Object Design Document (ODD)

MagazON Lab

Riferimento	nc34_magazonlab-ODD
Versione	V8
Data	30/01/2025
Destinatario	Studenti di Ingegneria del Software 2024/25
Presentato da	Battaglia Daniel, Vaiano Francesco, Gigante Ruben
Approvato da	Gravino Carmine

Team Members

Nome	Ruolo	Acronimo	E-mail
Daniel Battaglia	Team Member	DB	d.battaglia8@studenti.unisa.it
Francesco Vaiano	Team Member	FV	f.vaiano3@studenti.unisa.it
Ruben Gigante	Team Member	RG	r.gigante2@studenti.unisa.it

Revision History

Data	Versione	Descrizione	Autore
09/12/2024	V1	Prima stesura dell'introduzione e del punto 2	DB,FV
10/12/2024	V2	Inizio stesura punto 3 e punto 5	DB
13/12/2024	V3	Stesura punti 3.1, 3.2, 3.3, 3.4, 3.5, 3.6 e 3.7	RG,FV
18/12/2024	V4	Class diagram ristrutturato	FV
26/12/2024	V5	Design patterns + revisione	DB
11/01/2025	V6	Revisione documento	RG

24/01/2025	V7	Inizio revisioni finali	RG
30/01/2025	V8	Revisione finale completa	DB, FV RG

Sommario

1. Introduzione 5
1.1. Object Design Goals 5
1.2. Analisi trade-off 5
1.3. Componenti off-the-shelf 6
1.4. Linee guida per la documentazione dell'interfaccia 7
1.5. Definizioni, acronimi e abbreviazioni 8
1.6. Riferimenti 8
2. Packages 9
3. Class interfaces 12
3.1. Package Entity 12
3.2. Package GestioneProdottiDAO 15
3.3. Package GestioneCategorieDAO 18
3.4. Package GestioneLogisticaDAO 20
3.5. Package GestioneUtentiDAO 23
3.6. Package GestioneListeDAO 26
3.7. Package GestioneNotificheDAO 29
3.8. Package AutenticazioneDAO 32
4. Class Diagram Ristrutturato 34
5. Design patterns 35
5.1. Facade 35
5.2. Singleton 36
6. Glossario 38

1. Introduzione

MagazON Lab si propone di semplificare ed ottimizzare attività quotidiane di un supermercato e di conseguenza ridurre il più possibile gli errori umani.

Il documento di **Object Design** ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. Inoltre, verranno definiti i packages, le interfacce delle classi e verrà ristrutturato il class diagram realizzato durante l'analisi dei requisiti.

1.1. Object Design Goals

Modularità

Il sistema deve basarsi su alta coesione e basso accoppiamento tra le sue componenti.

Robustezza

Il sistema dovrà garantire robustezza reagendo correttamente a situazioni impreviste usufruendo di opportuni controlli degli errori e gestione delle eccezioni.

Incapsulamento

Il sistema deve limitare l'accesso diretto alle sue componenti *(Information hiding)*.

1.2. Analisi trade-off

Facilità d'uso vs Complessità

L'introduzione di funzionalità aggiuntive più specifiche rispetto a quelle già progettate, accrescerebbe la complessità del sistema rendendolo più difficile da utilizzare per gli utenti meno esperti. Si è scelto così di dare priorità alla facilità di utilizzo del sistema.

Gestione dei fallimenti vs Tempi di risposta

Avere procedure troppo complesse di gestione dei fallimenti potrebbe rallentare la risoluzione di problemi e influire sui tempi di risposta.

Motivo per cui, si è deciso di effettuare un'accurata gestione dei fallimenti, senza influire negativamente sulle performance del sistema.

Riservatezza vs condivisione dei dati

Per garantire una giusta sicurezza ai diversi utenti che usufruiscono del sistema, si è deciso di effettuare una netta distinzione tra le operazioni concesse ad ognuno di essi. Tutto ciò a discapito della comunicazione e della condivisione dei dati tra gli utenti.

1.3. Componenti off-the-shelf

Per il progetto software che si vuole realizzare, facciamo uso di componenti **off-the-shelf**. Sono componenti software disponibili sul mercato che servono a facilitare il lavoro di molti developers.

Una di queste componenti è **AJAX**, acronimo di *Asynchronous JavaScript and XML*. AJAX è un insieme di tecniche e metodologie di sviluppo software per la realizzazione di applicazioni web interattive. Si basa su uno scambio di dati in background fra web browser e server, consentendo così l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

Come formato per lo scambio dati tra una servlet ed una pagina web in modo dinamico, si è deciso di utilizzare **JSON** (*JavaScript Object Notation*); esso è un formato di scambio dati leggero, facile da leggere e scrivere.

Un'altra componente è **JSTL** (*JavaServer Pages Standard Tag Library*), una libreria standard per **JSP** (*Java Server Pages*) che facilita lo sviluppo di applicazioni web eliminando la necessità di scrivere codice Java direttamente nelle pagine JSP. Grazie a JSTL, le pagine JSP risultano più leggibili e manutenibili, poiché la logica di presentazione è separata dal codice Java. Inoltre, essendo parte dello standard Java EE, è una libreria affidabile e ampiamente supportata.

1.4. Linee guida per la documentazione dell'interfaccia

È richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell'intero progetto e facilitare la comprensione delle funzionalità.

Di seguito alcuni link alle convenzioni usate per definire le linee guida:

- W3Schools – HTML Style Guide.

Per avere una documentazione corretta e completa adottiamo lo strumento **Javadoc**.

Per ogni classe devono essere definiti:

- Autore;
- Descrizione.

Per ogni metodo devono essere definiti:

- Descrizione;
- Parametri in input;
- Tipo di ritorno;
- Eccezioni.

