Università degli Studi di Salerno
Corso di Ingegneria del Software

McDelivery
Object Design Document
Versione 1.2

LOGO PROGETTO

Data: 30/12/2020

Partecipanti:

Nome	Matricola
GIGLIO SIMONE	0512105952
SORRENTINO GIOVANNI BATTISTA	0512105712
TORNINCASA GIUSEPPE	0512105994
ZIZZARI ANTONIO	0512105892

Scritto da:	GIGLIO, SORRENTINO, TORNINCASA, ZIZZARI

Revision History

Data	Versione	Descrizione	Autore
12/12/2020	1.0	Introduzione	Giglio, Tornincasa
12/12/2020	1.0	Packages	Sorrentino, Zizzari
13/12/2020	1.0	Classi Interfaces da 3.1 a 3.3	Giglio, Sorrentino
14/12/2020	1.0	Classi Interfaces da 3.4 a 3.7	Tornincasa, Zizzari
16/12/2020	1.0	Revisione intero Object Design Document	Giglio, Sorrentino, Tornincasa, Zizzari
24/12/2020	1.1	Correzione Class Interfaces	Giglio, Sorrentino, Tornincasa, Zizzari
30/12/2020	1.2	Correzioni varie	Giglio, Sorrentino, Tornincasa, Zizzari

Indice

1. *INTRODUZIONE*

1.1. Object Design Trade-offs

Comprensibilità vs Tempo: Il codice verrà reso il più comprensibile possibile per una migliore manutenibilità e per facilitare future modifiche. Per fare ciò, il codice sarà integrato da commenti volti a descrivere le funzionalità dei componenti, tuttavia questo richiederà un maggiore sforzo in termini di tempo e di sviluppo;

Manutenibilità vs Performance: Il sistema sarà progettato in modo da avere un'ottima manutenibilità garantita dal minimo accoppiamento possibile. Per ottenere ciò, bisognerà implementare un codice modulare per minimizzare la coesione, a discapito però delle performance del sistema;

Sicurezza vs Costi: La sicurezza sarà una componente fondamentale del nostro sistema, tuttavia, per mancanza di tempi di sviluppo, ci limiteremo ad implementare sistemi di sicurezza solamente per email e password;

Funzionalità vs Usabilità: Verrà realizzata un'interfaccia grafica chiara e intuitiva, con le sole funzionalità necessarie per l'uso del sistema. L'interfaccia verrà creata usando form e pulsanti ben descritti. In caso di errore da parte dell'utente, il sistema risponderà con messaggi che sottolineeranno l'errore effettuato dall'utente.

Sviluppo Rapido vs Funzionalità: Per mancanza di tempi di sviluppo, verranno implementate solamente le funzionalità richieste nel documento RAD.

1.2. Tool

Bootstrap: Bootstrap è un toolkit opensource per la creazione di siti e applicazioni Web. Contiene modelli di progettazione basati su HTML e CSS per la costruzione di varie componenti dell'interfaccia, come moduli, pulsanti e navigazione.
JQuery: una libreria JavaScript per applicazioni web il cui obiettivo è quello di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML. Il suo utilizzo ha lo scopo di ridurre la complessità del codice JavaScript durante l'implementazione.
AJAX: è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive che si basa su uno scambio di dati in background fra web browser e server, consentendo l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. Tramite il suo utilizzo è possibile effettuare richieste asincrone al database.
Selenium: E' un framework per il testing di applicazioni web;
JUnit: E' un framework per il testing di unità per il linguaggio di programmazione Java;

1.3. Linee guida per la documentazione delle interfacce

1.3.1. *Package, model, control e bean*

Le classi Java in questi package prevedono le seguenti convenzioni:

- Le classi presentano il seguente schema:
  1. Clausole Import;
  2. Dichiarazione della classe;
  3. Variabili d'istanza;
  4. Metodi;
- Il nome di una classe inizia con lettera maiuscola e se è formata da più parole, le

seguenti parole inizieranno con lettera maiuscola. Tra le parole non ci sarà lo spazio. Esempio:
NomeClasse;

