Object Design Document

Agency Formation

Riferimento	AF_ODD_v1.1
Versione	1.1
Data	20/01/2022
Destinatario	Filomena Ferrucci (FF), Fabio Palomba (FP)
Presentato da	GC, LG, GL, MN, DP, ES, PS
Approvato da	AC, VP

Revision History

Data	Versione	Descrizione	Autori
09/01/2021	1.0	Prima stesura	GC, GL, DP, LG, MN, ES, PS
20/01/2022	1.1	Modifiche delle class e dei metodi dovute alle class e ai metodi usati con l'implementazione accorgimenti grafici	DP, ES

Tabella dei contenuti

1. Introduzione 5

1.1.Linee guida per la documentazione dell'interfaccia 5

1.2.Definizioni, acronimi e abbreviazioni 6

1.3..... Riferimenti 7

2. Packages 8

3. Interfacce delle classi 12

3.1 Package agency_formation 12

3.1.1 it.unisa.agencyFormation.autenticazione.AutenticazioneControl 12

3.1.2 it.unisa.agencyFormation.formazione.FormazioneControl 16

3.1.3 it.unisa.agencyFormation.reclutamento.control.ReclutamentoControl 21

3.1.4 it.unisa.agencyFormation.team.control.TeamControl 32

3.1.5 it.unisa.agencyFormation.autenticazione.manager.AutenticazioneManager 40

3.1.6 it.unisa.agencyFormation.reclutamento.manager.ReclutamentoManager 42

3.1.7 it.unisa.agencyFormation.team.manager.TeamManager 44

3.1.8 it.unisa.agencyFormation.formazione.manager.FormazioneManager 46

3.1.9 it.unisa.agencyFormation.autenticazione.dao.UtenteDAO 48

3.2.0 it.unisa.agencyFormation.autenticazione.dao.DipendenteDAO 49

3.2.1 it.unisa.agencyFormation.reclutamento.dao.CandidaturaDAO 50

3.2.2 it.unisa.agencyFormation.team.dao.TeamDAO 52

3.2.3 it.unisa.agencyFormation.formazione.dao.DocumentoDAO 54

3.2.4 it.unisa.agencyFormation.autenticazione.dao.SkillDAO 55

4. Design pattern con class diagram 57

4.1.4 Class Diagram 58
5. Glossario 61

1. Introduzione

In questa sezione vengono descritti i trade-off, le linee guida e le convenzioni da adottare.

1.0 Object design trade-off

Tempo di risposta vs Costi

Per ottimizzare i tempi di risposta del sistema Agency Formation, si può ricorrere all'utilizzo di: memorie ad alta velocità, utilizzo di librerie affidabili e sempre aggiornate, ed una banda larga efficiente che facilita sia l'upload che il download in modo tale da ottimizzare le prestazioni del sistema.

Disponibilità vs Tolleranza ai guasti

Nel caso si verifichi un errore all'interno del sistema, vengono bloccate, temporaneamente, le funzionalità che interessano il guasto; il tutto serve a garantire maggiore disponibilità.

Tempo di rilascio vs Funzionalità

Per rispettare il tempo di rilascio si potrebbero omettere o tralasciare certe funzionalità, oppure rilasciare tali funzioni in versione beta (quindi senza testarle o con qualità più bassa).

Server proprietario vs Server provider

Per avere maggiore manutenibilità sia Hardware che Software, e maggior sicurezza, si potrebbe pensare di utilizzare un server provider al posto di un server proprietario. Tuttavia, tale scelta può portare svantaggi in ambito della modularità Hardware del server, del completo controllo del server e della sua configurazione.

1.1. Linee guida per la documentazione dell'interfaccia

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

1.1.1. Nomi dei file

- Le classi devono avere nomi al singolare.
- I metodi devono essere scritti in inglese, la specifica sarà in italiano.
- I nomi dei file sorgente Java devono essere uguali al nome della classe top level.
- I nomi dei file generati da compilatori, sistemi di build, o altri tool non devono essere modificati.
- I nomi delle classi di test di unità devono avere il suffisso "Test", (es. classTest.java).
- I nomi delle classi di test di integrazione devono avere il suffisso "IT", (es. classIT.java).
- In tutti gli altri casi, i nomi dei file devono rappresentare bene il contenuto e devono contenere solo lettere minuscole, maiuscole, cifre ed underscore.

1.1.2. Struttura dei file sorgente

La struttura dei file sorgente viene dettata dal sistema di build Gradle.

- I file relativi all'implementazione del sistema seguiranno la seguente struttura:
  - *src/main/java/{system}/{subsystem}/{package}/{file}.java*
- I file relativi alle view seguiranno la seguente struttura: *src/main/webapp/html/{file}.html*
- I file di stile css seguiranno la seguente struttura: *src/main/webapp/css/{file}.css*
- I file di javascript seguiranno la seguente struttura: *src/main/webapp/js/{file}.js*
- I file relativi al testing seguiranno la seguente struttura:

src/test/java/{system}/{subsystem}/{package}/{fileTest}.java

1.1.3. Formattazione

Per la formattazione del file Java si seguiranno le convenzioni della Sun di Java, mentre per i file XML, HTML 5, CSS e JS si userà il formatter di IntelliJ di default.

1.1.4. Dichiarazioni

Ogni dichiarazione di variabile locale può definire più di una variabile, mentre ogni dichiarazione di variabile di istanza deve definire solo una variabile. Le variabili d'istanza devono essere private. Ad una dichiarazione di variabile locale deve seguire l'inizializzazione nella stessa linea oppure in quella seguente.

1.1.5. Nomenclatura

Di seguito sono mostrati i vincoli di nomi delle componenti software del sistema:

- Package: solo lettere in *lowerCamelCase*
- Classi: solo lettere in *UpperCamelCase*
- Metodi: solo lettere in *lowerCamelCase*, devono contenere nel nome solo verbi e nomi degli
- attributi della classe
- Costanti: solo lettere e underscore in CONSTANT_CASE
- Variabili: solo lettere in *lowerCamelCase*
- Parametri: solo lettere in *lowerCamelCase*. In particolare, dello stesso nome delle relative variabili di istanza nei metodi setter e nei costruttori, possono essere solo sostantivi

1.1.6. Convenzioni

- Le condizioni d'errore lanciano delle eccezioni e non valori di ritorno.
- Uso del for-each loop quando bisogna iterare per intero una collezione iterabile.
- Per gli if usare sempre le graffe, anche con singoli statement.
- Gli if sui booleani non devono avere == true o == false.

1.1.7. Documentazione del codice

I commenti di documentazione saranno nel formato di Javadoc; verrà scritto un commento doc per gli elementi del codice.

1.2. Definizioni, acronimi e abbreviazioni

Vengono riportati di seguito alcune definizioni presenti nel documento corrente:

- Package: raggruppamento di classi ed interfacce correlate.
- Subsystem: insieme di servizi legati da una relazione funzionale.
- Interfacce delle classi: insieme di signature delle operazioni offerte dalla classe.
- Design pattern: template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.
- Domain: identifica il dominio dell'autonomia amministrativa, dell'autorità o del controllo, è
  - costituito da una serie di stringhe separate da punti.
- COTS: componenti software disponibili sul mercato per gruppi di sviluppo interessati ad utilizzarli nei loro progetti.
- lowerCamelCase: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione nel
  - mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi.

- UpperCamelCase: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi
  - con una lettera maiuscola, senza spazi o punteggiatura intermedi.
- CONSTANT_CASE: è la pratica di scrivere frasi in maiuscolo, gli spazi sono rappresentati con
  - un underscore.

1.3. Riferimenti

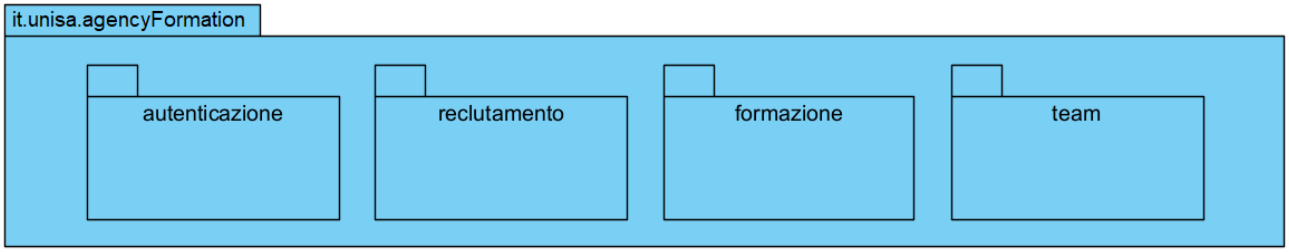
- Documento di Statement of Work relativo a questo progetto. Link alla risorsa: SOW
- Requirements Analysis Document relativo a questo progetto. Link alla risorsa: RAD
- System Design Document relativo a questo progetto. Link alla risorsa: SDD

2. Packages

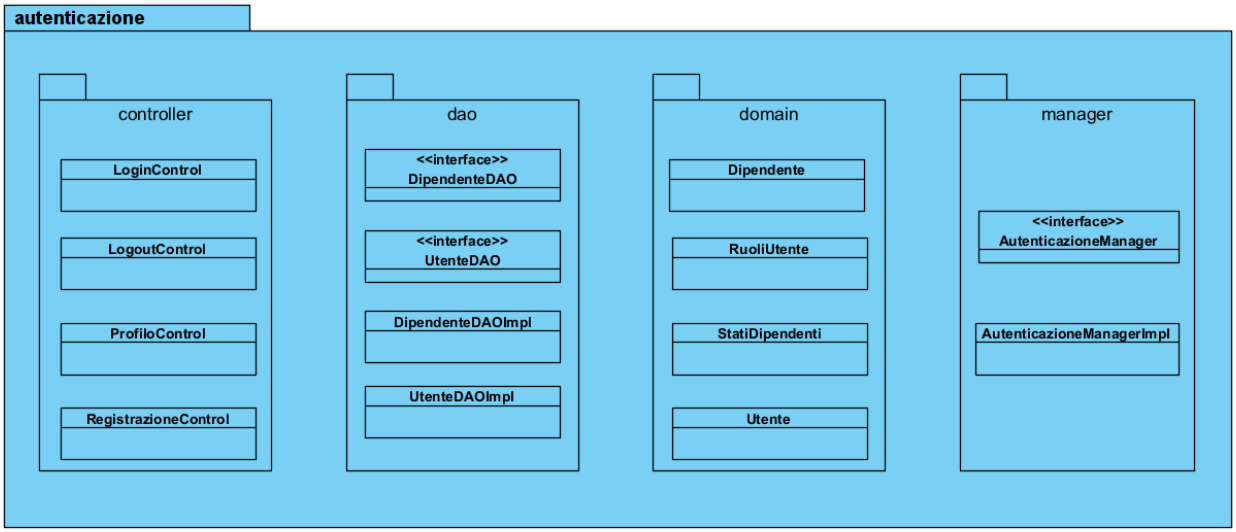
In questa sezione viene rappresentata la suddivisione in package del sistema in base all'architettura scelta. La seguente suddivisione ricalca, oltre l'architettura del sistema, anche la struttura di directory imposta da ***Gradle***.
La struttura della directory è la seguente:

- **.idea**
- **.gradle**, che contiene i file di configurazione per Gradle
- **src**, contiene tutti i file sorgente del sistema
    - **main**
        - **java**, composta dai vari sotto-package che contengono le classi Java relative ai Control e Model.
        - **resources**
        - **webapp**
            - **css,** composta dai fogli di stile
            - **js,** composta dai file javaScript
            - **static,** contenente pagine jsp che non cambiano il loro contenuto in base ai dati persistenti
            - **img**
            - **WEB-INF**
                - **jsp,** composta da tutte le pagine dinamiche relative alle View.
    - **test**
        - **java**, contenente le classi per il testing.

