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.

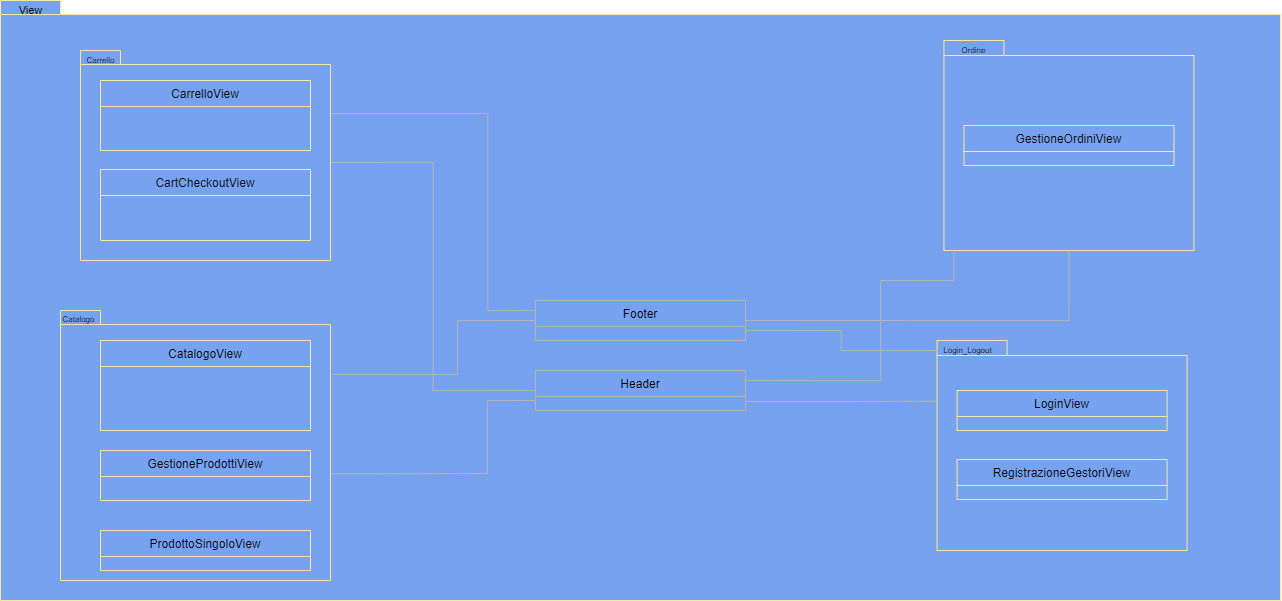
|  |  |
| --- | --- |
| Main Package | Control |
| Descrizione | Contiene i package che si occupano del controllo della logica di business e al fine di produrre i risultati da inoltrare al Layer di Presentazione |
|  |  |
| Nome Package | Product\_Manager |
| Descrizione | Contiene le Servlet che permettono 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 |

