Università degli Studi di Salerno
Corso di Ingegneria del Software

HomeDecore
Object Design Document
Versione 1.2

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

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

Src

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

DB

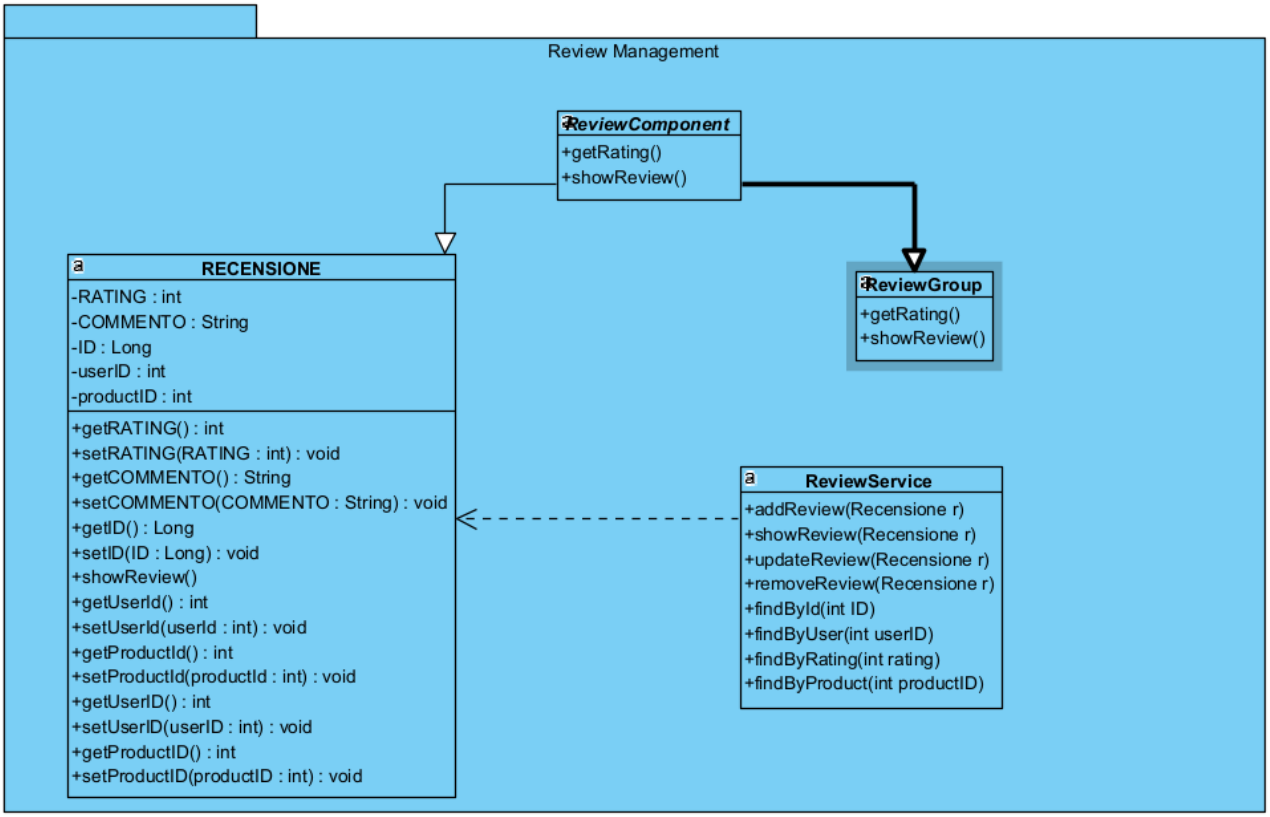
Questa directory conterrà lo schema del database

