 ODD  
Object Design Document

|  |  |
| --- | --- |
|  |  |
| Versione | 1.0 |
| Data | 05/02/2022 |
| Destinatario | Prof. Carmine Gravino |
| Presentato da | Abbatiello Giuseppe, Ancona Marco, Boudad El Mehdi, De Palma Marco |
|  |  |

# Revision History

Di seguito è riportata la revision history del documento, per maggiori dettagli vedere il file excel del report contributi.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 21/01/2022 | 0.1 | Prima stesura | Team |
| 22/01/2022 | 0.2 | Stesura sezione 1 | Team |
| 23/01/2022 | 0.3 | Stesura sezione 1.1 | Team |
| 24/01/2022 | 0.4 | Stesura sezione 1.2 | Team |
| 25/01/2022 | 0.5 | Stesura sezione 1.3 | MA |
| 26/01/2022 | 0.6 | Stesura sezione 2 | EB |
| 27/01/2022 | 0.7 | Stesura sezione 3 | Team |
| 28/01/2022 | 0.8 | Stesura sezione 4 | Team |
| 29/01/2022 | 0.9 | Stesura sezione 5 | Team |
| 30/01/2022 | 1.0 | Stesura sezione 6 | MDP |
| 05/02/2022 | 1.0 | Revisione | Team |

# Membri del team

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Ruolo nel progetto | Acronimo | Informazioni di contatto |
| Giuseppe Abbatiello | Team Member | GA | g.abbatiello@studenti.unisa.it |
| El Mehdi Boudad | Team Member | EB | e.boudad@studenti.unisa.it |
| Marco De Palma | Team Member | MDP | [m.depalma6@studenti.unisa.it](mailto:m.depalma6@studenti.unisa.it) |
| Marco Ancona | Team Member | MA | m.ancona2@studenti.unisa.it |

Sommario

[Revision History 1](#_Toc2136806710)

[1 Introduzione 3](#_Toc1918862700)

[1.1 Object design goal 4](#_Toc750288533)

[1.2 Object trade off 4](#_Toc543364121)

[1.3 Definizioni, acronimi e abbreviazioni 4](#_Toc770427806)

[2 Packages 5](#_Toc980171226)

[3 Class Interfaces 9](#_Toc505907928)

[3.1 Package Registrazione 9](#_Toc672273507)

[3.2 Package Autenticazione 10](#_Toc1777316613)

[3.3 Package Gestione Utente 10](#_Toc992634286)

[3.4 Package Gestione Prodotto 11](#_Toc1667110835)

[3.5 - Package Gestione Acquisti 14](#_Toc766781743)

[3.6 Package Segnalazione 15](#_Toc1782760760)

[4 Class Diagram 16](#_Toc144714062)

[5 Design Patterns 17](#_Toc250965197)

[Façade 17](#_Toc109137095)

[DAO 17](#_Toc294542457)

[Singleton 18](#_Toc229016304)

[6 Glossario 18](#_Toc2006049947)

# Introduzione

Lo scopo del sistema PartnerShop è quello di permettere ai negozi fisici di raggiungere una maggiore platea di utenti e offrire un sito web facile utilizzare utilizzando attraverso una semplice registrazione.

## Object design goal

* **Sicurezza**: Il sistema PartnerShop offrirà una maggiore sicurezza delle password attraverso una criptazione SHA1.
* **Robustezza**: Il sistema dovrà risultare più robusto reagendo a situazioni di errore attraverso un’attenta gestione delle eccezioni.
* **Incapsulamento**: Il sistema garantisce l’uso dei dettagli implementativi attraverso l’uso di interfacce, facendo in modo che gli altri componenti possano interagire tra di loro solo attraverso esse.

## Object trade off

**Sicurezza vs Tempi di risposta**: Il server ogni volta che una password è inserita da qualsiasi utente prima di confrontare la password presente nel database andrà a trasformarla nella corrispondente stringa SHA1 così facendo ci sarà una maggiore sicurezza a discapito delle prestazioni.

**Robustezza vs Leggibilità**: Per realizzare la gestione delle eccezioni sarà necessario aggiungere molto codice al sistema per ogni componente, rendendolo di conseguenza meno leggibile.

