Università degli Studi di Salerno **Corso di Ingegneria del Software**

***Object Design Document 1.0***

***09/12/2022***

****

0512107476 **– Salvatore Santoriello**0512107395 **– Bruno Farano**0512106126 **– Emanuele Milito**0512107080 **– Gianmichele Cancellaro**

***Revision History***

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 09/12/2022 | 1.0 | Prima stesura del documento | Membri del team |

***Indice***

**1.** Introduzione  
 **1.1***Object Design Trade-Offs* **1.1.1** *Sicurezza VS Costi* **1.1.2** *Tempo di risposta VS Hardware* **1.1.3** *Persistenza VS Efficienza* **1.1.4** *Comprensibilità VS Tempo* **1.2***Riferimenti*

**2.** Packages and Class Interfaces

**2.1***Packages* **2.2***Class Interfaces*

**3.** Design Pattern

**1.** Introduzione  
 **1.1** *Object Design Trade-Offs* **1.1.1** *Sicurezza VS Costi*

A causa dei tempi di sviluppo ridotti e del budget per il progetto ridotto, si sceglie di limitare la sicurezza implementandola solamente attraverso la cifratura della password degli utenti della piattaforma. La sicurezza dei dati personali sarà preservata attraverso l’utilizzo di credenziali di accesso all’area personale (email e password).

**1.1.2** *Tempo di risposta VS Hardware*

Il tempo di risposta è un aspetto fondamentale per la piattaforma e per la sua utenza, esso verrà comunque influenzato anche dall’hardware sul quale verrà installato il sistema.

**1.1.3** *Persistenza VS Efficienza*

Si è scelto di optare per una memorizzazione persistente dei dati mediante un DBMS per un database relazionale, aumentando il tempo di risposta e quindi diminuendo l’efficienza.

**1.1.4** *Comprensibilità VS Tempo*

Il codice deve essere quanto più comprensibile possibile per favorire la manutenibilità e quindi eventuali modifiche e risoluzione di problemi futuri. Il codice quindi dovrà essere accompagnato da commenti che permettono di comprendere meglio le operazioni effettuate. Questa caratteristica richiederà più tempo per lo sviluppo del progetto.

**1.2** *Riferimenti*

Si fa riferimento ai seguenti documenti:

* Requirements Analysis Document (RAD eSoccerce)
* System Design Document (SSD eSoccerce)

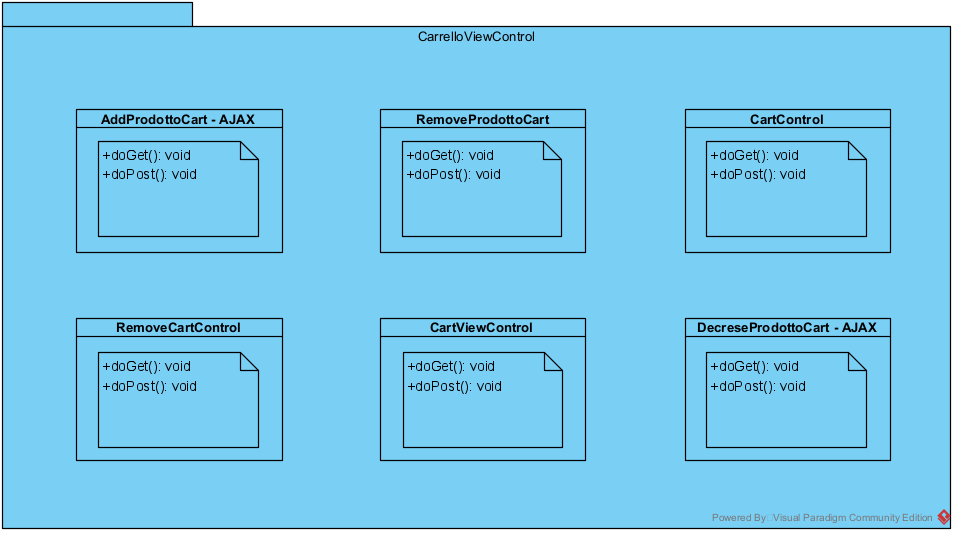
**2.** Packages and Class Interfaces

**2.1***Packages*

Con la fase di System Decomposition sono stati individuati diversi packages.  
Per il livello View-Control:

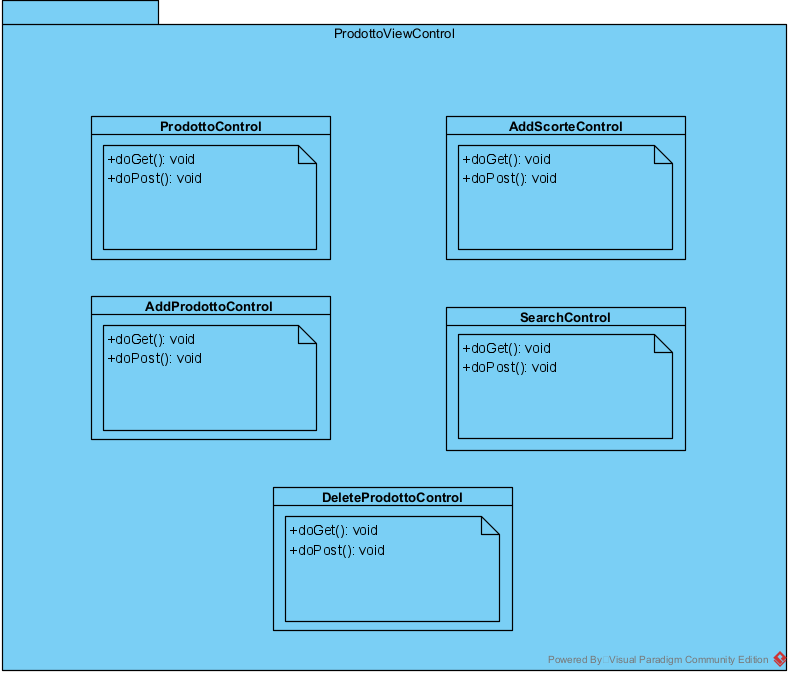
* *CarrelloViewControl*
* *ProdottoViewControl*
* *RecensioniViewControl*
* *OrdiniViewControl*
* *ProfiloViewControl*

**P(1) | CarrelloViewControl**

**

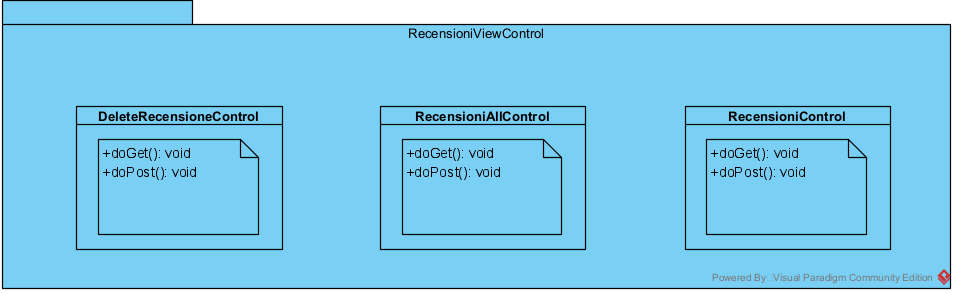
* **AddProdottoCart –** è la servlet che permette di aumentare la quantità di un prodotto da acquistare all’interno dal carrello usando AJAX
* **RemoveProdottoCart –** è la servlet che permette di rimuovere un prodotto dal carrello
* **CartControl –** è la servlet che permette di aggiungere un prodotto al carrello
* **RemoveCartControl –** è la servlet che permette di svuotare il carrello
* **DecreaseProdottoCart –** è la servlet che permette di diminuire la quantità di un prodotto da acquistare all’interno del carrello
* **CartViewControl** – è la servlet utilizzata per la visualizzazione della pagina del carrello

