UNIVERSITÀ DEGLI STUDI
DI SALERNO

Corso di Laurea in Informatica

Ingegneria del Software

Object Design Document - BeHub

Anno Accademico: 2022/23

Docente:

Prof. Andrea De Lucia

Studenti:

Mirko Danilo Pacelli	0512112321
Carlo Perilli	0512112306
Eljon Hida	0512109978

Data	Versione	Descrizione	Autori
10/12/2022	0.1	Creazione document e naming conventions	Tutto il team
14/12/2022	0.2	Design pattern e packages	Mirko Danilo Pacelli
18/12/2022	0.3	Class Interfaces	Carlo Perilli
22/12/2022	0.4	Class Diagram	Eljon Hida
11/02/2023	2.0	Revisione finale e ultime correzioni	Tutto il team

Sommario

1. Introduzione

Il seguente documento si pone come obbiettivo quello di descrivere le linee guida in fase di implementazione del sistema BeHub. In questo documento verranno descritti i trade-off generali e le convenzioni per la nomenclatura ed implementazione di classi ed interfacce.

Object-Design Trade-Offs

Prestazioni vs Costi:

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

Affidabilità vs Usability:

Un software altamente affidabile potrebbe avere un'interfaccia utente meno intuitiva, mentre un software intuitivo presenta un'affidabilità leggermente inferiore.

Nel nostro abbiamo scelto di garante al nostro software un livello di medio di usability e affidabilità.

2. Linee Guida

Naming Conventions

I nomi dovranno rispettare le seguenti caratteristiche:

- Descrittivi
- Di uso comune
- Non abbreviati

- Utilizzando caratteri alfanumerici e simboli
- Cercare di mantenere una lunghezza medio-corta

2.1 Variabili

I nomi delle variabili dovranno iniziare con lettera minuscola e le parole successive dovranno avere la prima lettera maiuscola.

La dichiarazione delle variabili viene fatta di norma alla fine facendo in modo che vi sia una sola variabile per riga, rendendo così più leggibile e chiaro il tutto.

2.2 Metodi

Per i nomi dei metodi vige la stessa regola delle variabili.

Nella maggior parte dei casi il nome del metodo sarà un verbo che rappresenta la sua funzione e sarà seguito dal nome dell'oggetto su cui opera.

Per quanto riguarda i vari metodi di accesso e modifica di una variabile dovranno sempre avere la seguente forma: getPassword(), setPassword();

2.3 Commenti

I commenti, se utilizzati, dovranno essere:

- Per variabili: scritti sulla stessa riga
- Per metodi: scritti appena prima quest'ultimo è stato dichiarato

2.4 Classi ed Interfacce

I nomi delle classi e delle interfacce dovranno seguire la "CamelCase"; quindi, ogni parola inizia con la lettera maiuscola.

I nomi delle classi dovranno essere, come per i metodi, evocativi.

La dichiarazione di una classe è caratterizzata da:

1. Dichiarazione della classe public
2. Costruttore
3. Possibile commento e dichiarazione metodi e variabili
4. Dichiarazioni di costanti con possibile commento
5. Dichiarazioni di variabili di classe con possibile commento

6. Dichiarazioni di variabili d’istanza con possibile commento

2.5 Pagine

I nomi delle pagine dovranno essere scritti in minuscolo e vi sarà un “– “ a dividere le parole.

2.6 Acronimi ed abbreviazioni

Acronimi:

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document

Abbreviazioni:

- DB: Database

2.7 Riferimenti

- RAD BeHub;
- Problem Statement BeHub;
- SDD BeHub;
- Libro Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition Autori: -- Bernd Bruegge & Allen H. Dutoit;

3. Design Pattern

3.1 Data Access Object (DAO)

È un pattern architetturale che consente di separare i servizi della logica di business dalle operazioni per la gestione dei dati persistenti. La funzionalità di questo pattern è nascondere dall’applicazione tutte le complessità coinvolte nell’esecuzione dell’operazioni che interagiscono con la sorgente dei dati in modo da permettere ad entrambi i livelli di evolversi separatamente senza sapere nulla dell’altro.

