



**Università degli Studi di Salerno**  
Corso di Ingegneria del Software

# **Audire**

## **Object Design Document**

**Versione 1.7**



22 Dicembre 2025

Gabriele Malanga - 0512119344  
Gennaro Carmine Tozza - 0512120382

Tabella 1: Revision History

<b>Data</b>	<b>Versione</b>	<b>Descrizione</b>	<b>Autore</b>
01/12/2025	1.0	Creazione documento	Gennaro Carmine Tozza
16/12/2025	1.1	Stesura sezione introduttiva	Gabriele Malanga, Gennaro Carmine Tozza
17/12/2025	1.2	Inizio stesura sezione Packages	Gabriele Malanga, Gennaro Carmine Tozza
20/12/2025	1.3	Fine stesura sezione Packages e inizio sezione Interfacce di classe	Gennaro Carmine Tozza
21/12/2025	1.4	Inseriti design pattern utilizzati e glossario	Gennaro Carmine Tozza
03/01/2026	1.5	Aggiunte specifiche di due metodi a TeamDAO	Gennaro Carmine Tozza
11/01/2026	1.6	Aggiunte specifiche di nuovi metodi per CastingDAO e ProductionDAO	Gennaro Carmine Tozza
14/01/2026	1.7	Aggiunte specifiche di nuovi metodi per PerformerDAO e ApplicationDAO	Gennaro Carmine Tozza

## Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
1.1	Object design trade-offs . . . . .	4
1.2	Linee guida per la documentazione dell'interfaccia . . . . .	4
1.3	Definizioni, acronimi e abbreviazioni . . . . .	4
1.4	Riferimenti . . . . .	5
<b>2</b>	<b>Packages</b>	<b>5</b>
<b>3</b>	<b>Interfacce di classe</b>	<b>11</b>
3.1	Object Constraint Language . . . . .	15
<b>4</b>	<b>Design Patterns</b>	<b>28</b>
4.1	MVC . . . . .	28
4.2	DAO . . . . .	28
4.3	Singleton . . . . .	28
<b>5</b>	<b>Glossario</b>	<b>29</b>

# 1 Introduzione

Audire è una piattaforma web progettata per centralizzare e digitalizzare l'intero processo di casting nel settore dell'intrattenimento, facilitando l'incontro tra talenti (Performer) e opportunità professionali gestite da Production Manager e Casting Director.

## 1.1 Object design trade-offs

### **Performance vs. Sicurezza**

Si accettano tempi di risposta leggermente superiori per garantire, tramite controlli rigorosi, la protezione delle informazioni degli utenti.

### **Comprensibilità vs Tempo**

Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Per rispettare queste linee guida il codice sarà accompagnato da commenti volti a semplificarne la comprensione. Ovviamente questo comporterà un aumento del tempo di sviluppo del nostro progetto.

### **Information Hiding vs Efficienza**

L'incapsulamento dei dati rende il codice più ordinato e sicuro. Si accetta un impercettibile rallentamento delle operazioni in cambio di una gestione del software più facile e priva di errori.

## 1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori devono rispettare durante la progettazione e implementazione delle interfacce.

La stesura del codice e la formattazione dei commenti seguono le convenzioni standard Java definite da Sun Microsystems. Si fa riferimento specifico alle linee guida ufficiali per la struttura delle classi e l'uso di Javadoc, consultabili al seguente indirizzo: [https://checkstyle.sourceforge.io/sun\\_style.html](https://checkstyle.sourceforge.io/sun_style.html)

## 1.3 Definizioni, acronimi e abbreviazioni

- **ODD:** - Object Design Document
- **RAD:** Requirements Analysis Document - documento di analisi dei requisiti
- **SDD:** System Design Document - documento di design del sistema
- **Package:** - raggruppamento di classi, interfacce o file correlati
- **OCL:** - Object Constraint Language

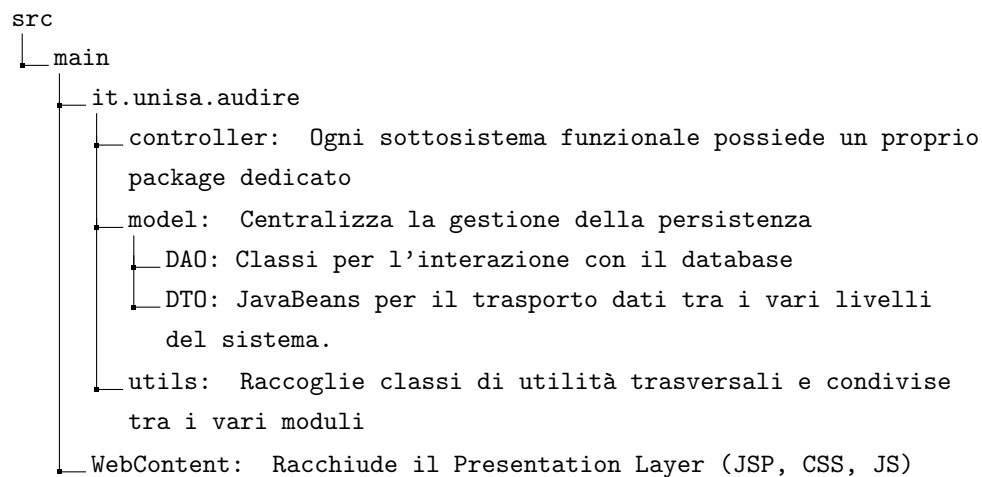
- **DAO** - Data Access Object
- **DTO** - Data Transfer Object

