Object Design Document

DryBlue

Riferimento	ODD_C8
Versione	1.0
Data	19/12/2021
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Piero Agosto, Michele Bisaccia, Sabrina Ceccarelli, Piero Dello Buono, Miriam Ferrara, Stefano Mungiello, Antonio Sellitto
Approvato da	Alessia Natale, Saverio De Stefano

Revision History

Data	Versione	Descrizione	Autori
16/12/2021	1.0	Object design trade-offs	SM,AS
16/12/2021	1.0	Componenti off-the-shelf	SC
16/12/2021	1.0	Linee guida documentazione	PA,SM
16/12/2021	1.0	Design pattern	MF,AS
16/12/2021	1.0	Packages	MB,PDB
19/12/2021	1.0	Interfacce delle classi	PA,SC,AS
19/12/2021	1.0	Diagramma delle classi	PDB,SM

Tabella dei contenuti

1. Introduzione

1.1. Object design trade-offs

- **Spazio di memoria vs tempi di risposta**
  - Il team di analisi e sviluppo conviene sul fatto che, volendo digitalizzare i servizi di un'azienda che, per ora, lavora solo su cartaceo, i tempi di risposta siano un punto fondamentale su cui premere. A questo scopo, si sceglie di dar più importanza a questo aspetto, contenendo il quantitativo di memoria utilizzata.

- **Comprensibilità vs tempi di consegna**
  - Il team di analisi e sviluppo si impegna nello scrivere un codice che sia il più leggibile e comprensibile possibile (tramite l'utilizzo di commenti, formattazioni e tutte le convenzioni che verranno elencate nella sezione delle linee guida). Questo, ovviamente, potrebbe portare a dei ritardi nei tempi di consegna. Fermo restando che il team si impegna a consegnare entro i tempi stabiliti, si impegna altresì a rispettare tutte le convenzioni elencate in modo tale da rendere facile la lettura sia al team stesso sia ad altri sviluppatori che lavoreranno col codice in futuro (in caso di manutenzione o aggiornamento).

- **Robustezza vs tempi di risposta**
  - Il sistema che si va a creare dovrà prevedere ragionevoli soluzioni a situazioni impreviste di errori o malfunzionamenti. Questo, ovviamente, potrebbe portare a tempi di risposta maggiori.

1.2. Componenti off-the-shelf

Le componenti off-the-shelf permettono di creare il nostro sistema hardware/ software. Nel nostro sistema utilizzeremo diverse tecnologie quali:

- Java: Linguaggio di programmazione ObjectOriented;
- Spring: nuovo framework OpenSource per lo sviluppo di applicazioni su piattaforma Java. Utilizza le annotazioni che permettono di modificare il comportamento dello Spring Framework;
- H2: per il database in fase di produzione
- Intellij IDEA: ambiente di sviluppo integrato (IDE) per il linguaggio di programmazione Java

1.3. Linee guida documentazione

Le linee guida di questo documento racchiudono le principali regole e convenzioni che il team di sviluppo seguirà durante tutto il processo di sviluppo delle interfacce.

1. **Nomi dei file**
   - Le classi devono avere nomi al singolare.
     - Se sono classi di test di unità devono avere il suffisso "Test"
     - Se sono classi di test di integrazione devono avere il suffisso "IT"
   - I nomi dei file di configurazione XML devono rispettare i requisiti del framework o tool che li usa.
   - I nomi dei file generati da compilatori o altri tool non devono essere modificati.

   In tutti gli altri casi, i nomi dei file devono rappresentare le funzionalità contenute al loro interno e dovranno essere composte solo da lettere minuscole e cifre.

2. Formattazione

Per la formattazione del file Java si seguiranno le convenzioni **Sun Java Coding Conventions** (reperibili al seguente link: https://checkstyle.sourceforge.io/sun_style.html), mentre per i file XML, YAML, HTML, CSS e JS si userà il formatter di IntelliJ di default.

3. Dichiarazioni

Per ogni dichiarazione di variabile locale:

- Possono essere definite più di una variabile
- Deve seguire l'inizializzazione nella stessa linea o in quella seguente

Ogni dichiarazione di variabile d'istanza deve definire solo una variabile, che dovrà essere privata.

4. Nomenclatura

Per la nomenclatura, verranno utilizzati caratteri:

- In *lowerCamelCase* per package, metodi, variabili e parametri
- In *UpperCamelCase* per le classi
- In CONSTANT_CASE per le costanti

I nomi dei metodi dovranno contenere solo verbi e nomi degli attributi della classe.

5. Convenzioni

- Le condizioni d'errore devono lanciare delle eccezioni.
- Per le condizioni e per i cicli dovranno sempre essere utilizzate le parentesi graffe, anche con singoli statement.
- Le condizioni sui valori booleani dovranno seguire il seguente pattern: **if(variabile)** se si vuole controllare che il valore della variabile sia true, **if(!variabile)** viceversa

1.4. Definizioni, acronomi e abbreviazioni

Definizioni

- Off-the-shelf: servizi esterni al sistema di cui viene fatto utilizzo;
- JQuery: libreria JavaScript per applicazioni web;

Acronimi

- RAD: Requirements Analysis Document;
- SDD: System Design Document;
- ODD: Object Design Document;
- HTML: Hyper Text Markup Language;
- CSS: Cascading Style Sheet;
- XML: eXtensible Markup Language;

