TechZone

Object Design Document

Gerardo Festa	0512105908
Carmine Federico	0512106094
Emanuele Medaglia	0512105842
Gianluca Astorino	0512105998

Sommario

1. Introduzione

1.1 Object Design Trade-Offs

Per avere la possibilità di implementare il nostro sistema in modo da garantire una riduzione dei tempi e dei costi, si è deciso di utilizzare delle componenti off-the-shelf che rendano più veloce il processo di sviluppo.

Per garantire, inoltre, una personalizzazione del back-end, abbiamo deciso di limitare l'utilizzo di queste componenti al solo front-end (interfaccia utente).

1.1.1 Componenti Off-the-shelf

Il front-end verrà realizzato con l'ausilio del framework Bootstrap. Quest'ultimo è molto utilizzato nell'ambito dell'informatica per la creazione di siti e applicazioni Web, ed è fondamentalmente una libreria basata sull'utilizzo di HTML, CSS, Javascript.

Grazie a Bootstrap si riducono i tempi di sviluppo dell'interfaccia utente e si utilizzano delle strutture e una palette cromatica familiare all'utente, in quanto presente su molti altri siti web.

1.1.2 Design Patterns

Abbiamo utilizzato il design pattern *Façade.*
Ogni sottosistema che andiamo a realizzare implementa parte della logica di business dell'intero sistema. Per fornire un accesso più compatto ai sottosistemi, abbiamo deciso di mascherare la business logic tramite dei Façade, implementati dalle classi Manager a cui i Control effettueranno l'accesso. Tali classi Manager incapsulano tutta la business logic, invocando le operazioni delle classi che effettuano l'accesso al database e delle classi Entity.

1.2 Linee guida per la documentazione dell'interfaccia

1.2.1 Nomenclatura delle componenti

Per garantire l'integrità del progetto e la comprensione delle funzionalità di ogni componente, è necessario rispettare le seguenti linee guida.

1.2.1.1 Nomi delle classi

- Ogni nome della classe deve cominciare con una lettera maiuscola
- Ogni nome della classe deve essere univoco
- Ogni classe che modella la connessione al DB e le query associate all'entità di cui si occupa ha nel nome "DAO"
- Ogni classe servlet ha nel nome "Control"
- Ogni classe che modella la logica di business ha la terminazione del nome "Manager".
    - N.B. Attenzione, sono presenti delle classi che contengono Manager nel nome ma non terminano con esso. Queste non rispettano la legge sopra descritta. Infatti, si riferiscono alla tipologia di utente manager, amministratore del sito.

1.2.1.2 Nomi dei metodi

- Ogni nome del metodo deve cominciare con una lettera minuscola

1.2.1.3 Nomi delle pagine front-end

- Ogni nome della pagina deve cominciare con una lettera maiuscola

1.2.1.4 Organizzazione delle componenti

- Tutte le classi che compongono un layer devono risiedere nello stesso package.
- Tutte le componenti che realizzano l'interfaccia grafica devono trovarsi in */TechZone/WebContent/.* Nello specifico:
    - Gli script JavaScript devono trovarsi in */TechZone/WebContent/scripts/*
    - I file CSS devono trovarsi in */TechZone/WebContent/css/*

1.3 Definizioni, acronimi e abbreviazioni

DAO: pattern architetturale per la gestione della persistenza

SDD: System Design Document

RAD: Requirement Analysis Document

DB: Database

1.4 Relazioni con gli altri documenti

Sono previsti riferimenti al SDD per quanto riguarda la suddivisione in sottosistemi e al RAD per quanto riguarda il Class Diagram

2. Packages

2.1 Divisione in pacchetti

Come già detto nel SDD, il sistema si basa su un'architettura three-tier. Il sistema di accesso ai dati persistenti viene realizzato implementando il DAO con un DriverManager che utilizza JDBC.

2.2 Organizzazione del codice in file

- Ogni classe sarà collocata nel relativo file
- Ogni pacchetto avrà come prefisso *it.techzone*
- Ogni package sarà mappato nel relativo percorso /TechZone/src/it/techzone/
- L'interfaccia utente sarà collocata all'interno della cartella */TechZone/WebContent/*

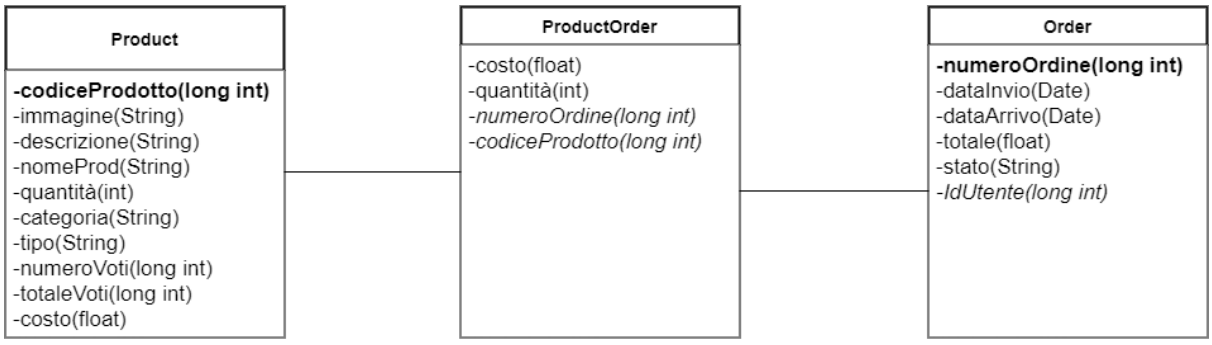
3. Interfacce delle classi

3.1 Trasformazione class diagram in modello relazionale

Per quanto riguarda la ristrutturazione del Class Diagram presentato nel RAD, in modo da formare un corretto modello relazionale, abbiamo apportato alcune modifiche.

