

**Future Forge Gear
Object Design Document
Versione 1.0**



Data: 22/12/2025

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Francesco Mascolo	0512117352
Francesco Lamanna	0512117166
Luigi Mazza	0512118612

Scritto da:	Francesco Lamanna
--------------------	-------------------

Revision History

Data	Versione	Descrizione	Autore
15/12/2025	0.1	Creazione Del documento	Full Team
18/12/2025	0.2	Aggiunta Product Managment e OrderService	Francesco Lamanna
20/12/2025	0.3	Aggiunta RequestService	Francesco Mascolo
22/12/2025	1.0	Revisione del documento	Full Team

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

Indice

Introduzione	4
Object Design trade-offs	4
Robustezza VS Tempo	4
Attendibilità VS Tempo.....	4
Linee Guida	4
Riferimenti	4
Directory.....	4
Src.....	5
Database	5
Control.....	5
Control.CartControl.....	5
Control.OrderControl	5
Control.RequestControl.....	6
Control.ProductControl	6
Packages	7
OrderManagement.....	7
UserManagement.....	7
ProductManagement	8
RequestManagement	8
Interfacce di classe	9
Catalogo	9
CartService	10
OrderService.....	11
RequestService	12

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

INTRODUZIONE

Il presente documento di Object Design Document (ODD) descrive le principali scelte di progettazione a livello di oggetti adottate per il sistema Future Forge Gear.

Il documento ha lo scopo di illustrare i trade-off, le linee guida di sviluppo e l'organizzazione del codice.

Object Design trade-offs

Robustezza VS Tempo

Per il sistema Future Forge Gear si è scelto di privilegiare una prima forma di robustezza, concentrata principalmente sul controllo degli input e sulla gestione dei casi di errore più significativi.

Attendibilità VS Tempo

L'attendibilità rappresenta un requisito necessario per il corretto funzionamento del sistema.

Per questo motivo l'attendibilità è garantita per tutti i metodi messi a disposizione dal sistema durante la sua esecuzione, assicurando un comportamento coerente e affidabile delle funzionalità principali.

Linee Guida

Al fine di mantenere coerenze e uniformità nello sviluppo del codice, sono state adottate le seguenti convenzioni:

- Le interfacce sono denominate utilizzando il suffisso “-Interface”.
- La gestione degli errori avviene tramite eccezioni controllate.
- Le classi responsabili dell'interazione con il database utilizzano il suffisso “Service”.
- Le operazioni CRUD sono denominate secondo la convenzione “CrudOperationEntity”
- I metodi di recupero dei dati utilizzano il prefisso “findBy”, seguito dal criterio di ricerca.

Riferimenti

Le scelte progettuali descritte nel presente documento fanno riferimento ai seguenti documenti:

- RAD_FutureForgeGear
- SDD_FutureForgeGear

Directory

Nella prima versione di implementazione del sistema Future Forge Gear, la struttura del progetto è organizzata in directory e package, ciascuno con un ruolo ben definito.

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

Src

La directory src contiene il codice Java necessario all'implementazione del sistema.

Database

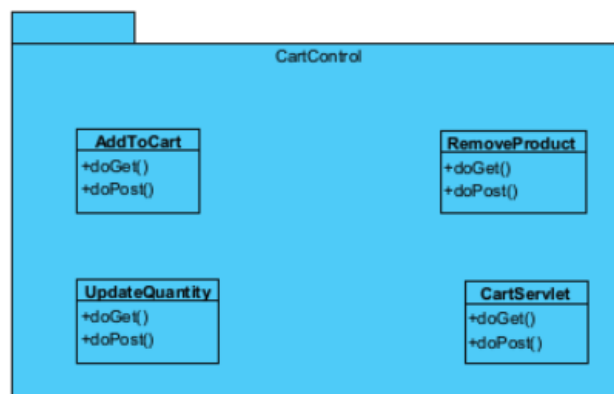
La directory Database contiene lo schema del database e le risorse per la gestione della persistenza dei dati.

Control

Il package Control contiene le Servlet necessarie all'interazione tra l'utente ed il sistema.

