

e-Sport

Object Design Document

1 Introduzione	1
1.1 Object design trade-offs	1
1.2 Linee guida per la documentazione delle interfacce	2
2 Package	4
2.1 Struttura sistema	4
2.2 Descrizione delle classi	5
2.2.1 Data Layer	5
2.2.1.1 Beans	5
2.2.1.2 Model	6
2.2.2 Application Layer	6
2.2.2.1 Controller	7
2.2.3 Presentation Layer	9
2.2.3.1 View	9
3 Interfacce delle classi	12
3.1 Definizione contratti	15
4 Design pattern	22

1 Introduzione

1.1 Object design trade-offs

Dopo aver stilato il Requirements Analysis Document e il System Design Document, in cui il sistema è descritto in modo sommario, definendo gli obiettivi ma tralasciando gli aspetti implementativi, si procede ora con lo stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti.

In particolar modo, nell'ODD si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti in fase di System Design. Inoltre sono specificati i trade-offs e le linee guida.

Comprensibilità vs Tempo:

Il codice del sistema deve essere il più comprensibile possibile in modo da facilitare la fase di testing e migliorare la supportabilità del sito per eventuali future modifiche da apportare.

Per rispettare queste linee guida il codice sarà accompagnato da log e commenti volti a semplificarne la comprensione. Ovviamente questo comporterà un aumento del tempo di sviluppo.

Prestazioni vs Costi:

Poiché il nostro progetto è sprovvisto di budget, per poter mantenere i costi e i tempi di sviluppo verranno utilizzati dei framework open source esterni, in particolare Bootstrap.

Interfaccia vs Usabilità:

L'interfaccia grafica è stata realizzata in modo da essere semplice e con un look minimale così da risultare chiara e concisa.

Verranno utilizzati form e pulsanti con lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:

La sicurezza, come descritto nei requisiti non funzionali del RAD, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username, password e ruolo degli utenti.

1.2 Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno seguire determinate linee guida nella stesura del codice.

1.2.1 Naming Convention

È buona norma utilizzare nomi:

- Descrittivi

- Pronunciabili
- Non abbreviati
- Utilizzando Camel Case
- Utilizzando solo caratteri consentiti

1.2.1.1 Variabili

- I nomi delle variabili devono iniziare con la lettera minuscola e le parole successive, che li compongono, con la lettera maiuscola. In ogni riga di codice vi deve essere una sola dichiarazione di variabile e va effettuato l'allineamento per migliorare la leggibilità.
- In caso di costanti statiche, utilizzare solo caratteri maiuscoli.
- In determinati casi, è possibile utilizzare il carattere underscore “_”, ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche.

1.2.1.2 Metodi

- I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive, che li compongono, con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un'azione, seguito dal nome di un oggetto.
- I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo `getNomeVariabile()` e `setNomeVariabile()`.
- Ai metodi va aggiunta una descrizione che deve essere posizionata prima della dichiarazione del metodo e che deve descriverne lo scopo, i parametri, il valore di ritorno ed eventualmente le eccezioni che può lanciare. I metodi devono essere raggruppati in base alla loro funzionalità.

1.2.1.3 Classi e Pagine (JSP/HTML)

- I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all'interno del nome devono iniziare con la lettera maiuscola. I nomi delle classi e delle pagine devono essere evocativi in modo da fornire informazioni sullo scopo di quest'ultime.
- Ogni file sorgente *.java deve contenere una singola classe e deve essere strutturato in un determinato modo:

1. Introduzione alla classe:

```
/**  
 * Astrazione modellata dalla classe  
 **/
```

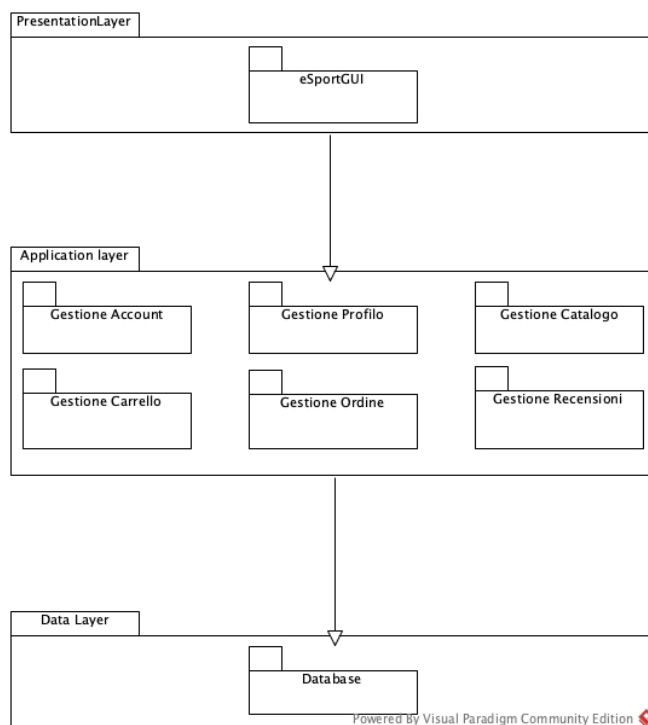
2. Dichiarazione della classe, costituita da:

- Dichiarazione della classe pubblica
- Dichiarazione delle costanti
- Dichiarazione delle variabili d'istanza
- Dichiarazione di variabili statiche
- Costruttore di default
- Dichiarazione metodi che definiscono il comportamento della classe

2 Package

La struttura del sistema è modellata secondo una divisione in package e sotto-package che raggruppano le classi che hanno il compito di gestirne la logica in base alle richieste dell'utente.

