Object Design Document

## Introduzione

1.1 Object Design Trade-offs

Questa sezione descrive i compromessi generali fatti dagli sviluppatori. I trade-offs individuati sono i seguenti.

Funzionalità vs Usabilità: Trattandosi di un sito di e-commerce si preferisce dare priorità all’usabilità rispetto alla funzionalità

Affidabilità vs Costo: Nonostante risulti essere più dispendioso, si preferisce utilizzare meccanismi di controllo dell’inserimento dei dati in input al fine di evitare errori

Prestazioni vs Modificabilità: Trattandosi di una piattaforma web si preferisce dare priorità alla performance piuttosto che decomporre il sistema a grana fine

Buy vs Build: il nostro sistema si concretizza in una web-app, per la sua realizzazione faremo uso delle numerose librerie e framework, off-the-shelf, sfruttando anche i principi del riuso.

Le componenti off-the-shelf sono state selezionate sulla base sia delle funzionalità offerte da tali tecnologie, sia delle conoscenze pregresse dei membri del team al fine di evitare lo spreco di risorse temporali per familiarizzare con una nuova tecnologia sconosciuta.

1.1.2 Componenti off-the-shelf

Per il nostro Sistema utilizzeremo componenti off-the-shelf, che sono componenti software già disponibili per facilitare la creazione del software. Useremo tools basati su HTML e CSS per le icone ed i font.

Per ottenere alcuni effetti visuali nonché́ velocizzare il sistema utilizzeremo inoltre JQuery e Javascript che permetteranno all’interfaccia di rispondere alle azioni dell’utente agilmente, e un miglioramento dell’esperienza con le tecniche basate su Ajax.

1.1.3 Design Pattern

Per realizzare il riuso si è scelto di utilizzare soluzioni a problemi ricorrenti, queste soluzioni sono realizzate dai seguenti design patterns.

Nome: Singleton

Descrizione Patern: Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Descrizione del problema: nel nostro caso vogliamo avere una sola istanza della connessione al database

1.2 Linee guida per la documentazione dell’interfaccia

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Regola per la Nomenclatura** | **Esempi** |
| **Package** | I nomi dei package devono iniziare con una lettera maiuscola e tutte minuscole  I nomi dei package non devono contenere spazi. L’unico carattere speciale permesso è “\_” | View  Model  Control  Gestione\_Utente |
| **Classi** | Le classi sono denominate con nomi singolari.  Le classi relative ai Service devono avere la nomenclatura: EntitàService  Le classi relative ai Control devono avere la nomenclature: SostativoControl  Le classi relative ai DAO devono avere la nomenclatura: EntitàDAO  Le classi relative ai DTO devono avere la nomenclatura: EntitàDTO | ProdottoService  OrdineControl  ProdottoDAO  ProdottoDTO |
| **JSP** | I documenti JSP devono avere la nomenclatura: SostantivoView | LoginView |
| **Metodi** | I metodi sono denominati con frasi verbali, campi e parametri con frasi nominali. | saveOrdine() |
| **Eccezioni** | Ogni eccezione deve avere nome esplicativo del problema segnalato | ProdottoNonDisponibile |

1.3 Definizioni, acronimi e abbreviazioni

N/A

1.4 Riferimenti

Requisiti non funzionali: 2.3 del RAD

## Pacchetti

2.1 Divisione in pacchetti

Il sistema sarà suddiviso in 3 package principali:

* Model
* View
* Control

L’utilizzo del pattern architetturale Model-View-Control è stato deciso nella fase di System Design.