**P(2) | ProdottoViewControl**

****

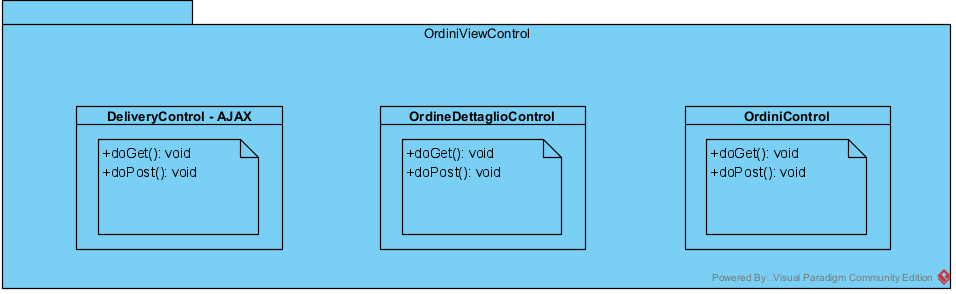
* **ProdottoControl –** è la servlet che permette di visualizzare la pagina dei prodotti, in base al parametro passato permette di visualizzare una determinata categoria di prodotti
* **AddScorteControl** – è la servlet che permette di aggiungere scorte di un prodotto esistente sulla piattaforma da parte di un admin
* **AddProdottoControl** – è la servlet che permette di aggiungere un nuovo prodotto sulla piattaforma da parte di un admin
* **DeleteProdottoControl** – è la servlet che permette di rimuovere un prodotto dalla piattaforma da parte di un admin
* **SearchControl –** è la servlet che permette di ricercare un prodotto tramite navbar a un utente

**P(3) | RecensioniViewControl**

****

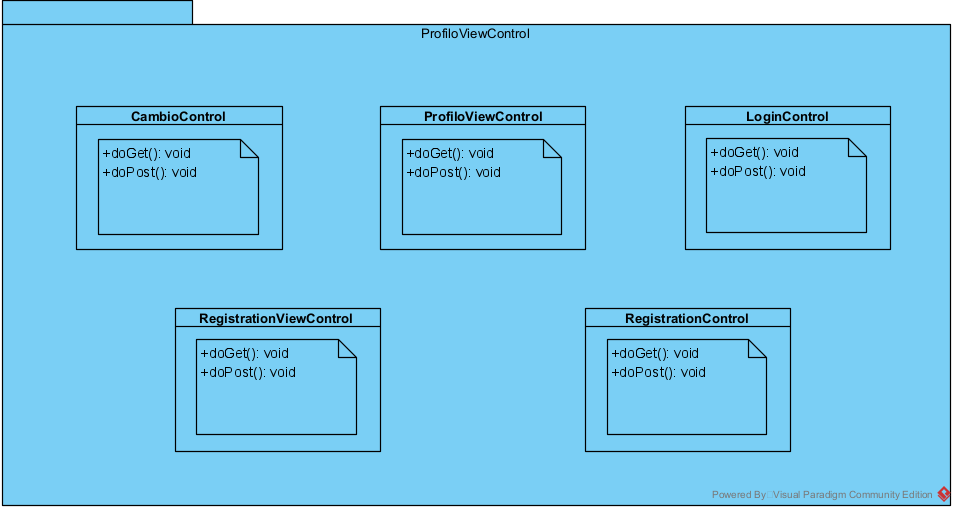
* **DeleteRecensioneControl –** è la servlet che permette all’utente di rimuovere una propria recensione
* **RecensioniAllControl –** è la servlet che permette all’utente di visualizzare tutte le recensioni per un determinato prodotto
* **RecensioniControl –** è la servlet che permette all’utente di aggiungere una recensione per un determinato prodotto

**P(4) | OrdiniViewControl**

****

* **DeliveryControl –** è la servlet che permette all’utente di notificare l’avvenuta spedizione di un proprio ordine tramite AJAX
* **OrdineDettaglioControl –** è la servlet che permette all’utente di visualizzare i dettagli di un determinato ordine da lui effettuato
* **OrdiniControl –** è la servlet che permette all’utente di terminare il pagamento e concludere un ordine

