Object Design Document

INDITEX ReSkin

Riferimento	INDITEXReSkin_ODD_2.0
Versione	2.0
Data	27/12/2023
Destinatario	C. Gravino
Presentato da	A. Ruggiero, G. Gurrali

Team composition

Ruolo	Nome	Acronimo	Contatti
Top Management	Carmine Gravino	CG	gravino@unisa.it
Team Member	Andrea Ruggiero	AR	a.ruggiero150@studenti.unisa.it
Team Member	Giovanni Gurrali	GG	g.gurrali@studenti.unisa.it

Sommario

Revision history

Data	Versione	Descrizione	Autori
28/12/2023	0.1	Stesura documento	[TEAM MEMBER]
29/12/2023	0.2	Stesura scopo del sistema, Object Design Goal, Trade-Off	GG
03/01/2024	0.3	Completamento dell'Introduzione	[TEAM MEMBER]
10/01/2024	0.4	Package	GG
13/01/2024	0.5	Class Interface	[TEAM MEMBER]
20/01/2024	0.6	Class Diagram	AR
25/01/2024	0.7	Glossario	GG
27/01/2024	0.8	Elementi di riuso	AR
30/01/2024	0.9	Revisione package	GG
3/02/2024	0.10	Revisione Class Diagram	AR
5/02/2024	1.0	Prima revisione	[TEAM MEMBER]
7/02/2024	1.1	Revisione Class Interface	[TEAM MEMBER]
11/02/2024	1.2	Revisione Elementi di riuso	AR
14/02/2024	2.0	Revisione finale	[TEAM MEMBER]

1. Introduzione

1.1. Scopo del sistema

Nel corso dell'ultimo decennio, nonostante gli sforzi considerevoli compiuti da agenzie e governi per promuovere una vera e propria transizione ecologica attraverso ampi programmi di sviluppo, diversi settori continuano a lottare per adottare le pratiche comuni necessarie per lo smaltimento adeguato dei rifiuti. Uno dei settori più inquinanti che ha scalato posizioni, raggiungendo il secondo posto nella graduatoria tra quelli più dannosi per l'ambiente, è senz'altro quello della moda, in particolare il segmento noto come fast fashion, che si concentra sulla produzione in massa di capi d'abbigliamento. Diversi report hanno evidenziato la presenza di metano e petrolio, tra altre sostanze chimiche, in materiali ampiamente utilizzati dalle grandi aziende, come cotone, seta e fibre sintetiche. Queste sostanze, se non smaltite correttamente, possono contaminare le acque sotterranee o persistere nell'ambiente senza degradarsi, causando un inquinamento irreversibile del nostro pianeta. Inoltre, oggigiorno, le aziende producono all'incirca 92 milioni di tonnellate di rifiuti tessili ogni anno, per ogni chilo di cotone consumano più di 20.000 litri d'acqua e quasi il 10% delle microplastiche disperse nell'oceano provengo dai tessuti non adeguatamente smaltiti. Problemi singoli ma che nell'insieme macchiano tutti i buoni propositi che da anni cerchiamo di portare avanti; propositi quali la sostenibilità ed il riciclo consapevole. Risolvere tutto e subito è irreale ma è bene iniziare a pensare a soluzioni alternative che possano modificare questa tendenza negativa. L'azienda INDITEX propone quindi un sistema innovativo chiamato INDITEX ReSkin. L'obiettivo principale di questo sistema è creare un canale di comunicazione che consenta a piccoli artigiani e innovative start-up italiane di acquistare a prezzi vantaggiosi il materiale di produzione in eccesso. L'azienda potrà dunque mettere in vendita sottoprodotti (ovvero uno scarto di lavorazione derivante da processi industriali, il quale viene riutilizzato in un altro processo produttivo come materia prima non vergine) derivanti dalla produzione di capi per i suoi principali brand di Fast Fashion. Questo approccio permette all'azienda di recuperare una parte delle spese sostenute per l'acquisto dei materiali, riducendo al contempo gli sprechi e promuovendo il riutilizzo delle stoffe e di altri materiali altrimenti destinati allo spreco. In particolare, l'azienda ha deciso di puntare sul mercato italico proprio per la grande tradizione che l'Italia ha e ancora oggi mette in campo quando si parla di artigianato e innovazione nell'ambito della moda, puntando quindi a valorizzare questa eredità culturale ma al tempo stesso valorizzando i giovani talenti del territorio, che si mettono in gioco creando start-up innovative e all'insegna del green.

1.2. Object Design Goals

Priorità	ID	Descrizione	Categoria	Origine	Trade-off
1	ODG_1 Persistenza dei dati	Il sistema deve poter immagazzinare ed interrogare i dati su un database SQL	Dependability	DG_1	Tempi di risposta VS Sicurezza
2	ODG_2 Riservatezza	Il sistema garantisce la riservatezza dei dati sensibili personali, criptandoli in fase di acquisizione	Dependability	DG_2	Tempi di risposta VS Sicurezza
3	ODG_3 Evoluzione	Il sistema potrà essere facilmente ampliato con nuove funzionalità	Maintenance	DG_3	Tempo di rilascio VS Qualità
4	ODG_4 User Friendly	Il sistema deve essere di facile utilizzo per tutti gli utenti. Ogni utente deve poter navigare facilmente al sito, compiendo ogni azione senza ambiguità. Ogni aspetto grafico del sito deve favorire la user experience	End-user	DG_5	Tempo di rilascio VS Qualità

1.3. Object Design Trade-offs

Trade-off	Descrizione
Tempo di risposta VS Sicurezza	Il sistema dovendo gestire dati personali ed effettuare operazioni su database si è deciso di dilatare i tempi di risposta al fine di garantire un livello di sicurezza adeguato. I tempi di risposta per accedere ed interagire con il database rimarranno comunque al di sotto dei tre secondi
Tempo di rilascio VS Qualità	Al fine di rispettare la deadline del progetto, i tempi di consegna avranno priorità rispetto alla qualità del software; pur garantendo che la qualità rimarrà al di sopra degli standard minimi pattuiti.

1.4. Linee guida per la Documentazione delle Interfacce

Nel documentare le nostre interfacce andremo ad utilizzare le seguenti convenzioni al fine di evitare di creare delle discrepanze tra ciò che è stato prodotto a livello di codice e come viene documentato nel seguente documento:

- **File classi di tipo controller:**
  - Per i nomi dei controller è stata utilizzata una formattazione “camelCase”
  - Ogni nome specifica la funzione della classe
  - Ogni nome termina con la dicitura “Servlet”
  - Ogni nome è definito in lingua inglese
- **File classi di tipo DAOStorage:**
  - Ogni nome inizia per lettera maiuscola
  - I metodi all’interno del DAO sono tutti statici
  - Ogni nome fa riferimento al sottosistema su cui opera
  - Ogni nome termina con la dicitura “DAO”
- **File classi di tipo EntityStorage:**
  - Ogni nome inizia con la lettera maiuscola
  - Ogni nome di classe fa riferimento ad un oggetto persistente
  - Ogni nome è definito al singolare
- **File interfacce utente:**
  - Per i nomi delle interfacce è stata utilizzata una formattazione “camelCase”
  - Ogni nome inizia con la dicitura “page”
- **Metodi e campi privati di una classe**
  - I campi privati relativi ad una classe sono indicati tramite sostantivi
  - Ogni campo inizia con una lettera minuscola
  - Ogni campo è definito al singolare
  - Il nome di ogni metodo descrive l’operazione che verrà eseguita
  - Il nome di ogni metodo è indicato tramite verbo
  - Per i nomi dei metodi è stata utilizzata una formattazione “camelCase”
- **Query database:**
  - Ogni query definita nei DAOStorage segue la formattazione dei metodi sopracitati
  - Il nome di ogni query definisce esplicitamente l’operazione che verrà eseguita
  - Il nome di ogni query è definito in inglese
  - Lo status di errore è definito tramite valore di ritorno e non tramite eccezione
- **Immagini:**
  - Ogni immagine importata è di tipo JPEG
- **Dati immagazzinati in sessione**:
  - Intero che rappresenta il tipo di customer connesso
    - Guest: 0
    - Cliente: 1
    - Admin: 2
  - Dati customer

Di seguito riportiamo la documentazione ufficiale dei linguaggi utilizzati:

Linguaggi di programmazione:

- Java: https://dev.java/learn/
- HTML: https://developer.mozilla.org/en-US/docs/Web/HTML
- CSS: https://developer.mozilla.org/en-US/docs/Web/CSS
- Bootstrap: https://getbootstrap.com/docs/5.3/getting-started/introduction/
- MySQL: https://dev.mysql.com/doc/
- JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Ajax: https://api.jquery.com/category/ajax/
- JQuery: https://api.jquery.com/

