PROGETTO COLLEGAMENTI DEL CORSO DI INGEGNERIA DEL SOFTWARE

COLLEGAMENTI

Renato Milano 0512103100

Ciro Valerio Cerchia 0512103698

Francesco Buonomo 0512103794

O.D.D.

Data	Versione	Descrizione	Autori
2/06/19		Stesura punti 1 e 2	Buonomo, Cerchia, Milano
3/06/19		Stesura punto 3	Buonomo, Cerchia, Milano
4/06/19		Stesura punto 4	Milano

1.Introduzione

1.1 Trade-Off

Scalabilità vs Prestazioni

Non ci aspettiamo di avere molti utenti connessi contemporaneamente al sistema. La maggior parte delle operazioni che gli utenti possono fare sono semplici e rapide e non è quasi mai necessario un utilizzo prolungato del sistema. Ci siamo quindi concentrati sulle prestazioni senza però rendere il nostro sistema poco scalabile.

Tempi vs Costi

Per l'implementazione del sistema si è scelto di utilizzare Java Enterprise nonostante la poca esperienza del team. L'utilizzo di queste specifiche ci permette di risparmiare moltissimo tempo sulla scrittura di codice che verrà generato automaticamente. Questo risparmio è vantaggioso rispetto al tempo necessario all'apprendimento.

Comprensibilità vs Costi

Si è scelto di utilizzare la lingua inglese per l'implementazione e la documentazione. Questo per rendere comprensibile il codice anche a chi non ha partecipato allo sviluppo. L'intera documentazione e i commenti nel codice, in inglese, facilitano la comprensione. Migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

Tempi di risposta vs Memoria

Si è deciso di utilizzare un DB per il mantenimento dei dati persistenti. Per poter garantire operazioni rapide agli utenti si è scelto di utilizzare più memoria per la conservazione di informazioni.

Usabilità vs Costi

Si è deciso di aggiungere costi per rendere il sistema quanto più usabile possibile dagli utenti. La navigazione sarà facile ed intuitiva e le operazioni saranno quanto più immediate possibili.

1.2 Commercial Off-The-Shelf

Per il progetto software che si intende realizzare, utilizzeremo una componente off-the-shelf, ovvero componenti software disponibili sul mercato per facilitare la creazione del progetto.

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

Il framework che andremo ad utilizzare è JQuery, il quale è open source e contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web.

Inoltre, presenta modelli di progettazione basati su JavaScript, sia per la tipografia, sia per le varie componenti dell'interfaccia.

La libreria può essere scaricata dal sito ufficiale www.jquery.com in cui è anche presente un ampia documentazione per facilitare l'utilizzo della stessa.

1.3 Linee Guida per la Documentazione dell'Interfaccia

Il codice prodotto dovrà rispettare gli Standards sul codice di Google per quanto concerne la scrittura di codice sorgente in Java.

1.3.1 Nome dei File

Il nome del file sorgente corrisponde al nome della classe contenuta al suo interno con distinzione tra maiuscole e minuscole.

Al nome della classe possono essere aggiunti due suffissi:

- .java per il file sorgente;
- .class per il Bytecode del file.

1.3.2 Struttura dei File Sorgente

Ogni file sorgente contiene esattamente una sola classe pubblica o interfaccia. Ogni sezione del File deve essere separata con una linea vuota. La struttura interna del File è la seguente:

- *Descrizione Classe*: Tutti i file devono iniziare con un commento nel quale è esplicitato il nome della classe, una breve descrizione, l'autore, la versione e la data di creazione e infine le informazioni di Copyright.

```
/**
 * Nome della Classe
 *
 * Descrizione
 *
 * @author Autore
 * @version Versione
 * @since Data di rilascio
 *
 * 2019 - Copyright by Collegamenti
 */
```

- *Dichiarazione del Package e degli Import*: La prima istruzione dopo la descrizione della classe deve essere la dichiarazione del Package, successivamente è possibile dichiarare gli import necessari. Le dichiarazioni del Package e degli Import sono separati da una linea vuota. Gli Import devono essere specifici e non può essere usata la dichiarazione tramite Wildcard (*). Infine gli Import sono ordinati come segue:
    - Tutti gli Import Statici in un singolo blocco;
    - Tutti gli Import Non Statici in un singolo blocco.