Il DAO implementa il meccanismo di accesso richiesto per lavorare con la sorgente dei dati. I moduli della logica di business utilizzano l’interfaccia esposta dal DAO, tale interfaccia definisce le operazioni standard da eseguire. La classe DAO concreta che implementerà l’interfaccia è responsabile dell’archiviazione e restituzione dei dati. Per archiviare i dati recuperati dalla classe DAO si utilizzano gli oggetti Bean che contengono i metodi get/set per memorizzare i dati recuperati.

Il design pattern sarà usato per la gestione della persistenza dei dati. L'oggetto del controller comunicherà con l'interfaccia DAO che verrà implementata dalle classi dao specifiche.

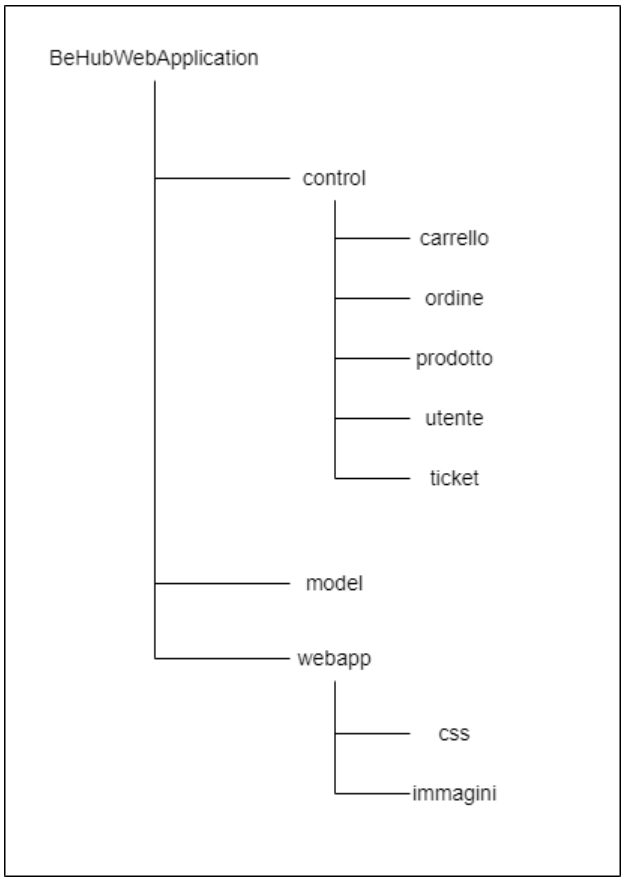
3.2 Architettura Three-Tier

Lo schema che abbiamo identificato è esattamente quello proposto dall'architettura three-tier:

- *Model:* contiene i metodi di accesso ai dati.
- *View:* si occupa di visualizzare i dati all'utente e gestisce l'interazione fra quest'ultimo e l'infrastruttura sottostante(webapp).
- *Controller:* riceve i comandi dell'utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.

La suddivisione dei package in servlets, model e webcontent è per l'appunto motivata dall'utilizzo dell'architettura MVC, la quale permette di separare la logica di business (servlets) dalla presentazione (webapp) e dalla gestione dei dati (model). Il package model è a sua volta suddiviso in bean e dao. Le servlet nel package servlets si occuperanno del