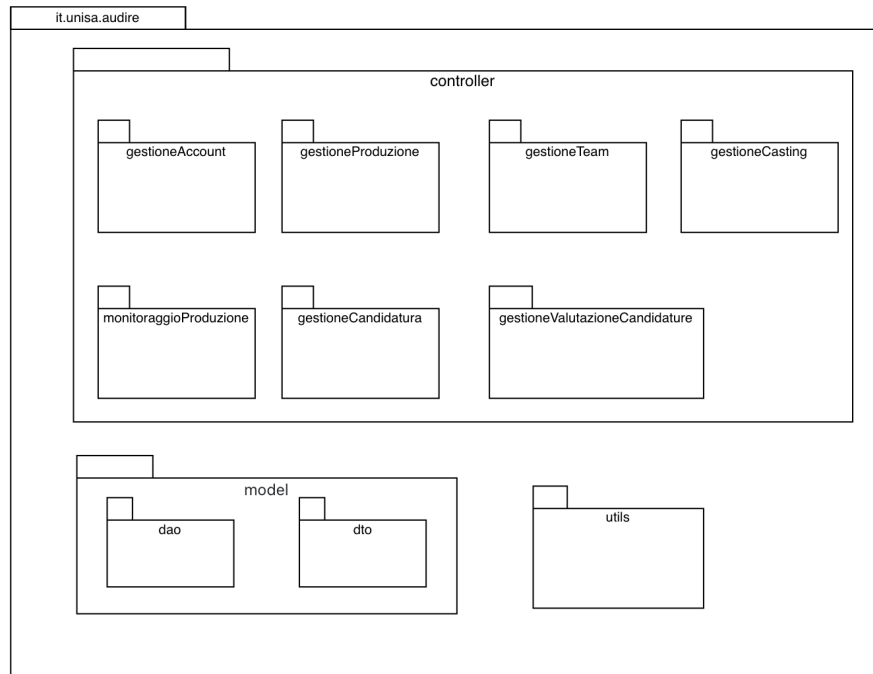
## 1.4 Riferimenti

1. System Design Document (SDD) Audire, Versione 1.5, 25 Novembre 2025, Gabriele Malanga e Gennaro Carmine Tozza
2. Requirements Analysis Document (RAD) Audire, Versione 2.3, 11 Novembre 2025, Gabriele Malanga e Gennaro Carmine Tozza
3. Object Design Document Template, TUM Applied Software Engineering
4. Object-Oriented Software Engineering Using UML, Patterns, and Java, Third Edition, Bernd Bruegge & Allen H. Dutoit
5. Materiale didattico del Corso di Ingegneria del Software, Università degli Studi di Salerno, A.A. 2025/2026
6. MVC Design Pattern, DAO Design Pattern - Geeksforgeeks.org

## 2 Packages

Si è scelto di incapsulare ogni sottosistema funzionale in un proprio package, separando nettamente la logica di business dalla persistenza e dalla presentazione. La struttura è organizzata come segue:

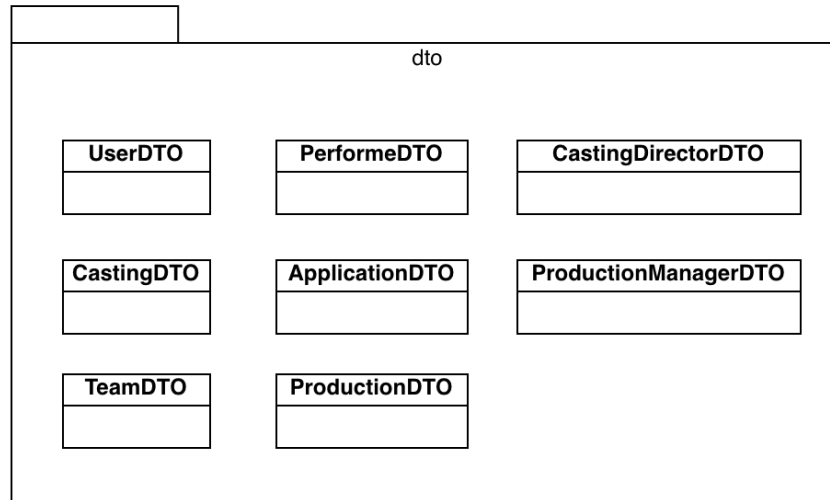




## Package model

### Package model.dto

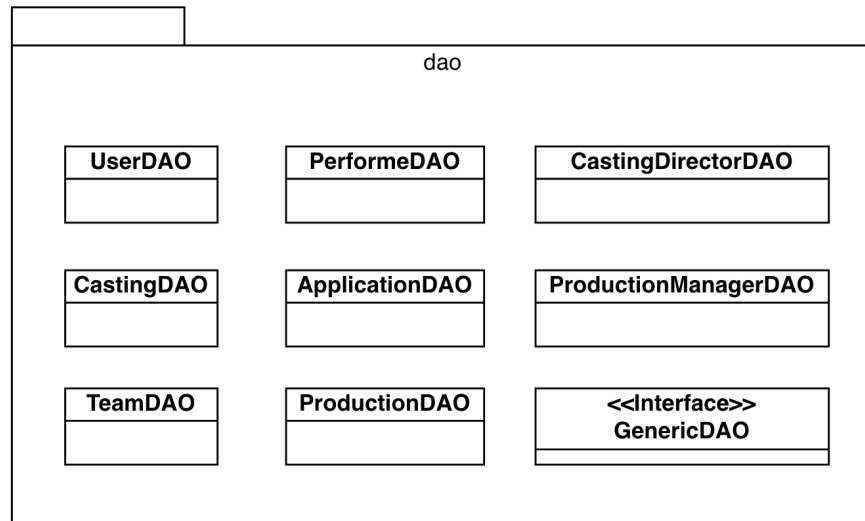
Ogni classe DTO corrisponde a una tabella del database garantendo che i dati siano serializzabili e leggeri. Tutte le classi in questo package definiscono attributi privati, metodi di accesso pubblici (getter e setter) e un costruttore di default.



Nome Classe	Descrizione
UserDTO	Rappresenta l'utente generico del sistema.
PerformerDTO	Contiene i dati del profilo specifici per un Performer.
CastingDirectorDTO	Rappresenta un Casting Director.
ProductionManagerDTO	Descrive un Production Manager
ProductionDTO	Rappresenta un'entità di produzione (es. un film o uno spettacolo teatrale). Include dettagli come titolo, descrizione, genere e il Production Manager responsabile.
CastingDTO	Rappresenta uno specifico annuncio di casting creato all'interno di una produzione.
ApplicationDTO	Rappresenta una candidatura inviata da un Performer per un determinato Casting.
TeamDTO	Rappresenta i casting director che sono associati a una produzione.

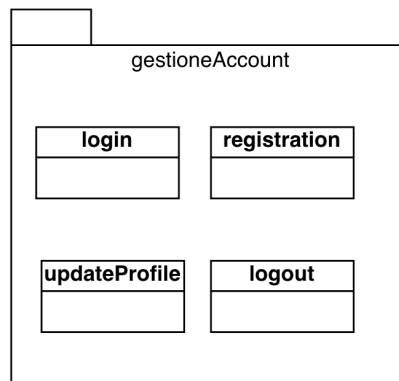
### Package model.dao

Ogni classe DAO gestisce le operazioni CRUD (Create, Read, Update, Delete) per una specifica entità, occupandosi di stabilire la connessione tramite un DataSource, mappare i ResultSet SQL nei corrispondenti oggetti DTO.