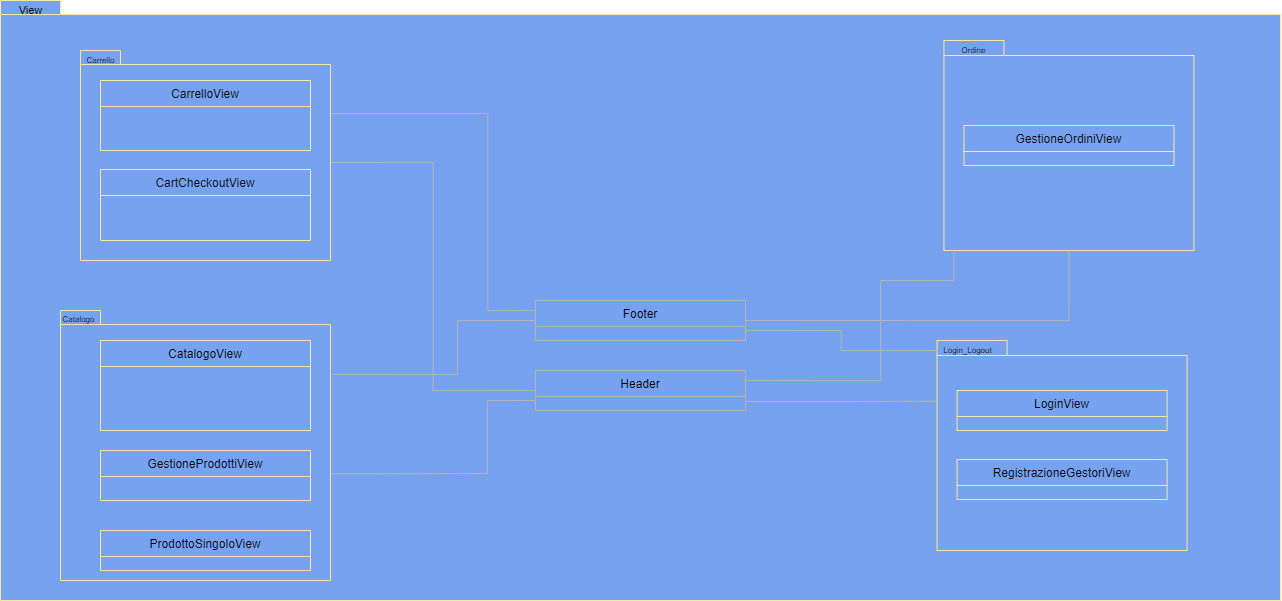
|  |  |
| --- | --- |
| Main Package | Control |
| Descrizione | Contiene i package che si occupano del controllo della logica di business e al fine di produrre i risultati da inoltrare al Layer di Presentazione |
|  |  |
| Nome Package | Product\_Manager |
| Descrizione | Contiene le Servlet che permettono di: visualizzare, inserire, aggiornare e rimuovere un prodotto dal catalogo; visualizzare i prodotti in base alla categoria; inserire un prodotto nel carrello; cancellare un prodotto dal carrello; modificare la quantità di un prodotto nel carrello. |
| Dipendenze | Services |
|  |  |
| Nome Package | Order\_Manager |
| Descrizione | Contiene le Servlet che permettono la visualizzazione dello storico degli ordini |
| Dipendenze | Services |
|  |  |
| Nome Package | User\_Manager |
| Descrizione | Contiene le Servlet che permettono la registrazione di utenti e gestori, permettono di gestire il login e il logout e la gestione dei dati del cliente |
| Dipendenze | Services |
|  |  |

|  |  |
| --- | --- |
| Main Package | Model |
| Descrizione | Contiene i package che gestiscono i dati persistenti e la logica di business |
|  |  |
| Nome Package | DAO |
| Descrizione | Contiene le classi che gestiscono i dati persistenti |
| Dipendenze | N/A |
|  |  |
| Nome Package | Services |
| Descrizione | Contiene le classi che si occupano della logica di business |
| Dipendenze | DAO |
|  |  |
| Nome Package | DTO |
| Descrizione | Contiene i DTO che rappresentano le entità persistenti |
| Dipendenze | N/A |

