UNIVERSITÀ DEGLI STUDI
DI SALERNO

Ingegneria Del Software

Object Design Document

Progetto: GetThatSound.it

Coordinatore del progetto:

- *Andrea De Lucia*

Partecipanti:

- *Adriano Molli* *0512104348*
- *Pierluigi Epifania* *0512104714*

Sommario

1.Introduzione

1.1. Object design trade - offs

[Interfaccia vs easy – use]

L'interfaccia di GetThatSound.it sarà minimale, intuitiva e non complessa al fine di garantire una facilità d'uso anche ad utenti meno esperti, così da non richiedere grandi skills per utilizzare il sistema. Il sistema userà font di una grandezza tale da facilitare la lettura ad un vasto insieme di utenti, la navigazione è agevole e concreta. Inoltre l'interfaccia si adatta a qualsiasi dispositivo l'utente usi per accedere al sito.

1.2. Linee guida documentazione interfaccia

Prima dell'implementazione vera e propria, risulta opportuno delineare delle linee guida o "regole" che si devono rispettare per la scrittura del codice.

[Spostamento di linee]

Quando un'espressione supera la lunghezza della linea, occorre spezzarla secondo i seguenti principi generali:

- Interrompere la linea dopo una virgola;
- Interrompere la linea prima di un operatore;
- Allineare la nuova linea con l'inizio dell'espressione nella linea precedente.

[Indentazione]

Viene utilizzata l'indentazione del codice, soprattutto nei cicli FOR e WHILE, nel blocco di istruzioni delle condizioni IF, nel codice HTML se qualche tag è incluso in un altro tag.

Es1.

```
For (i=0; i<10; i++) {
        istruzione;
}
```

Es2.

```
<html>
        <body>
```

```
        </body>
    <html>
```

[Parentesi]

Qualora dovesse esserci anche una sola istruzione, che viene eseguita in seguito al soddisfacimento di una condizione IF o all'interno di un ciclo, deve essere racchiusa tra parentesi graffe.

1.3. Definizioni, acronimi e abbreviazioni

RAD: Requirements Analysis Document.

SDD: System Design Document.

ODD: Object Design Document.

DB: Database.

Gestore brani: utente del sito che gestisce i brani che vengono caricati.

Amministratore: amministratore del sito. Gestisce tutti gli account e crea nuovi gestori dei brani.

HTML: Linguaggio di Mark-up per pagine web.

GetThatSound.it: la piattaforma per la condivisione di brani che si intende sviluppare.