2.1 Struttura sistema

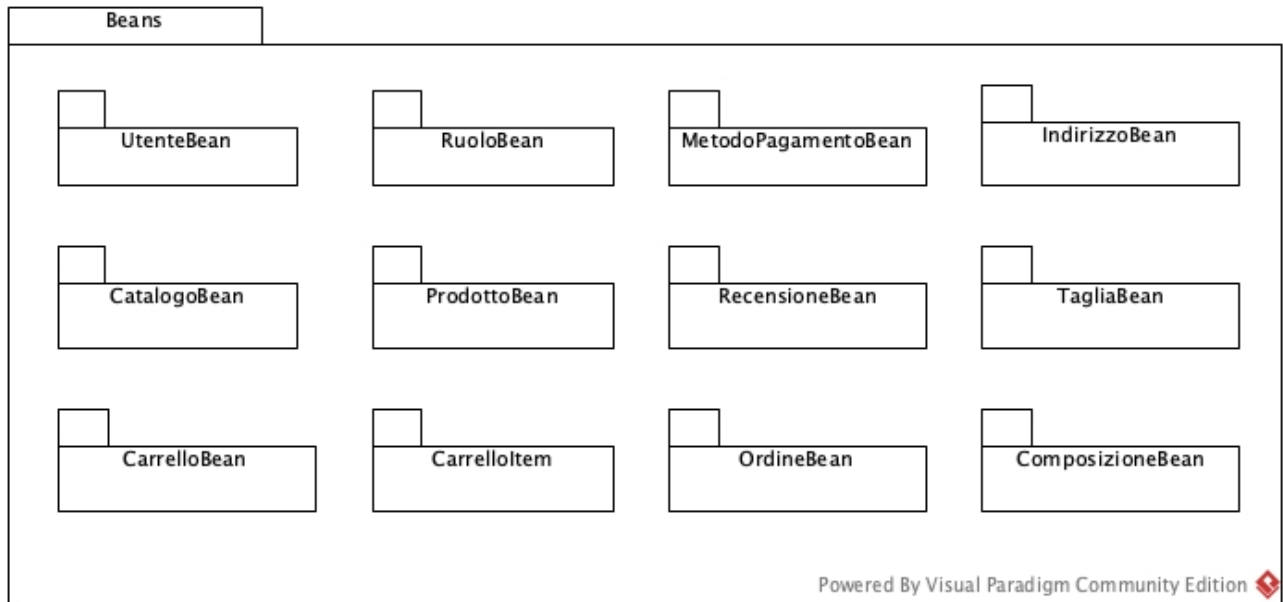


2.2 Descrizione delle classi

2.2.1 Data Layer

2.2.1.1 Beans

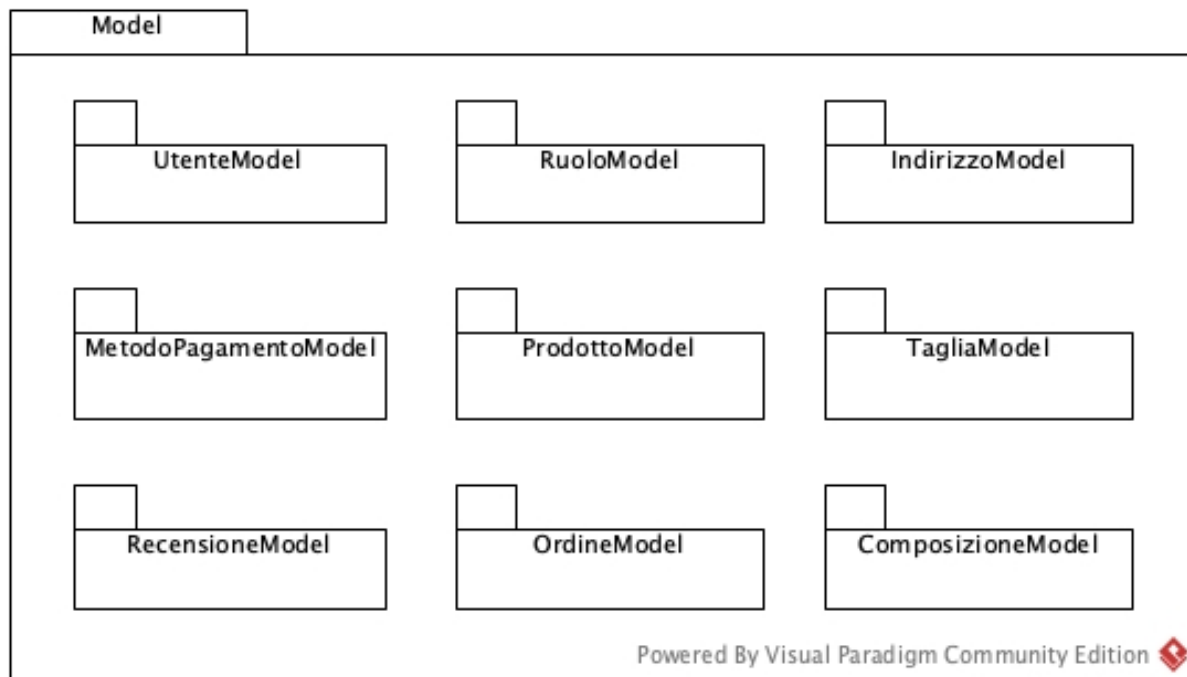
Contiene le classi che modellano le entità del sistema.



Classe	Descrizione
UtenteBean	Modella un utente registrato
RuoloBean	Modella un ruolo dell'utente
MetodoPagamentoBean	Modella un metodo di pagamento dell'utente
IndirizzoBean	Modella un indirizzo dell'utente
CatalogoBean	Modella il catalogo
ProdottoBean	Modella un prodotto
RecensioneBean	Modella una recensione di un prodotto
TagliaBean	Modella una taglia di un prodotto
CarrelloBean	Modella il carrello
CarrelloItem	Modella un elemento del carrello
OrdineBean	Modella un ordine
ComposizioneBean	Modella la composizione di un ordine

2.2.1.2 Model

Contiene le classi che modellano la conoscenza sulle entità del sistema.

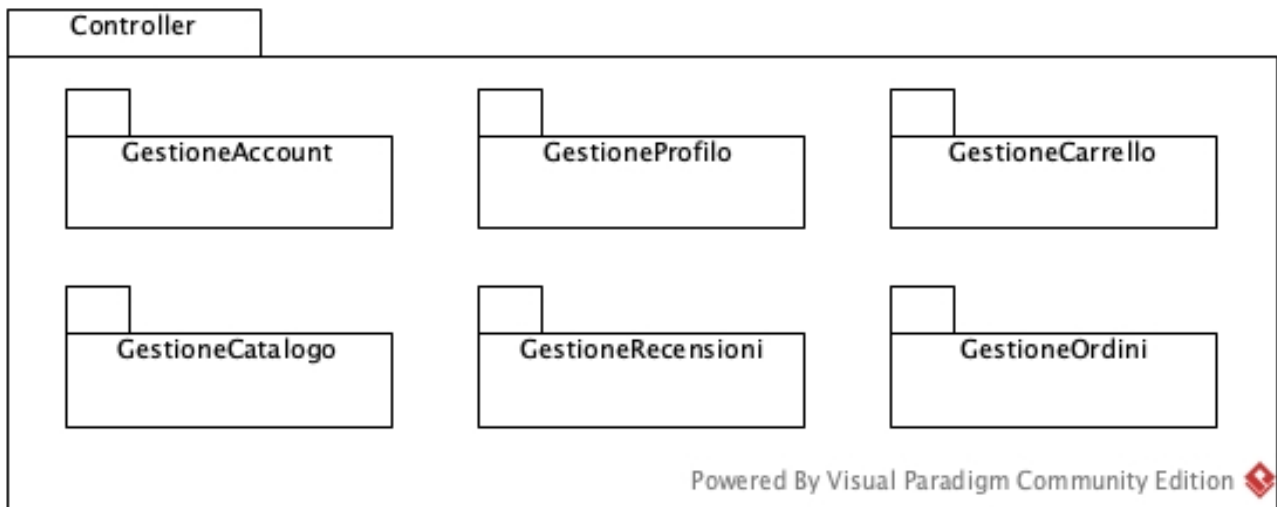


Classe	Descrizione
UtenteModel	Responsabile della conoscenza dei dati sugli utenti registrati
RuoloModel	Responsabile della conoscenza dei dati sui ruoli degli utenti registrati
MetodoPagamentoModel	Responsabile della conoscenza dei dati sui metodi di pagamento degli utenti registrati
IndirizzoModel	Responsabile della conoscenza dei dati sugli indirizzi degli utenti registrati
ProdottoModel	Responsabile della conoscenza dei dati sui prodotti
RecensioneModel	Responsabile della conoscenza dei dati sulle recensioni dei prodotti
TagliaModel	Responsabile della conoscenza dei dati sulle taglie dei prodotti
OrdineBean	Responsabile della conoscenza dei dati sugli ordini
ComposizioneBean	Responsabile della conoscenza dei dati sulle composizioni degli ordini

