Laurea Triennale in informatica-Università di Salerno
Corso di *Ingegneria del Software*- Prof.ssa F. Ferrucci

Object Design Document

US UniSeat

Riferimento	
Versione	1.2
Data	06/12/2019
Destinatario	Prof.ssa Ferrucci Filomena
Presentato da	De Caro Antonio, De Santis Marco, Spinelli Gianluca, Capozzoli Lorenzo, Rocco Simone Pasquale
Approvato da	

Revision History

Data	Versione	Cambiamenti	Autori
06/12/2019	0.1	Stesura scheletro documento e aggiunta capitolo 1	De Caro Antonio
11/12/2019	0.2	Aggiunto il Class Diagram	[tutti]
13/12/2019	0.3	Aggiunto il capitolo 2	De Caro Antonio
13/12/2019	0.3	Aggiunto il capitolo 3	De Caro Antonio
14/12/2019	0.4	Aggiunto il capitolo 4	[tutti]
16/12/2019	1.0	Revisione documento	[tutti]
19/12/2019	1.1	Effettuata operazione di Reverse Engineering e aggiunta la classe Disponibilità Checker nel control	De Caro Antonio
15/01/2020	1.2	Aggiunto il collegamento alla JavaDoc	De Caro Antonio

Sommario

1. Introduzione

1.1 Object Design Trade-Off

Durante la fase di analisi e di progettazione del sistema abbiamo individuato diversi compromessi per lo sviluppo del sistema. Anche durante la fase di Object Design sorgono diversi compromessi che andremo ad analizzare in questo paragrafo:

Memoria / Estensibilità: Il sistema garantisce un'alta modularità come è stato specificato dai requisiti non funzionali (US_RAD_V1.4 capitolo 3 paragrafo 3) e dai trade off individuati nella fase di System Design (US_SDD_V1.0 capitolo 1 paragrafo 2). Il sistema deve quindi permettere l'estensibilità a discapito della memoria utilizzata.

Affidabilità / Tempo di risposta: Essendo il nostro sistema un sistema "Transaction Based", deve sempre garantire una risposta affidabile e consistente, a discapito del tempo di risposta.

Criteri di manutenzione / 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.

Costi / Sicurezza: Per evitare una situazione di over-budget, il sistema garantisce il minimo livello di sicurezza sufficiente (definito nel Glossario, capitolo 5), senza soddisfare tutti i criteri di sicurezza previsti dalla fondazione OWASP (https://www.owasp.org/index.php/Security_by_Design_Principles).

Nella seguente tabella, il Design Goal in **grassetto** indica il design goal prioritario.

Trade-Off	
Memoria	Estensibilità
Affidabilità	Tempo di risposta
Criteri di manutenzione	Criteri di performance
Costi	Sicurezza

1.2 Componenti Off-the-Shelf

Utilizzeremo diverse componenti Off-the-Shelf nell'implementazione del nostro sistema.

Per quanto riguarda il Front End, utilizzeremo il framework Bootstrap (https://getbootstrap.com/docs/4.4/getting-started/introduction/) per una gestione dello stile più semplice. Questo framework è open source e integra all'interno di esso sistemi per gestire in maniera semplificata il codice HTML e CSS. Questo framework inoltre include alcune funzioni definite in JavaScript che dovranno essere integrate.

Dal punto di vista funzionale invece, utilizzeremo JQuery, un framework di JavaScript che permette di agire sul DOM del documento HTML, di effettuare chiamate AJAX e di gestire le animazioni della pagina web in maniera molto più semplice.

Per l'interfaccia grafica e il lato Front End, sono stati utilizzati i seguenti template:

https://bootstrapmade.com/regna-bootstrap-onepage-template/

https://themeforest.net/item/snoopy-multipurpose-bootstrap-admin-dashboard-template-ui-kit/21458186

Per quanto riguarda il lato Back End, utilizzeremo il Web Container di JavaEE ed il WebServer Tomcat per gestire le richieste al nostro sistema.