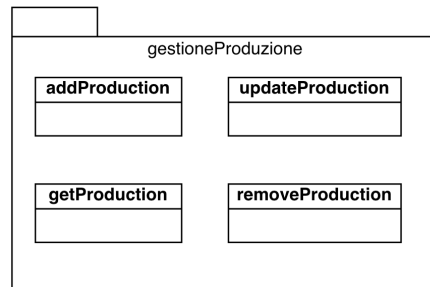
## Package controller

Package gestioneAccount

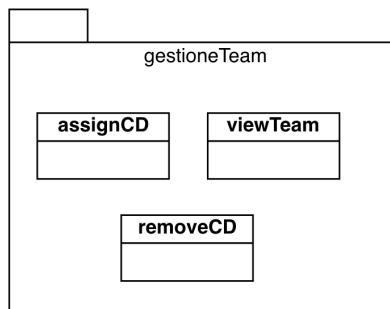




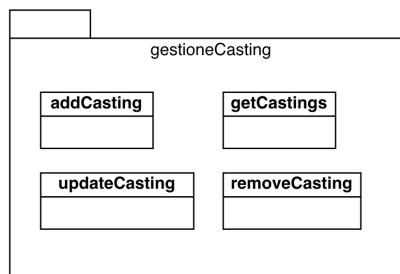
## Package gestioneProduzione



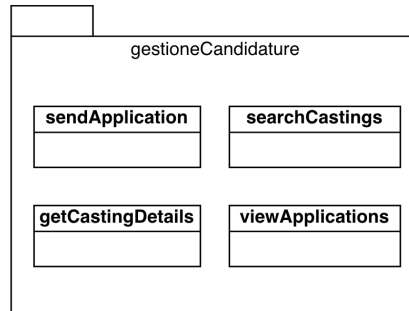
## Package gestioneTeam



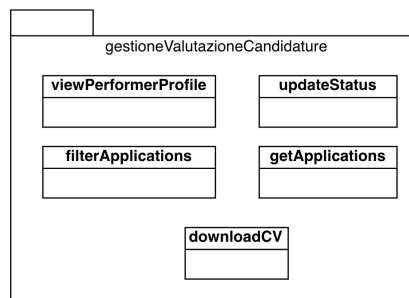
## Package gestioneCasting



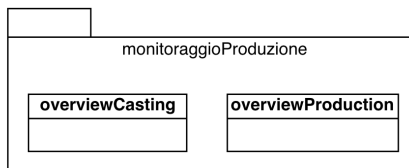
## Package gestioneCandidature



## Package gestioneValutazioneCandidature



## Package monitoraggioProduzione



### 3 Interfacce di classe

UserDAO	
<b>Descrizione</b>	Questa classe gestisce tutte le operazioni del database riguardanti gli utenti, comprese le operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD), nonché l'hashing delle password.
<b>Metodi</b>	+ save(UserDTO user) : void
	+ delete(Integer userID) : boolean
	+ getByID(Integer userID) : UserDTO
	+ getByEmail(String email) : UserDTO
	+ getAll(String order) : Collection<UserDTO>
	+ hashPassword(String plainPassword) : String
	+ verifyPassword(String plainPassword, String storedHash) : boolean
<b>Invariante di classe</b>	/

PerformerDAO	
<b>Descrizione</b>	Questa classe gestisce tutte le operazioni del database riguardanti i performer, comprese le operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD).
<b>Metodi</b>	+ save(PerformerDTO performer) : void
	+ delete(Integer performerID) : boolean
	+ getByID(Integer performerID) : PerformerDTO
	+ getByUserID(Integer userID) : PerformerDTO
	+ getAll(String order) : Collection<PerformerDTO>
	+ getCvData(Integer performerID) : PerformerDTO
<b>Invariante di classe</b>	/

<b>CastingDirectorDAO</b>	
<b>Descrizione</b>	Questa classe gestisce tutte le operazioni del database riguardanti i casting director, comprese le operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD).
<b>Metodi</b>	+ save(CastingDirectorDTO cd) : void
	+ delete(Integer cdID) : boolean
	+ getByID(Integer cdID) : CastingDirectorDTO
	+ getByUserID(Integer userID) : CastingDirectorDTO
	+ getAll(String order) : Collection( <i>CastingDirectorDTO</i> )
<b>Invariante di classe</b>	/

<b>ProductionManagerDAO</b>	
<b>Descrizione</b>	Questa classe gestisce tutte le operazioni del database riguardanti i production manager, comprese le operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD).
<b>Metodi</b>	+ save(ProductionManagerDTO pm) : void
	+ delete(Integer pmID) : boolean
	+ getByID(Integer pmID) : ProductionManagerDTO
	+ getByUserID(Integer userID) : ProductionManagerDTO
	+ getAll(String order) : Collection( <i>ProductionManagerDTO</i> )
<b>Invariante di classe</b>	/

<b>TeamDAO</b>	
<b>Descrizione</b>	Questa classe gestisce la relazione multi-a-molti tra Produzioni e Direttori del Casting. Permette di aggiungere membri a un team di produzione, rimuoverli e recuperare le composizioni del team.
<b>Metodi</b>	+ save(TeamDTO team) : void
	+ delete(int productionID, int cdID) : boolean
	+ getByProductionID(int productionID) : List<TeamDTO>
	+ getByCdID(int cdID) : List<TeamDTO>
	+ exists(int productionID, int cdID) : boolean
	+ getTeamMembers(int productionID) : List<UserDTO>
	+ getAvailableCastingDirectors(int productionID) : List<UserDTO>
<b>Invariante di classe</b>	/

<b>ProductionDAO</b>	
<b>Descrizione</b>	Questa classe gestisce la persistenza delle Produzioni. Gestisce il ciclo di vita di una produzione, di proprietà di un Production Manager.
<b>Metodi</b>	+ save(ProductionDTO production) : void
	+ delete(Integer productionID) : boolean
	+ getByID(Integer productionID) : ProductionDTO
	+ getByPmID(Integer pmID) : Collection<ProductionDTO>
	+ getAll(String order) : Collection<ProductionDTO>
	+ getProductionsByCdID(Integer cdID) : List<ProductionDTO>
	+ getTitleByID(Integer productionID) : String
<b>Invariante di classe</b>	/