Packages

View

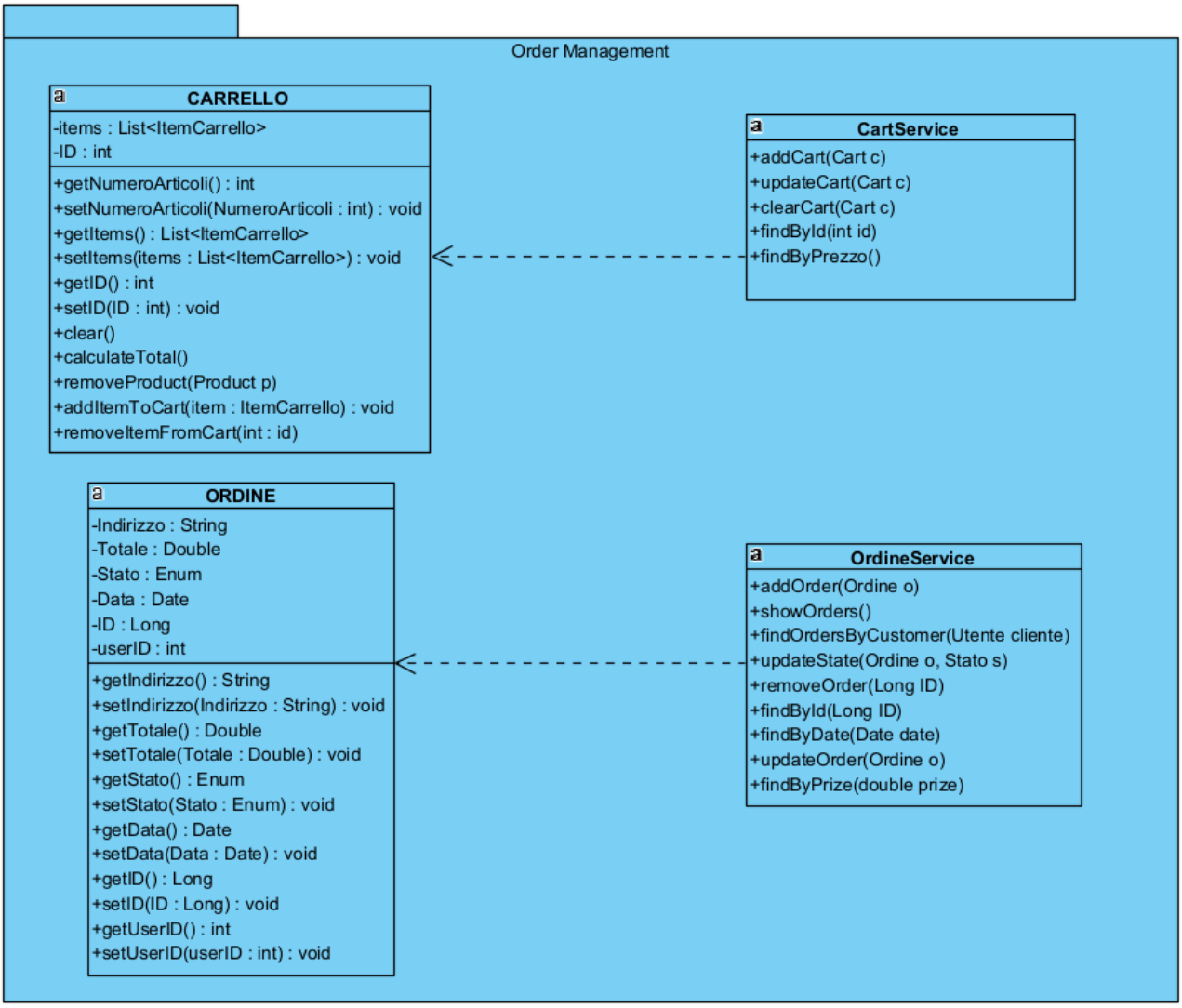
ReviewManagement

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



OrderManagement

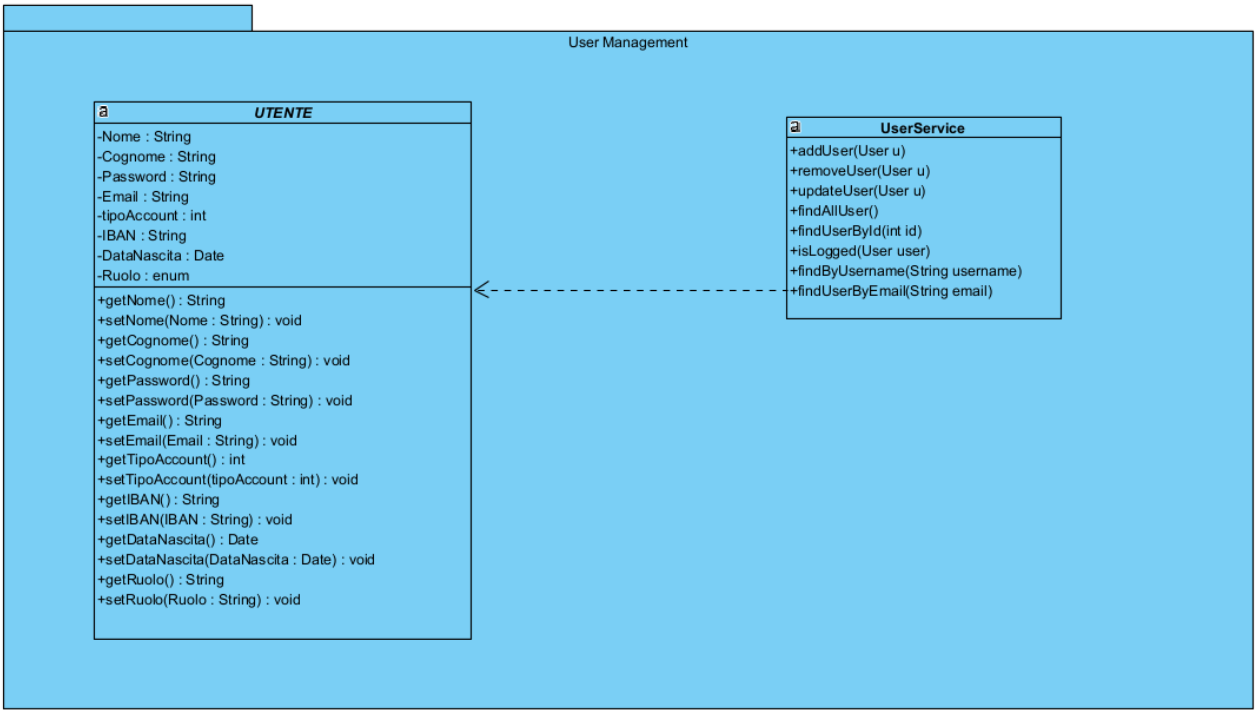
Questo package contiene le classi Bean e Service a cui viene affidata la gestione e finalizzazione degli acquisti del cliente.



Cart deve avere riferimento all'utente? E viceversa?
Clear deve essere un metodo di CartService?