Per quanto riguarda l'invio delle e-mail utilizzeremo la libreria JavaMail (https://javaee.github.io/javamail/).

1.3 Design Pattern

Il nostro sistema utilizzerà diversi Design Pattern:

- **Singleton**
  La classe DBConnection, che si occupa di creare e mantenere una connessione al database, è una classe singleton, così che possa essere acceduta in maniera atomica, senza creare molteplici connessioni.
- **DAO (Data Access Object)**
  Per garantire un accesso alle informazioni persistenti senza prescindere dal sistema utilizzato per la persistenza, utilizzeremo un'interfaccia DAO che ci permetterà di disaccoppiare la logica del sistema dal gestore della persistenza (che potrà cambiare nel tempo in maniera completamente isolata).
- **Proxy Pattern**
  Per semplificare l'interfaccia della libreria JavaMail, utilizzeremo la classe "EmailManager" che fungerà da proxy per la libreria. In questo modo le servlet si limiteranno ad interfacciarsi con una classe che definisce pochi metodi, senza preoccuparsi di tutti i criteri per inviare e-mail.

1.4 Linee guida per la documentazione delle interfacce

In questo paragrafo verranno definite le linee guida a cui un programmatore deve attenersi nell'implementazione del sistema.

In ciascun sotto paragrafo, quando si parla di indentazione ci si riferisce ad una tabulazione.

1.4.1 Classi e Interfacce Java

Lo standard nella definizione delle classi e delle interfacce Java è quello definito da Google (http://google.github.io/styleguide/javaguide.html). Ciascuna classe e ciascun metodo deve essere documentato seguendo lo stile di documentazione JavaDoc.

1.4.2 Java Servlet Pages (JSP)

Le JSP costruiscono pagine HTML in maniera dinamica che devono rispettare il formato definito nel sotto paragrafo sottostante. Devono anche attenersi alle linee guida per le classi Java definito nel sotto paragrafo soprastante. Inoltre, le JSP devono attenersi alle 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 che non contiene nient'altro;
3. Istruzioni Java che consistono in una sola riga possono contenere il tag di apertura e chiusura sulla stessa riga;

1.4.3 Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML5. Inoltre, il codice HTML deve utilizzare l'indentazione per facilitare la lettura e di conseguenza la manutenzione. Le regole di indentazione sono le seguenti:

1. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
2. Ogni tag di chiusura deve avere lo stesso livello di indentazione di quello di apertura;

1.4.4 File JavaScript

Gli script JavaScript seguono anche essi le linee guida definite dallo standard Google. Sia i file che i metodi JavaScript devono essere documentati seguendo lo stile di JavaDoc.

1.4.5 Fogli di stile CSS

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

Ogni regola CSS deve essere formattata nel seguente modo:

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;

1.4.6 Script SQL

Le istruzioni e le clausole SQL devono essere costituite da sole lettere maiuscole.

I nomi delle tabelle hanno la prima lettera maiuscola e le successive minuscole e il loro nome deve essere un sostantivo singolare.

I nomi degli attributi devono essere costituiti da sole lettere minuscole e possono contenere più parole separate da un underscore (_).

1.5 Definizioni, acronimi, abbreviazioni

Definizioni

DOM = Modella la struttura di pagine web.

Acronimi

ODD = Object Design Document

CD = Class Diagram

GUI = Graphical User Interface (Interfaccia grafica utente)

HTML = HyperText Markup Language

CSS = Cascading Style Sheet

DOM = Document Object Model

AJAX = Asynchronous JavaScript And XML

JavaEE = Java Enterprise Edition

1.6 Riferimenti

Libro:

-- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition

Autori:

-- Bernd Bruegge

-- Allen H. Dutoit

Documenti:

-- US_RAD_V1.5.pdf – Requirements Analysis Document

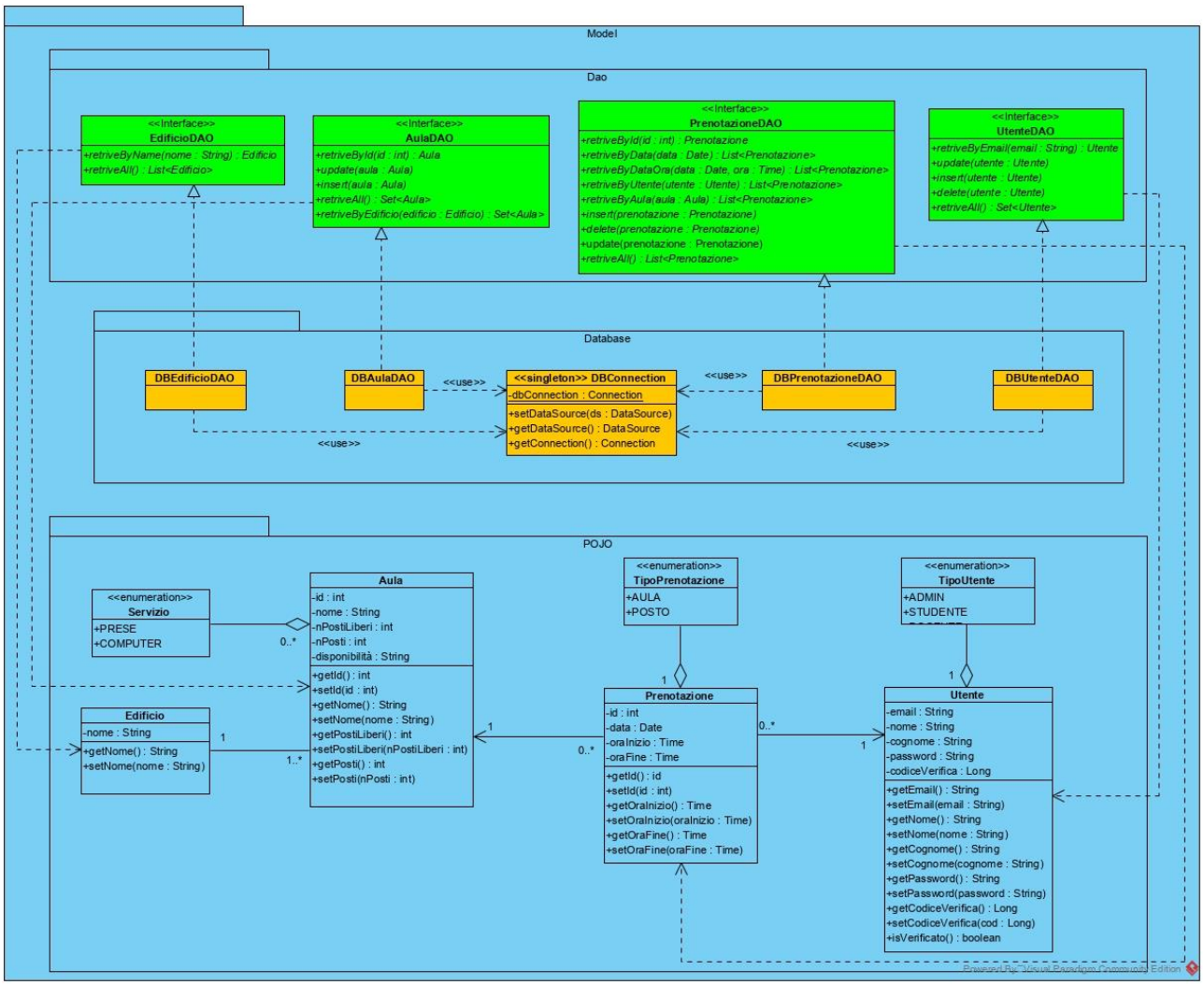
-- US_SDD_V1.2.pdf – System Design Document

2. Packages

2.1 Model

Il package Model contiene le classi che si occupano di modellare la logica di business del sistema. È suddiviso in 3 sottopackage:

- **POJO**
  Contiene le classi che modellano gli oggetti del dominio applicativo. Ciascuna classe definisce dei metodi get e set. Contiene inoltre delle classi Enum utili al mantenimento di un'alta estensibilità del codice.
- **Dao**
  Contiene le interfacce dei Data Access Object (DAO), ovvero oggetti che si relazionano con il gestore della persistenza per effettuare operazioni di tipo CRUD (Create, Read, Update, Delete)
- **Database**
  Contiene le implementazioni delle interfacce definite nel package Dao e contiene una classe DBConnection attraverso la quale avviene la connessione al database.



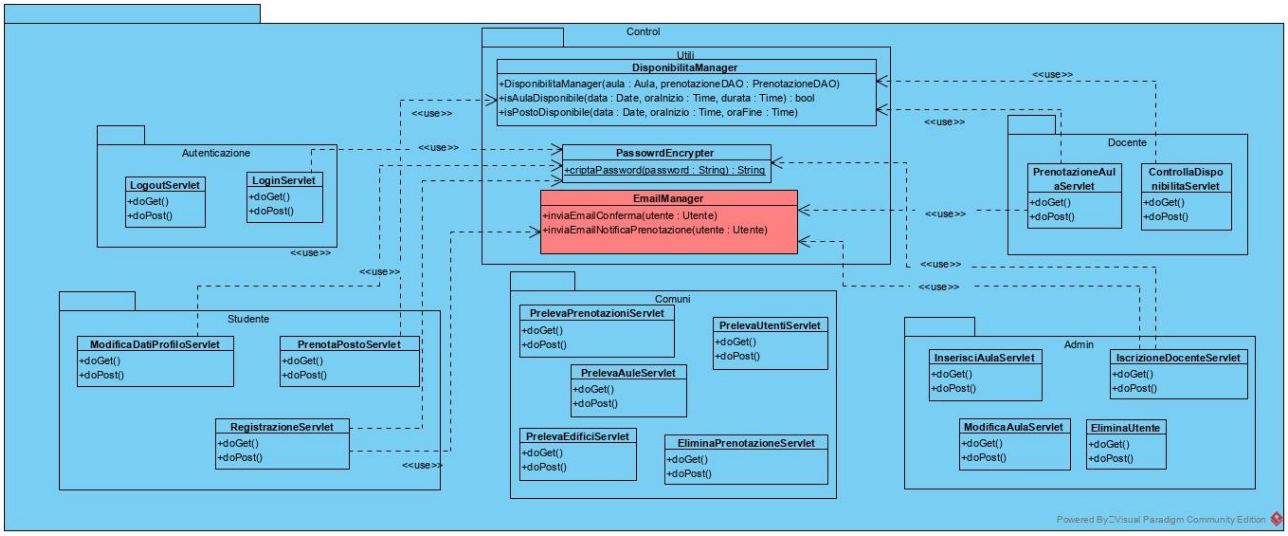
2.2 Control

Il package Control contiene le classi che gestiscono le richieste del client. Tutti i sottopackage contengono classi che estendono la classe **HttpServlet**, ad eccezione del package **Utili** che invece contiene una classe per effettuare operazioni ricorrenti.

Sottopackage:

- **Autenticazione:**
  Contiene le servlet per effettuare le operazioni di autenticazione e di log-out;
- **Studente:**
  Contiene le servlet per gestire le richieste degli studenti;
- **Comuni:**
  Contiene le servlet che gestiscono le richieste più comuni;
- **Docente:**
  Contiene le servlet per gestire le richieste dei docenti;
- **Admin:**
  Contiene le servlet per gestire le richieste dell'admin;
- **Utili:**
  Contiene classi che fungono d'ausilio alle precedenti.

Sarà presente inoltre una classe **SessionManager** all'interno del package **control**, che si occuperà di gestire le operazioni più comuni con la sessione HTTP. Al fine di non peggiorare la leggibilità del diagramma, questa classe non sarà specificata in esso.