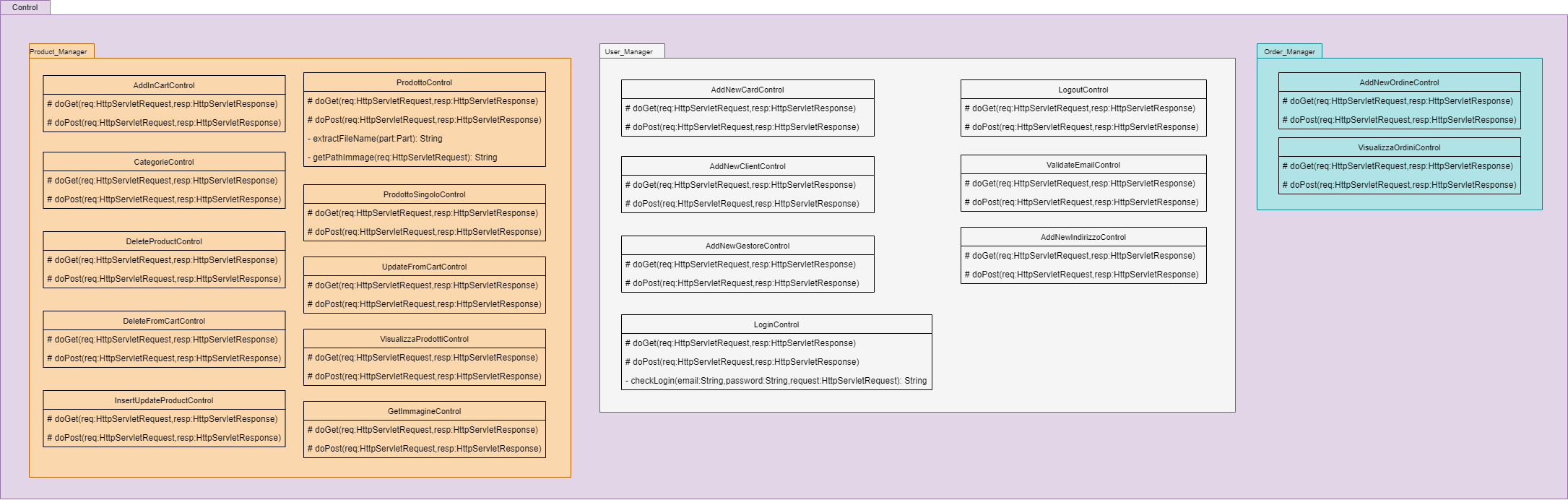
|  |  |
| --- | --- |
| Main Package | View |
| Descrizione | Contiene i package che si occupano dell’interazione attraverso componenti grafiche che rappresentano le funzionalità del sistema |
|  |  |
| Nome Package | Catalogo |
| Descrizione | Contiene le JSP che raggruppa l’interfaccia grafica per 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

