Università degli Studi di Salerno
Corso di Ingegneria del Software

Audire
Object Design Document
Versione 1.4

22 Dicembre 2025

Gabriele Malanga - 0512119344
Gennaro Carmine Tozza - 0512120382

Tabella 1: Revision History

Data	**Versione**	**Descrizione**	**Autore**
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

Indice

1 Introduzione **4**
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

2 Packages **5**

3 Interfacce di classe **11**
3.1 Object Constraint Language 14

4 Design Patterns **25**
4.1 MVC 25
4.2 DAO 25
4.3 Singleton 25

5 Glossario **26**

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$_style.html$

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

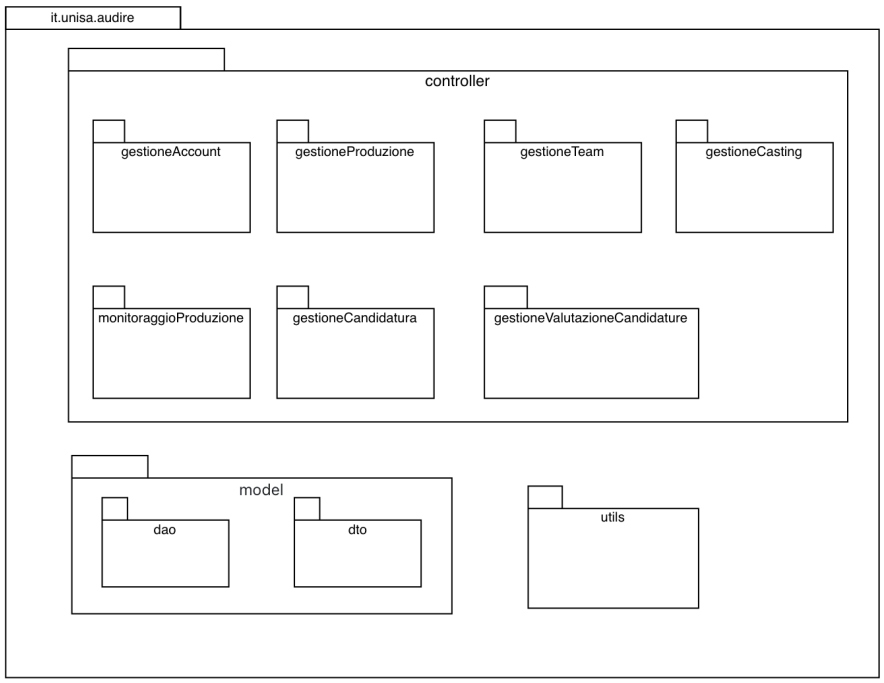
1.4 Riferimenti

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:

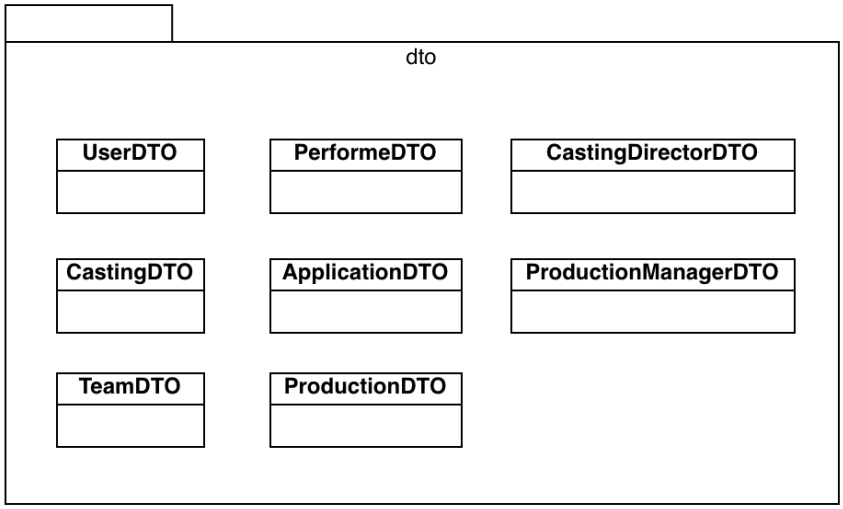
```
src
└── main
    ├── it.unisa.audire
    │   ├── controller:  Ogni sottosistema funzionale possiede un proprio
    │   │   package dedicato
    │   ├── model:  Centralizza la gestione della persistenza
    │   │   ├── DAO: Classi per l'interazione con il database
    │   │   └── DTO: JavaBeans per il trasporto dati tra i vari livelli
    │   │       del sistema.
    │   └── utils:  Raccoglie classi di utilità trasversali e condivise
    │       tra i vari moduli
    └── WebContent:  Racchiude il Presentation Layer (JSP, CSS, JS)
```