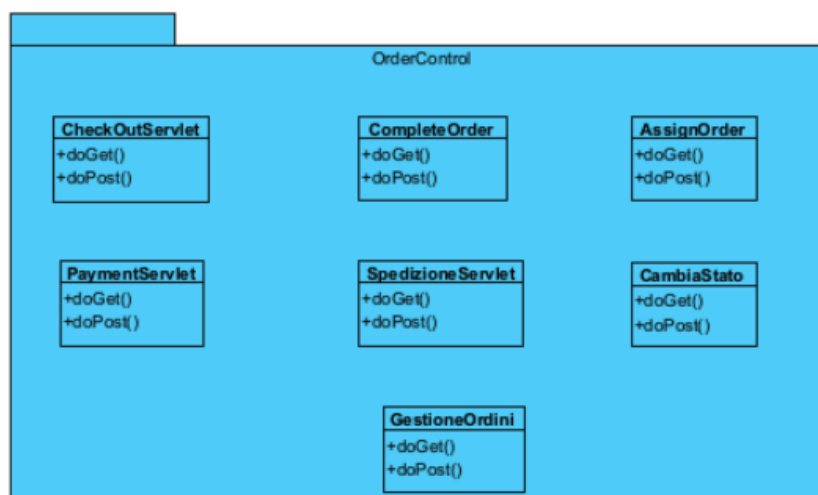
Control.CartControl

Il package Control.CartControl contiene le servlet responsabili della coordinazione delle operazioni relative alla gestione del carrello, quali l'aggiunta e la rimozione dei prodotti, la visualizzazione del contenuto del carrello e l'aggiornamento delle quantità selezionate.



Control.OrderControl

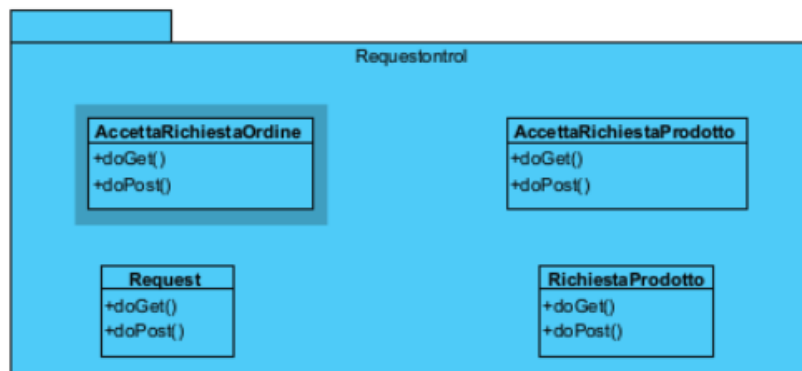
Il package Control.OrderControl contiene le servlet responsabili della coordinazione delle operazioni relative al completamento dell'acquisto, inclusa la gestione della fase di checkout e la conferma dell'ordine.



Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

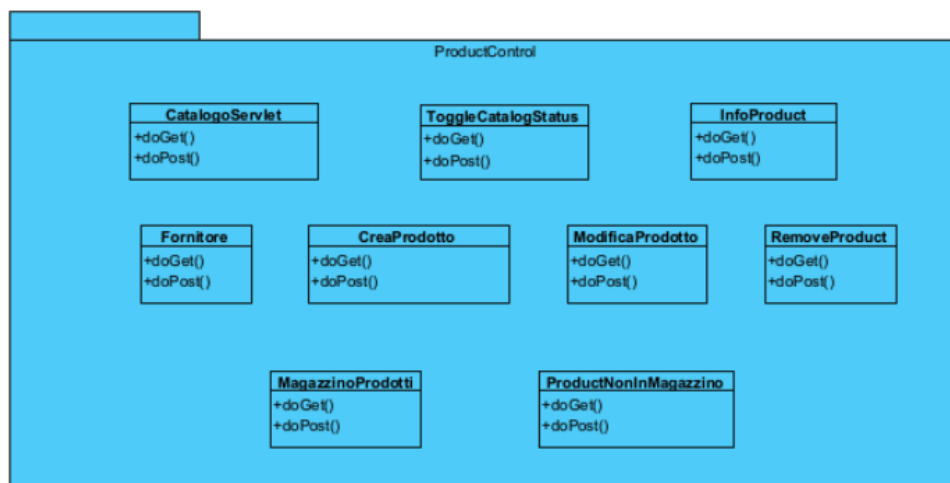
Control.RequestControl

Il package Control.RequestControl contiene le servlet responsabili della gestione delle richieste verso i fornitori.



Control.ProductControl

Il package Control.ProductControl contiene le servlet responsabili della gestione delle operazioni relative ai prodotti.