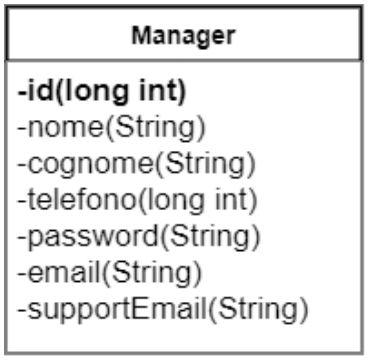
Innanzitutto, rispetto al Class Diagram, il modello relazionale in questione non presenta "Cart" e "ProductCart": questo perché non c'è bisogno di creare entità nel Database per renderli persistenti, dato che abbiamo progettato di farlo tramite la gestione di una sessione.

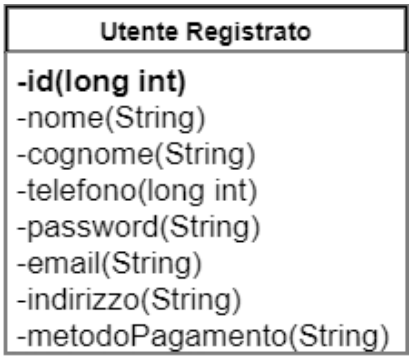
Per quanto riguarda "Order" e "ProductOrder", è necessario che queste siano entità persistenti del Database. Esse sono collegate nel modello relazionale all'entità "Product", ma rispetto al Class Diagram, "ProductOrder" ora si frappone tra "Product" e "Order" tramite relazioni. Esso è collegato a "Product" da una relazione 1 a 1, e a "Order" da una relazione 1 a molti (un "Order" può avere tanti "ProductOrder"). "ProductOrder" ha come chiave composta le chiavi primarie di "Product" e "Order". Su "Order" viene posta una collection di "ProductOrder".



Per quanto riguarda la gestione della generalizzazione tra “Utente”, “Manager” e “UtenteRegistrato”, c’è poca differenza pratica tra le varie tipologie di scioglimento per quanto riguarda il nostro progetto, ma fondamentalmente ci sono due scelte sensate: riunire le entità figlie nel padre, oppure riunire l’entità padre nei figli, visto che l’unica con relazioni al di fuori della gerarchia è Utente Registrato. Nel primo caso, si dovrebbe inserire nel padre un attributo che definisce se un “Utente Registrato” è manager oppure no, mentre nel secondo caso è semplicemente necessario aggiungere tutti gli attributi di “Utente” ai figli.

Abbiamo optato per il secondo caso, mantenendo quindi le entità “Utente Registrato” e “Manager”. Nell’implementazione Java, abbiamo mantenuto la classe padre come *astratta*.





Nella relazione uno a molti che collega “Utente Registrato” a “Order”, in Java abbiamo previsto una collection di “Order” su “UtenteRegistrato” e un riferimento a “Utente Registrato” su “Order”.

3.2 Specifica interfacce

Mostriamo ora i metodi con *pre-conditions*, *post-conditions* ed *invariante* per ogni classe. Ogni tabella è realizzata con sintassi OCL.

3.2.1 Bean

Nome Classe	User
Sottosistema	Gestione Utenti
Variabili di istanza	- id:long int - nome:String - cognome:String - telefono:long int - password:String - email:String
Descrizione	Questa entity contiene i metodi per creare un oggetto di tipo User, con i relativi metodi per ottenere e impostare i suoi parametri. Si tratta della classe padre di Utente Registrato e Manager.
Signature metodi	+ User():User + User(id:long int, nome:String, cognome:String, telefono:long int, password:String, email:String, indirizzo:String, metodoPagamento:String):User + User(nome:String, cognome:String, telefono:long int, password:String, email:String, indirizzo:String, metodoPagamento:String):User + getId():longInt + getNome():String + getCognome():String + getTelefono():long int + getPassword():String + getEmail():String + setId(id:long int):void + setNome(nome:String):void + setCognome(cognome:String):void + setTelefono(telefono:long int):void + setPassword(password:String):void + setEmail(email:String):void
Pre-condizioni	N/A
Post-condizioni	• context User::User() **post:** • context User::User(id,nome,cognome,telefono,password,email) **post:** result.id = id and result.nome = nome and result.cognome = cognome and result.telefono = telefono and result.password = password and result.email = email • context User::User(nome,cognome,telefono,password,email)

	post: result.nome = nome and result.cognome = cognome and result.telefono = telefono and result.password = password and result.email = email • context User::getNome() **post:** result.nome = nome • context User::getCognome() **post:** result.cognome = cognome • context User::getEmail() **post:** result.email = email • context User::getTelefono() **post:** result.telefono = telefono • context User::getPassword() **post:** result.password = password • context User::setNome(nome) **post:** this.nome = nome • context User::setCognome(cognome) **post:** this.cognome = cognome • context User::setTelefono(telefono) **post:** this.telefono = telefono • context User::setPassword(password) **post:** this.password = password • context User::setEmail(email) **post:** this.email = email
Invarianti	