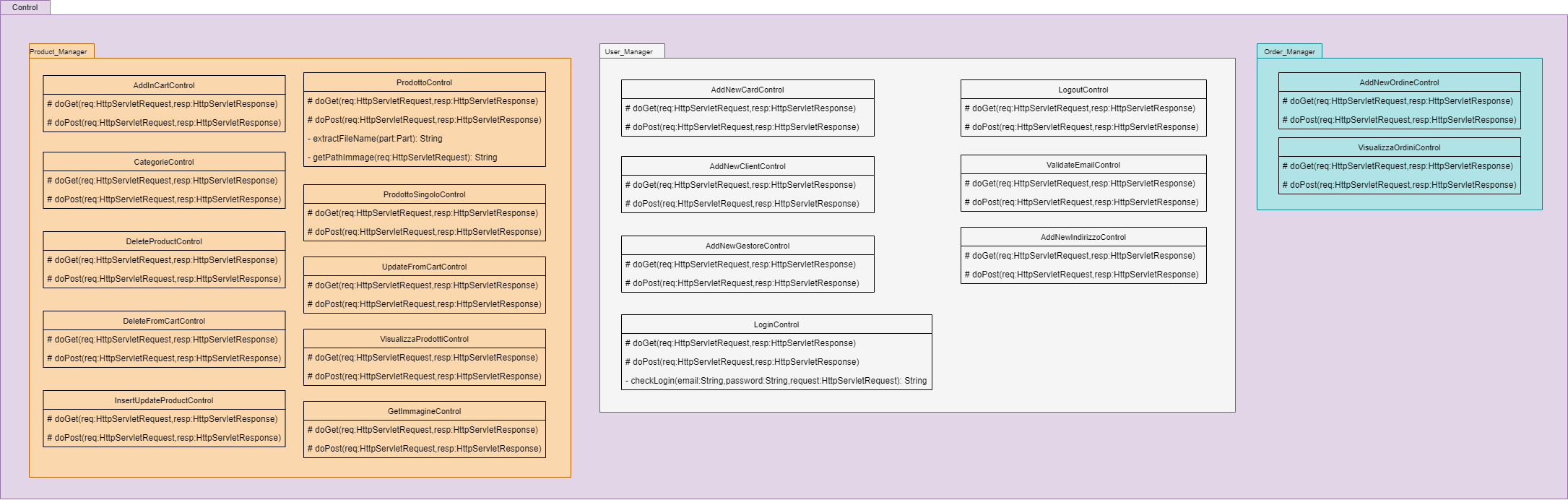
|  |  |
| --- | --- |
| Main Package | View |
| Descrizione | Contiene i package che si occupano dell’interazione attraverso componenti grafiche che rappresentano le funzionalità del sistema |
|  |  |
| Nome Package | Catalogo |
| Descrizione | Contiene le JSP che raggruppa l’interfaccia grafica per la visualizzazione e la ricerca dei prodotti e per la gestione del catalogo |
| Dipendenze | Product\_Manager |
|  |  |
| Nome Package | Carrello |
| Descrizione | Contiene le JSP che raggruppa l’interfaccia grafica per l’inserimento e la gestione dei prodotti nel carrello |
| Dipendenze | Product\_Manager |
|  |  |
| Nome Package | Ordine |
| Descrizione | Contiene le JSP che raggruppa l’interfaccia grafica per la visualizzazione degli ordini complessivi |
| Dipendenza | Order\_Manager |
|  |  |
| Nome Package | Login\_Logout |
| Descrizione | Contiene le JSP che raggruppa l’interfaccia grafica per: login/logout, registrazione di un cliente o gestore |
| Dipendenze | User\_Manager |

