Università degli Studi di Salerno
Corso di Ingegneria del Software

HomeDecore
Object Design Document
Versione 1.4

Data: 09/12/2024

Progetto: HomeDecore	Versione: 1.1
Documento: Object Design	Data: 09/12/2024

Coordinatore del progetto:

Nome	**Matricola**

Partecipanti:

Nome	**Matricola**
Alfieri Riccardo	0512116533
Cammarota Lucageneroso	0512116941
Fasolino Pietro	0512116473
Marino Michele Graziano	0512117109

Scritto da:	Lucageneroso Cammarota Alfieri Riccardo Fasolino Pietro

Revision History

Data	**Versione**	**Descrizione**	**Autore**
03/01/2025	1.1	Modifiche al OrderManagement	Lucageneroso Cammarota
05/01/2025	1.2	Modifiche a UserManagement	Lucageneroso Cammarota
13/02/2025	1.3	Modifiche finali	Alfieri Riccardo

Sommario

Introduzione

Object Design trade-offs

Robustezza VS Tempo

Il tempo per rendere il sistema robusto a tutti i possibili eventi causa di errore sarebbe più lungo di quanto messo a disposizione per la consegna del progetto. All'interno del sistema HomeDecore è stato decido di dare priorità a controlli sull'input, per offrirne robustezza in una prima sua versione.

Attendibilità VS Tempo

L'attendibilità, essendo un requisito necessario per il funzionamento del sistema, sarà garantita per tutti i metodi messi a disposizione nel sistema nel momento della sua esecuzione.

Linee guida

Di seguito si riportano le convenzioni concordate dal team per garantire un vocabolario standard in fase di sviluppo del codice.

- I nomi delle interfacce avranno il suffiso "-Interface".
- Gli errori saranno gestiti tramite eccezioni controllate.
- Le classi che si occuperanno di relazionarsi col DB avranno il suffisso –"Service".
- I nomi dei metodi per le retrieve avranno il prefisso "findBy" e il suffisso evocherà il criterio in base al quale si effettuerà la ricerca.
- Le operazioni CRUD saranno così nominate: "CrudOperationEntity"

Riferimenti

Per le scelte progettuali che portano alla stesura di questo documento si farà riferimento a:

- RAD_HomeDecore
- SDD_HomeDecore

Directory

Nella prima versione di implementazione del progetto, sarà riportata la struttura del sistema tramite suddivisione in pacchetti, cartelle e file, con rispettivi ruoli

Src

Questa cartella conterrà il codice java per l'implementazione del sistema.

Database

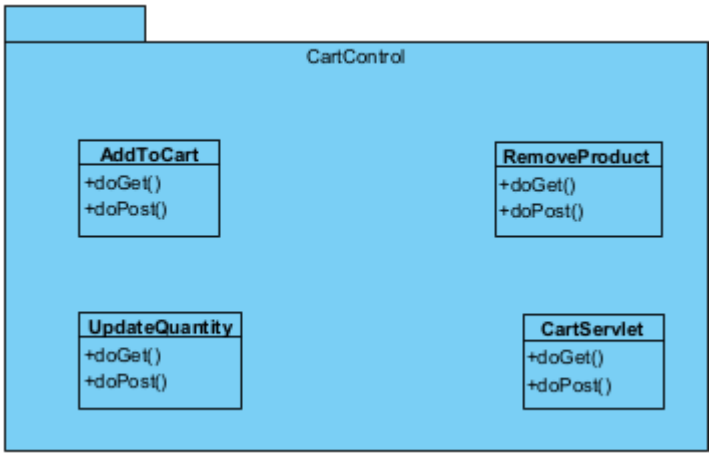
Questa directory conterrà lo schema del database

Control

Questo package conterrà le servlet necessarie all'interazione tra l'utente e il sistema.