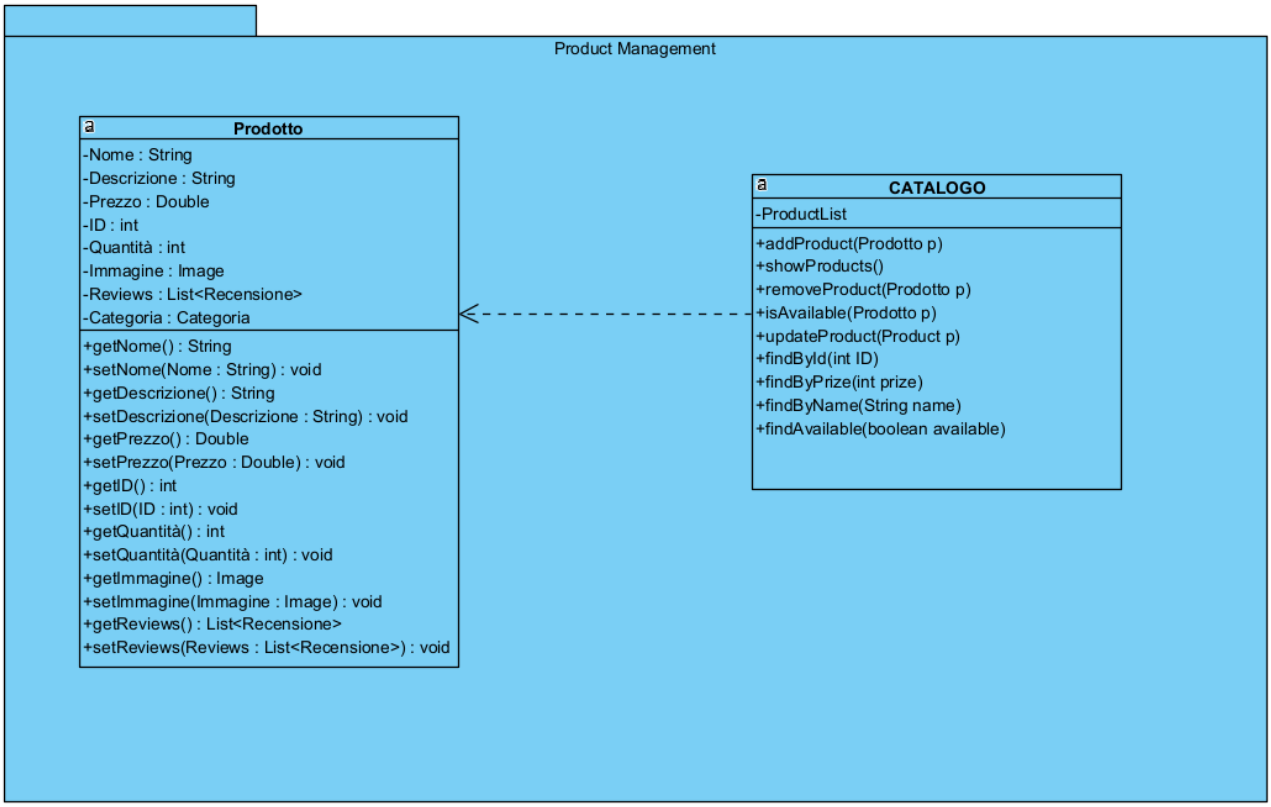
UserManagement

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.
// !!! AGGIUNGERE MAGAZZINO NEL PRODUCT MANAGEMENT PER EFFETTUARE LE MODIFICHE



PRODOTTO non dovrebbe avere la lista di recensioni associata, è compito di reviewService associare la recensione al prodotto.
updateProduct è un metodo di Magazzino? O deve averlo anche Catalogo per aggiornare il prodotto dopo che il magazzino lo aggiorna?
Prodotto dovrebbe avere l'attributo categoria (enum?)
GetProducts più evocativo di showProducts?

ChatManagement

Interfacce di classe

Product Management

Catalogo	
Descrizione	Questa classe, interfacciandosi con il DBMS, permette di effettuare operazioni di ricerca sui prodotti presenti nel magazzino
Pre-Condizione	**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)

Post-Condizione	**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)
Invarianti	