1.5. Riferimenti

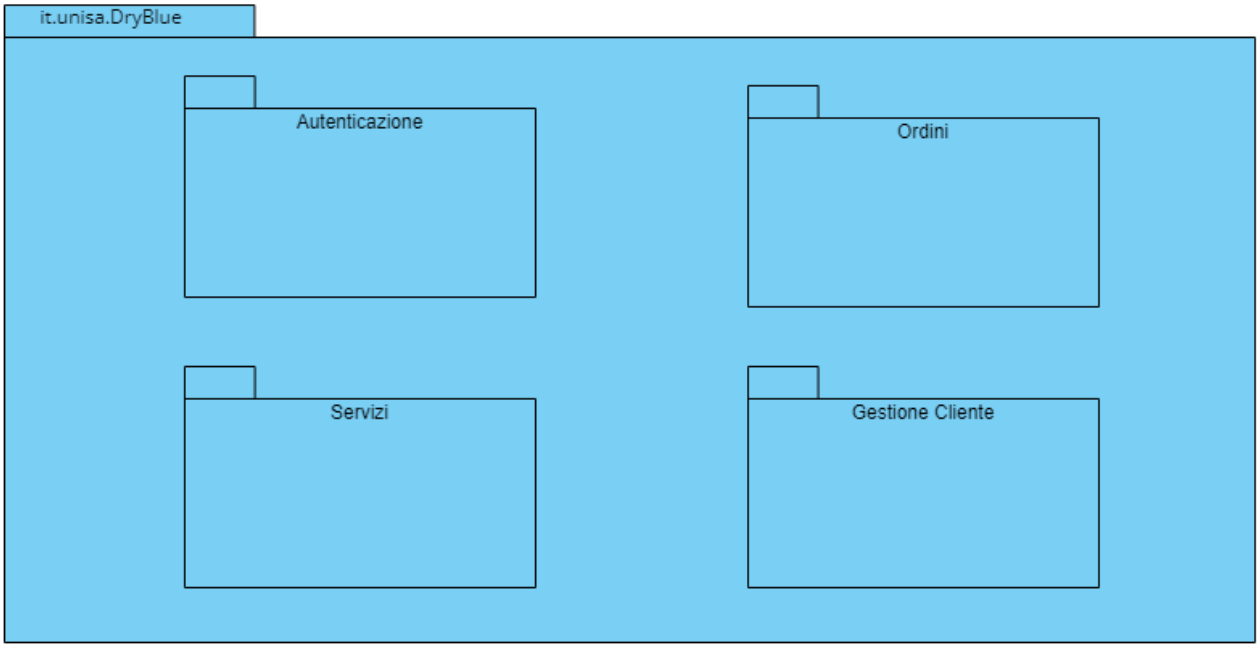
Object-Oriented Software Engineering Using UML, Patterns, and Java - Bernd Bruegge & Allen H. Dutoit- Pearson.
C08_DryBlue_RAD
C08_DryBlue_SDD

2.Packages (Responsabili: PDB, MF)

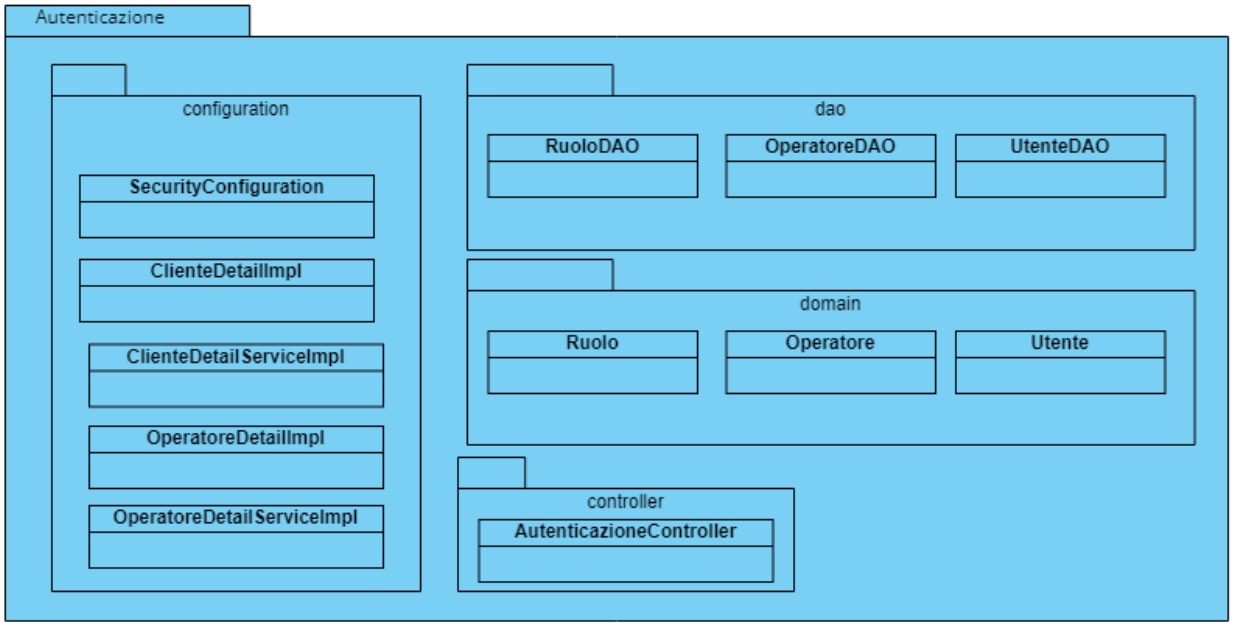
In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven.

- **.idea**
- **.mvn**, contiene tutti i file di configurazione del Maven
- **src**, contiene tutti i file sorgente
    - **main**
        - **java**, contiene le classi Java relative alle componenti Control e Model
        - **resources**, conitene i file relativi alle componenti View
            - **static**, contiene i fogli di stile CSS e gli script JavaScript
            - **templates**, contiene i file HMTL
    - **test**, contiene tutto il necessario per il testing
        - **java**, contiene le classi Java per l'implementazione del testing
- **target**, contiene tutti i file prodotti dal sistema di build Maven

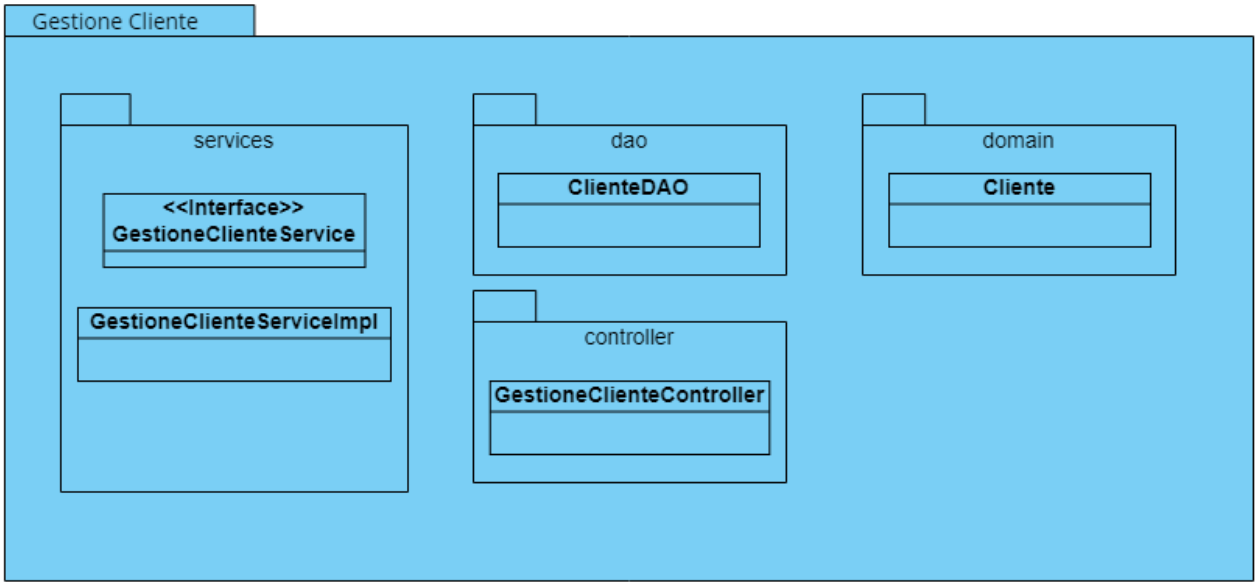
Viene mostrata prima il top-level, per poi mostrare il dettaglio di ciascun sottopackage, che ricalcano il pattern architetturale Three Layers.



