Object Design Document

AFR Management

Docente
Gravino Carmine

Tutor
Valeria Pontillo

Studenti
Errico Annunziata 0512110892
Felice Pio Rispoli 0512110475
Domenico Falco 0512112500

Revision History

Data	Versione	Descrizione	Autori
09/01/2023	0.1	Creazione documento e relativa struttura	Tutto il team
10/01/2023	0.1.1	Continuo implementazione documento	Tutto il team
13/02/2023	0.1.2	Implementazione sez 2-3	EA
17/02/2023	0.2	Implementazione sez 4	Tutto il team

Team Members

Nome	Ruolo nel progetto	Email	Acronimo
Errico Annunziata	Team member	e.annunziata24@studenti.unisa.it	EA
Felice Pio Rispoli	Team member	f.rispoli18@studenti.unisa.it	FR
Domenico Falco	Team member	d.falco8@studenti.unisa.it	DF

Sommario

1. Introduzione

Nella prima sezione del documento andremo ad affrontare i trade-offs e le linee guida della fase di implementazione, nomenclatura, documentazione e convezione sui formati.

1.1. Object design goals

Come per la fase di analisi e progettazione del sistema abbiamo individuato diversi compromessi per lo sviluppo, anche durante la fase di ODD scegliamo diversi compromessi che andremo a definire in questo paragrafo:

- **Performance**: Verrà favorito un'archiviazione efficace delle informazioni all'interno del sistema al fine di garantire occupazione di memoria ottimale.
- **Maintenance**: La soluzione modulare del sistema fa sì che sia molto semplice l'aggiunta di nuove componenti o la modifica di quelle esistenti.
- **Dependability**: il sistema sarà affidabile e dunque avere una buona consistenza delle informazioni e robusto ovvero deve continuare a funzionare anche se vengono inseriti dati errati (tramite un sistema di notifica degli errori verso l'utente).

1.2. Linee guida per la documentazione dell'interfaccia

Le linee guida sono un insieme di regole che gli sviluppatori adottano durante la progettazione e sviluppo delle interfacce.

1.2.1. Classi ed Interfacce Java

Linee guida per la creazione di classi ed interfacce:

- I nomi delle classi devono iniziare con la lettera maiuscola;
- Sia le variabili che le classi seguono lo stile camelStyle
- Il nome dei metodi è composto da lettere minuscole

1.2.2. Java Server Pages (JSP)

- Apertura e chiusura "<%" e "%>" vengono poste all'inizio di una riga che non contiene altro.
- Istruzioni che contengono solo una riga possono contenere nella stessa anche i tag di apertura e chiusura.

1.2.3. HTML5

- Seguiamo regole di implementazione di HTML5

1.2.4. JavaScript (JS)

- Seguiamo lo standard ECMA-262

1.2.5. Fogli di stile (CSS)

- Seguiamo regole di implementazione di CSS e validato tramite il servizio offerto da W3

1.2.6. Query (JPQL)

- Le clausole delle Query in SQL devono essere scritte tutte in maiuscole

1.3. Definizioni, acronimi, e abbreviazioni

- **Package**: raggruppamento di classi, interfacce o file correlati;
- **Design pattern**: descrizione o modello logico da applicare per la risoluzione di problemi ricorrenti impiegati per ottenere riuso e flessibilitá;
- **Interfaccia**: insieme di firme delle operazioni offerte dalla classe;
- **lowerCamelCase**: pratica utilizzata per scrivere frasi dove ogni parola inizia con la lettera minuscola senza utilizzo di spazi o punteggiatura;

- **UpperCamelCase**: pratica utilizzata per scrivere frasi dove ogni parola inizia con la lettera maiuscola senza utilizzo di spazi o punteggiatura;
- **Javadoc**: sistema di documentazione offerta da Java, generato sotto forma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.4. Riferimenti

Libro:

- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition

Autori:

- Bernd Bruegge & Allen H. Dutoi

Lista di riferimenti utili alla conoscenza del progetto:

- Statement Of Work;
- Requirements Analysis Document;
- System Design Document;
- Object Design Document;
- Test Plan;
- Matrice di tracciabilitá;
- Manuale di installazione;
- Manuale Utente;

2. Packages

src

files: contiene file di configurazione

db.sql: file contenente la configurazione del database persistente

main: contiene il codice sorgente del sistema software

java: contiene il codice Java del sistema

resource: contiene i file di configurazione dell'ambiente e file di configurazione XML

webapp: contiene file per le applicazioni web (JavaScript,CSS,HTML,JSP)