  Ogni blocco è separato da una linea vuota.

```
package nomepackage;

import static java.lang.Math.PI;
import static java.lang.Math.pow;

import java.util.List;
import java.io.Serializable;
```

- *Dichiarazione della Classe o dell'Interfaccia*: Il file sorgente deve contenere una sola Classe o una sola Interfaccia. Gli elementi contenuti al loro interno devono essere ordinati nel seguente modo:
    - Dichiarazione Variabili Statiche della Classe: Prima le variabili di classe public, poi le variabili protected e infine quelle private;

- Dichiarazioni Variabili di istanza: Prima le variabili di classe public, poi le variabili protected e infine quelle private;
- Costruttori;
- Metodi: Questi metodi devono essere raggruppati in base alla loro funzionalità piuttosto che in base a regole di visibilità o accessibilità. I metodi con nomi simili devono essere vicini tra loro.

1.3.3 Formattazione

L'indentazione del codice deve avvenire tramite l'utilizzo degli spazi e non tramite l'utilizzo delle tabulazioni. Ogni volta un nuovo blocco o un costrutto simile sono aperti, l'indentazione aumenta di due spazi. Quando il blocco termina, l'indentazione ritorna al livello di indentazione precedente. Il livello di indentazione deve essere applicato sia al codice che ai commenti del blocco.

Il codice Java ha un limite per riga di cento caratteri. Ogni riga che non rispetta questo limite deve essere divisa in più linee. Questo limite non viene rispettato dalle eccezioni mostrate di seguito:

- Linee dove non è possibile rispettare il limite imposto come ad esempio gli URL molto lunghi;
- Le dichiarazioni dei Package e degli Import;
- Comandi presenti in un commento che possono essere copiati e incollati nella Shell.

Lo scopo delle interruzioni è di rendere il codice più comprensibile e non di adattare il codice nel minor numero di linee possibile.

Un altro importante criterio da utilizzare per rendere il codice chiaro è l'utilizzo degli spazi o delle linee vuote:

- *Linee vuote*: Vanno inserite delle linee vuote nei seguenti casi:
  - Tra membri o inizializzazioni di una classe: campi, costruttori, metodi, classi innestate, inizializzazioni statiche e inizializzazioni di istanze;
  - Tra dichiarazioni se si vuole creare delle divisioni logiche;
  - Tra altre parti del file specificato in precedenza.
- *Spazi*: Devono essere utilizzati, se non specificati in altri casi, nel seguente modo:

- Per separare qualsiasi parola riservata, come if, for o catch, da una parentesi tonda aperta che la segue nella stessa linea;
- Per separare qualsiasi parola riservata, come else o catch, da una parentesi graffa aperta che la precede in quella linea;
- Prima di una parentesi graffa aperta, con due eccezioni:
  - @QualcheAnnotazione({a, b}) (non è usato nessuno spazio);
  - String[][] x = {{"stringa"}} (non è presente nessuno spazio tra le parentesi graffe).
- Su entrambi i lati di un operatore binario o ternario.

  E' applicato anche per i seguenti simboli:
  - Il colon (:)nella dichiarazione di un for o foreach;
  - La freccia in una espressione lambda: (String str) -> str.length();
  - Non è applicata al separatore punto che è scritto come object.toString();
- Dopo ,:; oppure le parentesi chiuse di un cast;
- Su entrambi i lati di un doppio slash che inizia un commento su una sola linea;
- Tra il tipo e la dichiarazione di una variabile: List<String> list;

1.3.4 Dichiarazioni

Ogni dichiarazione di variabile deve definire una sola variabile. Questo favorisce l'uso di commenti su una singola linea.

```
String name;     // Commento sul nome
String surname;  // Commento sul cognome
```

Le variabili locali devono essere dichiarate vicino al punto dove vengono utilizzate per la prima volta così da minimizzare il loro scope. Le variabili locali devono essere inizializzate durante la loro dichiarazione o immediatamente dopo.

La dichiarazione di un array deve avere le parentesi quadre vicino al tipo della variabile e non vicino la variabile stessa. Inoltre, l'inizializzazione di un array deve essere formattata secondo uno dei seguenti modi:

```
int array = new int[] {0, 1, 2, 3};
int array = new int[]{
    0, 1, 2, 3
};
int array = new int[]{
    0,
    1,
    2,
    3
};
int array = new int[]{
    0, 1,
    2, 3
};
int array = new int[]
    {0, 1, 2, 3};
```

1.3.5 Nomenclatura

Nella scelta dei nomi non bisogna usare prefissi o suffissi speciali come ad esempio name_, nName, s_name oppure kName.