4. Packages:



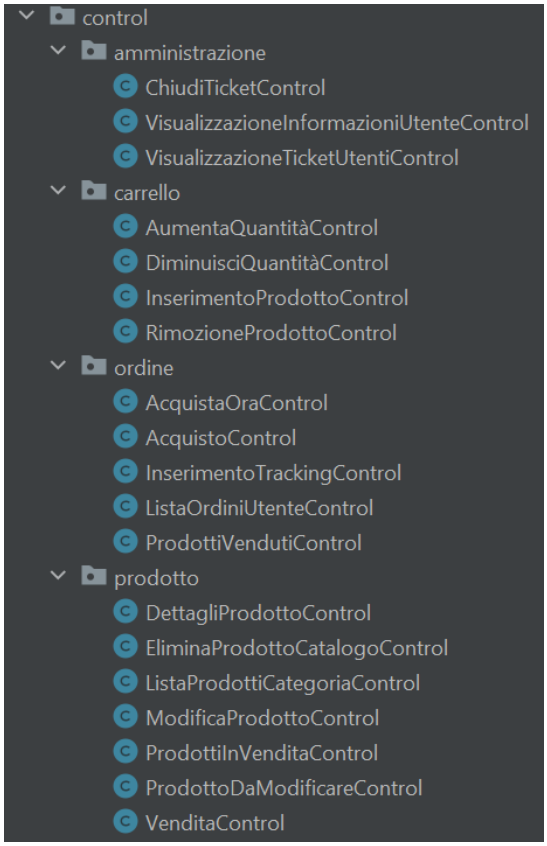
I tre pacchetti principali sono control, model e webapp.

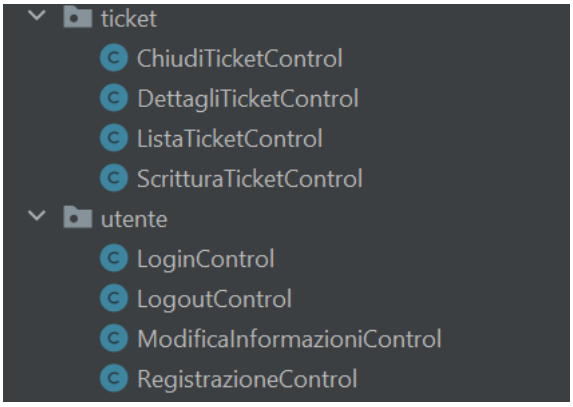
Il pacchetto control è suddiviso a sua volta in altri sotto pacchetti che presentato al proprio interno le servlet.

Nel pacchetto model abbiamo, non solo i model, ma sono presenti anche tutti gli oggetti entity che definiscono gli oggetti di dominio e i quali saranno utilizzati come bean dalle servlet presenti nel package control.

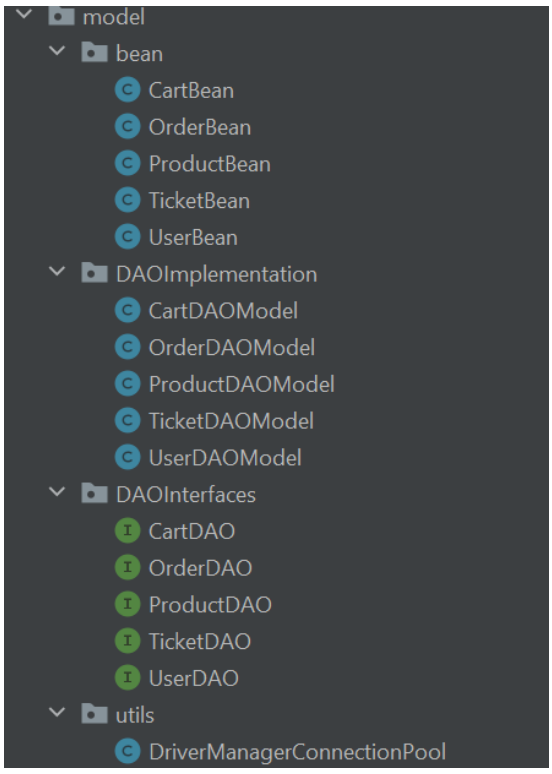
Infine, il pacchetto webapp oltre ad avere tutti i file jsp, sorgenti HTML e js presenta altri due pacchetti: uno contenente tutti i file css e l'altro tutte le immagini usate nel sito.