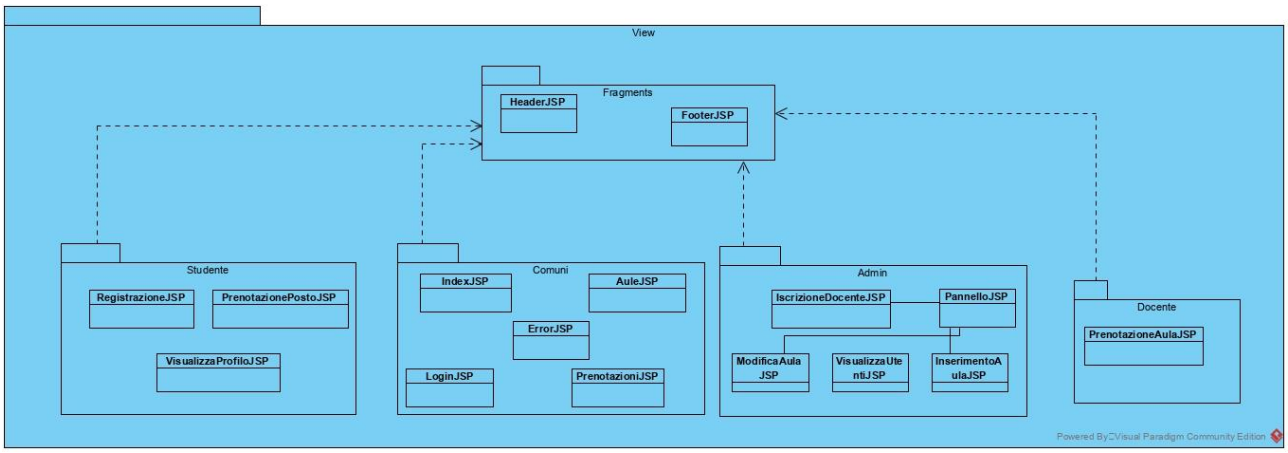


2.3 View

Il package view contiene tutte le pagine JSP tramite le quali gli utenti interagiscono con il sistema.

Il package si divide in diversi sottopackage:

- **Studente**
  Contiene le pagine JSP attraverso le quali lo studente comunica con il sistema;
- **Admin**
  Contiene le pagine JSP attraverso le quali l'admin comunica con il sistema;
- **Docente**
  Contiene le pagine JSP attraverso le quali il docente comunica con il sistema;
- **Comuni**
  Contiene pagine JSP che vengono utilizzati da tutti gli attori del sistema per effettuare le operazioni più comuni.
- **Fragments**
  Contiene frammenti di pagine JSP che sono ricorrenti in tutte le altre pagine.



3. Interfacce delle Classi

Il seguente capitolo fornirà le interfacce delle classi del package "**View**". L'interfaccia delle classi dei package restanti verrà introdotta tramite JavaDoc una volta implementate.

Nome Classe:	**IndexJSP**
Descrizione:	Questa classe è la classe da dove parte ogni operazione. Rappresenta la home page del sistema. Contiene l'elenco di tutti gli edifici del sistema.
Pre-condizione:	
Post-condizione:	
Invariante:	

Nome Classe:	**LoginJSP**
Descrizione:	Questa classe permette agli utenti di autenticarsi se sono già registrati.
Pre-condizione:	
Post-condizione:	
Invariante:	

Nome Classe:	**AuleJSP**
Descrizione:	Questa classe permette agli utenti di visualizzare le aule di un determinato edificio.
Pre-condizione:	
Post-condizione:	
Invariante:	

Nome Classe:	**PrenotazioniJSP**
Descrizione:	Questa classe permette al docente e allo studente di visualizzare le prenotazioni da loro effettuate, mentre all'admin mostra tutte le prenotazioni effettuate da tutti gli utenti.
Pre-condizione:	**doGet(request, response):** l'utente deve essere autenticato al sistema.
Post-condizione:	**doGet(request, response):** il sistema mostra le prenotazioni.
Invariante:	