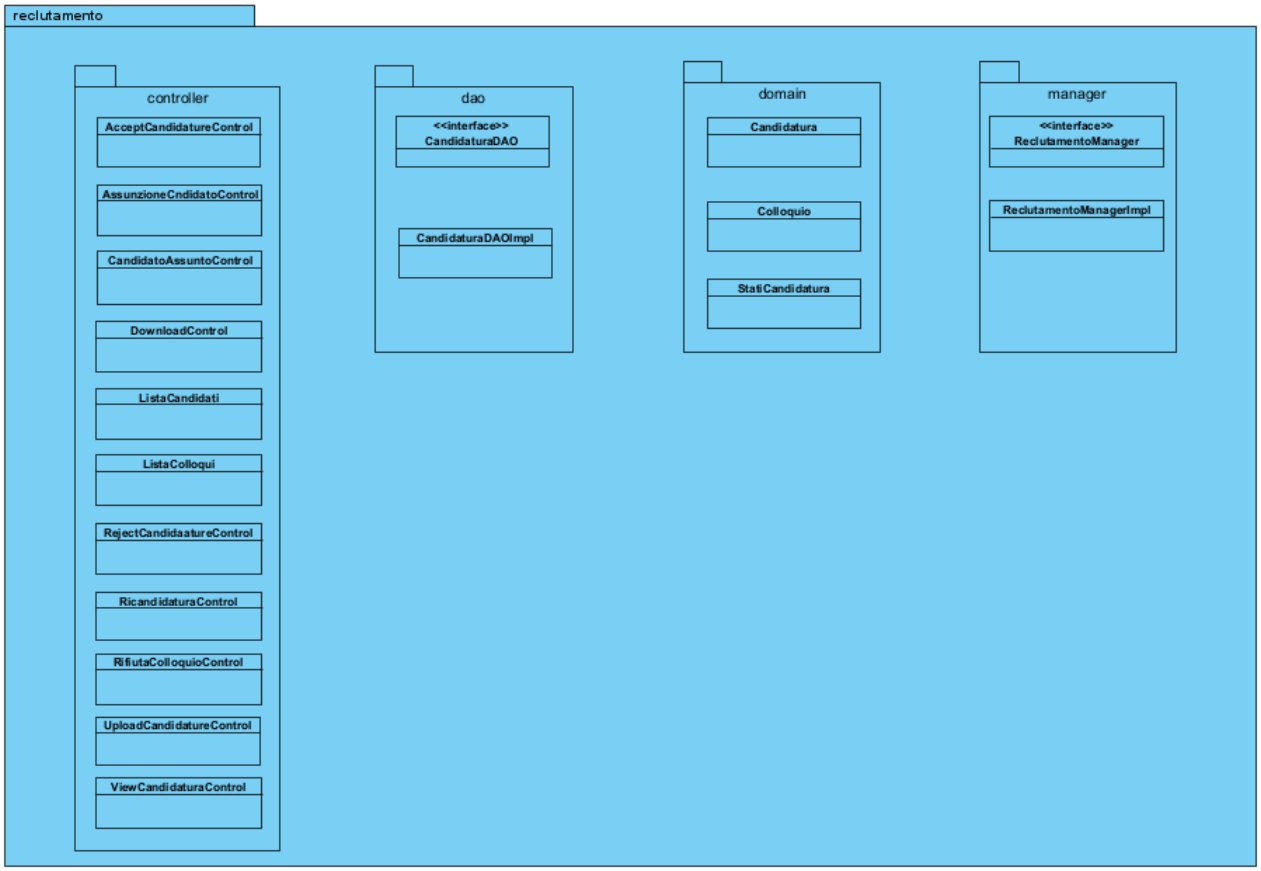
Nella directory principale, src/main/java, è presente la seguente suddivisione in package Java. La metodologia di visualizzazione della directory è impostata mostrando prima il top-level, per poi mostrare ciascun sotto package nel loro dettaglio.



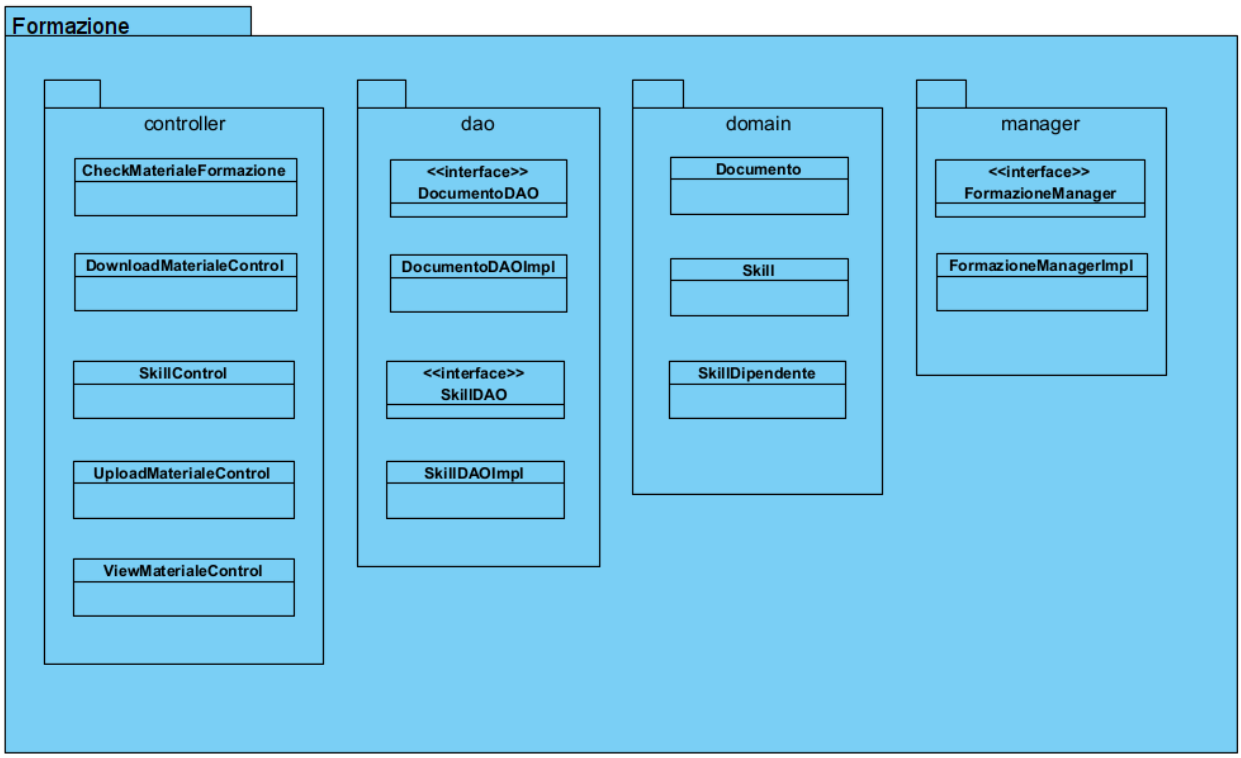
PD_1: *Struttura package presente in src/main/java.* **Autori:** GC, PS.



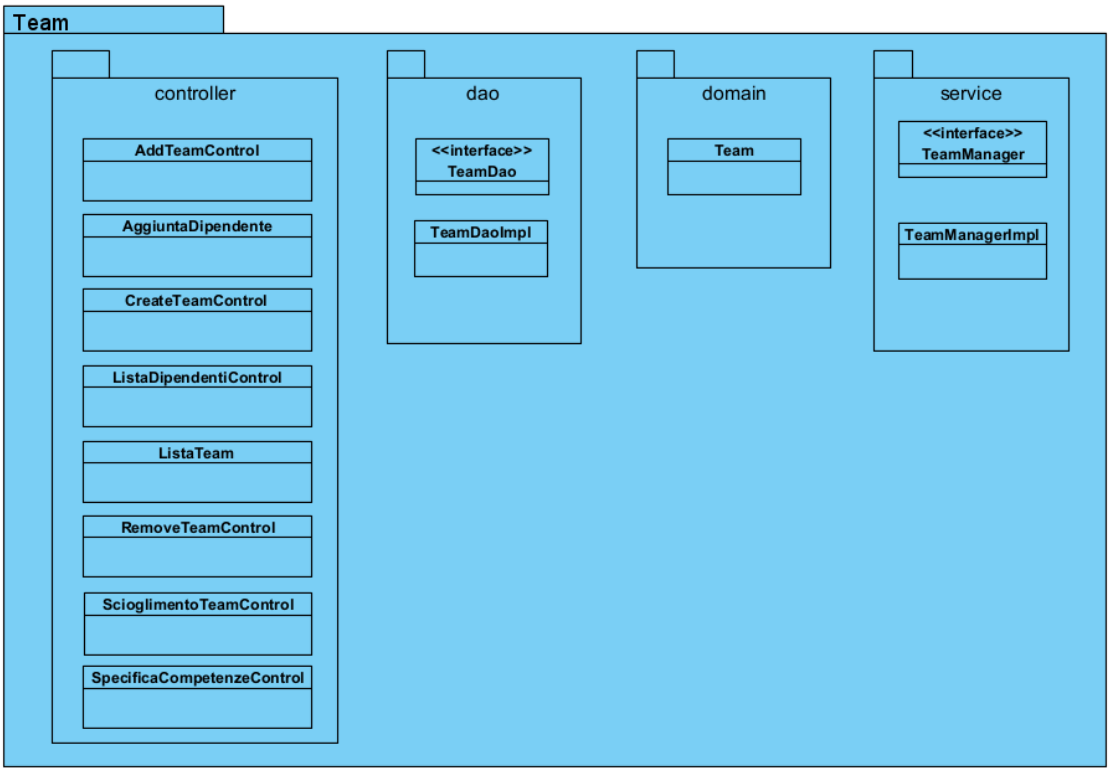
PD_2: *Struttura package Autenticazione.* **Autori:** GC, PS



PD_3: *Struttura package Reclutamento.* **Autori:** GC, PS.