Control Package

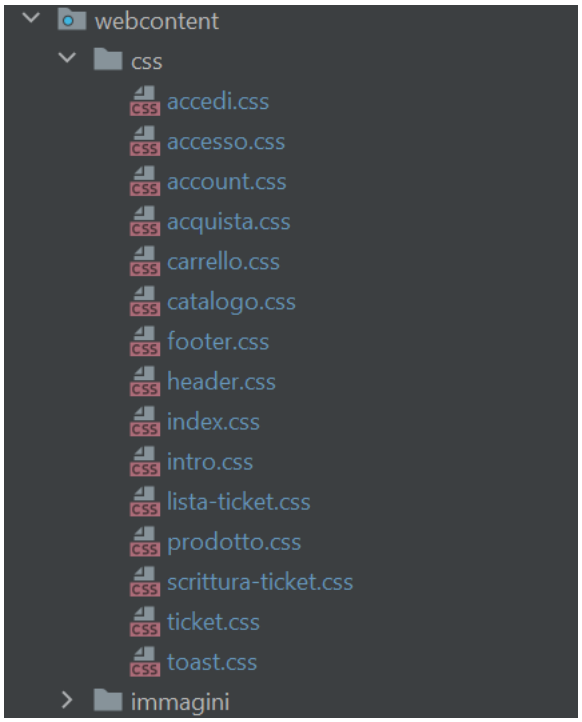


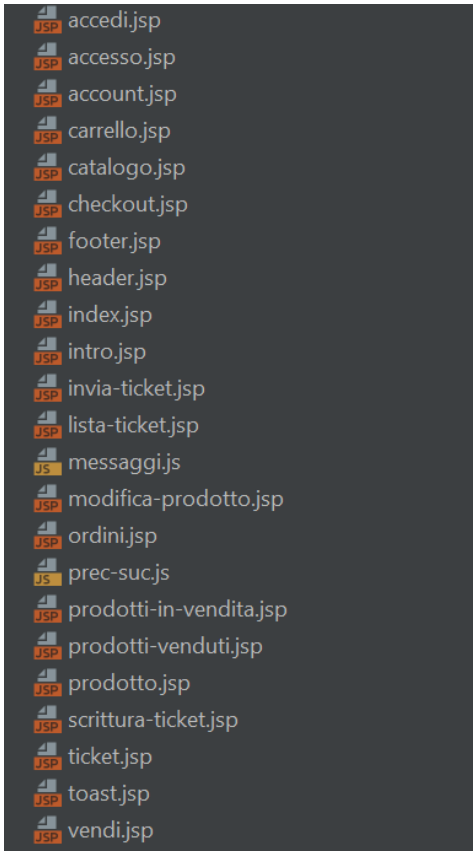


Model Package



WebApp Package





5. Class Interfaces

CartBean

Nome Classe	CartBean
Descrizione	Permette di gestire le informazioni riguardanti il carrello
Metodi	+setCarrello(ProductBean newProdotto): void +retrieveByKey(int codiceProdotto): ProductBean +addQuantity(int codiceProdotto): void +decreaseQuantity(int codiceProdotto): void +getCarrello(): Collection<ProductBean> +removeItem(int codiceProdotto): void +isEmpty(): boolean +removeAllItems(): void +setLista(Collection<ProductBean> lista): void

Nome Metodo	+setCarrello(ProductBean newProdotto)
Descrizione	Permette di inserire prodotti nel carrello; ne aumenta la quantità se il prodotto inserito era già presente
Context	CartBean::setCarrello(newProdotto)
Pre-condizione	newProdotto<>null
Post-condizione	carrello->include(newProdotto)

Nome Metodo	+retrieveByKey(int codiceProdotto)
Descrizione	Permette di ritrovare un prodotto grazie al suo codice
Context	CartBean::retrieveByKey(codiceProdotto)
Pre-condizione	codiceProdotto<>null and carrello->exist(prodotto|prodotto.codiceProdotto==codiceProdotto)
Post-condizione	Prodotto<>null

