

ODD Object Design Document

TutoratoSmart

|  |  |
| --- | --- |
| Riferimento |  |
| Versione | 0.4 |
| Data | 17/12/2019 |
| Destinatario | Prof.ssa F. Ferrucci |
| Presentato da | Marco Delle Cave, Francesco Pagano,  Manuel Pisciotta, Alessia Olivieri |
| Approvato da |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autori |
| 11/12/19 | 0.1 | Definizione ed introduzione ODD, scelta dei trade-off | [tutti] |
| 14/12/19 | 0.2 | Scelta design pattern, definizione interfacce delle classi e package | [tutti] |
| 15/12/19 | 0.3 | Revisione generale ODD | [tutti] |
| 17/12/19 | 0.4 | Rettifica ODD | [tutti] |

Sommario

[1. Introduzione 3](#_Toc27579838)

[1.1 Trade-off 3](#_Toc27579839)

[1.1.1 Comprensibilità vs costi 3](#_Toc27579840)

[1.1.2 Tempo di risposta vs Affidabilità 3](#_Toc27579841)

[1.1.3 Manutenibilità vs efficienza 3](#_Toc27579842)

[1.2 Componenti off-the-shelf 3](#_Toc27579843)

[1.3 Linee guida per la documentazione dell’interfaccia 4](#_Toc27579844)

[1.3.1 Classi e interfacce Java 4](#_Toc27579845)

[1.3.2 Pagine lato Server (JSP) 5](#_Toc27579847)

[1.3.3 Pagine HTML 6](#_Toc27579848)

[1.3.5 Fogli di stile CSS 6](#_Toc27579849)

[1.3.6 Database SQL 7](#_Toc27579850)

[1.4 Design pattern 8](#_Toc27579851)

[1.5 Definizioni, acronimi e abbreviazioni 8](#_Toc27579852)

[1.6 Riferimenti 8](#_Toc27579853)

[2. Packages 9](#_Toc27579854)

[2.1 View 9](#_Toc27579855)

[2.2 Model 10](#_Toc27579856)

[2.3 Controller 12](#_Toc27579857)

[3. Interfacce delle classi 13](#_Toc27579858)

[4. Glossario 16](#_Toc27579859)

# 1. Introduzione

## Trade-off

### 1.1.1 Comprensibilità vs costi

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

### 1.1.2 Tempo di risposta vs Affidabilità

Il sistema sarà implementato in modo tale da preferire l’affidabilità al tempo di risposta, in modo tale da garantire un controllo più accurato dei dati in input a discapito del tempo di risposta del sistema.

### 1.1.3 Manutenibilità vs efficienza

Si preferisce avere una maggiore manutenibilità del sistema facendo in modo che ogni sottosistema non acceda direttamente allo storage, ma che l’accesso ai dati viene gestito da un sottosistema intermedio a discapito dell’efficienza e delle prestazioni generali

## Componenti off-the-shelf

Per il progetto software che si vuole realizzare facciamo uso di componenti***off-the-shelf****,* che sono componenti software disponibili sul mercato per facilitare la creazione del software.

Per il sistema che si vuole realizzare ci interessa un framework per applicazioni web e librerie per la gestione delle interfacce grafiche.

Il framework che andremo ad utilizzare per il comparto grafico è Bootstrap, che è un framework open source che contiene una raccolta di strumenti [liberi](https://it.wikipedia.org/wiki/Software_libero) per la creazione di [siti](https://it.wikipedia.org/wiki/Sito_web) e [applicazioni](https://it.wikipedia.org/wiki/Applicazione_web) per il [Web](https://it.wikipedia.org/wiki/Web). Essa contiene [modelli](https://it.wikipedia.org/wiki/Template) di progettazione basati su [HTML](https://it.wikipedia.org/wiki/HTML) e [CSS](https://it.wikipedia.org/wiki/CSS), sia per la [tipografia](https://it.wikipedia.org/wiki/Tipografia), che per le varie componenti dell'interfaccia, come [moduli](https://it.wikipedia.org/wiki/Form), bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di [JavaScript](https://it.wikipedia.org/wiki/JavaScript). 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.

## Linee guida per la documentazione dell’interfaccia

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 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.