Di seguito sono mostrati i vincoli che ogni elemento deve rispettare:

- *Package*: I nome dei Package sono tutti in minuscolo, con parole semplici concatenate insieme senza usare l'underscore. Ad esempio, com.example.modelsystem e non com.example.ModelSystem oppure com.examplemodel_system.;
- *Classi*: I nomi delle Classi devono essere scritti in UpperCamelCase. Sono tipicamente sostantivi o frasi con sostantivi. I nomi delle Interfacce, invece, possono essere sia sostantivi che aggettivi. Le classi di Test hanno il nome della classe da testare con la parola Test alla fine;
- *Metodi*: I nomi dei Metodi devono essere scritti in lowerCamelCase. Sono tipicamente verbi o frasi con verbi. L'underscore può apparire nei test dei metodi di JUnit per separare logicamente i componenti del nome. Un pattern è test<MetododaTestare >_<stato>, per esempio **testPop_emptyStack**;
- *Costanti*: I nomi delle Costanti devono essere completamente in maiuscolo con le parole separate da underscores. Le costanti sono campi static final i cui contenuti sono immutabili e i metodi non hanno effetti collaterali su di esse. Sono incluse primitive, stringhe, tipi immutabili e collezioni di tipi immutabili. Se uno qualsiasi

degli stati dell'istanza può cambiare allora non è una costante. Sono tipicamente sostantivi o frasi di sostantivi;

- *Campi non costanti*: I nomi dei campi che non sono costanti, statici e non, sono scritti in lowerCamelCase. Questi nomi sono tipicamente sostantivi e frasi con sostantivi.
- *Parametri*: I nomi dei parametri sono scritti in lowerCamelCase. I nomi dei parametri formati da un carattere devono essere evitati.
- *Variabili Locali*: I nomi delle variabili locali sono scritte in lowerCamelCase. Le variabili locali anche se sono final e immutabili non sono considerate delle costanti.

1.3.6 Javadoc

I programmi Java possono avere due tipi di commenti: commenti di implementazione e commenti di documentazione. I commenti di implementazione sono delimitati da /*...*/ e //.

I commenti di documentazione sono esclusivi di Java e sono delimitati da /**...*/. I commenti di documentazione possono essere estratti in file HTML utilizzando lo strumento Javadoc.

I commenti di implementazione sono dei mezzi per commentare il codice o per commentare una particolare implementazione. I commenti di documentazione vengono utilizzati per descrivere la specifica del codice da una prospettiva non implementativa, per essere letti da sviluppatori che non devono necessariamente avere il codice a portata di mano.

I commenti dovrebbero essere usati per dare una panoramica del codice e per fornire informazioni aggiuntive che non sono prontamente disponibili nel codice stesso. I commenti devono contenere solo informazioni rilevanti per leggere e comprendere il programma. Ad esempio, informazioni su come il package corrispondente è costruito o in quale directory risiede non dovrebbero essere incluse in un commento.

I programmi possono avere tre tipi di commenti di implementazione:

- *Commenti di blocco*: Sono usati per fornire descrizioni di file, metodi, strutture dati e algoritmi. Un commento di blocco dovrebbe essere preceduto da una linea bianca di separazione dal codice.

```
/*
 *  Qui c'è un commento di blocco
 */
```

- *Commenti a linea singola*: Sono brevi commenti che possono apparire su una singola linea di codice ed indentati al livello del codice che seguono. Un commento a linea singola deve essere preceduto da una linea bianca.

```
if(condition){

    /* Gestisce la condizione */
    ...
}
```

- *Commenti di fine linea*: Il delimitatore di commento // può commentare una linea completa o una parte di essa. Non dovrebbe essere usato su più linee consecutive per commenti testuali. Comunque, può essere usato su più linee consecutive per commentare sezioni del codice. Seguono tre esempi dei tre stili:

```
if(i>0){

 // Fa qualcosa
 ...
} else {
 i--; //Spiega il perchè qui
}

//if(x==0){
//
// Fa altro
//}
```

I commenti di documentazione descrivono classi Java, Interfacce, Costruttori, Metodi e Campi. Ogni commento di documentazione è compreso all'interno dei delimitatori di commento /**...*/, con un commento per ogni classe, interfaccia o membro. Questo commento deve apparire solo prima della dichiarazione:

```
/**
 *  La classe Esempio fornisce...
 *
 */

public class Esempio {

    ....

}
```