2.2 Organizzazione del codice in file

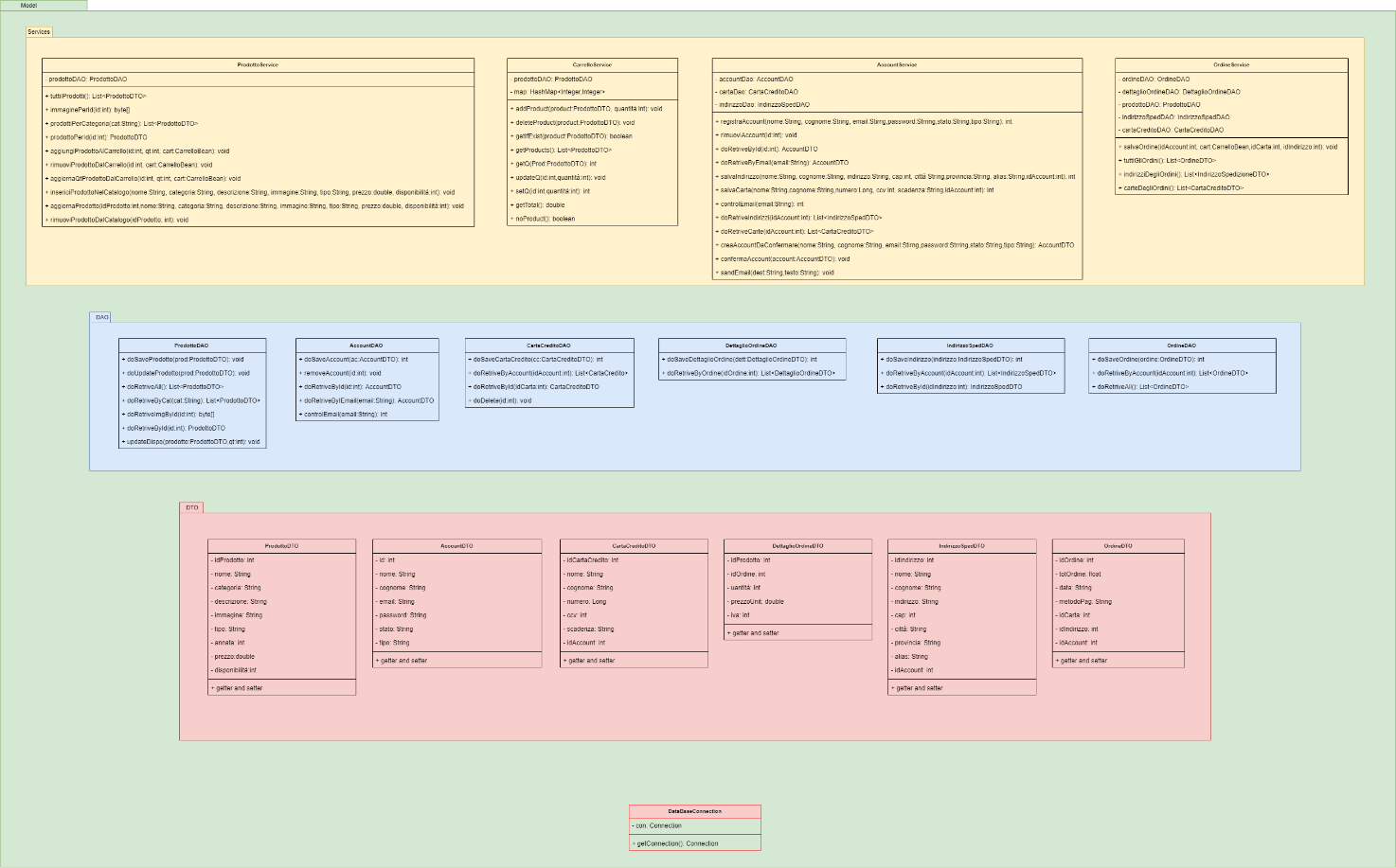
**2.2.1 View**



**2.2.2 Control**



**2.2.3 Model**



## Interfacce delle classi

**3.1 AccountService**

|  |  |
| --- | --- |
| Nome Classe | AccountService |
| Descrizione | Classe che si occupa della logica di business dedicata a tutto ciò che riguarda le operazioni effettuabili per la manipolazione degli account |
| Invarianti | Context AccountService **inv**: model<>null and modelInd<>null and modelCar<>null |
| METODI | |
| Metodo | public int registraAccount(String nome, String cognome, String email, String password, String stato, String tipo) |
| Descrizione | Metodo utilizzato per salvare un account nel database. Ritorna la chiave primaria dell’account salvato |
| Pre-condizione | nome!=null and cognome!=null and email!=null and password!=null and stato !=null tipo!=null |
| Post-condizione | Context AccountService:: registraAccount(String nome, String cognome, String email, String password, String stato, String tipo) **post:**  doRetriveById(valore di ritorno:id)=acccountDTO |
|  | |
| Metodo | public void removeAcount(int id) |
| Descrizione | Metodo utilizzato per eliminare un account dal database |
| Pre-condizione | Context AccountService::removeAccount(id) **pre:**  doRetriveById(id) <> null |
| Post-condizione | Context AccountService::removeAccount(id) **post:**  doRetriveById(id) = null |
|  | |
| Metodo | public AccountDTO doRetribeById(int id) |
| Descrizione | Metodo utilizzato per prelevare un account dal database tramite il su id |
| Pre-condizione | Context AccountService::doRetriveById(id) **pre:**  id>=0 |
| Post-condizione | Context AccountService::doRetriveById(id) **post:**  AccountDTO <> null se esiste un account associato a quell’id nel database |
|  | |
| Metodo | public AccountDTO doRetriveByEmail(String email) |
| Descrizione | Metodo utilizzato per prelevare un account dal database tramite la sua email |
| Pre-condizione | Context AccountService::doRetriveByEmail(email) **pre:**  email <> null |
| Post-condizione | Context AccountService::doRetriveByEmail(email) **post:**  AccountDTO <> null se esiste un account associato a quell’email nel database |
|  | |
| Metodo | public int salvaIndirizzo(String nome,String cognome,String indirizzo,int cap,String città,String provincia,String alias, int idAccount) |
| Descrizione | Metodo utilizzato per salvare nel database un indirizzo di spedizione associato ad un determinato account. Ritorna la chiave primaria dell’indirizzo |
| Pre-condizione | Context AccountService::salvaIndirizzo(nome,cognome,indirizzo,cap,città,provincia, alias,idAccount) **pre:**  nome <>null and cognome<> null and indirizzo<>null and cap>=0 and città <> null and provincia <> null and alias <> null and doRetriveById(idAccount)<>null |
