ODD: Object Design Document

EV English Validation

Riferimento	
Versione	0.1.1
Data	16/12/2018
Destinatario	Top Management
Presentato da	Alessandro Bacco, Ivan Buccella, Giuseppe Cirino, Alfonso Ingenito, Angelomaria Macellaro, Luigi Melchionno, Vincenzo Passariello
Approvato da	Giammaria Giordano, Valeria Pontillo

Revision History

Data	Versione	Descrizione	Autori
09/12/2018	0.0.1	Inserimento componenti off-the-shelf	Ivan Buccella
10/12/2018	0.0.2	Inserimento dei Design trade-offs	Ingenito Alfonso
10/12/2018	0.0.3	Inserimento Packages	Ivan Buccella
11/12/2018	0.0.4	Inserimento dei paragrafi 1.3.2, 1.3.3, 1.3.4, 1.3.5, 1.3.6	Giuseppe Cirino
12/12/2018	0.0.5	Completamento paragrafo 1.3	Luigi Melchionno
13/12/2018	0.0.6	Inserimento Class Interfaces	Luigi Melchionno
13/12/2018	0.0.7	Inserimento Class Diagram	Giuseppe Cirino
13/12/2018	0.0.8	Inserimento Glossario	Giuseppe Cirino
14/12/2018	0.0.9	Inserimento Design Pattern	Ingenito Alfonso
14/12/2018	0.1.0	Inserimento definizioni, acronimi e abbreviazioni	Luigi Melchionno
15/12/2018	0.1.1	Inserimento descrizioni packages	Giuseppe Cirino, Angelo Macellaro, Luigi Melchionno

Sommario

1. Introduzione 1
1.1 Object Design trade-offs 1
1.2 Componenti off-the-shelf 1
1.3 Interface Documentation guidelines 2
1.3.1 Classi e interfacce Java 2
1.3.2 Pagine lato Server (JSP) 3
1.3.3 Pagine HTML 3
1.3.4 Script Javascript 4
1.3.5 Fogli di stile CSS 5
1.3.6 Database SQL 5
1.4 Design Pattern 6
1.4.1 MVC 6
1.4.2 Singleton 6
1.5 Definizioni, Acronimi e abbreviazioni 8
2. Packages 9
2.1 Interface 9
2.2 View 10
2.3 Model 11
2.4 Controller 12
3. Class interfaces 13
4. Class diagram 17
5. Glossario 18

1. Introduzione

1.1 Object Design trade-offs

Nella fase dell'Object Design sorgono diversi compromessi di progettazione ed è necessario stabilire quali punti rispettare e quali rendere opzionali. Per quanto riguarda la realizzazione del sistema sono stati individuati i seguenti trade-off:

Memoria vs Estensibilità: Il sistema deve permettere l'estensibilità a discapito della memoria utilizzata così da dare la possibilità al cliente di richiedere lo sviluppo di nuove funzionalità, dando meno importanza alla memoria utilizzata da queste ultime.

Tempo di risposta vs Affidabilità: Il sistema sarà implementato in modo tale da preferire l'affidabilità al tempo di risposta, garantendo un controllo più accurato dei dati in input a discapito del tempo di risposta del sistema.

Disponibilità vs Tolleranza ai guasti: Il sistema dovrà sempre essere disponibile all'utente in caso di errore durante l'uso di una funzionalità, anche al costo di rendere non disponibile quest'ultima per un lasso di tempo.

Criteri di manutenzione vs Criteri di performance: Il sistema sarà implementato preferendo la manutenibilità alla performance in modo da facilitare gli sviluppatori nel processo di aggiornamento del software a discapito delle performance del sistema.

Comprensibilità vs Tempo: Uno degli obiettivi dell'implementazione sarà quello di scrivere del codice che rispetti lo standard proposto da Google per la programmazione nel linguaggio Java, oltre all'uso di commenti sui metodi non usuali. Ciò favorisce anche la comprensibilità, agevolando il processo di mantenimento e di modifica del progetto anche per futuri sviluppatori che non hanno lavorato dall'inizio al progetto stesso. Questo vantaggio tuttavia comporta un incremento del tempo per lo sviluppo e la realizzazione dell'intero sistema, che però è ripagato da una maggiore manutenibilità e chiarezza dei contenuti implementativi.

1.2 Componenti off-the-shelf

Per il nostro sistema utilizzeremo componenti off-the-shelf, che sono componenti software già disponibili utilizzati per facilitare la creazione del software. Il framework che utilizzeremo per il comparto grafico è Bootstrap, un framework open source adatto a creare la grafica di siti web e web application. Questo contiene tools basati su HTML e CSS sia per la tipografia che per l'interfaccia così come estensioni opzionali realizzate con Javascript.

Per ottenere alcuni effetti visuali nonché velocizzare il sistema utilizzeremo inoltre JQuery e Javascript che permetteranno all'interfaccia di rispondere alle azioni dell'utente agilmente, e un miglioramento dell'esperienza con le tecniche basate su Ajax.