Nome Classe	UtenteRegistrato <<extends>> User
Sottosistema	Gestione Utenti
Variabili di istanza	- indirizzo:String - metodoPagamento:String - ordini:ArrayList<Order>
Descrizione	Questa entity contiene i metodi per creare un oggetto di tipo UtenteRegistrato, con i relativi metodi per ottenere e impostare i suoi parametri. Si tratta di un utente registrato. Si tratta di una sottoclasse di User.
Signature metodi	+ UtenteRegistrato():UtenteRegistrato + UtenteRegistrato(id:long int, nome:String, cognome:String, telefono:long int, password:String, email:String, indirizzo:String, metodoPagamento:String):UtenteRegistrato + UtenteRegistrato(nome:String, cognome:String, telefono:long int, password:String, email:String, indirizzo:String, metodoPagamento:String):UtenteRegistrato + getIndirizzo():String + getOrdini():ArrayList<Order> + getMetodoPagamento():String + setIndirizzo(indirizzo:String):void + setMetodoPagamento(metodoPagamento:String):void + setOrdini(ordini:ArrayList<Order>):void
Pre-condizioni	N/A
Post-condizioni	• context UtenteRegistrato::UtenteRegistrato() **post:** • context User::UtenteRegistrato(id,nome,cognome,telefono,password,email,indirizzo,metodoPagamento) **post:** result.id = id and result.nome = nome and result.cognome = cognome and result.telefono = telefono and result.password = password and result.email = email and result.indirizzo = indirizzo and result.metodoPagamento = metodoPagamento and result.ordini!=null • context UtenteRegistrato::UtenteRegistrato(nome,cognome,telefono,password,email,indirizzo,metodoPagamento) **post:** result.nome = nome and result.cognome = cognome and result.telefono = telefono and result.password = password and

	result.email = email and result.indirizzo = indirizzo and result.metodoPagamento = metodoPagamento and result.ordini!=null • context User::getIndirizzo() **post:** result.indirizzo = indirizzo • context User::getMetodoPagamento() **post:** result.metodoPagamento = metodoPagamento • context User::getOrdini() **post:** result.ordini=ordini • context User::setIndirizzo(indirizzo) **post:** this.indirizzo = indirizzo • context User::setMetodoPagamento(metodoPagamento) **post:** this.metodoPagamento = metodoPagamento • context User::setOrdini(ordini) **post:** this.ordini = ordini
Invarianti	

Nome Classe	Product
Sottosistema	Gestione Prodotti
Variabili di istanza	- **codice: long int** - costo:float - immagine: ArrayList<Byte> - descrizione:String - nomeProd:String - quantita:int - categoria:String - tipo:String
Descrizione	Questa entity contiene i metodi per creare un oggetto di tipo Product, con i relativi metodi per ottenere e impostare i suoi parametri
Signature metodi	+ Product():Product + Product(costo:float, codice:long int, immagine:ArrayList<Byte>, descrizione:String, quantità: int, nomeProd:String, categoria:String, tipo:String):Product + Product(costo:float, immagine:String, descrizione:String, quantità: int,nomeProd:String, categoria:String, tipo:String):Product + getCosto(): float + getCodice():long int + getImmagine():ArrayList<Byte>

	+ getDescrizione():String + getNomeProd():String + getQuantita():int + getCategoria():String + getTipo():String + setCodice(codice:long int):void + setCosto(costo:float):void + setImmagine(immagine:ArrayList<Byte>):void + setDescrizione(descrizione:String):void + setNomeProd(nomeProd:String):void + setQuantita(quantita:int):void + setCategoria(categoria:String):void + setTipo(tipo:String):void
Pre-condizioni	N/A
Post-condizioni	• context Product::Product() **post:** • context Product::Product(codice,costo, immagine,descrizione,nomeProd,quantita,categoria,tipo) **post:** result.codice = codice and result.costo = costo and result.immagine = immagine and result.descrizione = descrizione and result.nomeProd = nomeProd and result.quantita = quantita and result.categoria = categoria and result.tipo = tipo • context Product::Product(costo, immagine,descrizione,nomeProd,quantita,categoria,tipo) **post:** result.costo = costo and result.immagine = immagine and result.descrizione = descrizione and result.nomeProd = nomeProd and result.quantita = quantita and result.categoria = categoria and result.tipo = tipo • context Product::getCodice() **post:** result.codice = codice • context Product::getCosto() **post:** result.costo = costo • context Product::getImmagine() **post:** result.immagine = immagine • context Product::getDescrizione() **post:** result.descrizione = descrizione

	• context Product::getNomeProd() **post:** result.nomeProd = nomeProd • context Product::getQuantita() **post:** result.quantita = quantita • context Product::getCategoria() **post:** result.categoria = categoria • context Product::getTipo() **post:** result.tipo = tipo • context Product::setCodice(codice) **post:** this.codice = codice • context Product::setCosto(costo) **post:** this.costo = costo • context Product::setImmagine(immagine) **post:** this.immagine = immagine • context Product::setDescrizione(descrizione) **post:** this.descrizione = descrizione • context Product::setNomeProd(nomeProd) **post:** this.nomeProd = nomeProd • context Product::setQuantita(quantita) **post:** this.quantita = quantita • context Product::setCategoria(categoria) **post:** this.categoria = categoria • context Product::setTipo(tipo) **post:** this.tipo= tipo
Invarianti	