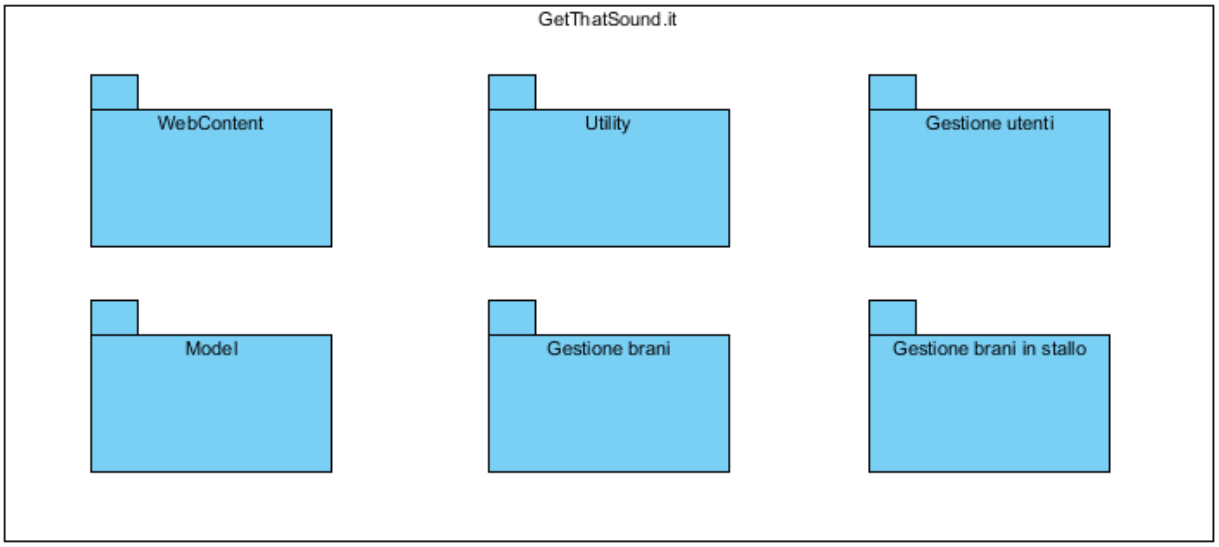
1.4. Riferimenti

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

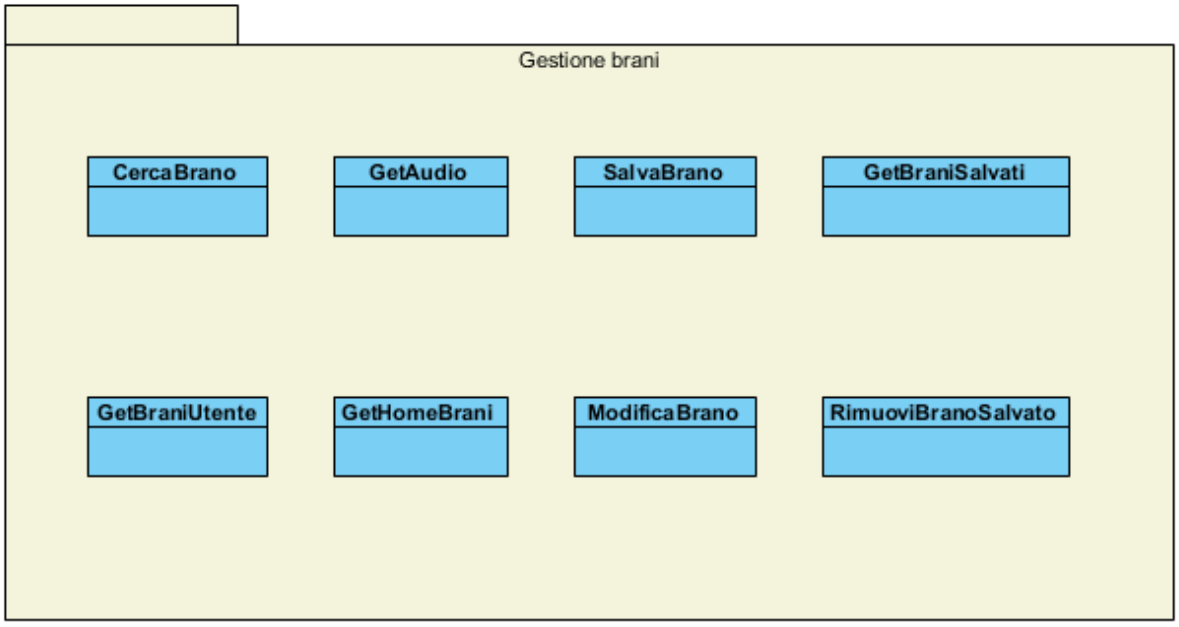
2.Packages

In questa sezione viene rappresentata la suddivisione in package del sistema in base all'architettura scelta. I packages principali sono:

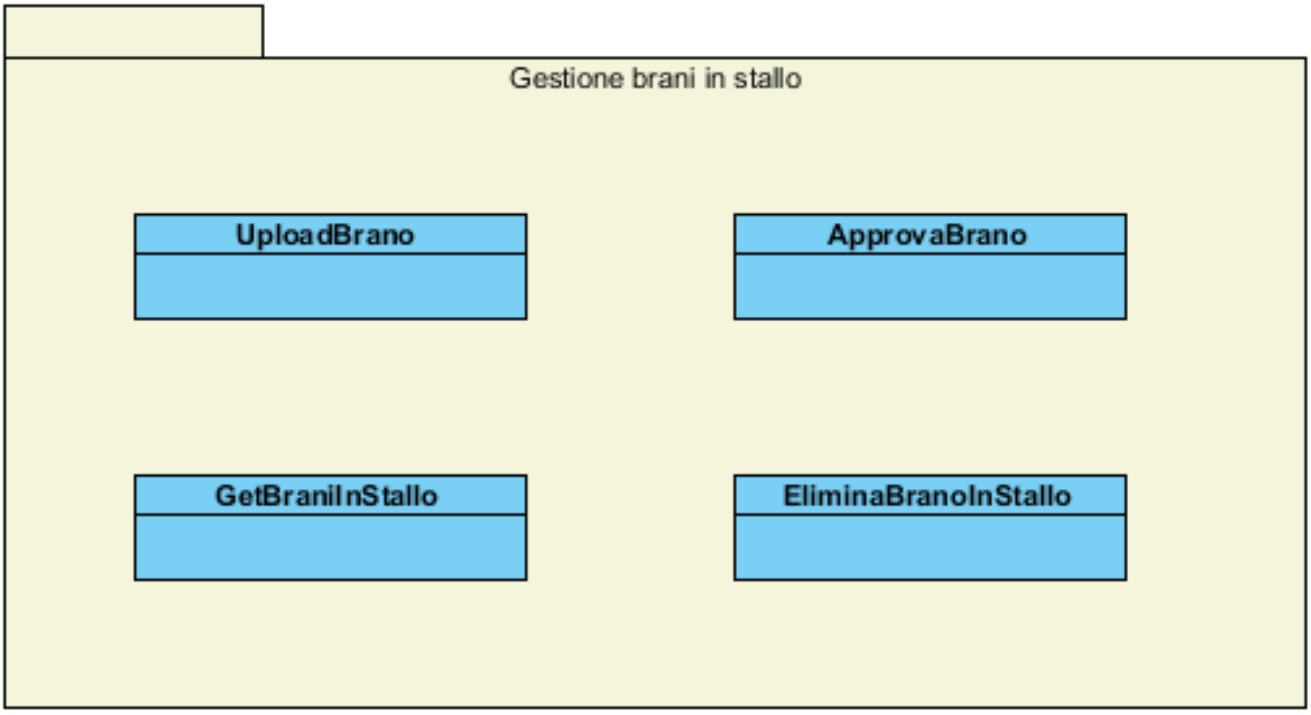
- Gestione brani
- Gestione utenti
- Gestione brani in stallo
- Model
- Utility
- WebContent