| Post-condizione | Context AccountService:: salvaIndirizzo(nome,cognome,indirizzo,cap,città,provincia, alias,idAccount) **post:**  doRetriveIndirizzi(idAccount).contains(indirizzo salvato) = true |
|  | |
| Metodo | public int salvaCarta(String nome, String cognome, Long numero, int ccv, Sring scadenza, int idAccount) |
| Descrizione | Metodo utilizzato per salvare nel database una carta di credito associata ad un determinato account. Ritorna la chiave primaria della carta |
| Pre-condizione | Context AccountService:: salvaCarta(nome, cognome, numero; ccv, scadenza, idAccount) **pre:**  nome<>null and cognome<>null and numero>0 and ccv>0 and scadeza <> null and doRetriveById(idAccount)<>null |
| Post-condizione | Context AccountService::doSaveCarta(carta,idAccount) **post:**  doRetriveCarte(idAccount).contains(carta salvata) = true |
|  | |
| Metodo | public int controlEmail(String email) |
| Descrizione | Metodo utilizzato per controllare se esistono account associati a quell’email. Ritorna 1 se esiste un account associato, altrimenti ritorna 0 |
| Pre-condizione | Context AccountService::controlEmail(email) **pre:**  email <> null |
| Post-condizione |  |
|  | |
| Metodo | public List<IndirizzoSpedDTO> doRetriveIndirizzi(int idAccount) |
| Descrizione | Metodo che restituisce tutti gli indirizzi di spedizione associati ad un account |
| Pre-condizione | Conext AccountService::doRetriveIndirizzi(idAccount) **pre:**  doRetriveById(idAccount)<>null |
| Post-condizione |  |
|  | |
| Metodo | public List<CartaCreditoDTO> doRetriveCarte(int idAccount) |
| Descrizione | Metodo che restituisce tutte le carte di credito associate ad un account |
| Pre-condizione | Conext AccountService::doRetriveCarte(idAccount) **pre:**  doRetriveById(idAccount)<>null |
| Post-condizione |  |
|  | |
| Metodo | public AccountDTO creaAccounDaConfermare(String nome, String cognome, String email, String password,String stato,String tipo) |
| Descrizione | Metodo utilizzato per creare un account temporaneo, da utilizzare nella sessione, in attesa che l’utente confermi l’account tramite mail |
| Pre-condizione | nome!=null and cognome!=null and email!=null and password!=null and stato =.equals(“non confermato”) and tipo!=null |
| Post-condizione |  |
|  | |
| Metodo | public void confermaAccout(AccountDTO account) |
| Descrizione | Metodo per confermare l’account |
| Pre-condizione | Conext AccountService::confermaAccout(account) **pre:**  account<>null |
| Post-condizione | Conext AccountService::confermaAccout(account) **post:**  account.getStato().equals(“confermato”) |
|  | |
| Metodo | public sandEmail(String desti, String testo) |
| Descrizione | Metodo per inviare un email al cliente per far confermare il proprio account |
| Pre-condizione | Conext AccountService::sandEmail(dest,testo) **pre:**  dest<>null and testo<>null |
| Post-condizione |  |