Nome Classe	ProductOrder
Sottosistema	Gestione Prodotti, Gestione Ordini
Variabili di Istanza	- costo: float - quantita: int - prodotto: Product
Descrizione	Questa entity descrive un prodotto ordinato di tipo ProductOrder, con i relativi metodi per ottenere e impostare i suoi parametri
Signature metodi	+ ProductOrder():ProductOrder + ProductOrder(costo:float quantità:int, prodotto:Product):ProductOrder + ProductOrder(prodotto_carr:ProductCart): ProductOrder + getCosto():float + getQuantita():int + getProduct():Product + setCosto(costo:float):void + setQuantita(quantita:int):void + setProduct(prodotto:Product):void
Pre-Condizioni	
Post-Condizioni	• context ProductOrder::ProductOrder() **post:** • context ProductOrder::ProductOrder(costo,quantita,prodotto) **post:** result.costo = costo and result.quantita = quantita and result.prodotto = prodotto and • context ProductOrder::ProductOrder(costo,quantita,prodotto, ordine) **post:** result.costo = costo and result.quantita = quantita and result.prodotto = prodotto and result.ordine = ordine • context ProductOrder::getCosto() **post:** result.costo = costo • context ProductOrder::getQuantita() **post:** result.quantita = quantita • context ProductOrder::getProduct() **post:** result.prodotto = prodotto • context ProductOrder::setCosto(costo) **post:** this.costo = costo • context ProductOrder::setQuantita(quantita) **post:** this.quantita = quantita

	• context ProductOrder::setProduct(prodotto) **post:** this.prodotto = prodotto
Invarianti	N/A

Nome Classe	Order
Sottosistema	Gestione ordini
Variabili di istanza	- numeroOrdine: long int - dataInvio: Timestamp - dataArrivo: Timestamp - totale: float - stato: String - utente: UtenteRegistrato - prodotti: ArrayList<ProductOrder>
Descrizione	Questa entity contiene i metodi per creare un oggetto di tipo Order, con i relativi metodi per ottenere e impostare i suoi parametri
Signature metodi	+ Order():Order + Order(numeroOrdine:long int,dataInvio:Timestamp, dataArrivo:Timestamp, totale:float, stato:String, utente:UtenteRegistrato):Order + Order(dataInvio:Timestamp, dataArrivo:Timestamp, totale:float, stato:String, utente: UtenteRegistrato, prodotti: ArrayList<ProductOrder>):Order + getNumeroOrdine():long int + getDataInvio():Timestamp + getDataArrivo():Timestamp + getTotale():float + getStato():String + getIdUtente():long int + getProdotti():ArrayList<ProductOrder> + setNumeroOrdine(numeroOrdine:long int):void + setDataInvio(dataInvio:Timestamp):void + setDataArrivo(dataArrivo:Timestamp):void + setTotale(totale:float):void + setStato(stato:String):void + setUtente(utente:UtenteRegistrato):void + setProdotti(prodotti: ArrayList<ProductOrder>): void
Pre-Condizioni	N/A
Post-Condizioni	• context Order::Order() **post:**

	• context Order:Order(numeroOrdine,dataInvio,dataArrivo, totale,stato,utente) **post:** result.numeroOrdine = numeroOrdine and result.dataInvio = dataInvio and result.dataArrivo = dataArrivo and result.totale = totale and result.stato = stato and result.utente = utente • context Order:Order(dataInvio,dataArrivo, totale,stato,utente) **post:** result.dataInvio = dataInvio and result.dataArrivo = dataArrivo and result.totale = totale and result.stato = stato and result.utente = utente • context Order::getNumeroOrdine() • **post:** result.numeroOrdine = numeroOrdine • context Order::getDataInvio() **post:** result.dataInvio = dataInvio • context Order::getDataArrivo() **post:** result.dataArrivo = dataArrivo • context Order::getTotale() **post:** result.totale = totale • context Order::getStato() **post:** result.stato = stato • context Order::getUtente() **post:** result.utente = utente • context Order::getProdotti() **post:** result.prodotti = prodotti • context Order::setNumeroOrdine(numeroOrdine) **post:** this.numeroOrdine = numeroOrdine • context Order::setDataInvio(dataInvio) **post:** this.dataInvio = dataInvio • context Order::setDataArrivo(dataArrivo)

	post: this.dataArrivo = dataArrivo • context Order::setTotale(totale) **post:** this.totale = totale • context Order::setStato(stato) **post:** this.stato = stato • context Order::setUtente(utente) **post:** this.utente = utente • context Order::setProdotti(prodotti) **post:** this.prodotti = prodotti
Invarianti	N/A