PD_4: *Struttura package Formazione.* **Autori:** GC, PS.



PD_5: *Struttura package Team.* **Autori:** GC, PS.

3. Interfacce delle classi

Di seguito vengono presentate le interfacce delle classi di ciascun package. Non saranno riportate:

- Le classi di dominio (package domain), in quanto dispongono di soli costruttori, getter e setter.

3.1 Package agency_formation

3.1.1 it.unisa.agencyFormation.autenticazione.AutenticazioneControl

NOME CLASSE	LoginControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative all'autenticazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidaturafromManager(int idCandidato): Candidatura +loginFromManager(String email, String pwd): Utente
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla i valori presi dalle view per effettuare il login
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidaturafromManager(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ritornare la candidatura di uno specifico candidato utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +loginFromManager (String email, String pwd): Utente	
DESCRIZIONE	Questo metodo permette di effettuare il login di un utente utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	LogoutControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative all'autenticazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla i valori presi dalle view per effettuare il login
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/

NOME CLASSE	ProfiloControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative all'autenticazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getAllDataDipFromManager(int id): Dipendente +getSkillDipendenteFromManager(int idDip): ArrayList<Skill>
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per visualizzare e modificare il profilo
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getAllDataDipFromManager(int id): Dipendente	
DESCRIZIONE	Questo metodo permette di ottenere i dati del dipendente utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getSkillDipendenteFromManager(int idDip): ArrayList<Skill>	
DESCRIZIONE	Questo metodo permette di ottenere tutte le skill di un dipendente utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	RegistrazioneControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative all'autenticazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +registrazioneFromManager(Utente user): boolean +loginFromManager(Strin email, String pwd): Utente
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare una registrazione
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +registrazioneFromManager(Utente user): boolean	
DESCRIZIONE	Questo metodo permette di registrare un utente utilizzando il manager
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +loginFromManager(Strin email, String pwd): Utente	
DESCRIZIONE	Questo metodo permette di effettuare il login di un utente utilizzando il manager
PRE-CONDIZIONE	/
	/

3.1.2 it.unisa.agencyFormation.formazione.FormazioneControl

NOME CLASSE	CheckMaterialeFormazioneControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative alla formazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getDocumentoFromManager(int idTeam): Documento
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla che i documenti soddisfino i requisiti
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getDocumentoFromManager(int idTeam): Documento	
DESCRIZIONE	Questo metodo permette di ottenere il materiale di formazione di un team utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	DownloadMaterialeControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative alla formazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getDipendentefromManager(int idUtente): Dipendente +getDocumentofromManger(int idTeam): Documento
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare il download del materiale di formazione
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getDipendenteFromManager(int idUtente): Dipendente	
DESCRIZIONE	Questo metodo permette di ottenere un dipendente utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getDocumentofromManager(int idTeam): Documento	
DESCRIZIONE	Questo metodo permette di ottenere i documenti interessati utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	SkillControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative alla formazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +addSkillFromManager(Skill skill): Boolean +getLastIdSkillCreatedFromManager(): int +addSkillDipFromManager(int idSkill, int idDipendente, int skillLivello): boolean +getDipendenteByIdFromManager(int idDip): Dipendente
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare l'aggiunta di una skill
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +addSkillFromManager(Skill skill): boolean	
DESCRIZIONE	Questo metodo permette di aggiungere una nuova skill utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getLastIdSkillCreatedFromManager(): int	
DESCRIZIONE	Questo metodo permette di ottenere l'ultima skill creata utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +addSkillDipFromManager(int idSkill, int idDipendente, int skillLivello): boolean	
DESCRIZIONE	Questo metodo permette di aggiungere una skill posseduta dal dipendente utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getDipendenteByIdFromManager(int idDip): Dipendente	
DESCRIZIONE	Questo metodo permette di ottenere il dipendente attraverso il suo id utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	UploadMaterialeControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative alla formazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +saveDocumentFromManager(Documento document): boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare l'upload del materiale di formazione
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +saveDocumentFromManager(Documento document): boolean	
DESCRIZIONE	Questo metodo permette di salvare il documento utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	ViewMaterialeControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative alla formazione
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getDipendenteFromManger(int idUtente): Dipendente +getDocumentoFromManager(int idTeam): Documento
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per visualizzare il materiale di formazione
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getDipendenteFromManager(int idUtente): Dipendente	
DESCRIZIONE	Questo metodo permette di ottenere un dipendente attraverso il duo id utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getDocumentoFromManager(int idTeam): Documento	
DESCRIZIONE	Questo metodo permette di ottenere un documento appartenente ad un team utilizzando il manager
PRE-CONDIZIONE	/
	/

3.1.3 it.unisa.agencyFormation.reclutamento.control.ReclutamentoControl

NOME CLASSE	AcceptCandidatureControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidaturaFromManager(int idCandidato): Candidatura +acceptCandidatureFromManager(int idCandidatura, int idHR, Timestamp timestamp): boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare l'accettazione di una candidatura
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidaturaFromManager(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ottenere una candidatura attraverso l'id del candidato utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +acceptCandidatureFromManager(int idCandidatura, int idHR, int Timestamp timestamp): boolean	
DESCRIZIONE	Questo metodo permette di accettare la candidatura di un candidato utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	AssunzioneCandidatoControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidaturaFromManager(int idCandidato): Candidatura +setStatoFromManager(int idCandidatura): boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare l'assunzione di un candidato
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidaturaFromManager(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ottenere una candidatura attraverso l'id del candidato utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +setStatoFromManager(int idCandidatura): boolean	
DESCRIZIONE	Questo metodo permette di settare lo stato della candidatura utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	CandidatoAssuntoControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +assumiCandidatoFromManager(Dipendente dipendente): boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per completare l'assunzione del candidato rendendolo dipendente
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +assumiCandidatoFromManager(Dipendente dipendente): boolean	
DESCRIZIONE	Questo metodo permette di ottenere una candidatura attraverso l'id del candidato utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +setStatoFromManager(int idCandidatura): boolean	
DESCRIZIONE	Questo metodo permette di settare lo stato della candidatura utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	DownloadControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidaturaFromManager(int idCandidato): Candidatura
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare il download del curriculum e possibili documenti aggiuntivi
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidaturaFromManager(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ottenere la candidatura attraverso l'id del candidato attraverso il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	ListaCandidatiControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidatesFromManager(): ArrayList<Utente>
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la visualizzazione di tutti i candidati che hanno effettuato una candidatura
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidatesFromManager(): ArrayList<Utente>	
DESCRIZIONE	Questo metodo permette di ottenere i candidati che hanno eseguito la candidatura utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	ListaColloquiControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidatiForColloquioFromManager(): ArrayList<Utente>
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per visualizzare la lista di tutti i candidati che ottengono un colloquio
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidatiForColloquioFromManager(): ArrayLiist<Utente>	
DESCRIZIONE	Questo metodo permette di ottenere la lista di candidati che devono svolgere il colloquio utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	RejectCandidatureControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +rejectCandidatura(int idCandidatura, int idHR): Boolean +delete(File file): void +getCandidatura(int idCandidato): Candidatura
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare il rifiuto della candidatura
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +rejectCandidatura(int idCandidatura, int idHR): boolean	
DESCRIZIONE	Questo metodo permette di rifiutare una candidatura utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +delete(File file): void	
DESCRIZIONE	Questo metodo permette di eliminare un file
PRE-CONDIZIONE	/
	/
METODO +getCandidatura(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ottenere la candidatura di un candidato utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	RicandidaturaControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +eliminaCandidaturaFromManager(int idCandidato): boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la ricandidatura di un candidato
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +eliminaCandidaturaFromManager(int idCandidato): boolean	
DESCRIZIONE	Questo metodo permette di eliminare la candidatura di un candidato utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	RifiutaColloquioControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidaturaFromManager(int idCandidato): Candidatura +rejectCandidaturaFromManager(int idCandidatura, int idHR): boolean +delete(File file): void
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare il rifiuto di un candidato
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidaturaFromManager(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ottenere la candidatura di un candidato utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +rejectCandidaturaFromManager(int idCandidatura, int idHR): boolean	
DESCRIZIONE	Questo metodo permette di rifiutare la candidatura utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +delete(File file): void	
DESCRIZIONE	Questo metodo permette di eliminare un file
PRE-CONDIZIONE	/
	/

NOME CLASSE	UploadCandidaturaControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidaturaFromManager(int idCandidato): Candidatura +uploadCandidaturaFromManager(Candidatura candidatura): boolean +getCandidaturaByIdFromManager(int idCandidato): Candidatura
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare l'upload del curriculum e possibili documenti aggiuntivi
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidaturaFromManager(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ottenere la candidatura di un candidato utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +uploadCandidaturaFromManager(Candidtura canditura): boolean	
DESCRIZIONE	Questo metodo permette di caricare la candidatura utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getCandidaturaByIdFromManager(int idCandidato): Candidattura	
DESCRIZIONE	Questo metodo permette di ottenere la candidatura di un candidato utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	ViewCandidaturaControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getCandidaturaByIdFromManager(int idCandidato): Candidatura
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la visualizzazione dei dettagli di una candidatura
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getCandidaturaByIdFromManager(int idCandidato): Candidatura	
DESCRIZIONE	Questo metodo permette di ottenere la candidatura di un candidato utilizzando il manager
PRE-CONDIZIONE	/
	/