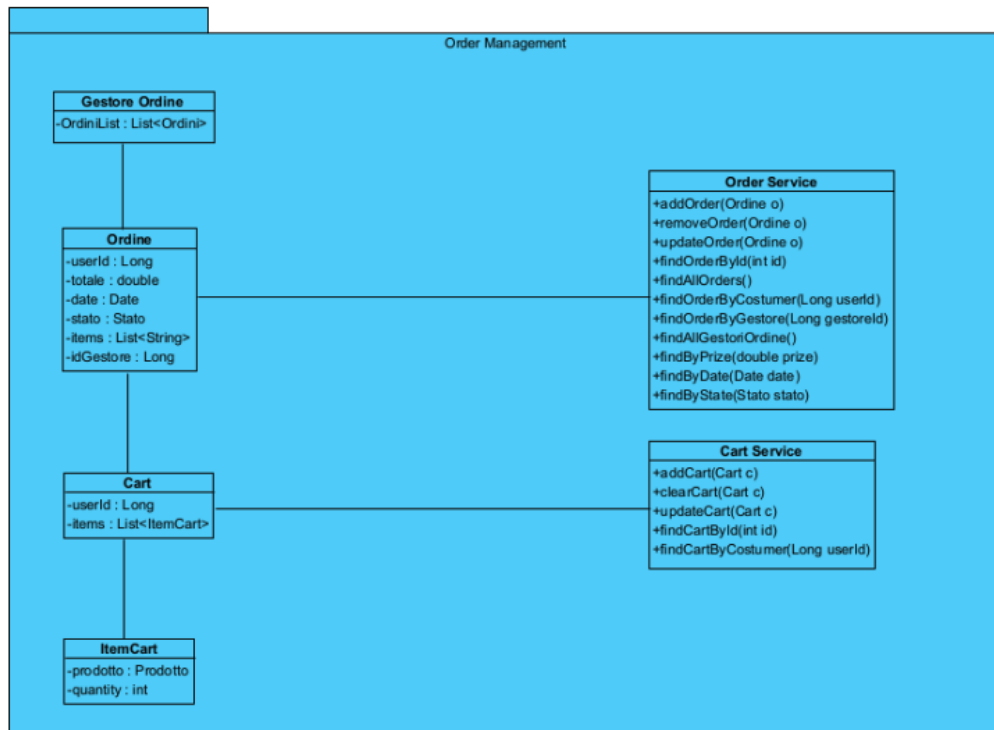


Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

Packages

OrderManagement

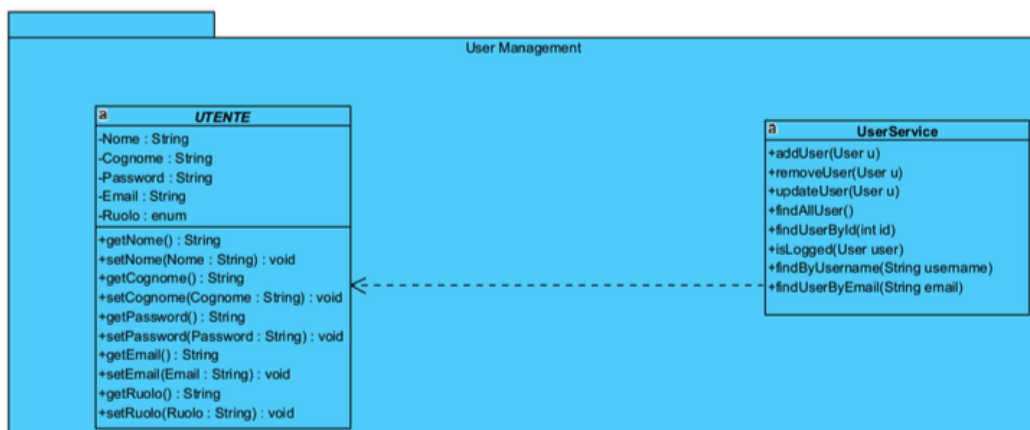
Il package OrderManagement contiene le classi Bean e Service dedicate alla gestione e alla finalizzazione degli acquisti effettuati dai clienti.



UserManagement

Il package UserManagement contiene le classi Bean e Service dedicate alla gestione delle informazioni degli utenti e dei relativi account.