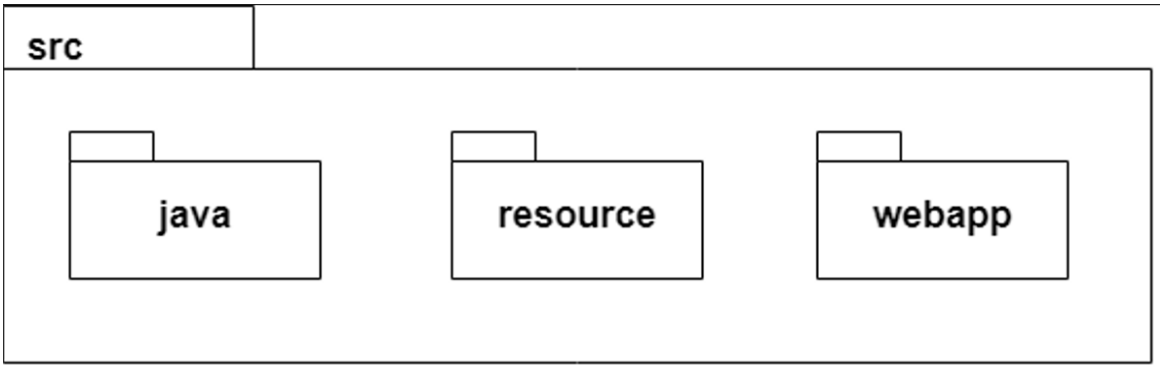
test: contiene il materiale per il testing di sistema

java: contiene le classi utilizzate per effettuare il testing del codice java

resources: contiene file necessari per il testing (configurazione e non)

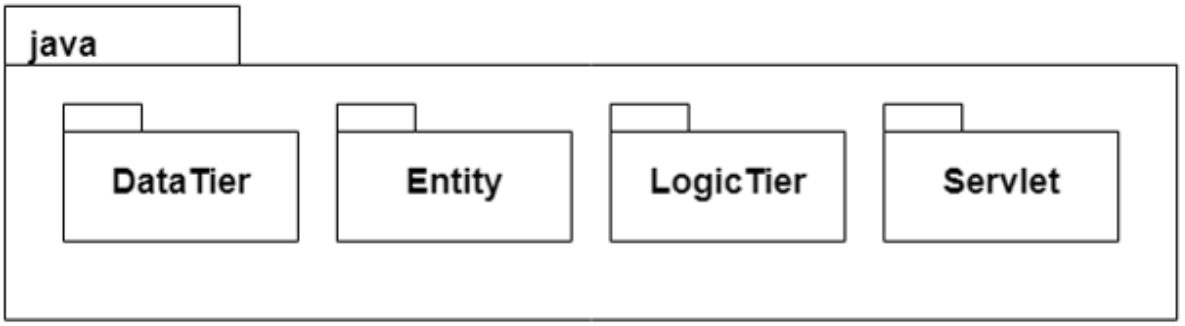
2.1. Package src

Il package contiene tutte le risorse che utilizzeremo nella nostra web application.



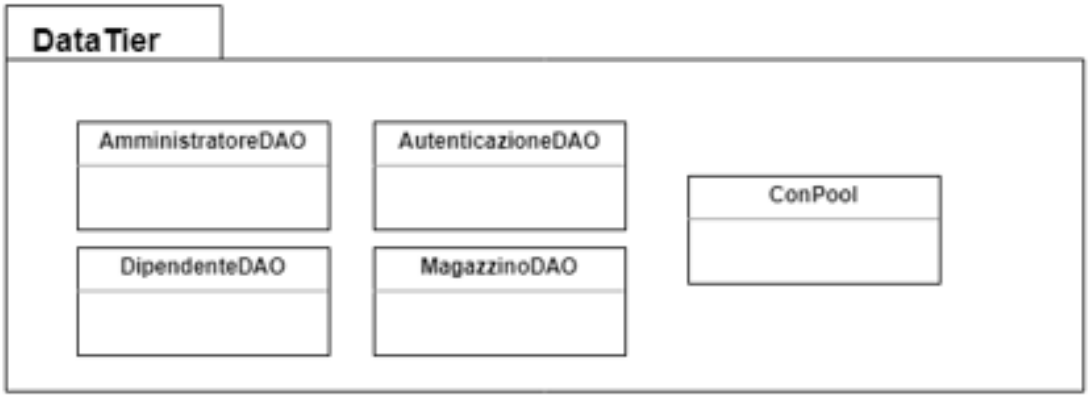
2.1.1. Package java:

Contiene il codice Java



2.1.2. Package DataTier:

Contiene le logiche di business utili alla gestione dei dati persistenti

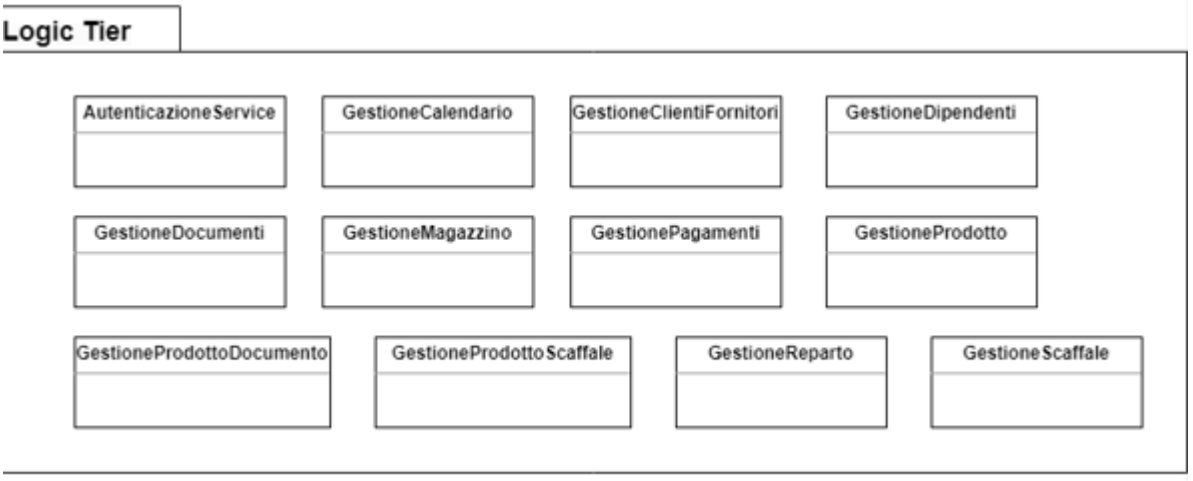


2.1.3. Package Entity:

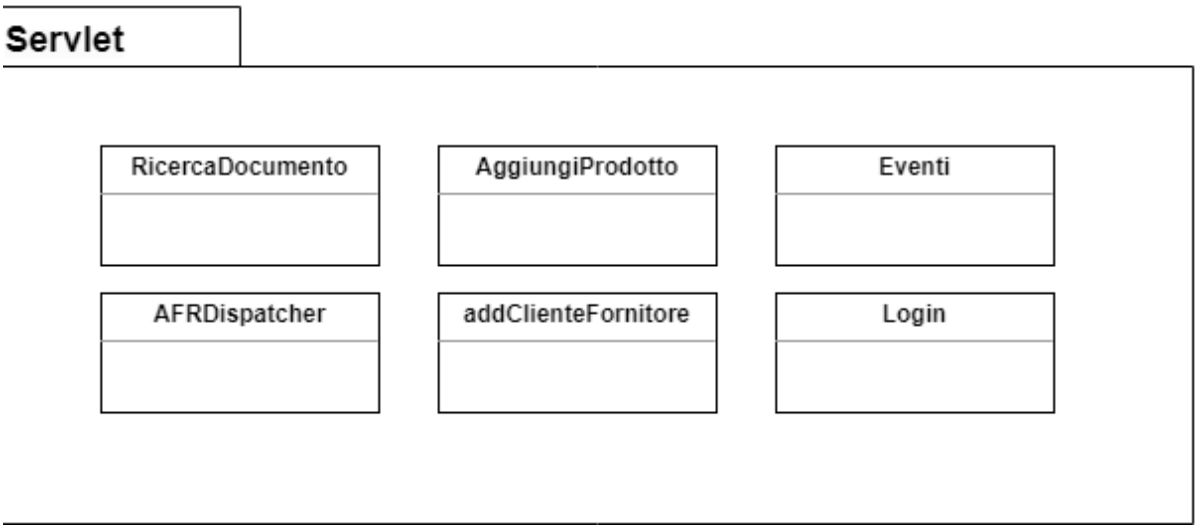
Contiene le classi (entitá) rese persistenti nel database

2.1.4. Package LogicTier:

Contiene le logiche di business che si interfacciano con i DAO per offrire servizi sui dati persistenti.



2.1.5. Package Servlet:



3. Class Interfaces

Di seguito vengono presentate le interfacce delle classi, esclusi gli Entity ed i LogicTier. JavaDoc di AFR Management. JavaDoc

3.1. main.java.DataTier.AutenticazioneDAO:

Nome classe	AutenticazioneDAO
Descrizione	Classe che permette di effettuare l'autenticazione dell'utente e inserimento all'interno del database dell'utente appena registrato
Metodi	-getSingle_instance():AutenticazioneDAO, +login(psw:String,eMail:String):Dipendente, +addDipendente(d:Dipendente).
Invariante di classe	//

Nome metodo	-getSingle_instance():AutenticazioneDAO
Descrizione	Essendo la classe AutenticazioneDAO caratterizzata dall'utilizzo del design pattern Singleton, quest'ultima non presenta alcun costruttore pubblico,infatti non vi è modo di inizializzare alcun oggetto al di fuori di esso, l'unico modo di accedervi è tramite suddetto metodo

Pre-Condizioni	single_instance == null
Post-Condizioni	//

Nome metodo	+login(psw:String,eMail:String)
Descrizione	Convalida l'accesso al sistema
Pre-Condizioni	psw!=null && eMail!=null
Post-Condizioni	//

Nome metodo	+addDipendente(d:Dipendente)
Descrizione	Richiama la classe addDipendente da DipendenteDAO per aggiungere il dipendente in fase di creazione.
Pre-Condizioni	psw!=null && eMail!=null
Post-Condizioni	//