Per il lato grafico e html è stato deciso di prendere in uso il seguente template: https://drive.google.com/open?id=1jZkvdO1c47Uil3Rt0TeJUQTQk7r5Diw- - Tipologia 2

Tale template verrà opportunamente modificato ed integrato in base alle esigenze del nostro sistema. Il tempo di realizzazione del template, rispetto il tempo di modifica, risulterebbe notevolmente maggiore nonché poco efficiente.

Inoltre, verrà utilizzato per gestire le connessioni, un sistema di connection pool, fornito da tomcat, riferimenti: https://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html. Risulta senza alcun'ombra di dubbio, più efficiente e veloce utilizzare una componente preesistente e fornita da un ente riconosciuto.

1.3 Interface Documentation guidelines

Nell'implementazione del sistema, i programmatori dovranno attenersi alle linee guida di seguito definite.

1.3.1 Classi e interfacce Java

Nella scrittura di codice per le classi Java ci si atterrà allo standard Google Java (http://google.github.io/styleguide/javaguide.html#s4.6-whitespace) nella sua interezza. Tale standard fornisce delle regole da seguire ad esempio ogni metodo ed ogni file possono non essere preceduti da un commento. Potranno esserci, inoltre, commenti e giustificazioni in merito a particolari decisioni o calcoli. La convenzione utilizzata dai team member per quanto riguarda i nomi delle variabili, è la nota lowerCamelCase, che consiste nello scrivere parole composte o frasi unendo tutte le parole tra loro. Quando si codificano classi e interfacce Java, si dovrebbero rispettare le seguenti regole di formattazione:

1. Non inserire spazi tra il nome del metodo e la parentesi tonda "(" che apre la lista dei parametri.
2. La parentesi graffa aperta "{"si trova alla fine della stessa linea dell'istruzione di dichiarazione.
3. La parentesi graffa chiusa "}" inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia.

```
return () -> {
 while (condition()) {
  method();
 }
};

return new MyClass() {
 @Override public void method() {
  if (condition()) {
   try {
    something();
   } catch (ProblemException e) {
    recover();
   }
  } else if (otherCondition()) {
   somethingElse();
  } else {
   lastThing();
  }
 }
};
```

Nel caso di istruzioni semplici, ogni linea deve contenere al massimo una sola istruzione. Mentre nel caso di istruzioni composte vanno rispettate le seguenti regole:

1. Le istruzioni racchiuse all'interno di un blocco (esempio: for), devono essere indentate di un'unità all'interno dell'istruzione composta.
2. La parentesi di apertura del blocco deve trovarsi alla fine della riga dell'istruzione composta.
3. La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell'istruzione composta

4. Le istruzioni composte formate da un'unica istruzione devono essere racchiuse da parentesi.

I nomi di classe devono essere sostantivi, con lettere minuscole e, sia la prima lettera del nome della classe sia la prima lettera di ogni parola interna, deve essere maiuscola. I nomi delle classi dovrebbero essere semplici, descrittivi e che rispettino il dominio applicativo. Non dovrebbero essere usati underscore per legare nomi. I nomi dei metodi iniziano con una lettera minuscola (non sono consentiti caratteri speciali) e seguono la notazione a cammello. Dovranno essere semplici, descrittivi e che rispettino il dominio applicativo.

1.3.2 Pagine lato Server (JSP)

Le pagine JSP devono, quando eseguite, produrre un documento conforme allo standard HTML 5. Il codice Java delle pagine deve aderire alle convenzioni per la codifica in Java, con le seguenti puntualizzazioni:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;
2. Il tag di chiusura (%>) si trova all'inizio di una riga;
3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione (<%= %>):

```
<!- Accettabile ->
<% for (String par : paragraphs) {%>
<p class='item'><% out.print(par); %></p>
<% } %>

<!- Non Accettabile ->
<p class='item'><% List<String> paragraphs = getParagraphs();
out.print(paragraphs.get(i++));%></p>
```

Per le Servlet è necessario far terminare il nome della classe con il suffisso Servlet.

1.3.3 Pagine HTML

Le pagine HTML, sia in forma statica che dinamica, devono essere conformi allo standard HTML 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

1. Un'indentazione consiste in una tabulazione;
2. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
3. Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;
4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

```
<!- Accettabile ->
<div>
      <span>
            <ul>
                  <li>
                      Uno
                  </li>
                  <li>
                      Due
                  </li>
            </ul>
      </span>
</div>

<!- Non Accettabile ->
<div><span>
<nl>
<li>Uno</li>
<li>
Due
</li>
</nl></span>
</div>
```

1.3.4 Script Javascript

Gli Script in Javascript devono rispettare le seguenti convenzioni:

1. Gli script che svolgono funzioni distinte dal mero rendering della pagina dovrebbero essere collocati in file dedicati.
2. Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.
3. I documenti Javascript devono essere iniziati da un commento analogo a quello presente nei file Java.
4. Le funzioni Javascript devono essere documentate in modo analogo ai metodi Java.
5. Gli oggetti Javascript devono essere preceduti da un commento in stile Javadoc, che segue il seguente formato:

```
/**
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti del costruttore (@param)
*
* Metodo nomeMetodo1
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti (@param)
* Specifica dei risultati (@return)
*
* Metodo nomeMetodo2
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti (@param)
* Specifica dei risultati (@return)
*
* ...
*/
function   fX (a, b, c) {
```

1.3.5 Fogli di stile CSS

I fogli di stile (CSS) devono seguire le seguenti convenzioni:

Tutti gli stili non inline devono essere collocati in fogli di stile separati.

