Università degli Studi di Salerno
Corso di Ingegneria del Software

BirdComics
Object Design Document
Versione 2.0

BirdComics

Data: 23/03/2025

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
De Rosa Giuseppe	0512110683
Buccino Adriano	0512120484

Scritto da:	

Revision History

Data	Versione	Descrizione	Autore
23/03/2025	2.0	Aggiunti OCL e trade offs	De Rosa Giuseppe Buccino Adriano
16/12/2024	1.0	Prima versione Object Design Document	De Rosa Giuseppe Buccino Adriano

1. Introduzione

1.1. Object design trade-offs

- **Usabilità vs tempistiche**
  Per implementare un sistema con un'interfaccia grafica intuitiva e messaggi di errore chiari, precisi e concisi, è necessario un investimento di tempo significativo, che non è compatibile con tempistiche di sviluppo brevi. Pertanto, si è optato per un compromesso tra usabilità e tempistiche, focalizzandosi sulla chiarezza dei messaggi di errore riguardanti l'usabilità delle operazioni più comuni, tralasciando momentaneamente i messaggi di errore delle funzionalità più complesse. Questo approccio consente di rilasciare un prodotto funzionale in tempi ridotti, pur garantendo un'esperienza utente soddisfacente per le attività principali.
- **Supportabilità vs tempistiche**
  La realizzazione di un sistema altamente modulare richiede un maggiore sforzo di progettazione e sviluppo, con un conseguente aumento dei tempi di implementazione. Per bilanciare questo aspetto, si è deciso di adottare un approccio intermedio, rendendo il sistema moderatamente modulare ma con la possibilità di estendere le funzionalità in futuro. Questo compromesso consente di rispettare le tempistiche di sviluppo iniziali, pur mantenendo una buona flessibilità per aggiornamenti e manutenzioni successive.
- **Sicurezza vs costi**
  L'implementazione di un sistema con un elevato livello di sicurezza richiede l'utilizzo di risorse hardware e software avanzate, che comportano costi significativi. Pertanto, si è deciso di adottare un approccio bilanciato, garantendo un livello di sicurezza adeguato alle esigenze del sistema senza eccedere in investimenti non strettamente necessari. Questo compromesso consente di proteggere il sistema da minacce comuni, pur mantenendo i costi entro limiti accettabili. Tuttavia, è prevista la possibilità di incrementare le misure di sicurezza in futuro, qualora le esigenze del sistema o il contesto di rischio lo richiedano.
- **Prestazioni vs costi**
  Le prestazioni del sistema sono state progettate in modo scalabile, al fine di ottimizzare i costi e garantire un funzionamento efficiente. Un sistema con prestazioni eccessivamente elevate rispetto alle reali necessità comporterebbe uno spreco di risorse economiche, mentre prestazioni insufficienti comprometterebbero l'affidabilità e l'usabilità del sito. Pertanto, si è scelto di adottare un'architettura scalabile, che consente di allocare le risorse hardware in modo dinamico, in base al carico di lavoro effettivo. Questo approccio garantisce un equilibrio ottimale tra prestazioni e costi operativi.

1.2. Linee guida per la documentazione dell'interfaccia

Linee Guida per la Stesura del Codice

Queste linee guida sono pensate per garantire uniformità, leggibilità e facilità di manutenzione del codice, seguendo le naming convention ufficiali di Oracle per il linguaggio Java.

Nomi delle Classi

I nomi delle classi devono seguire la notazione UpperCamelCase, ovvero iniziare con una lettera maiuscola e utilizzare una maiuscola per ogni parola successiva (es. NomeClasse).

Le classi che rappresentano Bean devono terminare con il suffisso Bean (es. UtenteBean).

Le classi dedicate alla gestione della persistenza devono terminare con il suffisso DAO (es. OrdineDAO).

Le classi che rappresentano oggetti di controllo (ad esempio Servlet) devono avere un nome che descriva chiaramente la loro funzionalità (es. LoginServletl).

Nomi delle Interfacce

I nomi delle interfacce devono seguire la notazione UpperCamelCase, rispettando lo stesso stile delle classi (es. NomeInterfaccia).

Nomi dei Metodi

I metodi devono seguire la notazione lowerCamelCase, iniziando con una lettera minuscola e utilizzando una maiuscola per ogni parola successiva (es. nomeMetodo).

Nomi delle Variabili

Variabili di proprietà: devono seguire lo stile lowerCamelCase, iniziando con una lettera minuscola (es. nomeVariabile).