Package model

Package model.dto

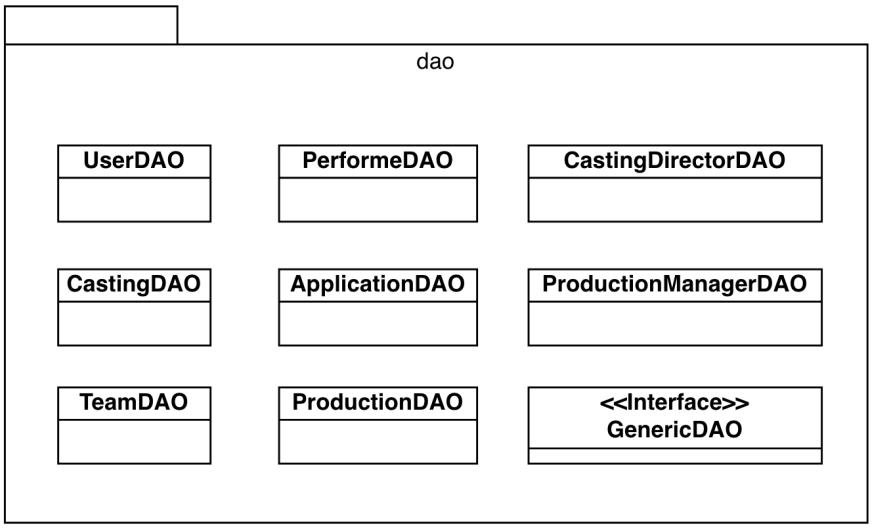
Ogni classe DTO corrisponde a una tabella del database garantendo che i dati siano serializzabili e leggeri. Tutte le classi in questo package definiscono attributi privati, metodi di accesso pubblici (getter e setter) e un costruttore di default.



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.

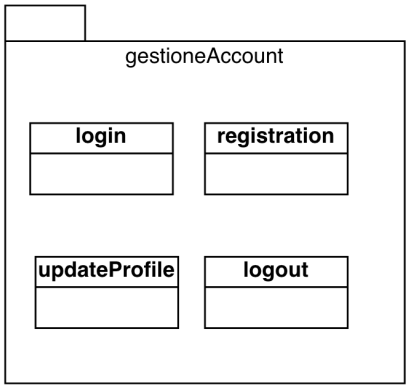
Package model.dao

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.