1.5. Definizioni e acronimi

Definizioni:

- **Controller:** Un componente che gestisce l'interazione tra l'utente e il sistema, spesso traducendo l'input dell'utente in azioni da eseguire sulle entità del sistema e mostrando i risultati all'utente
- **DAOStorage:** Directory contenente tutti i file di tipo DAO
- **EntityStorage:** Directory contenente tutti i file di tipo Entity
- **Query:** Una richiesta per ottenere dati da un database, seguendo un certo criterio
- **camelCase:** Una convenzione di denominazione in cui le parole composte vengono unite e ogni parola successiva dopo la prima inizia con una lettera maiuscola, senza spazi o segni di punteggiatura
- **Maven:** Uno strumento di gestione e automazione di progetti software in Java che facilita la gestione delle dipendenze, la compilazione e la distribuzione dei progetti

Acronimi:

- **DAO**:(Data Access Object) è un pattern architetturale che separa la logica di accesso ai dati dalla logica di business in un'applicazione software. Fornisce un'interfaccia astratta per accedere ai dati, indipendentemente dal tipo di archivio dati sottostante, promuovendo la modularità e la manutenibilità del codice
-

1.6. Riferimenti

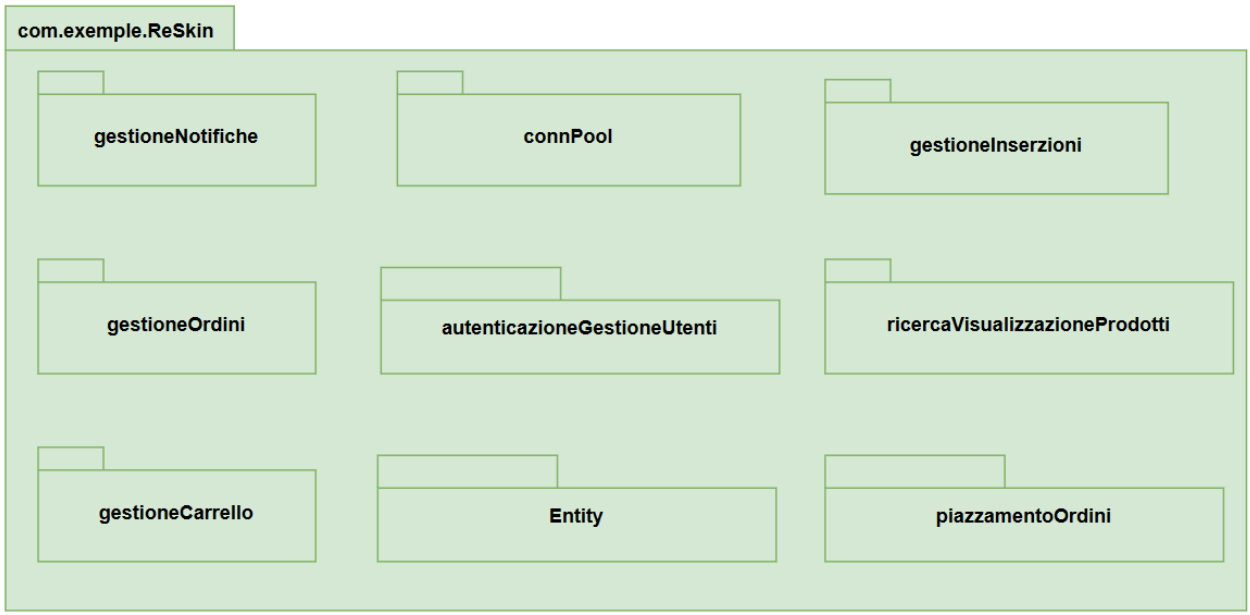
- **Libro:** “Object Oriented Software Engineering using UML, Patterns and Java”. Edizione: 3rd Edition Anno: 2014. Autori: Bernd Bruegge, Allen H. Dutoit
- **INDITEXReskin_SOW**
- **INDITEXReskin_RAD**
- **INDITEXReskin_SDD**
- **INDITEX**

2. Packages

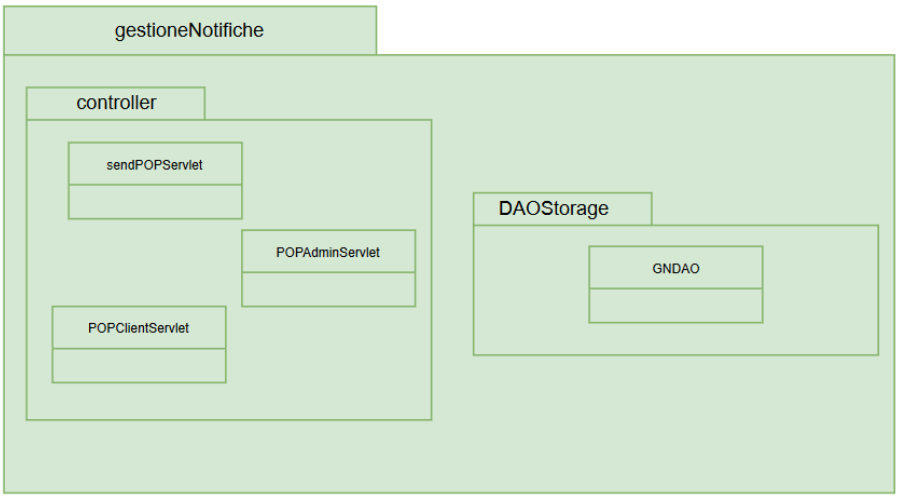
La suddivisione in package che proponiamo qui di seguito è motivata dalle scelte architetturali prese durante la fase di System Design integrando il tutto con la struttura di directory definita da Maven. Il sistema sarà quindi così strutturato:

- **.idea**
- **.mvn**: contiene tutti i file di configurazione per Maven
- **src**: contiene tutti i file sorgente
    - **main**
        - **java**: contiene i package e le classi Java relative al backend del sito
        - **webapp**
            - **css**
            - **resources**
            - **WEB-INF**
                - **interface**: contiene tutta l'interfaccia grafica frontend
    - **test**: contiene tutto il necessario per il testing di unità
- **target**: contiene tutti i file prodotti dal sistema di build di Maven.

Questo package contiene tutti i sottosistemi che costituiscono ReSkin:



Gestione delle Notifiche:

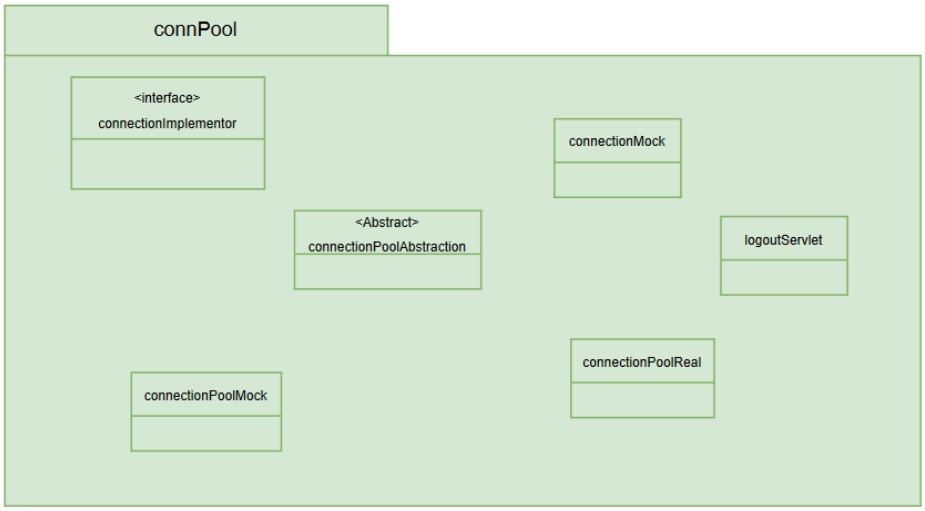


Package contente due layer:

- controller: gestisce la logica applicativa del sottosistema di Gestione delle Notifiche
- DAOStorage: gestisce la logica di interazione con il database per tutte le funzioni di gestione delle notifiche