## 1.3 Definizioni, acronimi e abbreviazioni

Vengono riportati di seguito alcune definizioni e abbreviazioni presenti nel documento:

* **Criptazione SHA1**: serie di algoritmi utilizzati per criptare file e password;
* **Incapsulamento**: caratteristica di Java che permette di aumentare la robustezza del codice, permettendo di accedere in modo controllato ad attributi o metodi;
* **Estensibilità**: facilità di aggiungere nuove funzionalità all’applicazione;
* **ODD**: Object Design Document;
* **JSP**: JavaServer Pages;

# 2 Packages

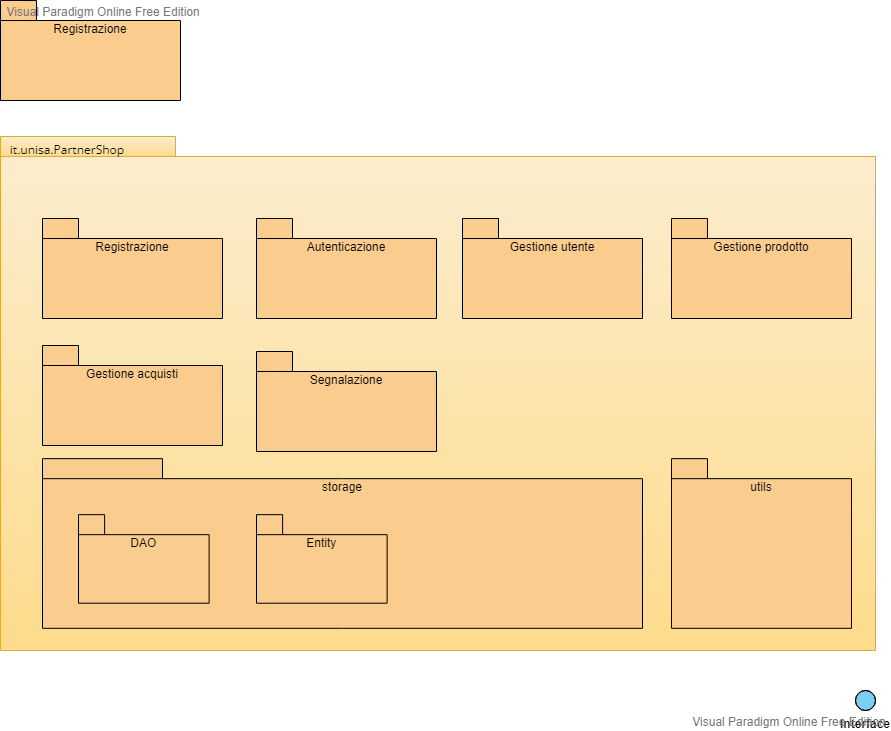
In questa sezione verranno descritti come sono suddivisi i package del sistema in base a quanto è stato descritto nel SDD. La struttura dei package riflette le scelte architetturali e rispecchia le directory utilizzate nell’implementazione finale.

* **.idea**
* **src,** contiene tutti i file sorgente;
* **main**
  + **java,** contiene le classi Java;
    - **PartnerShop,** contiene le cartelle relative alle varie funzionalità dell’app;
  + **webapp,** contiene tutti i file relativi al sito web;
    - **css,** contiene le pagine di stile;
    - **images,** contiene tutte le immagini relative al sito web;
    - **WEB-INF,** contiene tutte le JSP del sito web;
* **target,** contiene tutti i file prodotti dal sistema di build.

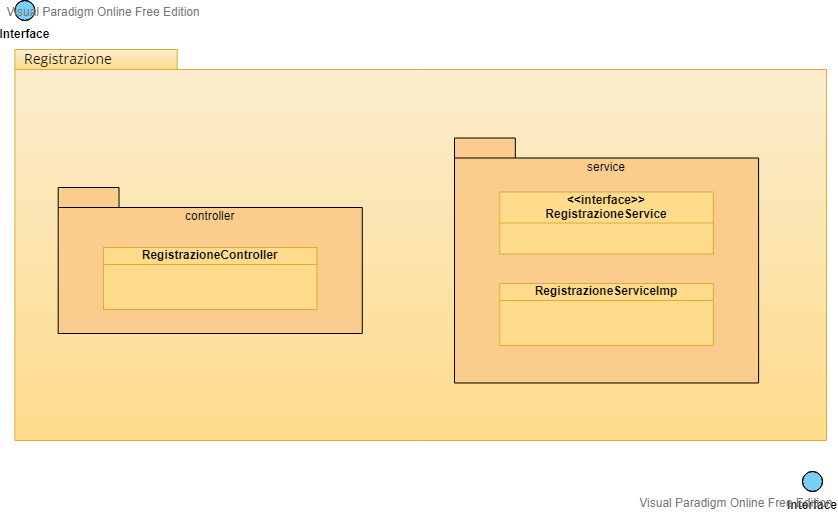
#### Package PartnerShop

In questa sezione viene mostrata la struttura del package principale di PartnerShop. La struttura generale è stata sviluppata in questo modo per tre motivi principali:

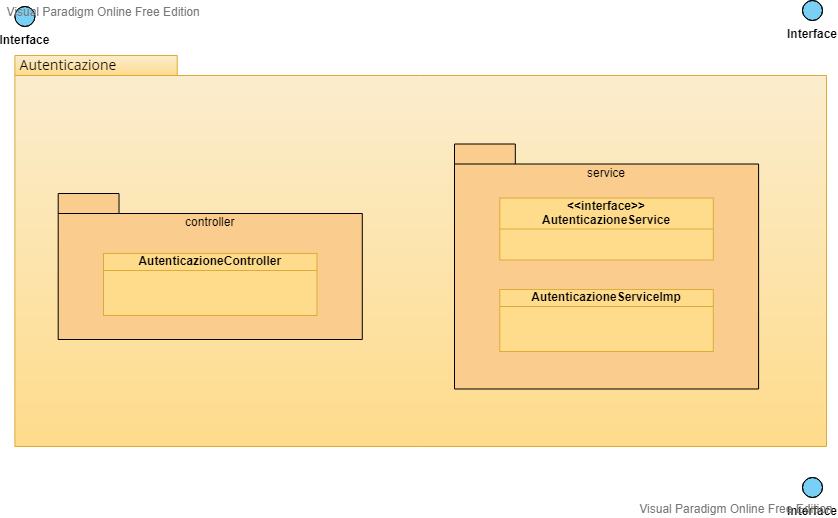
1. È stato creato un package separato per ogni sottosistema per contenere in maniera ordinata i service, i controller e altre eventuali classi usate soltanto dal sottosistema.
2. È stato creato un package separato per le classi storage, contenente l’entity e i DAO per l’accesso al DB. Scelta effettuata per tenere tutta la logica del database separata in un altro package ma collegata a tutti gli altri package dei sottosistemi.
3. È stato creato un package *utils* in cui vengono inserite classi utilizzate da più sottosistemi.