Organizzazione dei file

* + Ogni file deve essere:
* Sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un’unica funzionalità che persegue. Ogni pagina della Richiesta Tutorato (compilazione richiesta, visualizzazione stato richiesta, modifica prenotazione, etc.) deve essere implementata in file separati;
* La convenzione per quanto riguarda i nomi dei file, delle operazioni e delle variabili è quella di avere nomi evocativi, ma soprattutto in lingua inglese.
* Organizzare in una cartella i file delle librerie usate e le altre risorse scaricate necessarie per lo sviluppo del progetto.

### 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 () si trova all'inizio di una 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.

*Es.*

<!-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>

### 1.3.5 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:

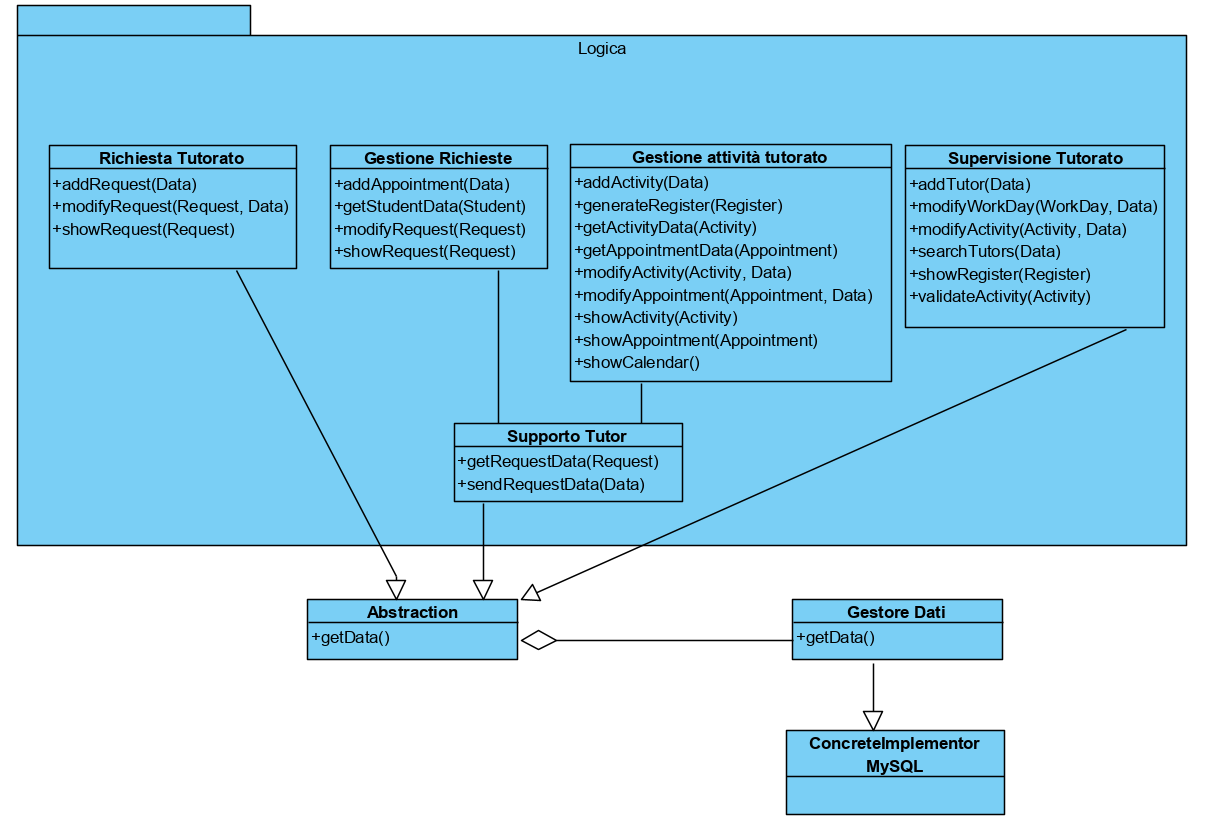
1. Devono essere costituiti da sole lettere maiuscole;

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.

## Design pattern

**Bridge pattern**



TutoratoSmart fa uso del Bridge Design Pattern perché abbiamo bisogno di utilizzare un’unica interfaccia per diversi accessi allo storage: offrendo un’unica interfaccia si garantisce che l’eventuale cambio di implementazione del database usato comporta la modifica solo a una componente e non a svariate componenti del sistema.