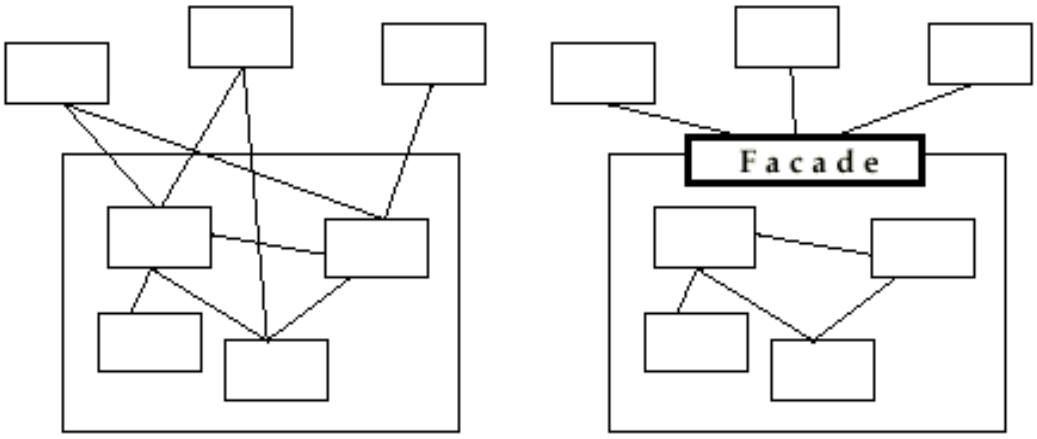
I commenti di documentazione contengono dei tag che devono essere visualizzati nell'ordine @param, @return, @throws, @deprecated e questi quattro tipi non devono mai essere presenti in una descrizione vuota.

```
/**
 *  Verifica l'equivalenza tra due oggetti.
 * Ritorna un boolean che indica se l'oggetto in cui mi trovo
 * è equivalente all'oggetto specificato come parametro.
 *
 * @author          Marco Rossi
 * @param   obj     L'oggetto che viene confrontato
 * @return  true    se due oggetti sono equivalenti
 *          false   altrimenti
 * @see             java.util
 *
 */

public boolean equals(Object obj) {
    return (this == obj);
}
```

1.4 Design Pattern

1.4.1 Facade Pattern



Problema: Fornisce una singola interfaccia per accedere ad oggetti di uno stesso sottosistema.

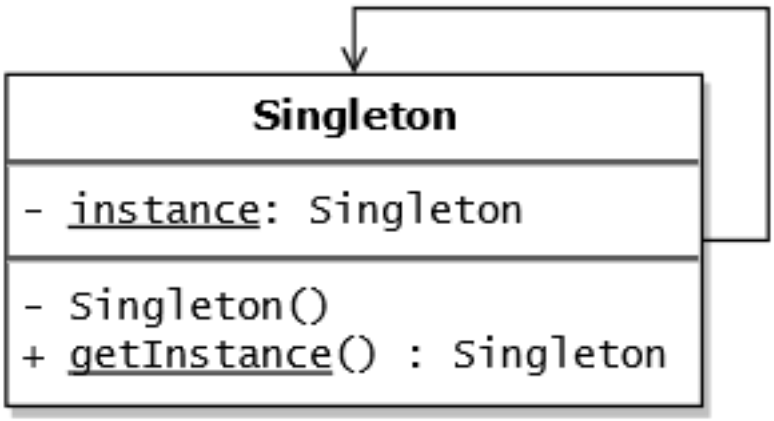
Soluzione: Creazione di una interfaccia che fornisce tutti i servizi di un sottosistema

Conseguenze:

- Fornisce interfacce ad alto livello che rendono facile l'utilizzo di sottosistemi
- Riduce il coupling tra client e componenti dei sottositemi

Il design pattern Facade fa parte dei pattern strutturali e Collegamenti lo utilizza permettendo al client di interagire con il sistema attraverso un'unica interfaccia. Il client accede, tramite quest'ultima, a tutte le funzionalità e fornisce una visione di insieme come un'architettura chiusa.

1.4.2 Singleton Pattern



Problema: Creazione di una e una sola istanza e di un punto di accesso globale a tale istanza.

Soluzione: Creazione di una classe che si occupa di istanziare un oggetto e di fornire accesso a quel singolo oggetto

Vincoli:

- non consente relazioni di ereditarietà
- impedisce la sostituibilità di oggetti legati da un vincolo di parentela

Il design pattern Singleton fa parte dei pattern creazionali e si basa sull'idea di restituire l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo. Collegamenti lo utilizza per istanziare la connessione con la base di dati.

1.5 Definizioni, Acronimi e Abbreviazioni

- *ODD:* Object Design Document

1.6 Riferimenti

- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (2nd edition), Prentice-Hall, 2003.

2. Packages

In questa sezione viene rappresentata la suddivisione in package del sistema in base all'architettura scelta.