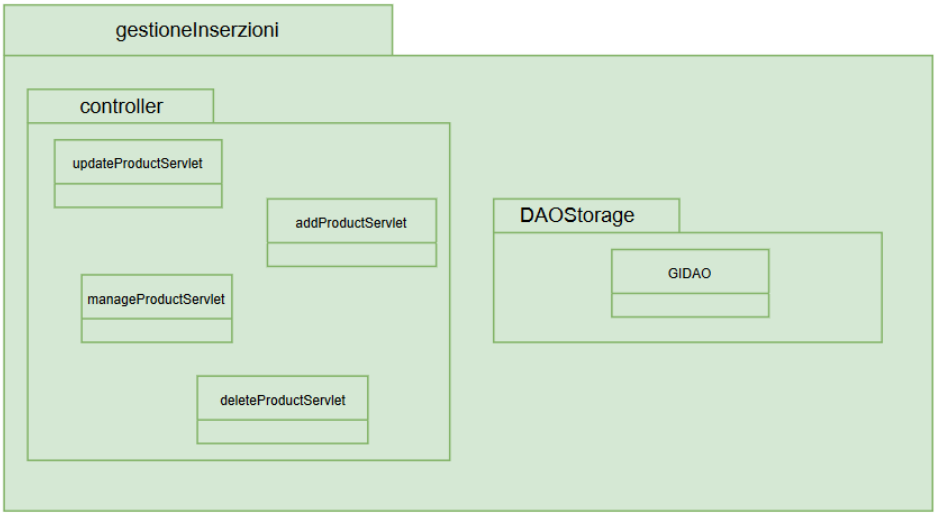
Questo package gestisce le notifiche all'interno del sistema. Gli utenti admin possono inviare notifiche POP (anche in modalità broadcast) e visualizzare le notifiche generali inviate. Gli utenti clienti possono visualizzare le notifiche ricevute in un'apposita sezione.

connPool:



Questo package contiene la logica di gestione del Design Pattern implementato

Gestione delle Inserzioni:

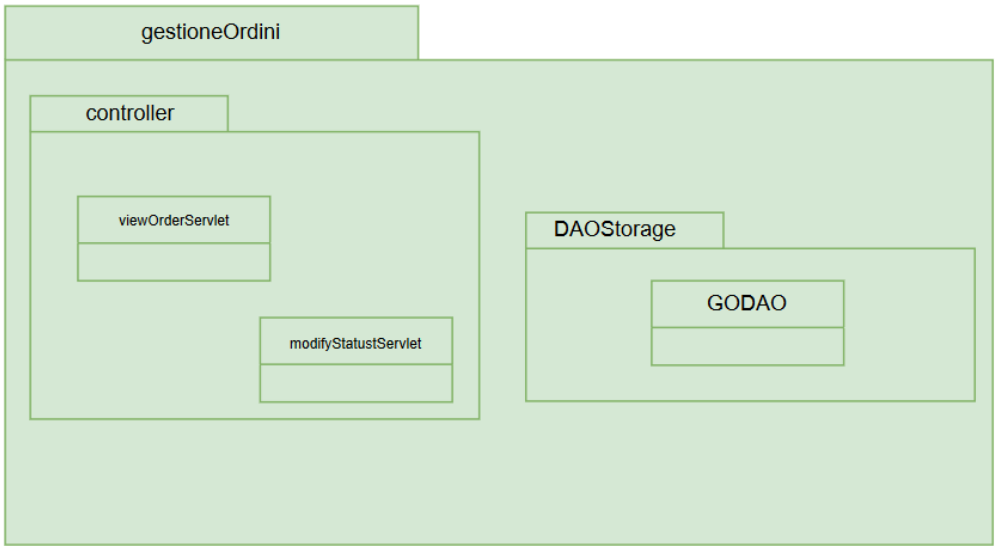


Package contente due layer:

- controller: gestisce la logica applicativa del sottosistema di Gestione delle Inserzioni
- DAOStorage: gestisce la logica di interazione con il database per tutte le funzioni di gestione delle inserzioni

Questo package è responsabile della gestione delle inserzioni nel sistema. Gli amministratori possono visualizzare gli annunci attivi, inserire nuovi annunci, modificarli o eliminarli dal sistema.

Gestione degli Ordini:

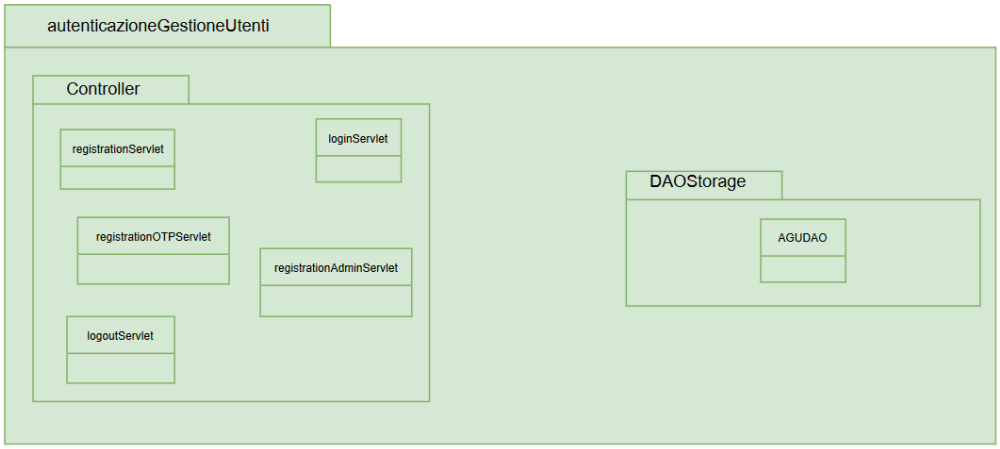


Package contente due layer:

- controller: gestisce la logica applicativa del sottosistema di Gestione degli Ordini
- DAOStorage: gestisce la logica di interazione con il database per tutte le funzioni di gestione degli ordini

Questo package si occupa della gestione degli ordini nel sistema. Gli amministratori possono accedere a una lista di ordini visualizzarne i dettagli e modificarne lo stato. I clienti possono fare lo stesso per i propri acquisti(ottenerne una lista e visualizzare i dettagli dei singoli acquisti)

Autenticazione e Gestione utenti:

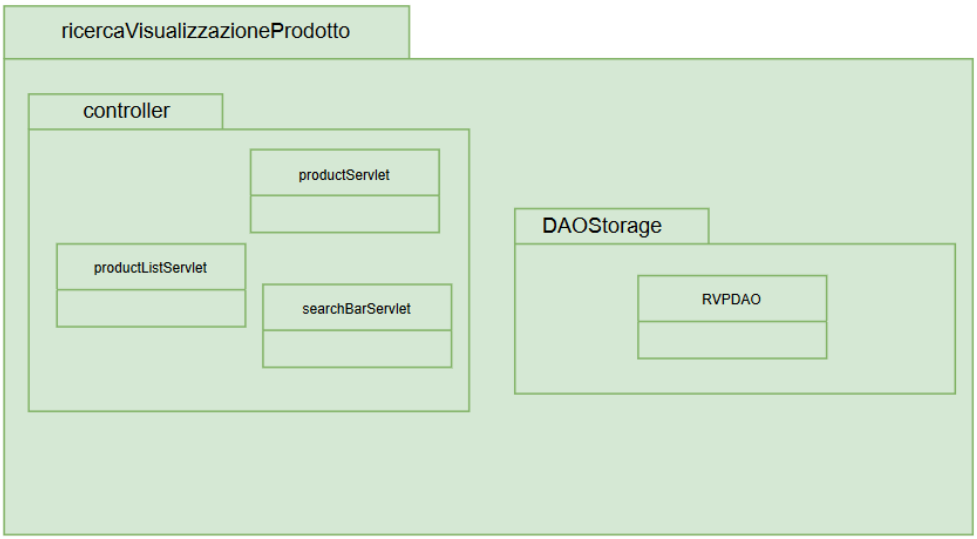


Package contente due layer:

- controller: gestisce la logica applicativa del sottosistema di Autenticazione e Gestione degli Utenti
- DAOStorage: gestisce la logica di interazione con il database per tutte le funzioni di autenticazione e gestione degli utenti

Questo package gestisce le operazioni legate all'autenticazione degli utenti e alla gestione dei loro account. Include la registrazione degli utenti admin attraverso nome-cognome-email-password e con la verifica attraverso un codice OTP, la registrazione degli utenti clienti attraverso nome-cognome-email-password, l'autenticazione tramite login e una funzione di logout.