3.1.4 it.unisa.agencyFormation.team.control.TeamControl

NOME CLASSE	AddTeamControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +setTeamDipendenteFromManager(int idDip, int idTeam): boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare l'aggiunta di un team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +setTeamDipendenteFromManager(int idDip, int idTeam): boolean	
DESCRIZIONE	Questo metodo permette di aggiungere un dipendente in un team utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	AggiuntaDipendente
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getSkillDipendenteFromManager(int idDip): ArrayList<Skill> +getTuttiDipendentiFromManager(): ArrayList<Dipendente>
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare l'aggiunta di un dipendente nel team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getSkillDipendenteFromManager(int idDip): ArrayList<Skill>	
DESCRIZIONE	Questo metodo permette di ottenere le skill possedute da un dipendente utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getTuttiDipendentiFromManager(): ArrayList<Dipendente>	
DESCRIZIONE	Questo metodo permette di ottenere tutti i dipendenti utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	CreateTeamControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +createTeamFromManager(Team team, int idTM): Boolean +getIdUltimoTeamCreatoFromManager(): int
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la creazione del team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +createTeamFromManager(Team team, int idTM): boolean	
DESCRIZIONE	Questo metodo permette di creare un team utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getIdUltimoTTeamCreatoFromManager(): int	
DESCRIZIONE	Questo metodo permette di ottenere l'ultimo team creato utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	ListaDipendentiControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +getTeamIdFromManager(int idTeam): Team +getSkillDipendenteFromManager(int idDip): ArrayList<Skill> +getTuttiDipendentiFromManager(): ArrayList<Dipendente>
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la visualizzazione di tutti i dipendenti
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +getSkillDipendenteFromManager(int idDip): ArrayList<Skill>	
DESCRIZIONE	Questo metodo permette di ottenere le skill possedute da un dipendente utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getTuttiDipendentiFromManager(): ArrayList<Dipendente>	
DESCRIZIONE	Questo metodo permette di ottenere tutti i dipendenti utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getTeamIdFromManager(): Team	
DESCRIZIONE	Questo metodo permette di ottenere un team attraverso il suo id utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	ListaTeam
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +recuperoDipendentiDiUnTeamFromManager(): ArrayList<Dipendente> +visualizzaTeamOfTMFromManager(int idTM): ArrayList<Team> +getAllDipendentiFromManager(): ArrayList<Dipendente> +visualizzaTeamsForHRFromManager(): ArrayList<Team>
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la visualizzazione dei team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +recuperoDipendentiDiUnTeamFromManager(): ArrayList<Dipendente>	
DESCRIZIONE	Questo metodo permette di ottenere i dipendenti membri di un team utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +visualizzaTeamOfTMFromManager(int idTM): ArrayList<Team>	
DESCRIZIONE	Questo metodo permette di visualizzare i team creati da un TM utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +getAllDipendentiFromManager(): ArrayList<Dipendente>	
DESCRIZIONE	Questo metodo permette di ottenere tutti i dipendenti utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +visualizzaTeamsForHRFromManager(): ArrayList<Team>	
DESCRIZIONE	Questo metodo permette di visualizzare tutti i team utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	RemoveTeamControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +rimuoviDipendenteFromManager(int idDip): Boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la rimozione di un dipendente del team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +rimuoviDipendenteTeamFromManager(int idDip): boolean	
DESCRIZIONE	Questo metodo permette di rimuovere un dipendente utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	ScioglimentoTeamControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +recuperaIdDipendentiFromManager(int idTeam): ArrayList<Integer> +updateStatoDipendenteFromManager(int idDipendente): Boolean +eliminaTeamFromManager(int idTeam): Boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare lo scioglimento del team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +recuperaIdDipendentiTeamFromManager(int idTeam): ArrayList<Integer>	
DESCRIZIONE	Questo metodo permette di ottenere tutti gli id dei dipendenti membri di un team utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +updateStatoDipendenteFromManager(int idDipendente): boolean	
DESCRIZIONE	Questo metodo permette di modificare lo stato di un dipendente membro di un team utilizzando il manager
PRE-CONDIZIONE	/
	/
METODO +eliminaTeamFromManager(int idTeam): boolean	
DESCRIZIONE	Questo metodo permette di eliminare un team utilizzando il manager
PRE-CONDIZIONE	/
	/

NOME CLASSE	SpecificaCompetenzeControl
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team
METODI	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest req, HttpServletRespons resp): void +inserimentoCompetenzeNelTeam(String competenze, int idTeam): boolean
INVARIANTE DI CLASSE	/
METODO +doGet(HttpServletRequest request, HttpServletResponse response): void	
DESCRIZIONE	Questo metodo controlla le operazioni per effettuare la specifica delle competenze
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +doPost(HttpServletRequest req, HttpServletResponse resp): void	
DESCRIZIONE	Questo metodo richiama il doGet
PRE-CONDIZIONE	/
	/
METODO +inserimentoCompetenzeNelTeam(String competenze, int idTeam): boolean	
DESCRIZIONE	Questo metodo permette di specificare le competenze ad un team utilizzando il manager
PRE-CONDIZIONE	/
	/

3.1.5 it.unisa.agencyFormation.autenticazione.manager.AutenticazioneManager

NOME CLASSE	AutenticazioneManager
DESCRIZIONE	Questa classe permette di gestire le operazioni relative all'autenticazione
METODI	+registration(Utente user): boolean +login(String email, String password): Utente +getDipendente(idUser): Dipendente +getCandidatiConCandidatura(): ArrayList<Utente> +getTuttiDipendenti(): ArrayList<Dipendente> +getDipendentiByStato():ArrayList<Dipendente> +modificaRuolo(int idUtente): boolean +getCandidatiColloquio(): ArrayList<Utente> +setTeamDipendente(int dip, int idTeam): boolean
INVARIANTE DI CLASSE	/
METODO +registration(Utente user):boolean	
DESCRIZIONE	Questa funzionalità consente la registrazione di un Utente
PRE-CONDIZIONE	Context:AutenticazioneManager::registration(user):boolean pre: user!=null
POST-CONDIZIONE	/
METODO +login(String email, String password):Utente	
DESCRIZIONE	Questa funzionalità consente il login all'utente
PRE-CONDIZIONE	Context: AutenticazioneManager:: login(String email, String password): Utente Pre: email!=null && password != null
POST-CONDIZIONE	/
METODO +getDipendente(int idUser): Dipendente	
DESCRIZIONE	Questa funzionalità consente di ritornare un dipendente.
PRE-CONDIZIONE	Context: AutenticazioneManager:: getDipendente(int idUser): Dipendente Pre: idUser != null
POST-CONDIZIONE	/
METODO +getCandidatiConCandidatura(): ArrayList<Utente>	
DESCRIZIONE	Questa funzionalità consente di ritornare gli utenti che hanno effettuato una candidatura
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +getTuttiDipendenti(): ArrayList<Dipendente>	
DESCRIZIONE	Questa funzionalità consente di ritornare tutti i dipendenti.
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +getDipendentiByStato(): ArrayList<Dipendente>	

DESCRIZIONE	Questa funzionalità consente di ritornare tutti i dipendenti con un determinato stato.
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +modificaRuolo(): boolean	
DESCRIZIONE	Questa funzionalità consente di modificare un determinato ruolo.
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +getCandidatiColloquio(): ArrayList<Utente>	
DESCRIZIONE	Questa funzionalità consente di ritornare i candidati che dovranno sostenere un colloquio.
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +setTeamDipendente(int dip, int idTeam): boolean	
DESCRIZIONE	Questa funzionalità consente di settare il dipendente all'interno di un team.
PRE-CONDIZIONE	Context: AutenticazioneManager:: setTeamDipendente(int dip, int idTeam): boolean Pre: dip > 0 && idTeam!=0
POST-CONDIZIONE	/

3.1.6 it.unisa.agencyFormation.reclutamento.manager.ReclutamentoManager

NOME CLASSE	ReclutamentoManager
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al reclutamento
METODI	+caricaCandidatura(Candidatura candidatura): boolean +getCandidaturaById(int idCandidato): Candidatura +accettaCandidatura(int idCand): boolean +rifiutaCandidatura(int idCand): boolean +getTutteCandidature(): ArrayList<Candidatura> +rifiutaCandidato(int idCandidatura): boolean +ricandidatura(int idCandidato): boolean +getCandidatiConColloquio(StatiCandidatura stato): ArrayList<Candidatura> +modificaStatoCandidatura(int idCandidato, StatiCandidatura stato): boolean
INVARIANTE DI CLASSE	/
METODO + caricaCandidatura(Candidatura candidatura): boolean	
DESCRIZIONE	Questa funzionalità consente il caricamento di una candidatura.
PRE-CONDIZIONE	context: ReclutamentoManager:: caricaCandidatura(Candidatura candidatura): boolean pre: candidature!=null
POST-CONDIZIONE	/
METODO +getCandidaturaById(int idCandidato): Candidatura	
DESCRIZIONE	Questa funzionalità permette di ritornare una candidatura in base all'id dell'utente.
PRE-CONDIZIONE	context: ReclutamentoManager::getCandidaturaById(idUtente):Candidatura pre: (idCandidato!=null)
POST-CONDIZIONE	/
METODO +accettaCandidatura(int idCand): boolean	
DESCRIZIONE	Questa funzionalità consente di accettare una candidatura.
PRE-CONDIZIONE	context: ReclutamentoManager::accettaCandidatura(idCand): boolean pre: (idCand!=null)
POST-CONDIZIONE	/
METODO +rifiutaCandidatura(idCand): boolean	
DESCRIZIONE	Questa funzionalità consente di rifiutare una candidatura.
PRE-CONDIZIONE	context: ReclutamentoManager::rifiutaCandidatura(idCand): boolean pre: (idCand!=null)
POST-CONDIZIONE	/
METODO +getTutteCandidature(): ArrayList<Candidature>	
DESCRIZIONE	Questa funzionalità ritorna tutte le candidature.
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO	

+rifiutaCandidato(int idCandidatura): boolean	
DESCRIZIONE	Questa funzionalità consente il rifiuto di un candidato.
PRE-CONDIZIONE	context: ReclutamentoManager::rifiutaCandidato(int idCandidatura): boolean pre: (idCandidatura!=null)
POST-CONDIZIONE	/
METODO +ricandidatura(int idCandidato): boolean	
DESCRIZIONE	Questa funzionalità consente una ricandidatura
PRE-CONDIZIONE	context: ReclutamentoManager:: ricandidatura(int idCandidato): boolean pre: (idCandidato!=null)
POST-CONDIZIONE	/
METODO +getCandidatiConColloquio(StatiCandidatura stato): ArrayList<Candidatura>	
DESCRIZIONE	Questa funzionalità consente di ritornare i candidati che dovranno sostenere un colloquio.
PRE-CONDIZIONE	context: ReclutamentoManager::getCandidatiConColloquio(StatiCandidatura stato): ArrayList<Candidatura> pre: (stato!=null)
POST-CONDIZIONE	/
METODO +modificaStatoCandidatura(int idCandidato, StatiCandidatura stato): boolean	
DESCRIZIONE	Questa funzionalità consente la modifica dello stato di una candidatura.
PRE-CONDIZIONE	context: ReclutamentoManager:: modificaStatoCandidatura(idUtente): boolean pre: (idCandidato!=null && stato!=null)
POST-CONDIZIONE	/