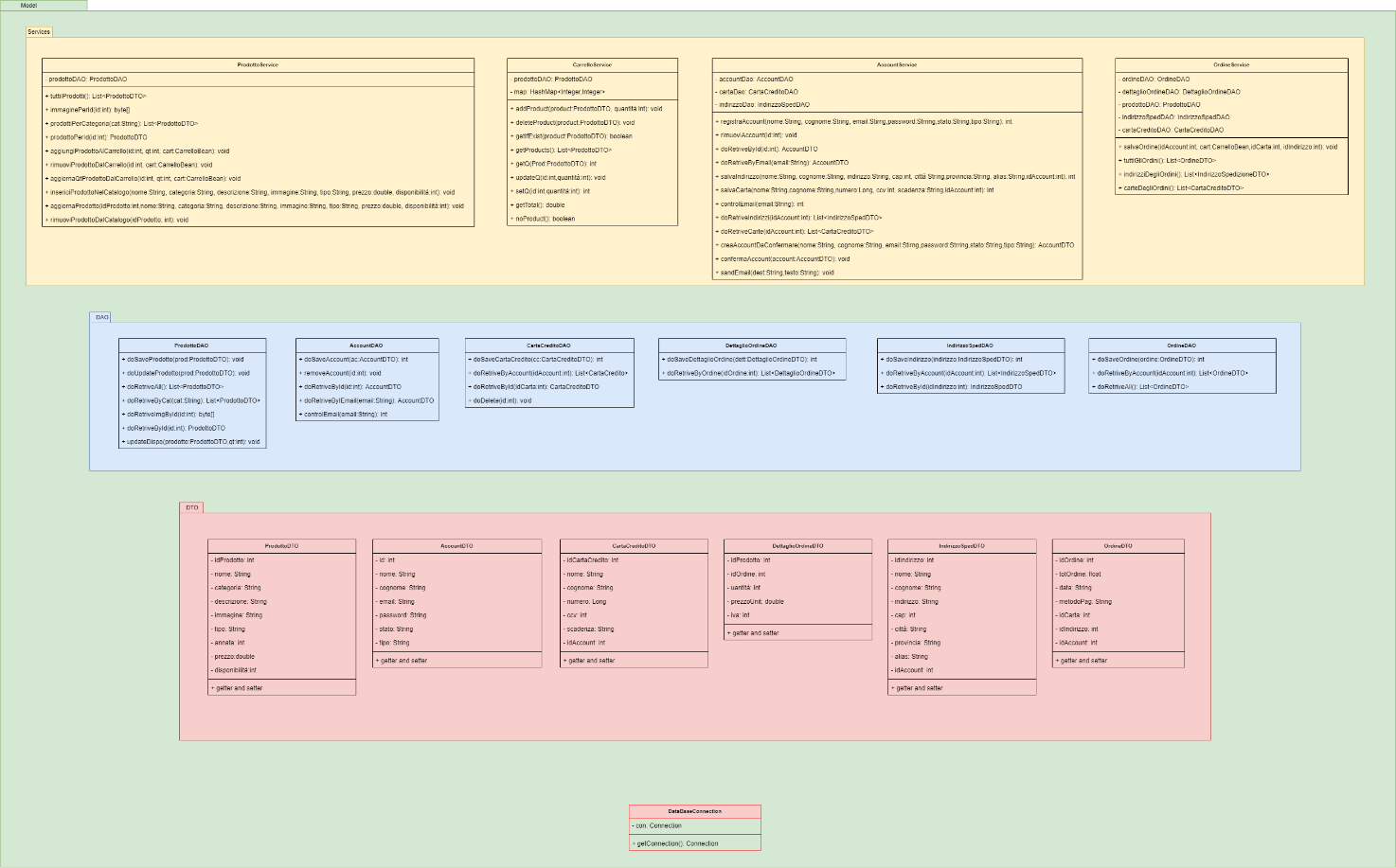
**2.2.1 View**



2.2.2 Control



2.2.3 Model



## Interfacce delle classi

|  |  |
| --- | --- |
| 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 AccountBean **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 AccountBean:: 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 AccountBean::removeAccount(id) **pre:**  doRetriveById(id) <> null |
| Post-condizione | Context AccountBean::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 AccountBean::doRetriveById(id) **pre:**  id>=0 |
| Post-condizione | Context AccountBean::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 AccountBean::doRetriveByEmail(email) **pre:**  email <> null |
| Post-condizione | Context AccountBean::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 AccountBean::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 AccountBean:: salvaIndirizzo(nome,cognome,indirizzo,cap,città,provincia, alias,idAccount) **post:**  doRetriveIndirizzi(idAccount).contains(indirizzo salvato) = true |
| **Continuare da qui** | |
| Metodo | public int doSaveCarta(CartaCreditoDTO carta, int idAccount) |
| Descrizione | Metodo utilizzato per salvare nel database una carta di credio associata ad un determinato account. Ritorna la chiave primaria della carta |
| Pre-condizione | Context AccountBean::doSaveCarta(carta,idAccount) **pre:**  carta<>null and doRetriveById(idAccount)<>null |
| Post-condizione | Context AccountBean::doSaveCarta(carta,idAccount) **post:**  doRetriveCarte(idAccount).contains(carta) = 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 AccountBean::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 AccountBean::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 AccountBean::doRetriveCarte(idAccount) **pre:**  doRetriveById(idAccount)<>null |
| Post-condizione |  |
|  | |
| Metodo | public void sandEmail(String dest,String testo) |
| Descrizione | Metodo per inviare un email al cliente per far confermare il proprio account |
| Pre-condizione | Conext AccountBean::sandEmail(dest,testo) **pre:**  dest<>null and testo<>null |
| Post-condizione |  |

|  |  |
| --- | --- |
| Nome Classe |  |
| Desrizione |  |
| Pre-condizione |  |
| Post-condizione |  |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome Classe |  |
| Desrizione |  |
| Pre-condizione |  |
| Post-condizione |  |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome Classe |  |
| Desrizione |  |
| Pre-condizione |  |
| Post-condizione |  |
| Invarianti |  |