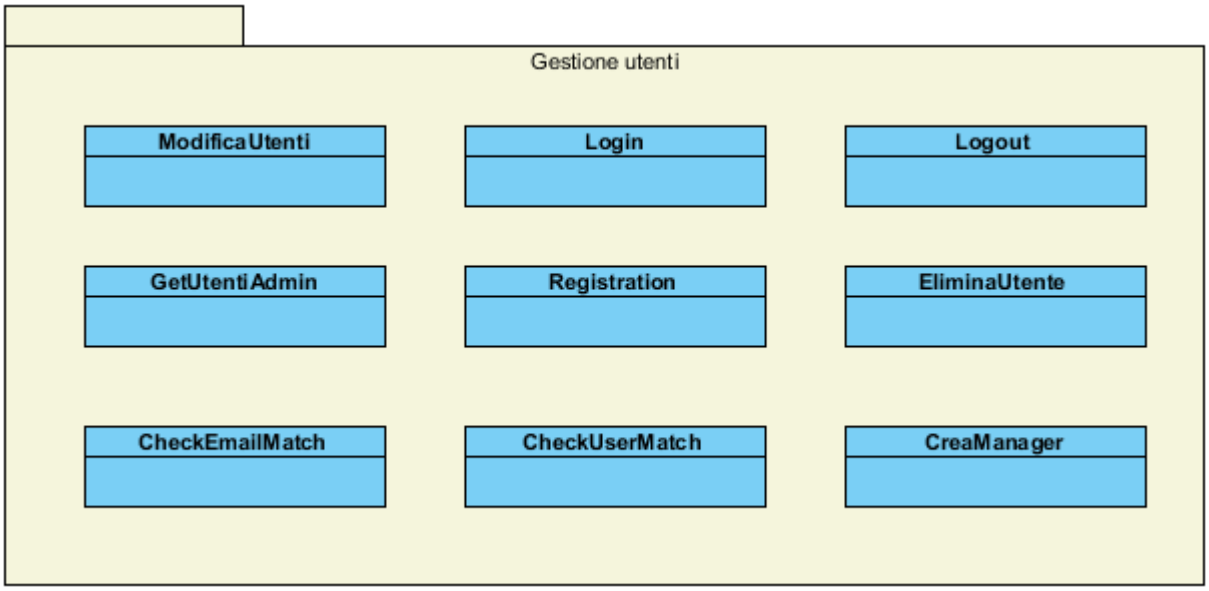
2.1. Gestione brani



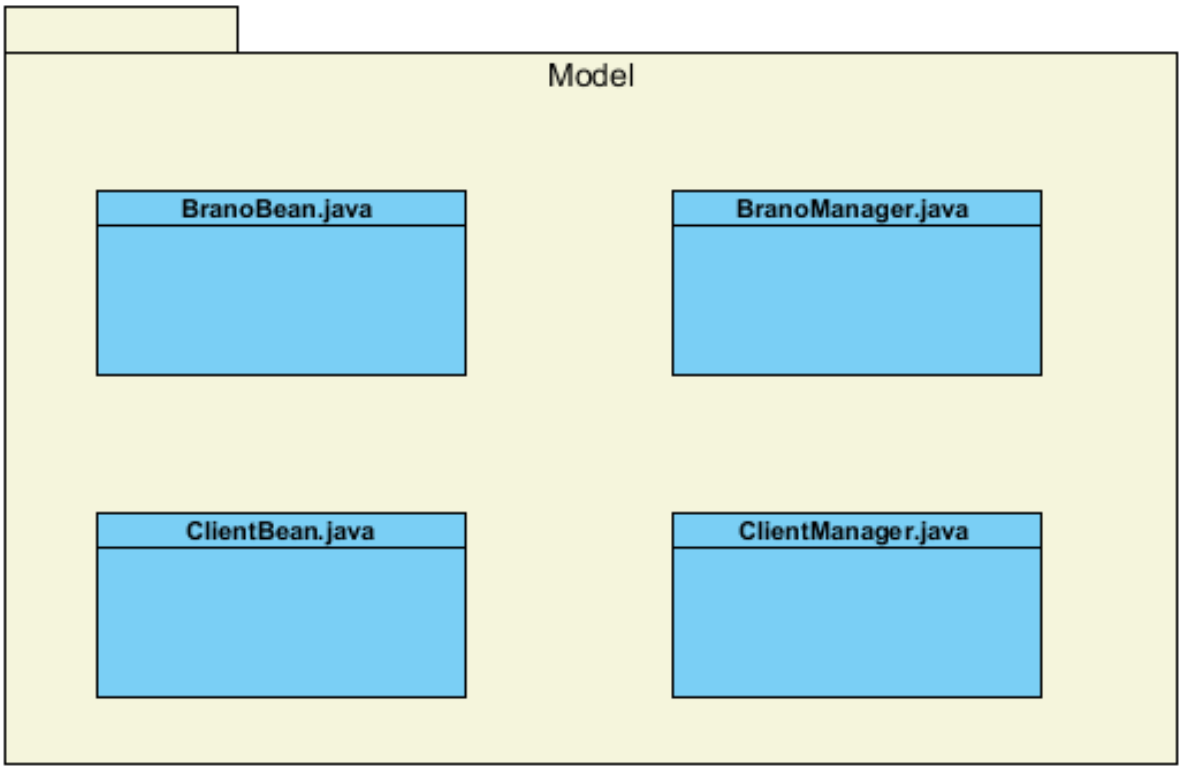
2.2. Gestione brani in stallo



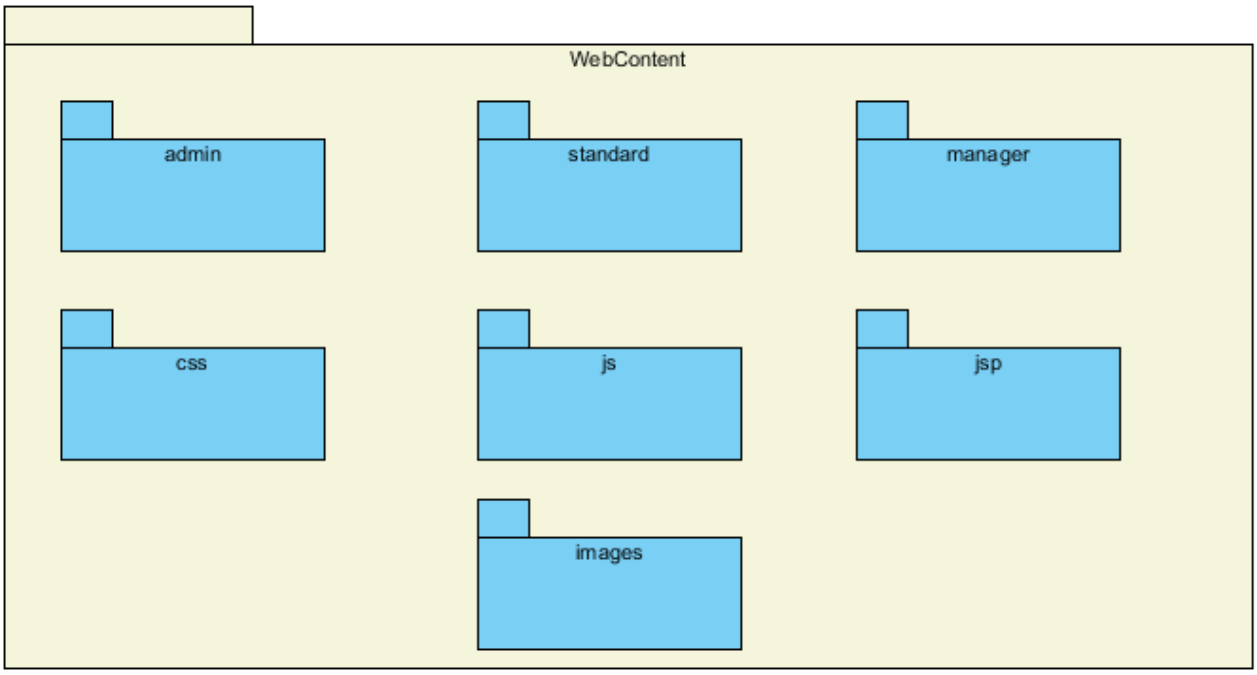
2.3. Gestione utenti



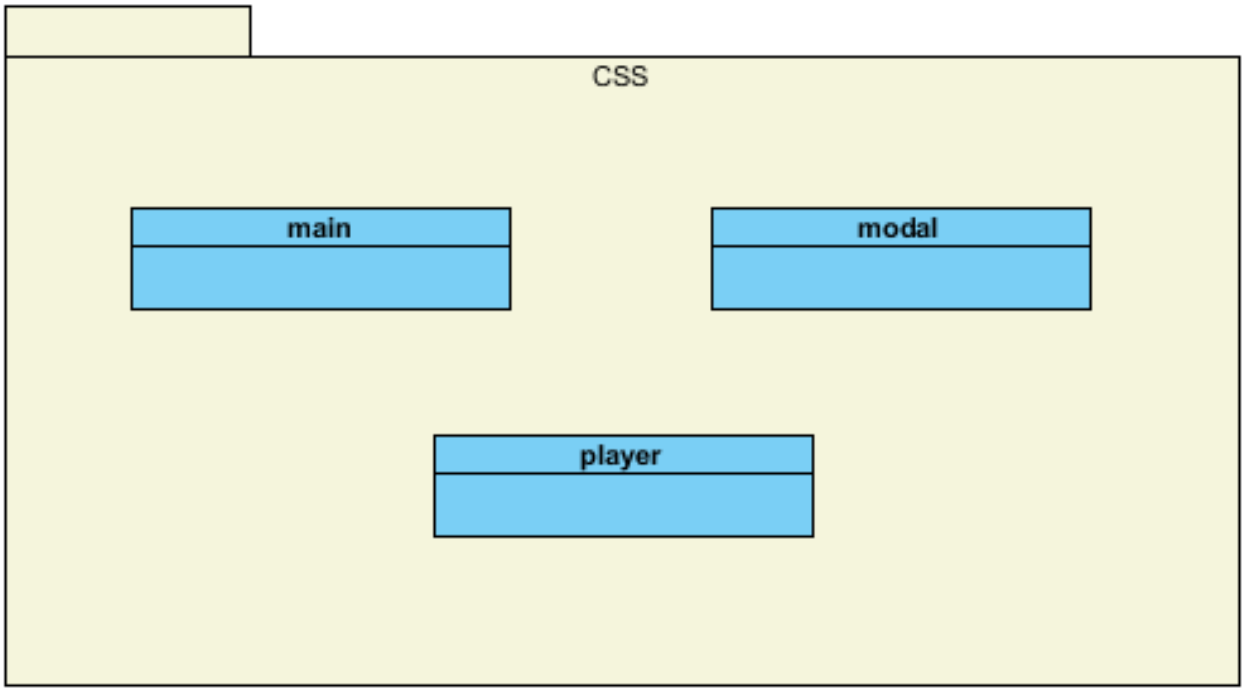
2.4. Model



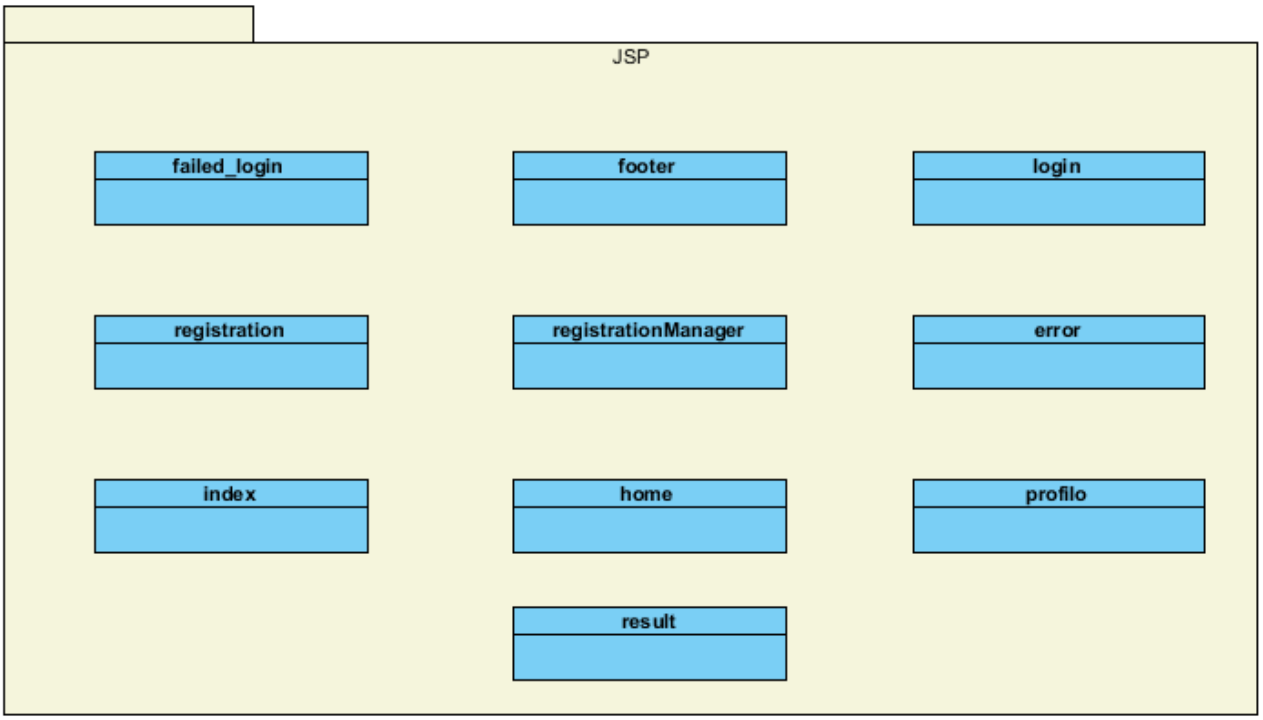
2.5. WebContent



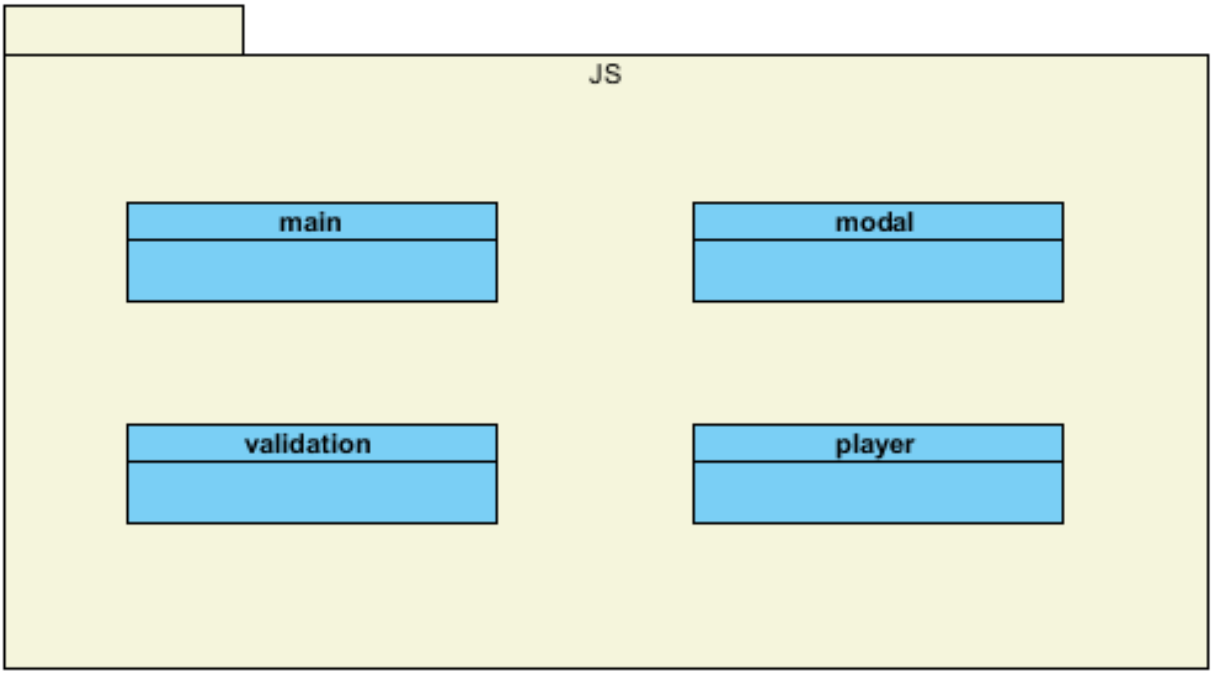
2.5.1. CSS



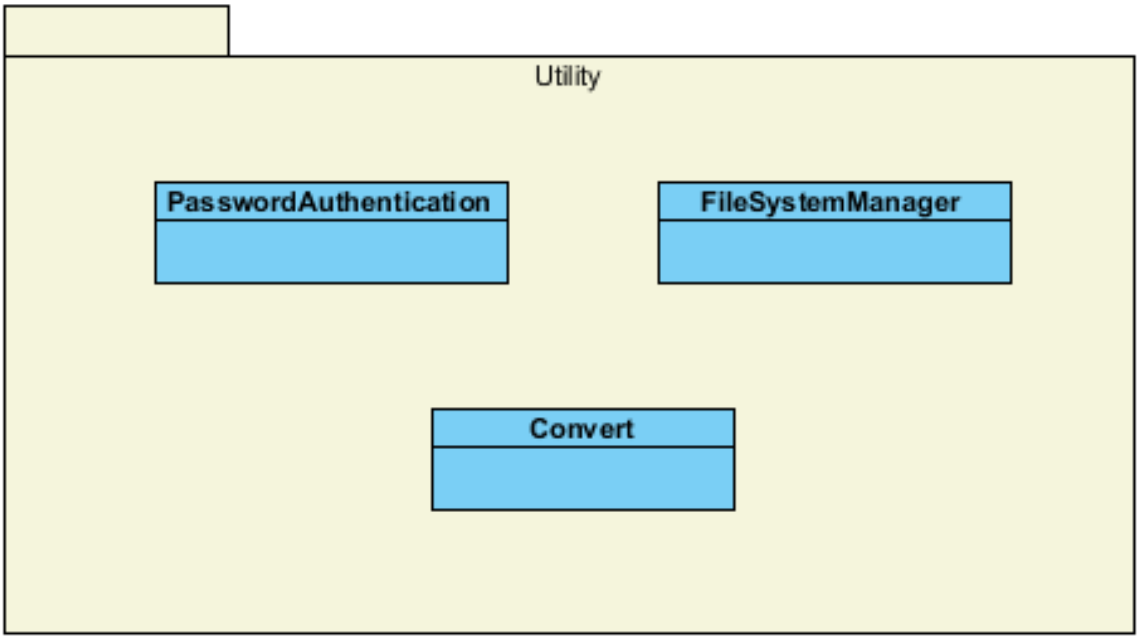
2.5.2. JSP



2.5.3. JS



2.6. Utility



3.Interfacce delle classi

3.1. Gestione brani

3.1.1. Cerca brano

Nome della classe	CercaBrano
Descrizione	Questa classe è una servlet che si occupa di gestire la ricerca di un brano.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa della ricerca di un brano. Ha come parametro la stringa "param", utilizza la funzione **cercaBrano(param)** per confrontare la stringa con i titoli, gli album e gli artisti presenti nel DB e restituisce in JSON i risultati.
Pre – condizione	**Context:** CercaBrano :: doGet(request, response) **Pre:** request.getParameter("search") != null
Post – condizione	**Context:** CercaBrano :: doGet(request,response) **Post:** //

3.1.2. Get Audio

Nome della classe	GetAudio
Descrizione	Questa classe è una servlet che si occupa di fornire al client il brano in formato mp3
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di creare un FileInputStream per leggere dal "path" fornito come parametro il file .mp3 e di trasmettere i byte sulla ServletOutputStream.