**3.2 carrelloService**

|  |  |
| --- | --- |
| Nome Classe | CarrelloService |
| Descrizione | Classe che si occupa della logica di business dedicata alla gestione del carrello |
| Invarianti | Context CarrelloService **inv**: prodottoDAO <> null ad map <>null |
| METODI | |
| Metodo | public int addProduct(ProdottoDTO product, int quantità) |
| Descrizione | Metodo utilizzato per aggiungere un prodotto al carrello |
| Pre-condizione | Conext CarrelloService::addProduct(product,quantità) **pre:**  Product <> null |
| Post-condizione | Conext CarrelloService::addProduct(product,quantità) **post:**  map.containsKey(product.getIdProdotto) = true |
|  | |
| Metodo | public void deleteProduct(ProdottoDTO product) |
| Descrizione | Metodo utilizzato per rimuovere un prodotto dal carrello |
| Pre-condizione | Conext CarrelloService:: deleteProduct (product) **pre:**  product <> null and getIfExist(product)=true |
| Post-condizione | Conext CarrelloService:: deleteProduct (product) **post:**  getIfExist(product) = false |
|  | |
| Metodo | public boolean getIfExists(ProdottoDTO product) |
| Descrizione | Metodo utilizzato per controllare se un prodotto è presente nel carrello. Ritorna true se il prodotto è presente |
| Pre-condizione | Conext CarrelloService::getIfExists(product) **pre:**  product <> null |
| Post-condizione |  |
|  | |
| Metodo | public List<ProdottoDTO> getProducts() |
| Descrizione | Metodo che restituisce la lista dei prodotti nel carrello |
| Pre-condizione |  |
| Post-condizione |  |
|  | |
| Metodo | public int getQ(ProdottoDTO prod) |
| Descrizione | Metodo che restituisce la quantità di un determinato prodotto nel carrello, 0 se il prodotto non è presente nel carrello |
| Pre-condizione | Conext CarrelloService::getQ(prod) **pre:**  prod <> null |
| Post-condizione |  |
|  | |
| Metodo | public void updateQ(int id,int quantità) |
| Descrizione | Metodo che aggiorna la quantità di un determinato prodotto nel carrello |
| Pre-condizione | Conext CarrelloService::updateQ(id,quantità) **pre:**  getIfExist(prodottoDao.prodottoPerId(id)) = true and quantità<= prodottoDao.prodottoPerId(id).getDisponibilità() |
| Post-condizione | Conext CarrelloService::updateQ(id,quantità) **post:**  getQ(prodottoDao.prodottoPerId(id)) = quantità |
|  | |
| Metodo | public void setQ(int id,int quantita) |
| Descrizione | Metodo utilizzato per aggiungere lo stesso prodotto nel carrello con una quantità maggiore |
| Pre-condizione | Conext CarrelloService::setQ(id,quantità) **pre:**  getIfExist(prodottoDao.prodottoPerId(id)) = true and (getQ(prodottoDao.prodottoPerId(id)) + quantità) <= prodottoDao.prodottoPerId(id).getDisponibilità() |
| Post-condizione | Conext CarrelloService::setQ(id,quantità) **post:**  getQ(prodottoDao.prodottoPerId(id)) = (**@Pre** getQ(prodottoDao.prodottoPerId(id)) + quantità) |
|  | |
| Metodo | public double getTotal() |
| Descrizione | Metodo utilizzato per calcolare il totale del carrello (Somma del prezzo di ogni prodotto \* quantità nel carrello) |
| Pre-condizione |  |
| Post-condizione |  |
|  | |
| Metodo | public boolean noProduct() |
| Descrizione | Metodo che restituisce true se non ci sono prodotti nel carrello e false altrimenti |
| Pre-condizione |  |
| Post-condizione |  |