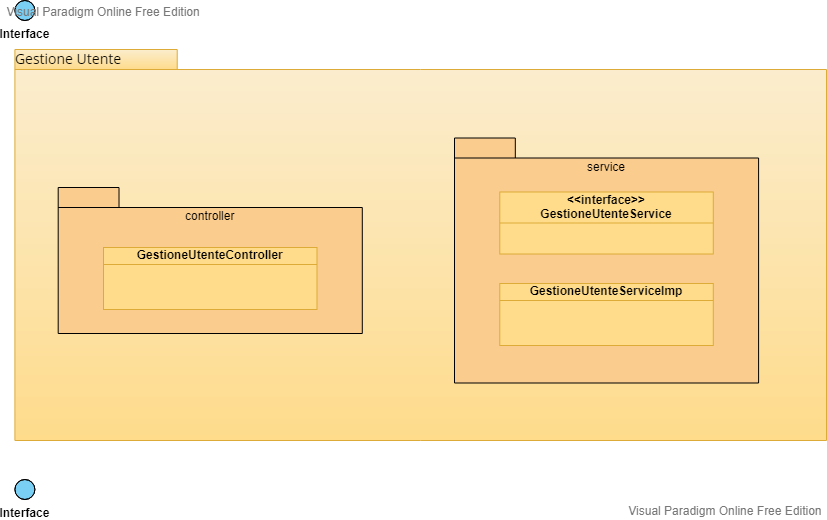
#### Package Registrazione



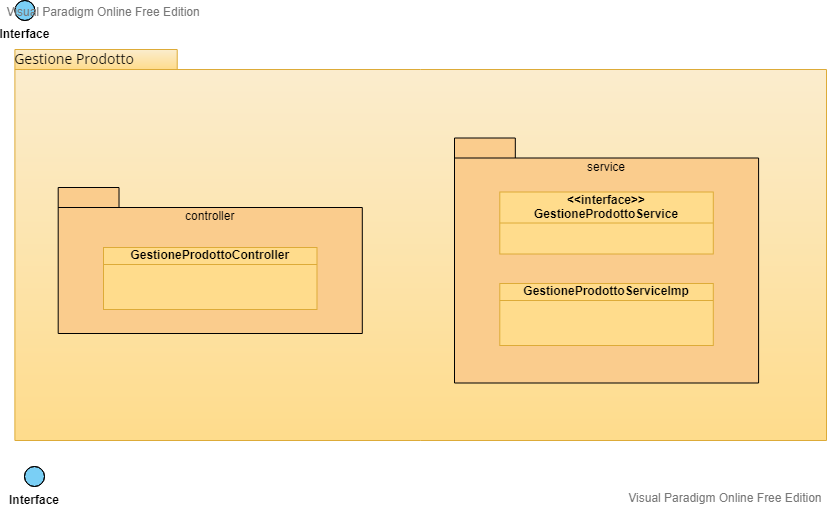
#### Package Autenticazione



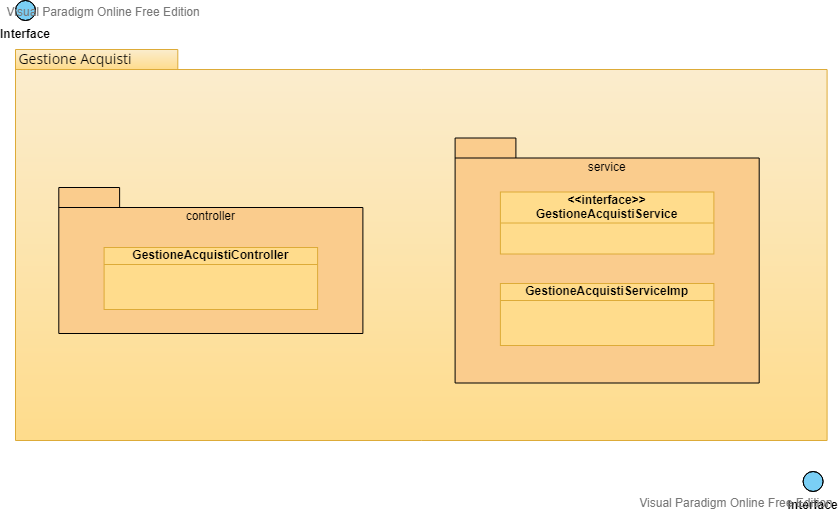
#### Package Gestione Utente



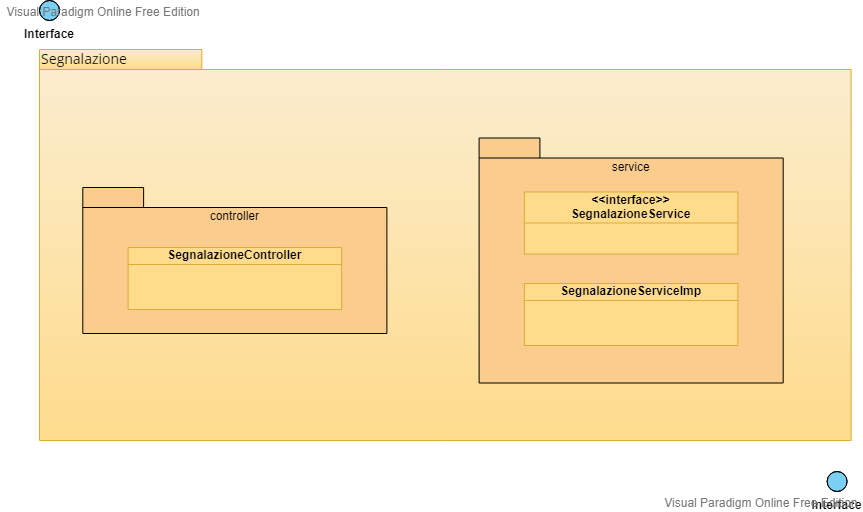
#### Package Gestione Prodotto



#### Package Gestione Acquisti



#### Package Segnalazione



# 3 Class Interfaces

Di seguito saranno presentate le interfacce di ciascun package le relative implementazioni e fornendo una descrizione solo dei loro metodi più rilevanti.

## 3.1 Package Registrazione

|  |  |
| --- | --- |
| Nome classe | RegistrazioneService |
| Descrizione | Permette di gestire le operazioni riguardanti la registrazione |
| Metodi | +registrazioneCliente(UtenteRegistrato ut): UtenteRegistrato  +registrazioneVenditore(UtenteRegistrato ut): UtenteRegistrato |
| Invariante di classe | / |

|  |  |
| --- | --- |
| Nome Metodo | +registrazioneCliente |
| Descrizione | Metodo che registra un nuovo cliente. |
| Pre-condizione | / |
| Post-condizione | **Context**: RegistrazioneService::RegistraCliente(Cliente cliente)  **Post**: ClienteDAO.doSave(cliente)==true |

|  |  |
| --- | --- |
| Nome Metodo | +registrazioneVenditore |
| Descrizione | Metodo che registra un nuovo venditore. |
| Pre-condizione | / |
| Post-condizione | **Context**: RegistrazioneService::RegistraVenditore(Venditore venditore)  **Post**: VenditoreDAO.doSave(venditore)==true |