- Il nome di un metodo o di una variabile d'istanza di una classe inizia con lettera minuscola e se è formata da più parole, le seguenti parole inizieranno con lettera maiuscola. Tra le parole non ci sarà lo spazio. Esempio:
  nomeMetodo;
- I nomi di classi, attributi e metodi devono risultare il quanto più possibile significativi per ciò che implementano;
- L'insieme delle dichiarazioni delle variabili d'istanza è preceduto e seguito da una riga vuota;
- L'implementazione di un metodo è preceduta e seguita da una riga vuota;
- Per le classi Bean vengono implementati i metodi Getters & Setters e ove necessario vengono implementati metodi equals(), clone() e toString();
- I commenti devono essere inseriti seguendo lo standard JavaDoc e vanno inseriti solo nei casi in cui sono ritenuti necessari per una buona leggibilità;
- I commenti possono essere inseriti sia con il formato
  //COMMENTO oppure /*COMMENTO*/;
- Qualsiasi variabile presente in un blocco più interno non è mai dichiarata con lo stesso nome di una variabile presente in un blocco più esterno;
- Il linguaggio deve essere indentato inserendo un carattere di tabulazione aggiuntivo ogni qualvolta che un blocco di codice è innestato all'interno di un altro;

1.3.2. *Le interfacce grafiche*

- Le pagine JSP devono, quando compilate, produrre un documento conforme allo standard HTML5;
- Il codice dell'HTML statico deve utilizzare l'indentazione secondo i seguenti criteri:
  - ✓ L'indentazione deve essere mantenuta tramite i caratteri di tabulazione;
  - ✓ I tag di chiusura e di apertura di un blocco devono essere allo stesso livello di indentazione;
  - ✓ Il blocco all'interno di un tag di apertura e chiusura deve avere un carattere di tabulazione aggiuntivo;
- Gli stili CSS non in-line devono essere collocati in file CSS separati;

1.3.3. *Database*

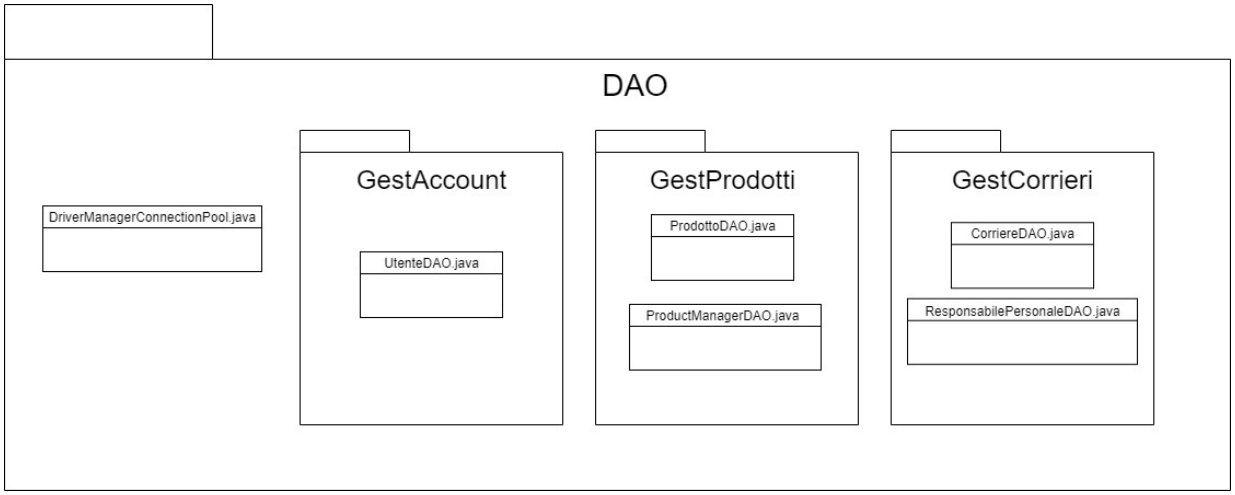
- Per le tabelle del Database vanno seguite le seguenti regole:
  - ✓ Il nome di una tabella inizia con lettera minuscola e se è formata da più parole, le seguenti parole inizieranno con lettera maiuscola. Tra le parole non ci sarà lo spazio.
  - ✓ Il nome dovrà essere un sostantivo singolare.
- Per gli attributi delle tabelle del Database vanno eseguite le seguenti regole:
  - ✓ Devono essere costruiti da sole lettere minuscole.
  - ✓ Se l'attributo è costruito da più parole, le due parole vengolo separate da un underscore. Esempio: nome_campo. Il nome deve essere un sostantivo singolare.