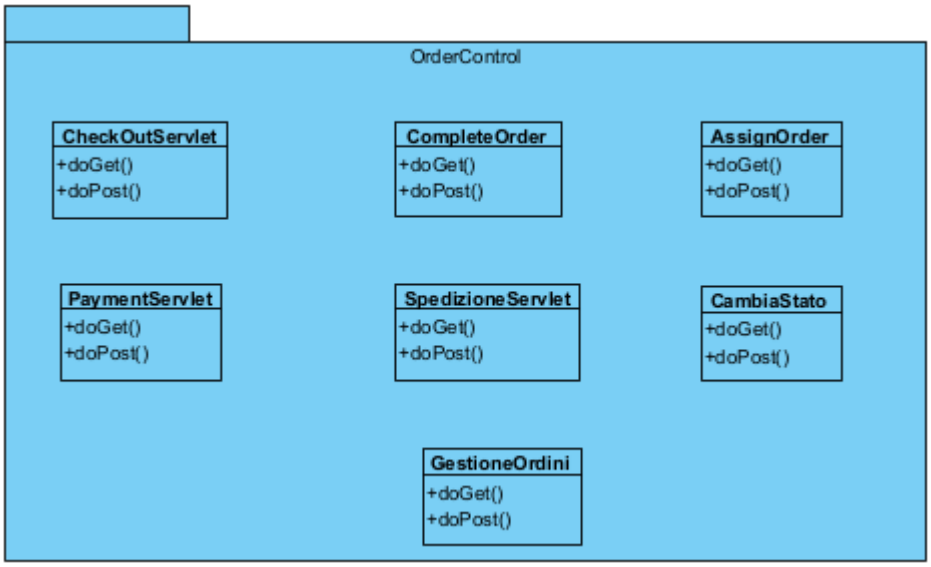
Control.CartControl

Questo package conterrà le servlet per la coordinazione delle operazioni relative al carrello.

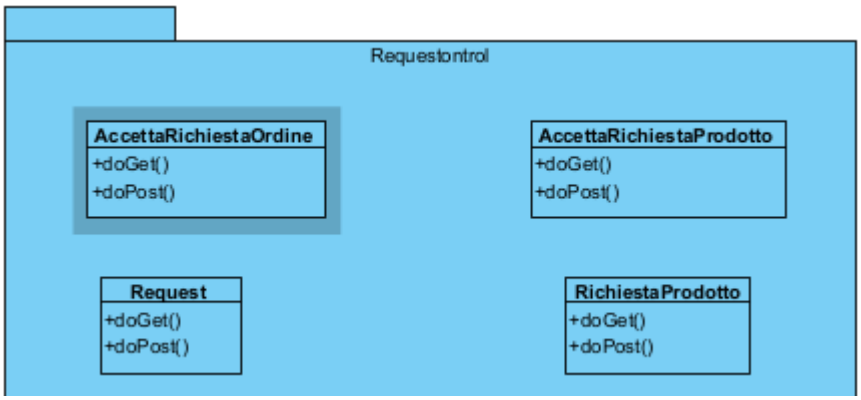


Control. OrderControl

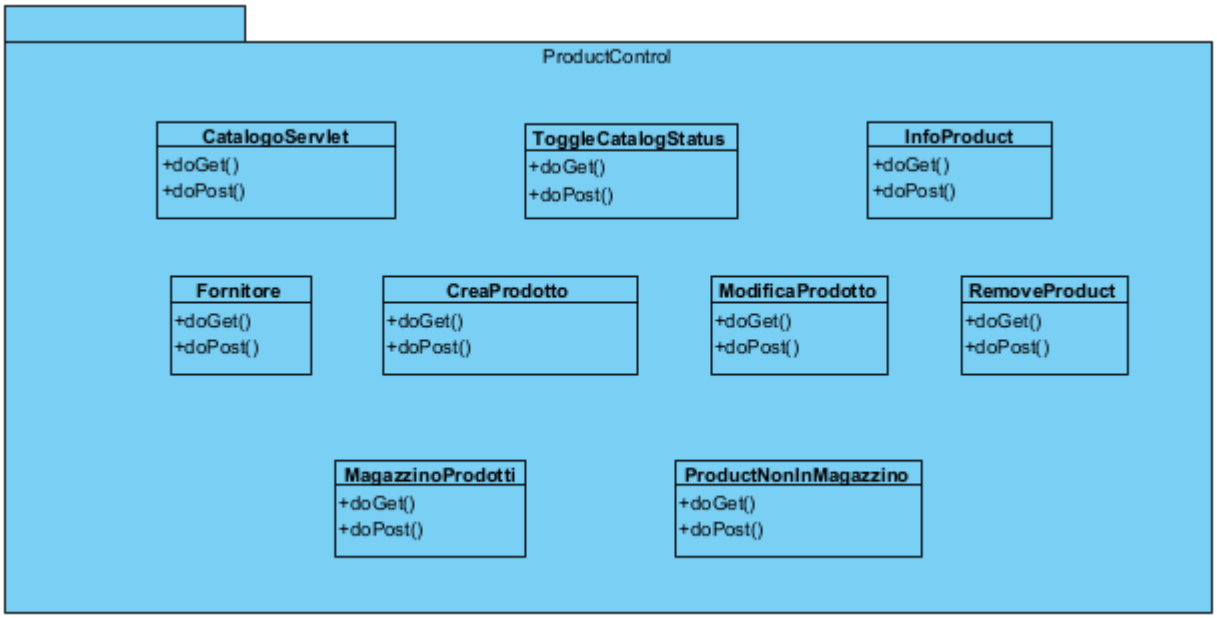
Questo package conterrà le servlet per coordinare le operazioni relative al completamento dell'acquisto.