<b>ApplicationDAO</b>	
<b>Descrizione</b>	Questa classe gestisce il ciclo di vita di candidatura. Collega un Performer a un Casting e tiene traccia dello stato dell'applicazione.
<b>Metodi</b>	+ save(ApplicationDTO app) : void
	+ delete(Integer applicationID) : boolean
	+ getByID(Integer applicationID) : ProductionDTO
	+ getByPerformerID(Integer performerID) : Collection<ApplicationDTO>
	+ getByCastingID(Integer castingID) : Collection<ApplicationDTO>
	+ getAll(String order) : Collection<ApplicationDTO>
	+ hasApplied(Integer performerID, Integer castingID) : boolean
<b>Invariante di classe</b>	/

<b>CastingDAO</b>	
<b>Descrizione</b>	Questa classe gestisce il ciclo di vita degli annunci di casting. Collega una specifica Produzione a un Casting Director e definisce i requisiti, le scadenze e i dettagli per i candidati.
<b>Metodi</b>	+ save(CastingDTO casting) : void
	+ delete(Integer castingID) : boolean
	+ getByID(Integer castingID) : CastingDTO
	+ getByProductionID(Integer productionID) : Collection<CastingDTO>
	+ getByCdID(Integer cdID) : Collection<CastingDTO>
	+ getAll(String order) : Collection<CastingDTO>
	+ getAllActive() : List<CastingDTO>
<b>Invariante di classe</b>	/

### 3.1 Object Constraint Language

#### UserDAO

<b>save(UserDTO user) : void</b>	
<b>Descrizione</b>	Rende persistente un oggetto Utente nel database. Se l'ID dell'utente è 0, viene eseguito un inserimento (INSERT) e generata una nuova chiave primaria. Se l'ID è positivo, vengono aggiornati i dati esistenti (UPDATE).
<b>Pre-condizione</b>	/
<b>Context</b>	UserDAO::save(user)
<b>Post-condizione</b>	$(user.userID == 0 \implies utenti \rightarrow includes(user) \text{ con nuovo ID}) \vee (user.userID > 0 \implies utenti \rightarrow select(u u.id == user.userID).data == user.data)$

<b>delete(Integer userID) : boolean</b>	
<b>Descrizione</b>	Rimuove un'utente dal database identificato dall'ID fornito.
<b>Pre-condizione</b>	$userID > 0$
<b>Context</b>	UserDAO::delete(userID)
<b>Post-condizione</b>	$!utenti \rightarrow exists(u u.userID == userID)$
<b>Result</b>	<b>true</b> se l'eliminazione ha successo, <b>false</b> altrimenti.

<b>getByID(Integer userID) : UserDTO</b>	
<b>Descrizione</b>	Recupera l'istanza di un utente tramite la sua chiave primaria.
<b>Pre-condizione</b>	$userID > 0$
<b>Context</b>	UserDAO::getByID(userID)
<b>Post-condizione</b>	$result == utenti \rightarrow select(u u.userID == userID) \rightarrow first()$

<b>getByEmail(String email) : UserDTO</b>	
<b>Descrizione</b>	Recupera un utente utilizzando il suo indirizzo email. Utilizzato principalmente durante la fase di autenticazione.
<b>Pre-condizione</b>	$email \neq null \wedge email \neq ""$
<b>Context</b>	UserDAO::getByEmail(email)
<b>Post-condizione</b>	$result == utenti \rightarrow select(u u.email == email) \rightarrow first()$

<b>getAll(String order) : Collection &lt;UserDTO&gt;</b>	
<b>Descrizione</b>	Restituisce una collezione contenente tutti gli utenti registrati nel sistema, ordinati secondo il parametro specificato.
<b>Pre-condizione</b>	$order \in \text{ALLOWED\_ORDER\_COLUMNS} \vee order == \text{null}$
<b>Context</b>	UserDAO::getAll(order)
<b>Post-condizione</b>	$result.size() == utenti \rightarrow size()$

<b>hashPassword(String plainPassword) : String</b>	
<b>Descrizione</b>	Genera un hash crittografico sicuro della password in chiaro utilizzando l'algoritmo Argon2.
<b>Pre-condizione</b>	$plainPassword \neq \text{null}$
<b>Context</b>	UserDAO::hashPassword(plainPassword)
<b>Post-condizione</b>	$result \neq plainPassword \wedge$ $result \text{ è una stringa hash valida}$

<b>verifyPassword(String plainPassword, String storedHash) : boolean</b>	
<b>Descrizione</b>	Verifica se la password in chiaro fornita corrisponde all'hash memorizzato nel database.
<b>Pre-condizione</b>	$plain \neq \text{null} \wedge hash \neq \text{null}$
<b>Context</b>	UserDAO::verifyPassword(plainPassword, storedHash)
<b>Result</b>	<b>true</b> se la verifica ha successo, <b>false</b> altrimenti.

## PerformerDAO

<b>save(PerformerDTO performer) : void</b>	
<b>Descrizione</b>	Salva o aggiorna un profilo Performer. Se l'ID è 0 esegue un inserimento, altrimenti aggiorna i dati esistenti.
<b>Pre-condizione</b>	$performer \neq \text{null} \wedge performer.userID > 0$
<b>Context</b>	PerformerDAO::save(performer)
<b>Post-condizione</b>	$(performer.performerID@pre == 0 \Rightarrow$ $performers \rightarrow includes(performer) \wedge$ $performer.performerID > 0) \vee$ $(performer.performerID@pre > 0 \Rightarrow$ $performers \rightarrow select(p p.performerID ==$ $performer.performerID).data == performer.data)$



<b>delete(Integer performerID) : boolean</b>	
<b>Descrizione</b>	Rimuove il profilo artistico specificato dal database. L'account utente associato non viene eliminato.
<b>Pre-condizione</b>	$performerID > 0$
<b>Context</b>	PerformerDAO::delete(performerID)
<b>Post-condizione</b>	$\neg performers \rightarrow exists(p p.performerID == performerID)$
<b>Result</b>	<b>true</b> se l'eliminazione ha successo (il record esisteva), <b>false</b> altrimenti.