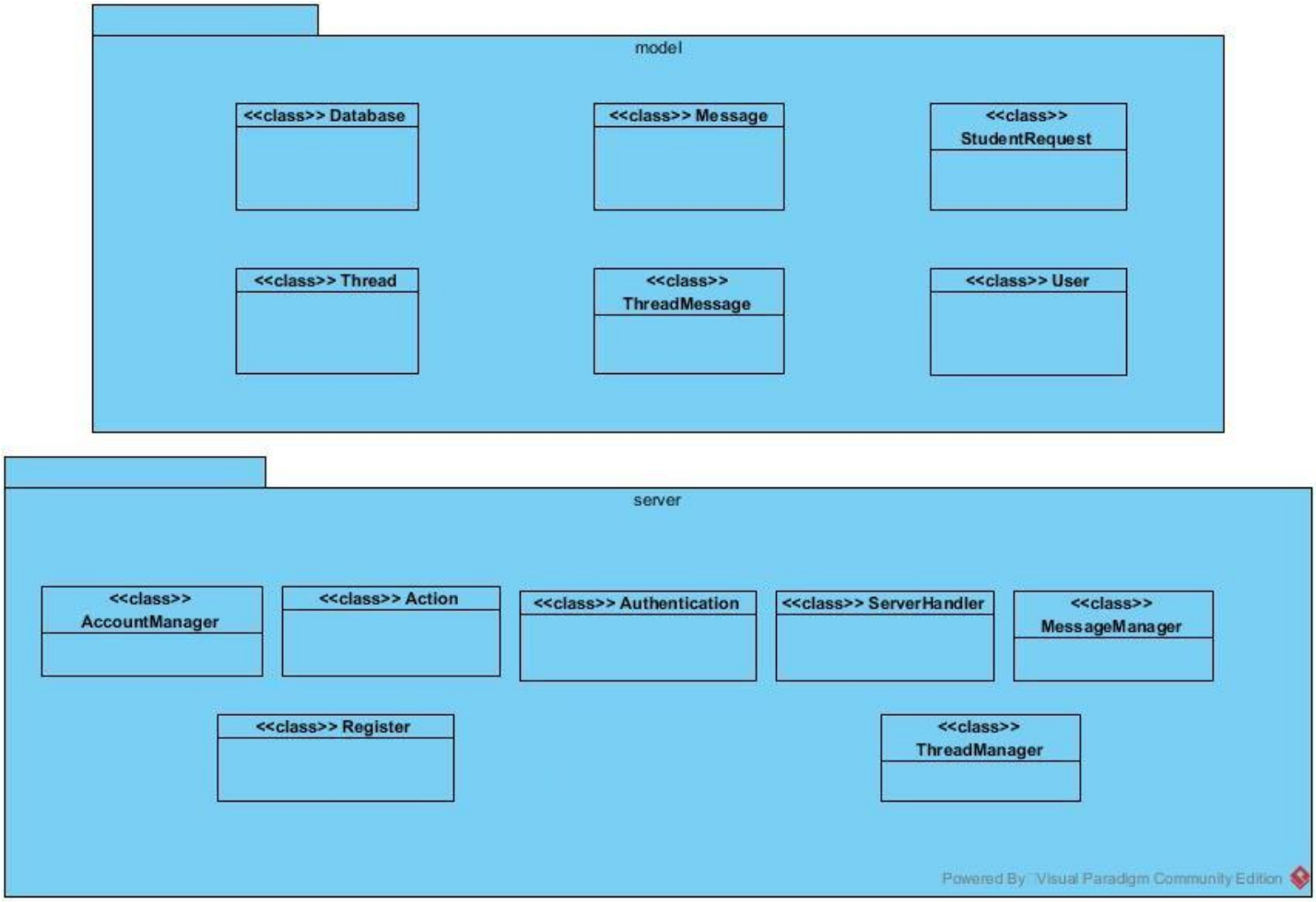
Si è scelto di dividere il sistema in due "progetti":

- CollegamentiServer, nel quale verranno inserite le classi relative al Model e al Control;
- CollegamentiClient, nel quale verranno inserite le classi e i file relativi all'interfaccia grafica View.

Visto che il sistema che si sta progettando è un sistema web si è scelta la seguente divisione.