Control. RequestControl



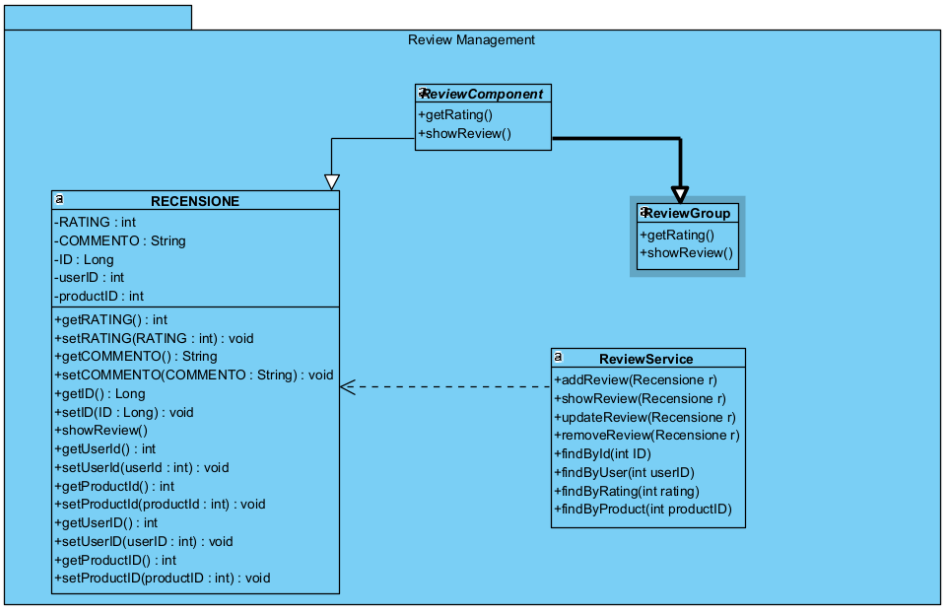
Control.ProductControl



Packages

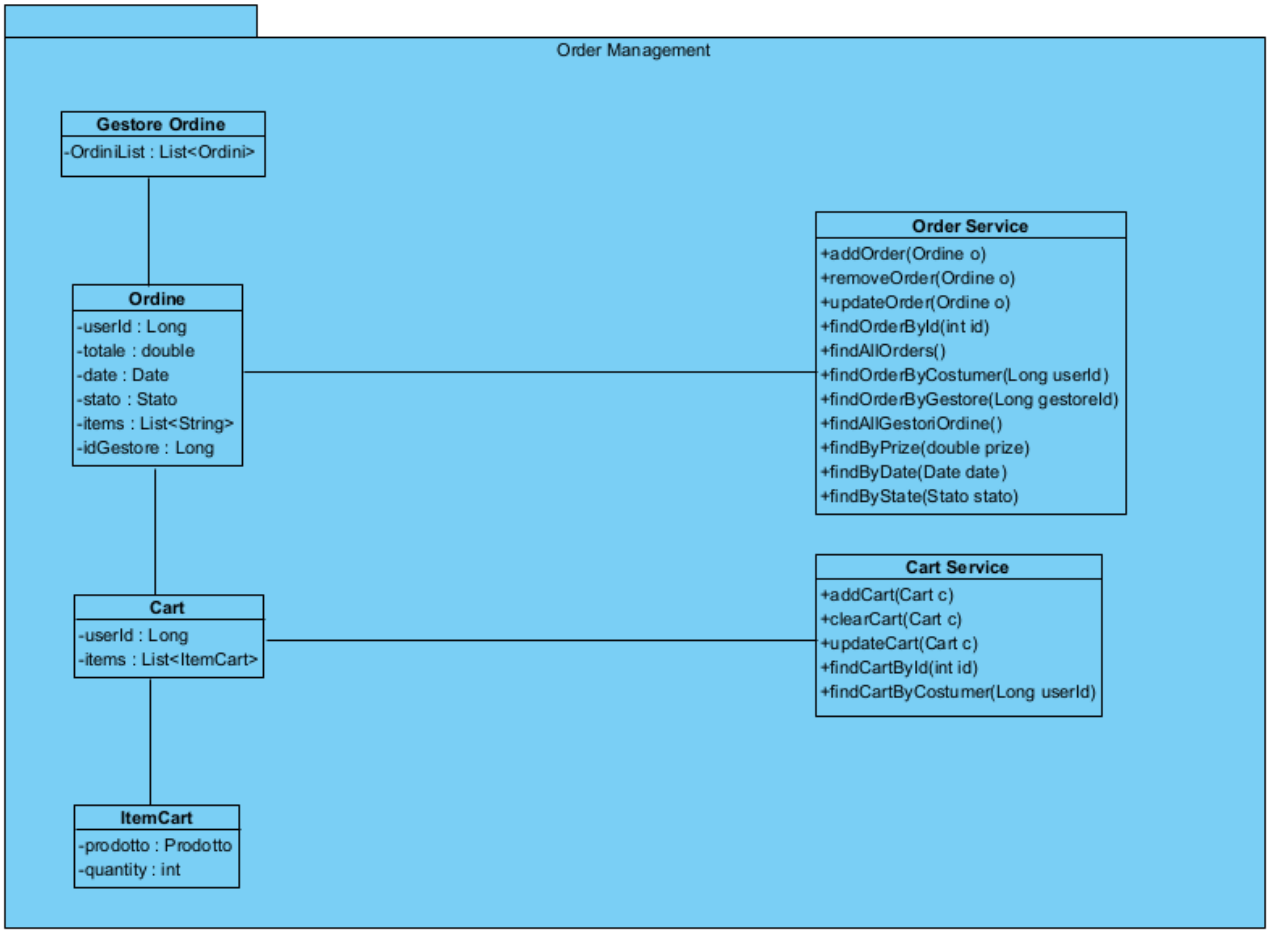
ReviewManagement

Questo package contiene le classi Bean e Service a cui viene affidata la gestione delle recensioni.