1.4.Riferimenti

RAD: Requirement Analysis Document

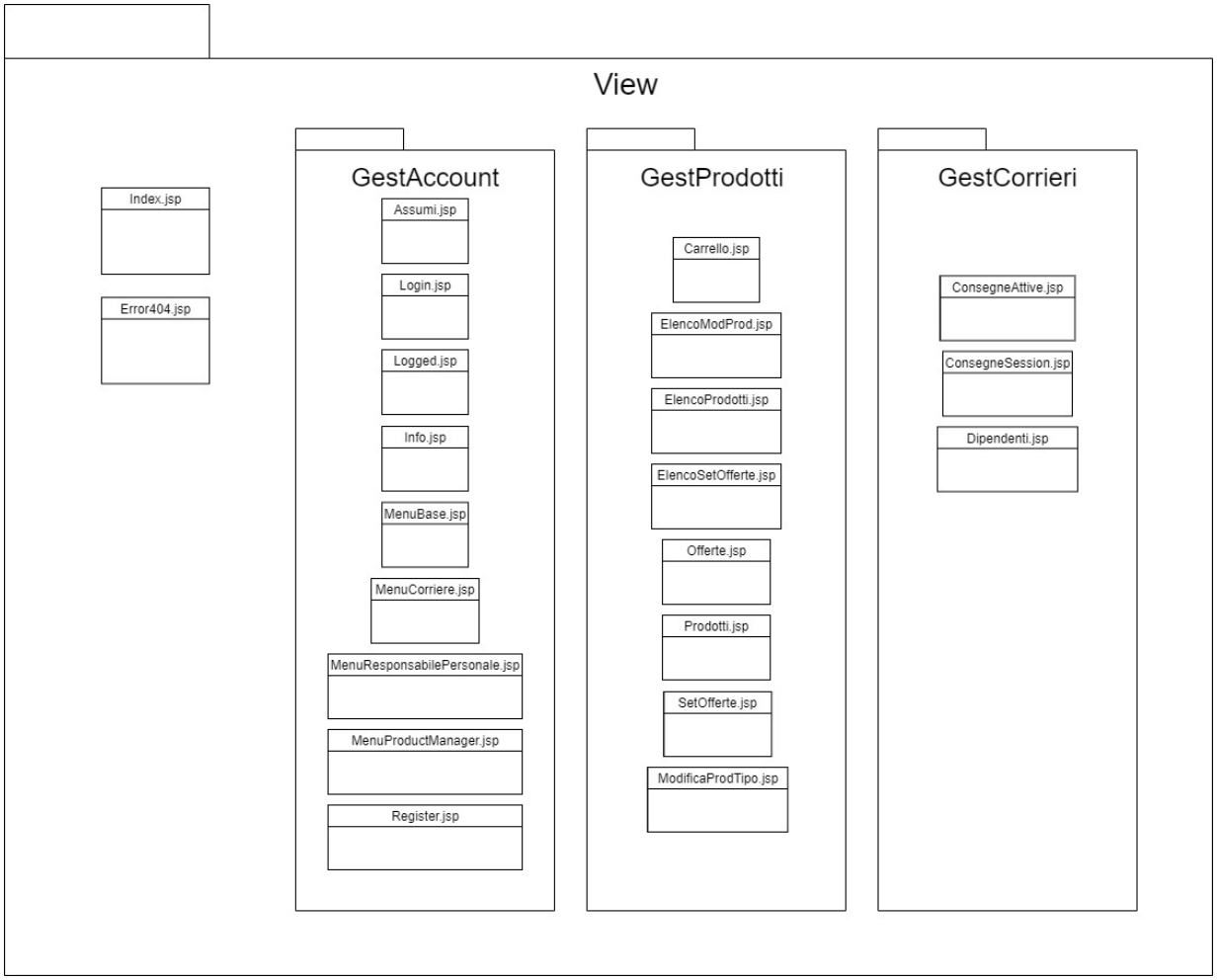
2. PACKAGES

2.1.Package Model



Classe	Descrizione
CorriereDAO.java	Contiene i metodi che permettono di manipolare le informazioni relative all'entità Corriere
ProdottoDAO.java	Contiene i metodi che permettono di manipolare le informazioni relative all'entità Prodotto
ProductDAO.java	Contiene i metodi che permettono di manipolare le informazioni relative all'utente Product Manager
ResponsabilePersonaleDAO.java	Contiene i metodi che permettono di manipolare le informazioni relative all'utente Responsabile del Personale
UtenteDAO.java	Contiene i metodi che permettono di manipolare le informazioni relative a tutti gli utenti
DriverManagerConnectionPool.java	Permette la connessione con il database