1.5. Definizioni, acronimi e abbreviazioni

- **Package**: raggruppamento di classi, interfacce o file correlati.
- **Design pattern:** template di soluzioni applicate a problemi ricorrenti, utili per avere riutilizzo ed espandibilità del sistema.
- **Interfaccia**: insieme di prototipi di metodi offerti dalla classe.
- **Javadoc**: applicativo utilizzato per generare automaticamente documentazione al fine di renderla facilmente accessibile e leggibile.
- **Camel case**: in italiano "notazione a cammello", consiste nello scrivere parole tralasciando gli spazi, ma con le iniziali maiuscole.

1.6. Riferimenti

- Libro di testo utilizzato durante il corso: *Object-Oriented Software Engineering;*
- **SOW;**
- **RAD;**
- **SDD.**

2. Packages

In questa sezione del documento mostriamo la suddivisione in package del sistema; tale suddivisione riprende l'organizzazione definita da **Maven**.

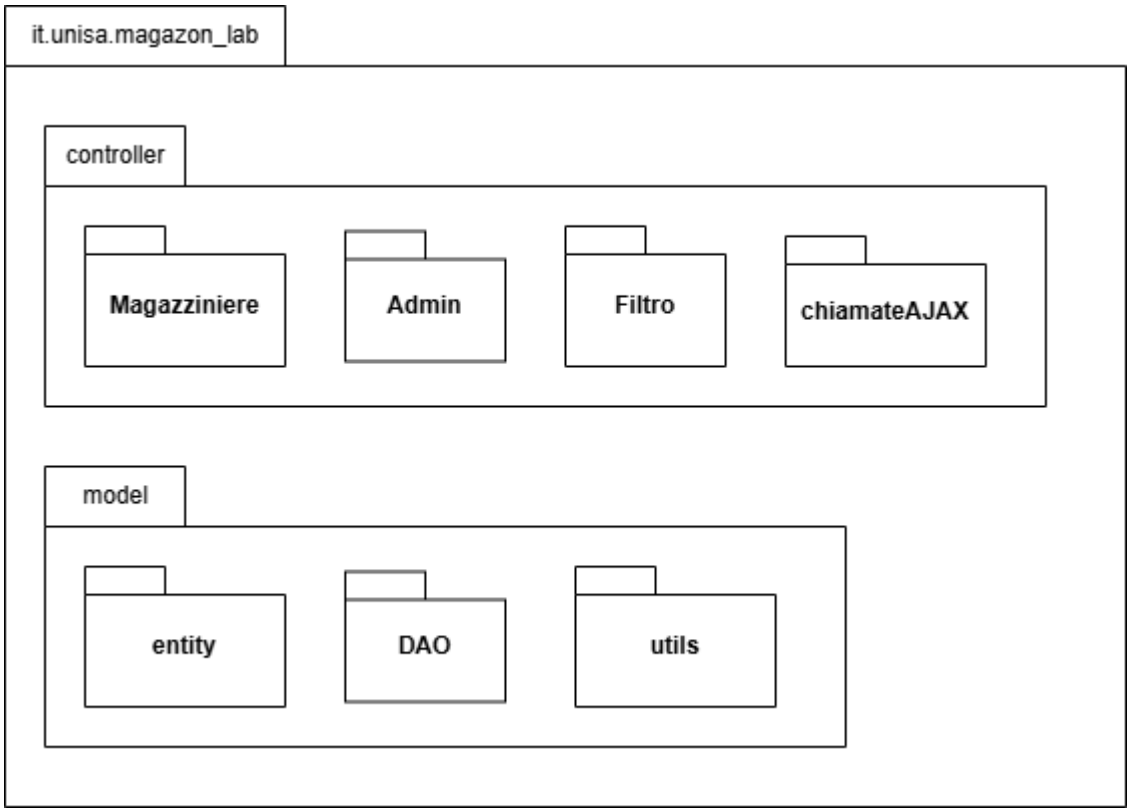
- **.idea**: configurazione di IntellIJ
- **.mvn**: configurazione di Maven
- **src**: codice sorgente
    - **main**
        - **java**
            - it.unisa.magazon_lab
        - **Webapp**
            - **css**
            - **img**
            - **js**
            - **WEB-INF**: file non accessibili direttamente dai client
                - **results**
    - **test**
        - **java**: classi java per il testing
    - **target**: file compilati

Package it.unisa.magazon_lab

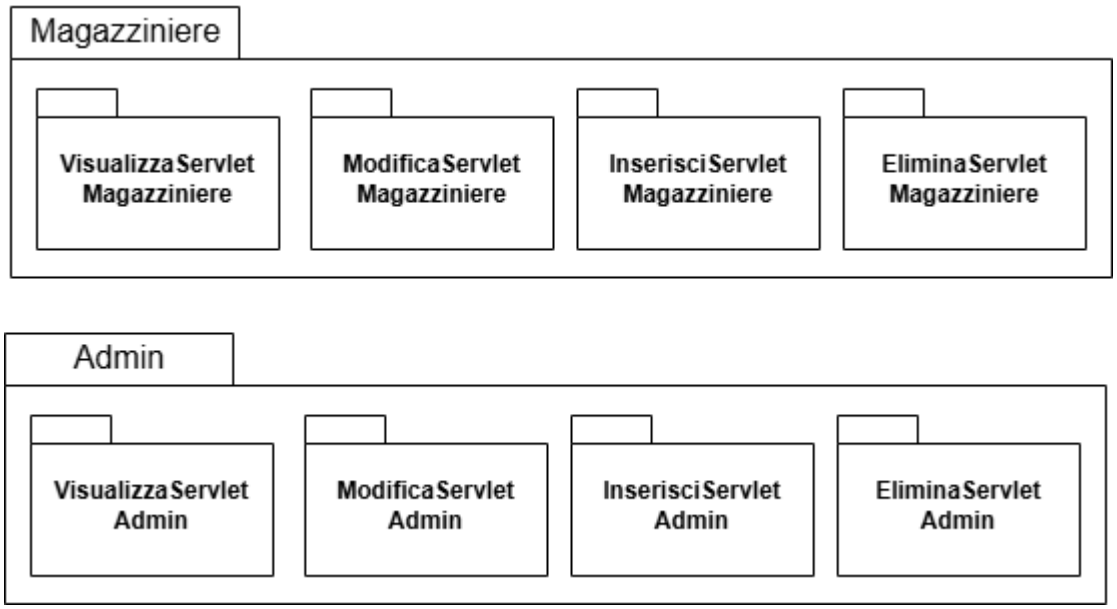
Il seguente package contiene:

- **controller**: contiene i package per svolgere i sottosistemi già individuati nel documento di System Design.
- **model**: contiene il package entity, il package DAO e il package utils.

UML Package Diagram



Magazziniere ed Admin



Questa organizzazione è stata ideata per permettere di differenziare le operazioni tra admin e magazziniere, oltre che facilitare l'applicazione dei filtri di accesso. Le quattro operazioni fondamentali nella nostra web app sono appunto visualizzazione, modifica, inserimento ed eliminazione. Le servlet le gestiranno. Per accedere ad una risorsa occorre obbligatoriamente interpellare **VisualizzaServletMagazziniere**, se l'utente ha stato=magazziniere, e **VisualizzaServletAdmin** se l'utente ha stato=admin. I 6 requisiti funzionali vengono elaborati all'interno delle servlet attraverso le risorse del model, ovvero le classi entity e DAO.

Filtro

Questo package conterrà servlet per garantire, lato magazziniere e lato admin, che certe risorse potranno essere accessibili solo se loggati e con un determinato ruolo.

ChiamateAJAX

Questo package conterrà servlet per rispondere ad eventuali richieste AJAX.

3. Class interfaces

Di seguito sono riportate le interfacce per ciascun package.

3.1. Package Entity

Nome classe	Prodotto
Descrizione	Questa classe contiene i dati per rappresentare i prodotti.
Attributi	-ID : int -IDcategoria : int -nomeCategoria : String -codice : String -stato : String -nome : String -descrizione : String -dataArrivo : Date -noteArrivo : String -partenza : String -dataSpedizione : Date -noteSpedizione : String -destinazione : String -noteGenerali : String
Metodi	Constructor @Getter @Setter

Nome classe	Arrivo
Descrizione	Questa classe contiene i dati per rappresentare gli arrivi.
Attributi	-ID : int -IDprodotto : int -codice : String -note : String
Metodi	Constructor @Getter @Setter
Nome classe	Spedizione

Descrizione	Questa classe contiene i dati per rappresentare le spedizioni.
Attributi	-ID : int -IDprodotto : int -codice : String -note : String
Metodi	Constructor @Getter @Setter

Nome classe	Categoria
Descrizione	Questa classe contiene i dati per rappresentare le categorie.
Attributi	-ID : int -nome : String -descrizione : String -note : String
Metodi	Constructor @Getter @Setter

Nome classe	Utente
Descrizione	Questa classe contiene i dati per rappresentare gli utenti.
Attributi	-ID : int -nome : String -cognome : String -ruolo : String -username : String -stato : String -email : String -telefono : String -dataDiNascita : Date -luogoDiNascita : String
Metodi	Constructor

	@Getter @Setter

Nome classe	Notifica
Descrizione	Questa classe contiene i dati per rappresentare le notifiche.
Attributi	-ID : int -IDutente : int -oggetto : String -messaggio : String -stato : String -dataDiInvio : String
Metodi	Constructor @Getter @Setter

Nome classe	Lista
Descrizione	Questa classe contiene i dati per rappresentare le liste di prodotti.
Attributi	-ID : int -nomeFile : String -note : String -dataInvio : Date
Metodi	Constructor @Getter @Setter

Nome classe	Connessione
Descrizione	Questa classe contiene i dati per connettersi e accedere al database.
Attributi	-connection : Connection -URL: String - USER: String - PASSWORD: String

Metodi	Constructor getConnection( ) : Connection closeConnection ( ) : void

3.2. Package GestioneProdottiDAO

Nome classe	GestioneProdottiDAO
Descrizione	Questa classe fornisce i metodi per gestire le operazioni CRUD (Create, Read, Update, Delete) relative ai prodotti nel database.
Metodi	+ getInstance() : GestioneProdottiDAO instance + visualizzaProdotti() : List<Prodotto> prodotti + visualizzaProdottiPerSpedizioneArrivo() : List<Prodotto> prodotti + cercaProdotto(int ID) : Prodotto prodotto + cercaProdottiFiltrati(String codice, Integer categoriaID, String nome, String stato, String dataArrivoStr, String dataSpedizioneStr) : List<Prodotto> prodotti + eliminaProdotto(int id) : String result + aggiungiProdotto(int idCategoria, String codice, String stato, String nome, String descrizione, String dataArrivoStr, String noteArrivo, String partenza, String dataSpedizioneStr, String noteSpedizione, String destinazione, String noteGenerali) : String result + modificaProdotto(int idProdotto, int idCategoria, String codice, String stato, String nome, String descrizione, String dataArrivoStr, String noteArrivo, String partenza, String dataSpedizioneStr, String noteSpedizione, String destinazione, String noteGenerali) : String result
Invariante di classe	/

Nome metodo	+ getInstance() : GestioneProdottiDAO instance
Descrizione	Questo metodo restituisce un'unica istanza di **GestioneProdottiDAO**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaProdotti() : List<Prodotto> prodotti
Descrizione	Questo metodo restituisce una **lista dei prodotti** presenti nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaProdottiPerSpedizioneArrivo() : List<Prodotto> prodotti
Descrizione	Questo metodo restituisce una **lista dei prodotti presenti** nel sistema che non sono nello stato "**in arrivo**" o "**in spedizione**".
Pre-condizione	/
Post-condizione	/

