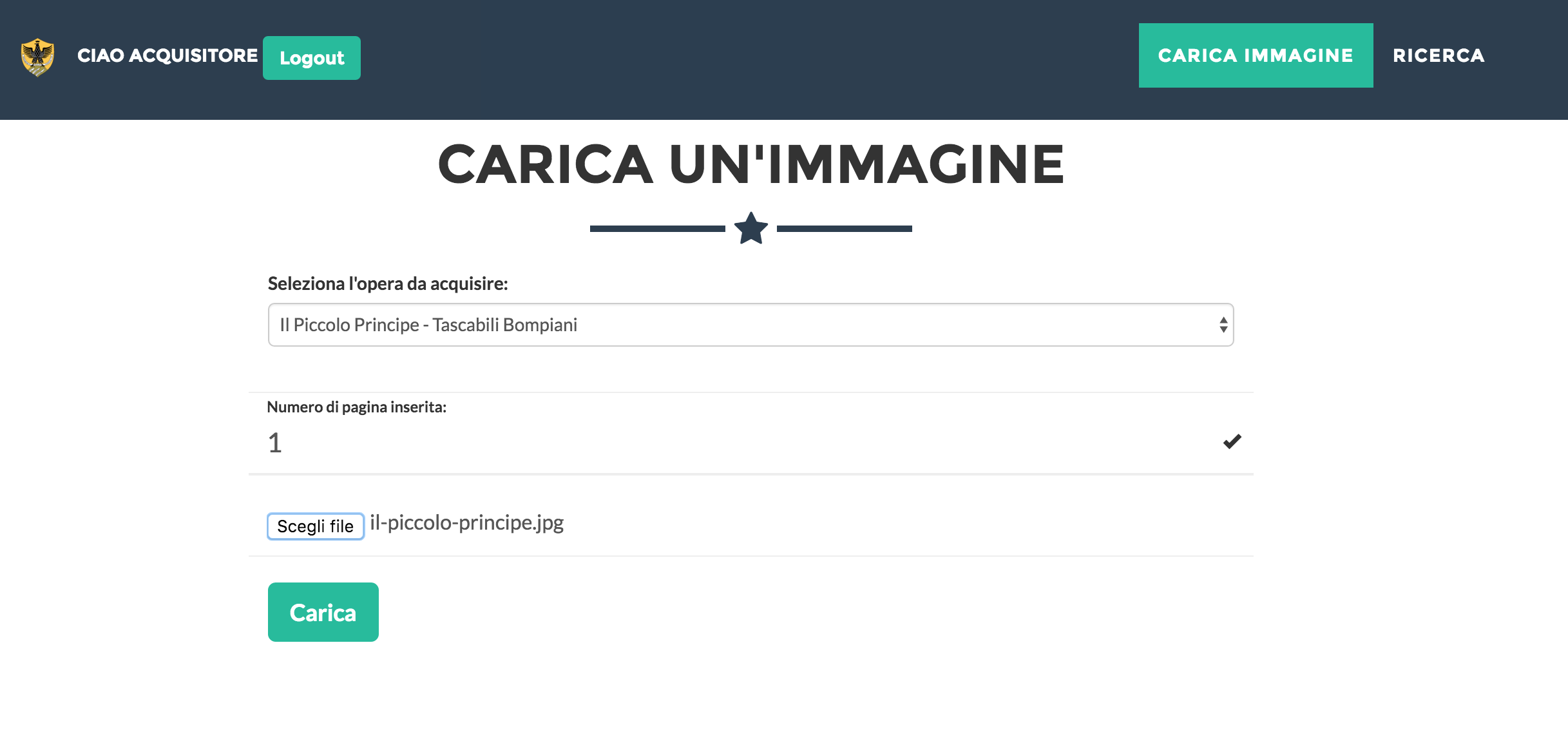
## Registrazione



Nella form registrazione vengono inseriti i campi relativi alle informazioni sull’utente con controlli sulla correttezza del campo email e controllo sullo username già presente.

Una volta registrati, viene aperta la pagina relativa all’utente base, dove è possibile cercare un’opera e sfogliarla completamente.