Ogni foglio di stile deve essere iniziato da un commento analogo a quello presente nei file Java.

Ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);
3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

1.3.6 Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere maiuscole;
2. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

I nomi dei campi devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere maiuscole;
2. Se il nome è costituito da più parole, è previsto l'uso di underscore (_);

3. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

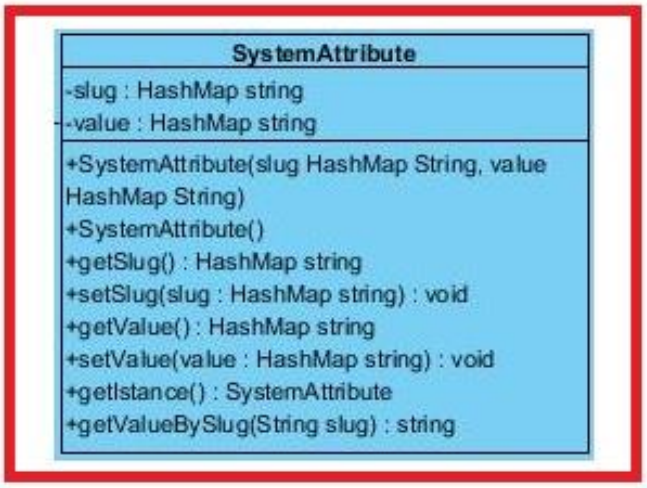
1.4 Design Pattern

1.4.1 MVC

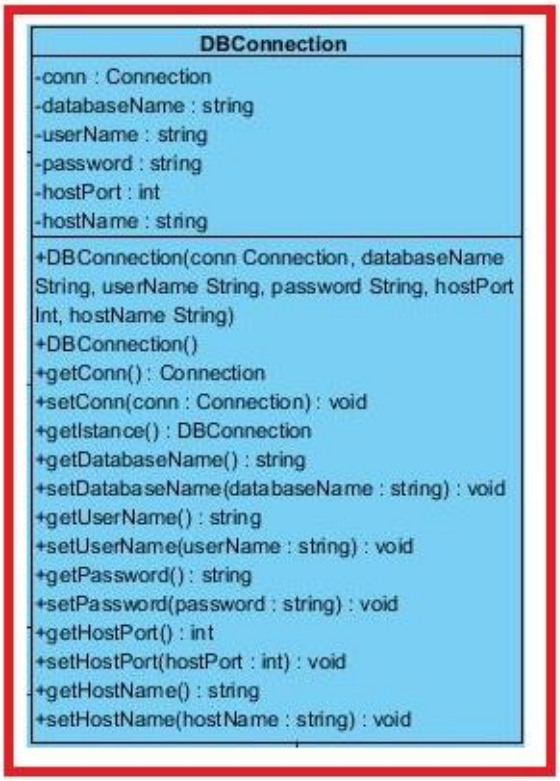
Il design pattern MVC consente la suddivisione del sistema in tre blocchi principali: Model, View e Controller. Il Model fornisce i metodi di accesso ai dati persistenti, il View si occupa dell'interazione con l'utente e della presentazione dei dati prelevati dal Model, il Controller riceve i comandi dell'utente attraverso il View e modifica lo stato di quest'ultimo e del Model. Nel nostro sistema le classi sono state divise in package avente nome richiamante il blocco di appartenenza nel design pattern.