Magazzino

Magazzino	
Descrizione	Questa classe, interfacciandosi con il DBMS, permette di effettuare operazioni di modifica sui prodotti del magazzino
Pre-Condizione	**context** Magazzino::addProduct(prodotto:Prodotto) **pre**: prodotto<>null and not self.productList->includes(prodotto) **context** Magazzino::removeProduct(prodotto:Prodotto) **pre**: prodotto<>null and self.productList->includes(prodotto) **context** Magazzino::modifyProduct(prodotto:Prodotto) **pre**: prodotto<>null and self.productList->includes(prodotto)
Post-Condizione	**context** Magazzino::addProduct(prodotto:Prodotto) **post**: self.productList=self@pre.productList->including(prodotto) **context** Magazzino::removeProduct(prodotto:Prodotto) **post**: self.productList=self@pre.productList->excluding(prodotto) **context** Magazzino::modifyProduct(prodotto:Prodotto) **post**: let oldProduct = self.productList ->select(p | p.ID = prodotto.ID)->first() self.ProdottiList = self.productList @pre->excluding(oldProduct)->including(prodotto)
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	

ChatService

CHAT SERVICE

Descrizione	Questa classe, si interfaccia con l'entità chat inclusa nel database, permette di effettuare operazioni riguardo la gestione della chat.
Pre-condizioni	**Context** ChatService::AvviaChat(UserSender sender,UserReceiver receiver) **pre**: sender <> null and receiver <> null and sender <> receiver Context ChatService::sendMessage(String message) pre: message <> null and message <> " " **context** ChatService::DisplayMessage(String text,UserSender sender,UserReceiver receiver) **pre**: text<>null and text<>" " and sender <>null and receiver <>null **Context** ChatService::SelezionaUtente(UserSender user,String nomeUser) **pre**: user<>null
Post-condizioni	**Context** ChatService::AvviaChat(UserSender sender,UserReceiver receiver) **post**: sender=sender and receiver=receiver and text=" " **Context** ChatService::sendMessage(message: String) **post**: let savedMessage : Message = MessageService.save(message) in savedMessage <> null and let retrievedMessage : Message = MessageService.getMessage(savedMessage) in retrievedMessage = savedMessage and display(retrievedMessage) **Context** ChatService::DisplayMessage(String text,UserSender sender,UserReceiver receiver) **post**: sender=display(message) and receiver= display(message) **Context** ChatService::SelezionaUtente(user: UserSender,String user) **post**: let listaUtenti : List(User) = UserService.ListaUtenti(user) in listaUtenti <> null and not listaUtenti->includes(user) and listaUtenti->includes(user)
Invarianti	

MessageService

MESSAGE SERVICE	
Descrizione	Questa classe, si interfaccia con l'entità message inclusa nel database, permette di effettuare operazioni riguardo la gestione dei messaggi.
Pre-condizioni	**Context** MessageService::saveMessage(message: Message, sender: UserSender)

	pre: message.message.size() > 0 -- Il messaggio non deve essere vuoto and messagesPerUser->includesKey(sender) **Context** MessageService::deleteMessage(message: Message, sender: UserSender) **pre:** messagesPerUser->includesKey(sender) -- Il sender deve esistere nella mappa and messagesPerUser->at(sender)->includes(message) **Contex**t MessageService::retrieveMessages(user: UserSender): **pre:** messagesPerUser->includesKey(user) **Context** MessageService::retrieveMessage(user: UserSender id: Integer): Message **pre:** messagesPerUser->includesKey(user) and messagesPerUser-> get(user)<>null
Post-condizioni	**Context** MessageService::saveMessage(message: Message, sender: UserSender) **post**: messagesPerUser->includesKey(sender) and messagesPerUser->at(sender)->includes(message) and messagesPerUser->at(sender)->size() = messagesPerUser@pre->at(sender)->size() + 1 **Context** MessageService::deleteMessage(message: Message, sender: UserSender) post: messagesPerUser->at(sender)->excludes(message) and messagesPerUser->at(sender)->size() = messagesPerUser@pre->at(sender)->size() - 1 **Contex**t MessageService::retrieveMessages(user: UserSender): **post:**result in ListaMess=messagesPerUser->at(user) **Context** MessageService::retrieveMessage(user: UserSender id: Integer): Message **post:** messagesPerUser->at(sender)->get(UserSender) in listaMess() and messageTrovato=listaMess()->includes(id) and return messageTrovato
Invarianti	