Nome Metodo	+addQuantity(int codiceProdotto)
Descrizione	Permette di incrementare la quantità di un prodotto già presente nel carrello
Context	CartBean::addQuantity(codiceProdotto)
Pre-condizione	codiceProdotto<>null and carrello->exist(prodotto|prodotto.codiceProdotto==codiceProdotto)
Post-condizione	prodotto.getQuantity() = prodotto.getQuantity()+1

Nome Metodo	+decreaseQuantity(int codiceProdotto)
Descrizione	Permette di decrementare la quantità di un prodotto già presente nel carrello
Context	CartBean::decreaseQuantity(codiceProdotto)
Pre-condizione	codiceProdotto<>null and carrello->exist(prodotto|prodotto.codiceProdotto==codiceProdotto)
Post-condizione	prodotto.getQuantity() = prodotto.getQuantity()-1

Nome Metodo	+removeItem(int codiceProdotto)
Descrizione	Permette di rimuovere un prodotto dal carrello
Context	CartBean::removeItem(codiceProdotto)
Pre-condizione	codiceProdotto<>null and carrello->exist(prodotto|prodotto.codiceProdotto==codiceProdotto)
Post-condizione	not carrello->include(prodotto|prodotto.codiceProdotto==codiceProdotto)

Nome Metodo	+isEmpty()
Descrizione	Restituisce true se il carrello è vuoto, false altrimenti
Context	CartBean::isEmpty()
Pre-condizione	n/a
Post-condizione	n/a

Nome Metodo	+removeAllItems()
Descrizione	Permette di eliminare tutti i prodotti dal carrello
Context	CartBean:: removeAllItems()
Pre-condizione	n/a
Post-condizione	carrello->isEmpty()

CartDAOModel

Nome Classe	CartModel
Descrizione	
Metodi	+updateCarrello(ProductBean bean, CartBean cart): CartBean +aggiungiAlCarrello(CartBean carrello, int codiceProdotto): CartBean +aggiungiAlCarrello(CartBean carrello, int codiceProdotto, int quantità): CartBean

Nome Metodo	+updateCarrello(ProductBean bean, CartBean cart)
Descrizione	Permette di modificare un prodotto già presente in un carrello
Context	CartBean:: updateCarrello(bean, cart)
Pre-condizione	cart<>null and bean<>null
Post-condizione	cart<>null

Nome Metodo	+aggiungiAlCarrello(CartBean carrello, int codiceProdotto)
Descrizione	Permette di aggiungere un prodotto al carrello attraverso il suo codice
Context	CartBean:: aggiungiAlCarrello(carrello, codiceProdotto)
Pre-condizione	carrello<>null and codiceProdotto<>null
Post-condizione	carrello->include(prodotto| prodotto.getCodice()==codiceProdotto)

Nome Metodo	+aggiungiAlCarrello(CartBean carrello, int codiceProdotto, int quantità)
Descrizione	Permette di aggiungere un prodotto al carrello attraverso il suo codice
Context	CartBean:: aggiungiAlCarrello(carrello, codiceProdotto, quantità)
Pre-condizione	carrello<>null and codiceProdotto<>null and quantità>0
Post-condizione	carrello->include(prodotto| prodotto.getCodice()==codiceProdotto and prodotto.getQuantity()==quantità)

OrderBean

Nome Classe	OrderBean
Descrizione	Permette di gestire le informazioni riguardanti l'ordine
Metodi	+setProdotto(ProductBean newProdotto): void +setData(Date newData): void +setCodice(int newCodice): void +setEmail(String newEmail): void +setTracking(String codiceTracking): void +setStato(int valoreStato): void +getProdotto(): ProductBean +getData(): Date

