Object Design Document

1. Introduction

1.1 Object design trade-offs

Comprensibilità vs Tempo:

Essendo i tempi di sviluppo del sistema limitati, NON sarà una priorità commentare dettagliatamente il codice di ogni metodo per permettere l'implementazione del sistema in tempi più contenuti.**Di conseguenza, la fase di testing ed eventuali future modifiche saranno più complesse.**

Riusabilità vs Costi:

Lo sviluppo del sistema non prevede l'utilizzo di librerie o componenti a pagamento, essendo il progetto privo di budget economico.

Sicurezza vs Efficienza:

Dati i tempi piuttosto ristretti per lo sviluppo, ci limiteremo ad implementare sistemi di sicurezza basati su username e password crittografata, garantendo un controllo sugli accessi basato su determinati ruoli.

Incapsulamento vs Efficienza:

Il sistema garantisce la segretezza sui dettagli implementativi delle classi grazie al vincolo che limita la manipolazione degli attributi di una classe esclusivamente mediante i suoi metodi.

Trasparenza vs Efficienza:

Sulla gestione della persistenza, ci affideremo ad un Database, che ci offre un DBMS per poter interagire con i nostri dati in maniera consistente e transazionale a discapito di una lieve perdita di efficienza. Utilizzeremo un DataSource per poter ottenere connessioni al database, gestione della sincronizzazione e interazione facilitata con il DBMS.

1.2 Interface documentation guidelines

guidelines and conventions (e.g., naming conventions, boundary cases, exception handling mechanisms) and an overview of the document.
Interface documentation guidelines and coding conventions a

1.3 Definitions, acronyms, and abbreviations

1.4 References

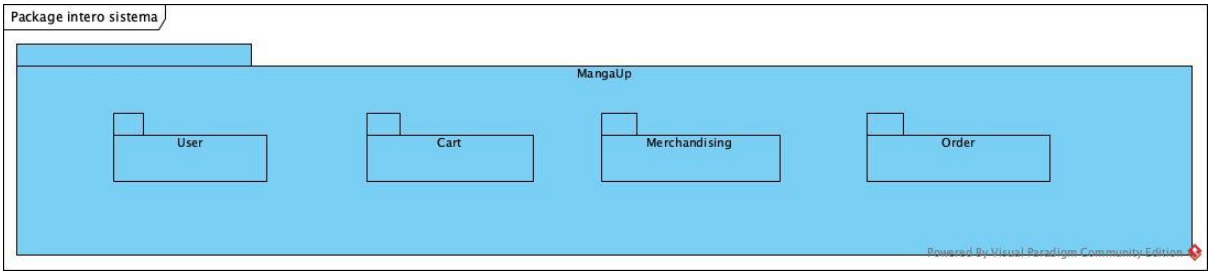
System Design Document(SDD)

Per ulteriori informazioni è possibile visualizzare questi documenti nella cartella Deliverables del progetto

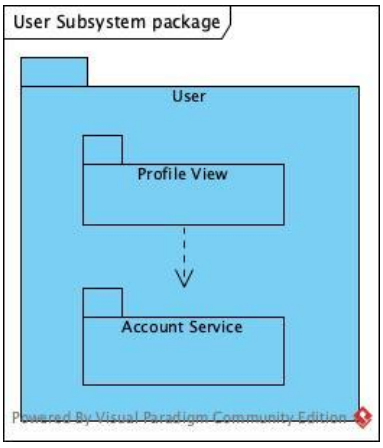
2. Packages

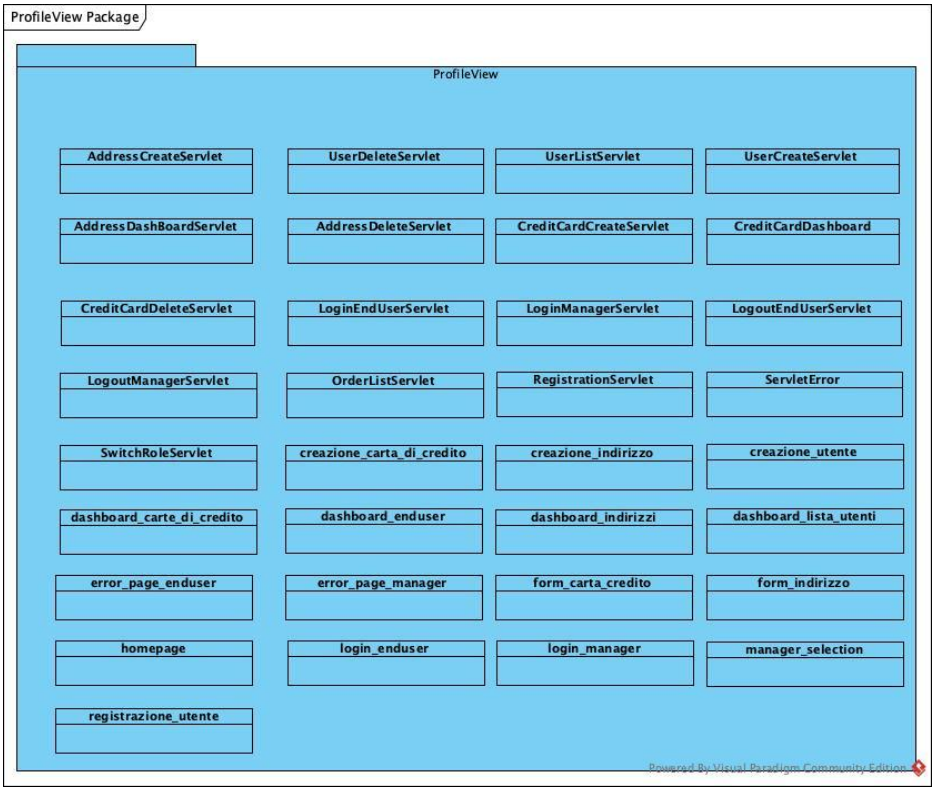
Nella presente sezione si mostra la struttura del package principale di MangaUP. La struttura generale è stata ottenuta a partire dalla divisione in partizioni e sottosistemi che sono stati individuati in fase di System Design come mostrato nel System Design Document (SDD). La struttura è quindi basata su un'architettura divisa su tre livelli e quindi ogni classe individuata sarà associata ad uno dei seguenti ruoli:

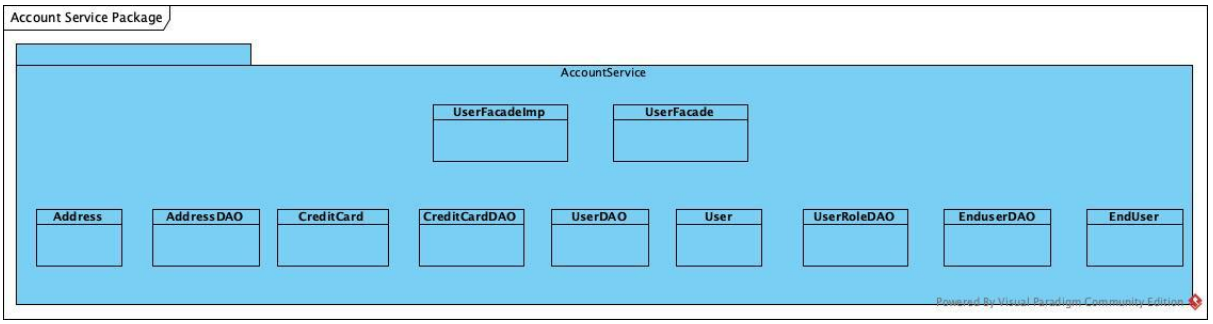
- Presentazione dati del modello
- Controllo delle attività del processamento interno del sistema
- Manipolazione dati dell'Application Domain.