Nome Classe	Cart
Sottosistema	Gestione Carrello
Variabili di istanza	- productList:ArrayList<ProductCart> - prezzoTotale:float
Descrizione	Questa entity contiene i metodi per creare un oggetto di tipo Carrello, con i relativi metodi per ottenere e impostare i suoi parametri
Signature metodi	+ Cart():Cart + Cart(productList:ArrayList<ProductCart>, prezzoTotale:float):Cart + getProductList():ArrayList<ProductCart> + getPrezzoTotale:float + setProductList(productList: ArrayList<ProductCart>):void + setPrezzoTotale(prezzoTotale:float):void + addToCart(prodotto:Product,quantita:int):boolean + removeFromCart(prodotto:Product):boolean + setQuantityInCart(prodotto:Product, quantita:int):boolean - calcolaPrezzoTotale:void - isInCart(prodotto:Product):int
Pre-Condizioni	• context Cart::addToCart(prodotto,quantita) **pre:** quantita<=quantita(relativa al Product) • context Cart::setQuantityInCart(prodotto,quantita) **pre:** quantita<=quantita(relativa al Product)
Post-Condizioni	• context Cart:Cart(productList, prezzoTotale)

	post: result.productList = productList and result.prezzoTotale = prezzoTotale
	
	• context Cart::getProductList()
	post: result.productList = productList
	
	• context Cart::getPrezzoTotale()
	post: result.prezzoTotale = prezzoTotale
	
	• context Cart::setProductList(productList)
	post: this.productList = productList
	
	• context Cart::setPrezzoTotale(prezzoTotale)
	post: this.prezzoTotale = prezzoTotale
	
	• context Cart::addToCart(prodotto,quantita)
	post: productList->includes(ProductCart c|c.prodotto=prodotto and c.quantita=quantita)
	
	• context Cart::removeFromCart(prodotto)
	post: productList->not exists(ProductCart c|c.prodotto=prodotto)
	
	• context Cart::setQuantityInCart(prodotto,quantita)
	post: productList->includes(ProductCart c|c.prodotto=prodotto and c.quantita=quantita)
	
	• context Cart::isInCart(prodotto)
	if(productList->exists(p|p.prodotto=prodotto))
	return posizione
	else
	return -1

Invarianti	

Nome Classe	ProductCart
Sottosistema	Gestione carrello Gestione prodotti

Variabili di istanza	- quantita(int) - prodotto(Product)
Descrizione	Questa entity contiene i metodi per creare un oggetto di tipo Product Cart, con i relativi metodi per ottenere e impostare i suoi parametri
Signature metodi	+ ProductCart():ProductCart + ProductCart(quantita,prodotto):ProductCart + getQuantita():int + getProduct():Product + setQuantita(quantita:int):void + setProduct(prodotto:Product):void
Pre-Condizioni	N/A
Post-Condizioni	• context ProductCart::ProductCart() **post:** • context ProductCart::ProductCart(quantita,prodotto) **post:** result.quantita = quantita and result.prodotto = prodotto • context ProductCart::getQuantita() **post:** result.quantita = quantita • context ProductCart::getProduct() **post:** result.prodotto = prodotto • context ProductCart::setQuantita(quantita) **post:** this.quantita= quantita • context ProductCart::setProduct(prodotto) **post:** this.prodotto = prodotto
Invarianti	N/A

Nome Classe	Manager <<extends>> User
Sottosistema	Gestione Utenti
Variabili di Istanza	- supportEmail(String)
Descrizione	Questa entity contiene i metodi per creare un oggetto di tipo Manager, con i relativi metodi per ottenere e impostare i suoi parametri
Signature metodi	+ Manager():Manager + Manager(int id, String nome, String cognome, longint telefono, String password, String email, String supportEmail):Manager + Manager(String nome, String cognome, longint telefono, String password, String email, String supportEmail):Manager + getSupportEmail:String + setSupportEmail:void
Pre-Condizioni	N/A
Post-Condizioni	• context Manager()::Manager **post:** • context Manager::Manager(id,nome,cognome,telefono,password,email,supportEmail) **post:** result.id = id and result.nome = nome and result.cognome = cognome and result.telefono = telefono and result.password = password and result.email = email and result.supportEmail = supportEmail • context Manager::Manager(nome,cognome,telefono,password,email,supportEmail) **post:** result.nome = nome and result.cognome = cognome and result.telefono = telefono and result.password = password and result.email = email and result.supportEmail = supportEmail • context Manager::getSupportEmail() **post:** result.supportEmail = supportEmail • context Manager::setSupportEmail(supportEmail) **post:** this.supportEmail = supportEmail
Invarianti	N/A

3.2.2 DAO

Nome Classe	UserDAO
Sottosistema	Gestione Utenti
Variabili di istanza	
Descrizione	Consente l'interfacciamento al DB via JDBC. In particolare cura le interrogazioni relative agli utenti.
Signature metodi	+ doSaveUser(user: UtenteRegistrato): boolean + authenticate(email: String, password: String): UtenteRegistrato + authenticateManager(email: String, password: String): Manager + checkEmailUsed(email: String): Boolean + doRetrieveByEmail(email:String):UtenteRegistrato + doRetrieveById(id:long):UtenteRegistrato
Pre-Condizioni	• context userdao::doSaveUser(user) **pre:** user<>null and users->not exists(u|u.email=emai) • context userdao::authenticate(email,password) **pre:** email<>null and password<>null and users->exists(u|u.email=email and u.password=password) • context userdao::authenticateManager(email,password) **pre:** email<>null and password<>null and managers->exists(m|m.email=email and m.password=password) • context userdao::checkEmailUsed(email) **pre:** email<>null • context userdao::doRetrieveByEmail(email) **pre:**email<>null and users->exist(u|u.email=email) • context userdao::doRetrieveById(id) **pre:**id<>null and users->exist(u|u.id=id)
Post-Condizioni	• context userdao::doSaveUser(user) **post:** users->exists(u|u.email=user.email) • context userdao::authenticate(email,password) **post:** result = users.select(u|u.email=email && u.password = password)