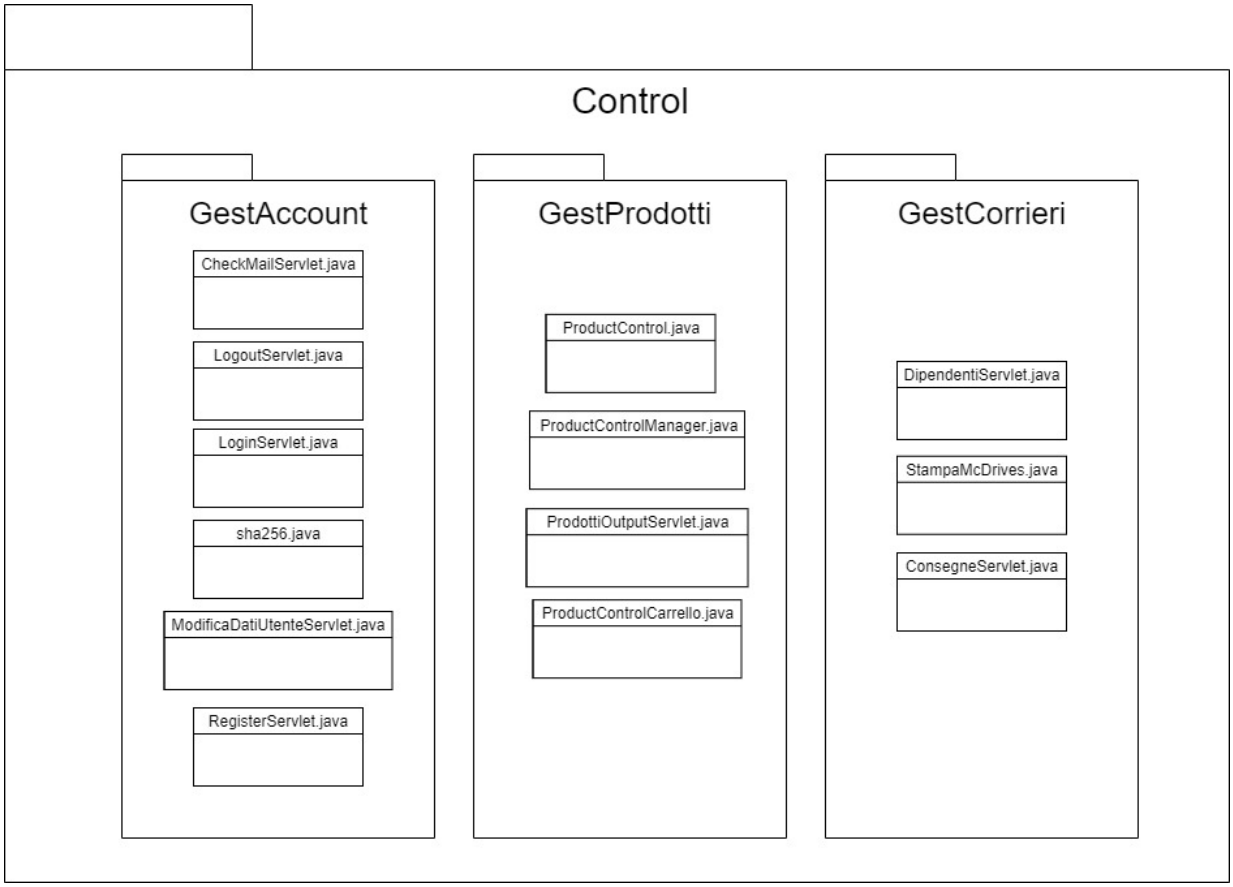
2.2.Package View



JSP	Descrizione
Assumi.jsp	Jsp per la costruzione della pagina per l'inserimento di un nuovo corriere nel sistema
Carrello.jsp	Jsp per la costruzione della pagina per la gestione e la visualizzazione del carrello
ConsegneAttive.jsp	Jsp per la costruzione della pagina per la visualizzazione delle consegne che si possono prendere in carico
ConsegneSession.jsp	Jsp per la costruzione della pagina per la visualizzazione delle consegne prese in carico nell'ultima sessione di lavoro
Dipendenti.jsp	Jsp per la costruzione della pagina per la visualizzazione e licenziamento dei corrieri
ElencoModProd.jsp	Jsp per la costruzione della pagina per la visualizzazione della pagina in cui è possibile effettuare modifiche ai prodotti
ElencoProdotti.jsp	Jsp per la costruzione della pagina per la visualizzazione dei prodotti di una determinata categoria
ElencoSetOfferte.jsp	Jsp per la costruzione della pagina per la visualizzazione della pagina in cui è possibile inserire uno sconto sui prodotti

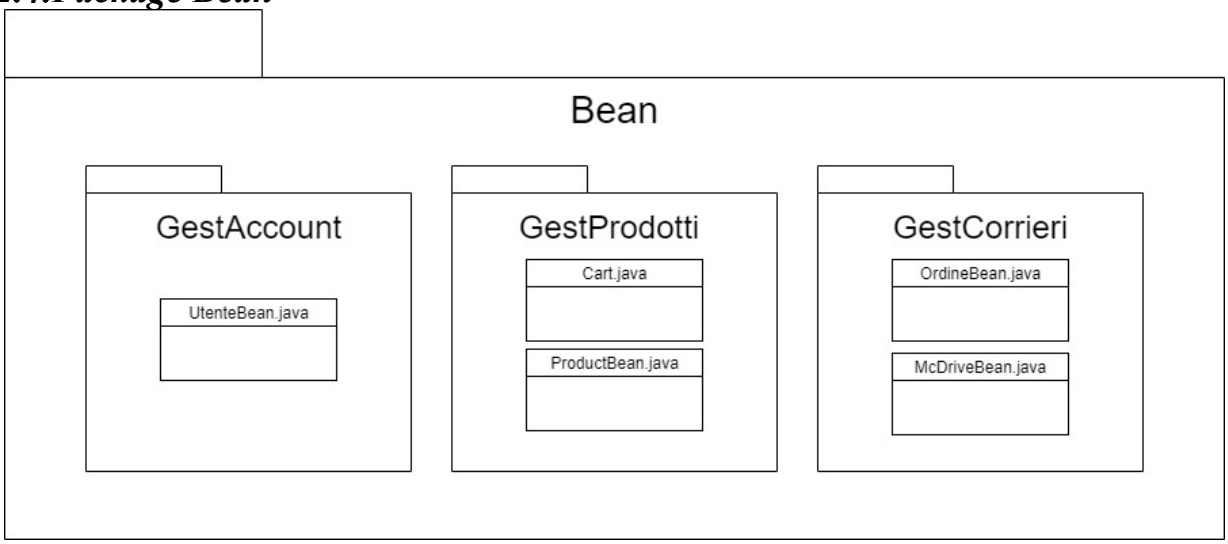
Error404.jsp	Jsp per la costruzione della pagina per visualizzare il messaggio di errore quando la pagina alla quale si vuole accedere non esiste
Index.jsp	Jsp per la costruzione della pagina per la pagina della homepage