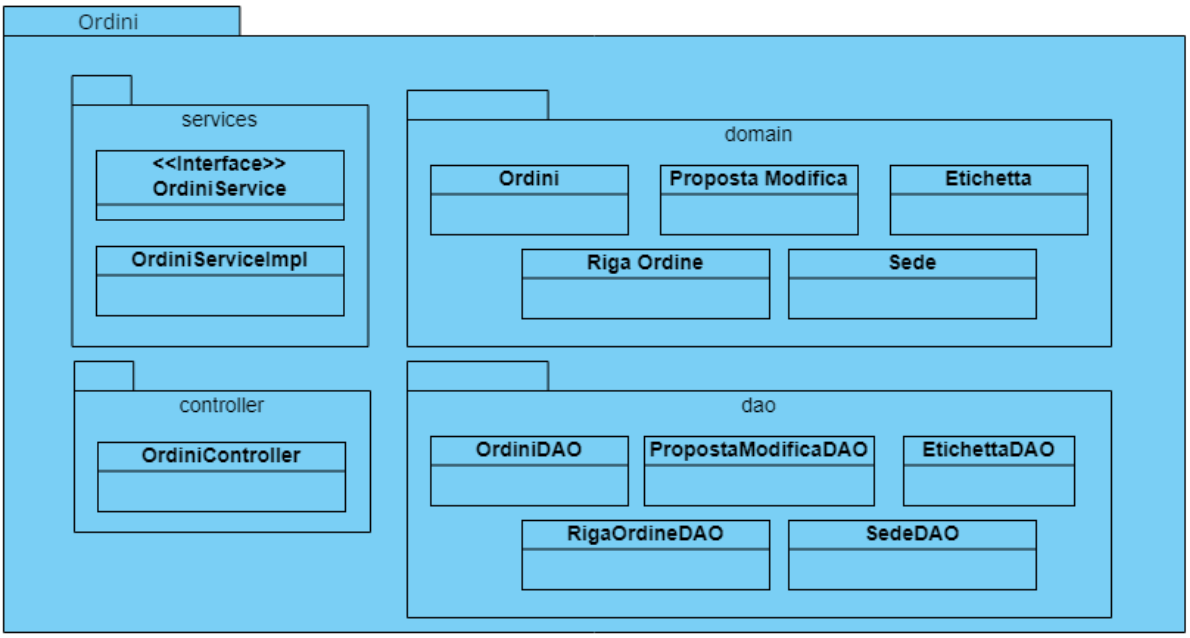
2.1. Package Autenticazione



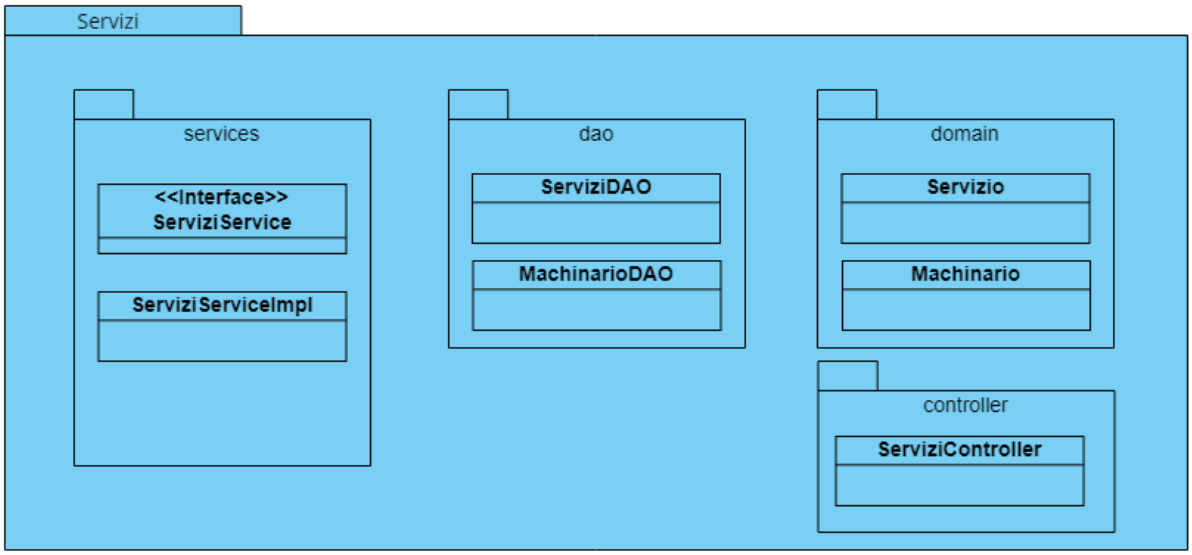
2.2. Package Gestione Cliente



2.3. Package Ordini



2.4. Package Servizi



3. Interfacce delle classi (Responsabili: PA, AS, SC)

Package Autenticazione

NOME CLASSE	UtenteDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Utente.
METODI	+findByUsername(String username): Utente +findByPassword(String password): Utente
INVARIANTE DI CLASSE	/

METODO +findByUsername (String Username): Utente	
DESCRIZIONE	Questa funzionalità consente di trovare un utente tramite la sua username.
PRE-CONDIZIONE	**context**: UtenteDAO:: findByUsername(username) **pre**: username!= null
POST-CONDIZIONE	/
METODO +login (String username, String password): Utente	
DESCRIZIONE	Questa funzionalità consente l'accesso alla piattaforma da parte dell'utente.
PRE-CONDIZIONE	**context**: UtenteDAO::findByPassword(password) **pre**: password!= null
POST-CONDIZIONE	/
METODO +findAll(): List <Utente>	
DESCRIZIONE	Questa funzionalità consente di trovare tutti gli utenti registrati.
PRE-CONDIZIONE	**context**: UtenteDAO:: findAll() **pre**:
POST-CONDIZIONE	/

NOME CLASSE	RuoloDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Utente.