Nome metodo	+ cercaProdotto(int ID) : Prodotto prodotto
Descrizione	Questo metodo **restituisce un prodotto** presente nel sistema identificato da un **ID**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ cercaProdottiFiltrati(String codice, Integer

	categoriaID, String nome, String stato, String dataArrivoStr, String dataSpedizioneStr) : List<Prodotto> prodotti
Descrizione	Questo metodo restituisce una **lista di prodotti filtrati** attraverso alcuni criteri.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ eliminaProdotto(int id) : String result
Descrizione	Questo metodo permette di **rimuovere un prodotto** presente nel sistema identificato da un ID e viene ritornato l'esito dell'operazione.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ aggiungiProdotto(int idCategoria, String codice, String stato, String nome, String descrizione, String dataArrivoStr, String noteArrivo, String partenza, String dataSpedizioneStr, String noteSpedizione, String destinazione, String noteGenerali) : String result
Descrizione	Questo metodo permette di **aggiungere un prodotto** al sistema restituendo l'esito dell'operazione.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ modificaProdotto(int idProdotto, int idCategoria, String codice, String stato, String nome, String descrizione, String dataArrivoStr, String noteArrivo, String partenza, String dataSpedizioneStr, String noteSpedizione, String destinazione, String noteGenerali) : String result
Descrizione	Questo metodo permette di **modificare un prodotto** presente nel sistema restituendo l'esito dell'operazione.
Pre-condizione	/
Post-condizione	/

3.3. Package GestioneCategorieDAO

Nome classe	GestioneCategorieDAO
Descrizione	Questa classe fornisce i metodi per gestire le operazioni CRUD (Create, Read, Update, Delete) relative alle categorie nel database.
Metodi	+ getInstance() : GestioneCategorieDAO instance + visualizzaCategorie() : List<Categoria> categorie + aggiungiCategoria(String nome, String descrizione, String note) : String result + modificaCategoria(int IDprodotto, String nome, String descrizione, String note) : String result + cercaCategoria(int ID) : Categoria categoria + eliminaCategoria(int ID) : String result
Invariante di classe	/

Nome metodo	+ getInstance() : GestioneCategorieDAO instance
Descrizione	Questo metodo restituisce un'unica istanza di **GestioneCategorieDAO**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaCategorie() : List<Categoria> categorie
Descrizione	Questo metodo restituisce **una lista delle categorie** presenti nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ aggiungiCategoria(String nome, String descrizione, String note) : String result
Descrizione	Questo metodo permette di **aggiungere una categoria** nel sistema restituendo l'esito dell'operazione.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ modificaCategoria(int IDprodotto, String nome, String descrizione, String note) : String result
Descrizione	Questo metodo permette di **modificare una categoria** presente nel sistema restituendo l'esito dell'operazione.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ cercaCategoria(int ID) : Categoria categoria

Descrizione	Questo metodo **restituisce una categoria** presente nel sistema identificata da ID.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ eliminaCategoria(int ID) : String result
Descrizione	Questo metodo **elimina una lista** presente nel sistema restituendo l'esito dell'operazione.
Pre-condizione	/
Post-condizione	/

3.4. Package GestioneLogisticaDAO

Nome classe	GestioneLogisticaDAO
Descrizione	Questa classe fornisce i metodi per gestire le operazioni CRUD (Create, Read, Update, Delete) relative alla logistica nel database.
Metodi	+ getInstance() : GestioneLogisticaDAO instance + visualizzaSpedizioni() : List<Spedizione> spedizioni + visualizzaArrivi() : List<Arrivo> arrivi + eliminaSpedizione(int IDspedizione, int IDprodotto) : void + eliminaArrivo(int IDarrivo, int IDprodotto) : void + visualizzaSpedizione(int IDspedizione) : Spedizione spedizione + visualizzaArrivo(int IDspedizione) : Arrivo arrivo + modificaNoteSpedizione(int IDspedizione, String nuovaNota) : void

	+ modificaNoteArrivo(int IDarrivo, String nuovaNota) : void + inserisciArrivo(int IDprodotto, String noteArrivo) : String result + inserisciSpedizione(int IDprodotto, String noteSpedizione) : String result
Invariante di classe	/

Nome metodo	+ getInstance() : GestioneLogisticaDAO instance
Descrizione	Questo metodo restituisce un'unica istanza di **GestioneLogisticaDAO**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaSpedizioni() : List<Spedizione> spedizioni
Descrizione	Questo metodo **restituisce una lista di spedizioni** presenti nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaArrivi() : List<Arrivo> arrivi
Descrizione	Questo metodo **restituisce una lista di arrivi** presenti nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ eliminaSpedizione(int IDspedizione, int IDprodotto) : void

Descrizione	Questo metodo **elimina una spedizione** presente nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ eliminaArrivo(int IDarrivo, int IDprodotto) : void
Descrizione	Questo metodo **elimina un arrivo** presente nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaSpedizione(int IDspedizione) : Spedizione spedizione
Descrizione	Questo metodo **restituisce una spedizione** presente nel sistema identificata da un ID.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaArrivo(int IDspedizione) : Arrivo arrivo
Descrizione	Questo metodo **restituisce un arrivo** presente nel sistema identificato da un ID.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ modificaNoteSpedizione(int IDspedizione, String nuovaNota) : void
Descrizione	Questo metodo permette di **modificare le note di spedizione** identificato da un ID.
Pre-condizione	/