Info.jsp	Jsp per la costruzione della pagina per le informazioni sui McDrive presenti nel database
Logged.jsp	Jsp per la costruzione della pagina per la visualizzazione e manipolazione dei dati personali dell'utente
Login.jsp	Jsp per la costruzione della pagina per poter effettuare il login
MenuBase.jsp	Jsp per la costruzione della pagina contentente le funzionalità disponibili per l'Utente Consumatore
MenuCorriere.jsp	Jsp per la costruzione della pagina contentente le funzionalità disponibili per l'Utente Corriere
MenuProductManager.jsp	Jsp per la costruzione della pagina contentente le funzionalità disponibili per l'Utente Product Manager
MenuResponsabilePersonale.jsp	Jsp per la costruzione della pagina contentente le funzionalità disponibili per l'Utente Responsabile del Personale
ModificaProdTipo.jsp	Jsp per la costruzione della pagina per poter modificare i prodotti
Offerte.jsp	Jsp per la costruzione della pagina per poter visualizzare i i prodotti scontati
Prodotti.jsp	Jsp per la costruzione della pagina per poter visualizzare le categorie di prodotti
SetOfferte.jsp	Jsp per la costruzione della pagina per poter inserire sconti sui prodotti
Register.jsp	Jsp per la costruzione della pagina per la registrazione di un visitatore come Utente Consumatore