	+getCodice(): int +getEmail(): String +getTracking(): String +getStato(): int

OrderDAOModel

Nome Classe	OrderModel
Descrizione	Fornisce i metodi per gestire gli ordini
Metodi	+doOrder(CartBean cart, UserBean user): boolean +doOrder(ProductBean prodotto, UserBean user): boolean +getOrdini(String email): Collection<OrderBean> +getProdottiVenduti(String email): Collection<OrderBean> +inserisciTracking(String tracking, int codiceOrdine): boolean +parseStato(String stato): int +parseStato(int valoreStato): String

Nome Metodo	**+doOrder(CartBean cart, UserBean user)**
Descrizione	Permette di creare l'ordine dei vari prodotti nel carrello
Context	CartBean:: doOrder(cart, user)
Pre-condizione	carrello<>null and user<>null
Post-condizione	n/a

Nome Metodo	**+doOrder(ProductBean prodotto, UserBean user)**
Descrizione	Permette di creare l'ordine di un solo prodotto nel carrello
Context	CartBean:: doOrder(prodotto, user)
Pre-condizione	prodotto<>null and user<>null
Post-condizione	n/a

Nome Metodo	**+getOrdini(String email)**
Descrizione	Permette ti ottenere la lista degli ordini di un utente attraverso la sua email
Context	CartBean:: getOrdini(email)
Pre-condizione	email<>null
Post-condizione	listaOrdini<>null

Nome Metodo	+getProdottiVenduti(String email)
Descrizione	Permette ti ottenere la lista dei prodotti venduti di un utente attraverso la sua email
Context	CartBean:: getProdottiVenduti(email)
Pre-condizione	email<>null
Post-condizione	listaOrdini<>null

Nome Metodo	+inserisciTracking(String tracking, int codiceOrdine)
Descrizione	Permette di inserire il codice di tracking nell'ordine
Context	CartBean:: inserisciTracking(tracking, codiceOrdine)
Pre-condizione	tracking<>null and codiceOrdine<>null
Post-condizione	n/a

Nome Metodo	+parseStato(String stato)
Descrizione	Converte una stringa stato nel suo valore numerico corretto
Context	CartBean:: parseStato(stato)
Pre-condizione	stato<>null
Post-condizione	n/a

Nome Metodo	+parseStato(int valoreStato)
Descrizione	Converte un valore numerico stato nella sua stringa corretta
Context	CartBean:: parseStato(valoreStato)
Pre-condizione	valoreStato<>null
Post-condizione	n/a

ProductBean

Nome Classe	ProductBean
Descrizione	Permette di gestire le informazioni riguardanti il prodotto
Metodi	+setCodice(int newCodice): void +setNome(String newNome): void +setDescrizione(String newDesc): void +setPrezzo(double newPrezzo): void +setSpedizione(double newSpeseSped): void +setEmail(String newEmail): void +setCategoria(int valoreCategoria): void +setData(Date newData): void +setImmagine(String newImmagine): void +setQuantity(int newQuantity): void +setCondizione(int valoreCondizione):void +setMaxQuantity(int newMaxQuantity):void +getCodice(): int +getNome(): String +getDescrizione(): String +getPrezzo(): double +getSpedizione(): double +getEmail(): String +getCategoria(): int +getData(): Date +getImmagine(): String +getQuantity(): int +getCondizione(): int +getMaxQuantity(): int +addQuantity(): void +decreaseQuantity(): void

Nome Metodo	**+decreaseQuantity()**
Descrizione	Permette di decrementare di uno la quantità dei prodotti
Context	CartBean:: decreaseQuantity()
Pre-condizione	n/a
Post-condizione	quantity=quantity-1

Nome Metodo	+ addQuantity()
Descrizione	Permette di incrementare di uno la quantità dei prodotti
Context	CartBean:: addQuantity()
Pre-condizione	n/a
Post-condizione	quantità=quantità+1