METODI	/
INVARIANTE DI CLASSE	/

NOME CLASSE	OperatoreDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Utente.
METODI	+findByUsername(String username): Operatore
INVARIANTE DI CLASSE	/

METODO +findByUsername (String Username): Operatore	
DESCRIZIONE	Questa funzionalità consente di trovare un utente tramite la sua username.
PRE-CONDIZIONE	**context**: OperatoreDAO:: findByUsername(username) **pre**: username!= null
POST-CONDIZIONE	/

Package Servizi

NOME CLASSE	ServizioService
DESCRIZIONE	Questa classe permette la gestione dei servizi.
METODI	+aggiungiMacchinario(String denominazione, String matricola, String caratteristiche, String costruttore, String manutentore, String telefonoManutenzione, String stato, Sede sede): Macchinario +aggiornaStatoMacchinario(String Matricola,String stato): void +aggiungiServizio(String nome, String tipologia, String caratteristiche, double prezzo):Servizio

INVARIANTE DI CLASSE	/

METODO +aggiungiMacchinario (String denominazione, String matricola, String caratteristiche, String costruttore, String manutentore, String telefonoManutenzione, String stato, Sede sede): Macchinario	
DESCRIZIONE	Questa funzionalità consente di aggiungere un macchinario.
PRE-CONDIZIONE	**context**: ServiziService:: aggiungiMacchinario(denominazione, matricola, caratteristiche, costruttore, manutentore, telefonoManutenzione, stato, sede) **pre**: matricola!= null, denominazione!=null, caratteristiche!=null, costruttore!=null, manutentore!=null, telefonoManutenzione!=null, stato!=null, sede!=null
POST-CONDIZIONE	/
METODO +aggiornaStatoMacchinario (String matricola,String stato): void	
DESCRIZIONE	Questa funzionalità consente di cambiare lo stato dell'ordine.
PRE-CONDIZIONE	**context**: ServiziService:: aggiornaStatoMacchinario(matricola,stato) **pre**: matricola!=null, stato!= null
POST-CONDIZIONE	/
METODO +aggiungiServizio(String nome, String tipologia, String caratteristiche, double prezzo):Servizio	
DESCRIZIONE	Questa funzionalità consente di aggiungere un servizio al listino.
PRE-CONDIZIONE	**context**: ServiziService:: aggiungiServizio(nome, tipologia, caratteristiche, prezzo) **pre**: id_Servizio!= null, nome!=null, tipologia!=null, caratteristiche!=null, prezzo!=null
POST-CONDIZIONE	/

NOME CLASSE	ServiziDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Servizio.
METODI	/
INVARIANTE DI CLASSE	/

NOME CLASSE	MacchinarioDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Macchinario.
METODI	/
INVARIANTE DI CLASSE	/

Package ordini

OrdiniService

NOME CLASSE	OrdiniService
DESCRIZIONE	Questa classe permette la gestione degli ordini
METODI	+creazioneOrdine(Set<RigaOrdine> rigaOrdine, String Cliente, String tipologiaRitiro, String sede, LocalDate dataConsegnaDesiderata, String note): boolean +propostaModifica(LocalDate data, String sede, Ordine ordine): void +valutazionePropostaModifica(int idProposta): boolean +modificaOrdine(LocalDate data, Sede sede, String stato, int idOrdine): boolean +visualizzaOrdiniTotali(): List<Ordine> +visualizzaOrdiniOperatore(String filtro): List<Ordine> +visualizzaOrdiniUtente(String filtro, String telefono): List<Ordine> +stampaEtichetta(Ordine ordine): Etichetta +visualizzaSedi(): List <sede> +FindById(int idOrdine): Ordine +creaRigaOrdine(RigaOrdine riga): void

	+findByIndirizzo(String indirizzo): Sede
INVARIANTE DI CLASSE	/

METODO +creazioneOrdine(Set<RigaOrdine> rigaOrdine, String Cliente, String tipologiaRitiro, String sede, LocalDate dataConsegnaDesiderata, String note) : boolean	
DESCRIZIONE	Questa funzionalità consente di creare un ordine.
PRE-CONDIZIONE	**context**: OrdiniService:: creazioneOrdine(rigaOrdine, Cliente, tipologiaRitiro, sede, dataConsegnaDesiderata, note) **pre**: rigaOrdine=null, cliente!=null, tipologiaRitiro!=null, sede!=null, dataConsegnaDesiderata!=null, note=null
POST-CONDIZIONE	/
METODO +propostaModifica(LocalDate data, String sede, Ordine ordine) : boolean	
DESCRIZIONE	Questa funzionalità consente ad un cliente di effettuare una proposta di modifica.
PRE-CONDIZIONE	**context**: OrdiniService:: propostaModifica(data, sede, ordine)**pre**: data!=null,sede!=null,ordine!=null
POST-CONDIZIONE	/
METODO +valutazionePropostaModifica(int idProposta) : boolean	
DESCRIZIONE	Questa funzionalità consente ad un operatore di valutare una proposta di modifica.
PRE-CONDIZIONE	**context**: OrdiniService:: valutazionePropostaModifica(idProposta) **pre**: idProposta!=null
POST-CONDIZIONE	/
METODO +modificaOrdine(**LocalDate data, Sede sede, String stato, int idOrdine**) : boolean	
DESCRIZIONE	Questa funzionalità consente ad un operatore di modificare un ordine.
PRE-CONDIZIONE	**context**: OrdiniService:: modificaOrdine(data,sede, stato, idOrdine **pre**: data!=null, sede!=null, stato!=null, idOrdine!=null
POST-CONDIZIONE	/
METODO	