Variabili locali: seguono lo stesso stile delle variabili di proprietà, con notazione lowerCamelCase (es. valoreTemporaneo).

Costanti: i nomi delle costanti devono essere scritti interamente in maiuscolo (es. MAXVALORE).

Organizzazione dei Package

Ogni sottosistema deve essere organizzato in un package dedicato.

All'interno del package devono essere incluse tutte le classi che realizzano il sottosistema, come Servlet, Service, DAO e Bean. Questo approccio aiuta a mantenere una struttura del progetto ordinata e facilmente navigabile.

1.3. Design Pattern

Si è scelto di utilizzare i seguenti pattern:

- **DAO (Data Access Object)**: Fornisce una separazione chiara tra l'accesso ai dati e il resto del sistema, favorendo la modularità e facilitando la gestione delle operazioni CRUD su un database.
- **MVC (Model-View-Controller)**: Aiuta a organizzare l'applicazione in tre componenti distinti: Model, View e Controller, migliorando la manutenzione e il riutilizzo del codice.
- **Facade Pattern**: Fornisce un'unica interfaccia per accedere ad un insieme di oggetti che compongono un sottosistema.
- **Singleton Pattern**: Garantisce che ci sia solo un'istanza di una classe, utile per gestire risorse condivise come configurazioni globali o connessioni a database, assicurando un accesso coerente e sicuro.

1.4. Definizioni, acronimi, e abbreviazioni

CRUD (Create, Read, Update, Delete): insieme delle quattro operazioni fondamentali per la gestione dei dati in un database.

DAO (Data Access Object): pattern di progettazione che fornisce un'interfaccia astratta per interagire con un database.

UML (Unified Modeling Language): linguaggio standard per la modellazione e visualizzazione di sistemi software.

OCL (Object Constraint Language): linguaggio dichiarativo usato per specificare vincoli e regole nei modelli UML.

1.5. Riferimenti

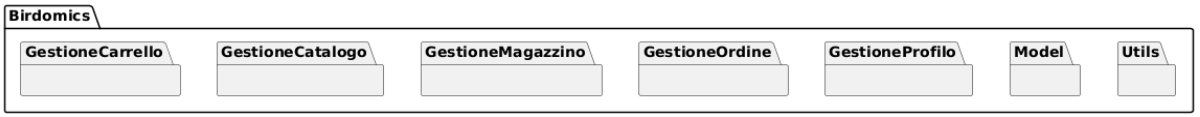
Nel corso del documento facciamo riferimento a artefatti precedenti quali il Requirements Analysis Document (RAD) e al System Design Document (SDD).

Oltre ai documenti del progetto si fa riferimento a concetti e approcci metodologici tratti da testi di riferimento e linee guida riconosciute nel campo dell'ingegneria del software:

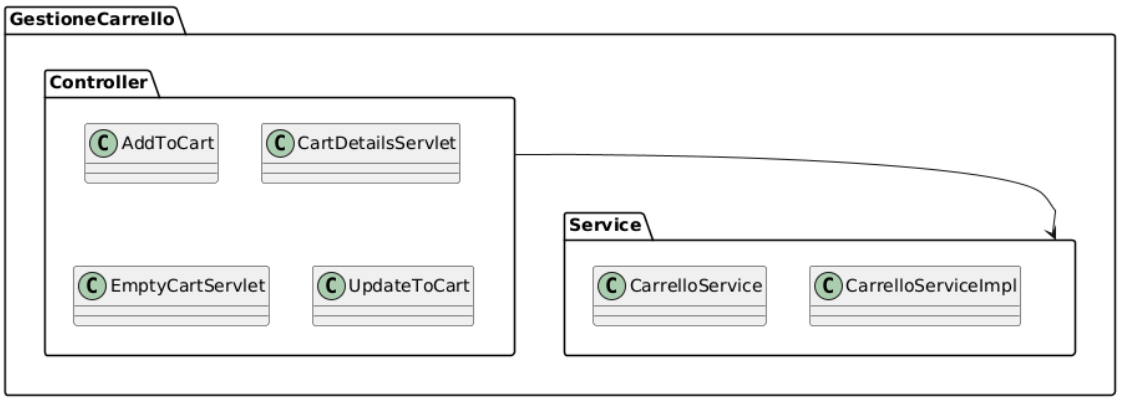
- Object-Oriented Software Engineering Using UML, Patterns, and Java™ di Bernd Bruegge & Allen H. Dutoit
- Oracle Naming Conventions Java
- Oracle Java Code Conventions