ProductDAOModel

Nome Classe	ProductModel
Descrizione	
Metodi	+doSave(ProductBean product): void +doRetrieveByKey(int code): ProductBean +doRetrieveAll(String where): Collection<ProductBean> +deleteProduct(int codiceProdotto, Collection<ProductBean> lista): Collection<ProductBean> +updateProduct(ProductBean bean): void +getProdottiInVendita(String email): Collection<ProductBean> +diminuisciQuantità(int codiceProdotto, int quantitàAcquistata): boolean +controllaCondizione(int valoreCondizione): String +controllaCategoria(int valoreCategoria): String +parseCategoria(String categoria): int +parseCondizione(String condizione): int

Nome Metodo	+ doSave(ProductBean product)
Descrizione	Permette di inserire nel database un prodotto passando in input un ProductBean
Context	CartBean:: doSave(product)
Pre-condizione	Product<>null
Post-condizione	n/a

Nome Metodo	+ doRetrieveByKey(int code)
Descrizione	Permette di ottenere un prodotto dal database attraverso il suo codice
Context	CartBean:: doRetrieveByKey(code)
Pre-condizione	code<>null
Post-condizione	beanProdotto.getCodice=code

Nome Metodo	+ doRetrieveAll(String where)
Descrizione	Permette di ottenere una collezione di oggetti di tipo ProductBean passandogli come parametro il nome della categoria
Context	CartBean:: doRetrieveAll(where)
Pre-condizione	where<>null and (where=Libri or where=Calzature or where=Elettronica or where=Abbigliamento or where=Giocattoli)
Post-condizione	products<>null

Nome Metodo	+ deleteProduct(int codiceProdotto, Collection<ProductBean> lista)
Descrizione	Permette di eliminare un prodotto da una collezione di ProductBean attraverso il suo codice
Context	CartBean:: deleteProduct(codiceProdotto, lista)
Pre-condizione	codiceProdotto<>null and lista<>null
Post-condizione	not lista->exist(prodotto| prodotto.getCodice()=codiceProdotto)

Nome Metodo	+ updateProduct(ProductBean bean)
Descrizione	Permette di aggiornare un prodotto
Context	CartBean:: updateProduct(ProductBean bean)
Pre-condizione	bean<>null
Post-condizione	n/a

Nome Metodo	+ diminuisciQuantità(int codiceProdotto, int quantitàAcquistata)
Descrizione	Permette di diminuire la quantità di un prodotto nel database dopo l'acquisto
Context	CartBean:: diminuisciQuantità(codiceProdotto, quantitàAcquistata)
Pre-condizione	codiceProdotto<>null and quantitàAcquistata<>null
Post-condizione	productBean.getCodice()=codiceProdotto and productBean.getQuantity()=productBean.getQuantity() - quantitàAcquistata

Nome Metodo	+ controllaCondizione(int valoreCondizione)
Descrizione	Trasforma la variabile intera valoreCondizione nella stringa corretta
Context	CartBean:: controllaCondizione(valoreCondizione)
Pre-condizione	valoreCondizione<>null
Post-condizione	n/a

Nome Metodo	+ controllaCategoria(int valoreCategoria)
Descrizione	Trasforma la variabile intera valoreCategoria nella stringa corretta
Context	CartBean:: controllaCategoria(valoreCategoria)
Pre-condizione	valoreCategoria<>null
Post-condizione	n/a

Nome Metodo	+parseCategoria(String categoria)
Descrizione	Trasforma la stringa categoria nella variabile intera corretta
Context	CartBean::parseCategoria(categoria)
Pre-condizione	categoria<>null
Post-condizione	n/a

Nome Metodo	+parseCondizione(String condizione)
Descrizione	Trasforma la stringa condizione nella variabile intera corretta
Context	CartBean:: parseCondizione(String condizione)
Pre-condizione	condizione<>null
Post-condizione	n/a