**P(5) | ProfiloViewControl**

****

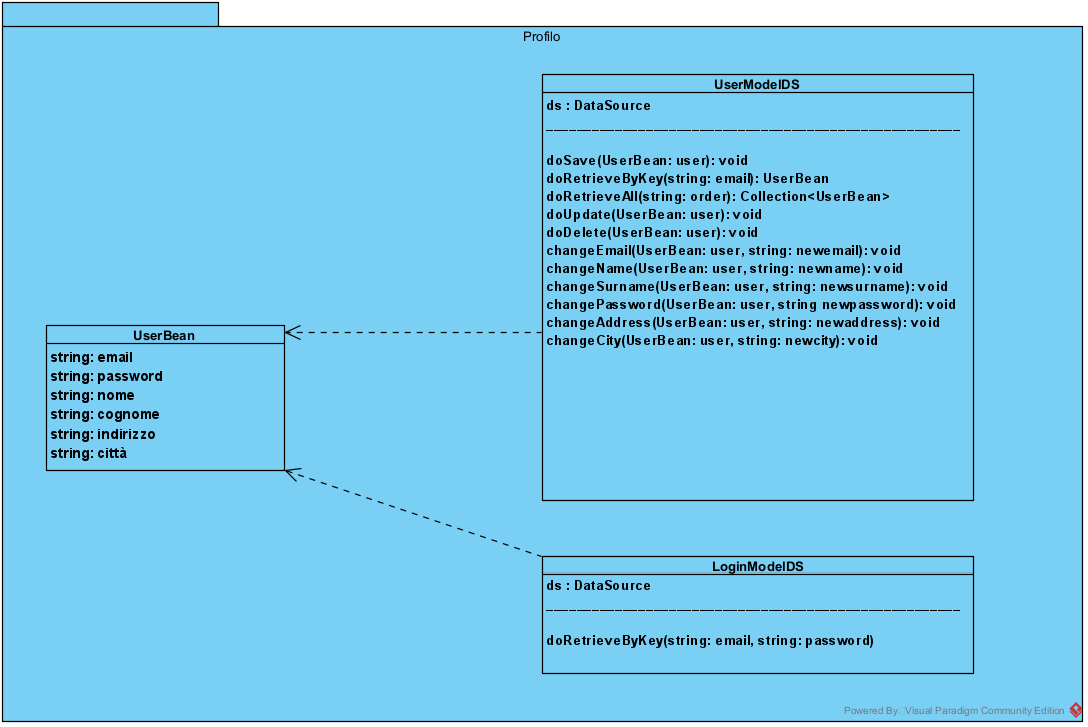
* **CambioControl –** è la servlet che permette all’utente di effettuare la procedura di una modificato di un dato personale
* **ProfiloViewControl –** è la servlet che permette all’utente di visualizzare la propria pagina del profilo
* **LoginControl -** è la servlet che permette di effettuare la procedura di login all’utente
* **RegistrationViewControl –** è la servlet che permette di visualizzare la pagina di registrazione all’utente
* **RegistrationControl –** è la servlet che permette di effettuare la procedura di registrazione all’utente

**2.2** *Class Interfaces*

Per il livello di logica di business sono state individuate le seguenti class interfaces:

* *Profilo*
* *Ordine*
* *Recensione*
* *Prodotto*
* *Carrello*

**C(1) | Profilo**

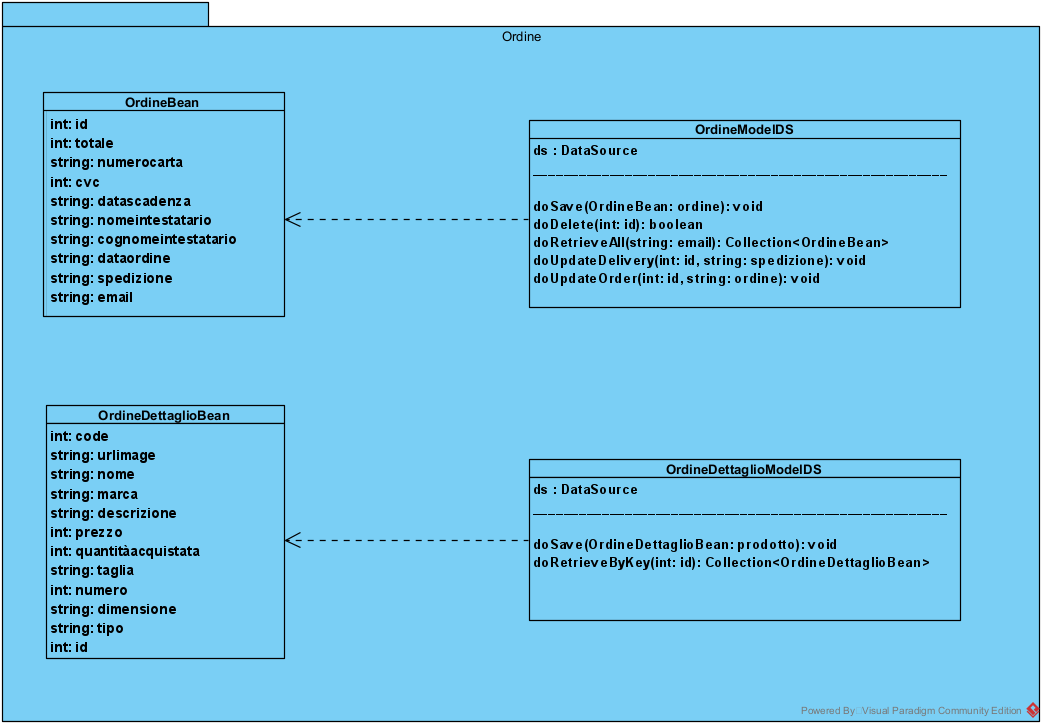
****

* **UserBean –** rappresenta l’entità Utente all’interno del database

|  |  |
| --- | --- |
| UserModelDS | |
| *Descrizione* | Questa classe è un manager che si occupa di interagire con il database e permette di interrogare la tabella Utente |
| *Pre-condizione* | **Context UserModelDS: doSave(UserBean: user)** *Pre: user != null* **Context UserModelDS: doRetrieveByKey(string: email)**  *Pre: email != null*  **Context UserModelDS: doRetrieveAll(string: order)**  *Pre: order != null*  **Context UserModelDS: doUpdate(UserBean: user)**  *Pre: user != null*  **Context UserModelDS: doDelete(UserBean: user)**  *Pre: user != null*  **Context UserModelDS: changeEmail(UserBean: user, string: newemail)**  *Pre: user != null, newemail != null*  **Context UserModelDS: changePassword (UserBean: user, string: newpassword)**  *Pre: user != null, newpassword != null*  **Context UserModelDS: changeName(UserBean: user, string: newname)**  *Pre: user != null, newname != null*  **Context UserModelDS: changeSurname(UserBean: user, string: newsurname)**  *Pre: user != null, newsurname != null*  **Context UserModelDS: changeAddress(UserBean: user, string: newaddress)**  *Pre: user != null, newaddress != null*  **Context UserModelDS: changeCity(UserBean: user, string: newcity)**  *Pre: user != null, newcity != null* |
| *Post-condizione* | **Context UserModelDS: doSave(UserBean: user)** *Post: viene istanziata una nuova riga nella tabella Utente, con gli attributi relativi all’oggetto user* **Context UserModelDS: doRetrieveByKey(string: email)**  *Post: return UserBean se esiste, null altrimenti*  **Context UserModelDS: doRetrieveAll(string: order)**  *Post: return Collection<UserBean> se esiste almeno uno, null altrimenti*  **Context UserModelDS: doUpdate(UserBean: user)**  *Post: return null, viene aggiornata la riga corrispettiva dell’user*  **Context UserModelDS: doDelete(UserBean: user)**  *Pre: return null, viene eliminata la riga all’interno della tabella Utente del corrispettivo user*  **Context UserModelDS: changeEmail(UserBean: user, string: newemail)**  *Post: return null, viene aggiornato il campo email della riga corrispettiva dell’user all’interno della tabella Utente*  **Context UserModelDS: changePassword (UserBean: user, string: newpassword)**  *Post: return null, viene aggiornato il campo password della riga corrispettiva dell’user all’interno della tabella Utente*  **Context UserModelDS: changeName(UserBean: user, string: newname)**  *Post: return null, viene aggiornato il campo nome della riga corrispettiva dell’user all’interno della tabella Utente*  **Context UserModelDS: changeSurname(UserBean: user, string: newsurname)**  *Post: return null, viene aggiornato il campo cognome della riga corrispettiva dell’user all’interno della tabella Utente*  **Context UserModelDS: changeAddress(UserBean: user, string: newaddress)**  *Post: return null, viene aggiornato il campo indirizzo della riga corrispettiva dell’user all’interno della tabella Utente*  **Context UserModelDS: changeCity(UserBean: user, string: newcity)**  *Post: return null, viene aggiornato il campo città della riga corrispettiva dell’user all’interno della tabella Utente* |
| *Invarianti* |  |