<b>getByID(Integer performerID) : PerformerDTO</b>	
<b>Descrizione</b>	Recupera un profilo Performer tramite la sua chiave primaria.
<b>Pre-condizione</b>	$performerID > 0$
<b>Context</b>	PerformerDAO::getByID(performerID)
<b>Post-condizione</b>	$result == performers \rightarrow select(p p.performerID == performerID) \rightarrow first()$

<b>getByUserID(Integer userID) : PerformerDTO</b>	
<b>Descrizione</b>	Recupera il profilo Performer associato a uno specifico account utente.
<b>Pre-condizione</b>	$userID > 0$
<b>Context</b>	PerformerDAO::getByUserID(userID)
<b>Post-condizione</b>	$result == performers \rightarrow select(p p.userID == userID) \rightarrow first()$

<b>getAll(String order) : Collection &lt;PerformerDTO&gt;</b>	
<b>Descrizione</b>	Restituisce una collezione di tutti i performer, ordinati secondo il criterio specificato.
<b>Pre-condizione</b>	$order \in \text{ALLOWED\_ORDER\_COLUMNS} \vee order == \text{null}$
<b>Context</b>	PerformerDAO::getAll(order)
<b>Post-condizione</b>	$result.size() == performers \rightarrow size() \wedge result \rightarrow forAll(p performers \rightarrow includes(p))$

<b>getCvData(Integer performerID) : PerformerDTO</b>	
<b>Descrizione</b>	Recupera esclusivamente i dati binari del Curriculum Vitae e il relativo tipo MIME per un performer specifico. È un metodo ottimizzato per il download che evita il caricamento dell'intero profilo utente.
<b>Pre-condizione</b>	$performerID > 0$
<b>Context</b>	PerformerDAO::getCvData(performerID)
<b>Post-condizione</b>	$  \begin{aligned}  & (performers \rightarrow exists(p p.performerID == performerID) \implies result.cvData == p.cvData \wedge result.cvMimeType == p.cvMimeType) \vee \\  & (\neg performers \rightarrow exists(p p.performerID == performerID) \implies result == null)  \end{aligned}  $

## CastingDirectorDAO

<b>save(CastingDirectorDTO cd) : void</b>	
<b>Descrizione</b>	Crea il ruolo di Casting Director per un utente.
<b>Pre-condizione</b>	$cd \neq null \wedge cd.userID > 0$
<b>Context</b>	CastingDirectorDAO::save(cd)
<b>Post-condizione</b>	$  (cd.cdID@pre == 0 \implies castingDirectors \rightarrow includes(cd) \wedge cd.cdID > 0)  $

<b>delete(Integer cdID) : boolean</b>	
<b>Descrizione</b>	Rimuove il profilo Casting Director specificato. L'operazione revoca il ruolo, ma mantiene attivo l'account utente generico associato.
<b>Pre-condizione</b>	$cdID > 0$
<b>Context</b>	CastingDirectorDAO::delete(cdID)
<b>Post-condizione</b>	$\neg castingDirectors \rightarrow exists(c c.cdID == cdID)$
<b>Result</b>	true se l'eliminazione ha successo, false altrimenti.

<b>getByID(Integer cdID) : CastingDirectorDTO</b>	
<b>Descrizione</b>	Recupera un profilo Casting Director tramite la sua chiave primaria.
<b>Pre-condizione</b>	$cdID > 0$
<b>Context</b>	CastingDirectorDAO::getByID(cdID)
<b>Post-condizione</b>	$  result == castingDirectors \rightarrow select(c c.cdID == cdID) \rightarrow first()  $

<b>getByUserID(Integer userID) : CastingDirectorDTO</b>	
<b>Descrizione</b>	Recupera i dati del profilo Casting Director associati all'utente specificato.
<b>Pre-condizione</b>	$userID > 0$
<b>Context</b>	CastingDirectorDAO::getByUserID(userID)
<b>Post-condizione</b>	$result == castingDirectors \rightarrow select(c c.userID == userID) \rightarrow first()$

<b>getAll(String order) : Collection &lt;CastingDirectorDTO&gt;</b>	
<b>Descrizione</b>	Restituisce una collezione di tutti i profili Casting Director presenti nel sistema, ordinati secondo il criterio specificato.
<b>Pre-condizione</b>	$order \in \text{ALLOWED\_ORDER\_COLUMNS} \vee order == \text{null}$
<b>Context</b>	CastingDirectorDAO::getAll(order)
<b>Post-condizione</b>	$result.size() == castingDirectors \rightarrow size() \wedge result \rightarrow forAll(c castingDirectors \rightarrow includes(c))$

## ProductionManagerDAO

<b>save(ProductionManagerDTO pm) : void</b>	
<b>Descrizione</b>	Crea il ruolo di Production Manager per un utente.
<b>Pre-condizione</b>	$pm \neq \text{null} \wedge pm.userID > 0$
<b>Context</b>	ProductionManagerDAO::save(pm)
<b>Post-condizione</b>	$(pm.pmID@pre == 0 \implies pms \rightarrow includes(pm) \wedge pm.pmID > 0)$

<b>delete(Integer pmID) : boolean</b>	
<b>Descrizione</b>	Rimuove il profilo Production Manager specificato. Rimuove solo il ruolo, l'account utente generico rimane attivo.
<b>Pre-condizione</b>	$pmID > 0$
<b>Context</b>	ProductionManagerDAO::delete(pmID)
<b>Post-condizione</b>	$\neg productionManagers \rightarrow exists(p p.pmID == pmID)$
<b>Result</b>	<b>true</b> se l'eliminazione ha successo, <b>false</b> altrimenti.