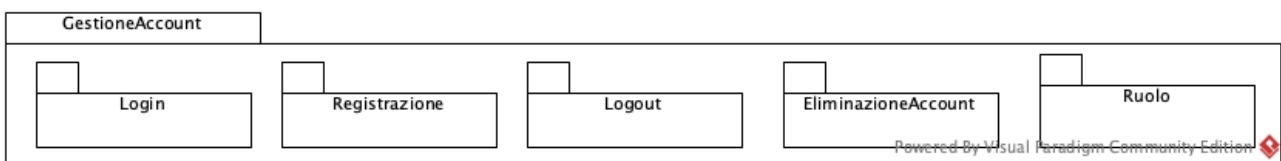
2.2.2 Application Layer

2.2.2.1 Controller

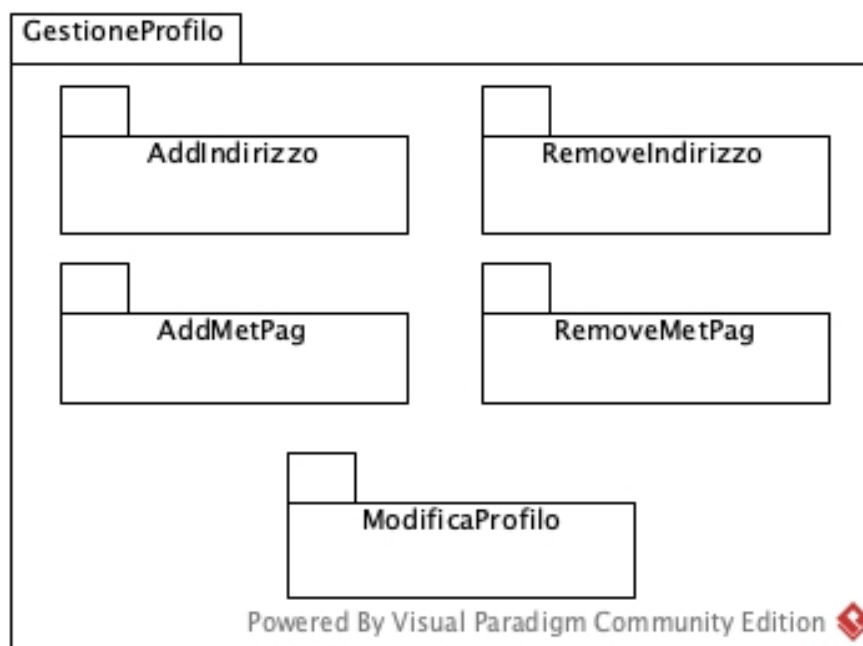
Contiene le servlet che modellano la logica di business del sistema.