## Accesso acquisitore

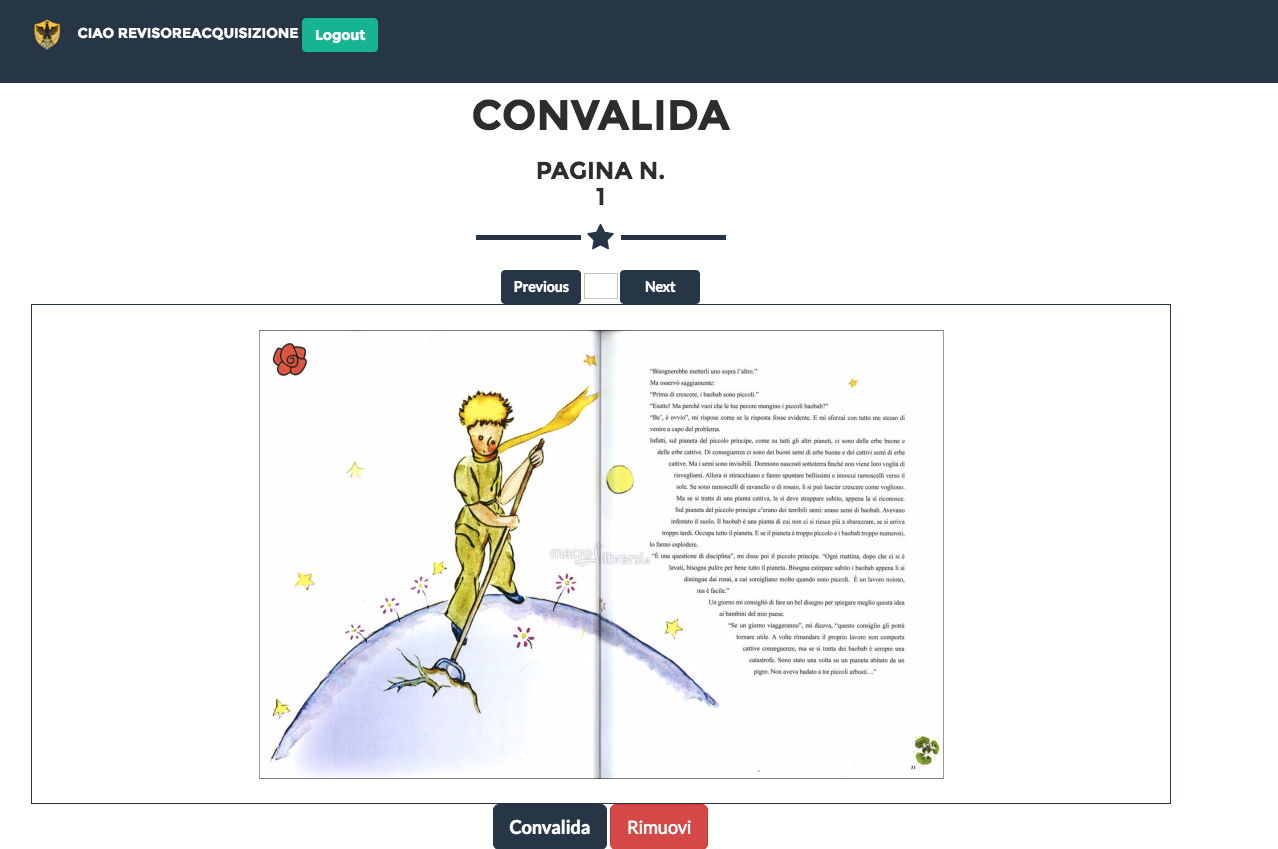


Il caricamento di un’immagine può essere effettuato solo da un acquisitore che tramite una select seleziona l’opera di cui vuole fare l’upload. Viene effettuato un controllo sul numero della pagina: se è già stata inserita non sarà possibile caricare l’immagine per quella pagina.