+visualizzaOrdiniOperatore(String filtro) : List<Ordine>	
DESCRIZIONE	Questa funzionalità consente di visualizzare una lista di ordini a seconda di un filtro passato per parametro.
PRE-CONDIZIONE	**context**: OrdiniService:: visualizzaOrdini(filtro)**pre**: filtro!=null
POST-CONDIZIONE	/
METODO +visualizzaOrdiniUtente(String filtro, String telefono) : List<Ordine>	
DESCRIZIONE	Questa funzionalità consente di visualizzare una lista di ordini a seconda di un filtro passato per parametro.
PRE-CONDIZIONE	**context**: OrdiniService:: visualizzaOrdini(filtro, telefono)**pre**: filtro!=null, telefono!=null
POST-CONDIZIONE	/
METODO +visualizzaOrdiniTotali() : List<Ordine>	
DESCRIZIONE	Questa funzionalità consente di stampare un'etichetta.
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +stampaEtichetta(Ordine ordine) : Etichetta	
DESCRIZIONE	Questa funzionalità consente di stampare un'etichetta.
PRE-CONDIZIONE	**context**: OrdiniService:: stampaEtichetta(ordine)**pre**: idOrdine!=null
POST-CONDIZIONE	/
METODO +visualizzaSedi() : List<sede>	
DESCRIZIONE	Questa funzionalità consente di visualizzare tutte le sedi.
PRE-CONDIZIONE	/
POST-CONDIZIONE	/
METODO +findById(int idOrdine) : Ordine	
DESCRIZIONE	Questa funzionalità consente di cercare un ordine.
PRE-CONDIZIONE	**context**: OrdiniService:: findById(int idOrdine)**pre**: idOrdine!=null
POST-CONDIZIONE	/
METODO +creaRigaOrdine(RigaOrdine riga) : void	
DESCRIZIONE	Questa funzionalità consente di creare una nuova riga ordine.

PRE-CONDIZIONE	**context**: OrdiniService::creaRigaOrdine(riga)**pre**: riga!=null
POST-CONDIZIONE	/
METODO +findByIndirizzo(String indirizzo) : Sede	
DESCRIZIONE	Questa funzionalità consente di cercare una sede tramite indirizzo.
PRE-CONDIZIONE	**context**: OrdiniService:: findByIndirizzo(indirizzo)**pre**: indirizzo!=null
POST-CONDIZIONE	/

OrdiniDAO

NOME CLASSE	OrdiniDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Ordini
METODI	+findAllByStato(String stato) : List<Ordine> +findAllByDataConsegna(Date data) : List<Ordine> +findAllByCliente(Cliente cliente) : List<Ordine>
INVARIANTE DI CLASSE	/

METODO +findAllByStato(String stato) : List<Ordine>	
DESCRIZIONE	Questa funzionalità consente di ricercare una lista di ordini tramite il loro stato.
PRE-CONDIZIONE	**context**: OrdiniDAO:: findAllByStato(stato)**pre**: stato!=null
POST-CONDIZIONE	/
METODO +findAllByDataConsegna(Date data) : List<Ordine>	
DESCRIZIONE	Questa funzionalità consente di ricercare una lista di ordini tramite la loro data.
PRE-CONDIZIONE	**context**: OrdiniService:: findAllByDataConsegna(date)**pre**: date!=null

POST-CONDIZIONE	/
METODO +findAllByCliente(Cliente cliente) : List<Ordine>	
DESCRIZIONE	Questa funzionalità consente di ricercare una lista di ordini tramite il cliente che li ha effettuati.
PRE-CONDIZIONE	**context**: OrdiniDAO:: findAllByCliente(cliente)**pre**: cliente!=null
POST-CONDIZIONE	/

PropostaModificaDAO

NOME CLASSE	PropostaModificaDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto PropostaModifica
METODI	+findById(int idProposta) : PropostaModifica
INVARIANTE DI CLASSE	/

METODO + findById(int idProposta) : PropostaModifica	
DESCRIZIONE	Questa funzionalità consente di ricercare una proposta di modifica tramite il suo ID.
PRE-CONDIZIONE	**context**: PropstaModificaDAO:: findById(idProposta)**pre**: idProposta!=null
POST-CONDIZIONE	/

EtichettaDAO

NOME CLASSE	EtichettaDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggeto Etichetta.
METODI	+findByOrdine(Ordine ordine) : Etichetta
INVARIANTE DI CLASSE	/

METODO +findById(int idEtichetta): Etichetta	
DESCRIZIONE	Questa funzionalità consente la stampa di un etichetta.

PRE-CONDIZIONE	**context**: EtichettaDAO:: findByOrdine(Ordine ordine) **pre**: ordine!=null
POST-CONDIZIONE	/

RigaOrdineDAO

NOME CLASSE	RigaOrdineDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggeto RigaOrdine.
METODI	+findAllByOrdine(int idOrdine) : List<RigaOrdine>
INVARIANTE DI CLASSE	/

METODO +findAllByOrdine(int idOrdine): List<RigaOrdine>	
DESCRIZIONE	Questa funzionalità consente la stampa di un'etichetta.
PRE-CONDIZIONE	**context**: RigaOrdineDAO:: findAllRigaOrdine(idOrdine) **pre**: idOrdine!=null
POST-CONDIZIONE	/

SedeDAO

NOME CLASSE	SedeDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto PropostaModifica
METODI	+findByIndirizzo(String indirizzo) : Sede
INVARIANTE DI CLASSE	/

METODO + find(int idSede) : PropostaModifica	
DESCRIZIONE	Questa funzionalità consente di ricercare una sede tramite il suo indirizzo.
PRE-CONDIZIONE	**context**: SedeDAO:: findByindirizzo(indirizzo)**pre**: indirizzo!=null
POST-CONDIZIONE	/

Package Gestione cliente

NOME CLASSE	ClienteDAO
DESCRIZIONE	Questa classe permette di gestire le query relative all'oggetto Cliente.
METODI	+findByNumTel(String numTel): Cliente +findByUsername(String username):Cliente +findByNome(String nome): Cliente +findByCognome(String cognome): Cliente
INVARIANTE DI CLASSE	/

METODO +findByNumTel(String numeroTel):Cliente	
DESCRIZIONE	Questa funzionalità consente di trovare un cliente dato il suo numero di telefono.
PRE-CONDIZIONE	**context**: ClienteDao:: findByNumTel(numTel) **pre**: numTel!= null
POST-CONDIZIONE	/
METODO +findByNumTel(String numeroTel):Cliente	
DESCRIZIONE	Questa funzionalità consente di trovare un cliente dato il suo numero di telefono.