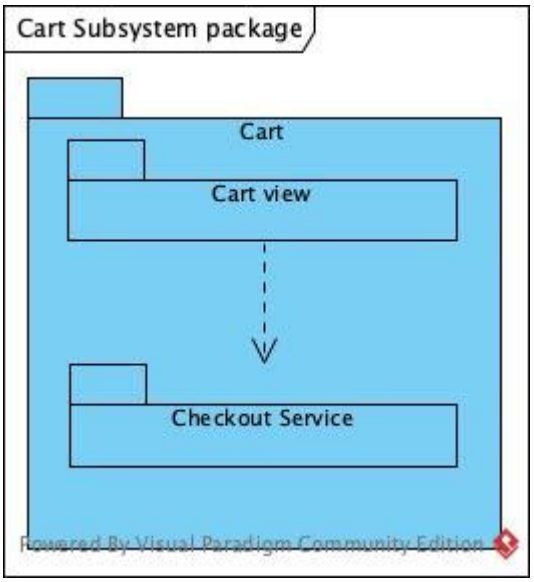
Partizione User

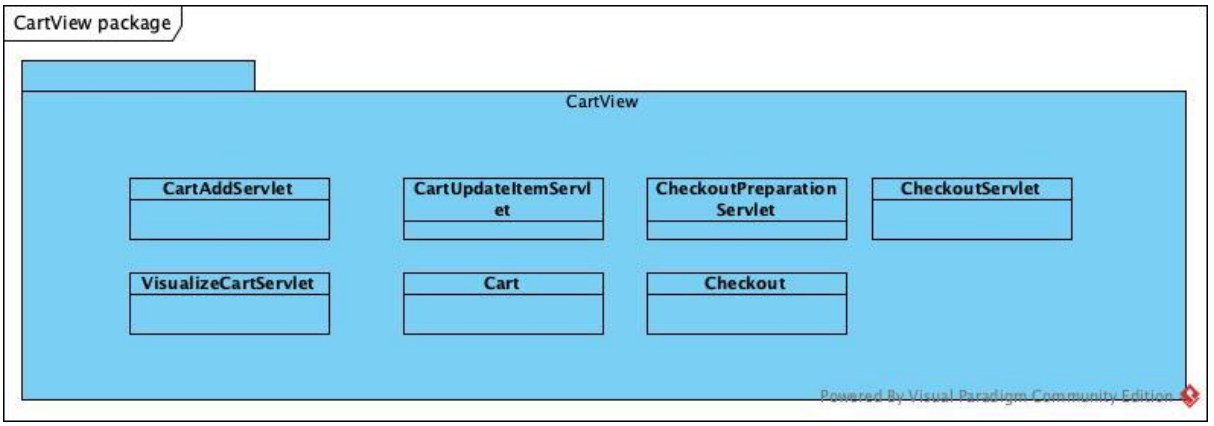


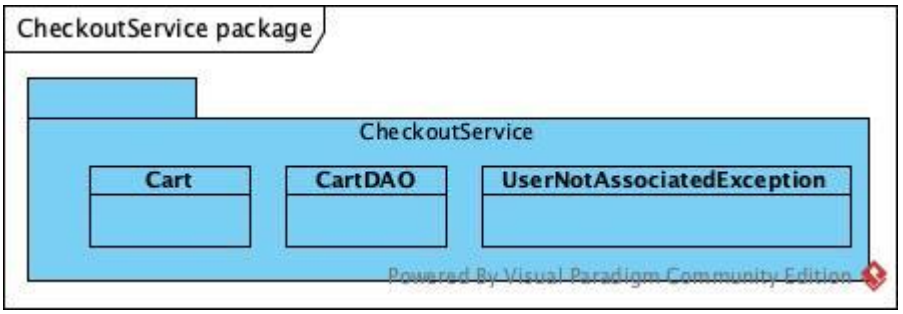




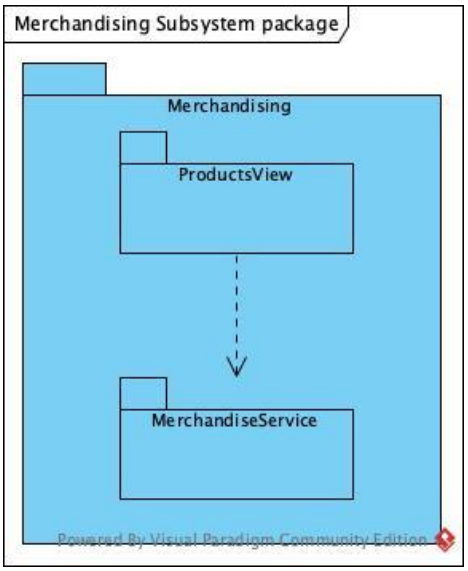
Partizione Cart

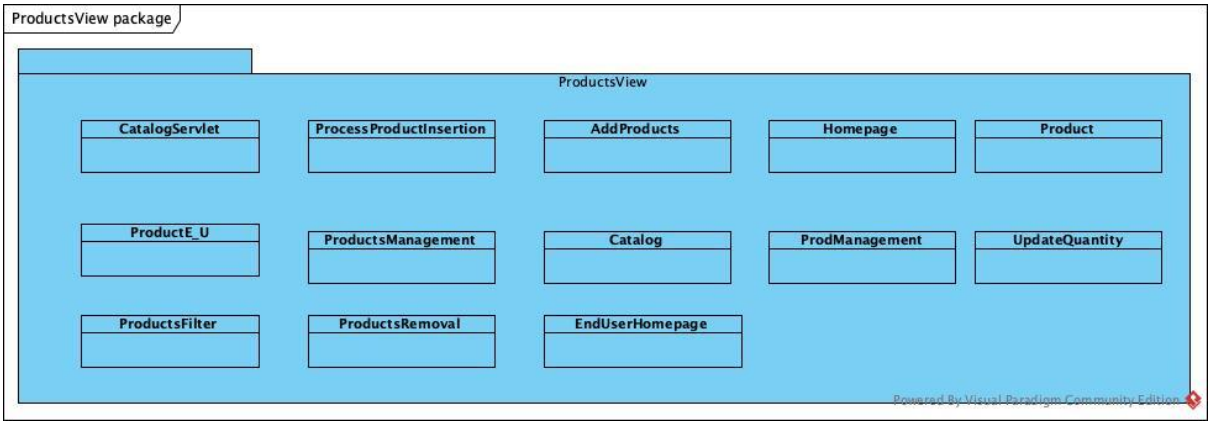


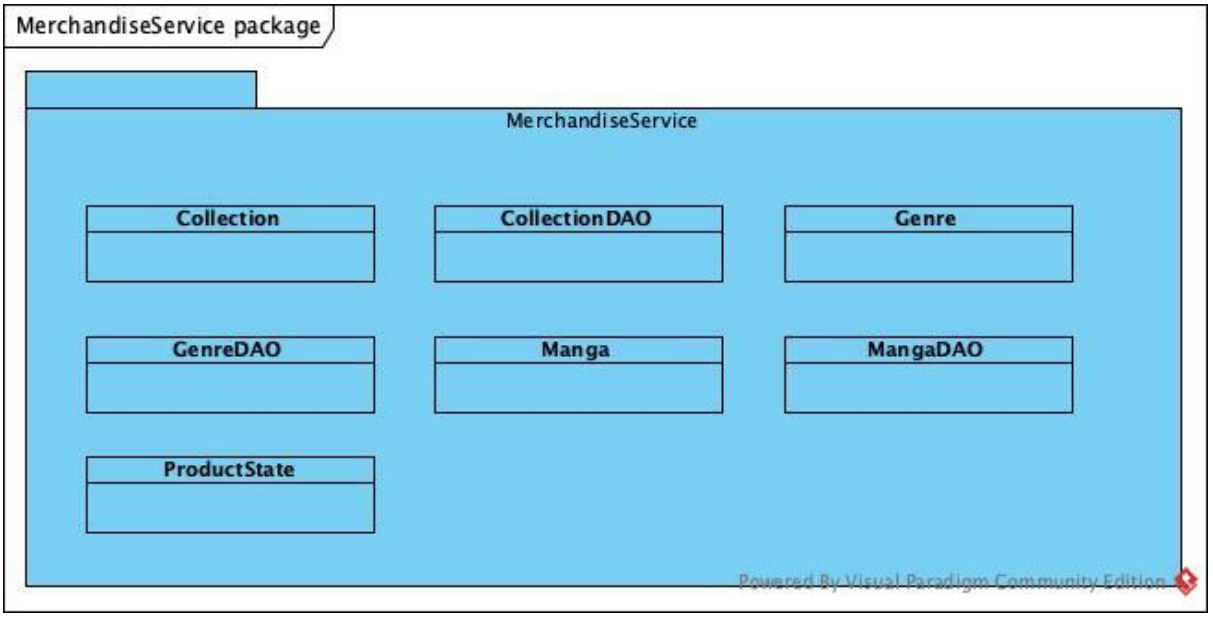




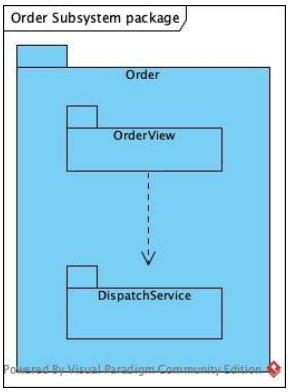
Partizione Merchandising

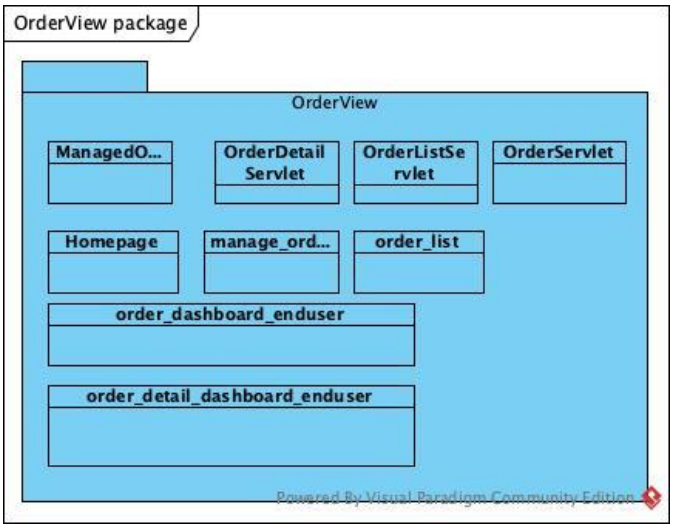


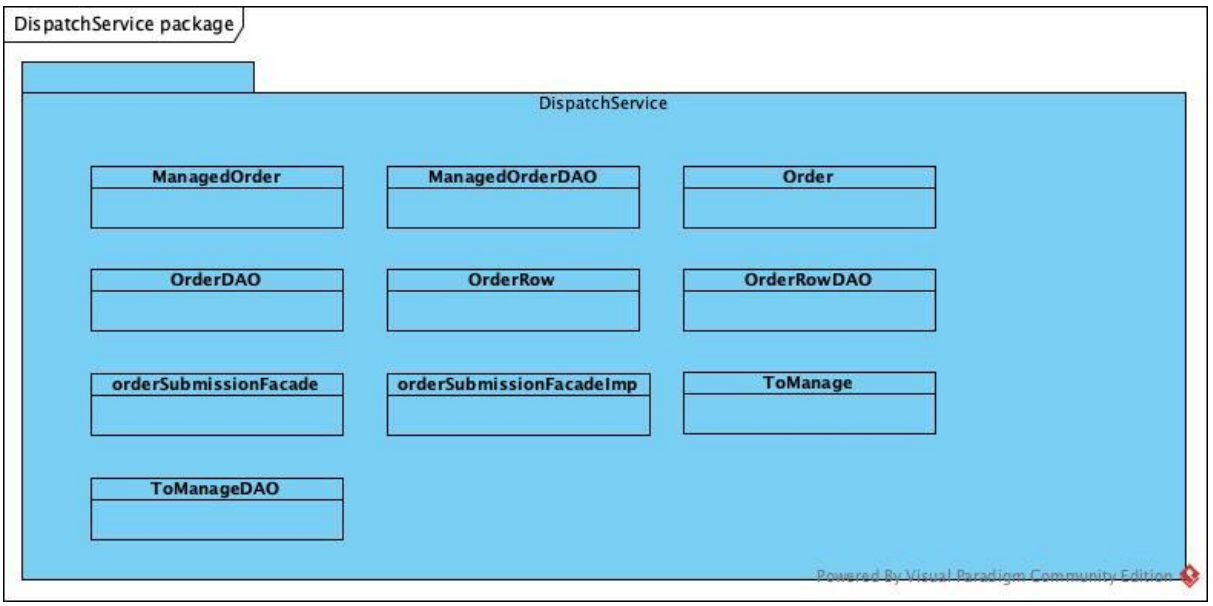




Partizione Order







3. Class interfaces

Proponiamo degli operatori ad hoc che consentono di utilizzare le regular expressions in OCL.
Tra gli operatori che definiamo vi è:

- isNull(): Boolean: restituisce true se l'oggetto stesso è di tipo null o invalido.
- x.subString(y): Boolean: restituisce true se y è sottostringa di x; false altrimenti.
- x.validate(): Boolean : restituisce true se x è del formato corretto secondo quanto definito nel R.A.D

Per semplicità quando viene usato il termine "Database" nelle nostre pre e post-condizioni, ci riferiamo ad un'astrazione che rappresenta il nostro Database dell'applicazione in un dato istante, le cui tabelle sono classi in relazione(1..*, 1,1) con esso.