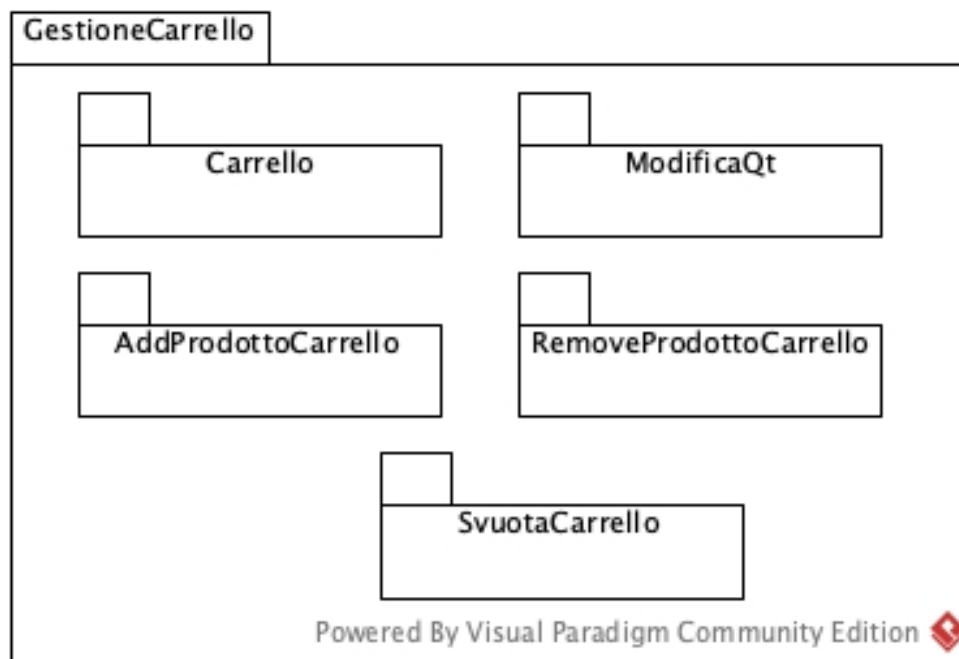
Gestione Account



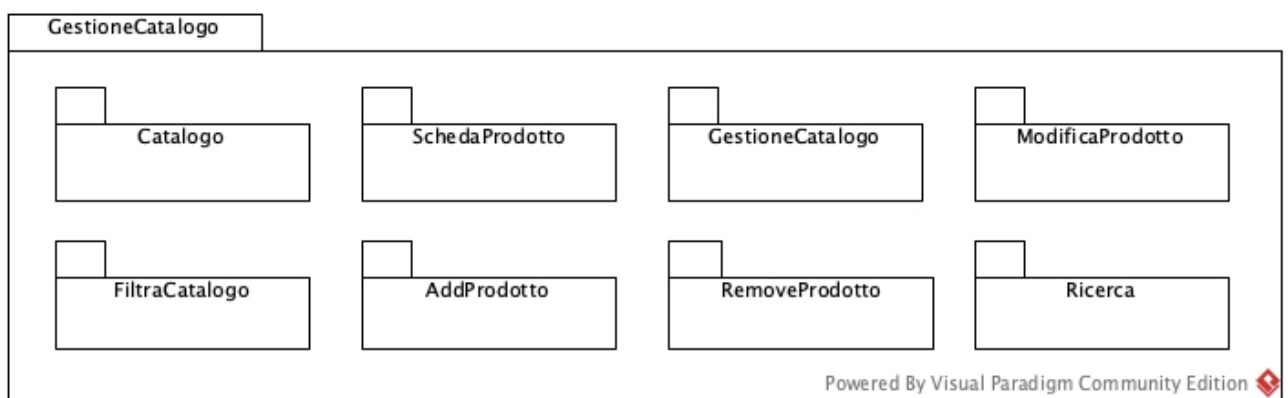
Gestione Profilo



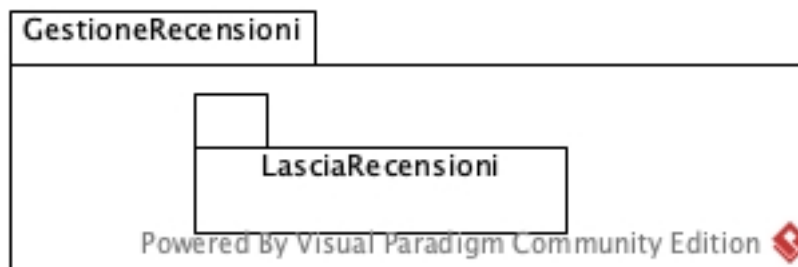
Gestione Carrello



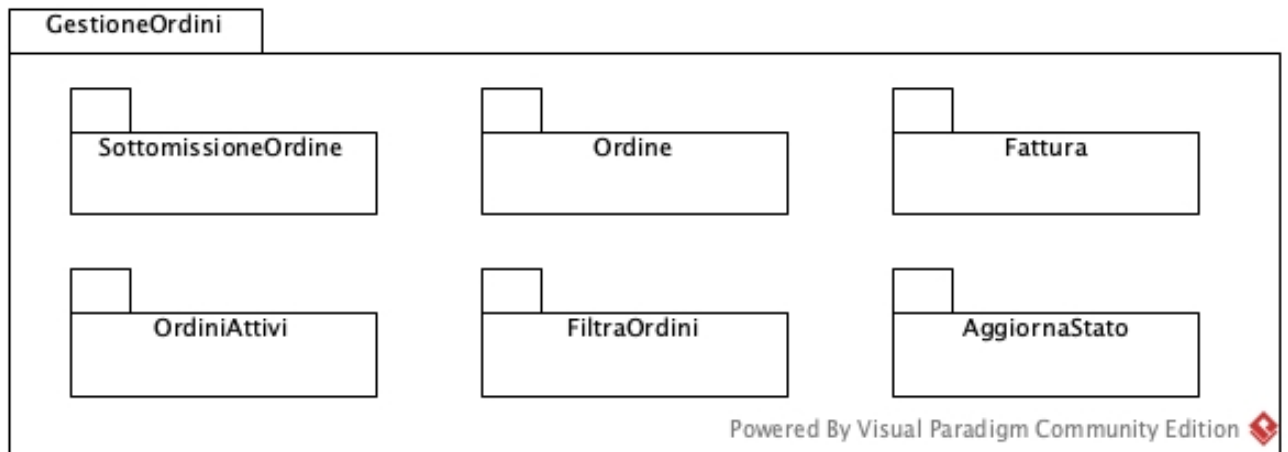
Gestione Catalogo



Gestione Recensioni