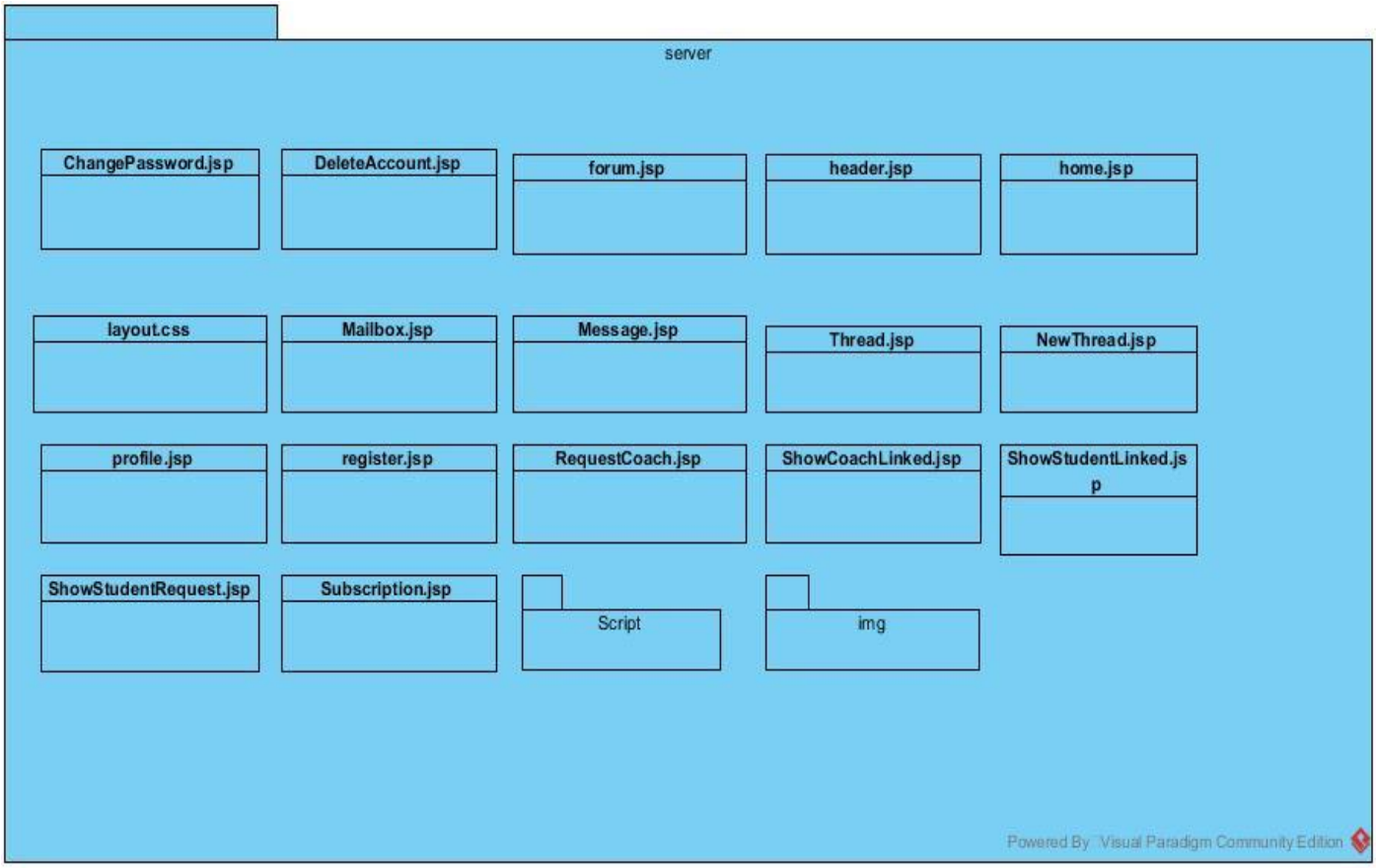
2.1 CollegamentiServer

Suddiviso in CollegamentiServer.model per le classi che gestiscono i singoli concetti e CollegamentiServer.service per le classi che gestiscono la logica computazionale dei vari sottosistemi del sito.



2.2 CollegamentiClient

CollegamentiClient comprende i file per la visualizzazione delle pagine , gli script per il tool JQuery e le immagini.



CLASSE	DESCRIZIONE
User	Descrive un Utente registrato sulla piattaforma
Messaggio	Descrive un messaggio tra Studente e Coach
StudentRequest	Descrive la richiesta di uno Studente per avere un Coach
Thread	Descrive una discussione aperta nella sezione forum
ThreadMessagge	Descrive una risposta data in un Thread
AccountManager	Gestisce le operazioni che riguardano l'account dell'utente

Authentication	Gestisce le operazioni che riguardano l'autenticazione di un utente.
Register	Gestisce le operazioni che riguardano la registrazione di un Utente alla piattaforma
MessageManager	Gestisce le operazioni che riguardano lo scambio di messaggi tra utenti.
ThreadManager	Gestisce le operazioni che riguardano la sezione forum
Action	Interfaccia per la creazione del Facade Pattern
Database	Descrive il collegamento tra sistema e DBMS
ServerHandler	Descrive la gestione delle richieste HTTP