3.1.7 it.unisa.agencyFormation.team.manager.TeamManager

NOME CLASSE	TeamManager
DESCRIZIONE	Questa classe permette di gestire le operazioni relative al team.
METODI	+creaTeam(Team team, int idUser): boolean +rimuoviDipendente(int idDip):boolean +visualizzaTeams(int idUtente): ArrayList<Team> +visualizzaTuttiTeams(): ArrayList<Team> +viewLastIdTeams(): int +recuperaIdDipendentiDelTeam():ArrayList<Integer> +updateDipsDisso(int idDip): boolean +sciogliTeam(int idTeam): boolean +recuperaDipendentiDelTeam(): ArrayList<Dipendente> +getTeamById(int idTeam): Team +modificaLeCompetenze(String competence, int idTeam): boolean
INVARIANTE DI CLASSE	/
METODO +creaTeam(Team team, int idUser):Team	
DESCRIZIONE	Questa funzionalità permette la creazione di un team.
PRE-CONDIZIONE	context: TeamManager:: creaTeam(Team team, int idUser):Team pre: idUser != null
POST-CONDIZIONE	/
METODO +rimuoviDipendente(int idDip):boolean	
DESCRIZIONE	Questa funzionalità permette di rimuovere un dipendente
PRE-CONDIZIONE	context: TeamManager:: rimuoviDipendente(int idDip): boolean pre: (idDip!=null)
POST-CONDIZIONE	/
METODO +visualizzaTeams(int idUtente): ArrayList<Team>	
DESCRIZIONE	Questa funzionalità permette di visualizzare un team
PRE-CONDIZIONE	context: TeamManager::visualizzaTeams(int idUtente):ArrayList<Team> pre: idUtente != null
POST-CONDIZIONE	/
METODO +visualizzaTuttiTeams(): ArrayList<Team>	
DESCRIZIONE	Questa funzionalità permette di visualizzare tutti i team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +viewLastIdTeam(): int	
DESCRIZIONE	Questa funzionalità permette di visualizzare l'ultimo team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +recuperaIdDipendentiDelTeam(int idTeam):ArrayList<Integer>	

DESCRIZIONE	Questa funzionalità permetti di recuperare un dipendente tramite il suo id.
PRE-CONDIZIONE	Context: TeamManager:: recuperaIdDipendentiDelTeam(int idTeam): ArrayList<Integer> Pre: idTeam != null
POST-CONDIZIONE	/
METODO +updateDipsDiss(int idDip): boolean	
DESCRIZIONE	Questa funzionalità permette di settare lo stato in "disponibile"
PRE-CONDIZIONE	context: TeamManager::updateDipsDiss(int idDip): boolean pre: idDip!=null
POST-CONDIZIONE	
METODO +sciogliTeam(int idTeam): boolean	
DESCRIZIONE	Questa funzionalità permette di sciogliere un team da parte di un TM.
PRE-CONDIZIONE	context: TeamManager:sciogliTeam(idTeam):boolean pre: idTeam!=null
POST-CONDIZIONE	/
METODO +recuperaDipendentiDelTeam(): ArrayList<Dipendente>	
DESCRIZIONE	Questa funzionalità permette di recuperare i dipendenti di un team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +getTeamById(int idTeam): Team	
DESCRIZIONE	Questa funzionalità ritorna un team tramite l'id
PRE-CONDIZIONE	context: TeamManager::getTeamById(idTeam):Team pre: idTeam!=null
POST-CONDIZIONE	/
METODO +modificaLeCompetenze(String competence, int idTeam): boolean	
DESCRIZIONE	Questa funzionalità permette di modificare le competenze
PRE-CONDIZIONE	context: TeamManager::modificaLeCompetenze(String competence, int idTeam): boolean pre: competence!=null && idTeam!=null
POST-CONDIZIONE	

3.1.8 it.unisa.agencyFormation.formazione.manager.FormazioneManager

NOME CLASSE	FormazioneManager
DESCRIZIONE	Questa classe permette di gestire le operazioni relative alla formazione
METODI	+creaCompetenza(int idTeam, String competenza):void +salvaDocumento(Documento documento): boolean +visualizzaCompetenza(int idTeam): String +caricaDocumenti(String MaterialeDiFormazione): void +getMaterialeByTeam(int idTeam): Documento +aggiungiSkill(Skill skill): boolean +visualizzaSkill(int idSkill): void +getUltimaSkill(int idSkill): int +addSkillDipendente(int idSkill, int iddip, int skillLivello): boolean +recuperoSkillConIdDipendente(int idDipendentel): ArrayLst<Skill>
INVARIANTE DI CLASSE	/
METODO +creaCompetenza(int idTeam, String Competenza):void	
DESCRIZIONE	Questa funzionalità permette la creazione di una competenza richiesta.
PRE-CONDIZIONE	context: FormazioneManager:: creaCompetenza(idTeam, String competenza):void pre: idTeam!=null && competenza!=null
POST-CONDIZIONE	/
METODO +salvaDocumento(Documento documento): boolean	
DESCRIZIONE	Questa funzionalità permette il salvataggio di un documento
PRE-CONDIZIONE	context: FormazioneManager:: salvaDocumento(Documento documento): boolean pre: documento != null
POST-CONDIZIONE	post: documento != null
METODO +visualizzaCompetenza(int idTeam): String	
DESCRIZIONE	Questa funzionalità permette la visualizzazione delle competenze.
PRE-CONDIZIONE	context: FormazioneManager:: visualizzaCompetenza(int idTeam): String pre: idTeam != null
POST-CONDIZIONE	
METODO +caricaDocumenti(String MaterialeDiFormazione): void	
DESCRIZIONE	Questa funzionalità permette di caricare i documenti
PRE-CONDIZIONE	context: FormazioneManager:: caricaDocumenti(String MaterialeDiFormazione):void pre: MaterialeFormazione !=null && MaterialeFormazione > 0
POST-CONDIZIONE	/
METODO +getMaterialeByIdTeam(int idTeam): Documento	
DESCRIZIONE	Questa funzionalità ritorna il materiale di formazione tramite l'id del team
PRE-CONDIZIONE	context: FormazioneManager:: getMaterialeByIdTeam(int idTeam): Documento

	pre: idTeam != null
POST-CONDIZIONE	/
METODO +aggiungiSkill(Skill skill): boolean	
DESCRIZIONE	Questa funzionalità permette l'aggiunta di una skill.
PRE-CONDIZIONE	context: FormazioneManager:: aggiumgiSkill(Skill skill): boolean pre: skill!=null
POST-CONDIZIONE	/
METODO +visualizzaSkill(int idSkill): void	
DESCRIZIONE	Questa funzionalità permette la visualizzazione di una skill.
PRE-CONDIZIONE	context: FormazioneManager:: visualizzaSkill(int idSkill):void pre: idSkill != null
POST-CONDIZIONE	/
METODO +getUltimaSkill(): int	
DESCRIZIONE	Questa funzionalità permette di ritornare l'ultima skill
PRE-CONDIZIONE	context: FormazioneManager:: getUltimaSkill(): int /
POST-CONDIZIONE	/
METODO +addSkillDipendente(int idSkill, int iddip, int skillLivello): boolean	
DESCRIZIONE	Questa funzionalità permette l'aggiunta di una skill da parte di un dipendente
PRE-CONDIZIONE	context: FormazioneManager:: addSkillDipendente(int idSkill. Int iddip, int skillLivello): boolean pre: idSkill != null
POST-CONDIZIONE	/
METODO +recuperoSkillConIdDipendente(int idDipendente): ArrayList<Skill>	
DESCRIZIONE	Questa funzionalità permette il recupero di una skill
PRE-CONDIZIONE	context: FormazioneManager:: recuperoSkillConIdDipendente(int idDipendente):ArrayList<Skill> pre: idDipendente != 0
POST-CONDIZIONE	/

3.1.9 it.unisa.agencyFormation.autenticazione.dao.UtenteDAO

NOME CLASSE	UtenteDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Utente.
METODI	+salvaUtente(Utente user): Boolean +login(String email, String pwd): Utente +doRetrieveUtenteByID(int idUtente): Utente +doRetrieveCandidatoConCandidatura(): ArrayList<Utente> +recuperoCandidatiColloquio(): ArrayList<Utente>
INVARIANTE DI CLASSE	/
METODO +salvaUtente(Utente user): boolean	
DESCRIZIONE	Questa funzionalità permette di salvare un utente.
PRE-CONDIZIONE	Context:UtenteDAO:: salvaUtente(Tente user): boolean Pre: user != null
POST-CONDIZIONE	/
METODO +login(String email,String pwd): Utente	
DESCRIZIONE	Questa funzionalità permette all'utente di effettuare il login
PRE-CONDIZIONE	Context: UtenteDAO:: login(String email, String pwd):Utente Pre: email != null && password != null
POST-CONDIZIONE	/
METODO +doRetrieveUtenteByID(int idUtente): Utente	
DESCRIZIONE	Questa funzionalità permette di recuperare un utente attraverso il suo id
PRE-CONDIZIONE	Context: UtenteDAO::doRetrieveUtenteByID(int idUtente): Utente Pre: id > 0;
POST-CONDIZIONE	/
METODO +doRetrieveCandidatoConCandidatura(): ArrayList<Utente>	
DESCRIZIONE	Questa funzionalità permette di recuperare un utente che ha presentato la propria candidatura
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +recuperoCandidatiColloquio(): ArrayList<Utente>	
DESCRIZIONE	Questa funzionalità permette di recuperare i candidati che dovranno sostenere un colloquio
PRE-CONDIZIONE	/
POST-CONDIZIONE	/

3.2.0 it.unisa.agencyFormation.autenticazione.dao.DipendenteDAO

NOME CLASSE	DipendenteDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Dipendente.
METODI	+ salvaDipendente(Dipendente dipendente): boolean +modificaRuoloUtente(int id): boolean + doRetrieveDipendenteById(int id): Dipendente +recuperaDipendenti(): ArrayList<Dipendente> +recuperaByStato(StatiDipendenti stato): ArrayList<Dipendente> +setTeamDipendente(int idDip, int idTeam): boolean
INVARIANTE DI CLASSE	/
METODO +salvaDipentente(Dipendente dipendente): boolean	
DESCRIZIONE	Questa funzionalità permette di salvare un dipendente
PRE-CONDIZIONE	Context: DipendenteDAO:: salvaDipendente(Dipendente dipendente): boolean Pre: dip !=null
POST-CONDIZIONE	/
METODO + modificaRuoloUtente(int id): boolean	
DESCRIZIONE	Questa funzionalità permette di modificare il ruolo di un utente
PRE-CONDIZIONE	Context: DipendenteDAO:: modificaRuoloUtente(int id): boolean Pre: id > 0
POST-CONDIZIONE	/
METODO +doRetrieveDipendenteById(int id): Dipendente	
DESCRIZIONE	Questa funzionalità permette di recuperare un dipendente tramite l'id
PRE-CONDIZIONE	Context: doRetrieveDipendenteById(int id): Dipendente Pre: id> 0
POST-CONDIZIONE	/
METODO +recuperaDipendenti(): ArrayList<Dipendente>	
DESCRIZIONE	Questa funzionalità permette di recuperare tutti i dipendenti
PRE-CONDIZIONE	/
POST-CONDIZIONE	Context: DipendenteDAO::recuperaDipendenti(): ArrayList<Dipendente> Post: dipendenti.size() > 0
METODO +recuperaByStato(StatiDipendenti stato): ArrayList<Dipendente>	
DESCRIZIONE	Questa funzionalità permette di recuperare un dipendente attraverso lo stato
PRE-CONDIZIONE	/
POST-CONDIZIONE	Context: DipendenteDAO:: recuperaByStato(int dip, int idTeam): boolean Post:dipendenti.size() > 0
METODO +setTeamDipendente(int idDip, int idTeam): boolean	
DESCRIZIONE	Questa funzionalità permette di settare il dipendente di un team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/