Nome Classe:	**RegistrazioneJSP**
Descrizione:	Questa classe permette agli studenti non registrati di registrarsi al sistema.
Pre-condizione:	
Post-condizione:	
Invariante:	

Nome Classe:	**PrenotazionePostoJSP**
Descrizione:	Questa classe permette agli studenti di prenotare un posto in un'aula.
Pre-condizione:	**doGet(request, response):** l'utente deve essere autenticato al sistema AND l'utente deve essere di tipo STUDENTE.
Post-condizione:	**doGet(request, response):** la pagina mostra il form per prenotare un posto.
Invariante:	

Nome Classe:	**PrenotazioneAulaJSP**
Descrizione:	Questa classe permette ai docenti di prenotare un'aula.
Pre-condizione:	**doGet(request, response):** l'utente deve essere autenticato al sistema AND l'utente deve essere di tipo DOCENTE.
Post-condizione:	**doGet(request, response):** la pagina mostra il form per prenotare l'aula.
Invariante:	

Nome Classe:	**IscrizioneDocenteJSP**
Descrizione:	Questa classe permette all'admin di registrare un nuovo docente al sistema.
Pre-condizione:	**doGet(request, response):** l'utente deve essere autenticato al sistema AND l'utente deve essere di tipo ADMIN.
Post-condizione:	**doGet(request, response):** la pagina mostra il form per iscrivere un nuovo docente.
Invariante:	

Nome Classe:	**InserimentoAulaJSP**
Descrizione:	Questa classe permette all'admin di registrare un nuovo docente al sistema.
Pre-condizione:	**doGet(request, response):** l'utente deve essere autenticato al sistema AND l'utente deve essere di tipo ADMIN.
Post-condizione:	**doGet(request, response):** la pagina mostra il form per inserire una nuova aula.
Invariante:	

Nome Classe:	**InserimentoAulaJSP**
Descrizione:	Questa classe permette all'admin di registrare un nuovo docente al sistema.
Pre-condizione:	**doGet(request, response):** l'utente deve essere autenticato al sistema AND l'utente deve essere di tipo ADMIN.
Post-condizione:	**doGet(request, response):** la pagina mostra il form per modificare un'aula.
Invariante:	

Nome Classe:	**PannelloJSP**
Descrizione:	Questa classe permette all'admin di scegliere quale operazione effettuare.
Pre-condizione:	**doGet(request, response):** l'utente deve essere autenticato al sistema AND l'utente deve essere di tipo ADMIN.
Post-condizione:	**doGet(request, response):** la pagina mostra il pannello di controllo dal quale l'admin può selezionare l'operazione.
Invariante:	