PACKAGE	DESCRIZIONE
Collegamenti.src	Contiene le classi relative al Model e al Control (entity, service)
.model	Contiene le classi del sistema Collegamenti
.service	Contiene tutti i servizi relativi ai model
Client	Contiene le classi e i file relativi all'interfaccia grafica View
Srcipt	Contieni gli Script JavaScript e JQuery
img	Contiene le immagini utilizzate dal sistema

3. Interfacce delle Classi

3.1 Gestione Registrazione

3.1.1 Register

NOME CLASSE	Register
DESCRIZIONE	Questa classe fornisce i servizi per la registrazione al sistema e la convalida di un Coach.
METODI	+ perform(HttpServletReguest request, HttpServletResponse response) : HttpSession - RegisterStudent(String Email, String Password, String Name, String Surname, int Type, String Qualification, String Category):void - RegisterStudent(String Email, String Password, String Name, String Surname, int Type, String Qualification, String Category):void - AcceptCoach(String Email): void - RejectCoach(String Email): void - getCategory():String

NOME CLASSE	Register
NOME METODO	- RegisterStudent(String Email, String Password, String Name, String Surname, int Type, String Qualification, String Category):void
DESCRIZIONE METODO	Questo metodo registra un nuovo Utente di tipo Studente al sistema.
PRE-CONDIZIONI	**context** Register :: RegisterStudent(Email,Password,Name,Surname,Type,Qualification,Category) **pre:** !exist(Email)

POST-CONDIZIONI	**context** Register :: (RegisterStudent(Email,Password,Name,Surname,Type,Qualification,Category) **post:** exist(Email)
NOME CLASSE	Register
NOME METODO	- RegisterCoach(String Email, String Password, String Name, String Surname, int Type, String Qualification, String Category):void
DESCRIZIONE METODO	Questo registra un nuovo Utente di tipo Coach al sistema.
PRE-CONDIZIONI	context Register :: RegisterCoach(Email,Password,Name,Surname,Type,Qualification,Category) pre: !exist(Email)
POST-CONDIZIONI	context Register :: (RegisterCoach(Email,Password,Name,Surname,Type,Qualification,Category) post: exist(Email)

NOME CLASSE	Register
NOME METODO	- AcceptCoach(String Email):void
DESCRIZIONE METODO	Questo metodo permette ad un amministratore di validare un Utente di tipo Coach al sistema.
PRE-CONDIZIONI	context Register :: AcceptCoach(Email) pre: exist(Email)
POST-CONDIZIONI	context Register :: AcceptCoach(Email) post: User.Type=1

NOME CLASSE	Register
NOME METODO	- RejectCoach(String Email):void

DESCRIZIONE METODO	Questo metodo permette ad un amministratore di scartare un Utente di tipo Coach dal sistema.
PRE-CONDIZIONI	context Register :: RejectCoach(Email) pre: exist(Email)
POST-CONDIZIONI	context Register :: RejectCoach(Email) post: !exist(Email)

3.2 Gestione Autenticazione

3.2.1 Authentication

NOME CLASSE	Authentication
DESCRIZIONE	Questa classe fornisce i servizi per l'accesso al proprio Account.
METODI	+ perform(HttpServletReguest request, HttpServletResponse response) : HttpSession - Login(String Email, String Password):User - Logout(): void

NOME CLASSE	Authentication
NOME METODO	- Login(String Email,String Password):User
DESCRIZIONE METODO	Questo metodo permette ad un Utente di accedere al sistema.
PRE-CONDIZIONI	context Authentication :: Login(Email,Password) pre: exist(Email)
POST-CONDIZIONI	/

NOME CLASSE	Authentication

NOME METODO	- Logout():void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di disconnettersi dal sistema.
PRE-CONDIZIONI	context Authentication :: Logout(Email,Password) pre: exist(Email)
POST-CONDIZIONI	/

3.3 Gestione Account

3.3.1 AccountManager

NOME CLASSE	AccountManager
DESCRIZIONE	Questa classe fornisce i servizi per la modifica del proprio Account , per la richiesta di coach e per la richiesta di eventuali Coach.
METODI	+ perform(HttpServletReguest request, HttpServletResponse response) : HttpSession - NewCoachRequest(String Email, String Category,String Description):void - AcceptStudent(String Email): void - NewSubscription(String Email): void - ChangePasword(String Email,String Password): User

NOME CLASSE	AccountManager
NOME METODO	- NewCoachRequest(String Email, String Category,String Description):void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di richiedere un Coach iscritto al sistema.