3.2.1 it.unisa.agencyFormation.reclutamento.dao.CandidaturaDAO

NOME CLASSE	CandidaturaDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Candidatura.
METODI	+ salvaCandidaturaSenzaDocumenti(Candidatura candidatura): boolean + aggiungiDocumentiAggiuntivi(String document, int idUtente): boolean +doRetrieveCandidaturaById(int idCandidato): Candidatura +recuperaCandidature(): ArrayList<Candidatura> +recuperaCandidatureByStato(StatiCandidature stato): ArrayList<Candidatura> +modificaStatoCandidatura(int idCandidatura, StatiCandidatura stato): boolean +rimuoviCandidatura(int idCandidato): boolean +rifiutaCandidatura(int idCandidatura, int idHR): boolean +accettaCandidatura(int idCandidatura, int idHr, Timestamp data): boolean
INVARIANTE DI CLASSE	/
METODO + salvaCandidaturaSenzaDocumenti(Candidatura candidatura): boolean	
DESCRIZIONE	Questa funzionalità permette di salvare una candidatura
PRE-CONDIZIONE	Context: CandidaturaDAO::salvaCandidaturaSenzaDocumenti(Candidatura candidatura): boolean Pre: candidatura != null
POST-CONDIZIONE	/
METODO + aggiungiDocumentiAggiuntivi(String document, int idUtente): boolean	
DESCRIZIONE	Questa funzionalità permette di aggiungere un documento
PRE-CONDIZIONE	Context: CandidaturaDAO::aggiungiDocumentiAggiuntivi(String document, int idUtente): boolean Pre: document!= null && idUtente>0
POST-CONDIZIONE	/
METODO +doRetrieveCandidaturaById(int idCandidato): Candidatura	
DESCRIZIONE	Questa funzionalità permette di recuperare la candidatura tramite l'id
PRE-CONDIZIONE	Context: CandidaturaDAO:: doRetrieveCandidaturaById(int idCandidatura): Candidatura Pre: idCandidato>0
POST-CONDIZIONE	/
METODO +recuperaCandidature(): ArrayList<Candidatura>	
DESCRIZIONE	Questa funzionalità permette di recuperare le candidature
PRE-CONDIZIONE	Context: CandidaturaDAO:: recuperaCandidature(): ArrayList<Candidatura>
POST-CONDIZIONE	Post: candidature.size()>0
METODO +recuperaCandidatureByStato(StatiCandidature stato): ArrayList<Candidature>	
DESCRIZIONE	Questa funzionalità permette di recuperare una candidatura tramite lo stato

PRE-CONDIZIONE	Context: CandidaturaDAO:: recuperaCandidatureByStato(StatiCandidature stato): ArrayList<Candidature> Pre: stato != null
POST-CONDIZIONE	/
METODO +modificaStatoCandidatura(int idCandidatura, StatiCandidatura stato): boolean	
DESCRIZIONE	Questa funzionalità permette di modificare lo stato di una candidatura
PRE-CONDIZIONE	Context: CandidaturaDAO:: modificaStatoCandidatura(int idCandidatura, StatiCandidatura sato): boolean Pre: stato != null && idCandidatura>0
POST-CONDIZIONE	/
METODO +rimuoviCandidatura(int idCandidato): boolean	
DESCRIZIONE	Questa funzionalità permette di rimuovere una candidatura
PRE-CONDIZIONE	Context: CandidaturaDAO:: rimuoviCandidatura(int idCandidato): boolean Pre: idCandidatura>0
POST-CONDIZIONE	/
METODO +rifiutaCandidatura(int idCandidatura, int idHr): boolean	
DESCRIZIONE	Questa funzionalità permette di rifiutare una candidatura
PRE-CONDIZIONE	Context: CandidaturaDAO:: rifiutaCandidatura(int idCandidato): boolean Pre: idCandidatura >0
POST-CONDIZIONE	/
METODO +accettaCandidatura(int idCandidatura, int idHr, Timestamp data): boolean	
DESCRIZIONE	Questa funzionalità permette di rifiutare una candidatura
PRE-CONDIZIONE	/
POST-CONDIZIONE	/

3.2.2 it.unisa.agencyFormation.team.dao.TeamDAO

NOME CLASSE	TeamDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Team
METODI	+ salvaTeam(Team team, int idUtente): boolean + rimuoviTeam(int idTeam):boolean +recuperaTeamById(int idTeam): Team +rimuoviDipendente(int idDipendente): boolean +recuperaTuttiTeam(): ArrayList<Team> +recuperaTeamDiUnTM(int idUtente): ArrayList<Team> +modificaCompetenze (String competence, int idTeam): boolean +recuperaCompetenze(int idTeam): String +recuperaIdUltimoTeamCreato(): int +recuperaIdTeamMemberFromTeam(int idTeam): ArrayList<Integer> +updateDipStateDissolution(int idDip): boolean +recuperaDipendentiDelTeam(): ArrayList<Dipendente>
INVARIANTE DI CLASSE	/
METODO +salvaTeam(Team team, int idUtente): boolean	
DESCRIZIONE	Questa funzionalità permette di salvare un team
PRE-CONDIZIONE	Context: TeamDAO:: salvaTeam(Team team, int idUtente): boolean Pre: team >0
POST-CONDIZIONE	/
METODO +rimuoviTeam(int idTeam): boolean	
DESCRIZIONE	Questa funzionalità permette di rimuovere un team
PRE-CONDIZIONE	Context: TeamDAO:: rimuoviTeam(idTeam): boolean Pre: idTeam > 0
POST-CONDIZIONE	/
METODO +recuperaTeamById(int idTeam): Team	
DESCRIZIONE	Questa funzionalità permette di recuperare un team tramite l'id
PRE-CONDIZIONE	Context: TeamDAO:: recuperaTeamById(idTeam,): Team /
POST-CONDIZIONE	/
METODO +rimuoviDipendente(int idDipendente): boolean	
DESCRIZIONE	Questa funzionalità permette di rimuovere dipendente da un team
PRE-CONDIZIONE	Context: TeamDAO:: rimuoviDipendente(idDipendente): boolean Pre: idTeam >0 && idDipendente >0
POST-CONDIZIONE	/
METODO +recuperaTuttiTeam():ArrayList<Team>	
DESCRIZIONE	Questa funzionalità permette di recuperare tutti i team presenti nella piattaforma

PRE-CONDIZIONE	Context TeamDAO:: recuperaTuttiITeam(): ArrayList<Team>
POST-CONDIZIONE	Post: teams.size>0
METODO +recuperaTeamDiUnTM(int idUtente):ArrayList<Team>	
DESCRIZIONE	Questa funzionalità permette di recuperare la lista dei team di un TM
PRE-CONDIZIONE	Context: TeamDAO:: recuperaTeamDiUnTM(idUtente):ArrayList<Team> Pre: idTM >0
POST-CONDIZIONE	/
METODO +modificaCompetenze(String competence, int idTeam): boolean	
DESCRIZIONE	Questa funzionalità permette di modificare le competenze di un team
PRE-CONDIZIONE	Context: TeamDAO:: modificaCompetenza(String competence, int idTeam): boolean Pre: competence != null && idTeam >0
POST-CONDIZIONE	/
METODO +recuperaCompetenze(int idTeam):String	
DESCRIZIONE	Questa funzionalità permette di recuperare le competenze specificate di un team
PRE-CONDIZIONE	Context: TeamDAO:: recuperaCompetenza(idTeam):String Pre: idTeam > 0
POST-CONDIZIONE	/
METODO +recuperaIdUltimoTeamCreato(): int	
DESCRIZIONE	Questa funzionalità permette di recuperare l'id dell'ultimo team creato
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +recuperaIdTeamMemberFromTeam(int idTeam): ArrayList<Integer>	
DESCRIZIONE	Questa funzionalità permette di recuperare il membro di un team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +updateDipStateDissolution(int idDip): boolean	
DESCRIZIONE	Questa funzionalità permette di aggiornare lo stato di un dipendente dopo lo scioglimento di un team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +recuperaDipendentiDelTeam(): ArrayList<Dipendente>	
DESCRIZIONE	Questa funzionalità permette di recuperare i membri in un team
PRE-CONDIZIONE	/
POST-CONDIZIONE	/

3.2.3 it.unisa.agencyFormation.formazione.dao.DocumentoDAO

NOME CLASSE	DocumentoDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Documento
METODI	+salvaDocumento(Documento doc):void +rimuoviDocumento(String materialeFormazione): boolean +recuperaDocumentoByTeam(int idTeam): Documento
INVARIANTE DI CLASSE	/
METODO +salvaDocumento(Documento doc):void	
DESCRIZIONE	Questa funzionalità permette di salvare un documento
PRE-CONDIZIONE	Context: DocumentoDAO:: salvaDocumento(Documento doc):void Pre: doc != null
POST-CONDIZIONE	/
METODO +rimuoviDocumento(String materialeFormazione): boolean	
DESCRIZIONE	Questa funzionalità permette di eliminare un documento
PRE-CONDIZIONE	Context: DocumentoDAO:: rimuoviDocumento(String MaterialeFormazione):boolean Pre: matForm != null
POST-CONDIZIONE	/
METODO +recuperaDocumentoByTeam(int idTeam): Documento	
DESCRIZIONE	Questa funzionalità permette di recuperare un documento tramite l'id id un team
PRE-CONDIZIONE	Context: DocumentoDAO:: recuperaDocumentoByTeam(idTeam): Documento Pre: idTeam > 0
POST-CONDIZIONE	/