## 3.2 Package Autenticazione

|  |  |
| --- | --- |
| Nome classe | AutenticazioneService |
| Descrizione | La classe permette di gestire le operazioni riguardanti l’autenticazione. |
| Metodi | +login(String email, String password): UtenteRegistrato  +logout(HttpSession sessione): Boolean |
| Invariante di classe | / |

|  |  |
| --- | --- |
| Nome Metodo | +login(String username, String password) |
| Descrizione | Questo metodo permette a un utente registrato di loggarsi. |
| Pre-condizione | **Context**: AutenticazioneService::login(String username, String password)  **Pre**: !username.isEmpty() and !password.isEmpty |
| Post-condizione | **Context**: AutenticazioneService::login(String username, String password)  **Post**: isCliente(ut) or isVenditore(ut) == true |

|  |  |
| --- | --- |
| Nome Metodo | +logout(HttpSession sessione) |
| Descrizione | Questo metodo permette a un loggato di effettuare il logout |
| Pre-condizione | **Context**: AutenticazioneService::logout(HttpSession sessione)  **Pre**: sessione.getAttribute(utente)!=null |
| Post-condizione | **Context**: AutenticazioneService:: logout(HttpSession sessione)  **Post**: sessione.getAttribute(utente)==null |

## 3.3 Package Gestione Utente

|  |  |
| --- | --- |
| Nome classe | GestioneUtenteService |
| Descrizione | La classe permette di gestire le operazioni riguardanti la gestione degli utenti. |
| Metodi | +ModificaDati(UtenteRegistrato ut): void  +CancellaUtente(String email): void  +VisualizzaUtenti(): ArrayList<UtenteRegistrato> |
| Invariante di classe | / |

|  |  |
| --- | --- |
| Nome Metodo | +ModificaDati(UtenteRegistrato ut) |
| Descrizione | Questo metodo permette a un utente registrato di modificare i propri dati personali. |
| Pre-condizione | **Context**: GestioneUtenteService::ModificaDati(UtenteRegistrato ut)  **Pre**: ut!=null |
| Post-condizione | **Context**: GestioneUtenteService::ModificaDati(UtenteRegistrato ut)  **Post**: ut!=null |

|  |  |
| --- | --- |
| Nome Metodo | +CancellaUtente(String email) |
| Descrizione | Questo metodo permette a un utente registrato di cancellare i propri dati personali. |
| Pre-condizione | **Context**: GestioneUtenteService::CancellaUtente(String email)  **Pre**: email!=null |
| Post-condizione | **Context**: GestioneUtenteService::CancellaUtente(String email)  **Post**: email==null |

|  |  |
| --- | --- |
| Nome Metodo | +VisualizzaUtenti() |
| Descrizione | Questo metodo permette a un amministratore di visualizzare tutti gli utenti registrati alla piattaforma. |
| Pre-condizione | **Context**: GestioneUtenteService::VisualizzaUtenti()  **Pre**: |
| Post-condizione | **Context**: GestioneUtenteService::VisualizzaUtenti()  **Post**: |

## 3.4 Package Gestione Prodotto

|  |  |
| --- | --- |
| Nome classe | GestioneProdottoService |
| Descrizione | La classe permette di gestire le operazioni riguardanti il prodotto. |
| Metodi | +getProdottoById(int id): Prodotto  +deleteProdottoById(int id): void  +doSaveProdotto(Prodotto p): void  +doUpdateProdotto(Prodotto p): void  +getAllProdotti( ): void  +getProdottiByVenditore(String email): ArrayList<Prodotto>  +getProdottiByCategoria(String cat): ArrayList<Prodotto>  +getProdottiByNome(String nome): ArrayList<Prodotto> |
| Invariante di classe | / |

|  |  |
| --- | --- |
| Nome Metodo | + getProdottoById(int id) |
| Descrizione | Questo metodo permette di visualzzare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: getProdottoById(int id)  **Pre**: id!=null && GestioneProdottoService.getProdottoById (id)!=null |
| Post-condizione | **Context**: GestioneProdottoService:: getProdottoById(int id)  **Post**: / |

|  |  |
| --- | --- |
| Nome Metodo | + deleteProdottoById(int id) |
| Descrizione | Questo metodo permette di eliminare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: deleteProdottoById(int id)  **Pre**: id!=null && GestioneProdottoService.getProdottoById (id)!=null |
| Post-condizione | **Context**: GestioneProdottoService:: deleteProdottoById(int id)  **Post**: GestioneProdottoService.getProdottoById (id)==null |

|  |  |
| --- | --- |
| Nome Metodo | + doSaveProdotto(Prodotto p) |
| Descrizione | Questo metodo permette di salvare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: doSaveProdotto(Prodotto p)  **Pre**: p!=null && GestioneProdottoService.getProdottoById (id)==null |
| Post-condizione | **Context**: GestioneProdottoService:: doSaveProdotto(Prodotto p)  **Post**: GestioneProdottoService.getProdottoById (id)!=null |

|  |  |
| --- | --- |
| Nome Metodo | + doUpdateProdotto(Prodotto p) |
| Descrizione | Questo metodo permette di eliminare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: doUpdateProdotto(Prodotto p)  **Pre**: p!=null && GestioneProdottoService.getProdottoById (id)!=null |
| Post-condizione | **Context**: GestioneProdottoService:: doUpdateProdotto(Prodotto p)  **Post**: / |