PRE-CONDIZIONE	**context**: ClienteDao:: findByNumTel(numTel) **pre**: numTel!= null
POST-CONDIZIONE	/
METODO +findByUsername():Cliente	
DESCRIZIONE	Questa funzionalità consente di trovare un cliente dato il suo username.
PRE-CONDIZIONE	**context**: ClienteDAO:: findByUsername():Cliente **pre**: N/A
POST-CONDIZIONE	/
METODO +findByNome(String nome): Cliente	
DESCRIZIONE	Questa funzionalità consente di trovare il cliente in base al nome e ritornarlo.
PRE-CONDIZIONE	**Context:** ClienteDAO:: findByNome(String nome) : Cliente **Pre:** nome!=null
POST-CONDIZIONE	/
METODO +findByCognome(String cognome): Cliente	
DESCRIZIONE	Questa funzionalità consente di trovare il cliente in base al cognome e ritornarlo
PRE-CONDIZIONE	**Context:** ClienteDAO:: findByCognome(String cognome) : Cliente **Pre:** cognome!=null
POST-CONDIZIONE	/

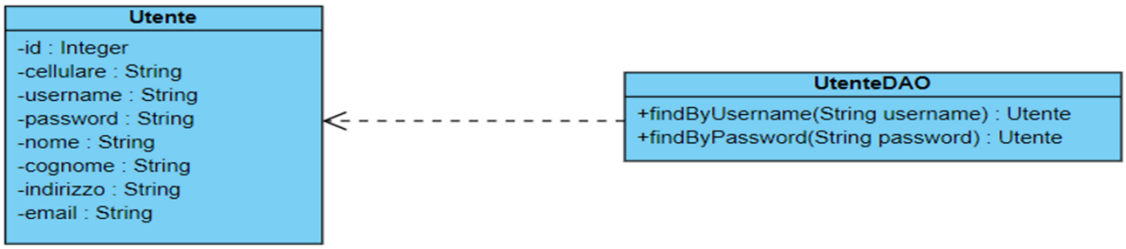
NOME CLASSE	GestioneClienteService
DESCRIZIONE	Questa classe permette la gestione del cliente.
METODI	+findTuttilClienti (): List<Cliente> +findByTelefono(String telefono): Cliente
INVARIANTE DI CLASSE	/

METODO

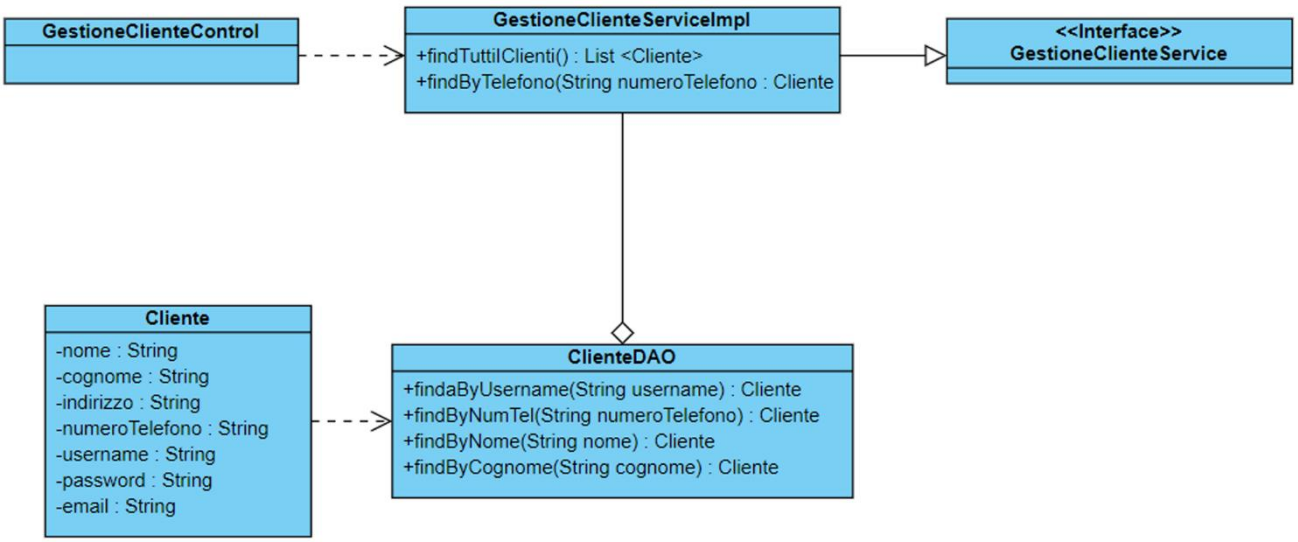
+findTuttiIClienti (): List<Cliente>	
DESCRIZIONE	Questa funzionalità consente di trovare tutti i clienti e restituire una lista.
PRE-CONDIZIONE	**Context**: GestioneClienteService::findTuttiIClienti(): List<Cliente> **pre**: N/A
POST-CONDIZIONE	/
METODO +findByTelefono(String telefono): Cliente	
DESCRIZIONE	Questa funzionalità permette di trovare un cliente dato il suo numero di telefono
PRE-CONDIZIONE	**Context:** GestioneClienteService:: findByTelefono(String telefono): Cliente **Pre:** N/A
POST-CONDIZIONE	/

Diagramma delle Classi (Responsabili: SM, PDB)