PRE-CONDIZIONI	context AccountManager :: NewCoachRequest(Email,Password) pre: exist(Email)
POST-CONDIZIONI	/

NOME CLASSE	AccountManager
NOME METODO	- AcceptStudent():void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di accettare uno Studente registrato al sistema.
PRE-CONDIZIONI	context AccountManager :: AcceptStudent(Email) pre: exist(Email)
POST-CONDIZIONI	/

NOME CLASSE	AccountManager
NOME METODO	- NewSubscription(String Email):void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di abbonarsi al sistema.
PRE-CONDIZIONI	context AccountManager :: NewSubscription(Email) pre: exist(Email)
POST-CONDIZIONI	/

NOME CLASSE	AccountManager
NOME METODO	- ChangePassword(String Email, String Password):User

DESCRIZIONE METODO	Questo metodo permette ad un Utente di cambiare la propria password per accedere al sistema.
PRE-CONDIZIONI	context AccountManager :: ChangePassword(String Email,String Password) pre: exist(Email)
POST-CONDIZIONI	context AccountManager :: :: ChangePassword(String Email,String Password) post: User.Password = Password

3.4 Gestione Comunicazione

3.4.1 MessageManager

NOME CLASSE	MessageManager
DESCRIZIONE	Questa classe fornisce i servizi per l'invio di Messaggi ad un altro utente.
METODI	+ perform(HttpServletReguest request, HttpServletResponse response) : HttpSession - SendMessage(String Object,String Reciever,String Sender): void -DeleteConversation(String Email):void

NOME CLASSE	MessageManager
NOME METODO	- SendMessage(String Object, String Reciever ,String Sender):void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di inviare un messaggio ad un altro utente del sistema.
PRE-CONDIZIONI	context MessageManager :: SendMessage(String Object, String Reciever ,String Sender) pre: exist(Sender)&&exist(Reciever)
POST-CONDIZIONI	/

NOME CLASSE	MessageManager
NOME METODO	- DeleteConversation(String Email):void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di eliminare l'associazione Studente-Coach tramite messaggi.
PRE-CONDIZIONI	context MessageManager :: SendMessage(String Object, String Reciever ,String Sender) pre: exist(Sender)&&exist(Reciever)
POST-CONDIZIONI	/

3.5 Gestione Forum

3.5.1 ThreadManager

NOME CLASSE	ThreadManager
DESCRIZIONE	Questa classe fornisce i servizi per la gestione di un Thread da parte di un Utente.
METODI	+ perform(HttpServletReguest request, HttpServletResponse response) : HttpSession - NewThread(String Title,String Description,String Category,String Creator): void -NewThreadAnswer(String Answer,String Email, Integer ID): void -AddVotes(Integer ID, Integer Order,Integer Votes):void -DeleteThread(Integer ID):void

NOME CLASSE	ThreadManager
NOME METODO	- NewThread(String Title,String Description,String Category,String Creator): void

DESCRIZIONE METODO	Questo metodo permette ad un Utente di aprire un Thread nella sezione Forum del sito.
PRE-CONDIZIONI	context ThreadManager :: NewThread(String Title,String Description,String Category,String Creator): pre: exist(Creator)
POST-CONDIZIONI	/

NOME CLASSE	ThreadManager
NOME METODO	-NewThreadAnswer(String Answer,String Email, Integer ID): void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di partecipare ad un Thread aperto nella sezione Forum del sito.
PRE-CONDIZIONI	context ThreadManager :: NewThreadAnswer(String Answer,String Email, Integer ID): pre: exist(Creator)
POST-CONDIZIONI	/

NOME CLASSE	ThreadManager
NOME METODO	- AddVotes(Integer ID, Integer Order,Integer Votes):void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di votare positivamente una risposta in un Thread aperto nella sezione Forum del sito.
PRE-CONDIZIONI	context ThreadManager :: -AddVotes(Integer ID, Integer Order,Integer Votes): pre: exist(Order)&& exist(ID)

POST-CONDIZIONI	/

NOME CLASSE	ThreadManager
NOME METODO	- DeleteThread(Integer ID):void
DESCRIZIONE METODO	Questo metodo permette ad un Utente di chiudere un Thread da lui aperto nella sezione Forum del sito.
PRE-CONDIZIONI	context ThreadManager :: DeleteThread(Integer ID): pre: exist(ID)
POST-CONDIZIONI	context ThreadManager :: DeleteThread(Integer ID): post: !exist(ID)

4.1 Class Diagram