TicketBean

Nome Classe	TicketBean
Descrizione	Permette di gestire le informazioni riguardanti il ticket
Metodi	+setCodice(int newCodice) +setTesto(String newTesto) +setOggetto(String newOggetto) +setData(Date newDate) +setEmailUtente(String newEmailUtente) +setStato(int codiceStato) +getCodice(): int +getTesto(): String +getOggetto(): String +getData(): Date +getEmailUtente(): String +getStato(): int

TicketDAOModel

Nome Classe	TicketModel
Descrizione	
Metodi	+aggiungiTicket(TicketBean ticket): boolean +getListaTicket(String emailUtente): Collection<TicketBean> +chiudiTicket(TicketBean ticket): boolean +getListaTicketAperti(String emailUtente): Collection<TicketBean> +retrieveByKey(int codiceTicket): TicketBean -controllaStato(int codiceStato): String -parseStato(String stato): int

Nome Metodo	**+aggiungiTicket(TicketBean ticket)**
Descrizione	Permette di aggiungere un ticket nel database
Context	CartBean:: aggiungiTicket(ticket)
Pre-condizione	ticket<>null
Post-condizione	n/a

Nome Metodo	+getListaTicket(String emailUtente)
Descrizione	Permette di ottenere una collezione di oggetti di tipo TicketBean di un certo utente
Context	CartBean:: getListaTicket(String emailUtente)
Pre-condizione	emailUtente<>null
Post-condizione	listaTicket<>null

Nome Metodo	+chiudiTicket(TicketBean ticket)
Descrizione	Permette di modificare lo stato di un ticket da aperto a chiuso
Context	CartBean:: chiudiTicket(ticket)
Pre-condizione	ticket<>null
Post-condizione	n/a

Nome Metodo	+retrieveByKey(int codiceTicket)
Descrizione	Permette di ottenere un ticket attraverso il suo codice
Context	CartBean:: retrieveByKey(codiceTicket)
Pre-condizione	codiceTicket<>null
Post-condizione	ticket<>null

Nome Metodo	+controllaStato(int codiceStato)
Descrizione	Trasforma la variabile intera codiceStato nella stringa corretta
Context	CartBean:: controllaStato(codiceStato)
Pre-condizione	codiceStato<>null
Post-condizione	n/a

Nome Metodo	+parseStato(String stato)
Descrizione	Trasforma la stringa stato nella variabile intera corretta
Context	CartBean:: parseStato(stato)
Pre-condizione	stato<>null
Post-condizione	n/a

UserBean

Nome Classe	UserBean
Descrizione	Permette di gestire le informazioni riguardanti l'utente
Metodi	+setEmail(String newEmail) +setNome(String newNome) +setCognome(String newCognome) +setIndirizzo(String newIndirizzo) +setTelefono(String newTelefono)) +setNumero(String newNumero) +setIntestatario(String newIntestatario) +setRole(String newRole) +setData(Date newData) +getEmail(): String +getNome(): String +getCognome(): String +getIndirizzo(): String +getTelefono(): String +getNumero(): String +getIntestatario(): String +getRole(): String +getData(): Date

UserDAOModel

Nome Classe	UserModel
Descrizione	
Metodi	+update(UserBean bean, String emailOld): boolean +insert(UserBean user, String psw): boolean +login(String email, String password): UserBean

Nome Metodo	**+update(UserBean bean, String emailOld)**
Descrizione	Permette di aggiornare le informazioni di un utente
Context	CartBean:: update(bean, emailOld)
Pre-condizione	bean<>null and emailOld<>null
Post-condizione	n/a

Nome Metodo	+insert(UserBean user, String psw)
Descrizione	Permette di registrare un nuovo utente inserendolo nel database
Context	CartBean:: insert(user, psw)
Pre-condizione	user<>null and psw<>null
Post-condizione	n/a

Nome Metodo	+login(String email, String password)
Descrizione	Permette ad un utente già registrato di accedere
Context	CartBean:: login(email, password)
Pre-condizione	email<>null and password<>null
Post-condizione	user<>null

6. Class Diagram