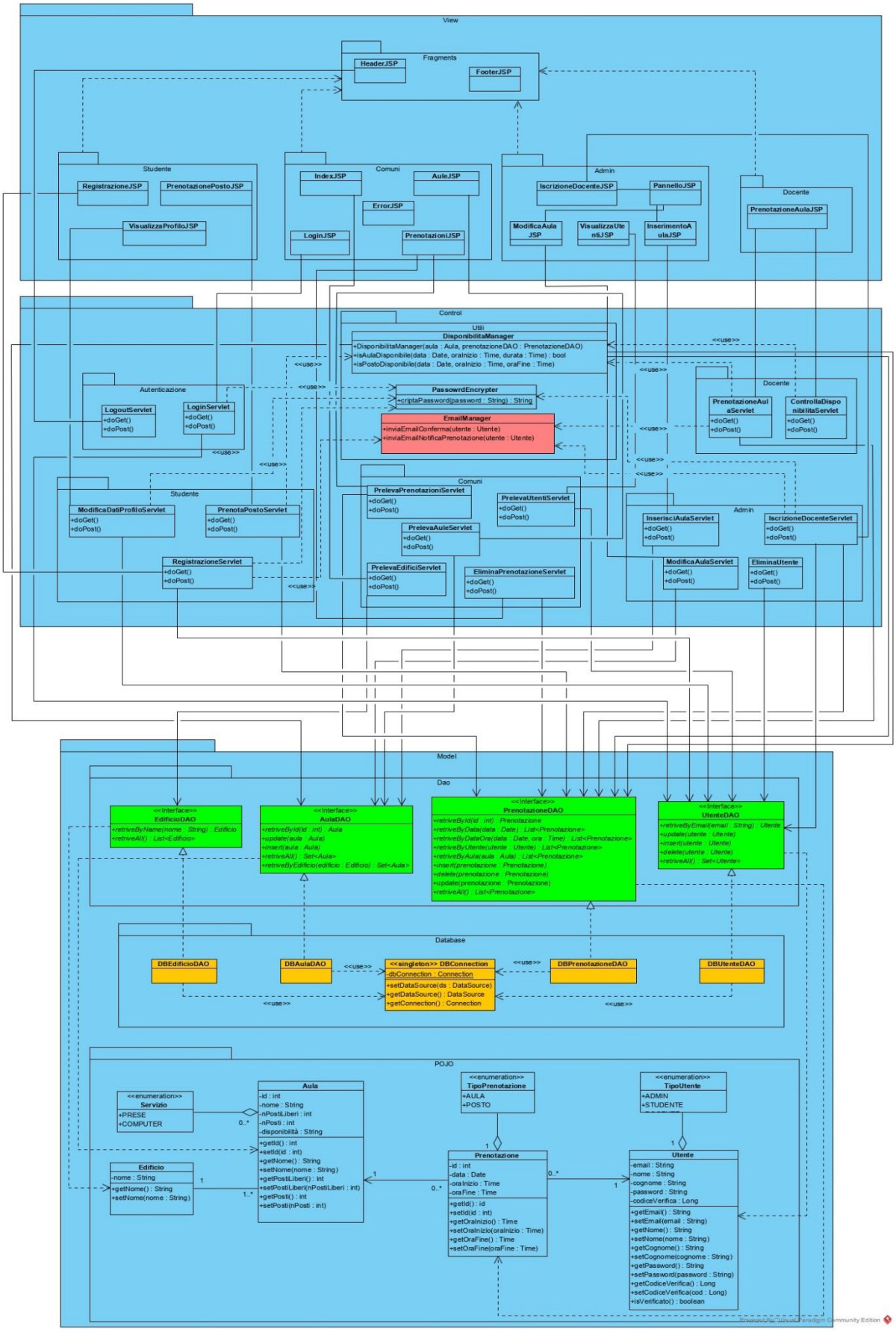
Nome Classe:	**HeaderJSP**
Descrizione:	Questa classe viene inclusa da tutte le altre pagine JSP, e rappresenta l'intestazione della pagina.
Pre-condizione:	
Post-condizione:	
Invariante:	

Nome Classe:	**FooterJSP**
Descrizione:	Questa classe viene inclusa da tutte le altre pagine JSP, e rappresenta il piè di pagina.
Pre-condizione:	
Post-condizione:	
Invariante:	

JavaDoc

La definizione delle interfacce delle classi presenti invece nei packages control e model, è possibile reperirla al seguente indirizzo (https://github.com/antonio-decaro/US_UniSeat/tree/master/documentazione/javadoc), che contiene il risultato dell'invocazione del comando javadoc dell'IDE IntelliJ.

4. Class Diagram



Legenda Colori:

DAO Pattern

Singleton Pattern

Proxy Pattern

5 Glossario

Standard di sicurezza minimo:

1. La password di ciascun utente deve essere criptata prima di essere memorizzata nel database;
2. Le pagine devono essere accessibili solo agli utenti che hanno il permesso di visualizzarla;
3. Le query al database devono essere sempre prima controllate per evitare attacchi di natura "SQL Injection";