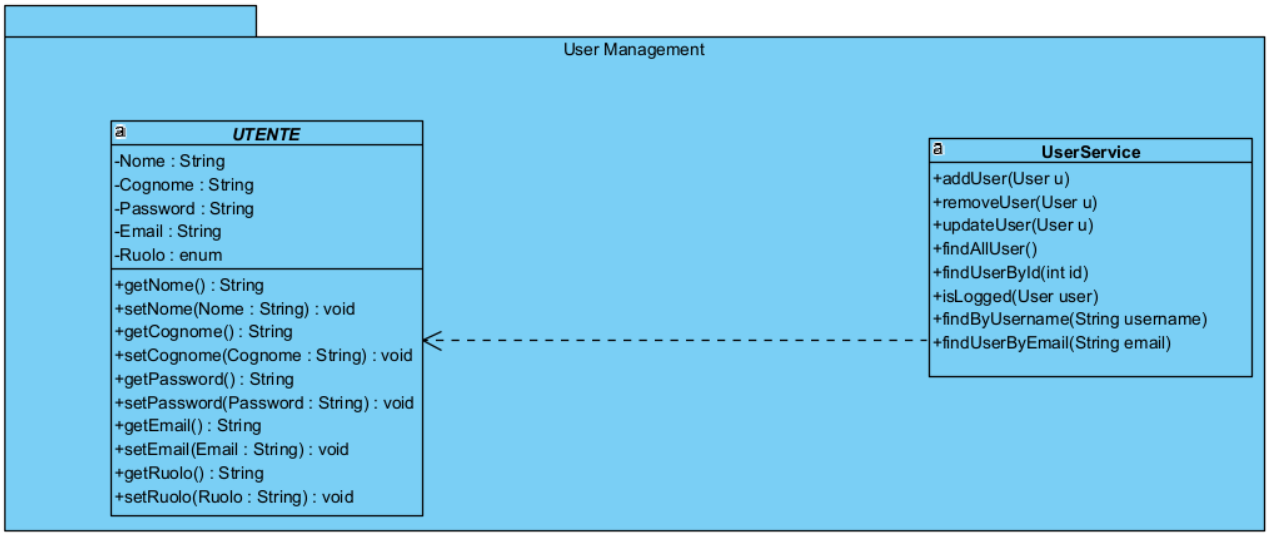
OrderManagement

Questo package contiene le classi Bean e Service a cui viene affidata la gestione e finalizzazione degli acquisti del cliente.



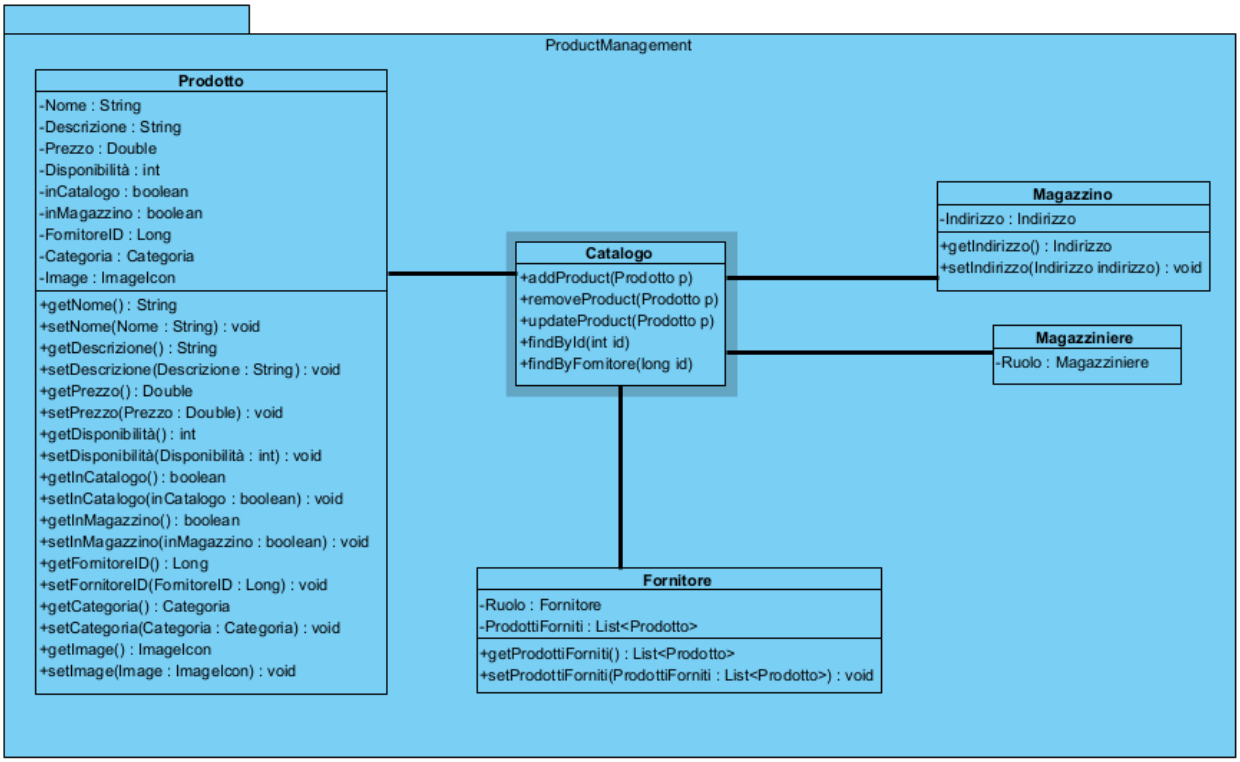
UserManagement

Questo package contiene le classi Bean e Service a cui viene affidata la gestione delle informazioni degli utenti e dei loro account.
NOTA: Utente è un'abstract class, nel package vanno aggiunte le implementazioni dei singoli utenti.