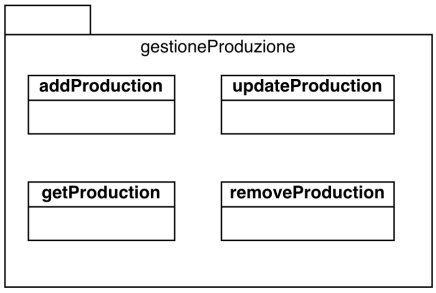


Package controller

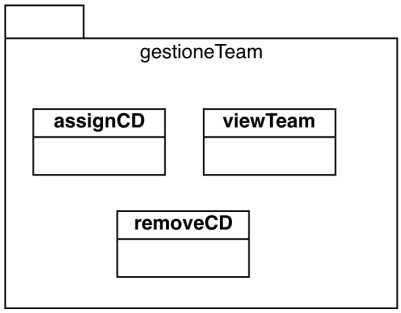
Package gestioneAccount



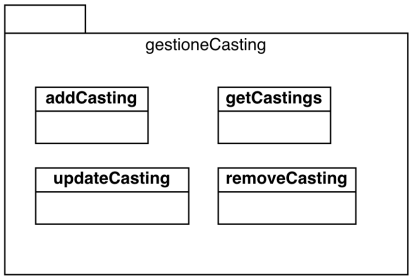
Package gestioneProduzione



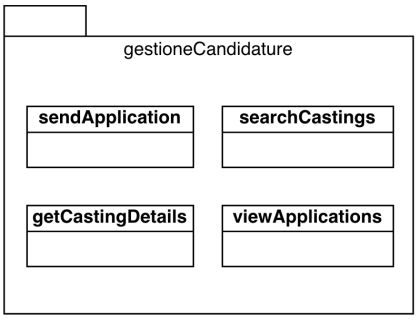
Package gestioneTeam



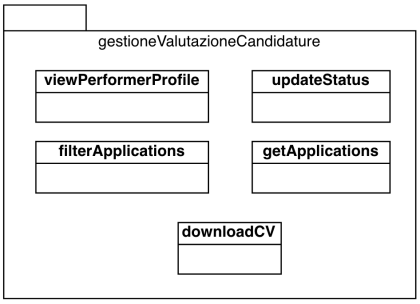
Package gestioneCasting



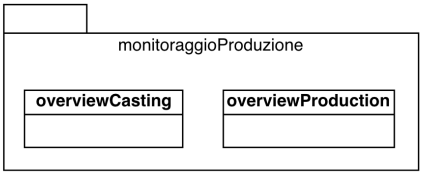
Package gestioneCandidature



Package gestioneValutazioneCandidature



Package monitoraggioProduzione



3 Interfacce di classe

UserDAO	
Descrizione	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.
Metodi	+ save(UserDTO user) : void
	+ delete(Integer userID) : boolean
	+ getByID(Integer userID) : UserDTO
	+ getByEmail(String email) : UserDTO
	+ getAll(String order) : Collection$\langle UserDTO \rangle$
	+ hashPassword(String plainPassword) : String
	+ verifyPassword(String plainPassword, String storedHash) : boolean
Invariante di classe	/

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

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

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

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

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

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

CastingDAO	
Descrizione	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.
Metodi	+ save(CastingDTO casting) : void
	+ delete(Integer castingID) : boolean
	+ getByID(Integer castingID) : CastingDTO
	+ getByProductionID(Integer productionID) : Collection$\langle CastingDTO \rangle$
	+ getByCdID(Integer cdID) : Collection$\langle CastingDTO \rangle$
	+ getAll(String order) : Collection$\langle CastingDTO \rangle$
Invariante di classe	/

3.1 Object Constraint Language

UserDAO

save(UserDTO user) : void	
Descrizione	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).
Pre-condizione	/
Context	UserDAO::save(user)
Post-condizione	$(user.userID == 0 \implies utenti \rightarrow includes(user)$ con nuovo ID)$\vee$ $(user.userID > 0 \implies utenti \rightarrow select(u\|u.id == user.userID).data == user.data)$

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

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

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

getAll(String order) : Collection $\langle UserDTO \rangle$	
Descrizione	Restituisce una collezione contenente tutti gli utenti registrati nel sistema, ordinati secondo il parametro specificato.
Pre-condizione	$order \in$ ALLOWED_ORDER_COLUMNS $\vee$ $order ==$ null
Context	UserDAO::getAll(order)
Post-condizione	$result.size() == utenti \rightarrow size()$

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

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

PeformerDAO

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

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

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

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

getAll(String order) : Collection $\langle PerformerDTO \rangle$	
Descrizione	Restituisce una collezione di tutti i performer, ordinati secondo il criterio specificato.
Pre-condizione	$order \in$ ALLOWED_ORDER_COLUMNS $\vee$ $order ==$ null
Context	PerformerDAO::getAll(order)
Post-condizione	$result.size() == performers \rightarrow size() \wedge result \rightarrow forAll(p \mid performers \rightarrow includes(p))$

CastingDirectorDAO

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

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

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

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

getAll(String order) : Collection $\langle CastingDirectorDTO \rangle$	
Descrizione	Restituisce una collezione di tutti i profili Casting Director presenti nel sistema, ordinati secondo il criterio specificato.
Pre-condizione	$order \in$ ALLOWED_ORDER_COLUMNS $\vee$ $order ==$ null
Context	CastingDirectorDAO::getAll(order)
Post-condizione	$result.size() == castingDirectors \rightarrow size() \wedge result \rightarrow forAll(c \mid castingDirectors \rightarrow includes(c))$

ProductionManagerDAO

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

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

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

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

getAll(String order) : Collection $\langle ProductionManagerDTO \rangle$	
Descrizione	Restituisce una collezione di tutti i Production Manager presenti nel sistema, ordinati secondo il criterio specificato.
Pre-condizione	$order \in$ ALLOWED_ORDER_COLUMNS $\vee\ order ==$ null
Context	ProductionManagerDAO::getAll(order)
Post-condizione	$result.size() == productionManagers \rightarrow size() \wedge result \rightarrow forAll(p \mid productionManagers \rightarrow includes(p))$