2. Packages

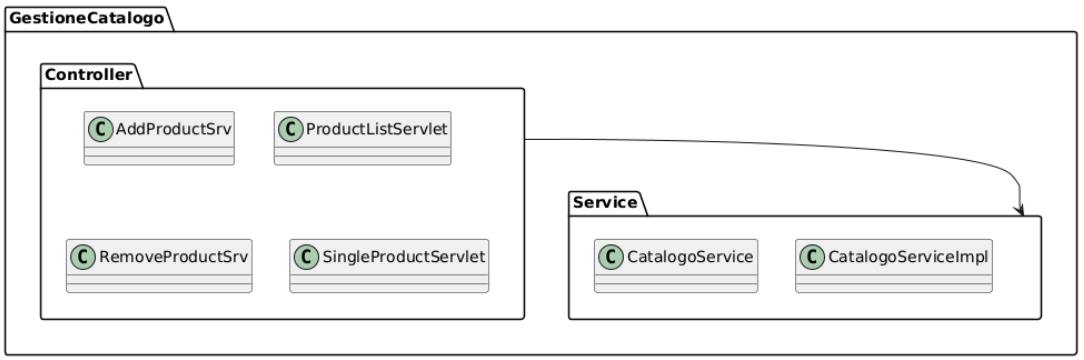
Questa sezione descrive i sottosistemi principali di BirdComics, definiti durante la fase di System Design e documentati nel System Design Document (SDD).