|  |  |
| --- | --- |
| Nome Metodo | + getAllProdotti( ) |
| Descrizione | Questo metodo permette di eliminare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: getAllProdotti( )  **Pre**: / |
| Post-condizione | **Context**: GestioneProdottoService:: getAllProdotti( )  **Post**: / |

|  |  |
| --- | --- |
| Nome Metodo | + getProdottiByVenditore(String email) |
| Descrizione | Questo metodo permette di eliminare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: getProdottiByVenditore(String email)  **Pre**: email!=null |
| Post-condizione | **Context**: GestioneProdottoService:: getProdottiByVenditore(String email)  **Post**: / |

|  |  |
| --- | --- |
| Nome Metodo | + getProdottiByCategoria(String cat) |
| Descrizione | Questo metodo permette di eliminare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: getProdottiByCategoria(String cat)  **Pre**: cat!=null |
| Post-condizione | **Context**: GestioneProdottoService:: getProdottiByCategoria(String cat)  **Post**: / |

|  |  |
| --- | --- |
| Nome Metodo | + getProdottiByNome(String nome) |
| Descrizione | Questo metodo permette di eliminare un prodotto. |
| Pre-condizione | **Context**: GestioneProdottoService:: getProdottiByNome(String nome)  **Pre**: nome!=null |
| Post-condizione | **Context**: GestioneProdottoService:: getProdottiByNome(String nome)  **Post**: / |

## 3.5 - Package Gestione Acquisti

|  |  |
| --- | --- |
| Nome classe | GestioneAcquistiService |
| Descrizione | La classe permette di gestire le operazioni riguardanti l’acquisto di un prodotto. |
| Metodi | +aggiungiAlCarrello(Carrello car, UtenteRegistrato ut, String prodottoIdStr, String quantStr):void  +rimuoviDalCarrello(UtenteRegistrato ut,Carrello car,int prodottoId,String setQuantStr):void  +acquistaProdotto(UtenteRegistrato ut,Carrello car,String indirizzo,String cardc):void  +visualizzaOrdini(UtenteRegistrato ut):ArrayList<Ordine> |
| Invariante di classe | **Context :** Carrello inv:  self!=nil |

|  |  |
| --- | --- |
| Nome Metodo | +aggiungiAlCarrello(Carrello car, UtenteRegistrato ut, String prodottoIdStr, String quantStr) |
| Descrizione | Questo metodo permette ad un cliente od Ospite di aggiungere un prodotto al carrello. |
| Pre-condizione | **Context**: GestioneAcquistiService::aggiungiAlCarrello(Carrello car, UtenteRegistrato ut, String prodottoIdStr, String quantStr)  **Pre**: !prodottoIdStr.isEmpty() AND !quantStr.isEmpty() AND quantitaProd = car.getQuant((Integer)prodottiIdStr) AND quantitaProd >= 0 |
| Post-condizione | **Context**: GestioneAcquistiService::aggiungiAlCarrello(Carrello car, UtenteRegistrato ut, String prodottoIdStr, String quantStr)  **Post**: car.getProdottoHash().contains((Integer)prodottoIdStr) AND quantitaProd+=(Integer)quantStr |

|  |  |
| --- | --- |
| Nome Metodo | +rimuoviDalCarrello(UtenteRegistrato ut,Carrello car,int prodottoId,String setQuantStr) |
| Descrizione | Questo metodo permette ad un cliente od Ospite di rimuovere un prodotto dal carrello. |
| Pre-condizione | **Context**: GestioneAcquistiService::rimuoviDalCarrello(UtenteRegistrato ut,Carrello car,int prodottoId,String setQuantStr)  **Pre**: !prodottoIdStr.isEmpty() AND !setQuantStr.isEmpty() |
| Post-condizione | **Context**: GestioneAcquistiService:: rimuoviDalCarrello(UtenteRegistrato ut,Carrello car,int prodottoId,String setQuantStr)  **Post**: !car.getProdottoHash().contains((Integer)prodottoIdStr) AND !car.getQuantHash.contains((Integer)prodottoIdStr) |

|  |  |
| --- | --- |
| Nome Metodo | +acquistaProdotto(UtenteRegistrato ut,Carrello car,String indirizzo,String cardc):void |
| Descrizione | Questo metodo permette ad un cliente od Ospite di rimuovere un prodotto dal carrello. |
| Pre-condizione | **Context**: GestioneAcquistiService:: acquistaProdotto(UtenteRegistrato ut,Carrello car,String indirizzo,String cardc)  **Pre**: car.getProdottiHash().size()>0 AND !indirizzo.isEmpty() AND !cardc.isEmpty() AND ut!=null AND ut.getTipo()==0 |
| Post-condizione | **Context**: GestioneAcquistiService::acquistaProdotto(UtenteRegistrato ut,Carrello car,String indirizzo,String cardc)  **Post**: car.getProdottiHash().size()==0 |