|  |  |
| --- | --- |
| LoginModelDS | |
| *Descrizione* | Questa classe è un manager che si occupa di interagire con il database e permette di interrogare la tabella Utente |
| *Pre-condizione* | **Context LoginModelDS: doRetrieveByKey(string: email, string: password)**  *Pre: email != null, password != null* |
| *Post-condizione* | **Context LoginModelDS: doRetrieveByKey(string: email, string: password)** *Post: return UserBean se esiste, null altrimenti* |
| *Invarianti* |  |

**C(2) | Ordine**

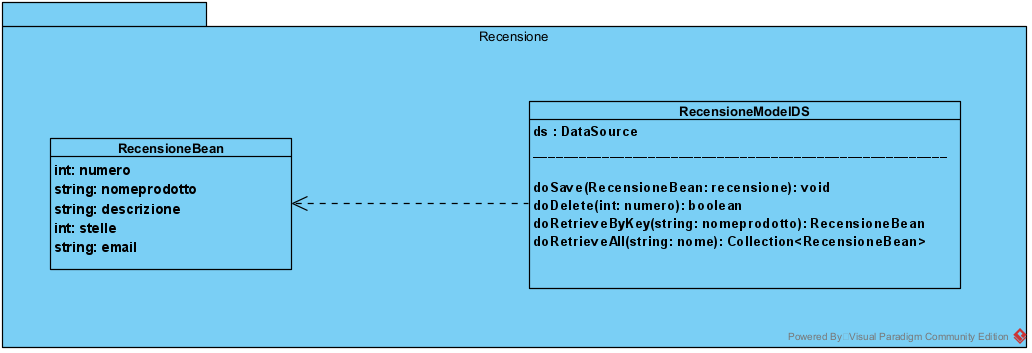
****

* **OrdineBean –** rappresenta l’entità Ordine all’interno del database
* **OrdineDettaglioBean –** rappresenta l’entità Ordine Dettaglio all’interno del databaseù

|  |  |
| --- | --- |
| OrdineModelDS | |
| *Descrizione* | Questa classe è un manager che si occupa di interagire con il database e permette di interrogare la tabella Ordine |
| *Pre-condizione* | **Context OrdineModelDS doSave(OrdineBan: ordine)**  *Pre: ordine != null*  **Context OrdineModelDS doDelete(int: id)**  *Pre: id != null*  **Context OrdineModelDS doRetrieveAll(string: email)**  *Pre: email != null*  **Context OrdineModelDS doUpdateDelivery(int: id, string: spedizione)**  *Pre: id != null, spedizione != null*  **Context OrdineModelDS doUpdateOrder(int: id, string: ordine)**  *Pre: id != null, ordine != null* |
| *Post-condizione* | **Context OrdineModelDS doSave(OrdineBan: ordine)**  *Post: return null, viene istanziata una nuova riga all’interno della tabella Ordine con gli attributi derivanti dall’oggetto ordine*  **Context OrdineModelDS doDelete(int: id)**  *Post: return true se viene trovato l’ordine con l’id rispettivo e viene eliminato, false altrimenti*  **Context OrdineModelDS doRetrieveAll(string: email)**  *Post: restituisce la lista degli ordini effettuati da quell’utente*  **Context OrdineModelDS doUpdateDelivery(int: id, string: spedizione)**  *Post: return null, viene aggiornato il campo spedizione della riga dell’ordine con quel determinato id*  **Context OrdineModelDS doUpdateOrder(int: id, string: ordine)**  *Post: return null, viene aggiornato il campo ordine della riga dell’ordine con quel determinato id* |
| *Invarianti* |  |