	• context userdao::authenticateManager(email,password) **post:** result = managers.select(m|m.email=email && m.password = password) • context userdao::checkEmailUsed(email) **post:** users->exists(u|u.email=email) • context userdao::doRetrieveByEmail(email) **post:** result = users(u|u.email=email) • context userdao::doRetrieveById(id) **post:** result = users(u|u.id=id)
Invarianti	N/A

Nome Classe	ProductDAO
Sottosistema	Gestione Prodotti
Variabili di istanza	
Descrizione	Consente l'interfacciamento al DB via JDBC. In particolare cura le interrogazioni relative ai prodotti.
Signature metodi	+ retrieveProductById(id: long int): Product + retrieveProductsByName(nameProd: String): ArrayList<Product> + retrieveProductsByCat(categoria: String): ArrayList<Product> + doDeleteProduct(id: long int): Boolean + doUpdateProduct(prodotto: Product): Boolean + doRetrieveAll(order:String):ArrayList<Product>
Pre-Condizioni	• context productdao::retrieveProductById(id) **pre:** id<>null and products->exists(p|p.id=id) • context productdao::retrieveProductsByName(nameProd) **pre:** nameProd<>null • context productdao::retrieveProductsByCat(categoria) **pre:** categoria<>null • context productdao::doDeleteProduct(id) **pre:** products->exists(p|p.id=id) • context productdao::doUpdateProduct(prodotto) **pre:** prodotto<>null and products ->includes(p|p.id = prodotto.id)

Post-Condizioni	• context productdao::retrieveProductById(id) **post:** result = products->select(p|p.id=id) • context productdao::retrieveProductsByName(nameProd) **post:** result = products->select(p|p.nameProd->includes (nameProd)) • context productdao::retrieveProductsByCat(categoria) **post:** result = products->select(p|p.categoria=categoria) • context productdao::doDeleteProduct(id) **post:** result = products->not exists(p|p.id=id) • context productdao::doUpdateProduct(prodotto) **post:** result = products->includes(p|p=prodotto) • context productdao::doRetrieveAll(order) **post:** result = products->select(p)
Invarianti	N/A

Nome Classe	OrderDAO
Sottosistema	Gestione Ordini
Variabili di istanza	
Descrizione	Consente l'interfacciamento al DB via JDBC. In particolare cura le interrogazioni relative agli ordini.
Signature metodi	+ doSaveOrder(order: Order): boolean + retrieveOrdersByMail(email: String): ArrayList<Order> + retrieveOrderById(id: long): Order + doUpdateOrder(order: Order): Boolean
Pre-Condizioni	• context orderdao::doSaveOrder(order) **pre:** order<>null • context orderdao::retrieveOrderByMail(email) **pre:** email<>null and orders->exists(o|o.utente.email=email) • context orderdao::retrieveOrdersById(id) **pre:** id<>null and orders->exists(o|o.id=id) • context orderdao::doUpdateOrder(order) **pre:** order<>null and orders ->exists(o|o.numeroOrdine =order.numeroOrdine)

Post-Condizioni	• context orderdao::doSaveOrder(order) **post:** orders->exists(o|o.numeroOrdine=order.numeroOrdine) • context orderdao::retrieveOrdersByMail(email) **post:** result =orders->select(o|o.email=email) • context orderdao::retrieveOrdersById(id) **post:** result =orders->select(o|o.id=id) • context orderdao::doUpdateOrder(order) **post:** result = orders->includes(o|o=order)
Invarianti	N/A

3.2.3 Manager

Nome Classe	UserManager
Sottosistema	Gestione Utenti
Variabili di istanza	- userdao: UserDao
Descrizione	Contiene la logica di business per la gestione degli utenti. Si interfaccia al Database per mezzo di UserDAO.
Signature metodi	+ saveUser(nome:String, cognome:String, telefono: long int, password:String, email:String, indirizzo:String, metodoPagamento:String):UtenteRegistrato + emailAlreadyUsed(email: String):boolean + authentication(email: String, password: String):UtenteRegistrato + authenticationManager(email: String, password: String):Manager
Pre-Condizioni	• context UserManager::saveUser(nome,cognome,telefono,password,email,indirizzo,metodoPagamento) **pre:** emailAlreadyUsed(email)=false • context UserManager:authentication(email,password) **pre:** password <> null and users->exists(u|u.password=password and u.email=email) and password.leght<=18 and password.leght>=6 • context UserManager:authenticationManager(email,password) **pre:** password <> null and users->exists(u|u.password=password and u.email=email) and password.legth<=18 and password.legth>=6 and email <> null
Post-Condizioni	• context UserManager::saveUser(nome, cognome, telefono, password, email, indirizzo, metodoPagamento) **post:** result = userdao.doSaveUser(UtenteRegistrato u| u.nome = nome, u.cognome = cognome,u.telefono = telefono,u.password = password, u.email = email, u.indirizzo = indirizzo,u.metodoPagamento = metodoPagamento) • context UserManager::emailAlreadyUsed(email) **post:** result = userdao.checkEmailUsed(email) • context UserManager::authentication(email,password) **post:** result = userdao.authenticate(email,password)

	• context UserManager::authenticationManager(email,password) **post:** result = userdao.authenticateManager(email,password)
Invarianti	N/A