|  |  |
| --- | --- |
| Nome Metodo | +visualizzaOrdini(UtenteRegistrato ut):ArrayList<Ordine> |
| Descrizione | Questo metodo permette ad un cliente od Ospite di rimuovere un prodotto dal carrello. |
| Pre-condizione | **Context**: GestioneAcquistiService::visualizzaOrdini(UtenteRegistrato ut)  **Pre**: ut!=null |
| Post-condizione | **Context**: GestioneAcquistiService::visualizzaOrdini(UtenteRegistrato ut)  **Post**: / |

## 3.6 Package Segnalazione

|  |  |
| --- | --- |
| Nome classe | SegnalazioneService |
| Descrizione | La classe permette di gestire le operazioni riguardanti la segnalazione di un utente. |
| Metodi | +aggiungiSegnalazione(Segnalazione segn):Boolean  +VisualizzaListaSegnalazioni(int par):ArrayList<Segnalazione>  +VisualizzaSegnalazione(int id): Segnalazione  +ChiudiSegnalazione(int id): Boolean |
| Invariante di classe | Context Segnalazione inv:  self.emailCliente != null |

|  |  |
| --- | --- |
| Nome Metodo | +aggiungiSegnalazione(Segnalazione segn) |
| Descrizione | Questo metodo permette ad un cliente di effettuare una segnalazione. |
| Pre-condizione | / |
| Post-condizione | **Context**: SegnalazioneService::aggiungiSegnalazione(Segnalazione segn)  **Post**: segn.doSave()==true |

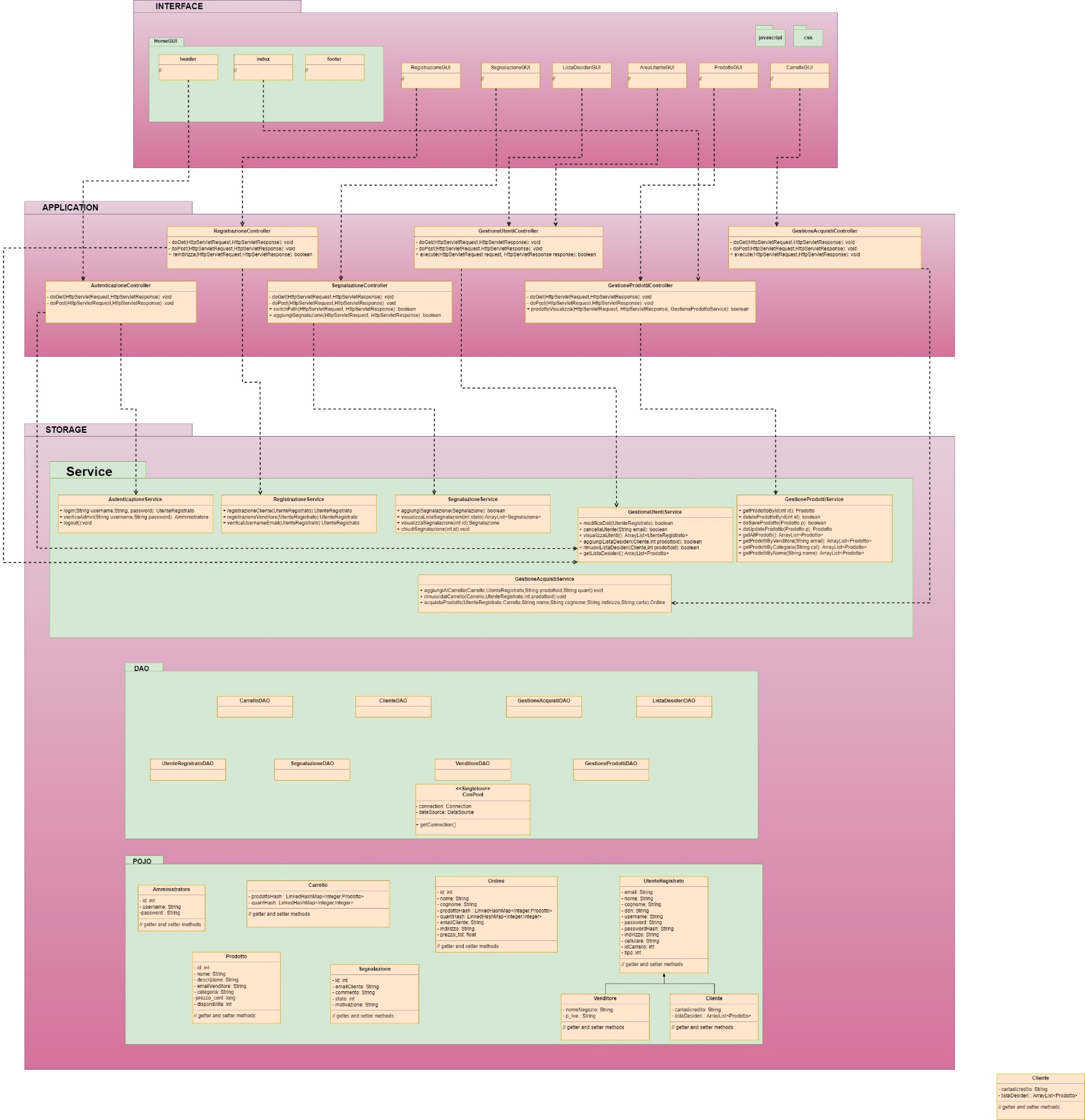
|  |  |
| --- | --- |
| Nome Metodo | +visualizzaSegnalazione(int id) |
| Descrizione | Questo metodo permette ad un amministratore di visualizzare una segnalazione. |
| Pre-condizione | **Context**: SegnalazioneService::visualizzaSegnalazione(int id)  **Pre**: Id>0 |
| Post-condizione | **Context**: SegnalazioneService::visualizzaSegnalazione(int id)  **Post**: segn!=null |