Pre – condizione	**Context:** GetAudio :: doGet(request, response) **Pre:** Il path è corretto è vi corrisponde un file mp3.
Post – condizione	**Context:** GetAudio :: doGet(request, response) **Post:** //

3.1.3. Get Brani Salvati

Nome della classe	**GetBraniSalvati**
Descrizione	Questa classe è una servlet che si occupa restituire la lista dei brani salvati dell'utente
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	**+doGet(HttpServletRequest request, HttpServletResponse response): void**
Descrizione	Questo metodo si occupa di richiamare il metodo **getBraniSalvati(user_id)** che effettua la ricerca nel DB di eventuali brani salvati per poi restituirli in formato JSON.
Pre – condizione	**Context:** GetBraniSalvati :: doGet(request, response) **Pre:** user_id!=null.
Post – condizione	**Context:** GetBraniSalvati :: doGet(request, response) **Post:** //

3.1.4. Get Brani Utente

Nome della classe	**GetBraniUtente**
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione dei brani di un utente.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di richiamare il metodo **getBraniUtente(utente)** che effettua la ricerca nel DB di eventuali brani per poi restituirli in formato JSON.
Pre – condizione	**Context:** GetBraniUtente :: doGet(request, response) **Pre:** utente!=null && utente è un intero.
Post – condizione	**Context:** GetBraniUtente :: doGet(request, response) **Post:** se utente = 0, vengono restituiti i brani dell'utente che richiama il metodo; se utente > 0, vengono restituiti i brani dell'utente con id > 0;

3.1.5. Get Home Brani

Nome della classe	GetHomeBrani
Descrizione	Questa classe è una servlet che si occupa di mostrare tutti i brani approvati di tutti gli utenti.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di richiamare il metodo **getHomeBrani()** che effettua la ricerca nel DB di eventuali brani per poi restituirli in formato JSON.
Pre – condizione	//
Post – condizione	//

3.1.6. Modifica Brano

Nome della classe	ModificaBrano
Descrizione	Questa classe è una servlet che si occupa di gestire la modifica di un prodotto.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la chiamata del metodo **modificaBrano()**
Pre – condizione	**Context:** ModificaBrano :: doGet(request, response) **Pre:** request.getParameter(“song_id”) != null, request.getParameter(“numero”) != null, request.getParameter(“titolo”) != null, request.getParameter(“album”) != null, request.getParameter(“artista”) != null, request.getParameter(“path”) != null, request.getParameter(“user_id”) != null.
Post – condizione	**Context:** ModificaBrano :: doGet(request, response) **Post:** brano modificato.

3.1.7. Salva Brano

Nome della classe	ModificaBrano
Descrizione	Questa classe è una servlet che si occupa di gestire la modifica di un prodotto.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la chiamata del metodo **modificaBrano()**
Pre – condizione	**Context:** ModificaBrano :: doGet(request, response) **Pre:** request.getParameter(“song_id”) != null, request.getParameter(“numero”) != null, request.getParameter(“titolo”) != null, request.getParameter(“album”) != null, request.getParameter(“artista”) != null, request.getParameter(“path”) != null, request.getParameter(“user_id”) != null.
Post – condizione	**Context:** ModificaBrano :: doGet(request, response) **Post:** brano modificato.

3.2. Gestione brani in stallo

3.2.1. Approva brano

Nome della classe	ApprovaBrano
Descrizione	Questa classe è una servlet che si occupa dell'approvazione di un brano da parte di un manager.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la chiamata del metodo **approvaBrano(id).**
Pre – condizione	**Context:** ApprovaBrano :: doGet(request, response) **Pre:** client.getRuolo()).equals("manager") && request.getParameter("brano_id")!=0
Post – condizione	**Context:** ApprovaBrano:: doGet(request, response) **Post:** brano approvato.

3.2.2. Visualizza brani in stallo

Nome della classe	GetBraniStallo
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione dei brani in stallo.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la chiamata del metodo **getBraniStallo()** che restituisce al manager tutti i brani non ancora approvati**.**
Pre – condizione	//
Post – condizione	//

3.2.3. Elimina brani in stallo

Nome della classe	EliminaBranoInStallo
Descrizione	Questa classe è una servlet che si occupa di gestire l'eliminazione di un brano in stallo.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la chiamata del metodo **eliminaBranoInStallo()** che elimina il brano dal database.
Pre – condizione	**Context:** EliminaBranoInStallo :: doGet(request, response) **Pre:** client.getRuolo()).equals("manager") && id!=0
Post – condizione	//

3.2.4. Carica un brano

Nome della classe	UploadBrano
Descrizione	Questa classe è una servlet che si occupa di gestire l'upload di un brano mp3.
Metodi	+doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doPost(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la chiamata al metodo **makeDir()** per la creazione del path dove salvare il file, crea il file e lo salva.
Pre – condizione	**Context:** VisualizzaOrdini :: doPost(request, response) **Pre:** (ServletFileUpload.isMultipartContent(request)) && il file è .mp3 && la grandezza del file è < 10MB && fi.getFieldName().equals("numero")!=null && fi.getFieldName().equals("artista")!=null && fi.getFieldName().equals("album")!=null && fi.getFieldName().equals("titolo")!=null && fi.getFieldName().equals("usr")!=null.

Post – condizione	**Context:** VisualizzaOrdini :: doPost(request, response) **Post:** il file è salvato nel giusto path.