Ricerca e Visualizzazione dei Prodotti:

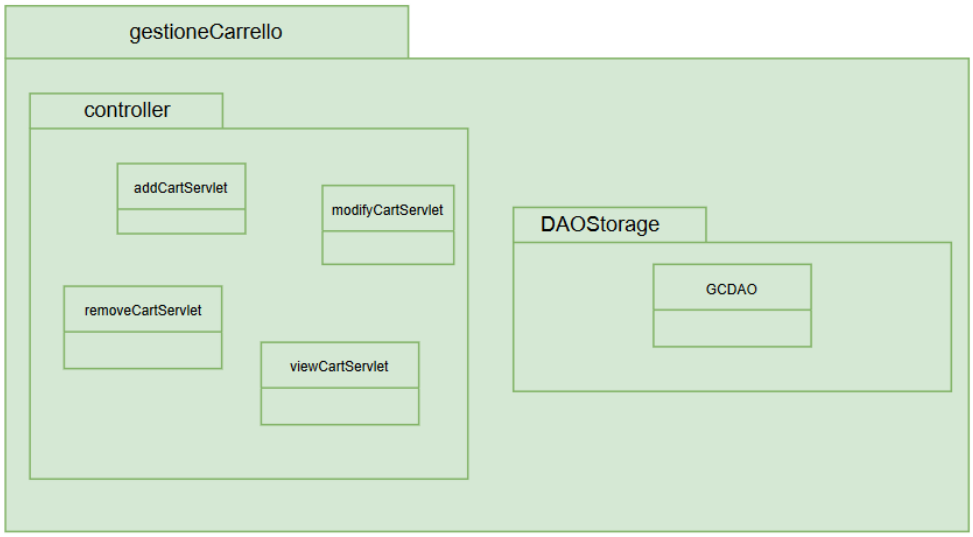


Package contente due layer:

- controller: gestisce la logica applicativa del sottosistema di Ricerca e Visualizzazione dei Prodotti
- DAOStorage: gestisce la logica di interazione con il database per tutte le funzioni di ricerca e visualizzazione dei prodotti

Questo package fornisce funzionalità di ricerca e visualizzazione dei sottoprodotti presenti nel database. Gli utenti possono effettuare ricerche generiche, filtrate o basate su parole chiave e della lista ottenuta andare alla pagina specifica del prodotto.

Gestione del Carrello:

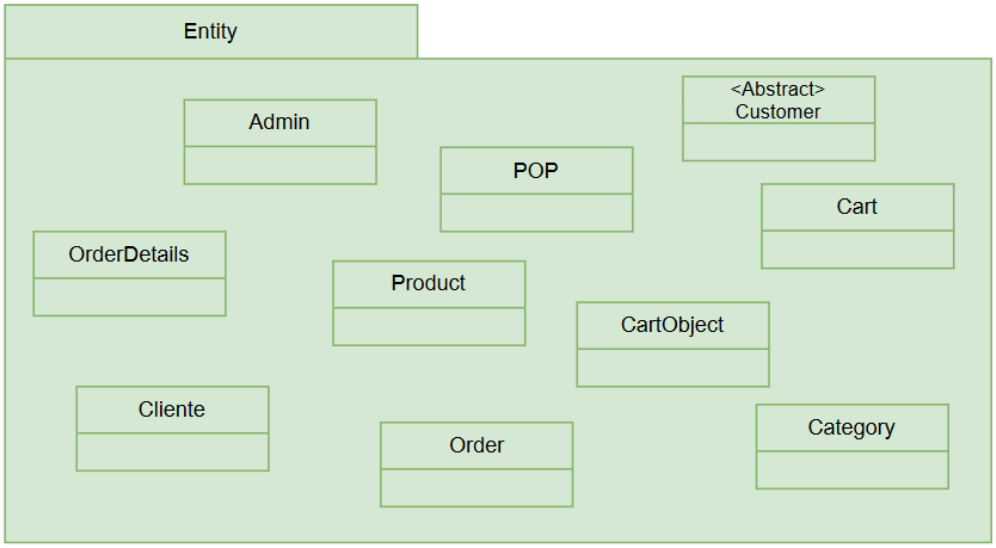


Package contente due layer:

- controller: gestisce la logica applicativa del sottosistema di Gestione del Carrello
- DAOStorage: gestisce la logica di interazione con il database per tutte le funzioni di gestione del carrello

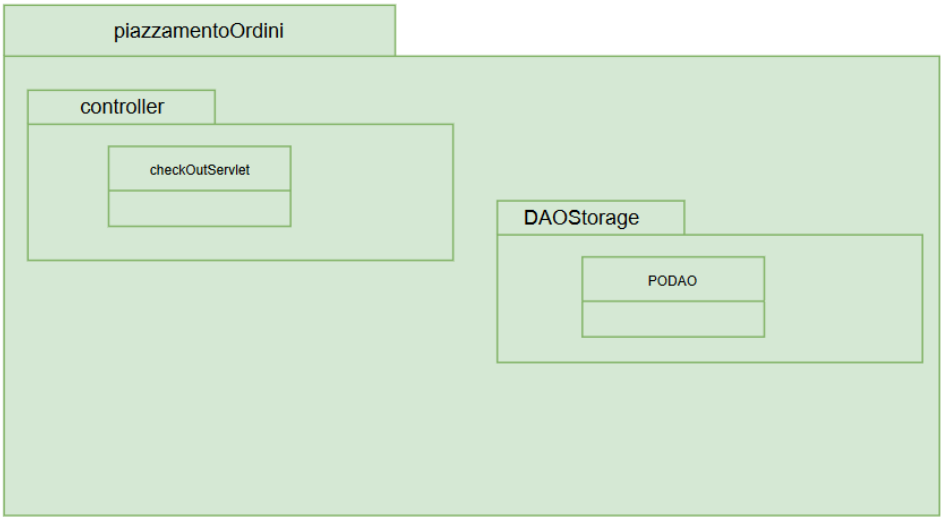
Questo package gestisce le operazioni legate al carrello degli acquisti. I clienti possono aggiungere prodotti al carrello, rimuoverli, modificare le quantità e visualizzare il contenuto del carrello.

Entity:



Questo package contiene al suo interno tutte le Entity del sistema al fine di aumentare la coesione e ridurre l'accoppiamento

Piazzamento Ordini:



Package contente due layer:

- controller: gestisce la logica applicativa del sottosistema di Piazzamento degli Ordini
- DAOStorage: gestisce la logica di interazione con il database per tutte le funzioni di piazzamento degli ordini

Questo package gestisce le fasi finali del processo di acquisto. I clienti possono aggiungere un indirizzo per la consegna e piazzare effettivamente l'ordine.

3. Class interfaces

Nome classe	loginServlet @extends HttpServlet
Descrizione	Questa classe gestisce il login degli utenti al sito
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doPost(HttpServletRequest ,HttpServletResponse)
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Verifica la correttezza dei dati ed esegue il login
Pre-condizione	//
Post-condizione	//

Nome classe	logoutServlet @extends HttpServlet
Descrizione	Questa classa permette ad un utente di eseguire il logout dal sito
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Il metodo esegue la funzione di logout perdendo traccia dell'utente in sessione
Pre-condizione	//
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doGet(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Nome classe	registrationServlet @extends HttpServlet
Descrizione	Questa classe permette ad un nuovo cliente di registrarsi al sito
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue un reindirizzamento alla pagina di inserimento dei dati per la registrazione del cliente
Pre-condizione	//
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Verifica i dati di inserimento ed esegue la registrazione del nuovo cliente
Pre-condizione	//
Post-condizione	//

Nome classe	registrationAdminServlet @extends HttpServlet
Descrizione	Questa classe esegue una registrazione di un nuovo admin
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue un reindirizzamento alla pagina per l'inserimento dei dati dell'admin
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Verifica i dati di inserimento e reindirizza l'admin alla pagina di inserimento dell'OTP
Pre-condizione	//
Post-condizione	//

Nome classe	registrationOTPServlet @extends HttpServlet
Descrizione	Questa classe permette di completare la registrazione lato admin
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Verifica i dati di inserimento e si collega ad un software esterno per verificare il codice OTP completando poi la registrazione
Pre-condizione	//
Post-condizione	//

Nome classe	searchBarServlet @extends HttpServlet
Descrizione	Questa classe recupera i prodotti dal database e li mostra in fase di ricerca tramite searchBar
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doPost()
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Ritorna in formato JSON una lista di nomi di prodotti in base ad una keyword
Pre-condizione	//
Post-condizione	//