## Accesso revisore acquisizione



Seleziona un’opera dalla lista di quelle con acquisizione da convalidare



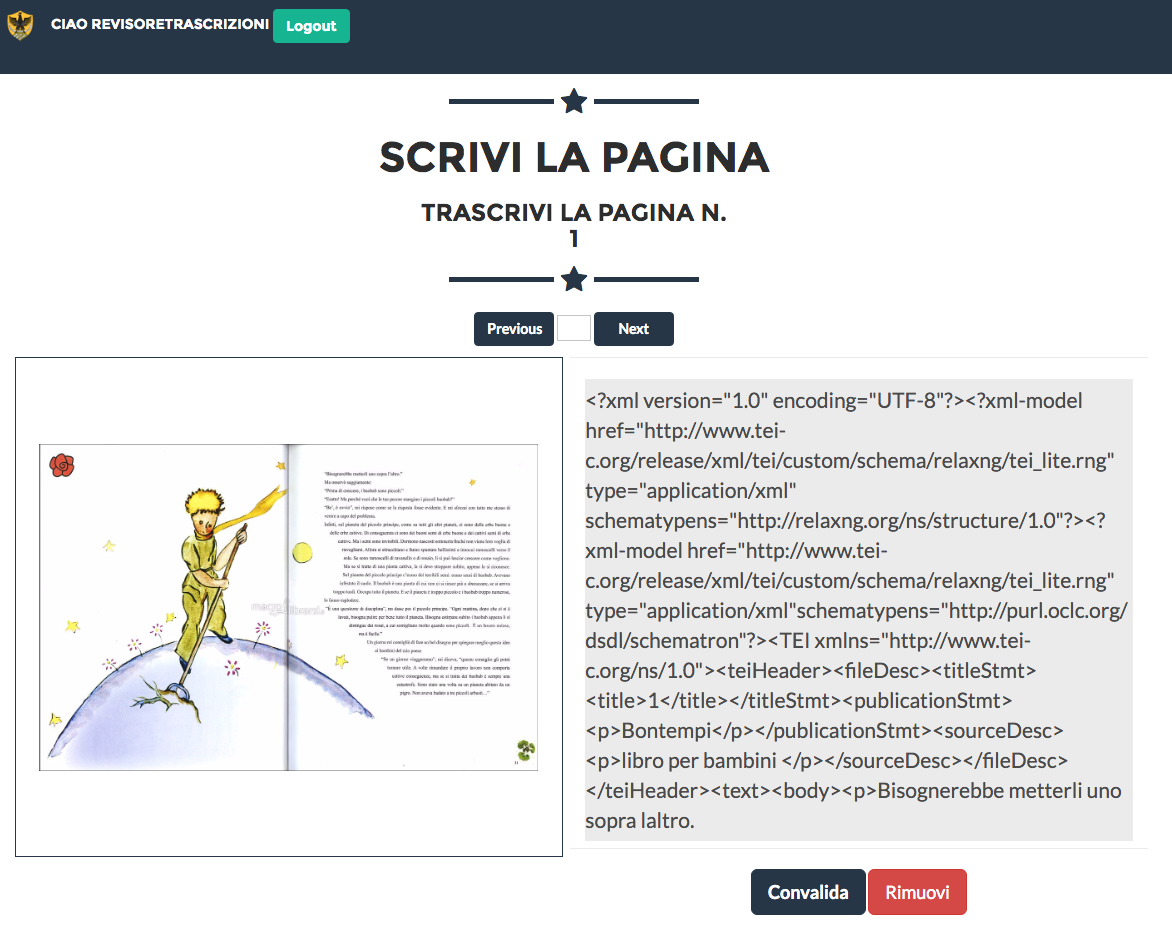
Il revisore acquisizione può sfogliare le varie pagine inserite e convalidare o non convalidare l’immagine. Se viene convalidata l’immagine non può essere più invalidata.

## Trascrizione



Il trascrittore una volta selezionata l’opera da trascrivere, potrà scorrere tra le pagine e riempire il campo testo con il codice TEI relativo solo al body. Una volta terminato il processo per ogni pagina, farà click sul bottone invia che salverà la trascrizione nel sistema.

## Revisione trascrizione



Il revisore trascrizione, una volta selezionata l’opera da validare, potrà scorrere le pagine e convalidare o rigettare la pagina già trascritta.