3.3. Gestione utenti

3.3.1. Login

Nome della classe	**Login**
Descrizione	Questa classe è una servlet che si occupa di gestire la procedura di login.
Metodi	+doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	**+doPost(HttpServletRequest request, HttpServletResponse response): void**
Descrizione	Questo metodo si occupa di effettuare la procedura di login.
Pre – condizione	**Context:** Login :: doPost(request, response) **Pre:** request.getParameter(“usr”) != null && request.getParameter(“usr”) != “ “ && request.getParameter(“pwd”) != null &&
Post – condizione	**Context:** Login :: doPost(request, response) **Post:** se il login viene effettuato da un user : (request.getSession().getAttribute(“client”)).getRuolo() == “standard” || se il login viene effettuato dal manager: (request.getSession().getAttribute(“client”)).getRuolo() == “manager” se il login viene effettuato dall’admin: (request.getSession().getAttribute(“client”)).getRuolo() == “admin”

3.3.2. Logout

Nome della classe	**Logout**
Descrizione	Questa classe è una servlet che si occupa di gestire la procedura di logout.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la procedura di logout.
Pre – condizione	**Context:** Logout :: doGet(request, response)
Post – condizione	**//**

3.3.3. Get Utenti Admin

Nome della classe	GetUtentiAdmin
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione delle informazioni essenziali riguardanti un utente iscritto da parte di un admin.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione dei dati.
Pre – condizione	**Context:** GetUtentiAdmin :: doGet(request, response) **Pre:** (cb.getRuolo().equals("admin")
Post – condizione	//

3.3.4. Modifica Utente

Nome della classe	ModificaUtente
Descrizione	Questa classe è una servlet che si occupa di gestire la modifica del ruolo di un utente.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la modifica del ruolo di un utente.

Pre – condizione	**Context:** ModificaUtente :: doGet(request, response) **Pre:** L'utente loggato è admin. Integer.*parseInt*(request.getParameter("id"))!=0 && (request.getParameter("ruolo")==”manager” || request.getParameter("ruolo")==”standard”)
Post – condizione	**Post:** (request.getParameter("ruolo")==”manager” || request.getParameter("ruolo")==”standard”)

3.3.5. Registration

Nome della classe	Registration
Descrizione	Questa classe è una servlet che si occupa di gestire la registrazione di un utente base.
Metodi	+doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doPost(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare il salvataggio dei dati del neo-iscritto.
Pre – condizione	**Context:** Registration :: doPost(request, response) **Pre:** . nome != null && nome.matches(“[0-9a-zA-Z' ']{1,30}”); Cognome != null && Cognome.matches(“[0-9a-zA-Z' ']{1,30}”); Email != null && Email.matches(“/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})$/”); Utente != null && Utente.matches(“[0-9a-zA-Z' ']{1,30}”); Ruolo == “standard”;
Post – condizione	**Post:** Il client è registrato sul sito.

3.3.6. Crea Manager

Nome della classe	Registration
Descrizione	Questa classe è una servlet che si occupa di gestire la creazione di un manager a cura dell'admin.
Metodi	+doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doPost(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare il salvataggio delle credenziali di accesso del manager.
Pre – condizione	**Context:** Registration :: doPost(request, response) **Pre:** . nome != null && nome.matches(“[0-9a-zA-Z' ']{1,30}”); Cognome != null && Cognome.matches(“[0-9a-zA-Z' ']{1,30}”); Email != null && Email.matches(“/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})$/”); Utente != null && Utente.matches(“[0-9a-zA-Z' ']{1,30}”); Ruolo == “standard”;
Post – condizione	**Post:** Il manager è registrato sul sito.

3.3.7. CheckUserMatch

Nome della classe	CheckUserMatch
Descrizione	Questa classe è una servlet che si occupa di controllare se la stringa user è gia presente come nome utente di un altro client.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la chiamata del metodo **isUtente()** e di rispondere in JSON.
Pre – condizione	**//**
Post – condizione	**//**

3.3.8. CheckEmailMatch

Nome della classe	CheckEmailMatch
Descrizione	Questa classe è una servlet che si occupa di controllare se la stringa email è gia presente come email di un altro client.

Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	**+doGet(HttpServletRequest request, HttpServletResponse response): void**
Descrizione	Questo metodo si occupa di effettuare la chiamata del metodo **isUtenteEmail()** e di rispondere in JSON.
Pre – condizione	**//**
Post – condizione	**//**

3.4. Brano Manager

Nome della classe	**BranoManager**
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti tutti i brani.
Metodi	+getHomeBrani() : String +cercaBrano(String param) : String +getBraniUtente(int s_user_id) : String +getBranoById(int id)) : BranoBean +getBraniSalvati(int s_user_id) : String +getBraniStallo() : String +uploadBranoInStallo(BranoBean bb) : boolean +salvaBrano(int brano_id, int user_id) : boolean +eliminaBranoSalvato(int brano_id, int user_id) : boolean +eliminaBranoInStallo(int brano_stallo_id) : boolean +checkSalvato(int brano_id, int user_id) : boolean +approvaBrano(int brano_stallo_id) : boolean +modificaBranoStallo(BranoBean bb) : boolean +modificaBrano(BranoBean bb) : boolean

Nome Metodo	+getHomeBrani() : String
Descrizione	Questo metodo si occupa di restituire tutti i brani approvati in formato JSON.
Pre – condizione	//
Post – condizione	//

Nome Metodo	+cercaBrano(String param) : String
Descrizione	Questo metodo si occupa di effettuare la ricerca di tutti i brani che contengono param nel nome, titolo o artista, presenti nel database, e li restituisce in formato JSON.
Pre – condizione	**Context:** BranoManager :: cercaBrano(param) **Pre:** (!param.equals(null))
Post – condizione	//

Nome Metodo	+getBraniUtente (int s_user_id) : String
Descrizione	Questo metodo si occupa di restituire tutti i brani approvati di un utente in formato JSON.
Pre – condizione	**Context:** BranoManager :: GetBraniUtente(s_user_id) **Pre:** s_user_id!=0
Post – condizione	//