3.1 Account Service

Nome Classe	UserDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono gli account degli utenti gestori
Metodi	+createUser(user: User) +removeUserByUsername(username: String) +getAllUsers(role: String): Collection +getAllBeginnerOrderManagers(): Collection +getTargetOrderManagerUserName(): String +getRoles(username: String): Collection +checkUser(user: User, roleToLog: String): Boolean +existsUsername(username: String): Boolean
Invariante di classe	self.ds != null

Nome Metodo	+createUser(user: User) void
Descrizione	Questo metodo consente di inserire un nuovo user gestore
Pre-condizione	**context:** UserDAO::createUser(user: User) **pre:** user != NULL AND user.username != NULL AND user.password!=null AND user.username.validate() AND user.password.validate() AND Database.users->!exists(User | User.username == user.username)
Post-condizione	**context:** UserDAO::createUser(user: User)

	post: Database.users->include(user)
Nome Metodo	+removeUserByUsername(username: String)
Descrizione	Questo metodo consente di rimuovere un gestore dal DB.
Pre-condizione	**context:** UserDAO::removeUserByUsername(username: String) **pre:** username != NULL AND Database.users-> exists(u | User.username == user.username)
Post-condizione	**context:** UserDAO::removeUser(user : User) **post:** Database.users->!include(user)
Nome Metodo	+getAllUsers(role : String): Collection
Descrizione	Questo metodo consente di ritornare tutti gli utenti di uno specifico ruolo e se l'input non è specificato vi è una restituzione di tutti gli utenti.
Pre-condizione	**context:** UserDAO::getAllUsers(role : String): Collection **pre:** true

Post-condizione	**context:** UserDAO::getAllUsers(role: Role): Collection
	
	post:
	if role == null
	then
	result = Database.users
	else
	result = Database.users->select(u | u.role == role)
	endif

Nome Metodo	+getAllBeginnerOrderManager(): Collection
Descrizione	Questo metodo consente di ritornare tutti i gestori degli ordini che non hanno mai ancora gestito un ordine
Pre-condizione	**context:** UserDAO::getAllBeginnersOrderManager(): User **pre:** true
Post-condizione	**context:** UserDAO::getAllBeginnersOrderManager(): Collection **post:** result= Database.users->select(u|Database.ToManage->Notexists(t.userId == u.id))
Nome Metodo	+getTargetOrderManagerUserName(): String
Descrizione	Questo metodo restituisce lo username del gestore al quale bisogna assegnare un ordine da gestire(gestore con meno ordini

	gestiti).
Pre-condizione	**context:** UserDAO::getTargetOrderManagerUserName(): String **pre:** true
Post-condizione	**context:** UserDAO::getTargetOrderManagerUserName(): String **post:** result = OrderSet(Database.MANAGES->select(t| Database.MANAGES->select(toM | toM.user_name != t.user_name) -> forAll(t_m | Database.MANAGES->select(l | l.user_name == t_m.user_name).size >= Database.MANAGES->select(k | k.user_name == t.user_name).size)).first().user_name
Nome Metodo	+getRoles(username: String): Collection
Descrizione	Questo metodo restituisce associati di un user con un username.
Pre-condizione	**context:** UserDAO::getRoles(username : String): Collection **pre:** true
Post-condizione	**context:** UserDAO::getRoles(username: String): Collection **post:** result = Database.user_roles->select(u | u.user_name == username)
Nome Metodo	+checkUser(user: User, role:String): Boolean
Descrizione	Questo metodo verifica che lo user abbia il ruolo 'role' associato nel DB.
Pre-condizione	**context:** UserDAO::checkUser(user: User, role: String): Boolean **pre:** true
Post-condizione	**context:** UserDAO::checkUser(user: User, role: String): Boolean **post:** result = Database.users->exists(u | u.username==username AND u.password = password) AND Database.user_role -> exists(o |o.user_name = user.username AND o.role_name = role)
Nome Metodo	+ existsUsername(username: String)
Descrizione	Questo metodo verifica l'esistenza di uno user che abbia un username 'username' del DB.
Pre-condizione	**context:** UserDAO::existsUsername(username: String): Boolean **pre** true
Post-condizione	**post:** result = Database.users->exists(u | u.username==username)

Nome Classe	EndUserDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono gli account degli utenti finali
Metodi	+create(endUser: EndUser): void +existEmail(email: String): bool +login(email: String, password: String): EndUser +findById(id: int): EndUser -find(endUser: Enduser): EndUser -map(set: ResultSet): EndUser
Invariante di classe	/
Nome Metodo	create(endUser: EndUser): void
Descrizione	Questo metodo permette di rendere persistente un oggetto che appartiene alla classi di EndUser
Pre-condizione	**context**: EndUserDAO:: create(endUser: EndUser): void **pre**: endUser != NULL enduser.email != NULL AND enduser.validate() AND enduser.name != NULL AND enduser.name.validate() AND enduser.surname != NULL AND !enduser.surname.validate() AND enduser.birthDate != NULL AND enduser.phoneNumber != NULL AND enduser.phoneNumber.validate() AND enduser.password != NULL AND enduser.password.validate()
Post-condizione	**context**: EndUserDAO:: create(endUser: EndUser): void **post:** Database.users ->include(endUser) AND endUser.id = Database.users-> select(e| e.name = endUser.name AND e.surname AND e.birthDate = endUser.birthDate AND endUser.phoneNumber = e.phoneNumber AND e.password = endUserPassword).first().id

Nome Metodo	+existEmail(email: String): bool
Descrizione	Questo metodo stabilisce se esiste un email associata ad un user nel database.
Pre-condizione	**context:** UserDAO::existEmail(email: String): bool **pre:** email == null OR email.size == 0
Post-condizione	**context:** UserDAO::existEmail(email: String): bool

	post: result = Database.end_user->exists(u | u.email == email)
Nome Metodo	+login(email: String,password: String): EndUser
Descrizione	Questo metodo effettua il login di un endUser.
Pre-condizione	**context:** EndUserDAO::login(username: String,password: String): EndUser **pre:** email == null OR password == null OR email.size == 0 OR password.size == 0
Post-condizione	**context:** EndUserDAO::login(username: String,password: String): EndUser **post:** result = Database.end_user->select(e | e.username == username AND e.password == password).first()
Nome Metodo	+findById(id: int): EndUser
Descrizione	Questo metodo una collezione di ruoli per l'username (associato ad un gestore) passato come parametro
Pre-condizione	**context:** EndUserDAO::findById(id: int): EndUser **pre:** id > 0
Post-condizione	**context:** EndUserDAO::findById(String username): EndUser **post:** result = Database.end_user-> select(u | u.id = id).first()

Nome Classe	AddressDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono gli address associati agli account degli enduser.
Metodi	+create(address: Address):void +delete(address: Address): void +findById(id:int): Address +findAssociatedAddresses(address: Address): Address -find(sql String, values.. Objects): Address -map(set: ResultSet): Address
Invariante di classe	/

Nome Metodo	+create(address: Address): void
Descrizione	Questo metodo restituisce l'utente al quale è associato l'username passato come parametro
Pre-condizione	**context:** AddressDAO::create(Address address): Address **pre:** address != NULL AND address.endUser != NULL AND address.endUser.id >= 0 AND address.country != NULL AND address.city != NULL AND address.street != NULL AND address.postalCode != NULL AND address.phoneNumber != NULL AND address.region != NULL AND address.country.validate() AND address.city.validate() AND address.street.validate() AND address.postalCode.validate() AND address.phoneNumber.validate() AND address.regione.validate() AND Database.user -> exist(u | u.id = address.user.id)
Post-condizione	**context:** AddressDAO::create(Address address): Address **post:** Database.addresses->include(address) AND addres.id = Database.addresses-> select(a | AND address.endUser.id >= 0 AND address.country == a.country AND address.city == a.city AND address.street == a.street AND address.postalCode == a.postalCode AND address.phoneNumber == a.phoneNumber AND address.region == a.region).first().id

Nome Metodo	+delete(Address: address): void
Descrizione	Questo metodo elimina l'indirizzo dal database.
Pre-condizione	**context:** AddressDAO::delete(id: int): void **pre:** address != null AND address.id > 0
Post-condizione	**context:** AddressDAO::delete(id: int): void **post:** result = Database.addresses ->select(address: address.id = id)->isEmpty()