Nome Classe	ProductManager
Sottosistema	Gestione Prodotti
Variabili di istanza	- productdao: ProductDao
Descrizione	Contiene la logica di business per la gestione dei prodotti. Si interfaccia al Database per mezzo di ProductDAO.
Signature metodi	+ retrieveProduct(id: long int): Product + searchProductsByName(nameProd: String): ArrayList<Product> + searchProductsByCat(categoria: String): ArrayList<Product> + deleteProduct(id: long): Boolean + updateProduct(idProdotto: long, descrizione: String, nomeProd: String, quantita: int, categoria: String, tipo: String, costo: float"): Boolean + getAllProducts(ordinevisualizzazione: String): ArrayList<Product>
Pre-Condizioni	• context ProductManager::retrieveProduct(id) **pre:** id<>null • context ProductManager::deleteProduct(id) **pre:** id<>null • context ProductManager::updateProduct(idProdotto, descrizione, nomeProd, quantita, categoria, tipo, costo) **pre:** prodotto<>null, descrizione<>null, nomeProd<>null, quantita>=0, categoria<>null, tipo<>null, costo>0 • context ProductManager::searchProductsByName(nameProd) **pre:** nameProd<>null • context ProductManager::searchProductsByCat(categoria) **pre:** categoria<>null

Post-Condizioni	• context ProductManager::retrieveProduct(id) **post:** result = productdao.retrieveProductById(id) • context ProductManager::searchProductsByName(nameProd) **post:** result = productdao.retrieveProductsByName(nameProd) • context ProductManager::searchProductsByCat(categoria) **post:** result = productdao.retrieveProductsByCat(categoria) • context ProductManager::deleteProduct(id) **post:** result = productdao.doDeleteProduct(id) • context ProductManager::updateProduct(prodotto) **post:** result = productdao.doUpdateProduct(prodotto) • context ProductManager::getAllProducts() **post:** result = productdao.doRetrieveAll(“”)
Invarianti	N/A

Nome Classe	OrderManager
Sottosistema	Gestione Ordini
Variabili di istanza	- orderdao: OrderDao
Descrizione	Contiene la logica di business per la gestione degli ordini. Si interfaccia al Database per mezzo di OrderDAO.
Signature metodi	+ placeOrder(user: UtenteRegistrato, cart: Cart): boolean + searchOrders(email: String): ArrayList<Order> + getOrdersByMail(email: String): ArrayList<Order> + getOrderById(id: long): Order + checkStatus(idOrder: long, status: String): Boolean + changeStatus(idOrder: long, status: String): Boolean
Pre-Condizioni	• context OrderManager::checkStatus(idOrder,status) **pre:** status=”In preparazione” or status=“Spedito” or status=“Spedizione in ritardo” or status=”Pacco smarrito” or status=”In consegna” or status=”Consegnato”, • context OrderManager::changeStatus(idOrder,status) **pre:** status=”In preparazione” or status=“Spedito” or status=“Spedizione in ritardo” or status=”Pacco smarrito” or status=”In consegna” or status=”Consegnato”
Post-Condizioni	• context OrderManager::placeOrder(user,cart) **post:** result=orderdao.doSaveOrder(user,cart) • context OrderManager:searchOrders(email) **post:** result = orderdao.retrieveOrders(email) • context OrderManager::getOrdersByMail(email) **post:** result = orderdao.retrieveOrdersByMail(id) • context OrderManager::getOrderById(id) **post:** result = orderdao.retrieveOrdersById(id) • context OrderManager::checkStatus(idOrder, status) **post:**

	if getOrderById(id).stato=”In preparazione”
	if(status=”Spedito”)
	result=true
	else
	result=false
	endif(getOrderById(id).stato=”Spedito”)
	if(status=”In consegna” || status=”Spedizione in ritardo”)
	result=true
	else
	result=false
	endif(getOrderById(id).stato=”In consegna”)
	if(status=”Spedizione in ritardo” || status=”Consegnato”)
	result=true
	else
	result=false
	endif(getOrderById(id).stato=”Spedizione in ritardo”)
	if(status=”Spedito” || status=”Pacco smarrito”)
	result=true
	else
	result=false
	endif(getOrderById(id).stato=”Consegnato”)
	result false
	endif(getOrderById(id).stato=”Pacco smarrito”)
	result false
	
	• context OrderManager::changeStatus(idOrder,status)
	post: orderdao.getOrderById(idOrder).getStatus()=status

Invarianti	N/A

Nome Classe	CartManager
Sottosistema	Gestione Carrello
Variabili di istanza	- productdao: ProductDao
Descrizione	Contiene la logica di business per la gestione del carrello.
Signature metodi	+ cartExists(session: HttpSession): boolean + retrieveCart(session: HttpSession): Cart + newCart(session: HttpSession): Cart + checkQuantity(idProd:long , quantità:int, session: HttpSession): boolean + modCart(idProd: long, quantità: int, sessione: HttpSession): Boolean
Pre-Condizioni	• context Cart::modCart(idProd, quantità, sessione) **pre:** idProd<>null and quantità>=0 • context Cart::checkQuantity(idProd, quantità, session) **pre:** idProd<>null and quantità>=0
Post-Condizioni	• context CartManager:: cartExists(session) **post:** if session->include(Cart c) return result = true else result = false • context CartManager:: retrieveCart(session) **post:** if session->include(Cart c) result= c else result= newCart(session) • context CartManager::newCart(session) **post:** result= Cart() • context CartManager::checkQuantity(idProd, quantita, session) **post:** if productdao.retrieveProductById(idProd).getQuantity() < quantita result=false else result=true
Invarianti	N/A