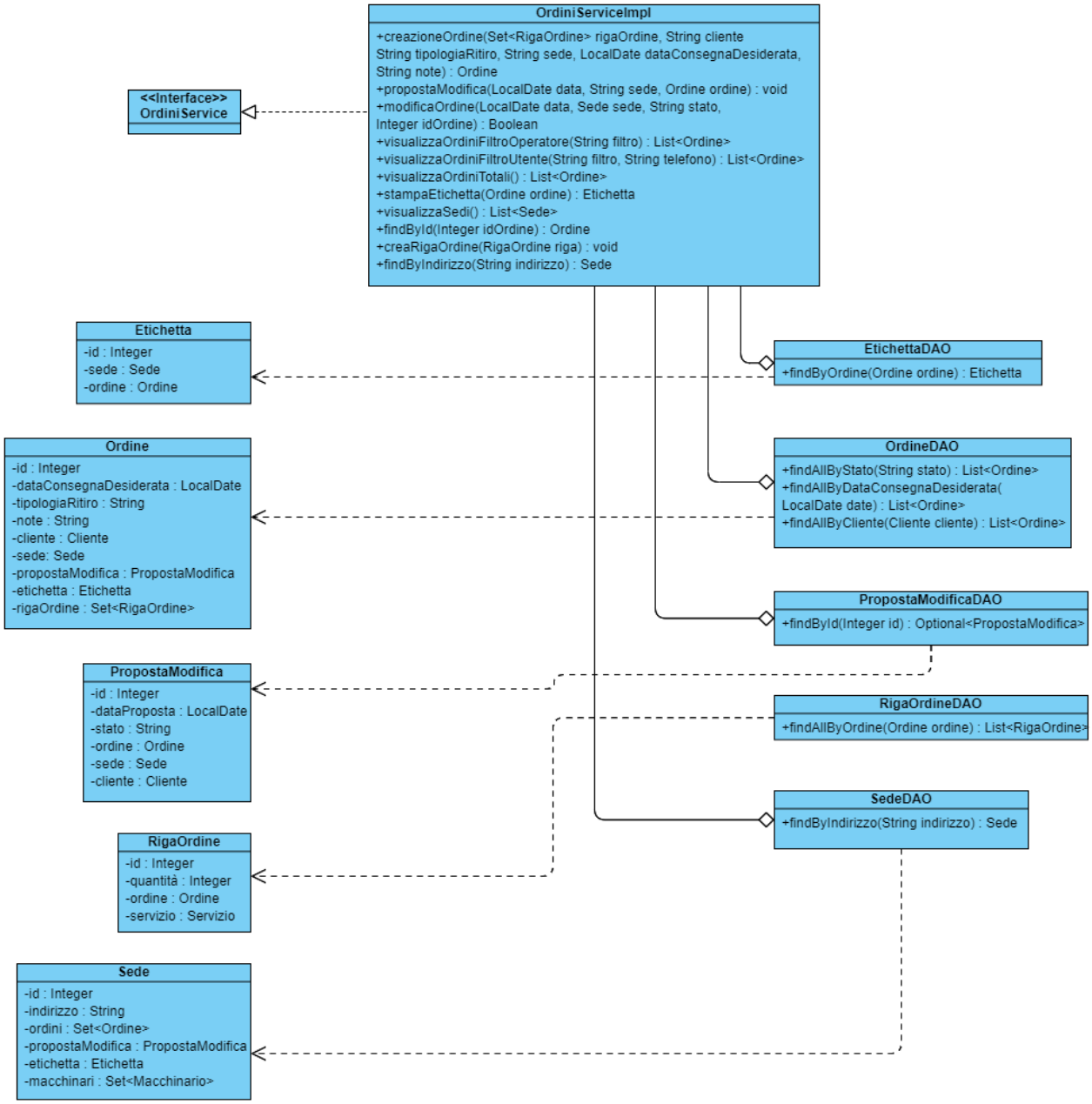
Autenticazione



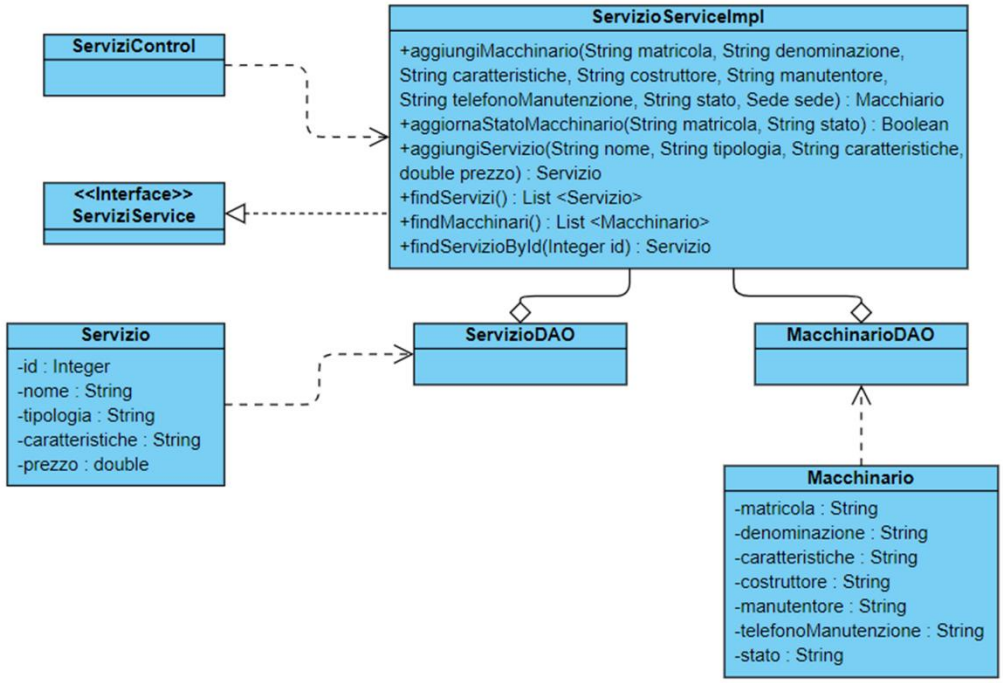
Gestione Cliente



Ordini


OrdiniServiceImpl
+creazioneOrdine(Set<RigaOrdine> rigaOrdine, String cliente
String tipologiaRitiro, String sede, LocalDate dataConsegnaDesiderata,
String note) : Ordine
+propostaModifica(LocalDate data, String sede, Ordine ordine) : void
+modificaOrdine(LocalDate data, Sede sede, String stato,
Integer idOrdine) : Boolean
+visualizzaOrdiniFiltroOperatore(String filtro) : List<Ordine>
+visualizzaOrdiniFiltroUtente(String filtro, String telefono) : List<Ordine>
+visualizzaOrdiniTotali() : List<Ordine>
+stampaEtichetta(Ordine ordine) : Etichetta
+visualizzaSedi() : List<Sede>
+findById(Integer idOrdine) : Ordine
+creaRigaOrdine(RigaOrdine riga) : void
+findByIndirizzo(String indirizzo) : Sede
<<Interface>>
OrdiniService
Etichetta
-id : Integer
-sede : Sede
-ordine : Ordine
EtichettaDAO
+findByOrdine(Ordine ordine) : Etichetta
Ordine
-id : Integer
-dataConsegnaDesiderata : LocalDate
-tipologiaRitiro : String
-note : String
-cliente : Cliente
-sede: Sede
-propostaModifica : PropostaModifica
-etichetta : Etichetta
-rigaOrdine : Set<RigaOrdine>
OrdineDAO
+findAllByStato(String stato) : List<Ordine>
+findAllByDataConsegnaDesiderata(
LocalDate date) : List<Ordine>
+findAllByCliente(Cliente cliente) : List<Ordine>
PropostaModificaDAO
+findById(Integer id) : Optional<PropostaModifica>
PropostaModifica
-id : Integer
-dataProposta : LocalDate
-stato : String
-ordine : Ordine
-sede : Sede
-cliente : Cliente
RigaOrdineDAO
+findAllByOrdine(Ordine ordine) : List<RigaOrdine>
SedeDAO
+findByIndirizzo(String indirizzo) : Sede
RigaOrdine
-id : Integer
-quantità : Integer
-ordine : Ordine
-servizio : Servizio
Sede
-id : Integer
-indirizzo : String
-ordini : Set<Ordine>
-propostaModifica : PropostaModifica
-etichetta : Etichetta
-macchinari : Set<Macchinario>