## Definizioni, acronimi e abbreviazioni

**DBMS:** Data Base Management System.

**Off-The-Shelf:** Servizi esterni di cui viene fatto utilizzo da terzi.

**Framework:** Software di supporto allo sviluppo web.

**HTML:** Linguaggio di mark-up per pagine web.

**CSS:** Linguaggio usato per definire la formattazione di pagine web.

**JavaScript:**  [Linguaggio di scripting](https://it.wikipedia.org/wiki/Linguaggio_di_scripting) [orientato agli oggetti](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti) e agli [eventi](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_eventi), comunemente utilizzato nella [programmazione Web](https://it.wikipedia.org/wiki/Programmazione_Web) [lato client](https://it.wikipedia.org/wiki/Lato_client) per la creazione, in [siti web](https://it.wikipedia.org/wiki/Sito_web) e [applicazioni web](https://it.wikipedia.org/wiki/Applicazione_web), di effetti dinamici [interattivi](https://it.wikipedia.org/wiki/Interattivit%C3%A0) tramite [funzioni](https://it.wikipedia.org/wiki/Funzione_(informatica)) di [script](https://it.wikipedia.org/wiki/Script) invocate da *eventi* innescati a loro volta in vari modi dall'utente sulla [pagina web](https://it.wikipedia.org/wiki/Pagina_web) in uso

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

**Camel Notation:** Consiste nello scrivere più parole insieme delimitando la fine e l’inizio di una nuova parola con una lettera maiuscola.

**Hard Coding:** Codifica fissa.

## Riferimenti

* Ian Sommerville, Software Engineering, Addison Wesely
* TS\_SDD\_0.4
* <http://getbootstrap.com/>

# 2. Packages

## 2.1 View

Il package View è formato a sua volta da tre packages: Studente, Tutor e CommissioneTutorato e in più dalle classi HomePage JSP, Login JSP e Logout JSP. Il package Studente viene implementato con le classi:

* RegistrationPage JSP: permette allo studente di registrarsi alla piattaforma;
* RequestPage JSP: mostra un’ interfaccia grafica che consente allo studente di inserire le informazioni per richiedere un appuntamento;
* RequestInfoPage JSP: pagina che mostra le informazioni relative ad una richiesta di appuntamento effettuata indicandone lo stato e consentendo la modifica o cancellazione della richiesta;
* RequestModifyPage JSP: pagina che consente allo studente di modificare il commento, il giorno e l’orario di un appuntamento richiesto;

Queste quattro classi, presenti nel package Studente, sono strettamente collegate con il package Controller.

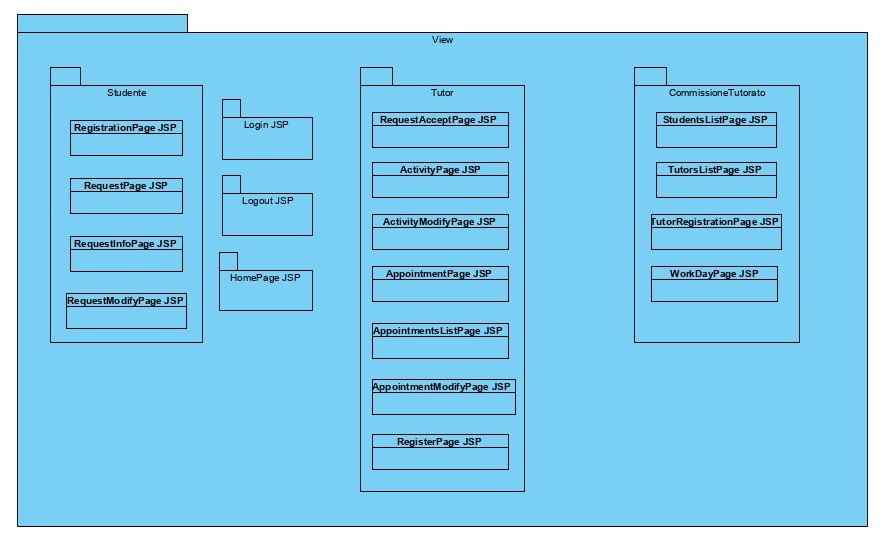
Il package Tutor viene implementato con le classi:

* RequestAcceptPage JSP: pagina che consente l’accettazione di una prenotazione;
* ActivityPage JSP: pagina che consente la registrazione di un’ attività lavorativa svolta;
* ActivityModifyPage JSP: pagina che consente al tutor di modificare i dati di un’attività lavorativa salvata sul registro;
* AppointmentPage JSP: pagina che consente la registrazione di un appuntamento;
* AppointmentsListPage JSP: pagina che elenca tutti gli appuntamenti confermati dal tutor;
* AppointmentModifyPage JSP: pagina che consente al tutor di modificare i dati di un appuntamento salvato;
* RegisterPage JSP: pagina che mostra le informazioni relative al registro di tutorato;

Il package CommissioneTutorato viene implementato con le classi:

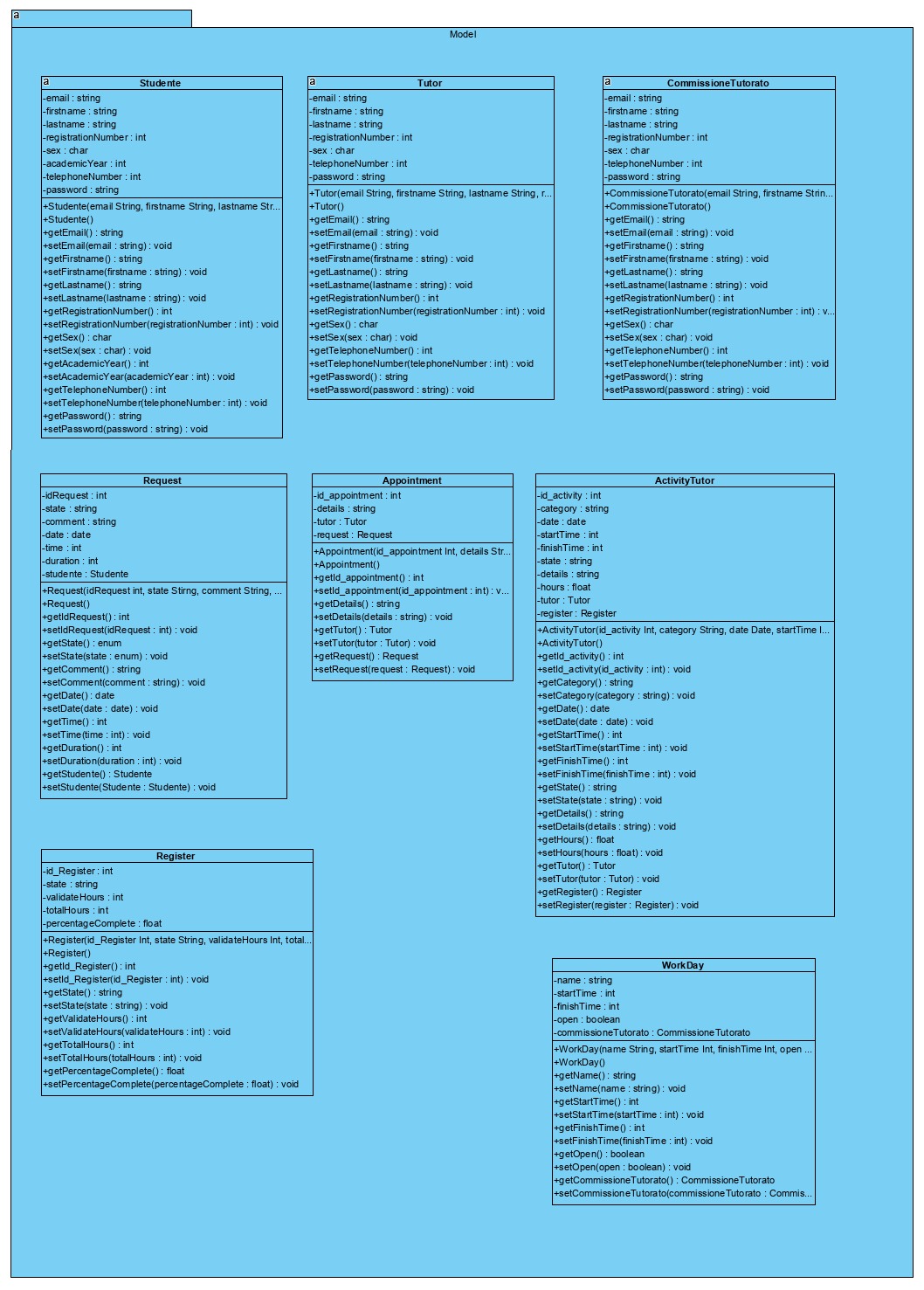
* StudentsListPage JSP: pagina che elenca tutti gli studenti trovati e i dettagli degli appuntamenti;
* TutorsListPage JSP: pagina che elenca tutti i tutor trovati;
* TutorRegistrationPage JSP: pagina che consente la registrazione di un nuovo tutor;
* WorkDayPage JSP: pagina che consente alla Commissione di specificare i giorni/orari in cui lo sportello informativo è attivo, attraverso un apposito form di compilazione;