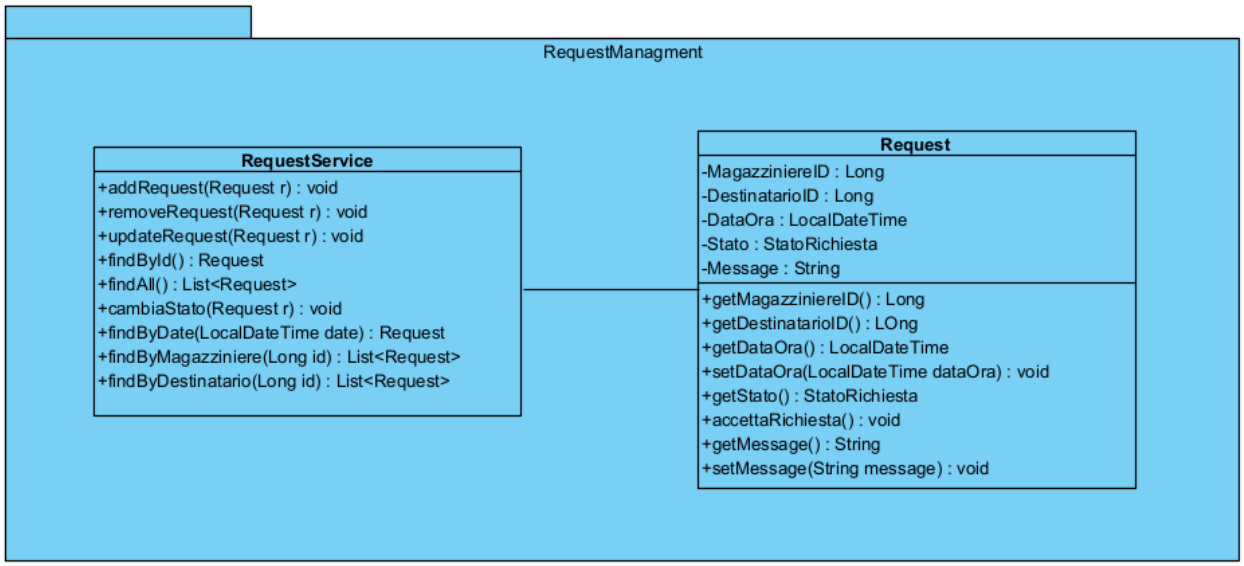
ProductManagement

Questo package contiene le classi Bean e Service a cui viene affidata la gestione delle informazioni dei prodotti e del catalogo.



RequestManagment

Questo package contiene classi e service per gestire le richieste, sia di prodotti che di gestione di ordini effettuate dai magazzinieri.
NOTA: Request è un'abstract class, nel package vanno aggiunte le implementazioni delle singole richieste



Interfacce di classe

Catalogo

CATALOGO

Descrizione	Questa classe, interfacciandosi con il DB, permette di effettuare operazioni di ricerca sui prodotti presenti nel magazzino
Pre-condizioni	context Catalogo::findBy(ID:Integer) Pre: ID<>null and ID>0 context Catalogo::findBy(price:Integer) Pre: price>=0 context Catalogo::findByName(name:String) Pre: name<>null context Catalogo::addProduct(prodotto:Prodotto) pre: prodotto<>null and magazzino::containsProduct(prodotto) and not self.productList->includes(prodotto) context Catalogo::removeProduct(prodotto:Prodotto) pre: prodotto<>null and magazzino::containsProduct(prodotto) and self.productList->includes(prodotto) context Catalogo::showProducts() Pre: true context Catalogo::updateProduct(prodotto:Prodotto) pre: prodotto<>null and self.productList->include(prodotto) context Catalogo::isAvailable(prodotto:Prodotto) pre: prodotto<>null and self.productList->include(prodotto) Context Catalogo :: getProductsInCatalogo() Pre: true Context Catalogo :: getProductsInMagazzino() Pre: true