Nome Metodo	+findAssociatedAddress(user: EndUser): Collection
Descrizione	Questo metodo restituisce tutte le carte associate all'account dell'utente.
Pre-condizione	**context:** AddressDAO::findAssociatedAddress(user: EndUser): Collection **pre:** user != null AND user.id >= 0
Post-condizione	**context:** AddressDAO::findAssociatedAddress(user: EndUser): Collection c **post:** result = Database.addresses -> select(a | a.userId = user.id)

Nome Classe	CreditCardDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono le carte di credito degli account degli EndUser.
Metodi	+create(creditCard: CreditCard): void +delete(id: int): void +findById(id:int): CreditCard +findAllByEndUser(id: int): CreditCard
Invariante di classe	/

Nome Metodo	+create(creditCard: CreditCard): void
Descrizione	Questo metodo rende persistente l'oggetto address associandolo all'account dell'endUser.
Pre-condizione	**context:** CreditCardDAO::create(creditCard: CreditCard): void **pre:** creditCard != NULL AND creditCard.id = 0 AND creditCard.cvc != NULL AND creditCard.cardHolder != NULL AND creditCard.expirationDate != NULL AND creditCard.cardNumber != NULL AND creditCard.cvc.validate() AND creditCard.cardHolder.validate() AND creditCard.expirationDate.validate() AND creditCard.cardNumber.validate()
Post-condizione	**context:** CreditCardDAO::create(creditCard: CreditCard): void

	post: Database.creditCards -> includes(creditCard)

Nome Metodo	+delete(id:int)
Descrizione	Questo metodo elimina la carta di credito dalla persistenza.
Pre-condizione	**context:** CreditCardDAO::create(creditCard: CreditCard): void **pre:** id != 0
Post-condizione	**context:** CreditCardDAO::create(creditCard: CreditCard): void **post:** Database.creditCards ->select(creditCard: CreditCard.id = id)->isEmpty()

Nome Metodo	+findAllByEndUser(id:int)
Descrizione	Questo restituisce la lista di carte di credito associate all'account.
Pre-condizione	**context:** CreditCardDAO::findAllByEndUser(id: int): Collection **pre:** id != 0
Post-condizione	**context:** CreditCardDAO::findAllByEndUser(id: int): Collection **post:** result = Database.addresses -> select(c : CreditCard| c.userId = id)

Nome Classe	UserFacadeImp
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono i prodotti di MangaUP
Metodi	+registration(endUser : EndUser): void +managerEngagement(): User - getRandomOrderManager(orderManagers: Collection): User

Invariante di classe	self.ds != null;
	self.userDAO != null;
	self.creditCardDAO != null;
	self.endUserDAO != null;
	self.addressDAO != null;

Nome Metodo	-getRandomOrderManager(orderManagers: Collection): User
Descrizione	Questo metodo seleziona un utente random dalla collezione di orderManagers.
Pre-condizione	**context:** UserFacadeImp:: getRandomOrderManager(orderManagers: Collection): Collection **pre:** true
Post-condizione	**context:** UserFacadeImp:: getRandomOrderManager(orderManagers: Collection): Collection **post:** result = orderManagers.random()

Nome Metodo	+managerEngagement(): User
Descrizione	Questo metodo seleziona uno tra gli utenti con il minor numero di ordini gestiti.
Pre-condizione	**context:** UserFacadeImp:: managerEngagement(): User **pre:** true

Post-condizione	**context:** UserFacadeImp:: managerEngagement(): User
	
	post:
	
	var noManaged = Database.User->select(u|
	Database.Manages->!contains(u))
	
	if(noManaged.size()>0)
	
	then
	
	result = getRandomOrderManager(noManaged)
	
	else
	
	result = Database.User(t | Database.Manages->select(
	toM | toM.user_name != t.user_name)->forAll(
	t_m | Database.Manages->select(
	l | l.user_name == t_m.user_name).size >=
	Database.Manages->select(
	k | k.user_name == t.user_name).size).first()

Nome Metodo	+registration(endUser : EndUser): void
Descrizione	Questo metodo seleziona uno tra gli utenti con il minor numero di ordini gestiti.
Pre-condizione	**context:** UserFacadeImp:: registration(endUser: EndUser): Collection **pre:** endUser != null AND endUser.addresses != NULL AND endUser.addresses.size > 0 endUser.cards != NULL AND endUser.cards.size > 0 endUser.name != NULL AND endUser.validate() AND endUser.cards.first() != NULL endUser.cards.first().cvc.validate() AND endUser.cards.first().cardHolder.validate() AND endUser.cards.first().expirationDate.validate() AND endUser.cards.first().cardNumber.validate() AND AND endUser.cards.first().expirationDate > currentDate AND endUser.addresses.country.validate() AND endUser.addresses.first().city.validate() AND endUser.addresses.first().street.validate() AND endUser.addresses.first().postalCode.validate() AND endUser.addresses.first().phoneNumber.validate() AND endUser.addresses.first().regione.validate() endUser.addresses.first().
Post-condizione	**context:** UserFacadeImp:: registration(endUser:EndUser):Collection **post:** Database.EndUser->exists(u|endUser.id=u.id) AND Database.Address->exists(a| a.usr_id==endUser.id) AND Database.CreditCard->exists(c|c.usr_id==endUser.id)

Nome Classe	UserRoleDAO
Descrizione	Questa classe permette di tenere gestire i ruoli associati ad un gestore.
Metodi	+getRoles(user: User): Collection +setRoles(user: User, roles: Collection): void

	+removeAllRoles(user: User): void
Invariante di classe	self.ds != null

Nome Metodo	+getRoles(user: User): Collection
Descrizione	Questo metodo seleziona un l'insieme dei ruoli associati allo User "user"
Pre-condizione	**context:** UserRoleDAO:: getRoles(user:User): Collection **pre:** user != null AND user.username != null
Post-condizione	**context:** UserRoleDAO:: getRoles(user:User): Collection **post:** result = Database.User_Roles->select(u_r | u_r.user_name==user.user_name)

Nome Metodo	+setRoles(user: User, roles: Collection): void
Descrizione	Questo metodo imposta allo User "user" tutti i ruoli contenuti in "roles"
Pre-condizione	**context:** UserRoleDAO:: setRoles(user:User, roles: Collection): void **pre:** user != null AND user.username != null AND roles!=null AND roles.length()>0 AND roles->forAll(r | r!=null AND r.length>0)
Post-condizione	**context:** UserRoleDAO:: setRoles(user:User, roles: Collection): void **post:** roles->forAll(r| Database.User_Roles->exists(u_r | u_r.user_name==user.user_name AND u_r.role_name==r))

Nome Metodo	+removeAllRoles(user: User): void

Descrizione	Questo metodo rimuove tutti i ruoli associati allo User “user”.
Pre-condizione	**context:** UserRoleDAO:: removeAllRoles(user:User): void **pre:** user!=null
Post-condizione	**context:** UserRoleDAO:: removeAllRoles(user:User): void **post:** Database.User_Roles->!exists(u_r | u_r.user_name==user.user_name)

3.2 Merchandise Service

Nome Classe	MangaDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono i prodotti di MangaUP
Metodi	+create(Manga manga): void +retrieveById(int id): Manga +retrieveAll(): Collection +updateQuantity(Manga manga, int quantity): void +filterForEndUsers(String name,String collection_name): Collection +filterForUsers(String name, float min_price, float max_price, String collection_name, bool order_subject, bool order_criteria): Collection +delete(int id): void
Invariante di classe	self.ds != null