Le classi Login JSP e Logout JSP vengono implementate per gestire le classiche azioni di accesso e disconnessione, mentre Index JSP è la prima pagina che viene mostrata all’utente quando viene aperto il sito.



## 2.2 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: Studente, Tutor, CommissioneTutorato, Request, Appointment, ActivityTutor, Register, WorkDay.



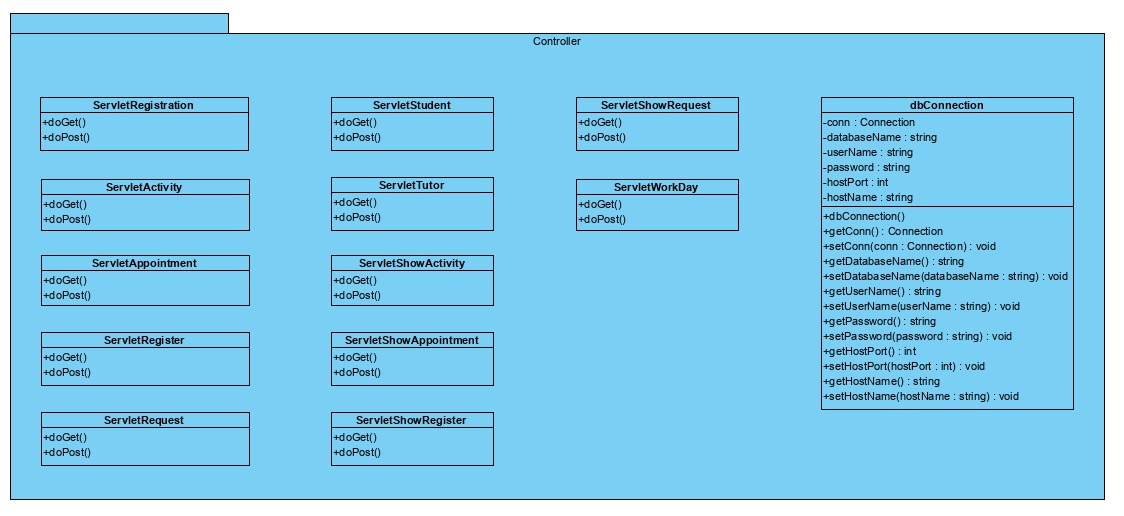
## 2.3 Controller

Il package controller riceve, tramite il pacchetto View, i comandi dell’utente.

Esso è formato da 12 Servlet:

* ServletRegistration: si occupa di gestire la Registrazione di un nuovo Utente;
* ServletActivity: si occupa di gestire la registrazione/modifica di un’ attività di tutorato;
* ServletAppointment: si occupa di gestire la registrazione/modifica di un appuntamento;
* ServletRegister: gestisce la Generazione di un file contenente le informazioni del registro di tutorato;
* ServletRequest: gestisce la registrazione/modifica di una richiesta;
* ServletStudent: si occupa di gestire la ricerca degli studenti e di visualizzarli in un elenco;
* ServletTutor: si occupa di gestire la ricercar dei tutor e di visualizzarli in un elenco;
* ServletShowActivity: gestisce la visualizzazione dei dettagli di un’attività di tutorato;
* ServletShowAppointment: gestisce la visualizzazione dei dettagli di un appuntamento;
* ServletShowRegister: gestisce la visualizzazione dei dettagli del registro di un tutor;
* ServletShowRequest: si occupa di gestire la visualizzazione dei dettagli di una richiesta;
* ServletWorkDay: si occupa di gestire la specifica dei giorni lavorativa in cui lo sportello è attivo;

Infine all’interno del pacchetto è presente la classe DBConnection che si occupa di gestire l’intero Database, e comunica con tutte la Servlet presenti in questo pacchetto.