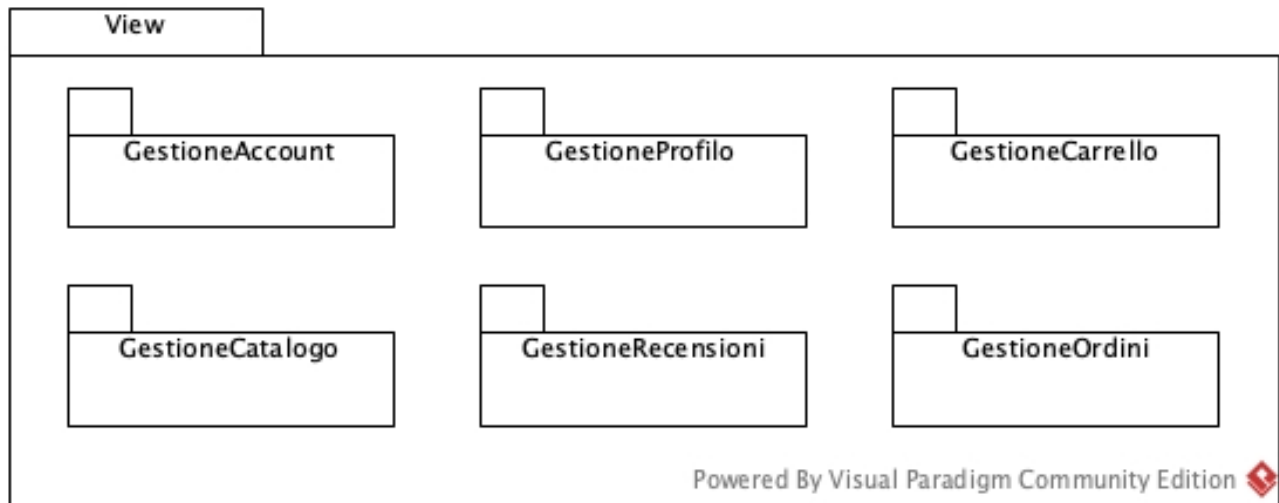
Gestione Ordini



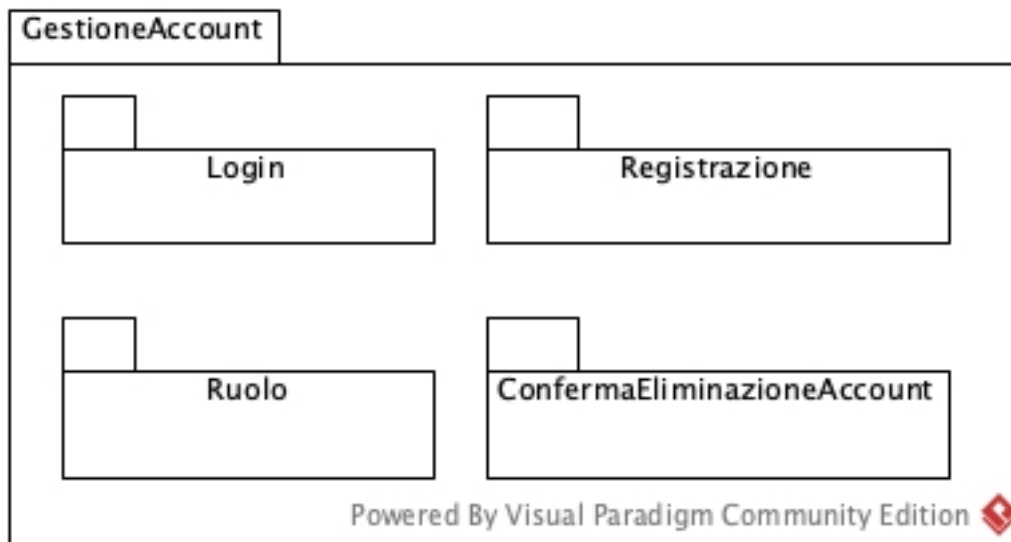
2.2.3 Presentation Layer

2.2.3.1 View

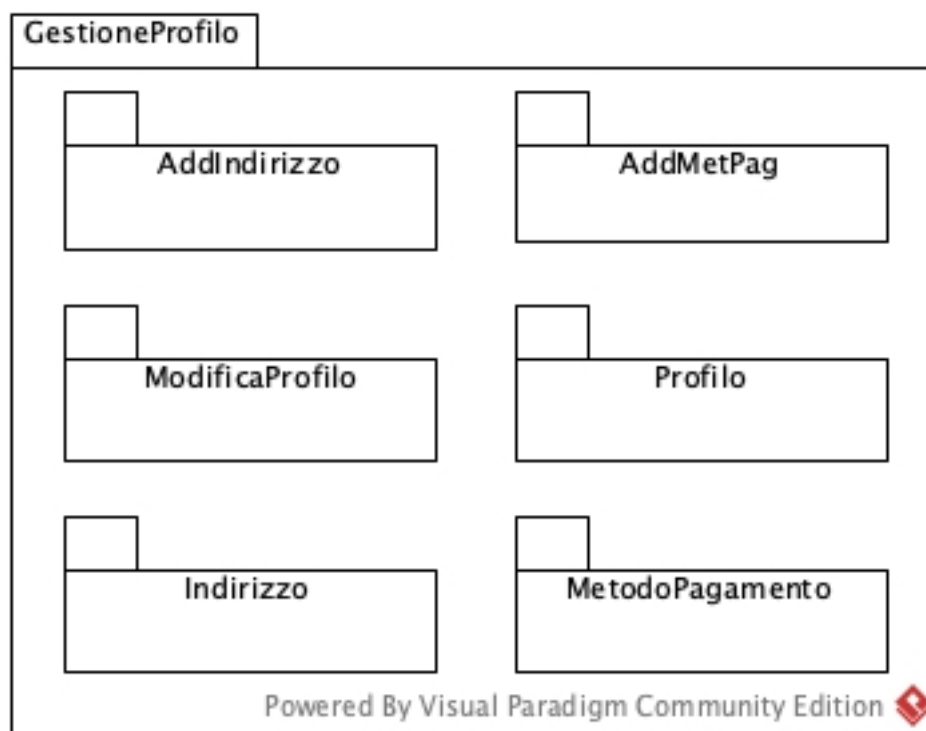
Contiene le pagine JSP e HTML che modellano l'interfaccia grafica del sistema.