1.4.2 Singleton

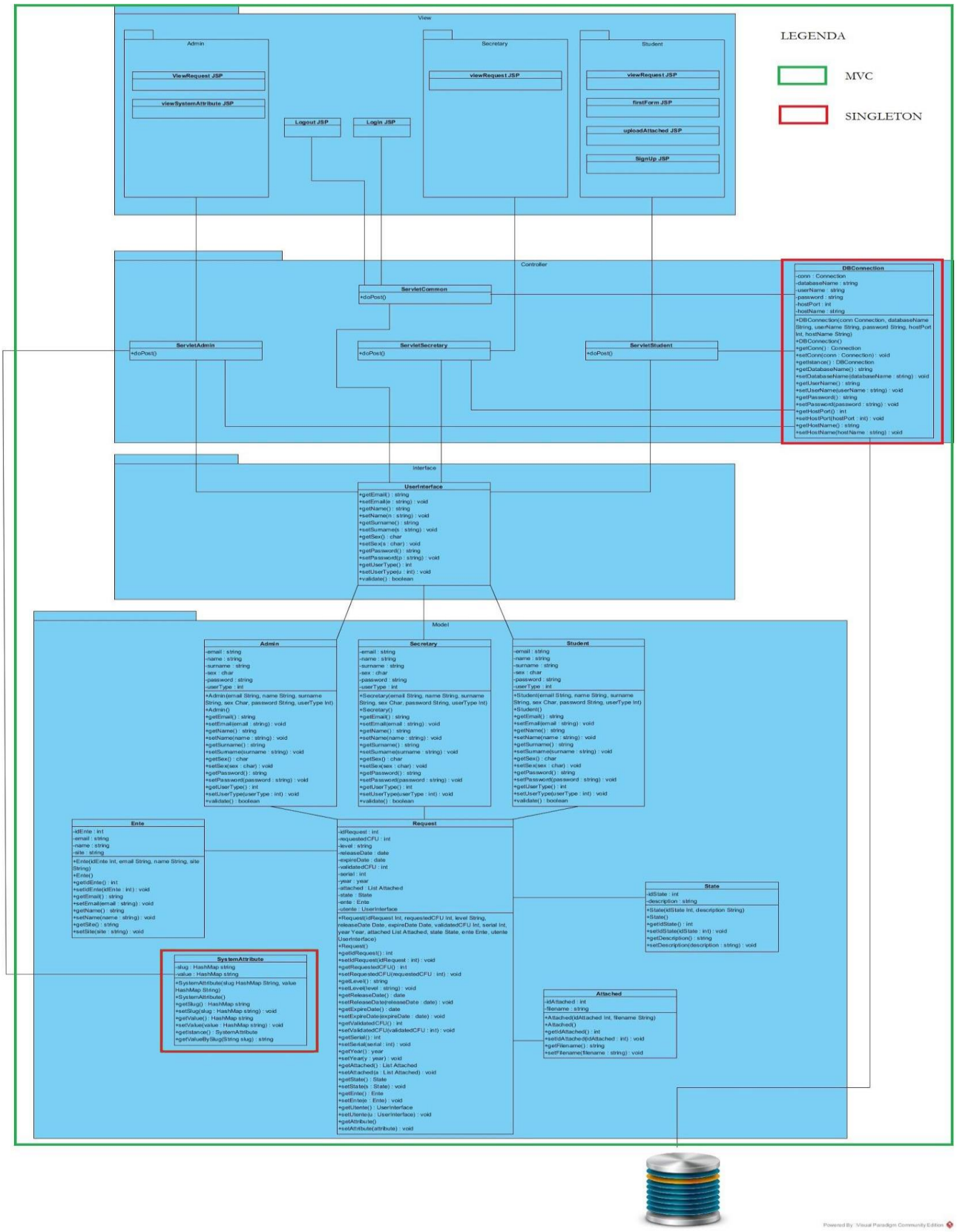
Abbiamo progettato una classe (SystemAttribute) avente il compito di mantenere le informazioni necessarie per il corretto funzionamento del software, con operazioni per poter cambiare queste ultime. Per evitare ridondanze e/o mancata coerenza tra più istanze si è deciso di sviluppare questa classe come Singleton.



Abbiamo progettato una singola classe (DBConnection) che consentisse di effettuare tutte le operazioni con il database. Per evitare la perdita di efficienza dovuta alla creazione di più istanze di questa classe si è deciso di renderla un Singleton.



Nella pagina successiva è riportata una vista completa del sistema:



1.5 Definizioni, Acronimi e abbreviazioni

JSP: acronimo di Java Scripting Preprocessor, è una tecnologia di programmazione web in java per lo sviluppo della logica di presentazione (tipicamente secondo il pattern MVC) di applicazioni web.

MVC: acronimo di Model-view-controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software**.**

Off-The-Shelf: Servizi esterni al sistema di cui viene fatto utilizzo.

Bootstrap: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web.

HTML: Linguaggio di programmazione utilizzato per lo sviluppo di pagine Web.

CSS: acronimo di Cascading Style Sheets è un linguaggio usato per definire la formattazione delle pagine Web.

JavaScript: JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

JQuery: JQuery è una libreria JavaScript per applicazioni web.

AJAX: AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive.

lowerCamelCase: Il lowerCamelCase è una tecnica di naming delle variabili adottata dallo standard Google Java. Essa consiste nello scrivere più parole insieme delimitando la fine e l'inizio di una nuova parola con una lettera maiuscola.