3.2. main.java.DataTier.AmministratoreDAO:

Nome classe	AmministratoreDAO
Descrizione	Il seguente DAO incorpora Pagamento, Documento, ProdottoDocumento ed ClienteFornitore, si occupa di effettuare query e aggiunta, rimozione, aggiornamenti dei seguenti oggetti nel nostro database.
Metodi	-getSingle_instance():AmministratoreDAO, +addPagamento(p:Pagamento), +removePagamento(nTransazione:int), +updatePagamento(p:Pagamento), +addDocumento(d:Documento), +removeDocumento(nDocumento:String), +updateDocumento(d:Documento), +addProdottoDocumento(pd:ProdottoDocumento), +removeProdottoDocumento(codiceArt:String,codiceDoc:String), +updateProdottoDocumento(pd:ProdottoDocumento), +addClienteFornitore(cf:ClienteFornitore), +removeClienteFornitore(cf:String), +updateClienteFornitore(cf:ClientFornitore), +ricercaIdP(id:int):Pagamento, +ricercaIdD(id:String):Documento,

	+ricercaIdPd(codiceArt:String,codiceDoc:String):ProdottoDocumento, +ricercaIdCf(id:String):ClienteFornitore, +ricercaTuttiP():List<Pagamento>, +ricercaTuttiD():List<Documento>, +ricercaTuttiPd():List<ProdottoDocumento>, +ricercaTuttiCf():List<ClienteFornitore>.
Invariante di classe	//

Nome metodo	-getSingle_instance():AmministratoreDAO
Descrizione	Essendo la classe AmministratoreDAO caratterizzata dall'utilizzo del design pattern Singleton, quest'ultima non presenta alcun costruttore pubblico, infatti non vi è modo di inizializzare alcun oggetto al di fuori di esso, l'unico modo di accedervi è tramite suddetto metodo
Pre-Condizioni	single_instance == null
Post-Condizioni	//

Nome metodo	+addPagamento(p:Pagamento)
Descrizione	Il metodo ci permette di aggiungere un Pagamento nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removePagamento(nTransazione:int)
Descrizione	Il metodo ci permette di rimuovere un'istanza Pagamento dal database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updatePagamento(p:Pagamento)
Descrizione	Il metodo ci permette l'aggiornamento dell'istanza all'interno del database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addDocumento(d:Documento)
Descrizione	Il metodo ci permette di aggiungere un documento al database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeDocumento(nDocumento:String)
Descrizione	Il metodo rimuove un documento dal database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateDocumento(d:Documento)
Descrizione	Il metodo aggiorna il documento nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addProdottoDocumento(pd:ProdottoDocumento)
Descrizione	Il metodo si occupa di aggiungere un nuovo prodottoDocumento al database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeProdottoDocumento(codiceArt:String,codiceDoc:String)
Descrizione	Il metodo ci permette di rimuovere un'istanza prodottoDocumento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateProdottoDocumento(pd:ProdottoDocumento)
Descrizione	Il meotodo permette di aggiornare l'istanza prodottodocumento nel database

Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addClienteFornitore(cf:ClienteFornitore)
Descrizione	Il metodo aggiunge un'istanza ClienteFornitore nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeClienteFornitore(cf:String)
Descrizione	Metodo per rimuovere un'istanza ClienteFornitore dal database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateClienteFornitore(cf:ClientFornitore)
Descrizione	Metodo per aggiornare un istanza nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdP(id:int):Pagamento
Descrizione	Metoodo per ricercare un istanza di Pagamento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdD(id:String):Documento
Descrizione	Il metodo permette la ricerca di un istanza Documento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdPd(codiceArt:String,codiceDoc:String):ProdottoDocumento

Descrizione	Il metodo permette la ricerca di un istanza Prodottodocumento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdCf(id:String):ClienteFornitore
Descrizione	Il metodo permette la ricerca di un istanza Clientefornitore
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiP():List<Prodotto>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Pagamento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiD():List<Documento>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Documento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiPd():List<ProdottoDocumento>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Prodottodocumento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiCf():List<ClienteFornitore>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Clientefornitore
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiCfS():List<ClienteFornitore>
Descrizione	Il metodo ci permette di ritornare un array di clienti o fornitori in base alla scelta
Pre-Condizioni	//
Post-Condizioni	//

3.3. main.java.DataTier.MagazzinoDAO:

Nome classe	MagazzinoDAO
Descrizione	Il seguente DAO incorpora Magazzino,Prodotto,Scaffale,ProdottoScaffale, è possibile effettuare query relative all'aggiunta, modifica e cancellazione degli oggetti all'interno del database
Metodi	-getSingle_instance():MagazzinoDAO +addMagazzino(m:Magazzino), +removeMagazzino(codiceMag:int), +updateMagazzino(m:Magazzino), +addProdotto(p:Prodotto), +removeProdotto(codiceArt,String), +updateProdotto(p:Prodotto), +addScaffale(s:Scaffale), +removeScaffale(codiceSc:int), +updateScaffale(s:Scaffale), +addProdottoScaffale(pS:ProdottoScaffale), +removeProdottoScaffale(codiceArt:String,codiceSc:String), +updateProdottoScaffale(pS:ProdottoScaffale), +ricercaIdP(id:String):Prodotto, +ricercaIdS(id:int):Scaffale, +ricercaIdM(id:int):Magazzino, +ricercaIdPs(idP:String,idS:int):ProdottoScaffale, +ricercaTuttiP():List<Prodotto>, +ricercaTuttiS():List<Scaffale>, +ricercaTuttiM():List<Magazzino>, +ricercaTuttiPs():List<ProdottoScaffale>.
Invariante di classe	//

Nome metodo	-getSingle_instance():MagazzinoDAO
Descrizione	Essendo la classe AmministratoreDAO caratterizzata dall'utilizzo del design pattern Singleton, quest'ultima non presenta alcun costruttore pubblico,infatti non vi è modo di inizializzare alcun oggetto al di fuori di

	esso, l'unico modo di accedervi è tramite suddetto metodo
Pre-Condizioni	single_instance == null
Post-Condizioni	//

Nome metodo	+addMagazzino(m:Magazzino)
Descrizione	Il metodo ci permette di aggiungere un magazzino al database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeMagazzino(codiceMag:int)
Descrizione	Il metodo ci permette di rimuovere un Magazzino al database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateMagazzino(m:Magazzino)
Descrizione	Il metodo permette l'aggiornamento di un istanza Magazzino nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addProdotto(p:Prodotto)
Descrizione	Il metodo permette l'aggiunta di un istanza Prodotto nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeProdotto(codiceArt,String)
Descrizione	Il metodo permette la rimozione di un istanza Prodotto nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateProdotto(p:Prodotto)
Descrizione	Il metodo permette l'aggiornamento di un prodotto nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addScaffale(s:Scaffale)
Descrizione	Il metodo ci permette di aggiungere un istanza Scaffale al database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeScaffale(codiceSc:int)
Descrizione	Metoodo che permette la rimozione di un istanza Scaffale dal database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateScaffale(s:Scaffale)
Descrizione	Metodo che permette l'aggiornamento di un istanza Scaffale del database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addProdottoScaffale(pS:ProdottoScaffale)
Descrizione	Il metodo permette di aggiungere un istanza ProdottoScaffale sul database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeProdottoScaffale(codiceArt:String,codiceSc:String)
Descrizione	Il metodo permette di rimuovere un'istanza ProdottoScaffale dal

	database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateProdottoScaffale(pS:ProdottoScaffale)
Descrizione	Metodo che permette l'aggiornamento dell'istanza ProdottoScaffale sul database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdP(id:String):Prodotto
Descrizione	Il metodo ricerca una determinata istanza Prodotto nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdS(id:int):Scaffale
Descrizione	Il metodo ricerca una determinata istanza Scaffale nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdM(id:int):Magazzino
Descrizione	Il metodo ricerca una determinata istanza Magazzino nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdPs(idP:String,idS:int):ProdottoScaffale
Descrizione	Il metodo ricerca una determinata istanza prodottoScaffale nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiP():List<Prodotto>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Prodotto
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiS():List<Scaffale>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Scaffale
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiM():List<Magazzino>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Magazzino
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiPs():List<ProdottoScaffale>
Descrizione	Metodo per ricercare tutti gli elementi della tabella ProdottoScaffale
Pre-Condizioni	//
Post-Condizioni	//

3.4. DipendenteDAO:

Nome classe	DipendenteDAO
Descrizione	Il seguente DAO incorpora Dipendente, Reparto, Calendario ed Evento, si occupa di effettuare query e aggiunta, rimozione, aggiornamenti dei seguenti oggetti nel nostro database.
Metodi	-getSingle_instance():DipendenteDAO +addDipendenti(d:Dipendente), +removeDipendente(matricola:String), +updateDipendente(d:Dipendente), +addReparto(r:Reparto),

	+removeReparto(codiceRep:int), +updateReparto(r:Reparto), +addCalendario(c:Calendario), +removeCalendario(codiceCal:String), +updateCalendario(c:Calendario), +addEvento(e:EventoCalendario), +removeEvento(codiceEv:String), +updateEvento(e:EventoCalendario), +ricercaIdD(id:String):Dipendente, +rimuoviPSW(id:String):Dipendente, +ricercaIdR(id:int):Reparto, +ricercaIdC(id:String):Calendario, +ricercaIdE(id:String):EventoCalendario, +ricercaTuttiD():List<Dipendente>, +ricercaTuttiR():List<Reparto>, +ricercaTuttiC():List<Calendario>, +ricercaTuttiE():List<EventoCalendario>, +ricercaRepC(int id):Calendario.
Invariante di classe	//