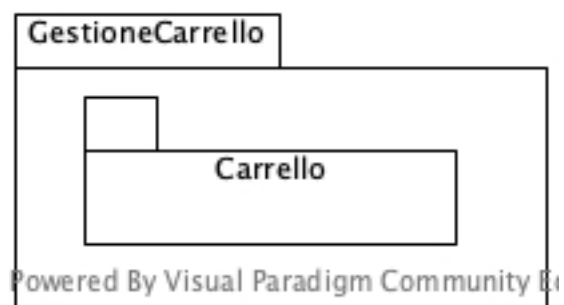
Gestione Account



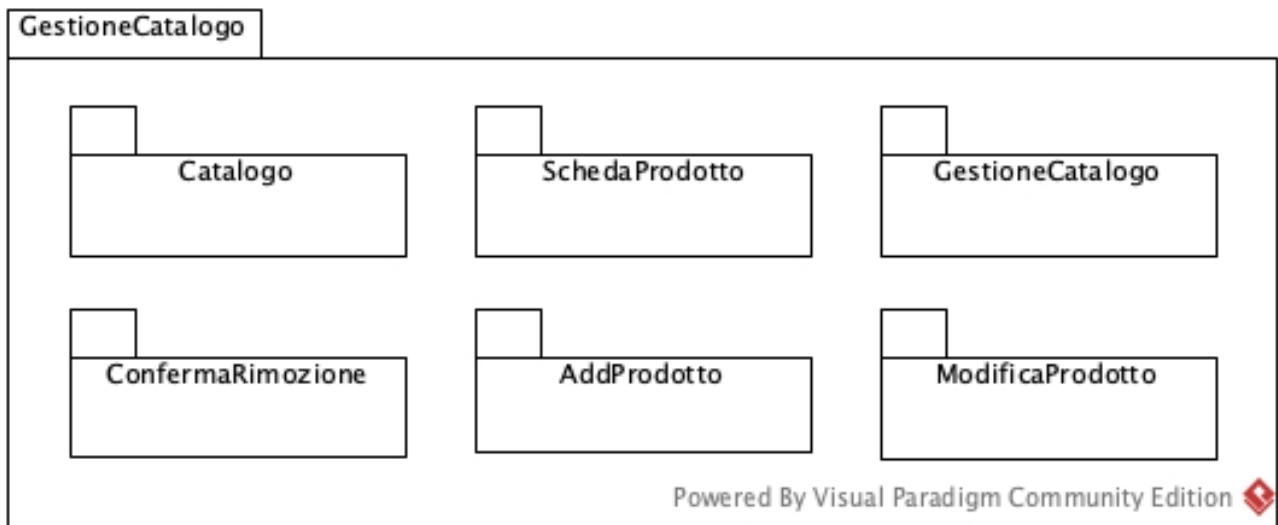
Gestione Profilo



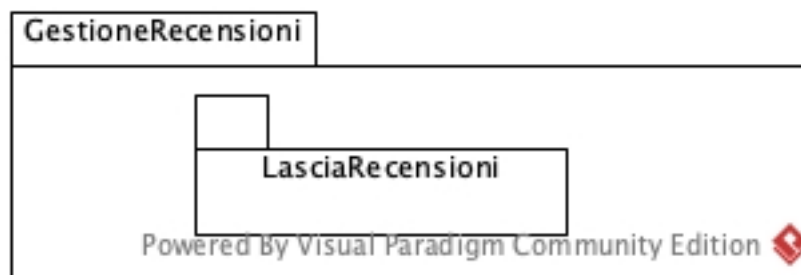
Gestione Carrello



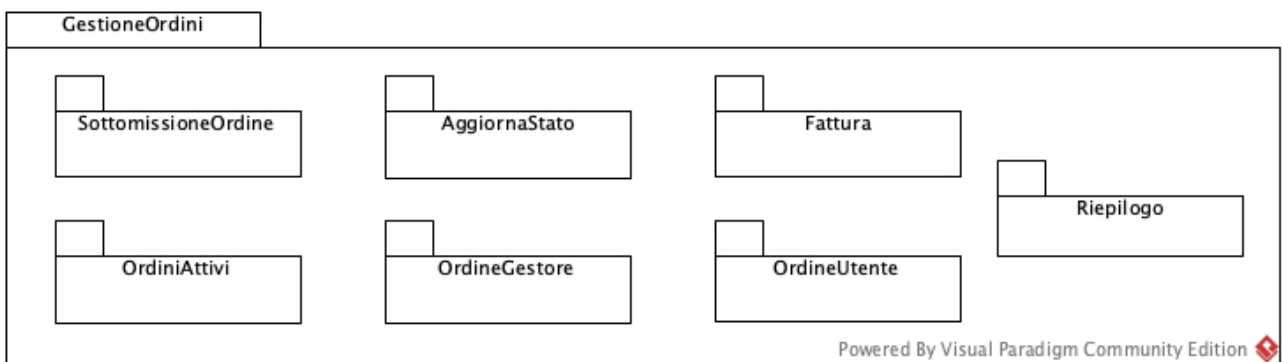
Gestione Catalogo



Gestione Recensioni



Gestione Ordini



3 Interfacce delle classi

UtenteModel

Servizio	Descrizione
public void doSave(UtenteBean utente)	Permette di salvare un utente nel DB
public UtenteBean validate(UtenteBean utente)	Permette di verificare che l'username sia presente nel DB e che la password sia quella corrispondente all'account identificato dall'username
public UtenteBean doRetrieveByUsername(String username)	Permette di ottenere un utente dal DB specificandone l'username
public boolean doUpdate(UtenteBean utente)	Permette di aggiornare i dati di un utente memorizzati nel DB
public boolean doDelete(UtenteBean utente)	Permette di eliminare un utente dal DB

RuoloModel

Servizio	Descrizione
public void doSave(RuoloBean ruolo)	Permette di salvare un ruolo per un utente nel DB
public Map<String, RuoloBean> doRetrieveByUtente(UtenteBean utente)	Permette di ottenere i ruoli di un utente dal DB

IndirizzoModel

Servizio	Descrizione
public void doSave(IndirizzoBean indirizzo)	Permette di salvare un indirizzo per un utente nel DB
public Set<IndirizzoBean> doRetrieveByUtente(UtenteBean utente)	Permette di ottenere gli indirizzi di un utente dal DB
public boolean doDelete(IndirizzoBean indirizzo)	Permette di eliminare un indirizzo di un utente dal DB

MetodoPagamentoModel

Servizio	Descrizione
public void doSave(MetodoPagamentoBean metodo)	Permette di salvare un metodo di pagamento per un utente nel DB
public Set<MetodoPagamentoBean> doRetrieveByUtente(UtenteBean utente)	Permette di ottenere i metodi di pagamento di un utente dal DB

public boolean doDelete(MetodoPagamentoBean metodo)	Permette di eliminare un metodo di pagamento di un utente dal DB
--	--

ProdottoModel

Servizio	Descrizione
public void doSave(ProdottoBean prodotto)	Permette di salvare un prodotto nel DB
public Set<ProdottoBean> doRetrieveByTipo(String tipo, String order)	Permette di ottenere i prodotti dal DB specificando un ordine
public ProdottoBean doRetrieveByCodice(String codice)	Permette di ottenere un prodotto dal DB specificandone il codice
public boolean doUpdate(ProdottoBean prodotto)	Permette di aggiornare i dati di un prodotto memorizzato nel DB
public boolean doDelete(ProdottoBean prodotto)	Permette di eliminare un prodotto dal DB

TagliaModel

Servizio	Descrizione
public void doSave(TagliaBean taglia)	Permette di salvare una taglia per un prodotto nel DB
public Set<TagliaBean> doRetrieveByProdotto(String prodotto)	Permette di ottenere le taglie di un prodotto dal DB

RecensioneModel

Servizio	Descrizione
public void doSave(RecensioneBean recensione)	Permette di salvare una recensione per un prodotto nel DB
public Set<RecensioneBean> doRetrieveByProdotto(String prodotto, String order)	Permette di ottenere le recensioni di un prodotto dal DB specificando un ordine
public String correzione(String commento)	Permette di modificare i caratteri speciali presenti nella recensione
private boolean hasSpecialChars(String input)	Permette di verificare che la recensione contenga o meno caratteri speciali

OrdineModel

Servizio	Descrizione
public void doSave(OrdineBean ordine)	Permette di salvare un ordine nel DB

<code>public Set<OrdineBean> doRetrieveAll(String order)</code>	Permette di ottenere tutti gli ordini dal DB specificando un ordinamento
<code>public OrdineBean doRetrieveByNumero(String numero)</code>	Permette di ottenere un ordine dal DB specificandone il numero
<code>public Set<OrdineBean> doRetrieveByUtente(UtenteBean utente, String order)</code>	Permette di ottenere gli ordini di un utente dal DB specificando un ordinamento
<code>public Set<OrdineBean> doRetrieveIAttivi(String order)</code>	Permette di ottenere tutti gli ordini attivi dal DB specificando un ordinamento
<code>public void aggiornaStato(OrdineBean ordine)</code>	Permette di aggiornare lo stato di un ordine memorizzato nel DB
<code>public String generatoreSottomissione()</code>	Permette di generare la data di sottomissione dell'ordine
<code>public String generatoreConsegna()</code>	Permette di generare la data di consegna dell'ordine
<code>public String generatoreNumero()</code>	Permette di generare il numero identificativo dell'ordine
<code>public int doCount()</code>	Permette di ottenere il numero totale di ordini sottomessi e memorizzati nel DB

ComposizioneModel

Servizio	Descrizione
<code>public void doSave(ComposizioneBean composizione)</code>	Permette di salvare un prodotto di un ordine nel DB
<code>public Set<ComposizioneBean> doRetrieveByOrdine(OrdineBean ordine)</code>	Permette di ottenere dal DB la composizione di un ordine

CarrelloBean

Servizio	Descrizione
<code>public void addProdotto(CarrelloItem carrelloItem)</code>	Permette di aggiungere un prodotto al carrello
<code>public void reAddProdotto(CarrelloItem carrelloItem)</code>	Permette di aggiungere un prodotto al carrello oppure incrementarne la quantità se esso è già presente
<code>public CarrelloItem getProdotto(String codiceProdotto)</code>	Permette di ottenere un prodotto dal carrello specificandone il codice
<code>public void removeProdotto(CarrelloItem carrelloItem)</code>	Permette di rimuovere un prodotto al carrello
<code>public Set<CarrelloItem> getCarrello()</code>	Permette di ottenere il carrello
<code>public void setCarrello(Set<CarrelloItem> carrello)</code>	Permette di salvare il carrello
<code>public void modificaQt(String codiceProdotto, String taglia, String action)</code>	Permette di modificare la quantità di un prodotto nel carrello, aumentandola o diminuendola

public boolean isEmpty()	Permette di verificare che il carrello sia vuoto o meno
public boolean contains(CarrelloItem carrelloItem)	Permette di verificare che un prodotto sia presente nel carrello
public void svuotaCarrello()	Permette di svuotare il carrello