Post-condizione	/

Nome metodo	+ modificaNoteArrivo(int IDarrivo, String nuovaNota) : void
Descrizione	Questo metodo permette di **modificare le note di arrivo** identificato da un ID.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ inserisciArrivo(int IDprodotto, String noteArrivo) : String result
Descrizione	Questo metodo permette di **inserire un arrivo** nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ inserisciSpedizione(int IDprodotto, String noteSpedizione) : String result
Descrizione	Questo metodo permette di **inserire una spedizione** nel sistema.
Pre-condizione	/
Post-condizione	/

3.5. Package GestioneUtentiDAO

Nome classe	GestioneUtentiDAO
Descrizione	Questa classe fornisce i metodi per gestire le operazioni CRUD (Create, Read, Update, Delete) relative agli utenti nel database.
Metodi	+ getInstance() : GestioneUtentiDAO instance

	+ visualizzaUtenti() : List<Utente> utenti + aggiornaStatoUtente(int userID, String stato) : void + aggiungiUtente(String nome, String cognome, String ruolo, String username, String password, String email, String telefono, String dataNascitaStr, String luogoNascita) : String result + eliminaUtente(int id) : String result + cercaUtente(int id) : Utente utente + cercaIDUtente(String user) : int ID + modificaUtente(int id, String nome, String cognome, String ruolo, String username, String password, String email, String telefono, String dataNascitaStr, String luogoNascita) : String result + setStato(int id, int statoInt) : void
Invariante di classe	/

Nome metodo	+ getInstance() : GestioneUtentiDAO instance
Descrizione	Questo metodo restituisce un'unica istanza di **GestioneUtentiDAO**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaUtenti() : List<Utente> utenti
Descrizione	Questo metodo **restituisce una lista degli utenti** presenti nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ aggiornaStatoUtente(int userID, String stato) : void
Descrizione	Questo metodo permette di **modificare lo stato** dell'utente identificato dal suo **ID**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ aggiungiUtente(String nome, String cognome, String ruolo, String username, String password, String email, String telefono, String dataNascitaStr, String luogoNascita) : String result
Descrizione	Questo metodo permette la **registrazione dell'utente** all'interno del sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ eliminaUtente(int id) : String result
Descrizione	Questo metodo permette di **eliminare un utente** dal sistema identificato da ID.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ cercaUtente(int id) : Utente utente
Descrizione	Questo metodo permette di **cercare l'utente** restituendo l'oggetto **Utente** con tutte le sue informazioni.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ modificaUtente(int id, String nome, String cognome, String ruolo, String username, String password, String email, String telefono, String dataNascitaStr, String luogoNascita) : String result
Descrizione	Questo metodo permette di **modificare le informazioni** dell'utente.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ setStato(int id, int statoInt) : void
Descrizione	Questo metodo permette di **modificare lo stato** dell'utente identificato dal suo ID.
Pre-condizione	/
Post-condizione	/

3.6. Package GestioneListeDAO

Nome classe	GestioneListeDAO
Descrizione	Questa classe fornisce i metodi per gestire le operazioni CRUD (Create, Read, Update, Delete) relative alle liste nel database.
Metodi	+ getInstance() : GestioneListeDAO instance + visualizzaListe() : List<Lista> lista + getListaFileName(int id) : String nomeFile + inserisciLista(String nomeFile, String note) : String result + inserisciLista(String nomeFile) : String result + eliminaLista(int id) : void

	+ aggiornaLista(int id, String note) : boolean result + cercaLista(int id) : Lista lista
Invariante di classe	/

Nome metodo	+ getInstance() : GestioneListeDAO instance
Descrizione	Questo metodo restituisce un'unica istanza di **GestioneListeDAO**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaListe() : List<Lista> lista
Descrizione	Questo metodo restituisce **tutte le liste** presenti nel sistema.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ getListaFileName(int id) : String nomeFile
Descrizione	Questo metodo restituisce il **fileName** di una determinata lista identificata da **ID**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ inserisciLista(String nomeFile, String note) : String result
Descrizione	Questo metodo permette di **inserire** nel sistema una **nuova lista**.
Pre-condizione	/

Post-condizione	/

Nome metodo	+ inserisciLista(String nomeFile) : String result
Descrizione	Questo metodo permette di **inserire** nel sistema una **nuova lista**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ eliminaLista(int id) : void
Descrizione	Questo metodo permette di **cancellare una lista** presente nel sistema attraverso il suo **ID**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ aggiornaLista(int id, String note) : boolean result
Descrizione	Questo metodo permette di **aggiornare le note di una lista** identificata attraverso il suo **ID**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ cercaLista(int id) : Lista lista
Descrizione	Questo metodo **restituisce una lista** presente nel sistema e identificata attraverso il suo **ID**.
Pre-condizione	/
Post-condizione	/

3.7. Package GestioneNotificheDAO

Nome classe	GestioneNotificheDAO
Descrizione	Questa classe fornisce i metodi per gestire le operazioni CRUD (Create, Read, Update, Delete) relative alle notifiche nel database.
Metodi	+ getInstance() : GestioneNotificheDAO instance + visualizzaNotifiche(int userID) : List<Notifica> notifiche + controlloNotifiche(int userID) : int notificationCount + modificaStatoNotifica(int notificaID, int utenteID, String nuovoStato) : String result + inviaNotifica(int ID, String oggetto, String messaggio) : String result + isValidNotify(String oggetto, String messaggio) : boolean result + isValidOggetto(String oggetto) : boolean result + isValidMessaggio(String messaggio) : boolean result
Invariante di classe	/