2.3.Package Control



Classe	Descrizione
CheckMailServlet.java	Classe che verifica la disponibilità di un'email all'interno del database
LogoutServlet.java	Classe che effettua il logout di un utente loggato nel sistema
LoginServlet.java	Classe che effettua il login di un utente nel sistema
sha256.java	Classe che effettua il checksum di una password
ModificaDatiUtenteServet.java	Classe che permette la modifica dei dati personali di un utente registrato
ProductControl.java	Classe che consente la visualizzazione dei prodotti di una determinata categoria e genera il carrello nel caso in cui non fosse stato creato in precedenza
RegisterServlet.java	Classe che permette la registrazione di un utente non registrato nel sistema
ProdottiOutputServlet.java	Classe che permette il redirect a più servlet per gestire i prodotti presenti nel sistema in base all'utente
ProductControlCarrello.java	Classe che permette la gestione del carrello di un utente consumatore
DipendentiServlet.java	Classe che permette la restituzione e il licenziamento dei corrieri
StampaMcDrives.java	Classe che permette la restituzione dei McDrives
ConsegnaServlet.java	Classe che gestisce la presa in carico di un ordine da parte di un Utente Corriere

2.4.Package Bean



Classe	Descrizione
UtenteBean.java	Classe descrivente un utente registrato nel sistema
Cart.java	Classe descrivente il carrello
ProductBean.java	Classe descrivente un prodotto nel sistema
OrdineBean.java	Classe descrivente un ordine nel sistema
McDriveBean.java	Classe descrivente un McDrive nel sistema

3. CLASS INTERFACES

3.1.UtenteDao.java

Nome Metodo	Context UtenteDao::doSaveUtente(nome, cognome, via, numeroCivico, cap, pwd1, pwd2, email, tipo)
Pre-Condizione	Pre: nome!= null && cognome != null && via != null && numeroCivico != null && cap != null && pwd1 != null && pwd2 != null && email != null && tipo != null && email.contains(".") && email.contains("@") && nome.lenght >=2 && nome.lenght <=20 && cognome.lenght >=2 && cognome.lenght <=20 && pwd1.lenght >=8 && pwd1.lenght <=30 && pwd1 contiene almeno un carattere minuscolo && pwd1 contiene almeno un carattere maiuscolo && pwd1 contiene almeno un carattere numerico

	&& pwd1 == pwd2 && numeroCivico.lenght >=1 && numeroCivico.lenght <=5 && numeroCivico è composto da soli caratteri numerici && cap.lenght ==5 && via.lenght >=2 && via.lenght <=50 && tipo == 0 || tipo == 1 && !self.utenti->exists(u|u.mail.equals(email))
Post-Condizione	Post: self.utenti->exists(u|u.mail.equals(email) && u.passw.equals(pwd1) && u.nome.equals(nome) && u.cognome.equals(cognome) && u.via.equals(via) && u.cap.equals(cap) && u.numeroCivico.equals(numeroCivico) && u.tipo.equals(tipo))
Nome Metodo	Context UtenteDao:: utentedoUpdate (nome, cognome, via, numeroCivico, cap, email)
Pre-Condizione	Pre: nome!= null && cognome != null && via != null && numeroCivico != null && cap != null && email != null && tipo != null && email.contains(".") && email.contains("@") && nome.lenght >=2 && nome.lenght <=20 && cognome.lenght >=2 && cognome.lenght <=20 && numeroCivico.lenght >=1 && numeroCivico.lenght <=5 && numeroCivico è composto da soli caratteri numerici && cap.lenght ==5 && via.lenght >=2 && via.lenght <=50 && tipo == 0 || tipo == 1 && self.utenti->exists(u|u.mail.equals(email))
Post-Condizione	Post: self.utenti->exists(u|u.mail.equals(email) && u.nome.equals(nome) && u.cognome.equals(cognome) && u.via.equals(via) && u.cap.equals(cap) && u.numeroCivico.equals(numeroCivico))
Nome Metodo	Context UtenteDao:: pwddoUpdate (email, pwd)