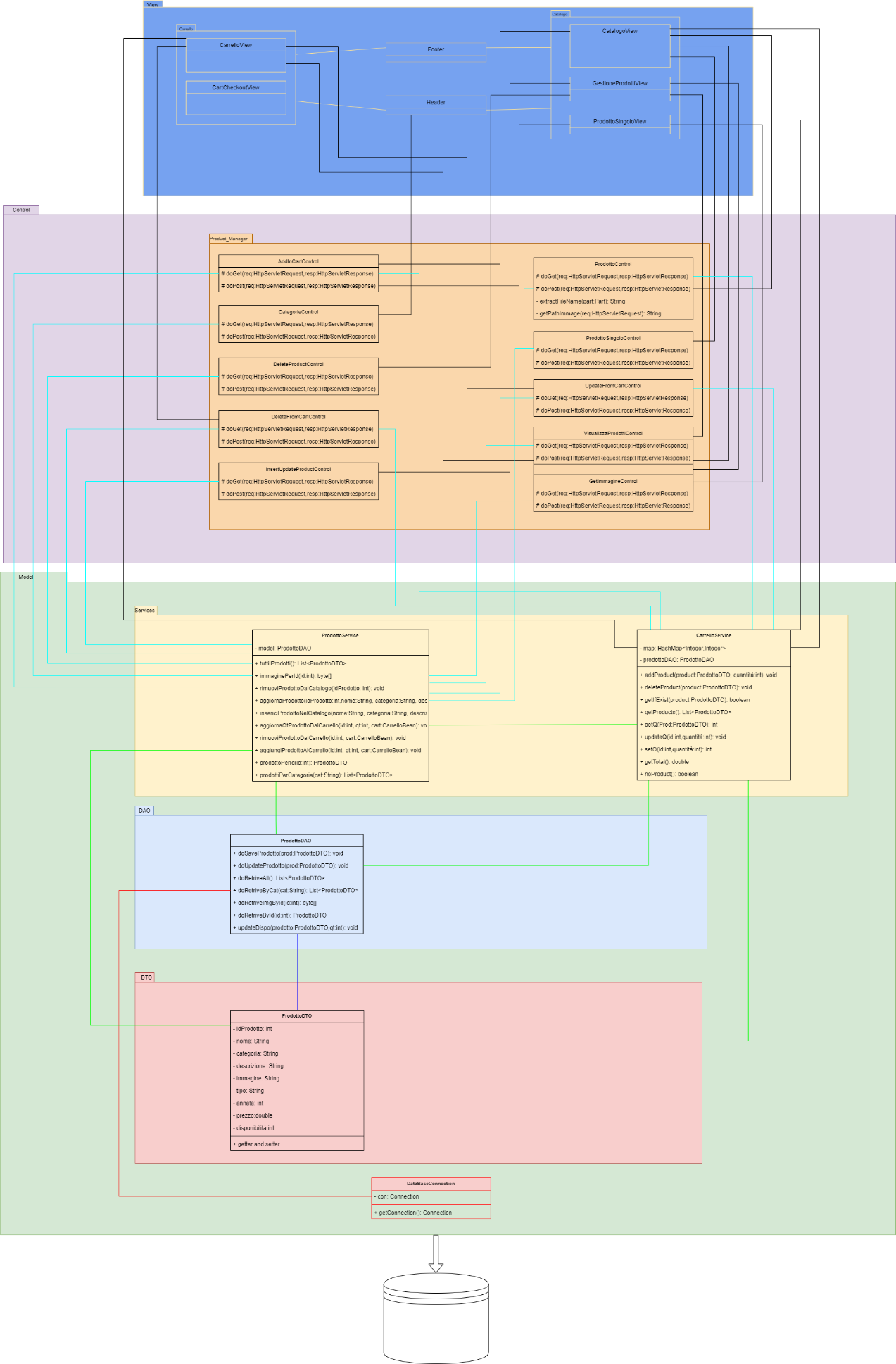
|  |  |
| --- | --- |
| Nome Classe |  |
| Desrizione |  |
| Pre-condizione |  |
| Post-condizione |  |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome Classe |  |
| Desrizione |  |
| Pre-condizione |  |
| Post-condizione |  |
| Invarianti |  |

|  |  |
| --- | --- |
| Nome Classe |  |
| Desrizione |  |
| Pre-condizione |  |
| Post-condizione |  |
| Invarianti |  |

## Class Diagram

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



## Glossario