Nome Metodo	+create(Manga manga): void |||
Descrizione	Questo metodo permette di creare un nuovo manga
Pre-condizione	**context:** MangaDAO::create(Manga manga): void **pre:** manga != null AND manga.name != null AND manga.name != “” AND manga.name.length()<=50 AND Database.Manga->!exists(Manga | Manga.name == manga.name) AND manga.editore != null AND manga.editore != “” AND manga.editore.length()<=50 AND

	manga.price>0 AND manga.weight >0 AND manga.height >0 AND manga.length >0 AND manga.state != null AND (manga.state == ProductState.NEW || manga.state == ProductState.USED) AND manga.description != null AND len(manga.description) <= 255 AND manga.ISBN != null AND len(manga.ISBN) == 13 AND manga.book_binding != null AND manga.book_binding != “” AND len(manga.book_binding)<=30 AND manga.volume != null AND manga.volume != “” AND len(manga.volume)<=20 AND manga.exitDate != null AND manga.exitDate <= CurrentDate AND manga.pageNumber >0 AND manga.quantity >0 AND manga.interior != null AND manga.interior != “” AND len(manga.interior)<=20 AND manga.language != null AND manga.language != “” AND len(manga.language)<=20 AND manga.storyMaker != null AND manga.storyMaker != “” AND len(manga.storyMaker)<=25 AND manga.genre != null AND manga.genre.name != null AND manga.genre.name != “” AND manga.collection != null AND manga.collection.name != null AND manga.collection.name != “” AND Database.Genre->include(manga.genre) AND Database.Collection->include(manga.collection)
Post-condizione	**context:** MangaDAO::create(Manga manga): void **post:**Database.Manga->include(manga)
Nome Metodo	+retrieveById(int id): Manga |||
Descrizione	Questo metodo permette di recuperare un manga esistente nel db in base al suo id
Pre-condizione	**context:** MangaDAO::retrieveById(int id): Manga **pre:**
Post-condizione	**context:** MangaDAO::retrieveById(int id): Manga

	post: result = Database.Manga->select(Manga | Manga.id == id)
Nome Metodo	+retrieveAll(): Manga |||
Descrizione	Questo metodo restituisce tutti i manga presenti nel db
Pre-condizione	**context:** MangaDAO::retrieveAll(): Manga **pre:** true
Post-condizione	**context:** MangaDAO::retrieveAll(): Manga **post:** result = Database.Manga
Nome Metodo	+updateQuantity(int quantity, int id): void |||
Descrizione	Questo metodo consente di aggiungere unità ad un prodotto presente nel db
Pre-condizione	**context:** MangaDAO::updateQuantity(int id, int quantity): void **pre:** Database.Manga->exists(Manga | Manga.id == id) AND quantity > 0
Post-condizione	**context:** MangaDAO::updateQuantity(ind id , int quantity): void **post:** Database.Manga->select(Manga | Manga.id == manga.id).quantity == @pre.Database.Manga->select(Manga | Manga.id == manga.id).quantity + quantity
Nome Metodo	+updateQuantity(Manga m): void |||
Descrizione	Questo metodo consente di aggiornare la quantità di un prodotto presente nel db
Pre-condizione	**context:** MangaDAO::updateQuantity(Manga m): void **pre:** m!=null AND Database.Manga->exists(Manga | Manga.id == manga.id) AND quantity >= 0

Post-condizione	**context:** MangaDAO::updateQuantity(Manga m): void
	
	post:
	
	if(quantity==0)
	
	then
	
	Database.Manga->!exists(m| m.id == id)
	
	else
	
	Database.Manga->select(Manga | Manga.id == manga.id).quantity == quantity

Nome Metodo	+filterForEndUsers(String name, String collection_name): Collection
Descrizione	Questo metodo permette agli end user di filtrare un insieme di Manga dal db in base a nome e collezione inseriti in un form
Pre-condizione	**context:** MangaDAO::filterForEndUsers(String name, String collection_name): Collection **pre:** if((name == null || name==””) (&& collection_name==null || collection_name==””)) then true else Database.Collection->exists(c|c.name==collection_name)

Post-condizione	**context:** MangaDAO::filterForEndUsers(String name, String collection_name): Collection
	
	post:
	
	if((name == null || name==””) && (collection_name==null || collection_name==””))
	
	then
	
	result = Database.Manga
	
	else
	
	result = Database.Manga->select(m | m.name.substring(name) AND manga.collection_name==collection_name))

Nome Metodo	+filterForUsers(String name, float min_price, float max_price, String collection_name, bool order_subject, bool order_criteria): Collection
Descrizione	Questo metodo permette al gestore del catalogo di filtrare un insieme di Manga dal db in base a nome, collezione,prezzo minimo, prezzo massimo e ordinamento basato su nome o su collezione e criterio crescente o decrescente
Pre-condizione	**context:** MangaDAO::filterForUsers(String name, float min_price, float max_price, String collection_name, bool order_subject, bool order_criteria): Collection **pre:** if((name == null || name==””) && (collection_name==null || collection_name==””) && (max_price==0 && min_price==0))

	then true else 0<=min_price<max_price AND Database.Collection->exists(c|c.name==collection_name)

Post-condizione	**context:** MangaDAO::filterForUsers(String name, float min_price, float max_price, String collection_name, bool order_subject, bool order_criteria): Collection
	
	post:
	
	if((name == null || name=="") && (collection_name==null || collection_name=="") && (max_price==0 && min_price==0))
	
	then
	
	result = Database.Manga
	
	else
	
	result= Database.Manga->select(m | m.name.substring(name) AND m.collection_name == collection_name AND min_price <= m.price <= max_price)

Nome Metodo	+delete(int id): void |||
Descrizione	Questo metodo permette al gestore del catalogo di eliminare un prodotto dal sistema
Pre-condizione	**context:** MangaDAO::delete(int id): void **pre:** Database.Manga->exists(Manga | Manga.id == manga.id)
Post-condizione	**context:** MangaDAO::delete(int id): void **post:** Database.Manga->!exists(Manga | Manga.id == manga.id)

Nome Classe	GenreDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono i generi contenuti nel db

Metodi	+retrieve(genre_name:String): Genre +retrieveAll():Collection
Invariante di classe	self.ds != null

Nome Metodo	+retrieve(String genre_name): Genre
Descrizione	Questo metodo permette recuperare un genere dal db
Pre-condizione	**context:** GenreDAO::retrieve(genre_name: String): Genre **pre:** genre_name!=null
Post-condizione	**context:** GenreDAO::retrieve(genre_name: String): Genre **post:** result = Database.Genre->select(g | genre_name == g.name)
Nome Metodo	+retrieveAll(): Collection
Descrizione	Questo metodo permette recuperare tutti i generi presenti nel db
Pre-condizione	**context:** GenreDAO::retrieveAll(): Collection **pre:** true
Post-condizione	**context:** GenreDAO::retrieveAll(): Collection **post:** result = Database.Genre

Nome Classe	CollectionDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono i generi contenuti nel db
Metodi	+retrieve(collection_name:String): Collection (Classe del modello a oggetti) +retrieveAll():Collection(Collezione)
Invariante di classe	/