Pre-Condizione	Pre: email != null && pwd != null && pwd contiene almeno un carattere minuscolo && pwd contiene almeno un carattere maiuscolo && pwd contiene almeno un carattere numerico && self.utenti->exists(u|u.mail.equals(email))
Post-Condizione	Post: self.utenti->exists(u|u.mail.equals(email) &&(u.passw.equls(pwd)))

3.2.CorriereDao.java

Nome Metodo	Context CorriereDao::doUpdateOrdine(ordine)
Pre-Condizione	Pre: ordine != null && ordine.presoInCarico == 0 self.ordini->exists(o|o.data.equals(ordine.data) && (o.numero== ordine.numero))
Post-Condizione	Post: ordine.presoInCarico == 1

3.3.ProductManagerDao.java

Nome metodo	Context ProductManagerDao:: doUpdate(idProdotto, nome, prezzo, foto)
Pre-condizione	Pre: nome != null && prezzo != null && prezzo > 0 && prezzo <= 999 && nome.lenght >=2 && nome.lenght <=40 && self.prodotti->exists(p|p.idProdotto == idProdotto)
Post-condizione	Post: self.prodotti->exists(p|p.idProdotto == idProdotto && p.nome.equals(nome) && p.prezzo == prezzo) && p.foto.equals(foto))
Nome metodo	Context ProductManagerDao:: doSave(nome, prezzo, foto)
Pre-condizione	Pre: nome != null && prezzo != null && prezzo > 0 && prezzo <= 999 && nome.lenght >=2 && nome.lenght <=40
Post-condizione	Post:

	self.prodotti->exists(p|p.nome.equals(nome) && p.prezzo == prezzo && p.foto.equals(foto))
Nome metodo	Context ProductManagerDao:: doDelete(idProdotto)
Pre-condizione	Pre: self.prodotti->exists(p|p.idProdotto = idProdotto)
Post-condizione	Post: !self.prodotti->exists(p|p.idProdotto.equals(idProdotto))
Nome metodo	Context ProductManagerDao:: scontidoUpdate(idProdotto, percentualeSconto)
Pre-condizione	Pre: percentualeSconto >= 0 && percentualeSconto < 100 && self.prodotti->exists(p|p.idProdotto.equals(idProdotto))
Post-condizione	Post: self.prodotti->exists(p|p.idProdotto.equals(idProdotto) && p.idProdotto.sconto == percentualeSconto))

3.4.ProdottoDao.java

Nome metodo	Context ProdottoDao::addOrdine(mail, carrello, mcdrive, totale)
Pre-Condizione	Pre: carrello!=null && mail!=null && mcdrive!=null && totale > 0 && selft.utenti->exists(u|u.mail == mail) && selft.mcdrives->exists(m|m.nome == mcdrive)
Post-Condizione	Post: self.ordini->exists((o|o.mail == mail) && (o.mcdrive == mcdrive) && (o.totale == totale))

3.5.ResponsabilePersonaleDao.java

Nome metodo	Context ResponsabilePersonaleDao::corrieredoDelete (mailCorriereLicenziato)
Pre-Condizione	Pre: mailCorriereLicenziato != null && self.corrieri->exists(c|c.mail.equals(mailCorriereLicenziato))
Post-Condizione	Post: !self.corrieri->exists(c|c.mail.equals(mailCorriereLicenziato))