Nome classe	productServlet @extends HttpServlet
Descrizione	Permette di recuperare i dati di un intero prodotto
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Tramite un ID restituisce i dati del prodotto richiesto
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doGet(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Nome classe	productListServlet @extends HttpServlet
Descrizione	Questa classe permette di visionare una lista di prodotti
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questa classe restituisce una lista di tutti i prodotti presenti nel database eventualmente filtrata per Categoria/Disponibilità/Nome
Pre-condizione	//
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doGet(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Nome classe	updateProductServlet @extends HttpServlet
Descrizione	Questa classe aggiorna i dati di un prodotto
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una redirect alla page di modifica prodotto recuperandone i dati
Pre-condizione	Context updateProductServlet :: doGet(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Verifica i dati di modifica inseriti ed aggiorna il prodotto
Pre-condizione	Context updateProductServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//

Nome classe	addProductServlet @extends HttpServlet
Descrizione	Questa classe aggiunge un nuovo prodotto nel database
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue la redirect alla pagina di inserimento di un nuovo prodotto
Pre-condizione	Context addProductServlet:: doGet(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Veridica i dati inseriti ed aggiunge il nuovo prodotto
Pre-condizione	Context addProductServlet:: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//

Nome classe	deleteProductServlet @extends HttpServlet
Descrizione	Questa classe elimina un prodotto dal database
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Viene eliminato il prodotto scelto in base al suo ID
Pre-condizione	Context deleteProductServlet :: doGet(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doGet(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Nome classe	manageProductServlet @extends HttpServlet
Descrizione	Questa classe permette di visualizzare la lista di inserzioni disponibili nel sito
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Recupera i prodotti disponibili e li immagazzina in una lista
Pre-condizione	Context manageProductServlet :: doGet(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Esegue una chiamata al doGet(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Nome classe	POPAdminServlet @extends HttpServlet
Descrizione	Questa classe permette ad un admin di visualizzare tutti i POP inviati
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Recupera tutti i POP inviati e li immagazzina in una lista
Pre-condizione	Context POPAdminServlet :: doGet(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doGet(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Nome classe	POPClientServlet @extends HttpServlet
Descrizione	Questa classe permette ad un cliente di visualizzare tutti i POP a lui inviati
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Recupera tutti i POP ricevuti e li immagazzina in una lista
Pre-condizione	Context POPClientServlet :: doGet(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==1
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doGet(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Nome classe	sendPOPServlet @extends HttpServlet
Descrizione	Questa classe permette ad un admin di inviare notifiche POP
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo permette ad un admin di inviare notifiche POP ad un singolo cliente/in modalità broadcast
Pre-condizione	Context sendPOPServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//

Nome classe	addCartServlet @extends HttpServlet
Descrizione	Questa classe permette di aggiungere un prodotto al carrello
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo permette di aggiungere una data quantità di un prodotto al carrello
Pre-condizione	Context addCartServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==1
Post-condizione	//

Nome classe	modifyCartServlet @extends HttpServlet
Descrizione	Questa classe permette di modificare un prodotto nel carrello
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo permette di modificare una data quantità di un prodotto nel carrello
Pre-condizione	Context modifyCartServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==1
Post-condizione	//

Nome classe	removeCartServlet @extends HttpServlet
Descrizione	Questa classe permette di eliminare un prodotto dal carrello
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo permette di eliminare un prodotto dal carrello
Pre-condizione	Context modifyCartServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==1
Post-condizione	//

Nome classe	viewCartServlet @extends HttpServlet
Descrizione	Questa classe permette di visualizzare la lista di prodotti nel carrello
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo permette di visualizzare il carrello del cliente
Pre-condizione	Context modifyCartServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==1
Post-condizione	//

Nome classe	viewOrderServlet @extends HttpServlet
Descrizione	Questa classe permette di visualizzare la lista di ordini effettuati
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo permette di visualizzare:ad un admin la lista degli ordini totali effettuati, ad un cliente la lista degli acquisti effettuati
Pre-condizione	Context viewOrdiniServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")!=0
Post-condizione	//

Nome classe	modifyStatusServlet @extends HttpServlet
Descrizione	Questa classe permette di modificare lo status di un ordine
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//

Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//

Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo verifica i dati di inserimento e modifica lo status di un ordine
Pre-condizione	Context modifyStatusServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==2
Post-condizione	//

Nome classe	checkOutServlet @extends HttpServlet
Descrizione	Questa classe permette di piazzare un ordine
Metodi	+ doGet(HttpServletRequest, HttpServletResponse) : void + doPost(HttpServletRequest, HttpServletResponse) : void
Invariante di classe	//
Metodo + doGet(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Effettua una chiamata al doPost(HttpServletRequest, HttpServletResponse)
Pre-condizione	//
Post-condizione	//
Metodo + doPost(HttpServletRequest, HttpServletResponse) : void @param HttpServletRequest request passata attraverso il protocollo HTTP @param HttpServletResponse response passata attraverso il protocollo HTTP @return void @throws throws ServletException, IOException	
Descrizione	Questo metodo verifica i dati di inserimento e permette ad un cliente di piazzare un ordine
Pre-condizione	Context modifyStatusServlet :: doPost(HttpServletRequest, HttpServletResponse) pre: session.getAttribute("loginStatus")==1
Post-condizione	//

Nome classe	AGUDAO
Descrizione	Questa classe gestisce la comunicazione con il database per le funzioni che riguardano l'accesso e la gestione degli utenti
Metodi	+ loginUtente(String, String, connectionPoolAbstraction) : int + returnCustomerData(String, connectionPoolAbstraction) : Customer + registerCliente(Cliente, connectionPoolAbstraction) : int + registerAdmin(Admin, connectionPoolAbstraction): int
Invariante di classe	//
Metodo + loginUtente(String, String, connectionPoolAbstraction) : int @param String email dell'utente @param String password dell'utente (non codificata) @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Questo metodo verifica se nel database è presente una entry in cui email e password date in input combaciano
Pre-condizione	//
Post-condizione	//
Metodo + returnCustomerData(String, connectionPoolAbstraction) : Customer @param String email dell'utente @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return Customer oggetto Customer contenente i dati presenti nel DB	
Descrizione	Data in input l'email di un customer ne recupera tutti i dati dal database
Pre-condizione	//
Post-condizione	//
Metodo + registerCliente(Cliente, connectionPoolAbstraction) : int @param Cliente oggetto Cliente contente i dati da immettere nel DB @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input un oggetto Cliente inserisce una nuova entry nel database con i suoi dati
Pre-condizione	//
Post-condizione	//
Metodo + registerAdmin(Admin, connectionPoolAbstraction): int @param Admin oggetto Admin contente i dati da immettere nel DB @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input un oggetto Admin inserisce una nuova entry nel database con i suoi dati
Pre-condizione	//
Post-condizione	//

Nome classe	GIDAO
Descrizione	Questa classe gestisce la comunicazione con il database per le funzioni che riguardano la gestione delle inserzioni
Metodi	+ updateProductInformation (Product, connectionPoolAbstraction) : int + deleteProduct (int, connectionPoolAbstraction) : int + addProduct (Product, connectionPoolAbstraction) : int
Invariante di classe	//
Metodo + updateProductInformation(Prodotto, connectionPoolAbstraction) : int @param Product oggetto Product contente i dati da modificare nel DB @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input un oggetto Product aggiorna l'entry del database con i nuovi dati
Pre-condizione	//
Post-condizione	//
Metodo + deleteProduct(int, connectionPoolAbstraction) : int @param int id del Product da eliminare nel DB @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input l'id di un prodotto elimina l'entry del database con quell'id
Pre-condizione	//
Post-condizione	//
Metodo + addProduct(Prodotto, connectionPoolAbstraction) : int @param Product oggetto Product contente i dati da immettere nel DB @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input un oggetto Product inserisce una nuova entry nel database con i suoi dati
Pre-condizione	//
Post-condizione	//

Nome classe	GNDAO
Descrizione	Questa classe gestisce la comunicazione con il database per le funzioni che riguardano la gestione delle notifiche POP
Metodi	+ clientPOP(int, connectionPoolAbstraction) : List<POP> + adminPOP(connectionPoolAbstraction) : List<POP> + sendIndividualPOP(String, String, connectionPoolAbstraction) : int + sendBroadcastPOP(String,connectionPoolAbstraction) : int
Invariante di classe	//
Metodo + clientPOP(int, connectionPoolAbstraction) : List<POP> @param int id del Customer di cui recuperare i POP @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<POP> lista dei POP recuperati dal DB	
Descrizione	Dato in input l'id di un Cliente restituisce tutti i POP presenti nel database per quell'id
Pre-condizione	//
Post-condizione	//
Metodo + adminPOP(connectionPoolAbstraction) : List<POP> @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<POP> lista dei POP recuperati dal DB	
Descrizione	Restituisce tutte le entry POP presenti nel database
Pre-condizione	//
Post-condizione	//
Metodo + sendIndividualPOP(int, String, connectionPoolAbstraction) : int @param int id del Customer a cui inviare il POP @param String testo da inviare al Customer @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input l'email di un Cliente ed il testo della notifica, dopo aver verificato l'esistenza del cliente inserisce nel database una entry POP
Pre-condizione	//
Post-condizione	//
Metodo + sendBroadcastPOP(String,connectionPoolAbstraction) : int @param String testo da inviare al Customer @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input il testo della notifica, inserisce nel database una entry POP per ogni Client
Pre-condizione	//
Post-condizione	//