# 3. Interfacce delle classi

|  |  |
| --- | --- |
| Nome classe | RichiestaTutorato |
| Descrizione | Rappresenta il gestore delle funzionalità relate alle richieste di tutorato. |
| Pre-condizione | **context** RichiestaTutorato:: addRequest(Data);  **pre:** Data!=null && isValid(Data)  **context** RichiestaTutorato:: modifyRequest(Request, Data);  **pre:** Request!=null && exists(Request) && Data!=null && isValid(Data)  **context** RichiestaTutorato:: showRequest(Request);  **pre:** Request!=null && exists(Request) |
| Post-condizione | **context** RichiestaTutorato:: addRequest(Data);  **post:** exists(Request) |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome classe | GestioneRichieste |
| Descrizione | Rappresenta il gestore delle funzionalità relate alla gestione delle richieste. |
| Pre-condizione | **context** GestioneRichieste:: addAppointment(Data);  **pre:** Data!=null && isValid(Data)  **context** GestioneRichieste:: getStudentData(Student);  **pre:** Student!=null && exists(Student)  **context** GestioneRichieste:: modifyRequest(Request);  **pre:** Request!=null && exists(Request)  **context** GestioneRichieste:: showRequest(Request);  **pre:** Request!=null && exists(Request) |
| Post-condizione | **context** GestioneRichieste:: addAppointment(Data);  **post:** exists(Appointment) |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome classe | GestioneAttivitàTutorato |
| Descrizione | Rappresenta il gestore delle funzionalità relate alla gestione dell’attività di tutorato. |
| Pre-condizione | **context** GestioneAttivitàTutorato:: addActivity(Data);  **pre:** Data!=null && isValid(Data)  **context** GestioneAttivitàTutorato:: generateRegister(Register);  **pre:** Register!=null && exists(Register)  **context** GestioneAttivitàTutorato:: getActivityData(Activity);  **pre:** Activity!=null && exists(Activity)  **context** GestioneAttivitàTutorato:: getAppointmentData(Appointment);  **pre:** Appointment!=null && exists(Appointment)  **context** GestioneAttivitàTutorato:: modifyActivity(Activity, Data);  **pre:** Activity!=null && exists(Activity) && Data!=null && isValid(Data)  **context** GestioneAttivitàTutorato:: modifyAppointment(Appointment, Data);  **pre:** Appointment!=null && exists(Appointment) && Data!=null && isValid(Data)  **context** GestioneAttivitàTutorato:: showActivity(Activity);  **pre:** Activity!=null && exists(Activity)  **context** GestioneAttivitàTutorato:: showAppointment(Appointment);  **pre:** Appointment!=null && exists(Appointment)  **context** GestioneAttivitàTutorato:: showCalendar(); |
| Post-condizione |  |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome classe | SupportoTutor |
| Descrizione | Rappresenta il gestore delle funzionalità relate al supporto del tutor. |
| Pre-condizione | **context** SupportoTutor:: getRequestData(Request);  **pre:** Request!=null && exists(Request)  **context** SupportoTutor:: sendRequestData(Data) ;  **pre:** Data!=null && isValid(Data) |
| Post-condizione |  |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome classe | SupervisioneTutorato |
| Descrizione | Rappresenta il gestore delle funzionalità relate alla supervisione dell’attività di tutorato. |
| Pre-condizione | **context** SupervisioneTutorato:: addTutor(Data);  **pre:** Data!=null && isValid(Data)  **context** SupervisioneTutorato:: modifyWorkDay(WorkDay, Data);  **pre:** WorkDay!=null && exists(WorkDay) && Data!=null && isValid(Data)  **context** SupervisioneTutorato:: modifyActivity(Activity, Data);  **pre:** Activity!=null && exists(Activity) && Data!=null && isValid(Data)  **context** SupervisioneTutorato:: searchTutors(Data);  **pre:** Data!=null && isValid(Data)  **context** SupervisioneTutorato:: showRegister(Register);  **pre:** Register!=null && exists(Register)  **context** SupervisioneTutorato:: validateActivity(Activity);  **pre:** Activity!=null && exists(Activity) |
| Post-condizione | **context** SupervisioneTutorato:: addTutor(Data);  **post:** exists(Tutor) |
| Invarianti |  |

# 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.

**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**: Le 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.