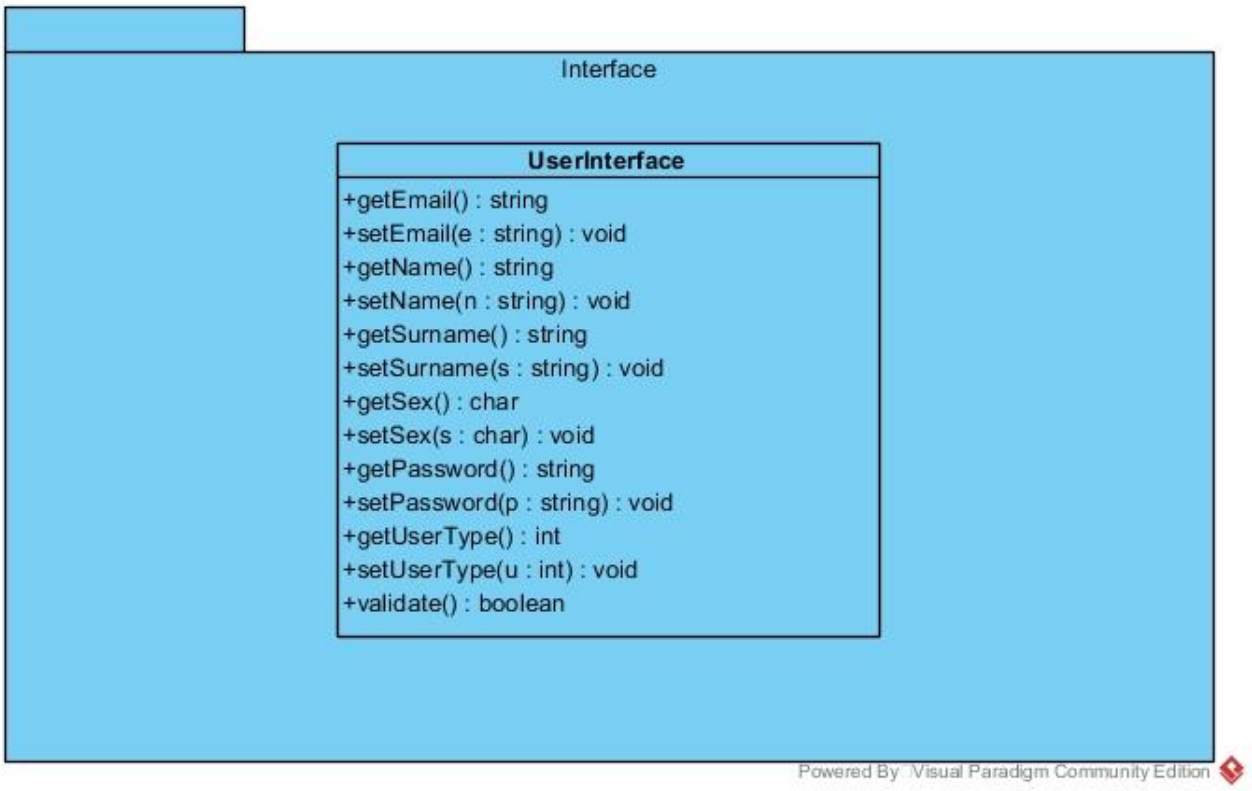
Servlet: i servlet sono oggetti scritti in linguaggio Java che operano all'interno di un server web.

Tomcat: Apache Tomcat è un web server open source. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

2. Packages

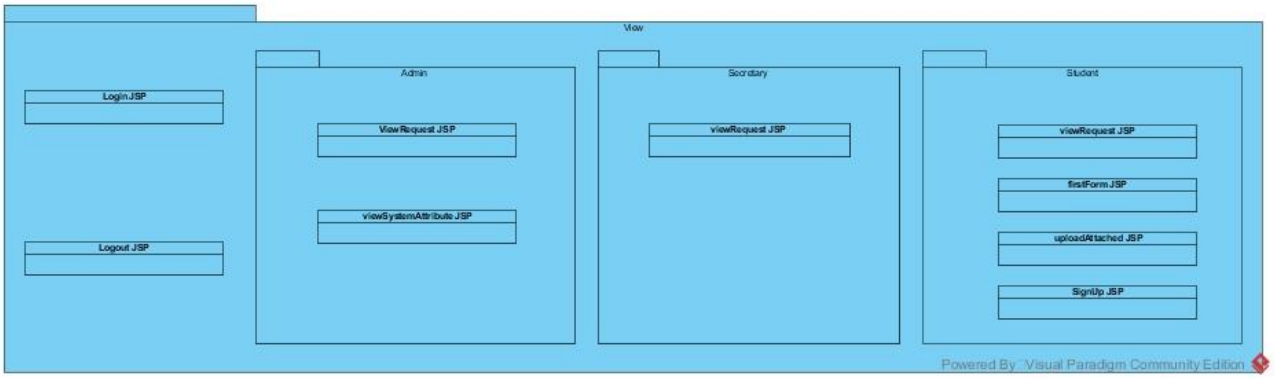
2.1 Interface

Il package Interface è implementato con la classe UserInterface che rappresenta l'interfaccia utente e che è collegato con il package Model con la classe Admin, Secretary e Student descritte successivamente.