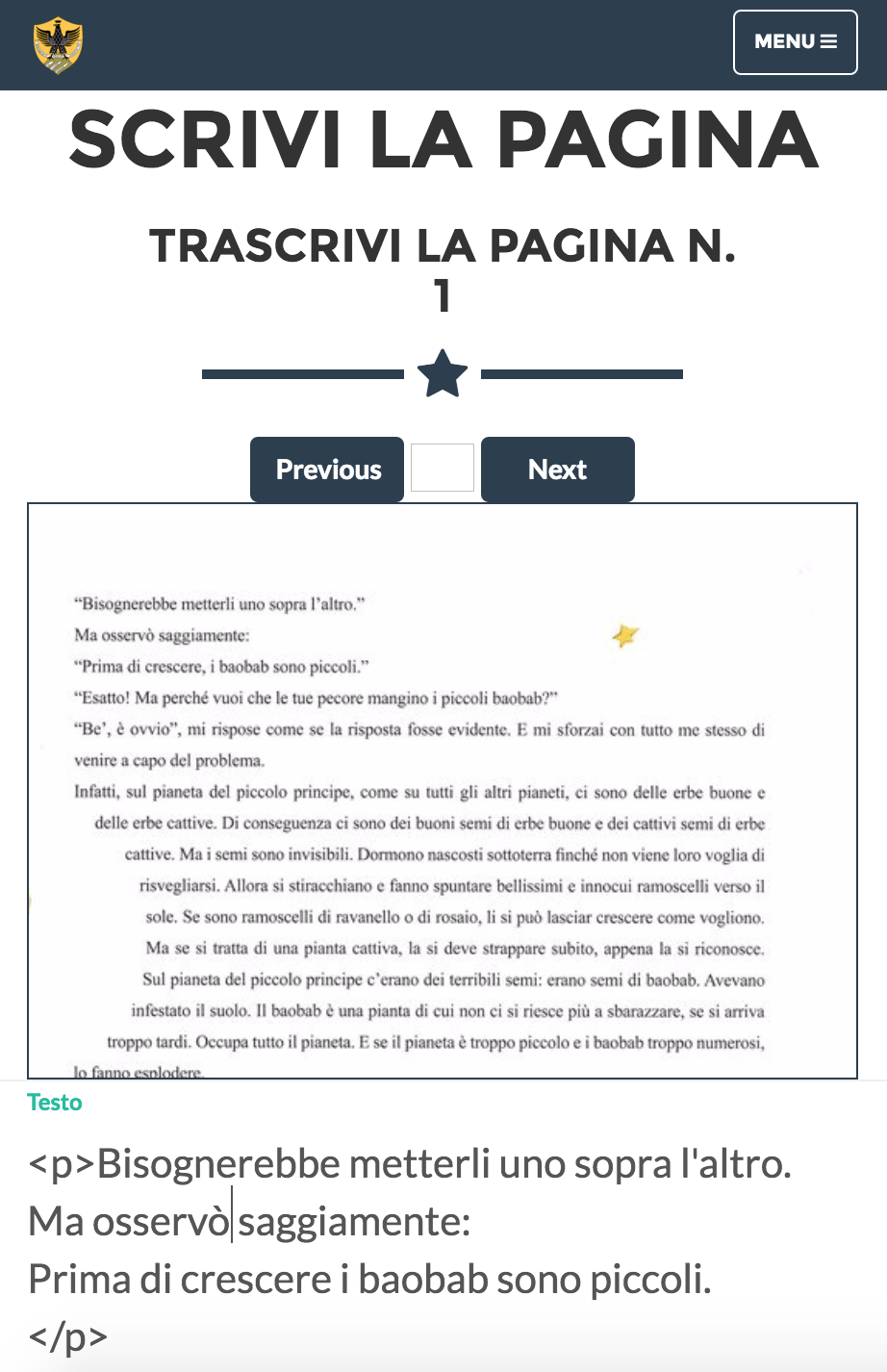
## Accesso admin



l’amministratore può rimuovere o pubblicare le opere in stato di “acquisite” o “trascritte” (come nello screenshot sopra), oppure può gestire gli utenti (screnshot sotto), inserire nuove opere e visualizzarne i dettagli.



## Accesso da dispositivo mobile

Tutte le funzionalità del sito sono accessibili anche da disposito mobile.