3.1 Definizione contratti

3.1.1 UtenteModel

Classe	UtenteModel
Pre-condizione	<p>context UtenteModel :: public doSave(UtenteBean utente) pre utente.username!=null && utente.password!=null && utente.nome!=null && utente.cognome!=null && utente.email!=null && utente.telefono!=null</p> <p>context UtenteModel :: public validate(UtenteBean utente) pre utente.username!=null && utente.password!=null</p> <p>context UtenteModel :: public doRetrieveByUsername(String username) pre username!=null</p> <p>context UtenteModel :: public doUpdate(UtenteBean utente) pre utente.username!=null && utente.password!=null && utente.nome!=null && utente.cognome!=null && utente.email!=null && utente.telefono!=null && doRetrieveByUsername(utente.username)!=null</p> <p>context UtenteModel :: public doDelete(UtenteBean utente) pre doRetrieveByUsername(utente.username)!=null</p>
Post-condizione	<p>context UtenteModel :: public doSave(UtenteBean utente) post doRetrieveByUsername(utente.username)!=null</p> <p>context UtenteModel :: public validate(UtenteBean utente) post utenteValidato!=null</p> <p>context UtenteModel :: public doRetrieveByUsername(String username) post utenteRestituito!=null</p> <p>context UtenteModel :: public doUpdate(UtenteBean utente) post doUpdate==true</p> <p>context UtenteModel :: public doDelete(UtenteBean utente) post doDelete==true</p>
Invarianti	

3.1.2 RuoloModel

Classe	RuoloModel
--------	------------

Pre-condizione	context RuoloModel :: public doSave(RuoloBean ruolo) pre ruolo.usr!=null && ruolo.permesso!=null context RuoloModel :: public doRetrieveByUtente(UtenteBean utente) pre utente.username!=null
Post-condizione	context RuoloModel :: public doSave(RuoloBean ruolo) post ruoli=doRetrieveByUtente(utente) -> ruoli!=null && ruoli!=empty && ruoli.contains(ruolo)==true context RuoloModel :: public doRetrieveByUtente(UtenteBean utente) post ruoliRestituiti!=null && ruoliRestituiti!=empty
Invarianti	

3.1.3 IndirizzoModel

Classe	IndirizzoModel
Pre-condizione	context IndirizzoModel :: public doSave(IndirizzoBean indirizzo) pre indirizzo.usr!=null && indirizzo.citta!=null && indirizzo.via!=null && indirizzo.civico!=null && indirizzo.cap!=null context IndirizzoModel :: public doRetrieveByUtente(UtenteBean utente) pre utente.username!=null context IndirizzoModel :: public doDelete(IndirizzoBean indirizzo) pre indirizzi=doRetrieveByUtente(utente) -> indirizzi!=null && indirizzi!=empty && indirizzi.contains(indirizzo)==true
Post-condizione	context IndirizzoModel :: public doSave(IndirizzoBean indirizzo) post indirizzi=doRetrieveByUtente(utente) -> indirizzi!=null && indirizzi!=empty && indirizzi.contains(indirizzo)==true context IndirizzoModel :: public doRetrieveByUtente(UtenteBean utente) post context IndirizzoModel :: public doDelete(IndirizzoBean indirizzo) post doDelete==true
Invarianti	

3.1.4 MetodoPagamentoModel

Classe	MetodoPagamentoModel
--------	----------------------

Pre-condizione	<p>context MetodoPagamentoModel :: public doSave(MetodoPagamentoBean metodo) pre metodo.usr!=null && metodo.tipo!=null && metodo.numero!=null</p> <p>context MetodoPagamentoModel :: public doRetrieveByUtente(UtenteBean utente) pre utente.username!=null</p> <p>context MetodoPagamentoModel :: public doDelete(MetodoPagamentoBean metodo) pre metodi=doRetrieveByUtente(utente) -> metodi!=null && metodi!=empty && metodi.contains(metodo)==true</p>
Post-condizione	<p>context MetodoPagamentoModel :: public doSave(MetodoPagamentoBean metodo) post metodi=doRetrieveByUtente(utente) -> metodi!=null && metodi!=empty && metodi.contains(metodo)==true</p> <p>context MetodoPagamentoModel :: public doRetrieveByUtente(UtenteBean utente) post</p> <p>context MetodoPagamentoModel :: public doDelete(MetodoPagamentoBean metodo) post doDelete==true</p>
Invarianti	

3.1.5 ProdottoModel

Classe	ProdottoModel
Pre-condizione	<p>context ProdottoModel :: public doSave(ProdottoBean prodotto) pre prodotto.codice!=null && prodotto.nome!=null && prodotto.tipo!=null && prodotto.marca!=null && prodotto.qt>0 && prodotto.prezzo>0 && prodotto.iva>0 && prodotto.descrizione!=null</p> <p>context ProdottoModel :: public doRetrieveByCodice(String codice) pre codice!=null</p> <p>context ProdottoModel :: public doRetrieveByTipo(String tipo, String order) pre tipo!=null</p> <p>context ProdottoModel :: public doUpdate(ProdottoBean prodotto) pre prodotto.codice!=null && prodotto.nome!=null && prodotto.tipo!=null && prodotto.marca!=null && prodotto.qt>0 && prodotto.prezzo>0 && prodotto.iva>0 && prodotto.descrizione!=null && doRetrieveByCodice(prodotto.codice)!=null</p> <p>context ProdottoModel :: public doDelete(ProdottoBean prodotto) pre doRetrieveByCodice(prodotto.codice)!=null</p>

Post-condizione	context ProdottoModel :: public doSave(ProdottoBean prodotto) post doRetrieveByCodice(prodotto.codice)!=null context ProdottoModel :: public doRetrieveByCodice(String codice) post prodottoRestituito!=null context ProdottoModel :: public doRetrieveByTipo(String tipo, String order) post prodottiRestituiti!=null && prodottiRestituiti!=empty context ProdottoModel :: public doUpdate(ProdottoBean prodotto) post doUpdate==true context ProdottoModel :: public doDelete(ProdottoBean prodotto) post doDelete==true
Invarianti	

3.1.6 TagliaModel

Classe	TagliaModel
Pre-condizione	context TagliaModel :: public doSave(TagliaBean taglia) pre taglia.prodotto!=null && taglia.misura!=null context TagliaModel :: public doRetrieveByProdotto(String prodotto) pre prodotto!=null
Post-condizione	context TagliaModel :: public doSave(TagliaBean taglia) post taglie=doRetrieveByProdotto(prodotto) -> taglie!=null && taglie!=empty && taglie.contains(taglia)==true context TagliaModel :: public doRetrieveByProdotto(String prodotto) post taglieRestituite!=null && taglieRestituite!=empty
Invarianti	