Nome classe	GCDAO
Descrizione	Questa classe gestisce la comunicazione con il database per le funzioni che riguardano la gestione del carrello
Metodi	+ viewCart(int, connectionPoolAbstraction) : Cart + addCartObject(int, int, int, connectionPoolAbstraction) : int + removeCartObject(int, int, connectionPoolAbstraction) : int + modifyCartObject(int, int, int, connectionPoolAbstraction) : int
Invariante di classe	//
Metodo + viewCart(int, connectionPoolAbstraction) : Cart @param int id del Customer di cui recuperare il Car @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al D @return Cart oggetto Cart contente il carrello del Customer	
Descrizione	Dato in input l'id di un Cliente restituisce il suo carrello con i relativi cartObject
Pre-condizione	//
Post-condizione	//
Metodo + addCartObject(int, int, int, connectionPoolAbstraction) : int @param int id del Customer al quale aggiungere prodotto nel Car @param int id del Product da aggiungere nel Car @param: int quantità di prodotto da aggiungere nel Cart @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input l'id di un Cliente ed il relativo id del prodotto da aggiungere lo aggiunge nel database nella quantità data in input dopo aver verificato l'effettiva disponibilità della stessa
Pre-condizione	//
Post-condizione	//
Metodo + removeCartObject(int, int, connectionPoolAbstraction) : int @param int id del Customer al quale rimuovere prodotto nel Cart @param int id del CartObject da rimuovere al Cart @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Dato in input l'id di un Cliente ed il relativo cartObjectID dopo aver verificato che appartenga effettivamente al suo carrello lo rimuove dal database
Pre-condizione	//
Post-condizione	//
Metodo + modifyCartObject(int, int, int, connectionPoolAbstraction) : int @param int id del Customer al quale aggiungere prodotto nel Cart @param int id del CartObject da modificare nel Cart @param int quantità di prodotto da modificare al CartObject @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	

Descrizione	Dato in input l'id di un Cliente ed il relativo cartObjectID dopo aver verificato che appartenga effettivamente al suo carrello ne modifica la quantità in base a quella data in input
Pre-condizione	//
Post-condizione	//

Nome classe	GODAO
Descrizione	Questa classe gestisce la comunicazione con il database per le funzioni che riguardano la gestione degli Ordini
Metodi	+ retriveOrder(connectionPoolAbstraction) : List<Order> + retriveCustomerOrder(int, connectionPoolAbstraction) : List<Order> - retriveOrderObject(Order, connectionPoolAbstraction) : List<OrderObject> +modifyOrderStatus(int, String, connectionPoolAbstraction): int
Invariante di classe	//
Metodo + retriveOrder (connectionPoolAbstraction) : List<Order> @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<Order> lista degli Order recuperati dal DB	
Descrizione	Recupera dal database tutti gli ordini effettuati
Pre-condizione	//
Post-condizione	//
Metodo + retriveCustomerOrder (int, connectionPoolAbstraction) : List<Order> @param int id del Customer di cui bisogna recuperare gli Order @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<Order> lista degli Order recuperati dal DB	
Descrizione	Recupera dal database tutti gli ordini effettuati del cliente dato il suo ID in input
Pre-condizione	//
Post-condizione	//
Metodo - retriveOrderObject (Order, connectionPoolAbstraction) : List<OrderObject> @param Order Order di cui bisogna recuperare gli OrderObject @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<OrderObject> lista degli OrderObject recuperati dal DB	
Descrizione	Recupera dal database tutti gli orderObjcet relativi all'ordine dato in input
Pre-condizione	//
Post-condizione	//
Metodo + modifyOrderStatus (int, String, connectionPoolAbstraction) : int @param int id dell'Order di cui bisogna modificare lo status @param String stato da aggiornare @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentante l'esito della query	
Descrizione	Modifica lo status di un ordine dato il suo ID e lo stato in cui deve essere aggiornato
Pre-condizione	//
Post-condizione	//

Nome classe	RVPDAO
Descrizione	Questa classe gestisce la comunicazione con il database per le funzioni che riguardano la ricerca e la visualizzazione dei prodotti
Metodi	+ searchDropdownProduct(String, connectionPoolAbstraction) : List<String> + getSearch(String, connectionPoolAbstraction) : List<Integer> + productFromID(int, connectionPoolAbstraction) : Product + allProduct(connectionPoolAbstraction) : List<Product> + allCategory(connectionPoolAbstraction) : List<Category> + getCategoryName(int, connectionPoolAbstraction) : String + productFilteredByCategory(int, connectionPoolAbstraction) : List<Product> + productFilteredByDisponibility(int, connectionPoolAbstraction) : List<Product>
Invariante di classe	//

Metodo
+ searchDropdownProduct(String, connectionPoolAbstraction) : List<String>

@param String nome del prodotto da ricercare
@param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB
@return List<String> Lisa contenente tutti i nomi presi dal database che contengono la stringa in input da mostrare nel suggerimento di inserimento

Descrizione	Data in input la stringa inserita dall'utente nella searchbar restituisce una lista di stringhe di tutti i nomi di prodotti presenti nel database che contengono quella stringa da visualizzare nel suggerimento di inserimento
Pre-condizione	//
Post-condizione	//

Metodo
+ getSearch(String, connectionPoolAbstraction) : List<Integer>

@param String nome del prodotto da cercare
@param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB
@return List<Integer> lista contenente tutti gli id presi dal database che contengono la stringa in input

Descrizione	Data in input la stringa inserita dall'utente nella searchbar restituisce una lista di contenente gli id di tutti prodotti presenti nel database
Pre-condizione	//
Post-condizione	//

Metodo
+ productFromID(int, connectionPoolAbstraction) : Product

@param int id del prodotto da cercare
@param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB
@return Product oggetto Product con lo stesso ID specificato in input

Descrizione	Dato in input l'id di un prodotto recupera dal database la entry con l'id indicato
Pre-condizione	//
Post-condizione	//

Metodo
+ allProduct(connectionPoolAbstraction) : List<Product>

@param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB

@return List<Product> lista contenente tutti gli oggetti Product presenti nel database	
Descrizione	Restituisce la lista di tutti i prodotti presenti nel database
Pre-condizione	//
Post-condizione	//
Metodo + allCategory(connectionPoolAbstraction) : List<Category> @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<Category> lista contenente tutti gli oggetti Category presenti nel database	
Descrizione	Restituisce la lista di tutte le categorie presenti nel database
Pre-condizione	//
Post-condizione	//
Metodo + getCategoryName(int, connectionPoolAbstraction) : String @param int id della categoria da cercare @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return Strign nome dell'oggetto Category con lo stesso ID specificato in input	
Descrizione	Dato l'id di una categoria ne restituisce il nome della entry che è associata all'id indicato
Pre-condizione	//
Post-condizione	//
Metodo + productFilteredByCategory(int, connectionPoolAbstraction) : List<Product> @param int id della categoria da cercare @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<Product> lista di oggetti Product che hanno lo stesso categoryID specificato in input	
Descrizione	Dato l'id di una categoria restituisce una lista di tutti i prodotti nel database che hanno come categoria quella in riferimento all'id inserito
Pre-condizione	//
Post-condizione	//
Metodo + productFilteredByDisponibility(int, connectionPoolAbstraction) : List<Product> @param int tipo di disponibilità da cercare @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return List<Product> lista di oggetti Product che hanno la stessa tipologia di disponibilità indicata in input	
Descrizione	Data una quantita (0 non disponibile o 1 disponibile) restituisce una lista di tutti i prodotti nel database che sono disponibili (caso 1) o non disponibili (caso 0)
Pre-condizione	//
Post-condizione	//