Post-condizioni	**context** Catalogo::findBy(ID:Integer) **post**:
	if Catalogo.productList->exists(p | p.id=ID) then
	result = self.ProdottiList->any(p | p.id = ID)
	else
	result=null
	endif
	
	context Catalogo::findBy(price:Integer) post:
	if Catalogo.productList->exists(p | p.prezzo<price) then
	result = self.ProdottiList->all(p | p.prezzo<price)
	else
	result=null
	endif

	context Catalogo::findByName(name:String) post:
	if Catalogo.productList->exists(p | p.nome=name) then
	result = self.ProdottiList->any(p | p.nome=name)
	else
	result=null
	endif
	
	context Catalogo::addProduct(prodotto:Prodotto) post:
	self.productList=self@pre.productList->including(prodotto)
	
	context Catalogo::removeProduct(prodotto:Prodotto) post:
	self.productList=self@pre.productList->excluding(prodotto)
	
	context Catalogo::showProducts() post:
	result=self.productList ->forAll(p | p <> null)
	
	context Catalogo::updateProduct(prodotto:Prodotto) post:
	let oldProduct = self.productList ->select(p | p.ID = prodotto.ID)->first() self.ProdottiList = self.productList @pre->excluding(oldProduct)-> including(prodotto)
	
	context Catalogo::isAvailable(prodotto:Prodotto) post:
	result= self.productList->all(prodotto | prodotto.disponibilità>0)
	
	Context Catalogo :: getProductsInCatalogo()
	Post: result->forall(p | p.inCatalogo = true)
	
	Context Catalogo :: getProductsInMagazzino()
	Post: result->forall(p | p.inMagazzino = true)

Invarianti	

CartService

CartService	
Descrizione	Questa classe, interfacciandosi con il DB, permette di gestire la persistenza del carrello in sessioni utente differenti.

Pre-condizioni	context CartServiceRemote::addCart(cart: Cart) pre: cart <> null context CartServiceRemote::clearCart(cart: Cart) pre: cart <> null and Cart.allInstances()->includes(cart) context CartServiceRemote::updateCart(cart: Cart) pre: cart <> null and Cart.allInstances()->includes(cart) context CartServiceRemote::findCartById(id: Integer) pre: id <> null context CartServiceRemote::findCartByCostumer(userId: Long) pre: userId > 0 and User.allInstances()->exists(u) context CartServiceRemote::removeProductFromCart(cartId: Integer, productId: Integer) pre: cartId > 0 and productId > 0 and Cart.allInstances()->exists(c)
Post-condizioni	context CartServiceRemote::addCart(cart: Cart) post: Cart.allInstances()->includes(cart) context CartServiceRemote::clearCart(cart: Cart) post: cart.products->isEmpty() context CartServiceRemote::updateCart(cart: Cart) post: Cart.allInstances()->exists(c) context CartServiceRemote::findCartById(id: Integer) post: result = Cart.allInstances()->any(c) context CartServiceRemote::findCartByCostumer(userId: Long) post: result = Cart.allInstances()->any(c) context CartServiceRemote::removeProductFromCart(cartId: Integer, productId: Integer) post: Cart.allInstances()->exists(c)
Invarianti	

ReviewService

REVIEW SERVICE	
Descrizione	La classe sarà usata dai client per accedere e manipolare l'entità Review nel DB.

Pre-condizioni	Context ReviewService :: addReview (review: Recensione) Pre: review<>null and review.rating <> null Context ReviewService :: showReview(review: Recensione) Pre: review<>null Context ReviewService:: updateReview(review: Recensione) Pre: review<>null Context ReviewService :: deleteReview(review: Recensione) Pre: review<>null and Recensione.allInstances(r| r=review) Context: ReviewService :: findById (ID : Integer) Pre: ID<>null and ID>0 Context: ReviewService :: findByUser(userID: Integer) Pre: userID<>null and Utente.allInstances() → exists (u | u.id= userID) Context: ReviewService :: findByRating(rating: Double) Pre: rating<>null and rating>0 Context: ReviewService :: findByProduct (productID: Integer) Pre: productID<>null and Prodotto.allInstances() → exists (p| p.ID= productID)
Post-condizioni	Context ReviewService :: addReview(review: Recensione) Post: Recensione.allInstances() → exists(r | r=review) Context ReviewService :: deleteReview(review: Recensione) Post: not [Recensione.allInstances() → exists(r | r=review)] Context ReviewService :: updateReview(review: Recensione) Post: Recensione.allInstances() → exists(r | r=review) Context: ReviewService :: findById (ID : Integer) Post: result= Recensione.allInstances() → any(r| r.ID =ID) Context: ReviewService :: findByUser(userID: Integer) Post: result= Recensione.allInstances() → any(r| r.userID =userID) Context: ReviewService :: findByRating(rating: Double) Post: result= Recensione.allInstances() → any(r| r.rating=rating) Context: ReviewService :: findByProduct (productID: Integer) Post: result= Recensione.allInstances() → any(r|r.productID= productID)
Invarianti	