3.1.7 RecensioneModel

Classe	RecensioneModel
Pre-condizione	context RecensioneModel :: public doSave(RecensioneBean recensione) pre recensione.prodotto!=null && recensione.usr!=null && recensione.voto<11 && recensione.voto>0 && recensione.commento!=null context RecensioneModel :: public doRetrieveByProdotto(String prodotto, String order) pre prodotto!=null
Post-condizione	context RecensioneModel :: public doSave(RecensioneBean recensione) post recensioni=doRetrieveByProdotto(prodotto) -> recensioni!=null && recensioni!=empty && recensioni.contains(recensione)==true context RecensioneModel :: public doRetrieveByProdotto(String prodotto, String order) post

Invarianti	
------------	--

3.1.8 OrdineModel

Classe	OrdineModel
Pre-condizione	<p>context OrdineModel :: public doSave(OrdineBean ordine) pre ordine.numero!=null && ordine.stato!=null && ordine.pagamento!=null && ordine.indirizzo!=null && ordine.totale>0 && ordine.sottomissione!=null && ordine.consegna!=null && ordine.usr!=null</p> <p>context OrdineModel :: public doRetrieveByNumero(String numero) pre numero!=null</p> <p>context OrdineModel :: public doRetrieveAll(String order) pre</p> <p>context OrdineModel :: public doRetrieveByUtente(UtenteBean utente, String order) pre utente.username!=null</p> <p>context OrdineModel :: public doRetrieveIAttivi(String order) pre</p> <p>context OrdineModel :: public aggiornaStato(OrdineBean ordine) pre ordine.numero!=null && ordine.stato!=null && ordine.consegna!=null && doRetrieveByNumero(ordine.codice)!=null</p>
Post-condizione	<p>context OrdineModel :: public doSave(OrdineBean ordine) post doRetrieveByNumero(ordine.numero)!=null</p> <p>context OrdineModel :: public doRetrieveByNumero(String numero) post ordineRestituito!=null</p> <p>context OrdineModel :: public doRetrieveAll(String order) post</p> <p>context OrdineModel :: public doRetrieveByUtente(UtenteBean utente, String order) post</p> <p>context OrdineModel :: public doRetrieveIAttivi(String order) post ordiniAttivi.stato!=CONSEGNATO</p> <p>context OrdineModel :: public aggiornaStato(OrdineBean ordine) post ordineAggiornato=doRetrieveByNumero(ordine.codice) -> ordineAggiornato!=null && ordineAggiornato.stato==ordine.stato && ordineAggiornato.consegna==ordine.consegna</p>
Invarianti	

3.1.9 ComposizioneModel

Classe	ComposizioneModel
--------	-------------------

Pre-condizione	<p>context ComposizioneModel :: public doSave(ComposizioneBean composizione)</p> <p>pre composizione.ordine!=null && composizione.prodotto!=null && composizione.nomeprodotto!=null && composizione.qt>0 && composizione.prezzoven>0 && composizione.ivaven>0 && taglia!=null</p> <p>context ComposizioneModel :: public doRetrieveByOrdine(OrdineBean ordine)</p> <p>pre ordine.numero!=null</p>
Post-condizione	<p>context ComposizioneModel :: public doSave(ComposizioneBean composizione)</p> <p>post composizioni=doRetrieveByOrdine(ordine) -> composizioni!=null && composizioni!=empty && composizioni.contains(composizione)==true</p> <p>context ComposizioneModel :: public doRetrieveByProdotto(OrdineBean ordine)</p> <p>post composizioni!=null && composizioni!=empty</p>
Invarianti	

3.1.10 CarrelloBean

Classe	CarrelloBean
--------	--------------

Pre-condizione	<p>context CarrelloBean :: public addProdotto(CarrelloItem carrelloItem) pre carrelloItem.prodotto!=null && carrelloItem.qt>0 && carrelloItem.taglia!=null</p> <p>context CarrelloBean :: public reAddProdotto(CarrelloItem carrelloItem) pre carrelloItem.prodotto!=null && carrelloItem.qt>0 && carrelloItem.taglia!=null</p> <p>context CarrelloBean :: public getProdotto(String codiceProdotto) pre codiceProdotto!=null</p> <p>context CarrelloBean :: public removeProdotto(CarrelloItem carrelloItem) pre carrelloItem.prodotto!=null && carrelloItem.qt>0 && carrelloItem.taglia!=null</p> <p>context CarrelloBean :: public getCarrello() pre</p> <p>context CarrelloBean :: public setCarrello(Set<CarrelloItem> carrello) pre carrello!=null && for each carrelloItem in carrello -> carrelloItem.prodotto!=null && carrelloItem.qt>0 && carrelloItem.taglia!=null</p> <p>context CarrelloBean :: public modificaQt(String codiceProdotto, String taglia, String action) pre codiceProdotto!=null && taglia!=null && action!=null</p> <p>context CarrelloBean :: public contains(CarrelloItem carrelloItem) pre carrelloItem.prodotto!=null && carrelloItem.qt>0 && carrelloItem.taglia!=null</p> <p>context CarrelloBean :: public svuotaCarrello() pre</p>
----------------	--

Post-condizione	<pre> context CarrelloBean :: public addProdotto(CarrelloItem carrelloItem) post getProdotto(carrelloItem.prodotto)==carrelloItem context CarrelloBean :: public reAddProdotto(CarrelloItem carrelloItem) post if carrello.contains(carrelloItem)==true -> incrementa getProdotto(carrelloItem.prodotto).qt else -> getProdotto(carrelloItem.prodotto)==carrelloItem context CarrelloBean :: public getProdotto(String codiceProdotto) post prodottoRestituito!=null context CarrelloBean :: public removeProdotto(CarrelloItem carrelloItem) post carrello.contains(carrelloItem)==false context CarrelloBean :: public getCarrello() post context CarrelloBean :: public setCarrello(Set<CarrelloItem> carrello) post carrello.getCarrello()==carrello context CarrelloBean :: public modificaQt(String codiceProdotto, String taglia, String action) post if action==plus -> incrementa getProdotto(codiceProdotto).qt else -> decrementa getProdotto(codiceProdotto).qt context CarrelloBean :: public contains(CarrelloItem carrelloItem) post true if carrelloItem in carrello false altrimenti context CarrelloBean :: public svuotaCarrello() post carrello==empty </pre>
Invarianti	

4 Design pattern

4.1 MVC

Il pattern architetturale Model-View-Controller (MVC) è molto diffuso nello sviluppo di sistemi software e permette la separazione della logica di business dalla logica di presentazione dei dati.

MVC prevede un'architettura Three-Tier:

- Model: responsabile della conoscenza del dominio applicativo e implementa lo store principale di dati
- View: responsabile di mostrare gli oggetti del dominio applicativo all'utente

- Controller: responsabile dell'interazione con l'utente e di notificare le View dei cambiamenti nel Model. Esso detta esplicitamente il flusso di controllo