TeamDAO

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

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

getByProductionID(int productionID) : List$\langle TeamDTO \rangle$	
Descrizione	Restituisce tutti i membri del team assegnati a una produzione specifica.
Pre-condizione	$productionID > 0$
Context	TeamDAO::getByProductionID(productionID)
Post-condizione	$result \rightarrow forAll(t \mid t.productionID == productionID) \land result.size() == teams \rightarrow select(t \mid t.productionID == productionID) \rightarrow size()$

getByCdID(int cdID) : List$\langle TeamDTO \rangle$	
Descrizione	Restituisce tutte le associazioni (e quindi le produzioni) in cui è coinvolto un determinato Casting Director.
Pre-condizione	$cdID > 0$
Context	TeamDAO::getByCdID(cdID)
Post-condizione	$result \rightarrow forAll(t \mid t.cdID == cdID) \land result.size() == teams \rightarrow select(t \mid t.cdID == cdID) \rightarrow size()$

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

ProductionDAO

save(ProductionDTO production) : void	
Descrizione	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.
Pre-condizione	$production \neq \text{null} \land production.pmID > 0$
Context	ProductionDAO::save(production)
Post-condizione	$(production.id@pre == 0 \implies productions \rightarrow includes(production) \land production.id > 0) \lor (production.id@pre > 0 \implies productions \rightarrow select(p \mid p.id == production.id).data == production.data)$

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

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

getByPmID(Integer pmID) : Collection$\langle ProductionDTO \rangle$	
Descrizione	Recupera tutte le produzioni gestite da un determinato Production Manager.
Pre-condizione	$pmID > 0$
Context	ProductionDAO::getByPmID(pmID)
Post-condizione	$result \rightarrow forAll(p \mid p.pmID == pmID) \land result.size() == productions \rightarrow select(p \mid p.pmID == pmID) \rightarrow size()$

getAll(String order) : Collection$\langle ProductionDTO \rangle$	
Descrizione	Restituisce la lista completa di tutte le produzioni nel sistema.
Pre-condizione	$order \in$ ALLOWED_ORDER_COLUMNS $\lor$ $order ==$ null
Context	ProductionDAO::getAll(order)
Post-condizione	$result.size() == productions \rightarrow size()$

ApplicationDAO

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

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

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

getByPerformerID(Integer performerID):Collection$\langle ApplicationDTO \rangle$	
Descrizione	Recupera tutte le candidature inviate da un determinato Performer.
Pre-condizione	$performerID > 0$
Context	ApplicationDAO::getByPerformerID(performerID)
Post-condizione	$result \rightarrow forAll(a\|a.performerID == performerID) \wedge result.size() == applications \rightarrow select(a\|a.performerID == performerID) \rightarrow size()$

getByCastingID(Integer castingID) : Collection$\langle ApplicationDTO \rangle$	
Descrizione	Recupera tutte le candidature ricevute per un determinato annuncio di Casting.
Pre-condizione	$castingID > 0$
Context	ApplicationDAO::getByCastingID(castingID)
Post-condizione	$result \rightarrow forAll(a \mid a.castingID == castingID) \land result.size() == applications \rightarrow select(a \mid a.castingID == castingID) \rightarrow size()$

getAll(String order) : Collection $\langle ApplicationDTO \rangle$	
Descrizione	Restituisce l'elenco completo di tutte le candidature presenti nel sistema, ordinabile per data, stato o casting.
Pre-condizione	$order \in$ ALLOWED_ORDER_COLUMNS $\lor$ $order ==$ null
Context	ApplicationDAO::getAll(order)
Post-condizione	$result.size() == applications \rightarrow size() \land result \rightarrow forAll(a \mid applications \rightarrow includes(a))$

CastingDAO

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

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

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

getByProductionID(Integer productionID) : Collection$\langle CastingDTO \rangle$	
Descrizione	Recupera tutti i casting aperti per una specifica produzione.
Pre-condizione	$productionID > 0$
Context	CastingDAO::getByProductionID(productionID)
Post-condizione	$result \rightarrow forAll(c \mid c.productionID == productionID) \wedge result.size() == castings \rightarrow select(c \mid c.productionID == productionID) \rightarrow size()$

getByCdID(Integer cdID) : Collection$\langle CastingDTO \rangle$	
Descrizione	Recupera tutti i casting gestiti da un determinato Casting Director.
Pre-condizione	$cdID > 0$
Context	CastingDAO::getByCdID(cdID)
Post-condizione	$result \rightarrow forAll(c \mid c.cdID == cdID) \wedge result.size() == castings \rightarrow select(c \mid c.cdID == cdID) \rightarrow size()$

getAll(String order) : Collection$\langle CastingDTO \rangle$	
Descrizione	Restituisce l'elenco completo di tutti i casting presenti nel sistema.
Pre-condizione	$order \in$ ALLOWED_ORDER_COLUMNS $\vee$ $order ==$ null
Context	CastingDAO::getAll(order)
Post-condizione	$result.size() == castings \rightarrow size()$

4 Design Patterns

Un design un 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, astraendo 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.