OrderService

ORDER SERVICE	
Descrizione	Questa classe permette ai client di accedere e manipolare le informazioni sugli ordini effettuati sul DB

Pre-condizioni	Context OrdineService :: addOrder(o : Ordine) Pre: o<>null And o.userID <> null and Utente.allInstances() → exists (u|u.ID=userID) And o.ID<>null and o.ID>0 And o.Data<>null And o.Totale<> null and o.Totale>0 Context OrdineService :: removeOrder(id: Integer) Pre: id<>null and id>0 and Ordine.allInstances()→exists(o|o.id=id) Context OrdineService findOrdersByCustomer (u: Utente) Pre: u<>null Context OrdineService findByDate(d: Date) Pre: d<>null Context OrdineService updateOrder(o: Ordine) Pre: o<>null and Ordine.allInstances()→exists(or| or.id = o.id)
Post-condizioni	Context OrdineService :: addOrder(o : Ordine) Post: Ordine.allInstances() → exists(or | or.ID = o.ID) Context OrdineService :: removeOrder(id: Integer) Post: not (Ordine.allIstances()→exists(o| o.id=id)) Context OrdineService findOrdersByCustomer (u: Utente) Post: result= [Ordine.allIIstances()→any(o| o.userID = u.ID)] Context OrdineService findByDate(d: Date) Post: result= Ordine.allInstances()→select (o | o.date= d) context OrdineService::updateOrder(o: Ordine): Boolean post: (Ordine.allInstances()->exists(ord | ord.id = o.id) implies let updatedOrder = Ordine.allInstances()->any(ord | ord.id = o.id) in updatedOrder.data = o.data and updatedOrder.cliente = o.cliente and updatedOrder.totale = o.totale and result = true) and (not Ordine.allInstances()->exists(ord | ord.id = o.id) implies result = false)
Invarianti	

RequestService

REQUEST SERVICE	
Descrizione	Questa classe permette ai client di accedere e manipolare le informazioni sugllle richieste di prodotti e gestione di ordini sul DB
Pre-condizioni	Context RequestService :: addRequest(r : Request) Pre: r <> null

	and r.magazziniereID <> null and r.destinatarioID <> null and (r.statoRichiesta = 'accettato' or r.statoRichiesta = 'non accettato') Context RequestService:: removeRequest(r: Request) Pre: r<>null Context RequestService :: updateRequest(r: RequestService) Pre: r<>null Context RequestService :: findById(id: Long) Pre: id <> null and Request.allInstances()->exists(r | r.id = id) Context RequestService :: cambiaStato(r: Request) Pre: id <> null and Request.allInstances()->exists(r | r.id = id) Context RequestService :: findByMagazziniere (idMagazziniere: Long) Pre: id <> null and Magazziniere.allInstances()->exists(m | m.id = idMagazziniere) Context RequestService :: findByDestinatario(idDestinatario: Long) Pre: idDestinatario <> null and (Fornitore.allInstances()->exists(f | f.id = idDestinatario) or GestoreOrdini.allInstances()->exists(g | g.id = idDestinatario))
Post-condizioni	Context RequestService :: addRequest(r : Request) Post: Request.allInstances()->includes(r) Context RequestService :: removeRequest(r : Request) Post: not Request.allInstances()->includes(r) Context RequestService :: updateRequest(r: RequestService) Post: Request.allInstances()->exists(req | req.id = r.id and req = r) Context RequestService :: findById(id: Long) Post: result <> null implies Request.allInstances()->exists(r | r.id = id and r = result) Context RequestService :: cambiaStato(r: Request) Post: r.stato <> r.stato@pre Context RequestService :: findByMagazziniere (idMagazziniere: Long) Post: result->forall(r | r.magazziniereID = idMagazziniere) Context RequestService :: findByDestinatario(idDestinatario: Long) Post: result->forall(r | r.destinatarioID = idDestinatario)
Invarianti	