<b>getByID(Integer pmID) : ProductionManagerDTO</b>	
<b>Descrizione</b>	Recupera un profilo Production Manager tramite la sua chiave primaria.
<b>Pre-condizione</b>	$pmID > 0$
<b>Context</b>	ProductionManagerDAO::getByID(pmID)
<b>Post-condizione</b>	$result == productionManagers \rightarrow select(p p.pmID == pmID) \rightarrow first()$

<b>getByUserID(Integer userID) : ProductionManagerDTO</b>	
<b>Descrizione</b>	Recupera il profilo Production Manager associato all'utente specificato.
<b>Pre-condizione</b>	$userID > 0$
<b>Context</b>	ProductionManagerDAO::getByUserID(userID)
<b>Post-condizione</b>	$result == pms \rightarrow select(x x.userID == userID) \rightarrow first()$

<b>getAll(String order) : Collection &lt;ProductionManagerDTO&gt;</b>	
<b>Descrizione</b>	Restituisce una collezione di tutti i Production Manager presenti nel sistema, ordinati secondo il criterio specificato.
<b>Pre-condizione</b>	$order \in \text{ALLOWED\_ORDER\_COLUMNS} \vee order == \text{null}$
<b>Context</b>	ProductionManagerDAO::getAll(order)
<b>Post-condizione</b>	$result.size() == productionManagers \rightarrow size() \wedge result \rightarrow forAll(p productionManagers \rightarrow includes(p))$

## TeamDAO

<b>save(TeamDTO team) : void</b>	
<b>Descrizione</b>	Aggiunge un Casting Director al team di una produzione. Se l'associazione esiste già, non fa nulla.
<b>Pre-condizione</b>	$team \neq \text{null} \wedge team.productionID > 0 \wedge team.cdID > 0$
<b>Context</b>	TeamDAO::save(team)
<b>Post-condizione</b>	$teams \rightarrow exists(t t.productionID == team.productionID \wedge t.cdID == team.cdID)$

<b>delete(int productionID, int cdID) : boolean</b>	
<b>Descrizione</b>	Rimuove un Casting Director dal team di una specifica produzione.
<b>Pre-condizione</b>	$productionID > 0 \wedge cdID > 0$
<b>Context</b>	TeamDAO::delete(productionID, cdID)
<b>Post-condizione</b>	$\neg teams \rightarrow exists(t   t.productionID == productionID \wedge t.cdID == cdID)$

<b>getByProductionID(int productionID) : List&lt;TeamDTO&gt;</b>	
<b>Descrizione</b>	Restituisce tutti i membri del team assegnati a una produzione specifica.
<b>Pre-condizione</b>	$productionID > 0$
<b>Context</b>	TeamDAO::getByProductionID(productionID)
<b>Post-condizione</b>	$result \rightarrow forAll(t   t.productionID == productionID) \wedge result.size() == teams \rightarrow select(t   t.productionID == productionID) \rightarrow size()$

<b>getByCdID(int cdID) : List&lt;TeamDTO&gt;</b>	
<b>Descrizione</b>	Restituisce tutte le associazioni (e quindi le produzioni) in cui è coinvolto un determinato Casting Director.
<b>Pre-condizione</b>	$cdID > 0$
<b>Context</b>	TeamDAO::getByCdID(cdID)
<b>Post-condizione</b>	$result \rightarrow forAll(t   t.cdID == cdID) \wedge result.size() == teams \rightarrow select(t   t.cdID == cdID) \rightarrow size()$

<b>exists(int productionID, int cdID) : boolean</b>	
<b>Descrizione</b>	Verifica se esiste già un'associazione tra una specifica produzione e un Casting Director. Utile per evitare duplicati prima di un inserimento.
<b>Pre-condizione</b>	$productionID > 0 \wedge cdID > 0$
<b>Context</b>	TeamDAO::exists(productionID, cdID)
<b>Post-condizione</b>	$result == teams \rightarrow exists(t   t.productionID == productionID \wedge t.cdID == cdID)$

<b>getTeamMembers(int productionID) : List&lt;UserDTO&gt;</b>	
<b>Descrizione</b>	Restituisce la lista dei Casting Director (dettagli utente) che sono attualmente membri del team per una specifica produzione.
<b>Pre-condizione</b>	$productionID > 0$
<b>Context</b>	TeamDAO::getTeamMembers(productionID)
<b>Post-condizione</b>	$result \rightarrow \text{forAll}(u teams \rightarrow \text{exists}(t t.productionID == productionID \wedge t.cdID == u.cdID))$

<b>getAvailableCastingDirectors(int productionID) : List&lt;UserDTO&gt;</b>	
<b>Descrizione</b>	Restituisce la lista dei Casting Director disponibili che non sono attualmente assegnati alla produzione specificata. Utile per popolare liste di selezione.
<b>Pre-condizione</b>	$productionID > 0$
<b>Context</b>	TeamDAO::getAvailableCastingDirectors(productionID)
<b>Post-condizione</b>	$result \rightarrow \text{forAll}(u !teams \rightarrow \text{exists}(t t.productionID == productionID \wedge t.cdID == u.cdID))$

## ProductionDAO

<b>save(ProductionDTO production) : void</b>	
<b>Descrizione</b>	Crea o aggiorna una produzione. Se l'ID è 0, viene creata una nuova istanza associata al Production Manager specificato. Se l'ID è > 0 allora vengono aggiornati i dati esistenti.
<b>Pre-condizione</b>	$production \neq \text{null} \wedge production.pmID > 0$
<b>Context</b>	ProductionDAO::save(production)
<b>Post-condizione</b>	$(production.id@pre == 0 \implies productions \rightarrow \text{includes}(production) \wedge production.id > 0) \vee (production.id@pre > 0 \implies productions \rightarrow \text{select}(p p.id == production.id).data == production.data)$

<b>delete(Integer productionID) : boolean</b>	
<b>Descrizione</b>	Elimina una produzione e, a cascata, tutti i casting e le assegnazioni del team associati.
<b>Pre-condizione</b>	$productionID > 0$
<b>Context</b>	ProductionDAO::delete(productionID)
<b>Post-condizione</b>	$!productions \rightarrow \text{exists}(p p.productionID == productionID)$

<b>getByID(Integer productionID) : ProductionDTO</b>	
<b>Descrizione</b>	Recupera i dettagli di una specifica produzione.
<b>Pre-condizione</b>	$productionID > 0$
<b>Context</b>	ProductionDAO::getByID(productionID)
<b>Post-condizione</b>	$result == productions \rightarrow select(p p.productionID == productionID) \rightarrow first()$

<b>getByPmID(Integer pmID) : Collection&lt;ProductionDTO&gt;</b>	
<b>Descrizione</b>	Recupera tutte le produzioni gestite da un determinato Production Manager.
<b>Pre-condizione</b>	$pmID > 0$
<b>Context</b>	ProductionDAO::getByPmID(pmID)
<b>Post-condizione</b>	$result \rightarrow forAll(p p.pmID == pmID) \wedge result.size() == productions \rightarrow select(p p.pmID == pmID) \rightarrow size()$