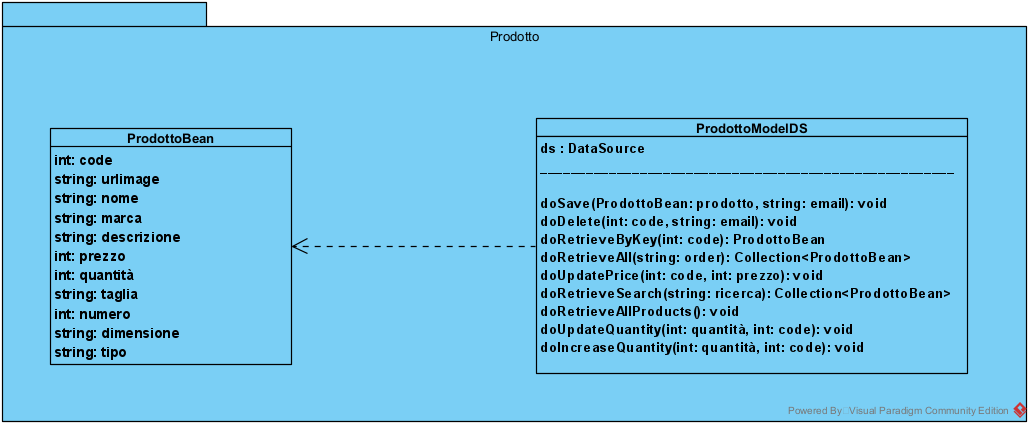
|  |  |
| --- | --- |
| OrdineDettaglioModelDS | |
| *Descrizione* | Questa classe è un manager che si occupa di interagire con il database e permette di interrogare la tabella Ordine Dettaglio |
| *Pre-condizione* | **Context OrdineDettaglioModelDS doSave(OrdineDettaglioBean prodotto)**  *Pre: prodotto != null*  **Context OrdineDettaglioModelDS doRetrieveByKey(int: id)**  *Pre: id != null* |
| *Post-condizione* | **Context OrdineDettaglioModelDS doSave(OrdineDettaglioBean prodotto)**  *Post: return null, viene istanziata una nuova riga all’interno della tabella Ordine Dettaglio con gli attributi relativi all’oggetto prodotto*  **Context OrdineDettaglioModelDS doRetrieveByKey(int: id)**  *Pre: restituisce la lista dei prodotti che hanno quel determinato id passato come parametro, null altrimenti* |
| *Invarianti* |  |

**C(3) | Recensione**

****

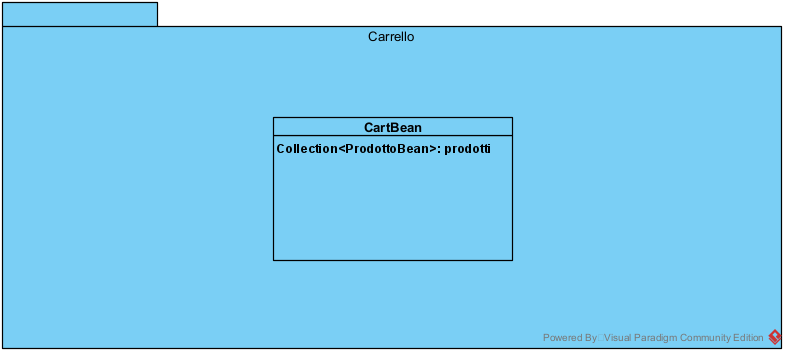
* **RecensioneBean –** rappresenta l’entità Recensione all’interno del database

|  |  |
| --- | --- |
| RecensioneModelDS | |
| *Descrizione* | Questa classe è un manager che si occupa di interagire con il database e permette di interrogare la tabella Recensione |
| *Pre-condizione* | **Context RecensioneModelDS doSave(RecensioneBean recensione)**  *Pre: recensione != null*  **Context RecensioneModelDS doDelete(int id)**  *Pre: id != null*  **Context RecensioneModelDS doRetrieveByKey(string: nomeprodotto)**  *Pre: nomeprodotto != null*  **Context RecensioneModelDS doRetrieveAll(string: nome)**  *Pre: nome != null* |
| *Post-condizione* | **Context RecensioneModelDS doSave(RecensioneBean recensione)**  *Post: return null, viene istanziata una nuova riga all’interno della tabella Recensione con gli attributi corrispettivi dell’oggetto recensione*  **Context RecensioneModelDS doDelete(int id)**  *Post: return null, elimina la recensione che ha come id lo stesso passato come parametro, null altrimenti*  **Context RecensioneModelDS doRetrieveByKey(string: nomeprodotto)**  *Post: return RecensioneBean che ha come parametro quel nome del prodotto*  **Context RecensioneModelDS doRetrieveAll(string: nome)**  *Post: restituisce la lista di recensioni per quel prodotto con quel determinato nome, null altrimenti* |
| *Invarianti* |  |