Sottosistema Gestione Carrello

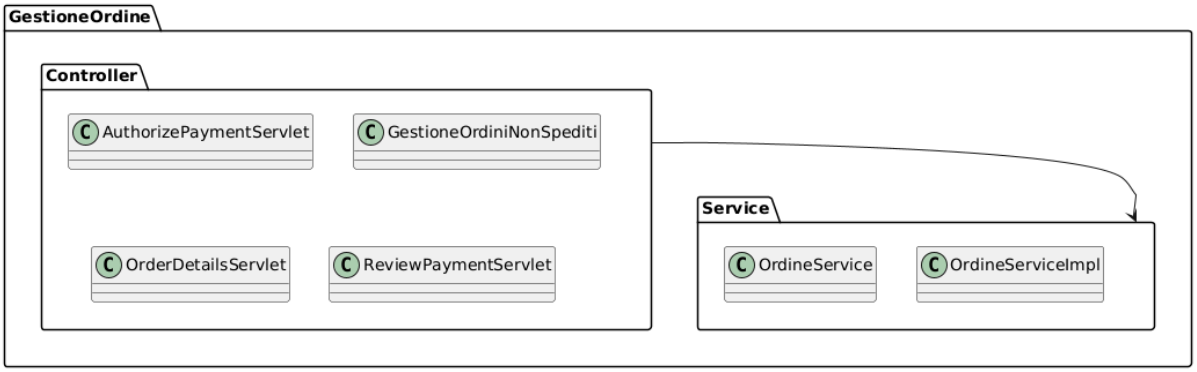


Sottositema Gestione Catalogo

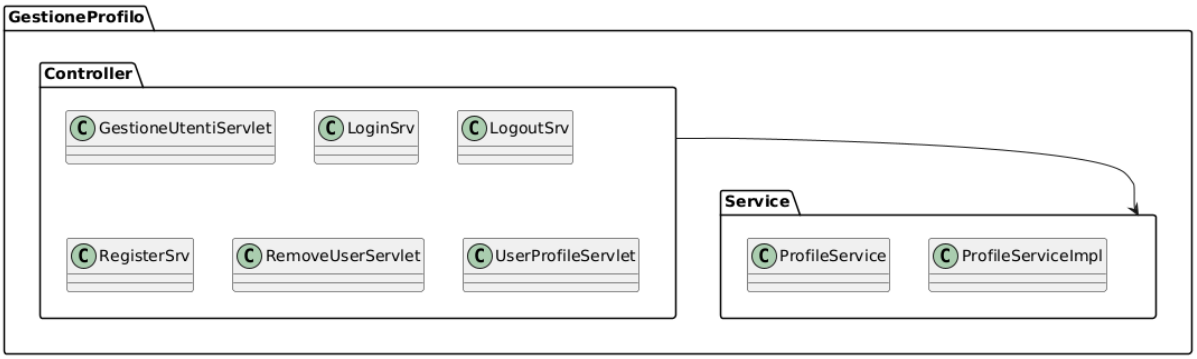


Sottosistema Gestione Magazzino

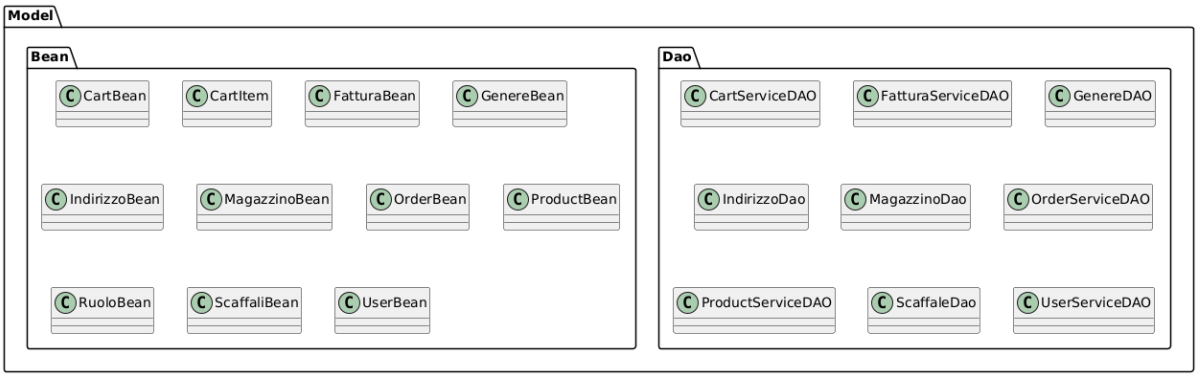
Sottosistema Gestione Ordine



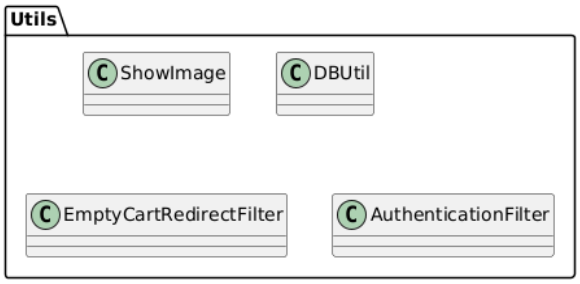
Sottosistema Gestione Profilo



Partizione Model



Partizione Utils



3. Interfaccia delle Classi (OCL)

3.1. Gestione Profilo

Interfaccia	ProfileService
Descrizione	Vedi descrizione del SSD
Metodi	+ registraAccount(emai:String, password: String, nome: String, cognome: String, telefono: String, dataNascita: sql.Date, citta: String, via: String, numeroCivico: String, cap: String, ruoli: List<RuoloBean>, magazzino:MagazzinoBean) : String + login : login(email: String, password : String) : String + logout(HttpServletRequest request) : void + getUserTypes(email: String): List<String> + rimuoviAccount(userEmail: String) : void + getUserDetails(email: String) : UserBean + getUsersByRole(roles: List<RuoloBean>, magazzino: String) : List<UserBean>
Invariante	

Nome Metodo	+ registraAccount(emai:String, password: String, nome: String, cognome: String, telefono: String, dataNascita: sql.Date, citta: String, via: String, numeroCivico: String, cap: String, ruoli: List<RuoloBean>, magazzino:MagazzinoBean) : String
Descrizione	Questo metodo serve a registrare un nuovo utente all'interno del sistema assegnando dei ruoli definiti: ● Nel caso del ruolo cliente, esso puo essere solo cliente e nessun ruolo manager, e la sede del magazzino deve essere vuota ● Nel caso si impostano dei ruoli come direttoreMagazzino oppure Hr, essi possono ricoprire solo ed esclusivamente un ruolo ● Nel caso in cui si devono impostare dei dipendenti di una sede, assunti da un HR, questi dipendenti possono ricoprire piu ruoli
Pre-Condizione	context ProfileServiceImpl::registraAccount(emai:String, password: String, nome: String, cognome: String, telefono: String, dataNascita: sql.Date, citta: String, via: String, numeroCivico: String, cap: String, ruoli: List<RuoloBean>, magazzino:MagazzinoBean) : String pre: email <> null and -- L'email non è null password <> null and and nome <> null and cognome <> null and not User.allInstances()->exists(u | u.email = email) and ruoli->forAll(r | r.toString() = 'Cliente' implies (ruoli->size() = 1 and magazzino = null)) and ruoli->forAll(r | (r.toString() = 'DirettoreMagazzino' or r.toString() = 'HR') implies ruoli->size() = 1) and ruoli->exists(r | r.toString() = 'Dipendente') implies (magazzino <> null and ruoli->size() >= 1)
Post-Condizione	context ProfileServiceImpl::registraAccount(emai:String, password: String, nome: String, cognome: String, telefono: String, dataNascita: sql.Date, citta: String, via: String, numeroCivico: String, cap: String, ruoli: List<RuoloBean>, magazzino:MagazzinoBean) : String post: User.allInstances()->exists(u | u.email = email)