<b>getAll(String order) : Collection&lt;ProductionDTO&gt;</b>	
<b>Descrizione</b>	Restituisce la lista completa di tutte le produzioni nel sistema.
<b>Pre-condizione</b>	$order \in ALLOWED\_ORDER\_COLUMNS \vee order == null$
<b>Context</b>	ProductionDAO::getAll(order)
<b>Post-condizione</b>	$result.size() == productions \rightarrow size()$

<b>getProductionsByCdID(Integer cdID) : List&lt;ProductionDTO&gt;</b>	
<b>Descrizione</b>	Recupera l'elenco delle produzioni assegnate a uno specifico Casting Director tramite la tabella di associazione Team.
<b>Pre-condizione</b>	$cdID > 0$
<b>Context</b>	ProductionDAO::getProductionsByCdID(cdID)
<b>Post-condizione</b>	$result \rightarrow forAll(p Team \rightarrow exists(t t.productionID == p.productionID \wedge t.cdID == cdID))$

<b>getTitleByID(Integer productionID) : String</b>	
<b>Descrizione</b>	Recupera esclusivamente il titolo di una produzione dato il suo ID. Metodo leggero utilizzato per popolare liste o tabelle.
<b>Pre-condizione</b>	$productionID > 0$
<b>Context</b>	ProductionDAO::getTitleByID(productionID)
<b>Post-condizione</b>	$(productions \rightarrow exists(p p.productionID == productionID) \implies result == p.title) \vee$ $(\neg productions \rightarrow exists(p p.productionID == productionID) \implies result == "Unknown")$

## ApplicationDAO

<b>save(ApplicationDTO app) : void</b>	
<b>Descrizione</b>	Inserisce una nuova candidatura o aggiorna lo stato/feedback di una esistente.
<b>Pre-condizione</b>	$app \neq null \wedge app.performerID > 0 \wedge app.castingID > 0$
<b>Context</b>	ApplicationDAO::save(app)
<b>Post-condizione</b>	$(app.id@pre == 0 \implies applications \rightarrow$ $includes(app) \wedge app.id > 0) \vee (app.id@pre > 0 \implies$ $applications \rightarrow select(a a.id == app.id).status == app.status)$

<b>delete(Integer applicationID) : boolean</b>	
<b>Descrizione</b>	Elimina (ritira) una candidatura.
<b>Pre-condizione</b>	$applicationID > 0$
<b>Context</b>	ApplicationDAO::delete(applicationID)
<b>Post-condizione</b>	$\neg applications \rightarrow exists(a a.applicationID == applicationID)$

<b>getByID(Integer applicationID) : ApplicationDTO</b>	
<b>Descrizione</b>	Recupera i dettagli di una specifica candidatura tramite il suo ID univoco.
<b>Pre-condizione</b>	$applicationID > 0$
<b>Context</b>	ApplicationDAO::getByID(applicationID)
<b>Post-condizione</b>	$result == applications \rightarrow$ $select(a a.applicationID == applicationID) \rightarrow first()$



<b>getByPerformerID(Integer performerID):Collection&lt;ApplicationDTO&gt;</b>	
<b>Descrizione</b>	Recupera tutte le candidature inviate da un determinato Performer.
<b>Pre-condizione</b>	$performerID > 0$
<b>Context</b>	ApplicationDAO::getByPerformerID(performerID)
<b>Post-condizione</b>	$result \rightarrow \text{forAll}(a a.performerID == performerID) \wedge result.size() == applications \rightarrow select(a a.performerID == performerID) \rightarrow size()$

<b>getByCastingID(Integer castingID) : Collection&lt;ApplicationDTO&gt;</b>	
<b>Descrizione</b>	Recupera tutte le candidature ricevute per un determinato annuncio di Casting.
<b>Pre-condizione</b>	$castingID > 0$
<b>Context</b>	ApplicationDAO::getByCastingID(castingID)
<b>Post-condizione</b>	$result \rightarrow \text{forAll}(a a.castingID == castingID) \wedge result.size() == applications \rightarrow select(a a.castingID == castingID) \rightarrow size()$

<b>getAll(String order) : Collection&lt;ApplicationDTO&gt;</b>	
<b>Descrizione</b>	Restituisce l'elenco completo di tutte le candidature presenti nel sistema, ordinabile per data, stato o casting.
<b>Pre-condizione</b>	$order \in \text{ALLOWED\_ORDER\_COLUMNS} \vee order == \text{null}$
<b>Context</b>	ApplicationDAO::getAll(order)
<b>Post-condizione</b>	$result.size() == applications \rightarrow size() \wedge result \rightarrow \text{forAll}(a applications \rightarrow includes(a))$

<b>hasApplied(Integer performerID, Integer castingID) : boolean</b>	
<b>Descrizione</b>	Verifica se esiste già una candidatura registrata nel sistema per la combinazione specifica di Performer e Casting indicata. Utilizzato per prevenire candidature duplicate.
<b>Pre-condizione</b>	$performerID > 0 \wedge castingID > 0$
<b>Context</b>	ApplicationDAO::hasApplied(performerID, castingID)
<b>Post-condizione</b>	$result == applications \rightarrow exists(a a.performerID == performerID \wedge a.castingID == castingID)$

## CastingDAO

<b>save(CastingDTO casting) : void</b>	
<b>Descrizione</b>	Inserisce un nuovo annuncio di casting o ne aggiorna i dettagli.
<b>Pre-condizione</b>	$casting \neq \text{null} \wedge casting.productionID > 0 \wedge casting.cdID > 0$
<b>Context</b>	CastingDAO::save(casting)
<b>Post-condizione</b>	$(casting.id@pre == 0 \implies castings \rightarrow includes(casting) \wedge casting.id > 0) \vee (casting.id@pre > 0 \implies castings \rightarrow select(c c.id == casting.id).data == casting.data)$

<b>delete(Integer castingID) : boolean</b>	
<b>Descrizione</b>	Rimuove un annuncio di casting dal sistema.
<b>Pre-condizione</b>	$castingID > 0$
<b>Context</b>	CastingDAO::delete(castingID)
<b>Post-condizione</b>	$\neg castings \rightarrow exists(c c.castingID == castingID)$

<b>getByID(Integer castingID) : CastingDTO</b>	
<b>Descrizione</b>	Recupera i dettagli completi di un singolo annuncio di casting tramite il suo identificativo univoco.
<b>Pre-condizione</b>	$castingID > 0$
<b>Context</b>	CastingDAO::getByID(castingID)
<b>Post-condizione</b>	$result == castings \rightarrow select(c c.castingID == castingID) \rightarrow first()$