Nome metodo	+ getInstance() : GestioneNotificheDAO instance
Descrizione	Questo metodo restituisce un'unica istanza di **GestioneNotificheDAO**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ visualizzaNotifiche(int userID) : List<Notifica> notifiche
Descrizione	Questo metodo restituisce le **notifiche** relative ad un utente specificato tramite **userID**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ controlloNotifiche(int userID) : int notificationCount
Descrizione	Questo metodo restituisce un **intero** che indica le **notifiche non lette** da un utente identificato tramite **userID**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ modificaStatoNotifica(int notificaID, int utenteID, String nuovoStato) : String result
Descrizione	Questo metodo permette di **modificare lo stato di una notifica** (Letto, Non Letto), restituendo **l'esito dell'operazione.**
Pre-condizione	/
Post-condizione	/

Nome metodo	+ inviaNotifica(int ID, String oggetto, String messaggio) : String result
Descrizione	Questo metodo permette di **inviare una notifica** ad un determinato utente identificato tramite ID, restituendo **l'esito dell'operazione.**
Pre-condizione	/
Post-condizione	/

Nome metodo	+ isValidNotify(String oggetto, String messaggio) : boolean result
Descrizione	Questo metodo verifica la validità della notifica da inviare (oggetto e messaggio). Se entrambe non sono null restituisce **true**, altrimenti **false**
Pre-condizione	/
Post-condizione	/

Nome metodo	+ isValidOggetto(String oggetto) : boolean result
Descrizione	Questo metodo verifica la validità dell'oggetto della notifica. Se non è null restituisce **true**, altrimenti **false**
Pre-condizione	/
Post-condizione	/

Nome metodo	+ isValidMessaggio(String messaggio) : boolean result
Descrizione	Questo metodo verifica la validità del messaggio della notifica. Se non è null restituisce **true**, altrimenti **false**
Pre-condizione	/
Post-condizione	/

3.8. Package AutenticazioneDAO

Nome classe	AutenticazioneDAO
Descrizione	Questa classe fornisce i metodi per gestire le operazioni di autenticazione di un utente.
Metodi	+ getInstance() : AutenticazioneDAO instance + loginUtente(String user, String password, String ruolo) : Utente utente + isValidCredentials(String username, String password) : boolean result + isValidUsername(String username) : boolean result + isValidPassword(String password) : boolean result
Invariante di classe	/

Nome metodo	+ getInstance() : AutenticazioneDAO instance
Descrizione	Questo metodo restituisce un'unica istanza di **AutenticazioneDAO**.
Pre-condizione	/
Post-condizione	/

Nome metodo	+ loginUtente(String user, String password, String ruolo) : Utente utente
Descrizione	Questo metodo restituisce un oggetto **Utente** se le credenziali corrispondono al record presente nel database.
Pre-condizione	/
Post-condizione	/

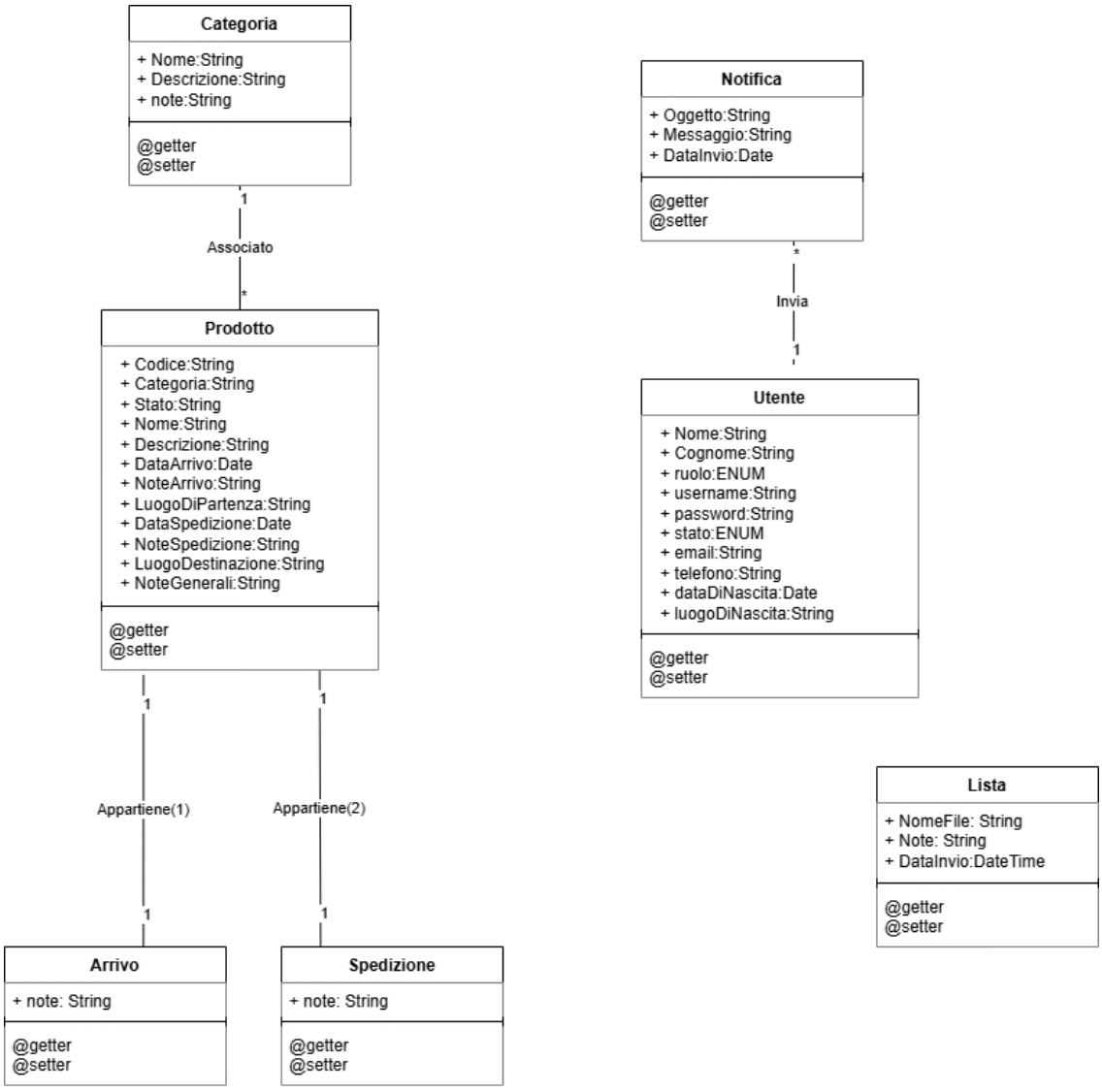
Nome metodo	+ isValidCredentials(String username, String password) : boolean result
Descrizione	Questo metodo verifica le credenziali fornite (nome utente e password). Se entrambe non sono null restituisce **true**, altrimenti **false**
Pre-condizione	/
Post-condizione	/