Nome Metodo	+retrieve(collection_name: String): Collection
Descrizione	Questo metodo permette recuperare una collezione
Pre-condizione	**context:** CollectionDAO::retrieve(collection_name: String): Collection **pre:** collection_name != null
Post-condizione	**context:** CollectionDAO::retrieve(): Collection **post:** result = Database.Collection->select(c |c.name == collection_name)
Nome Metodo	+retrieveAll(): Collection
Descrizione	Questo metodo permette di recuperare tutte le collezioni presenti nel db
Pre-condizione	**context:** CollectionDAO::retrieveAll(): Collection **pre:**
Post-condizione	**context:** CollectionDAO::retrieveAll(): Collection **post:** result = Database.Collection

3.2 Checkout Service

Nome Classe	CartDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono i prodotti contenuti nel carrello dell'end-user
Metodi	+addProduct(Manga manga,int quantity,EndUser user): void +retrieveByUser(EndUser user): Cart +updateProduct(Manga manga,int quantity,EndUser user): void +removeProduct(Manga manga,EndUser user): void +toEmptyCart(EndUser user): void
Invariante di classe	self.ds != null

Nome Metodo	+addProduct(manga: Manga, quantity: int, user: EndUser): void
Descrizione	Questo metodo permette di aggiungere un prodotto al carrello
Pre-condizione	**context:** CartDAO::addProduct(manga: Manga,quantity: int, user: Enduser): void **pre:** manga != null AND quantity>0 AND manga.quantity>=quantity AND Database.EndUser->include(user) AND Database.Manga->include(manga)
Post-condizione	**context:** CartDAO::addProduct(mang: Manga,quantity: int, user: Enduser): void **post:** Database.Cart->exists(c | c.manga_id = manga.manga_id AND c.userId == user.userId AND c.quantity==quantity)
Nome Metodo	+retrieveByUser(user: EndUser): Cart
Descrizione	Questo metodo recupera il carrello associato ad un utente
Pre-condizione	**context:** CartDAO::retrieveByUser(EndUser user): Cart **pre:** user != null AND Database.EndUser->exists(u | u.id == user.id)
Post-condizione	**context:** CartDAO::retrieveByUser(EndUser user): Cart **post:** result = Database.Cart->select(cart | cart.user_id == user.id)
Nome Metodo	+updateProduct(manga: Manga,quantity: int,user: Enduser): void
Descrizione	Questo metodo recupera il carrello associato ad un utente
Pre-condizione	**context:** CartDAO::updateProduct(Manga manga,int quantity,EndUser user): void **pre:** user != null AND manga != null AND quantity >0 AND quantity <= manga.quantity AND Database.Manga->include(manga) AND Database.EndUser->include(user)
Post-condizione	**context:** CartDAO::updateProduct(Manga manga,int quantity,EndUser user): void **post:** Database.Cart->exists(c | c.id == manga.id AND c.user_id==user.id AND c.quantity == quantity)
Nome Metodo	+removeProduct(Manga manga,EndUser user): void
Descrizione	Questo metodo rimuove un prodotto dal carrello di un utente

Pre-condizione	**context:** CartDAO::removeProduct(Manga manga,EndUser user): void **pre:** manga != null AND user != null AND Database.Manga->include(manga) AND Database.EndUser->include(user) AND Database.Cart->exists(cart | cart.userId == user.id AND cart.mangaId == manga.id)
Post-condizione	**context:** CartDAO::removeProduct(Manga manga,EndUser user): void **post:** Database.Cart->!exists(cart | cart.userId == user.id AND cart.mangaId == manga.id)
Nome Metodo	+toEmptyCart(EndUser user): void
Descrizione	Questo metodo recupera il carrello associato ad un utente
Pre-condizione	**context:** CartDAO::toEmptyCart(EndUser user): void **pre:** user!=null AND Database.EndUser->include(user)
Post-condizione	**context:** CartDAO::toEmptyCart(EndUser user): void **post:** Database.Cart->!exists(cart | cart.userId == user.id)

3.4 Dispatch Service

Nome Classe	OrderRowDAO
Descrizione	Questa classe permette di inserire e recuperare le righe d'ordine relative ad un acquisto di un endUser.
Metodi	+create(o:OrderRow): void +retrieveOrderRowAssociatedToOrder(order:Order): Collection
Invariante di classe	self.ds != null

Nome Metodo	+create(o:OrderRow): void
Descrizione	Questo metodo inserire una riga d'ordine nel db
Pre-condizione	**context:** OrderRowDAO::create(orow:OrderRow): void **pre:** m.order.id>0 AND m.user.id>0 AND m.manga.name() != null AND m.manga.name != "" AND manga.price>0 manga.quantity>0 AND Database.Order->exists(o | o.ord_id == orow.order.id)
Post-condizione	**context:** OrderRowDAO::retrieveOrderRowAssociatedToOrder(order:Order): Collection **post:** Database.OrderRow->include(orow)
Nome Metodo	
Descrizione	Questo metodo recupera tutte le righe d'ordine associate ad un ordine
Pre-condizione	**context:** OrderRowDAO::retrieveOrderRowAssociatedToOrder(order:Order): Collection **pre:** order != null
Post-condizione	**context:** OrderRowDAO::retrieveOrderRowAssociatedToOrder(order:Order): Collection **post:** Database.OrderRow->select(row | row.ordId == order.id)

Nome Classe	ManagedOrderDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono gli ordini gestiti dai gestori degli ordini.
Metodi	+create(ManagedOrder m): void +retrieveByOrder(int ordId): ManagedOrder
Invariante di classe	self.ds != null

Nome Metodo	+create(ManagedOrder m): void

Descrizione	Questo metodo permette di registrare la gestione di un ordine e contestualmente di spedire un ordine
Pre-condizione	**context:** ManagedOrderDAO::create(ManagedOrder m): void **pre:** m.orderDate != null AND m.state != Order.SENT AND m.totalPrice>0 AND m.user != null AND m.user.id >0 AND Database.User->exists(u | u.username == m.username) AND m.addressInfo != null AND m.addressInfo != “” AND m.creditCardInfo != null AND m.creditCardInfo != “” AND m.managerName != null AND m.managerName != “” AND m.shipmentDate != null AND m.shipmentDate > m.orderDate AND m.deliveryDate != null AND m.deliveryDate > m.orderDate AND m.deliveryDate > m.shipmentDate AND m.trackNumber != null AND m.trackNumber != ”” AND m.courierName != null AND m.courierName != “” AND Database.Order->select(o | o.id == m.id) AND Database.ManagedOrder->!exists(man | man.man_tracking_number == m.trackNumber)
Post-condizione	**context:** ManagedOrderDAO::create(ManagedOrder m): void **post:** Database.Manages->include(m)
Nome Metodo	+retrieveByOrder(int ordId): ManagedOrder
Descrizione	Questo metodo permette di recuperare un ordine gestito dal db
Pre-condizione	**context:** ManagedOrderDAO:: retrieveByOrder(int ordId): ManagedOrder **pre:** true
Post-condizione	**context:** ManagedOrderDAO:: retrieveByOrder(int ordId): ManagedOrder **post:** result = Database.Manages->select(m | m.ord_id == ordId)

Nome Classe	ToManageDAO
Descrizione	Questa classe consente di poter gestire gli assegnamenti degli ordini.
Metodi	+create(ToManage order): void +delete(ToManage order): void +hasOrders(String username): boolean
Invariante di classe	self.ds != null

Nome Metodo	+create(order: ToManage) void
Descrizione	Associa l'ordine al gestore degli ordini
Pre-condizione	**context:** ToManage::create(ToManage order): void **pre:** order.user != null AND order.order != null AND order.user.username != null AND order.user.username != “” order.orde.id > 0 AND Database.Order->exists(o | o.ord_end_user_id == order.user.id) Database.User->exists(u | u.username == order.user.username)
Post-condizione	**context:** ToManageDAO::create(ToManage order): Collection c **post:** Database.ToManage -> exists(t | t.user_name == order.user.username)