3.2.4 it.unisa.agencyFormation.autenticazione.dao.SkillDAO

NOME CLASSE	SkillDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Skill
METODI	+salvaSkill(Skill skill): boolean +rimuoviSkill(int IdSkill):void +recuperaSkills():ArrayList<Skill> +recuperaSkillByNome(String nomeSkill): Skill +salvaSkillDipendente(int idSkill, int idDip, int skillLivello): boolean +recuperaUltimaSkill(): int +recuperoSkillsByIdDipendente(int idDip): ArrayList<Skill>
INVARIANTE DI CLASSE	/
METODO +salvaSkill(String skill): boolean	
DESCRIZIONE	Questa funzionalità permette di salvare una nuova skill
PRE-CONDIZIONE	Context: SkillDAO:: doSaveSkill(skill):void Pre: skill != null && dip!null
POST-CONDIZIONE	/
METODO +rimuoviSkill(int IdSkill): void	
DESCRIZIONE	Questa funzionalità permette di rimuovere una skill persa
PRE-CONDIZIONE	Context: SkillDAO:: doRemoveDocument(IdSkill):void Pre: IdSkill > 1
POST-CONDIZIONE	
METODO +recuperaSkills():ArrayList<Skill>	
DESCRIZIONE	Questa funzionalità permette di recuperare tutte le skill
PRE-CONDIZIONE	Context SkillDAO:: recuperaSkills(): ArrayList<Skill> Pre: Skills.size()>0
POST-CONDIZIONE	/
METODO +recuperaSkillsByNome(String nomeSkill): Skill	
DESCRIZIONE	Questa funzionalità permette di recuperare skill in base al nome
PRE-CONDIZIONE	Context SkillDAO::recuperaSkillsByNome(String nomeSkill): Skill Pre: nomeSkill!=null
POST-CONDIZIONE	/
METODO +salvaSkillDipendente(int idSkill, int idDip, int skillLivello): boolean	
DESCRIZIONE	Questa funzionalità permette di salvare le skill del dipendente
PRE-CONDIZIONE	Context SkillDAO:: salvaSkillDipendente(int idSkill, int idDip, int skillLivello): boolean Pre: idSkill > 0 && dip!=null
POST-CONDIZIONE	/
METODO +recuperaUltimaSkill(): int	
DESCRIZIONE	Questa funzionalità permette di recuperare l'ultima skill
PRE-CONDIZIONE	/

POST-CONDIZIONE	/
METODO +recuperaSkillsByIdDipendente(int idDip): ArrayList<Skill>	
DESCRIZIONE	Questa funzionalità permette di recuperare la skill di un dipendente
PRE-CONDIZIONE	/
POST-CONDIZIONE	/

4. Design pattern con class diagram

In questo capitolo vengono presentati i design pattern utilizzati e i class diagram relativi alle classi del sistema.

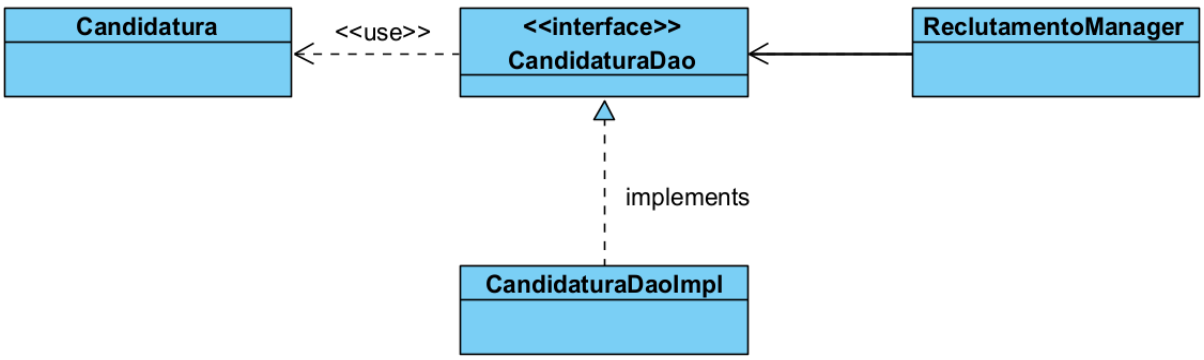
4.1. Design Pattern

4.1.1. Façade

Per realizzare i servizi dei sottosistemi è stato usato il pattern *Façade*. Questo permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi. Possiamo notare alcuni vantaggi che offre questo pattern: l'agevolazione su eventuali cambiamenti e la riduzione del numero delle associazioni; entrambi presi in considerazione. In questo contesto, ogni sottosistema possiede un sottopackage chiamato *services*. Quest'ultimo contiene le classi che implementano i metodi che corrispondono ai servizi offerti dal sottosistema. Questi, nella loro implementazione, utilizzeranno le classi nei package *domain* e *dao*, così si ottiene una maggior separazione della logica di controllo dalla logica di business.

4.1.2. Dao

Usiamo il pattern Data Access Object (*DAO*), un pattern architetturale utilizzato per separare i servizi di business dell'application processing layer dalle operazioni di accesso ai dati del data management layer. Il DAO implementa il meccanismo di accesso richiesto per lavorare con la sorgente dei dati e utilizza l'interfaccia esposta dal DAO. Il Data Access Object nasconde completamente i dettagli dell'interazione con la sorgente dati. In sostanza, il DAO funge da "adattatore" tra il componente della business logic e l'origine dati. Abbiamo pensato di utilizzare tale pattern per la stratificazione delle operazioni effettuabili sugli oggetti Entity, al fine di ottimizzare l'implementazione delle classi e la manutenibilità di esse. Di seguito un esempio d'uso del pattern DAO.

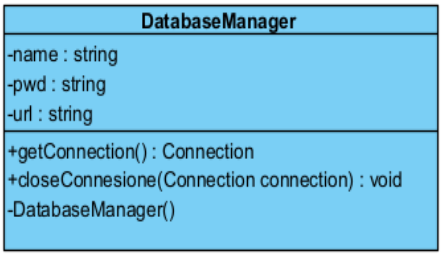


DP_1: *Diagramma design pattern DAO*. **Autori:** GC, PS.

In questa figura, la classe *ReclutamentoManager* rimane completamente all'oscuro su come vengono eseguite le operazioni di accesso ai dati persistenti.

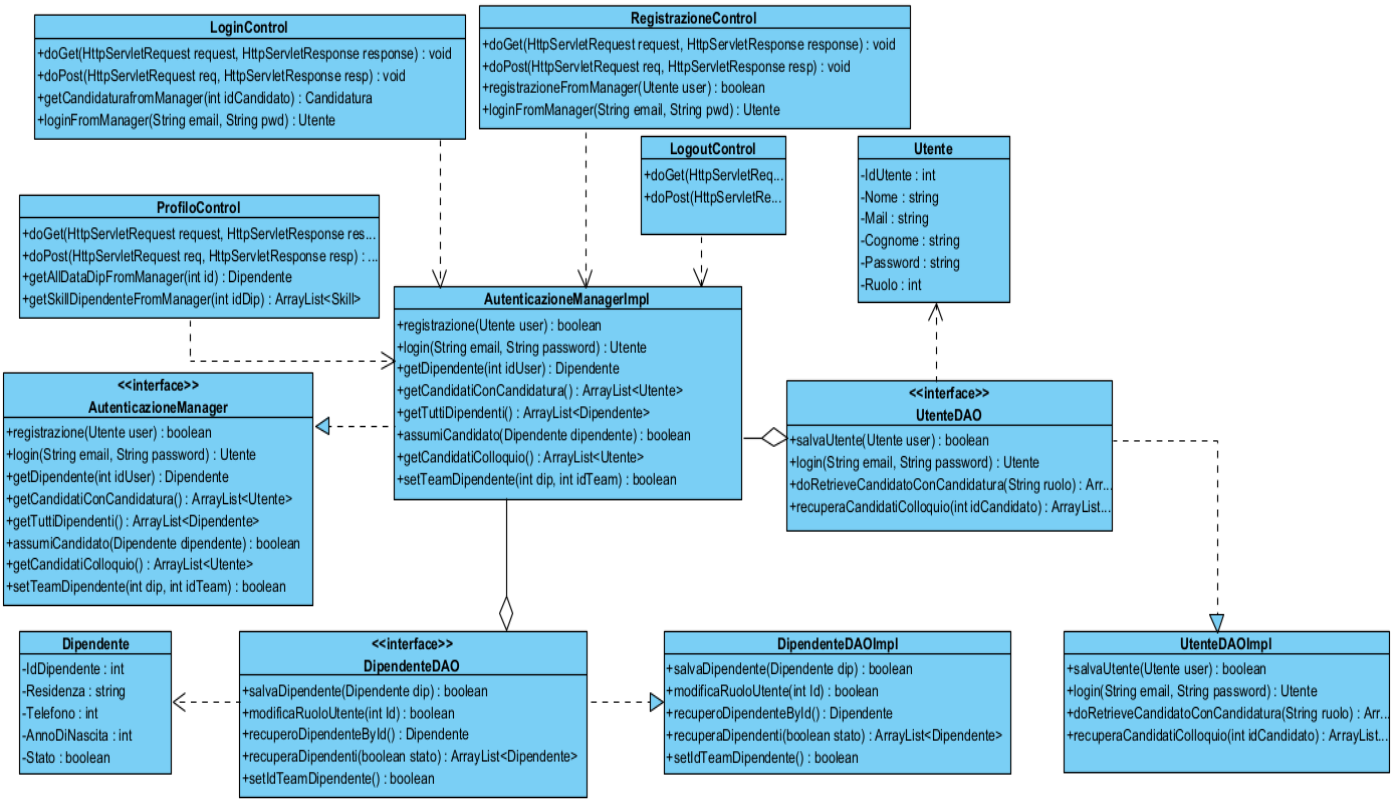
Singleton

Abbiamo progettato una singola classe che consente di effettuare tutte le operazioni con il database. Per evitare la perdita di efficienza dovuta alla creazione di più istanze di questa classe si è deciso di renderla un Singleton. Di seguito un esempio d'uso del pattern Singleton.