<b>getByProductionID(Integer productionID) : Collection&lt;CastingDTO&gt;</b>	
<b>Descrizione</b>	Recupera tutti i casting aperti per una specifica produzione.
<b>Pre-condizione</b>	$productionID > 0$
<b>Context</b>	CastingDAO::getByProductionID(productionID)
<b>Post-condizione</b>	$result \rightarrow forAll(c c.productionID == productionID) \wedge result.size() == castings \rightarrow select(c c.productionID == productionID) \rightarrow size()$

<b>getByCdID(Integer cdID) : Collection&lt;CastingDTO&gt;</b>	
<b>Descrizione</b>	Recupera tutti i casting gestiti da un determinato Casting Director.
<b>Pre-condizione</b>	$cdID > 0$
<b>Context</b>	CastingDAO::getByCdID(cdID)
<b>Post-condizione</b>	$result \rightarrow \text{forall}(c c.cdID == cdID) \wedge result.size() == castings \rightarrow select(c c.cdID == cdID) \rightarrow size()$

<b>getAll(String order) : Collection&lt;CastingDTO&gt;</b>	
<b>Descrizione</b>	Restituisce l'elenco completo di tutti i casting presenti nel sistema.
<b>Pre-condizione</b>	$order \in \text{ALLOWED\_ORDER\_COLUMNS} \vee order == \text{null}$
<b>Context</b>	CastingDAO::getAll(order)
<b>Post-condizione</b>	$result.size() == castings \rightarrow size()$

<b>getAllActive() : List&lt;CastingDTO&gt;</b>	
<b>Descrizione</b>	Recupera tutti i casting attivi (con scadenza odierna o futura), ordinati per data di pubblicazione decrescente.
<b>Pre-condizione</b>	/
<b>Context</b>	CastingDAO::getAllActive()
<b>Post-condizione</b>	$result \rightarrow \text{forall}(c c.deadLine \geq Date.now()) \wedge result.size() == castings \rightarrow select(c c.deadLine \geq Date.now()) \rightarrow size()$

## 4 Design Patterns

Un design pattern descrive un problema che si verifica ripetutamente nel nostro ambiente, e propone poi il nucleo della soluzione a quel problema, in modo tale da poter utilizzare questa soluzione innumerevoli volte senza mai farlo due volte allo stesso modo.

### 4.1 MVC

Il design pattern Model View Controller (MVC) specifica che un'applicazione è composta da un modello dati, informazioni di presentazione e informazioni di controllo. Il pattern richiede che ciascuno di questi sia separato in oggetti diversi. Il pattern MVC separa le esigenze di un'applicazione in tre componenti distinti, ciascuno responsabile di un aspetto specifico della funzionalità dell'applicazione. Questa separazione delle esigenze semplifica la manutenzione e l'estensione dell'applicazione, poiché le modifiche a un componente non richiedono modifiche agli altri componenti.

### 4.2 DAO

Il DAO Design Pattern è un modo di organizzare il codice per gestire la comunicazione tra il programma e un database. Il modello di progettazione DAO (Data Access Object) è come il piano di un architetto per la gestione dei dati nel software. Si tratta di un progetto che gli sviluppatori utilizzano per creare un sistema strutturato e ordinato per ottenere informazioni da una fonte di dati, come un database. Invece di occuparsi direttamente dei dettagli essenziali di come i dati vengono archiviati e recuperati, il modello DAO funge da guida, astruendo le complessità e offrendo un modo chiaro per interagire con i dati.

### 4.3 Singleton

Il pattern Singleton garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale ad essa. La creazione e la chiusura di connessioni al database sono operazioni costose in termini di risorse. Avere più istanze che gestiscono le connessioni in modo concorrente può portare a inconsistenze o esaurimento delle risorse.

In Audire, il pattern Singleton è applicato alla classe di gestione della connessione al database DataSource. Questo assicura che esista un'unica istanza responsabile della gestione del pool di connessioni, condivisa da tutti i DAO, ottimizzando le prestazioni e garantendo un accesso centralizzato alle risorse dati.

## 5 Glossario

**Application (Candidatura)** Rappresenta la manifestazione di interesse di un Performer per uno specifico Casting. Può assumere diversi stati (In attesa, Shortlist, Selezionata, Rifiutata) e contenere un feedback da parte del Casting Director.

**Casting** Annuncio di lavoro pubblicato all'interno di una Produzione, rivolto a specifiche categorie di Performer (es. attori, musicisti). Definisce i requisiti e le scadenze per le candidature.

**Casting Director (CD)** Attore del sistema responsabile della gestione operativa dei casting. I suoi compiti includono la pubblicazione degli annunci, la valutazione delle candidature e la selezione dei talenti.

**DAO (Data Access Object)** Design pattern strutturale che astrae e incapsula l'accesso alla base di dati. Fornisce un'interfaccia specifica per le operazioni CRUD senza esporre i dettagli del database sottostante.

**DTO (Data Transfer Object)** Oggetto semplice (JavaBean) utilizzato per trasportare dati tra i sottosistemi (es. dal Database alla View) riducendo il numero di chiamate remote. Non contiene logica di business, ma solo attributi, getter e setter.

**MVC (Model-View-Controller)** Pattern architetturale che separa la logica di presentazione (View) dalla logica di business (Controller) e dai dati (Model), facilitando la manutenzione e la scalabilità del software.

**OCL (Object Constraint Language)** Linguaggio formale dichiarativo utilizzato per descrivere regole che si applicano ai modelli UML. In questo documento, è utilizzato per specificare le pre-condizioni, post-condizioni e invarianti dei metodi delle classi DAO.

**Performer** Utente finale della piattaforma che cerca opportunità lavorative. Possiede un profilo artistico contenente dati personali, caratteristiche fisiche e materiale multimediale (CV, Foto).

**Production (Produzione)** Entità che raggruppa una serie di attività di casting. Rappresenta un progetto artistico (es. Film, Serie TV, Spettacolo Teatrale) ed è gestita da un Production Manager.

**Production Manager (PM)** Utente con permessi elevati responsabile della creazione delle produzioni e della gestione del Team di Casting Directors. Ha una visione globale sull'andamento dei casting.

**Servlet** Classe Java che estende le funzionalità di un server. Nel contesto di Audire, le Servlet fungono da Controller, gestendo le richieste HTTP, invocando la logica di business e selezionando la vista appropriata.