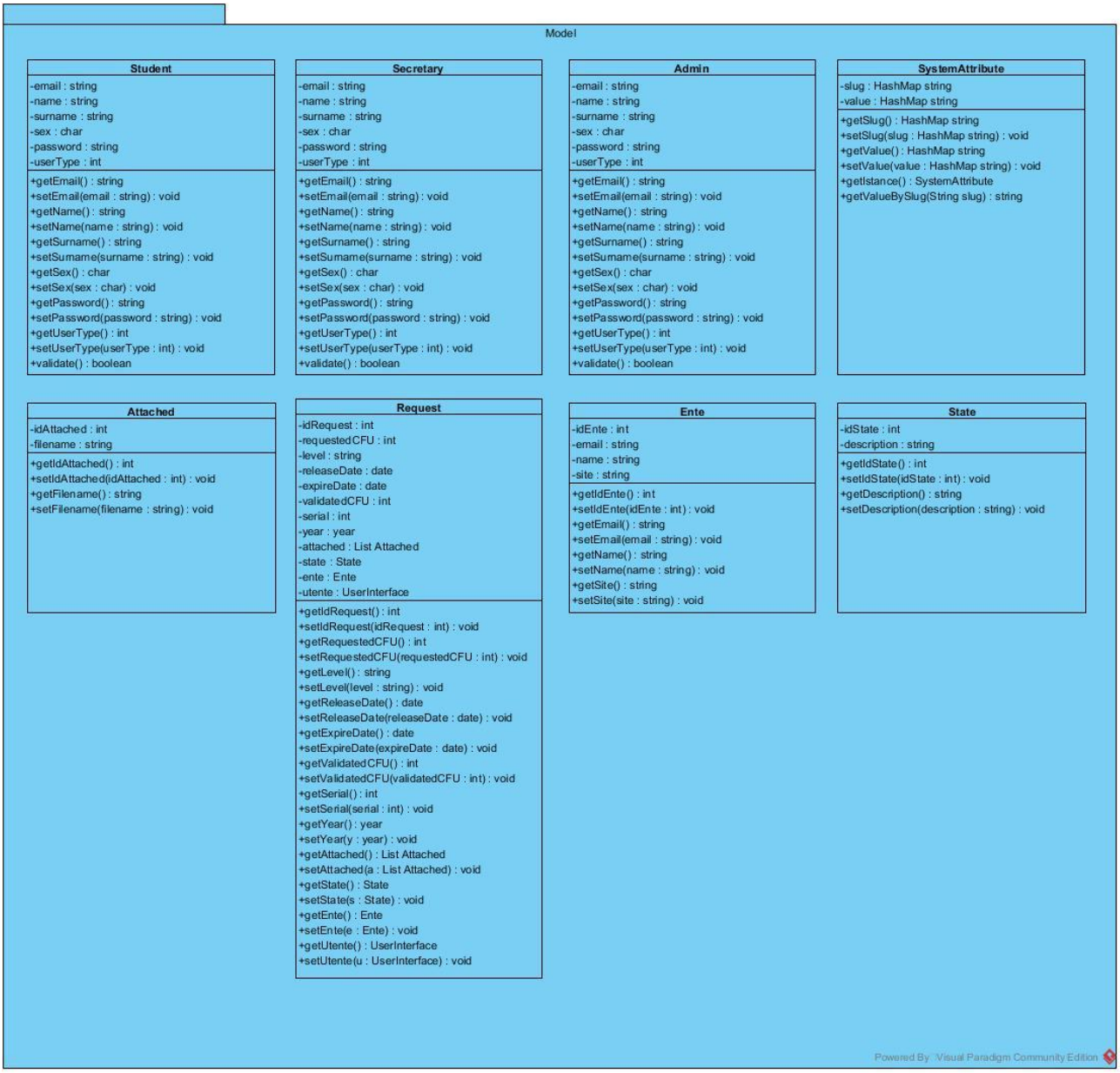
2.2 View

Il package View è formato a sua volta da tre packages: Admin, Secretary e Student; inoltre, sono presenti classi Login JSP e Logout JSP. Il package Admin viene implementato con la classe ViewRequest JSP che viene utilizzata per mostrare all'admin la lista degli studenti che hanno richiesto la convalida dei cfu, e dalla classe viewSystemAttribute JSP che permette, tramite interfaccia grafica, l'azione di accettare o rifiutare la richiesta dello studente. Queste due classi, presenti nel package Admin, sono strettamente collegate con il package Controller. Il package Secretary viene implementato con la classe ViewRequest JSP con la stessa funzione della ViewRequest JSP descritta precedentemente. Questo package è collegato con il package Controller, in modo specifico con la ServletSecretary che a sua volta è collegato con UserInterface del package Interface, e infine collegato con Secretary nel package Model. Il package Studente viene implementato con le classi ViewRequest JSP che viene utilizzata per mostrare lo stato della richiesta inviata. La classe firstForm JSP serve a far inserire i dati univoci allo studente. La classe uploadAttached che permette di allegare il certificato, e la classe signup che serve a registrarsi sul sito. Queste classi presenti nel package Student vengono collegate con il package Interface. Le classi Login JSP e Logout JSP vengono implementate per gestire le classiche azioni di accesso e disconnessione.