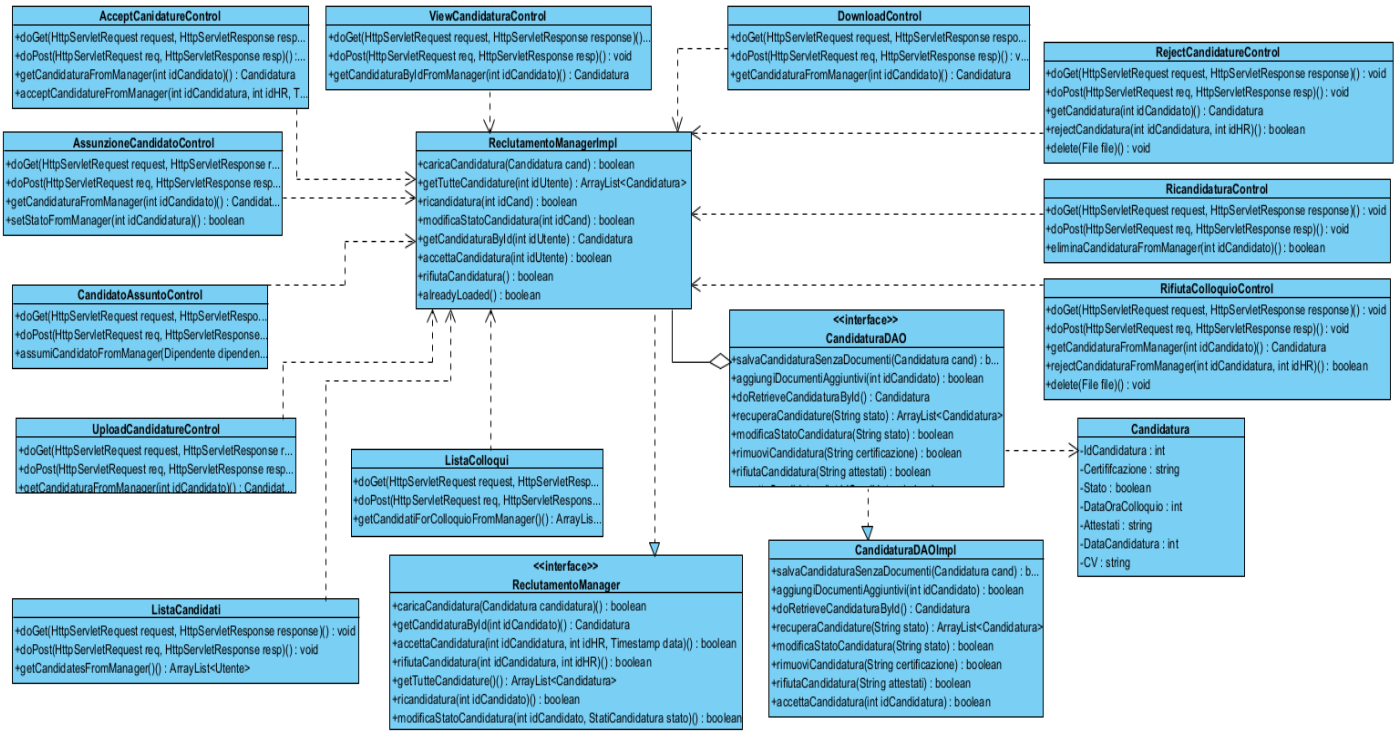


DP_2: *Diagramma design pattern Singleton.* **Autori:** GC, PS.

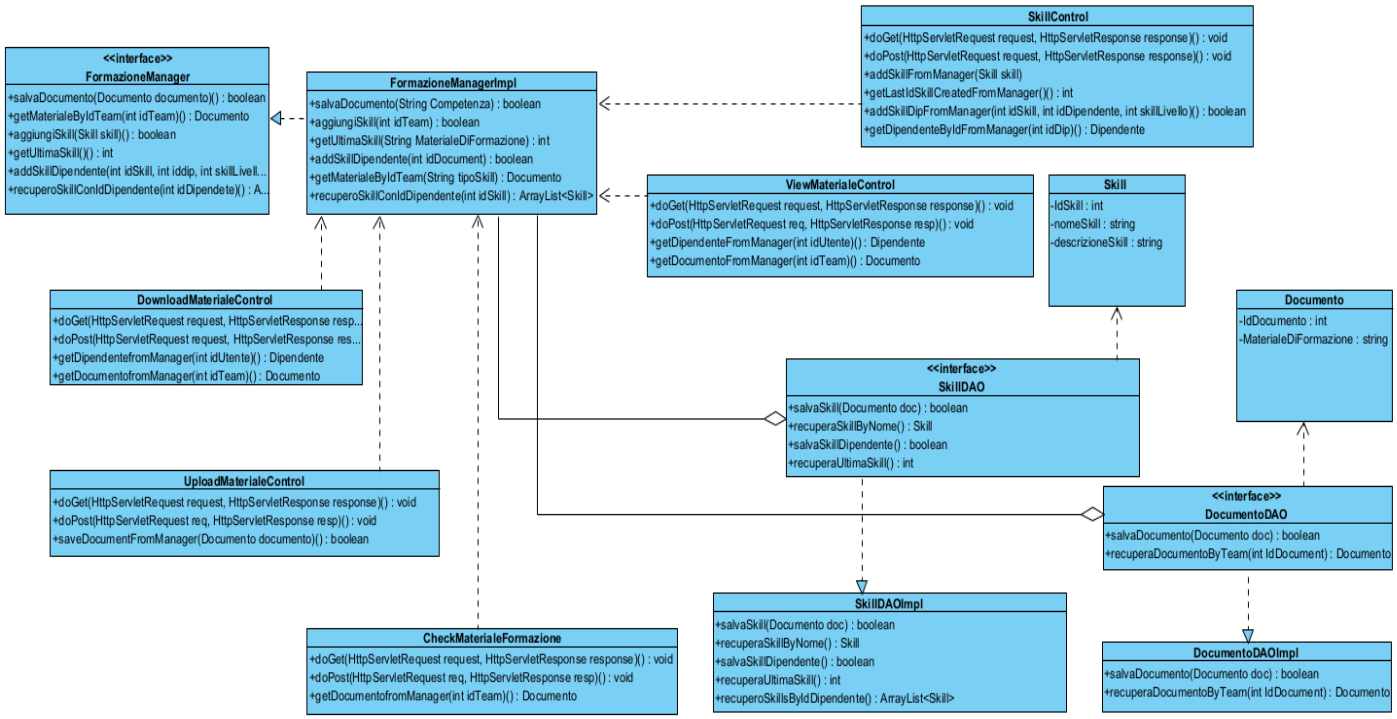
4.1.4 Class Diagram



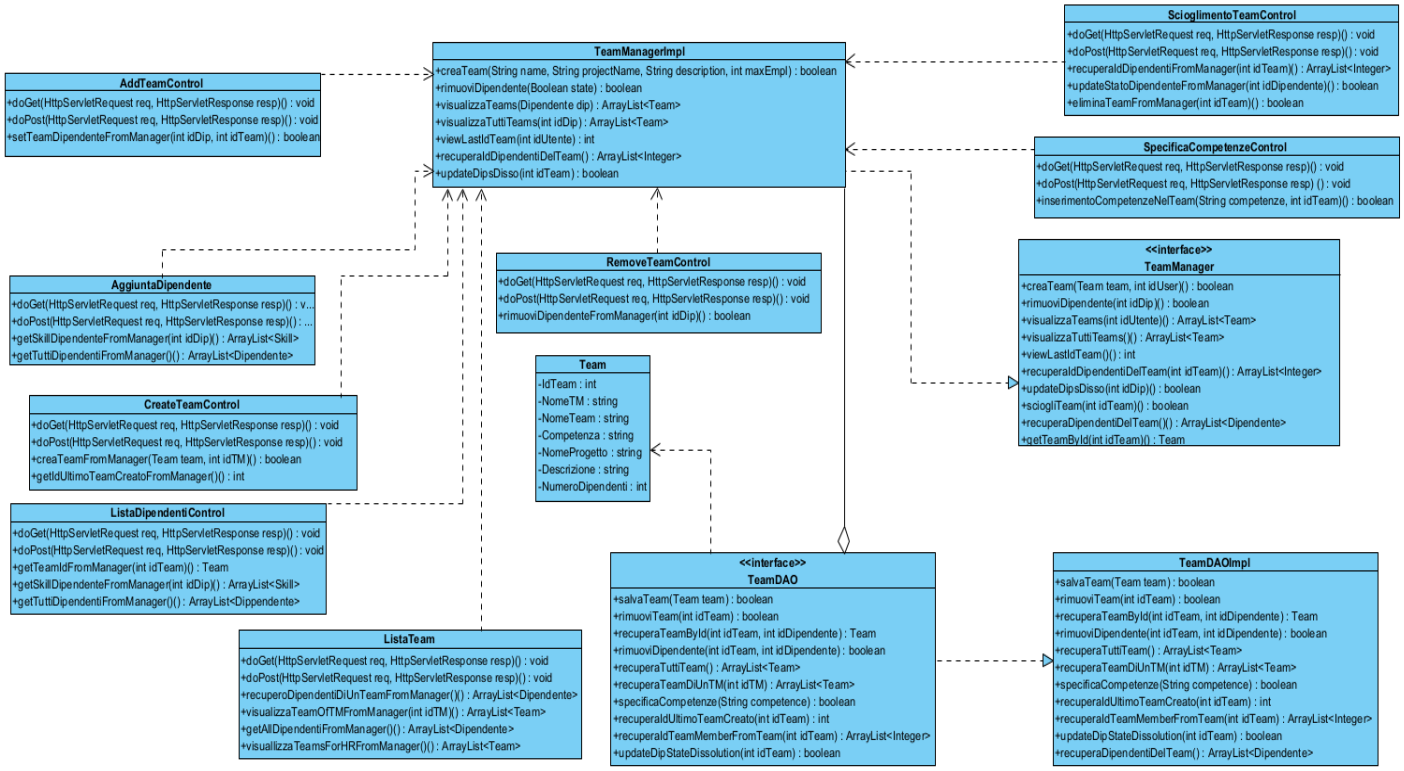
CD_1: *Class Diagram package Autenticazione.* **Autore:** GC, MN, GL, LG, ES, DP



CD_2: *Class Diagram package Reclutamento.* **Autore:** GC, MN, GL, LG, ES, DP



CD_3: *Class Diagram package Formazione.* **Autore:** GC, MN, GL, LG, ES, DP



CD_4: *Class Diagram package Team.* **Autore:** GC, MN, GL, LG, ES, DP

5. Glossario

Sigla/Termine	Definizione
PACKAGE	Raggruppamento di classi ed interfacce correlate.
INTERFACCIA	Insieme di signature delle operazioni offerte da una classe.
Dipendente	Si assume che il dipendente sia il membro che può essere inserito nei Team; quindi, il candidato quando viene assunto
DOMAIN	Identifica il dominio dell'autonomia amministrativa, dell'autorità o del controllo, è costituito da una serie di stringhe separate da punti
DAO	Data Access Object, implementazione dell'omonimo pattern architetturale che si occupa di fornire un accesso astratto ai dati persistenti
CONTROLLER	Classe che si occupa di gestire le richieste effettuate dal client.
MANAGER	Classe che implementa la logica di business che viene utilizzata dal controller o da un altro sottosistema.