Nome classe	PODAO
Descrizione	Questa classe gestisce la comunicazione con il database per le funzioni che riguardano il piazzamento degli ordini
Metodi	+ placeOrder(int, Cart, String, String, String, String, String, connectionPoolAbstraction) : int
Invariante di classe	//
Metodo + placeOrder(int, double, String, String, String, String, String, connectionPoolAbstraction) : int @param int id del cliente da associare all'ordine che si sta creando @param double prezzo totale dell'ordine @param String stringa che rappresenta al via dell'indirizzo @param String stringa che rappresenta la città dell'indirizzo @param String stringa che rappresenta la provincia dell'indirizzo @param String stringa che rappresenta il CAP dell'indirizzo @param String stringa che rappresenta lo Stato dell'indirizzo @param connectionPoolAbstraction oggetto contenente credenziali e metodi per collegamento al DB @return int intero rappresentate l'esito della query	
Descrizione	Questo metodo prende in input l'id del cliente, il totale e un insieme di stringhe che rappresenta il suo indirizzo. Da queste informazioni creerà l'ordine con i relativi orderObject e li inserirà nel database sottraendo infine alla quantità dei prodotti disponibili alla vendita la quantita di prodotti acquistati nell'ordine e svuotando il carrello
Pre-condizione	//
Post-condizione	//

NOTA: Per le classi Entity non saranno approfonditi i metodi getter/setter e costruttori

Nome classe	Category
Descrizione	Questa classe rappresenta l'oggetto Entity "Category" persistente salvato nel database
Variabili	- ID: int - Nome: String - Descrizione: String
Metodi	+ new + getter/setter
Invariante di classe	//

Nome classe	Product
Descrizione	Questa classe rappresenta l'oggetto Entity "Product" persistente salvato nel database
Variabili	- ID: int - Nome: String - Descrizione: String - Immagine: byte[] - Colore: String - Lunghezza: double - Larghezza: double - Quantita: int - Prezzo: double - Categoria: String
Metodi	+ new + getter/setter
Invariante di classe	//

Nome classe	Customer
Descrizione	Questa classe rappresenta l'oggetto Entity "Customer" persistente salvato nel database
Variabili	- ID: int - Nome: String - Cognome: String - Email: String - Password: String
Metodi	+ new + getter/setter + cryptPassword(String): String
Invariante di classe	//
Metodo + cryptPassword(String) : String @param String stringa contenente la password da cryptare @return String stringa cryptata con algoritmo SHA-1	
Descrizione	Questo metodo codifica la stringa presa in input in una stringa criptata secondo l'algoritmo "SHA-1"
Pre-condizione	//
Post-condizione	//

Nome classe	Order
Descrizione	Questa classe rappresenta l'oggetto Entity "Order" persistente salvato nel database
Variabili	- ID: int - ListaProdotti: OrderObject[] - CustomerID: int - DataPiazzamento: date - Via: String - Citta: String - Provincia: String - CAP: String - StatusOrder: String
Metodi	+ new + getter/setter +addListaProdotti(OrderObject): void +removeListaProdotti(OrderObject): void +getProdottoOrdine(number): OrderObject +isListaProdottiEmpty(): boolean +sizeListaProdotti(): int
Invariante di classe	//
Metodo + addListaProdotti(OrderObject): void @param OrderObject oggetto da aggiungere dalla lista	
Descrizione	Questo metodo dato un oggetto OrderObject in input lo aggiunge alla lista interna all'oggetto Order denominata "ListaProdotti"
Pre-condizione	//
Post-condizione	Context Order :: addListaProdotti(OrderObject) post: sizeListaProdotti() = @pre sizeListaProdotti() + 1
Metodo + removeListaProdotti(OrderObject): void @param OrderObject oggetto da rimuovere dalla lista	
Descrizione	Questo metodo dato un oggetto OrderObject in input lo rimuove alla lista interna all'oggetto Order denominata "ListaProdotti"
Pre-condizione	//
Post-condizione	Context Order :: addListaProdotti(OrderObject) post: sizeListaProdotti() = @pre sizeListaProdotti() - 1
Metodo + getProdottoOrdine(int): OrderObject @param int intero rappresentante la posizione nella lista dell'oggetto da recuperare @return OrderObject oggetto che si trova nella posizione scelta	
Descrizione	Questo metodo dato un indice in input restituisce l'oggetto che si trova in quella posizione nella lista interna all'oggetto Order denominata "ListaProdotti"
Pre-condizione	//
Post-condizione	//
Metodo +isListaProdottiEmpty(): Boolean @return boolean booleano true se la lista è vuota altrimenti false	

Descrizione	Questo metodo restituisce true se la lista interna all'oggetto Order denominata "ListaProdotti" è vuota, altrimenti false
Pre-condizione	//
Post-condizione	//
Metodo +sizeListaProdotti(): int @return int intero contenente il numero di oggetti nella lista ListaProdotti	
Descrizione	Questo metodo restituisce il numero di oggetti che si trovano all'interno della lista interna all'oggetto Order denominata "ListaProdotti"
Pre-condizione	//
Post-condizione	//

Nome classe	OrderObject
Descrizione	Questa classe rappresenta l'oggetto Entity "OrderObject" persistente salvato nel database
Variabili	- ID: int - Product: Product - Quantita: int
Metodi	+ new + getter/setter
Invariante di classe	//

Nome classe	CartObject
Descrizione	Questa classe rappresenta l'oggetto Entity "CartObject" persistente salvato nel database
Variabili	- ID: int - Product: Product - Quantita: int
Metodi	+ new + getter/setter
Invariante di classe	//

Nome classe	Cart
Descrizione	Questa classe rappresenta l'oggetto Entity "Cart" persistente
Variabili	- ID: int - Carrello: CartObject[] - Customer: int
Metodi	+ new + getter/setter + addCarrello(CartObject): void + removeCarrello(CartObject): void + getProdottoCarrello(int): CartObject + isCarrelloEmpty(): boolean + sizeCarrello(): int
Invariante di classe	//
Metodo + addCarrello(CartObject): void @param CartObject oggetto da aggiungere dalla lista	
Descrizione	Questo metodo dato un oggetto CartObject in input lo aggiunge alla lista interna all'oggetto Cart denominata "Carrello"
Pre-condizione	//
Post-condizione	Context Cart :: addCarrello(CartObject) post: sizeCarrello() = @pre sizeCarrello() + 1
Metodo + removeCarrello(CartObject): void @param CartObject oggetto da rimuovere dalla lista	
Descrizione	Questo metodo dato un oggetto CartObject in input lo rimuove alla lista interna all'oggetto Cart denominata "Carrello"
Pre-condizione	//
Post-condizione	Context Cart :: addCarrello(CartObject) post: sizeCarrello() = @pre sizeCarrello() -1
Metodo + getProdottoCarrello(int): CartObject @param int intero rappresentante la posizione nella lista dell'oggetto da recuperare @return CartObject oggetto che si trova nella posizione scelta	
Descrizione	Questo metodo dato un indice in input restituisce l'oggetto che si trova in quella posizione nella lista interna all'oggetto Cart denominata "Carrello"
Pre-condizione	//
Post-condizione	//
Metodo +isCarrelloEmpty(): boolean @return boolean booleano true se la lista è vuota altrimenti false	
Descrizione	Questo metodo restituisce true se la lista interna all'oggetto Cart denominata "Carrello" è vuota, altrimenti false
Pre-condizione	//
Post-condizione	//
Metodo +sizeCarrello(): int @return int intero contenente il numero di oggetti nella lista Carrello	