Nome Metodo	+ login : login(email: String, password : String) : String
Descrizione	Serve ad autenticare l'attore con il proprio account
Pre-Condizione	context ProfileServiceImpl::login(email: String, password: String, request: HttpServletRequest) : String pre: email <> null and password <> null and User.allInstances()->exists(u | u.email = email) and User.allInstances()->exists(u | u.email = email and u.password = password)
Post-Condizione	request.session <> null and request.session.user = User.allInstances()->select(u | u.email = email)->first()

Nome Metodo	+ logout(HttpServletRequest request) : void
Descrizione	Il metodo termina la sessione dell'utente, disconnettendolo dal sistema.
Pre-Condizione	context ProfileServiceImpl::logout(request: HttpServletRequest) : void pre: request.session <> null and request.session.user <> null
Post-Condizione	post: request.session = null

Nome Metodo	+ getUserTypes(email: String): List<String>
Descrizione	Il metodo restituisce la lista dei ruoli che ricopre l'utente associato all'email fornita.
Pre-Condizione	context ProfileServiceImpl:getUserTypes(email: String): List<String> pre: email <> null and email <> '' and User.allInstances()->exists(u | u.email = email)
Post-Condizione	result <> null and result->notEmpty()

Nome Metodo	+ rimuoviAccount(userEmail: String) : void
Descrizione	Il metodo rimuove un utente dal sistema in base all'email fornita
Pre-Condizione	context ProfileServiceImpl::rimuoviAccount(userEmail: String) : void pre: userEmail <> null User.allInstances()->exists(u | u.email = userEmail)
Post-Condizione	context ProfileServiceImpl::rimuoviAccount(userEmail: String) : void post: not User.allInstances()->exists(u | u.email = userEmail)

Nome Metodo	+ getUserDetails(email: String) : UserBean
Descrizione	Il metodo recupera i dettagli di un utente in base all'email fornita
Pre-Condizione	context UserService::getUserDetails(email: String) : UserBean pre: email <> null
Post-Condizione	context UserService::getUserDetails(email: String) : UserBean post: result <> null and result.email = email

Nome Metodo	+ getUsersByRole(roles: List<RuoloBean>, magazzino: String) : List<UserBean>
Descrizione	Il metodo restituisce una lista di utenti filtrata in base ai ruoli e al magazzino specificati
Pre-Condizione	context ProfileService::getUsersByRole(roles: List<RuoloBean>, magazzino: String) : List(UserBean) pre: (roles->exists(r | r.name = 'Cliente') and magazzino = null) or (roles->forAll(r | r.name <> 'Cliente') and magazzino <> null)
Post-Condizione	context ProfileService::getUsersByRole(roles: List<RuoloBean>, magazzino: String) : List(UserBean) post: result->forAll(user | user.email <> null) and result->forAll(user | user.ruolo->exists(r | roles->includes(r))) and (magazzino = null or result->forAll(user | user.magazzino.nome = magazzino))

3.2. Gestione Carrello

Interfaccia	CarrelloService
Descrizione	Vedi descrizione del SSD
Metodi	+ visualizzaProdottiCarrello(cartItems : List<CartItem>) : List<ProductBean> + calculateTotalAmount(cartItems : List<CartItem>) : float + svuotaCarrello(session : HttpSession, userId : String) : void + modificaQuantita(session : HttpSession, userId : String, prodId : String, pQty : int) :String + visualizzaCarrello(session : HttpSession, userId : String) : List<CartItem> + rimuoviFumetto(session : HttpSession, userId : String, prodId : String) : void + aggiungiFumetto(session : HttpSession, userId : String, prodId : String, pQty : int) : void + loadCartFromDB(session : HttpSession, email : String) : CartBean
Invariante	