Nome Metodo	+delete(order: ToManage) void
Descrizione	elimina l'ordine della lista associata degli ordini
Pre-condizione	**context:** ToManageDAO:: delete(ToManage order): void **pre:** order!=null
Post-condizione	**context:** ToManageDAO:: delete(ToManage order): void **post:** Database.ToManage -> !exists(t | t.user_name ==
Nome Metodo	+hasOrders (String username): boolean

Descrizione	Questo metodo verifica che un gestore degli ordini abbia o meno degli ordini da gestire
Pre-condizione	**context:**hasOrders (String username): boolean **pre:** username != NULL AND username
Post-condizione	**context:**hasOrders (String username): boolean **post:** result = Database.ToManage->exists(t | t.user_name == username)

Nome Classe	OrderDAO
Descrizione	Questa classe permette di gestire le operazioni che coinvolgono gli ordini effettuati dagli utenti
Metodi	+create(Order order): void +update(Order order): void +retrieve(int id) : Order +doRetrieveAllSpecificUser(String orderManagerUserName, String orderCriteria): Collection +retrieveOrdersAssociatedToUsers(EndUser user): Collection
Invariante di classe	self.ds != null

Nome Metodo	+create(Order order): void
Descrizione	crea un nuovo ordine
Pre-condizione	**context:** OrderDAO:: create(Order order): void **pre:** order.orderDate != null AND order.state != null AND order.state == Order.TO_SEND AND order.endUser != null AND Database.EndUser->contains(order.endUser) AND order.totalPrice<=0 AND order.endUser.id>0 AND order.addressInfo!=null AND order.addressInfo != “” AND order.creditCardInfo != null AND order.creditCardInfo != “”
Post-condizione	**context:** OrderDAO:: create(Order order): void **post:** Database.Order->include(order)

Nome Metodo	+update(Order order): void
Descrizione	questo metodo aggiorna lo stato di un ordine da “Da spedire” a “spedito”
Pre-condizione	**context:** OrderDAO:: update(int id): void **pre:** true
Post-condizione	**context:** OrderDAO:: update(int id): void **post:** Database.Order->select(o | o.order_id == id).first().state == Order.SENT
Nome Metodo	+retrieve(int id): void
Descrizione	questo metodo recupera uno specifico ordine dal db
Pre-condizione	**context:** OrderDAO:: retrieve(int id): Order **pre:** true
Post-condizione	**context:** OrderDAO:: retrieve(int id): Order **post:** result = Database.Order->select(o | o.order_id == id)
Nome Metodo	+doRetrieveAllSpecificUser(String orderManagerUserName, String orderCriteria): Collection
Descrizione	questo metodo recupera l’insieme di ordini che devono essere gestiti da un gestore
Pre-condizione	**context:** OrderDAO:: doRetrieveAllSpecificUser(String orderManagerUserName, String orderCriteria): Collection **pre:** true
Post-condizione	**context:** OrderDAO:: doRetrieveAllSpecificUser(String orderManagerUserName, String orderCriteria): Collection **post:** result = Database.ToManage->select(t | t.order_id == orderManagerId AND t.username == orderManagerUserName)
Nome Metodo	+retrieveOrdersAssociatedToUsers(EndUser user): Collection
Descrizione	Questo metodo recupera lo storico degli ordini effettuati da un end-user
Pre-Condizione	**context:** OrderDAO:: retrieveOrdersAssociatedToUsers(EndUser user) Collection **pre:** user != null

Post-condizione	+retrieveOrdersAssociatedToUsers(EndUser user) Collection result = Database.Order-> select(o | o.user_id = user.id)

Nome Classe	OrderSubmissionFacadeImp
Descrizione	Questa classe offre servizi per poter effettuare il pagamento, permette di creare l' ordine e le sue righe nel database e inoltre permette di assegnare un ordine da gestire al gestore degli ordini.
Metodi	+executeTask(managedOrder: ManagedOrder): void +createOrder(order : Order, products: Collection, selectedManager: User): void
Invariante di classe	self.ds != null AND OrdeDAO != NULL AND UserDAO != NULL AND ToManageDAO != NULL AND ManagedOrderDAO != NULL AND OrderRowDAO != NULL

Nome Metodo	+createOrder(order : Order, products: Collection, selectedManager: User): Collection
Descrizione	Questo metodo crea l'ordine e le relative order rows e lo assegna ad un gestore degli ordini.
Pre-condizione	**context:** OrderSubsmissionFacadeImp:: createOrder(order : Order, products: Collection, selectedManager: User): void **pre:** order != NULL AND products != NULL AND selectedManager != null
Post-condizione	**context:** OrderSubsmissionFacadeImp:: createOrder(order : Order, products: Collection, selectedManager: User): void **post:** **//Convenzione : total() produce il valore equivalente al prezzo totale dell'ordine** var total = products.total() Database.Order->exists(ord | ord.ord_total_price == total AND ord.ord_end_user_id == order.end_user_id AND ord.ord_state ==

	order.state AND ord.ord_date == order.orderDate AND ord.ord_address == order.endUseraddress AND ord.ord_card == order.endUserCard AND products-> ForAll(p | Database.OrderRow->exists(o | o.manga_name == p.name AND o.manga_price == p.price AND o.quantity == p.quantity AND o.ord_id == ord.ord_id AND o.user_id == ord.ord_end_user_id)) AND Database.ToManage->exists(toM | toM.user_name == selectedManager.username AND toM.order_id == ord.ord_id))
Nome Metodo	+executeTask(managedOrder: ManagedOrder): void
Descrizione	Questo metodo registra la gestione di un ordine da parte di uno specifico gestore degli ordini
Pre-condizione	**context:** OrderSubsmissionFacadeImp:: executeTask(managedOrder: ManagedOrder): void **pre:** managedOrder != null
Post-condizione	**context:** OrderSubsmissionFacadeImp:: executeTask(managedOrder: ManagedOrder): void **post:** Database.ManagedOrder->exists(m | m.man_user_name == managedOrder.user.username AND m.man_shipment_date == managedOrder.shipmentDate AND m.man_tracking_number == managedOrder.trackNumber AND m.man_courier == managedOrder.courierName AND m.man_delivery_date == managedOrder.deliveryDate AND Database.ToManage->!exists(t_m | t_m.user_name == managedOrder.user.username AND m.man_order_id == t_m.order_id) AND Database.Order->exists(o | o.ord_id == m.man_order_id AND o.ord_state == Order.Sent)

4. Design Patterns

I Design Patterns che sono stati impiegati per l'implementazione sono i seguenti:

- DAO (**D**ata **A**ccess **O**bject)
- Façade

Abbiamo usato un DAO per facilitare le operazioni di accesso e manipolazione dei dati nel database del nostro sistema per ogni tipo di entità esistente a livello applicazione.

Per quanto riguarda i façade, essi sono stati impiegati per fornire un'interfaccia semplificata per alcune operazioni che sarebbe stato necessario invocare in maniera separata (il che avrebbe potuto anche portare a potenziali utilizzi errati delle funzionalità offerte), riducendo l'accoppiamento tra le componenti e rendendo di conseguenza il sistema più manutenibile.

Di seguito riproponiamo una view di uno dei façade che abbiamo precedentemente mostrato nella sezione sui packages in maniera più dettagliata.