3.2.4 Control

Nome Classe	RegisterControl
Sottosistema	Control
Variabili di istanza	+ um: UserManager
Descrizione	Da questo Control viene gestita la fase di registrazione dell'utente.
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session -> not includes(utente) • session -> not includes(manager)
Post-Condizioni	context RegisterControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	LoginControl
Sottosistema	Control
Variabili di istanza	+ um: UserManager
Descrizione	Da questo Control viene gestita la fase di login dell'utente.
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session -> not includes(utente) • session -> not includes(manager)
Post-Condizioni	context LoginControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	OrderPlaceControl
Sottosistema	Control
Variabili di istanza	+ om: OrderManager
Descrizione	Da questo Control vengono gestite le operazioni che riguardano il completamento di un acquisto
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->includes(utente) • session ->includes(carrello)
Post-Condizioni	context OrderPlaceControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	LogoutControl
Sottosistema	Control
Variabili di istanza	
Descrizione	Da questo Control viene gestita la fase di logout dell'utente.
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->includes(utente) || session ->includes(manager)
Post-Condizioni	context LogoutControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	ProductCatalogueControl
Sottosistema	Control
Variabili di istanza	+ pm: ProductManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla ricerca di prodotti per categoria o per nome
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• request ->includes(q) • request ->exists(by| by==categoria || by == nome)
Post-Condizioni	context ProductCatalogueControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	ProductViewControl
Sottosistema	Control
Variabili di istanza	+ pm: ProductManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla visualizzazione di un prodotto o una lista di prodotti.
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	
Post-Condizioni	context ProductViewControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	DeleteProductControl
Sottosistema	Control
Variabili di istanza	+ pm: ProductManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla rimozione di un prodotto
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->includes(manager) • request ->includes(idProd)
Post-Condizioni	context DeleteProductControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	UpdateProductControl
Sottosistema	Control
Variabili di istanza	+ pm: ProductManager
Descrizione	Da questo Control vengono gestite le operazioni relative all'aggiornamento di un prodotto
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->includes(manager) • request ->includes(action) || request ->includes(idProd)
Post-Condizioni	context UpdateProductControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	CartViewControl
Sottosistema	Control
Variabili di istanza	+ cm: CartManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla visualizzazione del carrello
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->not includes(manager)
Post-Condizioni	context CartViewControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	AddToCartControl
Sottosistema	Control
Variabili di istanza	+ cm: CartManager
Descrizione	Da questo Control vengono gestite le operazioni relative all'aggiunta di prodotti al carrello.
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->not includes(manager)
Post-Condizioni	context AddToCartControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	ModInCartControl
Sottosistema	Control
Variabili di istanza	+ cm: CartManager
Descrizione	Da questo Control vengono gestite le operazioni relative all'aggiunta/sottrazione/rimozione di prodotti dal carrello
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session -> not includes(manager)
Post-Condizioni	context ModInCartControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	UserOrderSearchControl
Sottosistema	Control
Variabili di istanza	+ om: OrderManager
Descrizione	Da questo Control vengono gestite le operazioni da parte di uno specifico utente relative alla visualizzazione degli ordini da lui effettuati
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session->includes(utente)
Post-Condizioni	context UserOrderSearchControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	UserOrderViewControl
Sottosistema	Control
Variabili di istanza	+ om: OrderManager
Descrizione	Da questo Control vengono gestite le operazioni da parte di uno specifico utente relative alla visualizzazione dei dettagli di un ordine da lui effettuato
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session->includes(utente) • request ->includes(idOrd)
Post-Condizioni	context UserOrderViewControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	OrderStatusControl
Sottosistema	Control
Variabili di istanza	+ om: OrderManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla gestione/cambiamento dello stato dell'ordine
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session->includes(manager) • request->includes(orderId)
Post-Condizioni	context OrderStatusControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	ManagerOrderSearchIdControl
Sottosistema	Control
Variabili di istanza	+ om: OrderManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla ricerca di ordini specifici tramite inserimento di id.
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session -> includes(manager) • request -> includes(idOrder)
Post-Condizioni	context ManagerOrderSearchIdControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	ManagerOrderSearchMailControl
Sottosistema	Control
Variabili di istanza	+ om: OrderManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla ricerca di ordini specifici tramite inserimento di un'email
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->includes(manager) • request ->includes(mailOrd)
Post-Condizioni	context ManagerOrderSearchMailControl:: doGet(response, request) **post:** result = response
Invarianti	N/A

Nome Classe	ManagerOrderViewControl
Sottosistema	Control
Variabili di istanza	+ om: OrderManager
Descrizione	Da questo Control vengono gestite le operazioni relative alla visualizzazione di un ordine specifico da parte di un manager.
Signature metodi	+ doGet(request: HttpServletRequest, response: HttpServletResponse)
Pre-Condizioni	• session ->includes(manager) • request ->includes(idOrder)
Post-Condizioni	context ManagerOrderViewControl:: doGet(response, request) **post:** result = response
Invarianti	N/A