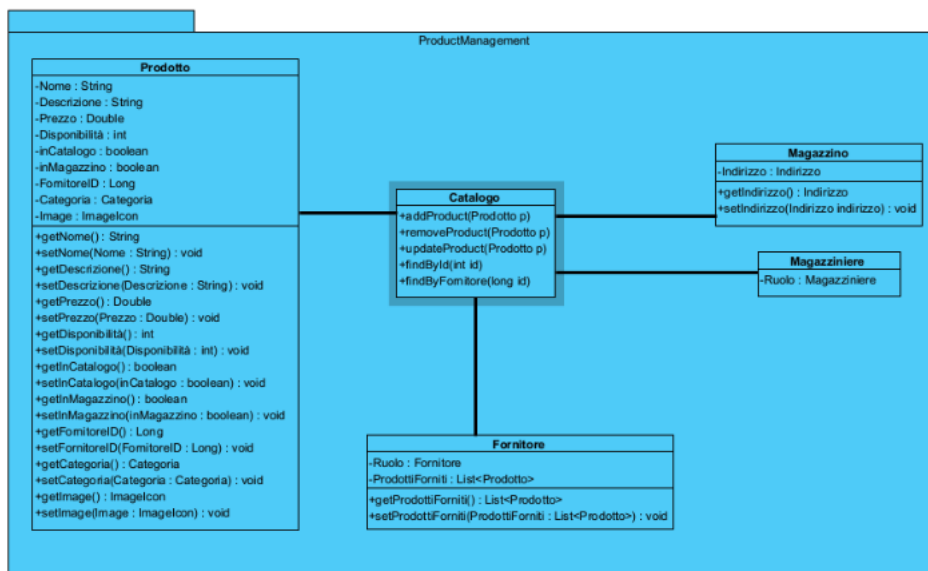
L'entità User è definita come classe astratta, all'interno del package sono presente delle implementazioni dei diversi tipi di utente previsti dal sistema.



Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

ProductManagement

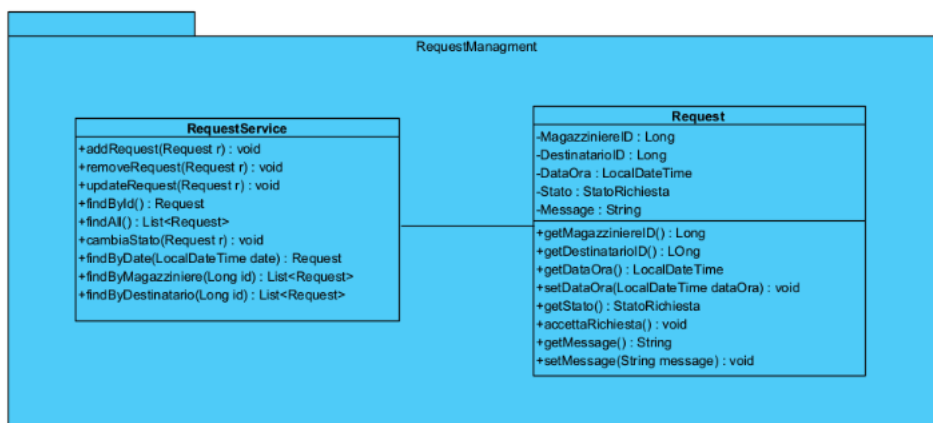
Il package ProductManagement contiene le classi Bean e Service dedicate alla gestione delle richieste, sia relative ai prodotti sia alla gestione degli ordini effettuate dai magazzinieri.



RequestManagement

Il package Request Management contiene le classi Bean e Service dedicate alla gestione delle richieste, sia relative ai prodtti sia alla gestione degli ordini effettuati dai magazzinieri.

L'entità Request è definita come classe astratta, all'interno del package sono presenti le implementazioni delle diverse tipologie di richiesta.



Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

INTERFACCE DI CLASSE

CATALOGO

CATALOGO	
Descrizione	Questa classe, interfacciandosi con il database, consente di effettuare operazioni di ricerca sui prodotti presenti nel magazzino.
Pre-condizioni	<pre> 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 </pre>
	<pre> 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 </pre>

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

Post-condizioni	<pre> 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) </pre>
Invarianti	

CartService

CartService	
Descrizione	Questa classe, interfacciandosi con il database, gestisce la persistenza del carrello tra diverse sessioni utente.
	<pre> 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) </pre>

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

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

OrderService

OrderService	
Descrizione	Questa classe consente ai clienti di accedere e manipolare le informazioni relative agli ordini memorizzati nel database.
Pre-condizioni	<p>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</p> <p>Context OrdineService :: removeOrder(id: Integer) Pre: id<>null and id>0 and Ordine.allInstances()->exists(o o.id=id)</p> <p>Context OrdineService findOrdersByCustomer (u: Utente) Pre: u<>null</p> <p>Context OrdineService findByDate(d: Date) Pre: d<>null</p>

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

	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.allInstances() -> exists(o o.id=id)) Context OrdineService findOrdersByCustomer (u: Utente) Post: result= [Ordine.allInstances() -> 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)
Invariante	

RequestService

RequestService	
Descrizione	Questa classe consente ai clienti di accedere e manipolare le informazioni relative alle richieste prodotti e alla gestione degli ordini memorizzate nel database.
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)

Progetto: Future Forge Gear	Versione: 1.0
Documento: Object Design	Data: 18/12/2025

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