**3.3 OrdineService**

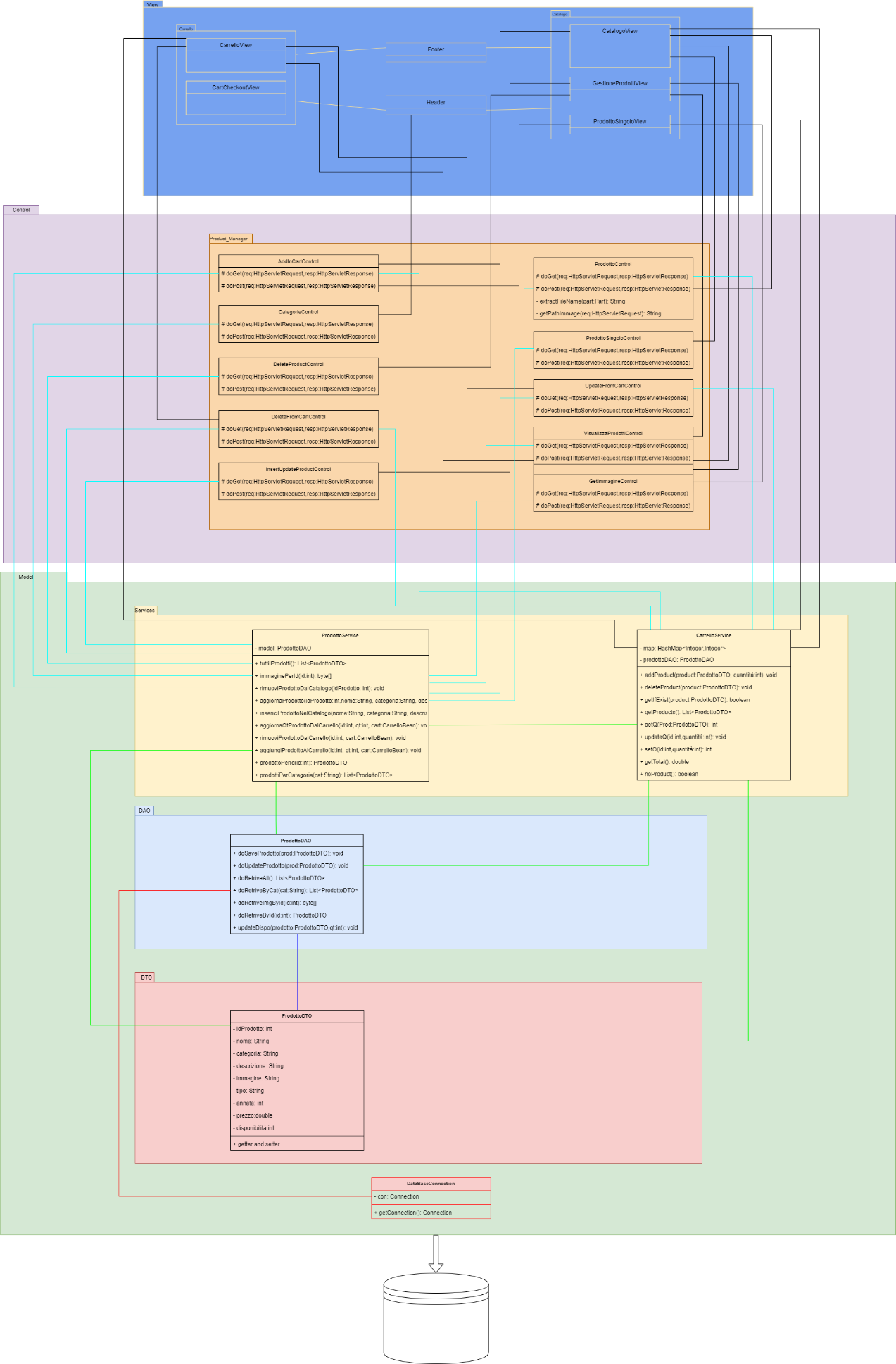
|  |  |
| --- | --- |
| Nome Classe | OrdineService |
| Descrizione | Classe che si occupa della logica di business dedicata a tutto ciò che riguarda le operazioni effettuabili per la manipolazione degli ordini |
| Invarianti | Context OrdineService **inv**: ordineDAO <> null ad dettaglioOrdineDAO <>null and prodottoDAO <> null and indirizzoSpedDAO <> null and cartaCreditoDao <> null |
| METODI | |
| Metodo | public int salvaOdine(int idAccount, CarrelloService cart, int idCarta, int idIndirizzo) |
| Descrizione | Metodo utilizzato per salvare un ordine e un dettaglio ordine nel database |
| Pre-condizione | Conext OrdnineService:: int salvaOdine( idAccount, cart, idCarta, idIndirizzo) **pre:**  accountDAO.doRetriveById(idAccount) <> null and cart <> null and cartaCreditoDAO.doRetriveById(idCarta) <> null and indirizzoSpedDAO.doRetriveById(idIndirizzo) <> null |
| Post-condizione | Conext OrdnineService:: int salvaOdine( idAccount, cart, idCarta, idIndirizzo) **post:**  ordineDAO.doRetriveById(valore di ritorno:id) <> null |
|  | |
| Metodo | public List<OrdineDTO> tuttiGliOrdini() |
| Descrizione | Metodo che ritorna la lista degli ordini presenti nel database |
| Pre-condizione |  |
| Post-condizione |  |
|  | |
| Metodo | public List<IndirizzoSpedDTO> indirizziDegliOrdini() |
| Descrizione | Metodo che ritorna la lista degli indirizzi di spedizione associate agli ordini |
| Pre-condizione |  |
| Post-condizione |  |
|  | |
| Metodo | public List<CartaCreditoDTO> carteDegliOrdini() |
| Descrizione | Metodo che ritorna la lista degli indirizzi di spedizione associate agli ordini |
| Pre-condizione |  |
| Post-condizione |  |