Nome Metodo	+getBranoById (int id) : String
Descrizione	Questo metodo si occupa di restituire il brano con “brano_id”==id dal database in formato JSON.
Pre – condizione	**Context:** BranoManager :: GetBranoById(id) **Pre:** id!=0
Post – condizione	//

Nome Metodo	+getBraniSalvati (int s_user_id) : String
Descrizione	Questo metodo si occupa di restituire tutti i brani salvati di un utente in formato JSON.
Pre – condizione	**Context:** BranoManager :: GetBraniSalvati(s_user_id) **Pre:** s_user_id!=0

Post – condizione	//

Nome Metodo	**+getBraniStallo () : String**
Descrizione	Questo metodo si occupa di restituire tutti i brani in stallo di tutti gli utenti in formato JSON.
Pre – condizione	//
Post – condizione	//

Nome Metodo	**+uploadBranoInStallo (BranoBean bb) : boolean**
Descrizione	Questo metodo si occupa di eseguire l'upload nel database di un nuovo brano.
Pre – condizione	**Context:** BranoManager :: UploadBranoInStallo(bb) **Pre:** bb!=null
Post – condizione	//

Nome Metodo	**+salvaBrano (int brano_id, int user_id) : boolean**
Descrizione	Questo metodo si occupa di restituire tutti i brani salvati di un utente in formato JSON.
Pre – condizione	**Context:** BranoManager :: SalvaBrano(brano _id, s_user_id) **Pre:** s_user_id!=0 && brano_id!=0
Post – condizione	//

Nome Metodo	**+eliminaBranoSalvato (int brano_id, int user_id) : boolean**
Descrizione	Questo metodo si occupa di eliminare il brano salvato dell'utente specificato.
Pre – condizione	**Context:** BranoManager :: EliminaBranoSalvato(brano_id, s_user_id) **Pre:** s_user_id!=0 && brano_id!=0
Post – condizione	//

Nome Metodo	**+checkSalvato (int brano_id, int user_id) : boolean**
Descrizione	Questo metodo si occupa di controllare se il brano è stato salvato dall'utente specificato.
Pre – condizione	**Context:** BranoManager :: CheckSalvato(brano_id, s_user_id) **Pre:** s_user_id!=0 && brano_id!=0

Post – condizione	//

Nome Metodo	**+approvaBrano (int brano_stallo_id, int user_id) : boolean**
Descrizione	Questo metodo si occupa di controllare se il brano è stato salvato dall'utente specificato.
Pre – condizione	**Context:** BranoManager :: approvaBrano(brano_stallo_id, s_user_id) **Pre:** s_user_id!=0 && brano_id!=0
Post – condizione	//

Nome Metodo	**+eliminaBranoInStallo (int brano_stallo_id) : boolean**
Descrizione	Questo metodo si occupa di eliminare un brano in stallo.
Pre – condizione	**Context:** BranoManager :: eliminaBranoInStallo(brano_stallo_id) **Pre:** brano_stallo_id!=0
Post – condizione	//

Nome Metodo	**+modificaBranoInStallo (BranoBean bb) : boolean**
Descrizione	Questo metodo si occupa di modificare un brano in stallo.
Pre – condizione	**Context:** BranoManager :: modificaBranoInStallo(brano_stallo_id) **Pre:** bb!=null;
Post – condizione	//

Nome Metodo	**+modificaBrano (BranoBean bb) : boolean**
Descrizione	Questo metodo si occupa di modificare un brano.
Pre – condizione	**Context:** BranoManager :: modificaBrano (brano _id) **Pre:** bb!=null;
Post – condizione	//

3.5. Client Manager

Nome della classe	ClientManager
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti gli utenti.
Metodi	+getUtenti (): String +checkOnDb (String utente, String login_password): ClientBean bran +getClientById (int id): ClientBean bean +getUtenti(): String +registraUtente (ClientBean cb, String token): boolean +setClientTo (int id, int caso): int +isUtente(String utente): boolean +IsUtenteEmail(String email): boolean

Nome Metodo	+getClientById (int id): ClientBean bean
Descrizione	Questo metodo si occupa di verificare se nel database è presente un Acquirente tramite una username specifica presa in input.
Pre – condizione	**Context:** ClientManager :: getClientById (id) **Pre:** id != 0
Post – condizione	//

Nome Metodo	+checkOnDb (String username, String login_password): ClientBean bean
Descrizione	Questo metodo si occupa di verificare se i dati immessi dall'utente per effettuare il login sono presenti nel database.
Pre – condizione	**Context:** ClientManager :: checkOnDb(username, login_password) **Pre:** username != null && login_password != null
Post – condizione	//

Nome Metodo	+getUtenti (String username): String
Descrizione	Questo metodo si occupa di verificare se il carrello associato ad un acquirente è presente nel database.

Pre – condizione	**Context:** ClientManager :: doRetrieveCartByUser (username) **Pre:** username != null
Post – condizione	//

Nome Metodo	**+registraUtente (ClientBean cb, String token): boolean**
Descrizione	Questo metodo si occupa di salvare i dati di un nuovo utente sul databane
Pre – condizione	**Context:** ClientManager :: registraUtente (cb, token) **Pre:** toUpdate != null && !token.equals(null)
Post – condizione	//

Nome Metodo	**+setClientTo (int id, int caso): int**
Descrizione	Questo metodo si occupa di aggiornare la lista degli articoli di un carrello associato ad un Acquirente presente nel database.
Pre – condizione	**Context:** ClientManager :: setClientTo (id, caso) **Pre:** id != 0
Post – condizione	//

Nome Metodo	**+isUtente (String utente): boolean**
Descrizione	Questo metodo si occupa di cercare eventuale omonimia con un un utente registrato.
Pre – condizione	**//**
Post – condizione	//

Nome Metodo	**+isUtenteEmail (String email): boolean**
Descrizione	Questo metodo si occupa di cercare eventuale omonimia con un un email di un utente registrato.
Pre – condizione	**//**
Post – condizione	//