UNIVERSITÀ DEGLI STUDI
DI SALERNO

Object Design Document

Partecipanti		
Nome	**Cognome**	**Matricola**
Giulio	Di Maria	03518
Matteo	Volpe	03656

Indice degli argomenti

Sommario

Introduzione

Object design trade-offs

Comprensibilità vs Costi

Si preferisce aggiungere costi relativi alle ore/uomo dedicate per la documentazione, al fine di rendere il codice comprensibile sia alle persone non coinvolte nel progetto che alle persone coinvolte che non hanno lavorato ad una particolare sezione del sistema. Saranno introdotti commenti nel codice, per facilitare la comprensione e la manutenzione.

Delivery Time VS Funzinalità

Per rispettare i tempi di consegna, si è deciso di sviluppare soltanto le funzionalità principali, evitando funzionalità che potranno essere aggiunte poi successivamente in futuri aggiornamenti, dove non saranno richiesti tempi da rispettare.

Linee Guida per la documentazione dell'interfaccia

Organizzazione dei file
I file verranno divisi in tre cartelle principali: interface, application e storage.
I nomi dei file saranno in italiano.
I file con estensione .js verranno inseriti in una cartella a parte.
I file con estensione .css verranno inseriti in una cartella a parte.

Indentazione
L'indentazione sarà fatta con una tabulazione. Le linee di codice non conterranno più di 80 caratteri: se supereranno gli 80 caratteri verranno spezzate in modo leggibile:

- La linea verrà interrotta dopo la "," o dopo il ";";
- La linea verrà interrotta prima di un operatore;
- La nuova riga con l'espressione verrà allineata allo stesso livello della linea precedente.
- Ogni riga conterrà al massimo una espressione;
- La linea verrà interrotta in corrispondenza di parentesi.

Di seguito un esempio di indentazione:
```
metodo (espressione1,
        espressione2, espressione3);
```

Parentesi
Per la chiarezza del codice verranno utilizzate più coppie di parentesi nel codice.

Di seguito un esempio di utilizzo delle parentesi:
```
tipoVariabile variabile = (var1-(var2*var3));
```

Commenti

verrà utilizzato lo standard Javadoc per agevolare la chiarezza del codice. Il blocco per la documentazione in Javadoc inizierà con i caratteri "/**" e finirà con i caratteri " */". Ogni riga del blocco inizierà con il carattere " * ".
Se è necessario commentare una sola riga di codice, verrà utilizzato il commento a singola linea (//). Nel caso si debba commentare un blocco di codice, verrà utilizzato /*…*/.
Nei commenti e nella documentazione in Javadoc, verranno utilizzati termini in italiano.

Di seguito un esempio di utilizzo di un commento di un blocco di codice:

```
/*commento*/
If (test) {
    …
}
```

Di seguito un esempio di utilizzo di un commento a una sola riga di codice:

```
var1 = 0; //commento
```

Di seguito un esempio di utilizzo di documentazione in Javadoc:

```
/**
*commento
*@param {tipo parametro} nome parametro [Descrizione parametro]
*@returns {tipo ritorno} [Descrizione ritorno]
** /
```

Istruzioni

Le istruzioni semplici saranno disposte una per riga.
Le istruzioni composte, alla riga successiva alla prima riga dell'istruzione, ci sarà una parentesi graffa aperta.

Di seguito un esempio di istruzione semplice:

```
var nome Variabile = valore;
```

Di seguito un esempio di istruzione composta:

```
if (test) {
    …
} else {
    …
}
```

Metodi

I nomi dei metodi:

- Saranno dei verbi;
- Saranno in italiano e in alcuni casi i prefissi saranno in inglese;
- L'iniziale sarà in minuscolo e ciascuna parola che compone il nome, avrà l'iniziale in maiuscolo.