Servizi

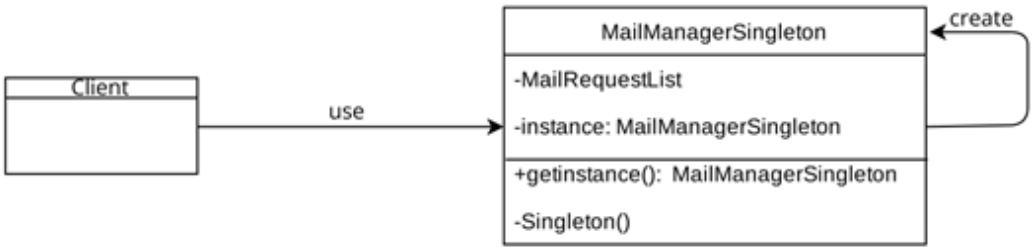


4. Design Patterns (Responsabili: AS, MF)

Singleton

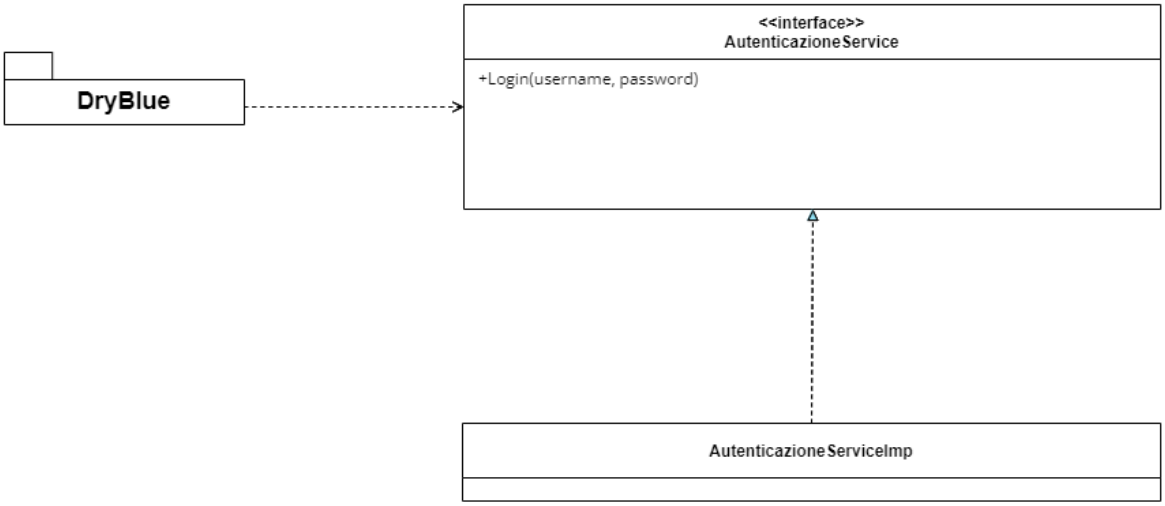
Singleton è un design pattern creazionale, si occupa di: istanziare degli oggetti, creare una sola istanza di una determinata classe e di fornire un solo punto d'accesso globale a quest'ultima.

DryBlue prevede un sistema di notifica via email per avvisare i clienti quando il capo è pronto o ci sono state modifiche all'ordine. Per gestire ciò senza accoppiare tra loro il sistema con un servizio di mailing o sistema di notifiche, viene creato un oggetto Singleton che alla modifica di determinati dati nel database, viene richiamato e genera la mail di notifica da inviare.



Facade

Il Facade è un design pattern che permette tramite un'interfaccia semplificata di accedere a sottosistemi più complessi. In questo modo si può nascondere al sistema la complessità dei framework o dei set di classi che si stanno utilizzando. Si garantisce in questo modo un alto disaccoppiamento e si rende la piattaforma più manutenibile ed aggiornabile.
DryBlue, essendo un sistema complesso, sfrutta il design pattern Facade per rendere più facile l'interfacciarsi con la logica di business. Nello specifico utilizza Facade per ogni suo sottosistema, implementandolo attraverso delle interfacce utilizzate per accedere ai metodi interni.
Di seguito un esempio di Facade nel sistema:



DAO

Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta per alcuni tipi di database. Il DAO fornisce alcune operazioni specifiche sui dati senza esporre i dettagli del database. I DAO sono utilizzabili nella maggior parte dei linguaggi e la maggior parte dei software con bisogni di persistenza. Principalmente viene associato con applicazioni JavaEE che usano database relazionali.
Essendo DryBlue una web application che punta a digitalizzare e gestire una lavanderia, che riceve anche molti ordini al giorno, ha bisogno di poter interagire con database in modo rapido e sicuro con numerose query per la moltitudine di dati da gestire. Per questo motivo abbiamo varie interfacce DAO all'interno del nostro sistema le quali, grazie all'utilizzo del framework di Spring (SpringJPARepository), vengono implementate in maniera del tutto automatica e trasparente al programmatore.
Qui è riportato un esempio di DAO utilizzato in DryBlue:

5. Glossario

Sigla/Termine	Definizione
Package Autenticazione	è una cartella contenente sottocartelle con risorse che rappresentano le informazioni del sistema di progettazione per autenticazione.
Package Gestione Cliente	è una cartella contenente sottocartelle con risorse che rappresentano le informazioni del sistema di progettazione per gestione Cliente.
Package Ordini	è una cartella contenente sottocartelle con risorse che rappresentano le informazioni del sistema di progettazione per Ordini.
Package Servizi	è una cartella contenente sottocartelle con risorse che rappresentano le informazioni del sistema di progettazione per Servizi.
Design Patterns	si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema ricorrente.
Singleton	è un design pattern creazionale che ha lo scopo di garantire che una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.
Facade	indica un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro.
DAO	è un pattern architetturale per la gestione della persistenza.