Nome metodo	-getSingle_instance():DipendenteDAO
Descrizione	Essendo la classe AmministratoreDAO caratterizzata dall'utilizzo del design pattern Singleton, quest'ultima non presenta alcun costruttore pubblico,infatti non vi è modo di inizializzare alcun oggetto al di fuori di esso, l'unico modo di accedervi è tramite suddetto metodo
Pre-Condizioni	single_instance == null
Post-Condizioni	//

Nome metodo	+addDipendenti(d:Dipendente)
Descrizione	Il metodo ci permette di aggiungere un nuovo dipendente al database.
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeDipendente(matricola:String)
Descrizione	Il metodo ci permette di rimuovere un dipendente da dentro al database
Pre-Condizioni	//

Post-Condizioni	//

Nome metodo	+updateDipendente(d:Dipendente)
Descrizione	Il metodo ci permette di aggiornare un dipendente all'interno del database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addReparto(r:Reparto)
Descrizione	Il metodo aggiunge una nuova istanza di Reparto al database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeReparto(codiceRep:int)
Descrizione	Il metodo rimuove un'istanza di Reparto all'interno del database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateReparto(r:Reparto)
Descrizione	Il metodo si occupa di aggiornare un istanza di Reparto nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addCalendario(c:Calendario)
Descrizione	Il metodo permette l'aggiunta di istanze Calendario al database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeCalendario(codiceCal:String)

Descrizione	Il metodo ci permette la rimozione di un istanza Calendario nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateCalendario(c:Calendario)
Descrizione	Il metodo aggiorna un'istanza di Calendario nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+addEvento(e:EventoCalendario)
Descrizione	Il metodo ci permette di aggiungere istanze nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+removeEvento(codiceEv:String)
Descrizione	Il metodo elimina un'istanza Evento dal database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+updateEvento(e:EventoCalendario)
Descrizione	Il metodo aggiorna un'istanza EventoCalendario nel database
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdD(id:String):Dipendente
Descrizione	Il metodo permette la ricerca di un Dipendente tramite la matricola
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+rimuoviPSW(id:String):Dipendente
Descrizione	Il metodo ci permette di ricercare un dipendente utilizzando la matricola e ritornarlo senza la password quale dato sensibile
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdR(id:int):Reparto
Descrizione	Il metodo ci permette di trovare un Reparto tramite il codiceRep
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdC(id:String):Calendario
Descrizione	Il metodo ricerca un'istanza Calendario in base all'id inserito
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaIdE(id:String):EventoCalendario
Descrizione	Il metodo restituisce un'istanza EventoCalendario
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiD():List<Dipendente>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Dipendente
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiR():List<Reparto>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Reparto

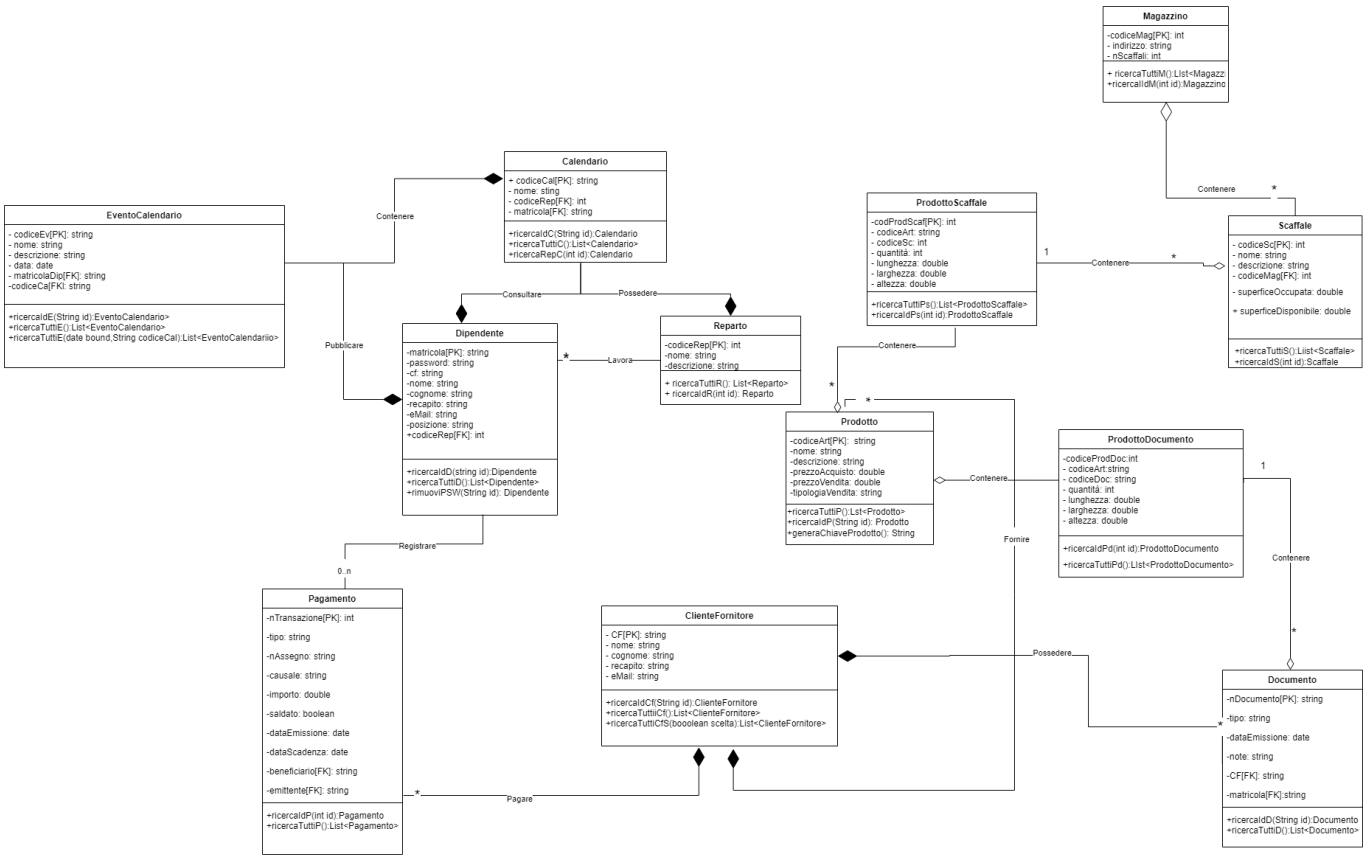
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiC():List<Calendario>
Descrizione	Metodo per ricercare tutti gli elementi della tabella Calendario
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	+ricercaTuttiE():List<EventoCalendario>,
Descrizione	Metodo per ricercare tutti gli elementi della tabella Evento
Pre-Condizioni	//
Post-Condizioni	//