|  |  |
| --- | --- |
| Nome Metodo | +visualizzaListaSegnalazione(int par) |
| Descrizione | Questo metodo permette ad un amministratore di visualizzare la lista delle segnalazioni |
| Pre-condizione | **Context**: SegnalazioneService::visualizzaListaSegnalazione(int par)  **Pre**: Id>0 |
| Post-condizione | **Context**: SegnalazioneService::visualizzaListaSegnalazione(int par)  **Post**: listSegn!=null |

|  |  |
| --- | --- |
| Nome Metodo | +chiudiSegnalazione(int id) |
| Descrizione | Questo metodo permette ad un amministratore di chiudere una segnalazione. |
| Pre-condizione | **Context**: SegnalazioneService::chiudiSegnalazione(int id)  **Pre**: Id>0 |
| Post-condizione | **Context**: SegnalazioneService::chiudiSegnalazione(int id)  **Post**: segn.aggiornaStato(id)==true |

# 4 CD\_ODD: Class diagram

Nel seguente Class Diagram sono state fatte delle modifiche rispetto a quello presente nel RAD sono stati aggiunti due attributi al carrello e all’ordine ProdottoHash e QuantitàHash necessari per l’implementazione



E’ possibile visualizzare il class diagram al seguente link:

[CD\_ODD](https://1drv.ms/u/s!AtmXvyhj0lGIg_YrWtbpb7J_MlvTZQ?e=9zBYna)

# 5 Design Patterns

## Façade

Il Facade è un design pattern che permette, implementando una interfaccia semplificata di realizzare un’architettura chiusa. In questo modo si può nascondere al sistema la complessità delle librerie, dei framework o dei set di classi che si stanno usando. Si garantisce così un alto disaccoppiamento e si rende la piattaforma più manutenibile e più aggiornabile, poiché basterà cambiare l’implementazione dei metodi dell’interfaccia per implementare le modifiche.

All’interno di PartnerShop il design Pattern façade viene utilizzato attraverso l’uso di interfacce e delle implementazioni ad esso associate che applicano la logica di business in modo che tutti gli altri sottosistemi possono interfacciarsi con esso.

## DAO

Un DAO (Data Access Object) è un pattern che offre un’interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell’applicazione allo stato persistente, 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.

## Singleton

Il singleton è il design pattern che appartiene alla categoria dei design pattern creazionali, ed ha come scopo quello di garantire che una determinata classe venga creata una ed una sola istanza, e di fornire un punto di accesso globale a tale istanza.

All’interno del sistema PartnerShop il singleton è stato implementato attraverso la classe ConPool per la connessione al database, rendendo privato il costruttore e facendo in modo che la classe sia istanziata solo una volta ovvero quando essa viene caricata in memoria.

# 6 Glossario

|  |  |
| --- | --- |
| Sigla/Termine | Definizione |
| Package | Raggruppamento di classi ed interfacce. | |
| DAO | Data Access Object, implementazione dell’omonimo pattern architetturale che si occupa di fornire un accesso in modo astratto ai dati persistenti. | |
| Controller | Classe che si occupa di gestire le richieste effettuate dal client. | |
| Service | Classe che implementa la logica di business, viene utilizzata dal controller o da un altro sottosistema. | |
| Facade | Un oggetto che permette, attraverso un’interfaccia più semplice, l’accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro. | |
| Singleton | È un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga struttura una solo istanza e di fornire un punto di accesso globale a tale istanza. | |