Nome metodo	+ isValidUsername(String username) : boolean result
Descrizione	Questo metodo verifica il nome fornito. Se non è null restituisce **true**, altrimenti **false**
Pre-condizione	/
Post-condizione	/

Nome metodo	+ isValidPassword(String password) : boolean result
Descrizione	Questo metodo verifica il cognome fornito. Se non è null restituisce **true**, altrimenti **false**
Pre-condizione	/
Post-condizione	/

4. Class Diagram Ristrutturato

Prima di passare all'implementazione, il **class diagram** ottenuto nella fase di analisi è stato ristrutturato per allinearsi con le decisioni prese durante la fase di progettazione.
La gerarchia precedentemente definita, che coinvolgeva la superclasse "Utente registrato" e le sue sottoclassi "Amministratore" e "Magazziniere", è stata rimossa. Al suo posto c'è la classe "Utente", alla quale è stato aggiunto l'attributo "Ruolo" per definire i diversi ruoli che i dipendenti possono assumere nel sistema. Inoltre, ci sono state revisioni e aggiunti nuovi attributi.



5. Design patterns

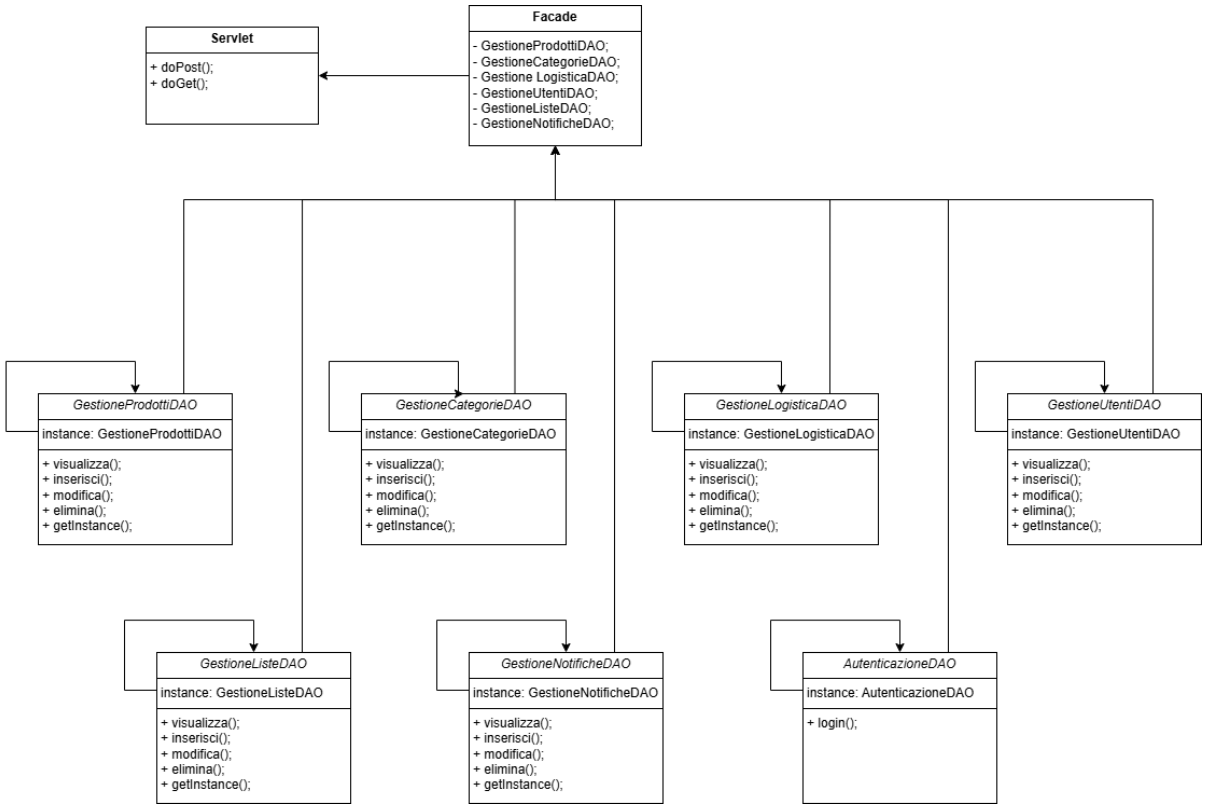
La seguente sezione descrive i design pattern utilizzati nel software. Per ogni pattern utilizzato, forniremo:

- Spiegazione a livello teorico del pattern;
- Il problema che risolve all'interno del sistema;
- Un diagramma che espone l'implementazione del pattern.

5.1. Facade

Facade è un design pattern strutturale, che permette di avere un'interfaccia unica e semplificata per accedere ad un insieme di operazioni e oggetti che compongono un sottosistema, garantisce un basso accoppiamento e rende la piattaforma più aggiornabile e manutenibile.