Nome Metodo	+ visualizzaProdottiCarrello(cartItems : List<CartItem>) : List<ProductBean>
Descrizione	Restituisce la lista dei prodotti presenti nel carrello.
Pre-Condizione	context CarrelloService::visualizzaProdottiCarrello(cartItems : List<CartItem>) : List<ProductBean> pre: cartItems <> null
Post-Condizione	context CarrelloService::visualizzaProdottiCarrello(cartItems : List<CartItem>) : List<ProductBean> post: result <> null and result->forAll(p | p <> null)

Nome Metodo	+ calculateTotalAmount(cartItems : List<CartItem>) : float
Descrizione	Calcola l'importo totale dei prodotti nel carrello.
Pre-Condizione	context CarrelloService::calculateTotalAmount(cartItems : List<CartItem>) : float pre: cartItems <> null
Post-Condizione	context CarrelloService::calculateTotalAmount(cartItems : List<CartItem>) : float post: result >= 0

Nome Metodo	+ svuotaCarrello(session : HttpSession, userId : String) : void
Descrizione	Svuota il carrello dell'utente.
Pre-Condizione	context CarrelloService::svuotaCarrello(session : HttpSession, userId : String) : void pre: session <> null and userId <> null
Post-Condizione	context CarrelloService::svuotaCarrello(session : HttpSession, userId : String) : void post: post: session.cart.getCartItems()->size() = 0

Nome Metodo	+ modificaQuantita(session : HttpSession, userId : String, prodId : String, pQty : int) : String
Descrizione	Modifica la quantità di un prodotto nel carrello.
Pre-Condizione	context CarrelloService::modificaQuantita(session : HttpSession, userId : String, prodId : String, pQty : int) : String pre: session <> null and session.carrello <> null and userId <> null and prodId <> null and pQty > 0 and session.carrello.getCartItems()->exists(p | p.id = prodId)
Post-Condizione	context CarrelloService::modificaQuantita(session : HttpSession, userId : String, prodId : String, pQty : int) : String post: session.carrello <> null and session.carrello.getCartItems()->size() > 0 and session.carrello.getCartItems()->exists(p | p <> null and p.quantity >= 1 and p.id = prodId and p.quantity = pQty)

Nome Metodo	+ visualizzaCarrello(session : HttpSession, userId : String) : List<CartItem>
Descrizione	Restituisce il contenuto del carrello dell'utente.
Pre-Condizione	context CarrelloService::visualizzaCarrello(session : HttpSession, userId : String) : List<CartItem> pre: session <> null and userId <> null
Post-Condizione	context CarrelloService::visualizzaCarrello(session : HttpSession, userId : String) : List<CartItem> post: result <> null

Nome Metodo	+ rimuoviFumetto(session : HttpSession, userId : String, prodId : String) : void
Descrizione	Rimuove un fumetto dal carrello.
Pre-Condizione	context CarrelloService::rimuoviFumetto(session : HttpSession, userId : String, prodId : String) : void pre: session <> null and userId <> null and prodId <> null and session.carrello <> null and session.carrello.getCartItems()->size() > 0 and session.carrello.getCartItems()->exists(p | p <> null and p.id = prodId and p.quantity >= 1)
Post-Condizione	context CarrelloService::rimuoviFumetto(session : HttpSession, userId : String, prodId : String) : void post: session.carrello <> null and session.carrello.getCartItems()->size() >= 0 and session.carrello.getCartItems()->forAll(p | p <> null and p.quantity >= 1) and not session.carrello.getCartItems()->exists(p | p.id = prodId)

Nome Metodo	+ aggiungiFumetto(session : HttpSession, userId : String, prodId : String, pQty : int) : void
Descrizione	Aggiunge un fumetto al carrello.
Pre-Condizione	context CarrelloService::aggiungiFumetto(session : HttpSession, userId : String, prodId : String, pQty : int) : void pre: session <> null and userId <> null and prodId <> null and pQty > 0 and session.cart <> null and session.cart.getCartItems()->exists(p | p.prodotto.id = prodId and p.quantity >= 1
Post-Condizione	context CarrelloService::aggiungiFumetto(session : HttpSession, userId : String, prodId : String, pQty : int) : void post: session.cart->exists(p | p.prodotto.id = prodId and p.quantity = p.quantity + pQty)