**C(4) | Prodotto**

|  |  |
| --- | --- |
| ProdottoModelDS | |
| *Descrizione* | Questa classe è un manager che si occupa di interagire con il database e permette di interrogare la tabella Prodotto |
| *Pre-condizione* | **Context ProdottoModelDS doSave(ProdottoBean: prodotto, string: email)**  *Pre: prodotto != null, email != null*  **Context ProdottoModelDS doDelete(int: code, string: email)**  *Pre: code != null, email != null*  **Context ProdottoModelDS doRetrieveByKey(int: code)**  *Pre: code != null*  **Context ProdottoModelDS doRetrieveAll(string: order)**  *Pre: order != null*  **Context ProdottoModelDS doUpdatePrice(int: code, int: prezzo)**  *Pre: code != null, prezzo != null*  **Context ProdottoModelDS doRetrieveSearch(string: ricerca)**  *Pre: ricerca != null*  **Context ProdottoModelDS doRetrieveAllProducts()**  *Pre: //*  **Context ProdottoModelDS doUpdateQuantity(int: quantità, int: code)**  *Pre: quantita != null, code != null*  **Context ProdottoModelDS doIncreaseQuantity(int: quantità, int: code)**  *Pre: quantita != null, code != null* |
| *Post-condizione* | **Context ProdottoModelDS doSave(ProdottoBean: prodotto, string: email)**  *Post: return null, viene istanziata una nuova riga all’interno della tabella Prodotto e associata all’email di chi l’ha aggiunta, con gli attributi corrispettivi dell’oggetto prodotto*  **Context ProdottoModelDS doDelete(int: code, string: email)**  *Post: return null, viene eliminata la riga corrispettiva del prodotto con quel determinato code*  **Context ProdottoModelDS doRetrieveByKey(int: code)**  *Post: return ProdottoBean con quel code, null altrimenti*  **Context ProdottoModelDS doRetrieveAll(string: order)**  *Post: restituisce la lista dei prodotti ordinati per la stringa passata come parametro, null altrimenti*  **Context ProdottoModelDS doUpdatePrice(int: code, int: prezzo)**  *Post: return null, viene aggiornato il campo prezzo della riga relativa al prodotto con quel determinato code*  **Context ProdottoModelDS doRetrieveSearch(string: ricerca)**  *Post: ritorna la lista dei prodotti risultanti dalla stringa di ricerca passata come parametro, null altrimenti*  **Context ProdottoModelDS doRetrieveAllProducts()**  *Post: ritorna la lista di tutti i prodotti presenti sulla piattaforma*  **Context ProdottoModelDS doUpdateQuantity(int: quantità, int: code)**  *Post: return null, diminuisce la quantità di una certa somma passata come parametro del prodotto che ha come code quello passato per parametro*  **Context ProdottoModelDS doIncreaseQuantity(int: quantità, int: code)**  *Post: return null, aumenta la quantità di una certa somma passata come parametro del prodotto che ha come code quello passato per parametro* |
| *Invarianti* |  |

**C(5) | Carrello**

****

* **CartBean –** rappresenta l’entità Carrello, questa, essendo volatile poiché istanziata solo quando l’utente effettua la procedura di login e assegnata a quest’ultimo mediante sessione, non rispetta il modello DAO poiché non c’è bisogno di persistere i dati relativi al carrello.

**3.** Design Pattern

Nel sistema realizzato viene utilizzato un pattern DAO, esso è usato per separare la logica di business dalla logica di accesso ai dati. Infatti, i componenti della logica di business non dovrebbero mai accedere direttamente al database, questo perché, sarebbe sintomo di scarsa manutenibilità.