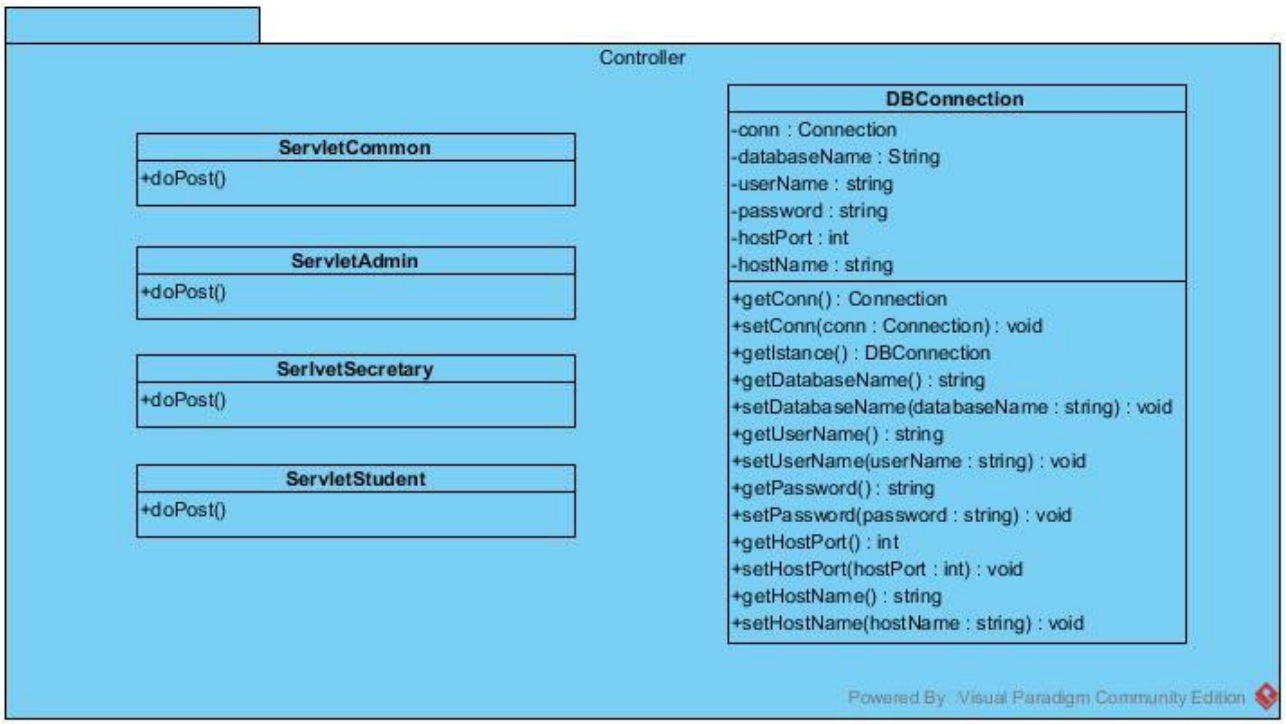
2.3 Model

Il package Model contiene tutte le classi dedite alla gestione dei dati persistenti. Esso si occupa di fare da tramite tra l'applicazione e il database sottostante. Ogni classe contenuta all'interno di questo pacchetto fornisce i metodi per accedere ai dati utili all'applicazione. Le classi contenute all'interno di questo package sono: Student, Secretary, Admin (tutte e tre comunicano con la classe UserInterface presente nel package Interface), Ente, State, Attached, Request (che presenta un collegamento diretto con tutte le classi presenti in questo package) e SystemAttribute.



2.4 Controller

Il package controller riceve, tramite il pacchetto View, i comandi dell'utente. Esso è formato da 4 Servlet: ServletCommon (si occupa della gestione di Login JSP e Logout JSP, e comunica direttamente con la classe UserInterface e DBConnection, contenuta anch'essa in questo pacchetto), ServletAdmin (si occupa di gestire i comandi lanciati dall'Admin, e comunica con la UserInteface e DBConnection), ServletStudent (si occupa di gestire i comandi lanciati dallo studente, e comunica con UserInterface e DBConnection), ServletSecretary (si occupa di gestire i comandi lanciati dall'utente di segreteria, e comunica con UserInterface e DBConnection). Infine all'interno del pacchetto è presente la classe DBConnection che si occupa di gestire l'intero Database, e, come detto prima, comunica con tutte la Servlet presenti in questo pacchetto.



3. Class interfaces

Nome classe	Login JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di login.
Pre-condizione	context Login JSP: validate(email, Password)0; pre: email!=null && password!=null email.equals(DB.email) && password.equals(DB.password)
Post-condizione	
Invarianti	

Nome classe	Logout JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di logout.
Pre-condizione	context Logout JSP: logout(User) pre: User.isLogged
Post-condizione	

Invarianti	

Nome classe	ViewRequest JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di visualizzazione delle richieste da parte dell'admin.
Pre-condizione	context ViewRequest JSP: viewRequest()
Post-condizione	
Invarianti	

Nome classe	ViewSystemAttribute JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di accettazione e rifiuto delle richieste.
Pre-condizione	
Post-condizione	
Invarianti	

Nome classe	ViewRequest JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di visualizzazione delle richieste da parte della segreteria.
Pre-condizione	context ViewRequest JSP: viewRequest()
Post-condizione	
Invarianti	

Nome classe	ViewRequest JSP

Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di visualizzazione della richiesta da parte dello studente.
Pre-condizione	context ViewRequest JSP: viewRequest()
Post-condizione	
Invarianti	