Nome metodo	ricercaRepC(int idReparto):Calendario.
Descrizione	Metodo per ricercare calendario relativo al reparto
Pre-Condizioni	//
Post-Condizioni	//

4. Class Diagram Ristrutturato



NOTA: I metodi non inseriti vengono giá descritti nel paragrafo precedente facendo parte dei sottosistemi DAO.

Nome Classe	EventoCalendario
Descrzione	Classe che rappresenta gli eventi del Calendario
Metodi	+ricercaTuttiE(date bound,String codiceCal):List<EventoCalendario>
Invariante di classe	//

Nome Classe	Calendario
Descrzione	Classe che rappresenta i calendari per ogni reparto
Metodi	+ricercaRepC(int id):Calendario
Invariante di classe	//

Nome Classe	Dipendente

Descrzione	Classe che rappresenta un Dipendente, utente primario del software
Metodi	+rimuoviPSW(String id): Dipendente
Invariante di classe	//

Nome Classe	Reparto
Descrzione	Classe che rappresenta i reparti che contengono i diversi dipendenti
Metodi	//
Invariante di classe	//

Nome Classe	Pagamento
Descrzione	Classe che rappresenta i pagamenti verso l'azienda da parte di Clienti/Fornitori che possono essere inseriti da un Dipendente
Metodi	//
Invariante di classe	//

Nome Classe	ClienteFornitore
Descrzione	Classe che rappresenta la figura del ClienteFornitore che puó essere salvata da un Dipendente
Metodi	//
Invariante di classe	//

Nome Classe	Documento
Descrzione	Classe che rappresenta l'oggetto Documento che puó essere salvato dal Dipendente
Metodi	//
Invariante di classe	//

Nome Classe	ProdottoDocumento
Descrzione	Classe che rappresenta
Metodi	//

Invariante di classe	//

Nome Classe	Prodotto
Descrzione	Classe che rappresenta
Metodi	+generaChiaveProdotto(): String
Invariante di classe	//

Nome Classe	ProdottoScaffale
Descrzione	Classe che rappresenta
Metodi	//
Invariante di classe	//

Nome Classe	Scaffale
Descrzione	Classe che rappresenta gli Scaffali che ospitano i Prodotti all'interno di un Magazzino
Metodi	//
Invariante di classe	//

Nome Classe	Magazzino
Descrzione	Classe che rappresenta un Magazzino, dove vengono posti gli Scaffali
Metodi	//
Invariante di classe	//

Nome Metodo	+ricercaTuttiE(date bound,String codiceCal):List<EventoCalendario>
Descrizione	Metodo che permette di ricercare un evento in base ad una specifica data
Pre-Condizioni	nessuna
Post-Condizioni	viene restituita una lista di EventoCalendario contenete tutti gli eventi in una determinata data

Nome Metodo	+ricercaRepC(int id):Calendario
Descrizione	Metodo che permette di ricercare un calendario di uno speciifico reparto
Pre-Condizioni	nessuna
Post-Condizioni	viene restituito il calendario del reparto desiderato

Nome Metodo	+rimuoviPSW(String id): Dipendente
Descrizione	Metodo che permette che permette di rimuovere la password quando vengono visualizzati i dipendenti
Pre-Condizioni	nessuna
Post-Condizioni	restituzione del dipendente senza la password