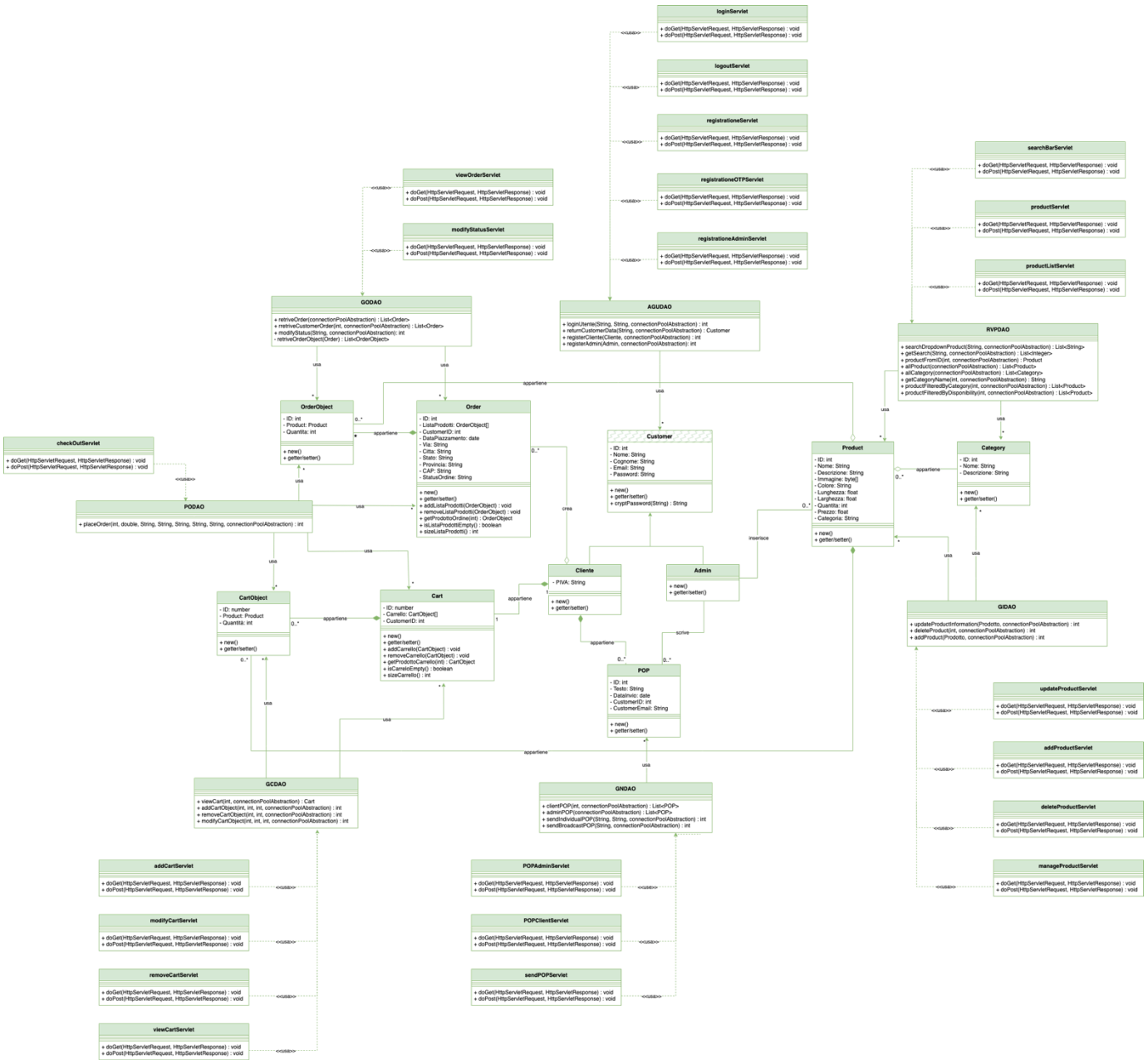
Descrizione	Questo metodo restituisce il numero di oggetti che si trovano all'interno della lista interna all'oggetto Cart denominata "Carrello"
Pre-condizione	//
Post-condizione	//

Nome classe	POP
Descrizione	Questa classe rappresenta l'oggetto Entity "POP" persistente
Variabili	- ID: int - Testo: String - DataInivio: Date - CustomerID: int - CustomerEmail: String
Metodi	+ new + getter/setter
Invariante di classe	//

Nome classe	Cliente @extends Customer
Descrizione	Questa classe rappresenta l'oggetto Entity "Cliente" persistente
Variabili	- PIVA: String
Metodi	+ new + getter/setter
Invariante di classe	//

Nome classe	Admin @extends Customer
Descrizione	Questa classe rappresenta l'oggetto Entity "Admin" persistente
Variabili	//
Metodi	+ new + getter/setter
Invariante di classe	//

4. Class Diagram

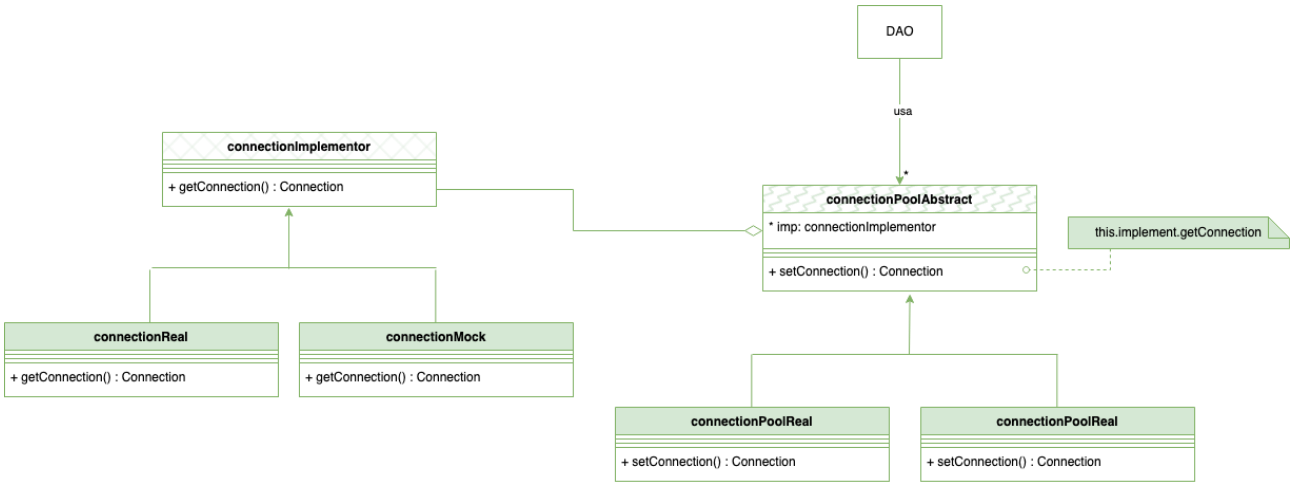


5. Elementi di riuso

5.1. Design Pattern usati

Nel Class Diagram illustrato precedentemente possiamo individuare sette classi di tipo DAO. Queste ultime devono poter interagire con il database e per farlo è necessario stabilirvi una connessione inserendone le credenziali. Essendo i vari team di sviluppo dislocati devono poter testare il codice sviluppato su database locali. Questo implicherebbe una modifica preventiva delle credenziali in ogni DAO ogni qualvolta un team inizi a lavorare in successione ad un team precedente. Inoltre in fase di testing è necessario testare i casi in cui il server sia irraggiungibile per verificare se questi vengano effettivamente previsti. Per ovviare a queste problematiche si è deciso di utilizzare un Bridge Pattern. In questo modo ogni metodo statico dei DAO dovrà necessariamente richiedere in input un connectionPoolAbstraction (classe astratta) estesa da due classi connectionPoolReal e connectionPoolMock (ovvero quelle che saranno effettivamente date in input al metodo). Queste ultime implementano il metodo ereditato dal padre il quale ritorna un oggetto Connection (utile per stabilire la connessione con il database) l'oggetto Connection viene ottenuto dal connectionImplementor interfaccia nella quale viene definita la firma del metodo con il quale viene prodotto l'oggetto Connection. Questo viene prodotto da due classi che implementano l'interfaccia:

- connectionReal dove l'oggetto Connection viene prodotto inserendovi credenziali effettive del database riducendosi ad unico punto dove i team di sviluppo dovranno cambiare le credenziali.
- connectionMock dove l'oggetto Connection viene prodotto inserendovi credenziali fittizie del database utile per testare i casi in cui il server sia irraggiungibile



6. Glossario

Sigla/Termine	Definizione
Package	Una struttura organizzativa in cui i componenti software correlati sono raggruppati insieme per scopi di organizzazione, manutenzione e riutilizzo
Class Interfaces	Un class interface definisce un contratto che specifica quali metodi una classe concreta deve implementare, senza fornire l'implementazione stessa
Invariante	Una proprietà che rimane costante durante l'esecuzione di un programma o in una determinata situazione, utile per garantire la correttezza del software
Pre-condizione	Una condizione che deve essere vera prima che un'operazione o funzione è stata eseguita con successo
Post-Condizione	Una condizione che deve essere vera dopo che un'operazione o funzione è stata eseguita con successo
Design pattern	Una soluzione generale e riutilizzabile a un problema comune di progettazione software, che fornisce una guida o uno schema per risolvere efficacemente il problema
Riuso	L'utilizzo di componenti software esistenti o di codice già sviluppato per risolvere un problema o per costruire nuove funzionalità, riducendo così lo sforzo di sviluppo e migliorando l'efficienza.
Package	Una struttura organizzativa in cui i componenti software correlati sono raggruppati insieme per scopi di organizzazione, manutenzione e riutilizzo
Class Interfaces	Un class interface definisce un contratto che specifica quali metodi una classe concreta deve implementare, senza fornire l'implementazione stessa