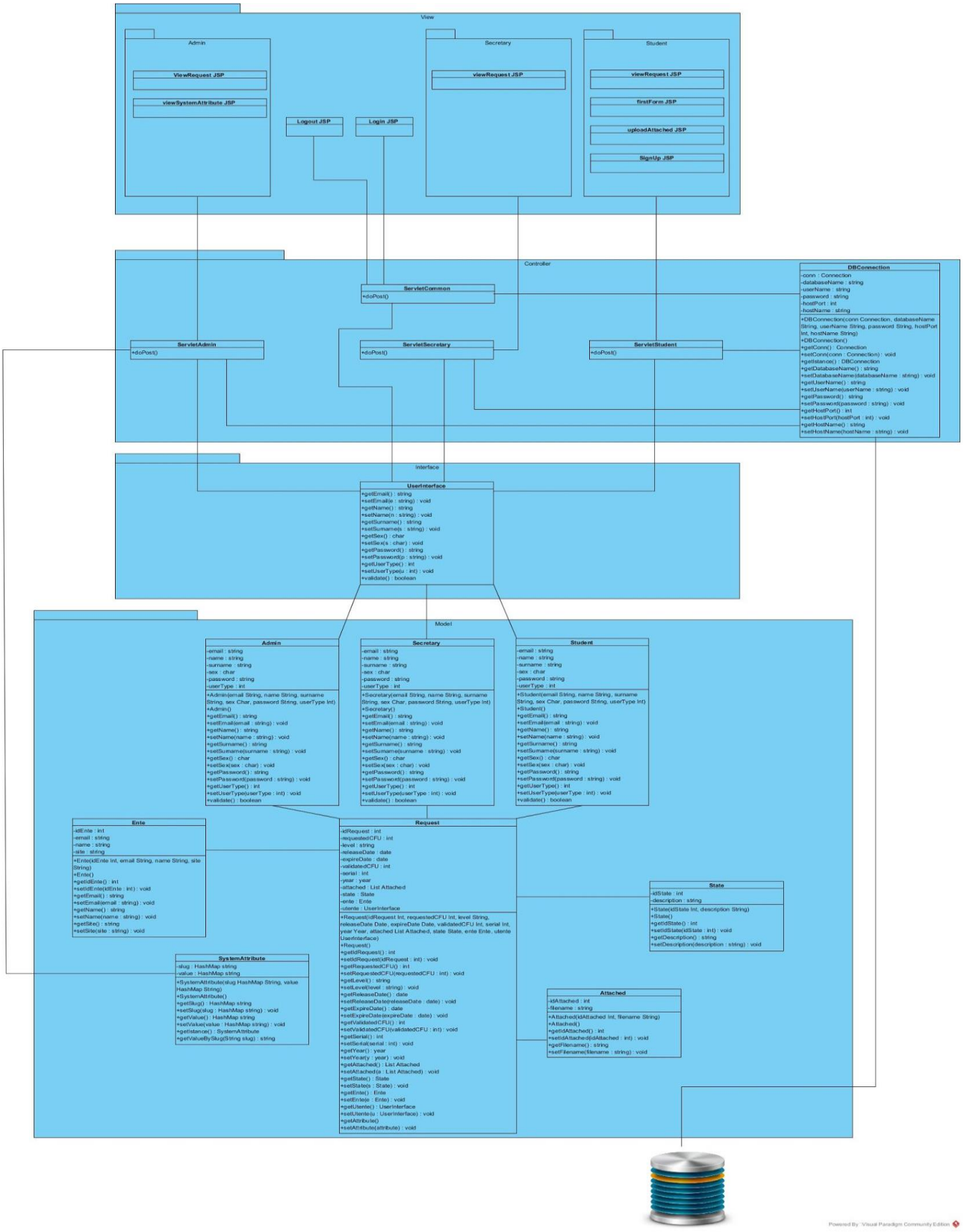
Nome classe	**uploadAttached JSP**
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di caricamento del certificato.
Pre-condizione	context uploadAttached JSP: uploadCertificate(certificate) pre: certificate.formato(pdf)
Post-condizione	
Invarianti	

Nome classe	**FirstForm JSP**
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di compilazione del primo form.
Pre-condizione	context FirstForm JSP:checkUser(Nome,Cognome,E-mai, cfu da convalidare) pre: campi!=null
Post-condizione	
Invarianti	

Nome classe	**SignUp JSP**

Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di registrazione dell'utente.
Pre-condizione	context SignUp JSP: checkUser(Nome,Cognome,E-mai,password) pre: campi!=null && DB.notExsist(utente)
Post-condizione	
Invarianti	

4. Class diagram



5. Glossario

Trade-off: Il Trade-off è una situazione che implica una scelta tra due possibilità, in cui la perdita di valore di una costituisce un aumento di valore in un'altra.

Off-The-Shelf: Servizi esterni al sistema di cui viene fatto utilizzo.

Bootstrap: Framework che contiene librerie utili per lo sviluppo responsive di pagine web.

HTML: Linguaggio di programmazione utilizzato per lo sviluppo di pagine Web.

CSS: Linguaggio usato per definire la formattazione delle pagine Web.

JavaScript: JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

JQuery: JQuery è una libreria JavaScript per applicazioni web.

AJAX: AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive.

lowerCamelCase: Il lowerCamelCase è una tecnica di naming delle variabili adottata dallo standard Google Java. Essa consiste nello scrivere più parole insieme delimitando la fine e l'inizio di una nuova parola con una lettera maiuscola.

Servlet: i servlet sono oggetti scritti in linguaggio Java che operano all'interno di un server web.

Tomcat: Apache Tomcat è un web server open source. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.