**3.4 ProdottoService**

|  |  |
| --- | --- |
| Nome Classe | ProdottoService |
| Descrizione | Classe che si occupa della logica di business dedicata a tutto ciò che riguarda le operazioni effettuabili per la manipolazione dei prodotti |
| Invarianti | Context OrdineService **inv**: prodottoDAO <> null |
| METODI | |
| Metodo | public List<ProdottoDTO> tuttiIProdotti() |
| Descrizione | Metodo che ritorna tutti i prodotti presenti nel database |
| Pre-condizione |  |
| Post-condizione |  |
|  | |
| Metodo | public byte[] immaginePerId(int id) |
| Descrizione | Metodo che ritorna l’immagine in byte associata al prodotto con idProdotto = id |
| Pre-condizione | Conext ProdottoService:: byte[] immaginePerId(id) **pre:**  prodottoPerId(id) <> null |
| Post-condizione |  |
|  | |
| Metodo | public List<ProdottoDTO> prodottiPerCategoria(String cat) |
| Descrizione | Metodo che ritorna la lista di prodotti di una determinata categoria |
| Pre-condizione | Conext ProdottoService:: List<ProdottoDTO> prodottiPerCategoria( cat) **pre:** cat.equals(“bi”) or cat.equals(“ro”) or cat.equals(“spu”) |
| Post-condizione |  |
|  | |
| Metodo | public ProdottoDTO prodottoPerId(int id) |
| Descrizione | Metodo che ritorna un prodotto con idProdotto = id |
| Pre-condizione | Conext ProdottoService:: ProdottoDTO prodottoPerId(id) **pre:** id>=0 |
| Post-condizione | Conext ProdottoService:: ProdottoDTO prodottoPerId(id) **post:**  prodotto ritornato <> null se l’id è associato a un prodotto |
|  | |
| Metodo | public void aggiungiProdottoAlCarrello(int id, int qt, CarrelloService cart) |
| Descrizione | Metodo per aggiungere un prodotto al carrello con una determinata quantità |
| Pre-condizione | Conext ProdottoService:: aggiungiProdottoAlCarrello(id, qt, cart) **pre:**  prodottoPerId(id) <> null and qt>=0 and cart <> null |
| Post-condizione | Conext ProdottoService:: aggiungiProdottoAlCarrello(id, qt, cart **post:**  CarrelloService.getIfExist(id) = true |
|  | |
| Metodo | public void rimuoviProdottoDalCarrello(int id, CarrelloService cart) |
| Descrizione | Metodo per rimuovere un prodotto dal carrello |
| Pre-condizione | Conext ProdottoService:: rimuoviProdottoDalCarrello(id, cart) **pre:**  prodottoPerId(id) <> null and CarrelloService.getIfExist(id) = true and cart <> null |
| Post-condizione | Conext ProdottoService:: rimuoviProdottoDalCarrello(id, cart) **post:**  CarrelloService.getIfExist(id) = false |
|  | |
| Metodo | public void aggiornaQtProdottoDalCarrello(int id, int qt, CarrelloService cart) |
| Descrizione | Metodo per aggiornare la quantità del prodotto nel carrello |
| Pre-condizione | Conext ProdottoService:: aggiornaQtProdottoDalCarrello(id, qt, cart)**pre:**  prodottoPerId(id) <> null and CarrelloService.getIfExist(id) = true and cart <> null |
| Post-condizione | Conext ProdottoService:: aggiornaQtProdottoDalCarrello(id, qt, cart)**post:**  CarrelloService.getQ(id)= (@pre CarrelloService.getQ(id) + qt) |
|  | |
| Metodo | public void inserisciProdottoNelCatalogo(String nome,String categoria, String descrizione,String immagine,String tipo,int annata,double prezzo,int disponibilità) |
| Descrizione | Metodo per inserire un nuovo prodotto nel catalogo |
| Pre-condizione | Conext ProdottoService:: inserisciProdottoNelCatalogo(nome, categoria, descrizione, immagine, tipo, annata, prezzo, disponibilità) **pre:**  nome <> null and categoria <> nulla and descrizione <> null and immagine <> null and tipo <> null and annata >=0 prezzo >= 0 and disponibilità >= 0 |
| Post-condizione | Conext ProdottoService:: inserisciProdottoNelCatalogo(nome, categoria, descrizione, immagine, tipo, annata, prezzo, disponibilità) **post:** |
|  | |
| Metodo | public void aggiornaProdotto(int idProdotto,String nome,String categoria, String descrizione,String immagine,String tipo,int annata,double prezzo,int disponibilità) |
| Descrizione | Metodo per aggiornare un prodotto nel catalogo |
| Pre-condizione | Conext ProdottoService:: inserisciProdottoNelCatalogo(idProdotto, nome, categoria, descrizione, immagine, tipo, annata, prezzo, disponibilità) **pre:**  prodottoPerId(idProdotto) <> null and nome <> null and categoria <> nulla and descrizione <> null and immagine <> null and tipo <> null and annata >=0 prezzo >= 0 and disponibilità >= 0 |
| Post-condizione |  |
|  | |
| Metodo | public void rimuoviProdottoDalCatalogo(int idProdotto) |
| Descrizione | Metodo utilizzato per rimuovere un prodotto dal catalogo |
| Pre-condizione | Context ProdottoService::rimuoviProdottoDalCatalogo(idProdotto) **pre**:  prodottoPerId(idProdotto) <> null |
| Post-condizione | Context ProdottoService::rimuoviProdottoDalCatalogo(idProdotto) **pre**:  prodottoPerId(idProdotto).getDisponibilità=0 |

## Class Diagram

Nella seguente immagine abbiamo un esempio di class diagram che coinvolge tutte le operazioni possibili sui prodotti.



## Glossario