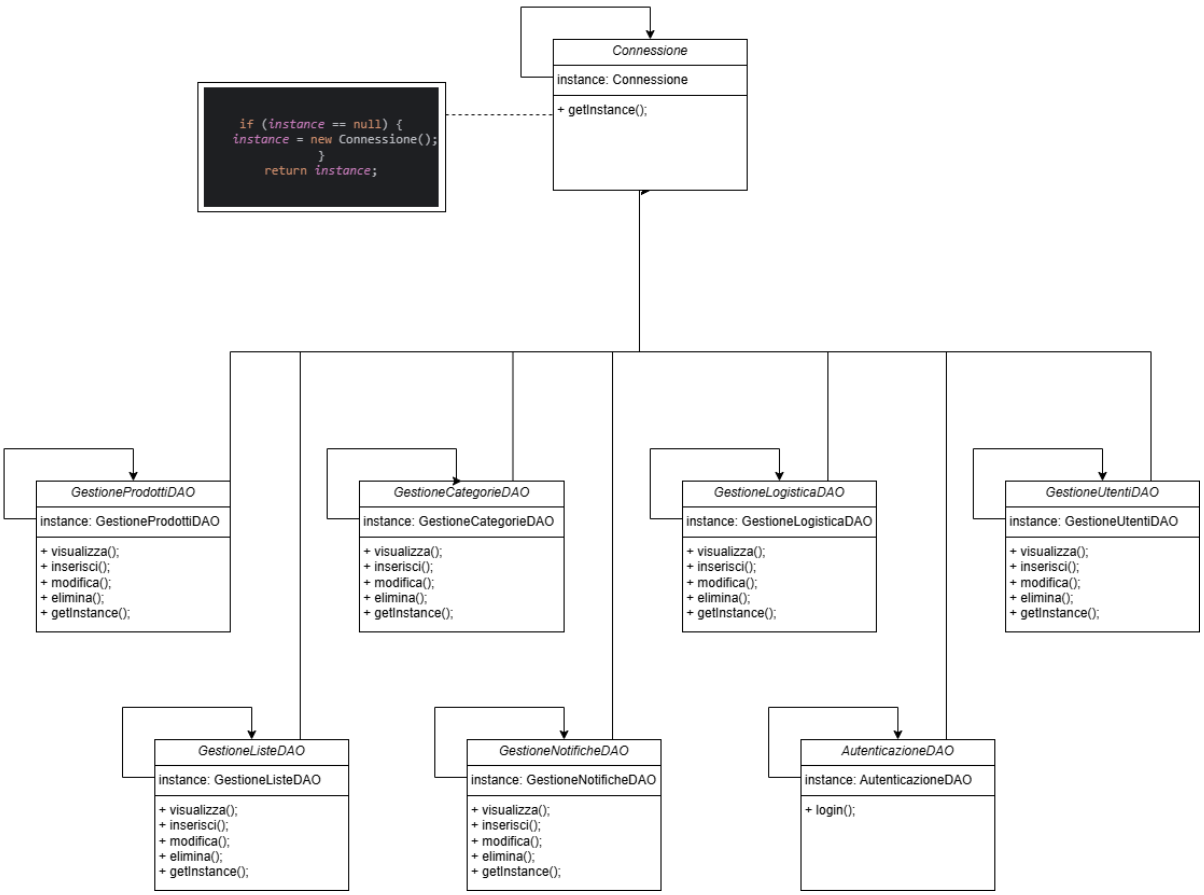
Utilizziamo una sola classe facade che fa da tramite con tutte le classi DAO, utili per le interrogazioni al database, e le servlet, essenziali per la gestione dei dati ricevuti in input e per l'invio di dati in output alla view. Quindi mettiamo a disposizione delle servlet la logica dei nostri sottosistemi, evitando che esse vadano ad interfacciarsi in modo diretto con le classi DAO.



5.2. Singleton

Singleton è un design pattern creazionale, che si occupa dell'istanziazione degli oggetti, con lo scopo di garantire che ogni classe abbia una sola istanza, offrendo un punto di accesso globale ad essa.

Utilizziamo questo design pattern per gestire la connessione al database e le classi DAO, così da garantire che ogni classe che interagisce col database abbia una sola istanza.


Connessione
instance: Connessione
+ getInstance();
if (instance == null) {
instance = new Connessione();
}
return instance;
GestioneProdottiDAO
instance: GestioneProdottiDAO
+ visualizza();
+ inserisci();
+ modifica();
+ elimina();
+ getInstance();
GestioneCategorieDAO
instance: GestioneCategorieDAO
+ visualizza();
+ inserisci();
+ modifica();
+ elimina();
+ getInstance();
GestioneLogisticaDAO
instance: GestioneLogisticaDAO
+ visualizza();
+ inserisci();
+ modifica();
+ elimina();
+ getInstance();
GestioneUtentiDAO
instance: GestioneUtentiDAO
+ visualizza();
+ inserisci();
+ modifica();
+ elimina();
+ getInstance();
GestioneListeDAO
instance: GestioneListeDAO
+ visualizza();
+ inserisci();
+ modifica();
+ elimina();
+ getInstance();
GestioneNotificheDAO
instance: GestioneNotificheDAO
+ visualizza();
+ inserisci();
+ modifica();
+ elimina();
+ getInstance();
AutenticazioneDAO
instance: AutenticazioneDAO
+ login();

6. Glossario

Sigla/termine	Descrizione
Piattaforma	Base software o hardware sulla quale è sviluppata o in esecuzione l'applicazione.
Design pattern	Template di soluzioni a problemi ricorrenti, impiegati per ottenere riuso e flessibilità
Package	Raggruppamento di classi, interfacce o file correlati
Componenti off-the-shelf	Si riferisce a quei componenti o moduli software che sono disponibili già pronti e possono essere acquistati o utilizzati senza richiedere una personalizzazione specifica