Nome Metodo	+generaChiaveProdotto(): String
Descrizione	Metodo che permette di generare la chiave primaria di un nuovo prodotto
Pre-Condizioni	nessuna
Post-Condizioni	viene restituita la stringa che contiene la password generata

5. Design Patterns

In questo paragrafo andremo a definire i design patterns utilizzati nello sviluppo del sistema, per ogni pattern verrá fornito:

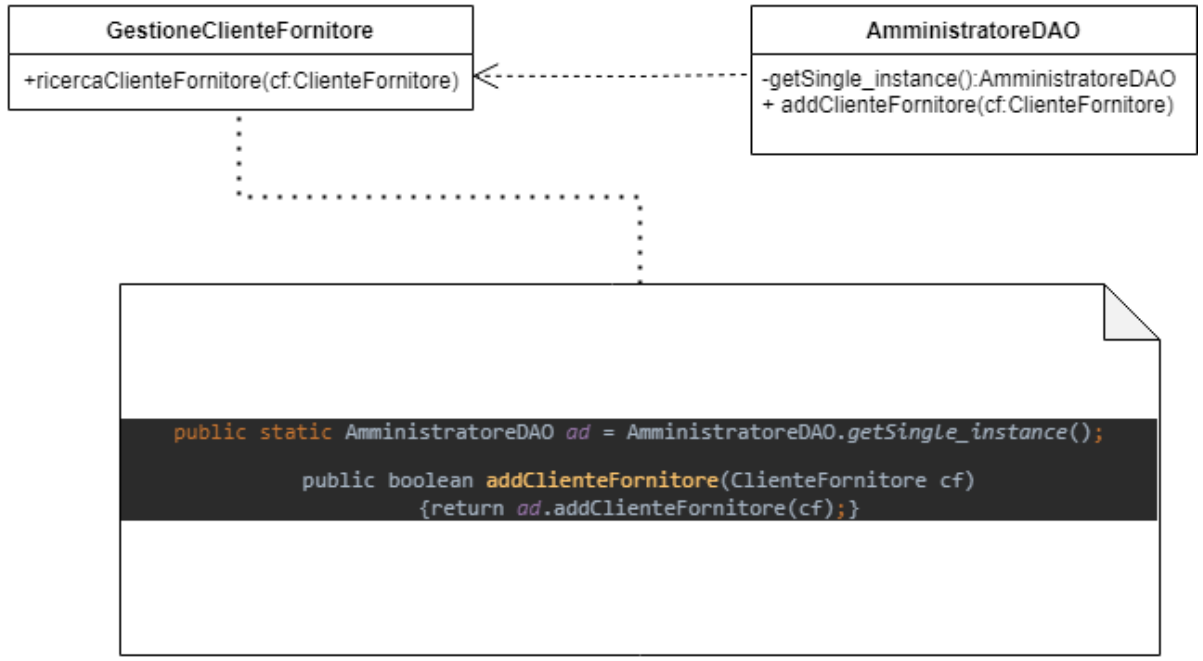
- Introduzione teorica (breve);
- Il problema che viene risolto in AFR Management;
- Come risolviamo il problema nel sistema;
- Un grafico della struttura delle classi che implementano il pattern.

Singleton

Creational Pattern, fornisce un'astrazione del processo di creazione degli oggetti ed aiutano a rendere un sistema indipendente dalle modalità di creazione, composizione e rappresentazione degli oggetti utilizzati. Il singleton ha il compito di garantire che di una determinata classe venga creata una sola istanza.

Nel nostro caso andremo a risolvere un problema riguardante le performance, per quanto riguarda l'utilizzo delle connessioni JPA, infatti creare e chiudere una connessione al database per ogni invocazione di un metodo DAO non è positivo se puntiamo ad un sistema performante.

Il design pattern Singleton risolve questo problema creando una sola connessione JPA che verrá utilizzata da tutte le invocazioni dei metodi DAO.
In questo esempio si può vedere il Singleton DAO all'opera, il cui compito è quello di interfacciarsi con il database e fornire metodi di inserimento, eliminazione e modifica di oggetti al suo interno.
Vista la natura Singleton del DAO, si garantisce un punto singolo di accesso ai dati persistenti da parte delle classi Service in tal modo si riesce a garantire integrità e consistenza dei dati.
Nell'esempio sottostante vediamo come nel nostro software il pattern Singleton, viene usato per gestire l'accesso ai metodi del database.



6. Glossario

- **DAO**: Data Access Object, pattern architetturale che ci occupa di fornire accesso ai dati persistenti.
- **Controller**: Una classe che si occupa della gestione delle richieste effettuate dal client
- **Service**: Classe che inserisce la logica di business, utilizzata dal controller o da altri sottosistemi.
- **JPA**: Java Persistence Api è un API che consente di eseguire query PQL all'interno di un'applicazione Java. Il driver JPA si occupa di tradurre le query e connettere il sistema al database.
- **Singleton**: E' un design pattern creazionale che ci assicura che la classe ha una sola sola istanza che provvede a definire un accesso globale.