Nome Metodo	+ loadCartFromDB(session : HttpSession, email : String) : CartBean
Descrizione	Carica il carrello dal database.
Pre-Condizione	context CarrelloService::loadCartFromDB(session : HttpSession, email : String) : CartBean pre: session <> null and email <> null
Post-Condizione	context CarrelloService::loadCartFromDB(session : HttpSession, email : String) : CartBean post: result <> null

3.3. Gestione Catalogo

Interfaccia	CatalogoService
Descrizione	Vedi descrizione del SSD
Metodi	+ visualizzaCatalogo() : List<ProductBean> + visualizzaDettagli(productId : String) : ProductBean + addFumetto(name : String, description : String, price : float, imagePath : String, genres : String[]) : String + ricercaID(search : String) : List<ProductBean> + ricercaGenere(type : String) : List<ProductBean> + ricercaTitolo(search : String) : List<ProductBean> + rmFumetto(productId : String) : String
Invariante	

Nome Metodo	+ visualizzaCatalogo() : List<ProductBean>
Descrizione	Restituisce la lista di tutti i prodotti nel catalogo.
Pre-Condizione	
Post-Condizione	context CatalogoService::visualizzaCatalogo() : List<ProductBean> post: result <> null and result->forAll(p | p <> null)

Nome Metodo	+ visualizzaDettagli(productId : String) : ProductBean
Descrizione	Restituisce i dettagli di un prodotto specifico.
Pre-Condizione	context CatalogoService::visualizzaDettagli(productId : String) : ProductBean pre: productId <> null
Post-Condizione	context CatalogoService::visualizzaDettagli(productId : String) : ProductBean post: result <> null and result.id = productId

Nome Metodo	+ addFumetto(name : String, description : String, price : float, imagePath : String, genres : String[]) : String
Descrizione	Aggiunge un nuovo fumetto al catalogo.
Pre-Condizione	context CatalogoService::addFumetto(name : String, description : String, price : float, imagePath : String, genres : String[]) : String pre: name <> null and name <> '' and description <> null and description <> '' and price > 0 and imagePath <> null and genres <> null and genres->size() > 0 and not Product.allInstances()->exists(p | p.id = id)
Post-Condizione	context CatalogoService::addFumetto(name : String, description : String, price : float, imagePath : String, genres : String[]) : String

	post: (result = 'Product Added Successfully' and Product.allInstances()->exists(p | p.name = name)) or (result = 'Error Adding Product' and not Product.allInstances()->exists(p | p.name = name))

Nome Metodo	+ ricercaID(search : String) : List<ProductBean>
Descrizione	Cerca prodotti per ID.
Pre-Condizione	context CatalogoService::ricercaID(search : String) : List<ProductBean> pre: search <> null
Post-Condizione	context CatalogoService::ricercaID(search : String) : List<ProductBean> post: result <> null and result->forAll(p | p.id.toLowerCase().indexOf(search.toLowerCase()) > 0)

Nome Metodo	+ ricercaGenere(type : String) : List<ProductBean>
Descrizione	Cerca prodotti per genere.
Pre-Condizione	context CatalogoService::ricercaGenere(type : String) : List<ProductBean> pre: type <> null and type <> ''
Post-Condizione	context CatalogoService::ricercaGenere(type : String) : List(ProductBean) post: result <> null and result->forAll(p | p.generi->exists(g | g.genere.toLowerCase() = type.toLowerCase()))

Nome Metodo	+ ricercaTitolo(search : String) : List<ProductBean>
Descrizione	Cerca prodotti per titolo.
Pre-Condizione	context CatalogoService::ricercaTitolo(search : String) : List<ProductBean> pre: search <> null and search <> ''
Post-Condizione	context CatalogoService::ricercaTitolo(search : String) : List<ProductBean> post: result <> null and result->forAll(p | p.titolo.toLower().includes(search.toLower()))

Nome Metodo	+ rmFumetto(productId : String) : String
Descrizione	Rimuove un fumetto dal catalogo.
Pre-Condizione	context CatalogoService::rmFumetto(productId : String) : String pre: productId <> null
Post-Condizione	context CatalogoService::rmFumetto(productId : String) : String post: (result = 'Product Removed Successfully' and not Product.allInstances()->exists(p | p.id = productId)) or (result = 'Error Removing Product' and Product.allInstances()->exists(p | p.id = productId))

3.4. Gestione Magazzino

Interfaccia	MagazzinoService
Descrizione	Vedi descrizione del SSD
Metodi	+ getScaffaleMagazzino(email : String) : List<ScaffaliBean> + getAllMagazzini() : List<MagazzinoBean>
Invariante	

Nome Metodo	+ getScaffaleMagazzino(email : String) : List<ScaffaliBean>
Descrizione	Restituisce la lista degli scaffali di un magazzino.
Pre-Condizione	context MagazzinoService::getScaffaleMagazzino(email : String) : List<ScaffaliBean> email <> null and email <> '' and User.allInstances()->exists(u | u.email = email)
Post-Condizione	context MagazzinoService::getScaffaleMagazzino(email : String) : List<ScaffaliBean> post: result->forAll(s | User.allInstances()->select(u | u.email = email)->first().magazzino.scaffali->includes(s))

Nome Metodo	+ getAllMagazzini() : List<MagazzinoBean>
Descrizione	Restituisce la lista di tutti i magazzini.
Pre-Condizione	
Post-Condizione	context MagazzinoService::getAllMagazzini() : List<MagazzinoBean> post: result <> null and result->forAll(m | m <> null)

3.5. Gestione Ordine

Interfaccia	OrdineService
Descrizione	Vedi descrizione del SSD
Metodi	+ authorizePayment(product : List<Transaction>, payer : Payer) : String + processaPagamento(paymentId : String, payerId : String) : Payment + getPaymentDetails(paymentId : String) : Payment + getOrdiniNonSpediti() : List<OrderBean> + getOrdiniPerUtente(email : String) : List<OrderBean> + creaOrdine(paymentId : String, payerId : String, email : String, session : HttpSession) : void
Invariante	

Nome Metodo	+ authorizePayment(product : List<Transaction>, payer : Payer) : String
Descrizione	Autorizza il pagamento per un ordine.
Pre-Condizione	context OrdineService::authorizePayment(product : List<Transaction>, payer : Payer) : String pre: product <> null and not product->isEmpty() and payer <> null
Post-Condizione	context OrdineServiceImpl::authorizePayment(product : List(Transaction), payer : Payer) : String post: result <> null and result.size() > 0 and result.substring(1, 24) = "https://www.paypal.com/"

Nome Metodo	+ processaPagamento(paymentId : String, payerId : String) : Payment
Descrizione	Processa il pagamento di un ordine.
Pre-Condizione	context OrdineService::processaPagamento(paymentId : String, payerId : String) : Payment pre: paymentId <> null and payerId <> null
Post-Condizione	context OrdineService::processaPagamento(paymentId : String, payerId : String) : Payment post: result <> null and result.state = 'approved'

Nome Metodo	+ getPaymentDetails(paymentId : String) : Payment
Descrizione	Restituisce i dettagli di un pagamento.
Pre-Condizione	context OrdineService::getPaymentDetails(paymentId : String) : Payment pre: paymentId <> null
Post-Condizione	context OrdineService::getPaymentDetails(paymentId : String) : Payment post: result <> null and result.id = paymentId

Nome Metodo	+ getOrdiniNonSpediti() : List<OrderBean>
Descrizione	Restituisce la lista degli ordini non spediti.
Pre-Condizione	context OrdineService::getOrdiniNonSpediti() : List<OrderBean> pre: true
Post-Condizione	context OrdineService::getOrdiniNonSpediti() : List<OrderBean> post: result <> null and result->forAll(o | o.stato = 'ORDER PLACED')

Nome Metodo	+ getOrdiniPerUtente(email : String) : List<OrderBean>
Descrizione	Restituisce la lista degli ordini di un utente.
Pre-Condizione	context OrdineService::getOrdiniPerUtente(email : String) : List<OrderBean> pre: email <> null
Post-Condizione	context OrdineService::getOrdiniPerUtente(email : String) : List<OrderBean> post: result <> null and result->forAll(o | o.email = email)

Nome Metodo	+ creaOrdine(paymentId : String, payerId : String, email : String, session : HttpSession) : void
Descrizione	Crea un nuovo ordine.
Pre-Condizione	context OrdineService::creaOrdine(paymentId : String, payerId : String, email : String, session : HttpSession) : void pre: paymentId <> null and payerId <> null and email <> null and session <> null
Post-Condizione	context OrdineService::creaOrdine(paymentId : String, payerId : String, email : String, session : HttpSession) : void post: Order.allInstances()->exists(o | o.idPaypal = paymentId and o.emailUtente = email and o.shipped = 'ORDER PLACED')