La dichiarazione del metodo verrà fatta in questo modo:

- La parentesi graffa aperta verrà posta alla fine della prima riga della dichiarazione senza spazio;
- Il corpo del metodo sarà indentato;
- L'ultima riga della dichiarazione sarà la parentesi graffa chiusa

Di seguito un esempio di nome del metodo:

getNome();

Di seguito un esempio di dichiarazione del metodo:
public String getNome(){
…
}

Variabili
Le variabili:

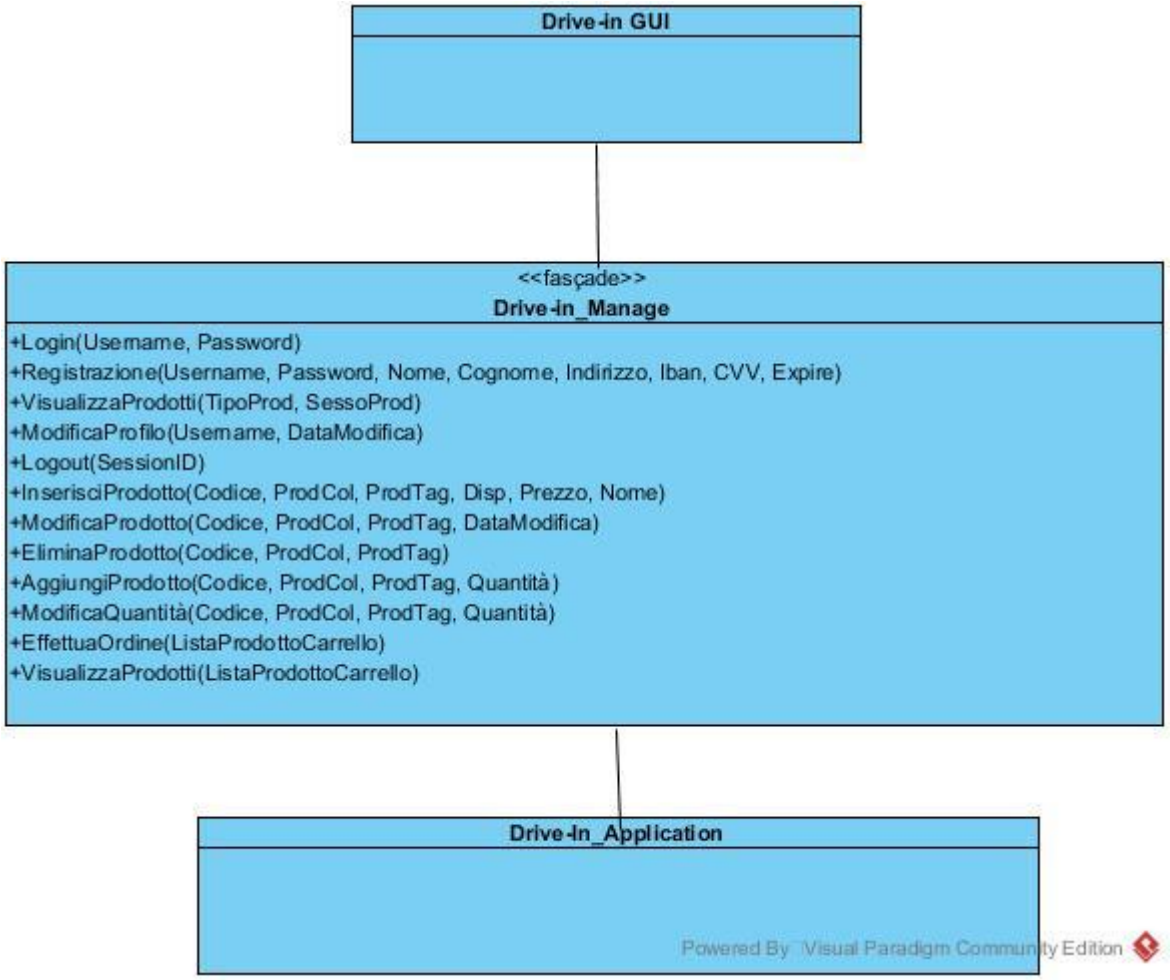
- Verranno dichiarate all'inizio di un blocco;
- Verranno dichiarate una per riga;
- Verranno inizializzate nel punto in cui verranno dichiarate, impostando il valore di default o il risultato di un metodo.

La dichiarazione delle variabili verrà allineata per renderle più leggibili e verranno utilizzati gli spazi per separare i diversi elementi in una espressione.

Di seguito un esempio di dichiarazione di una variabile:
int numero = 0;

Design Pattern

Façade Patten



Façade permette agli utenti di TS di interagire con il sistema tramite l'interfaccia e di quindi fare uso di tutte le funzionalità che la piattaforma offre.

Definizioni, acronimi e abbreviazioni

1.1.1 **DBMS**: Database Management System.

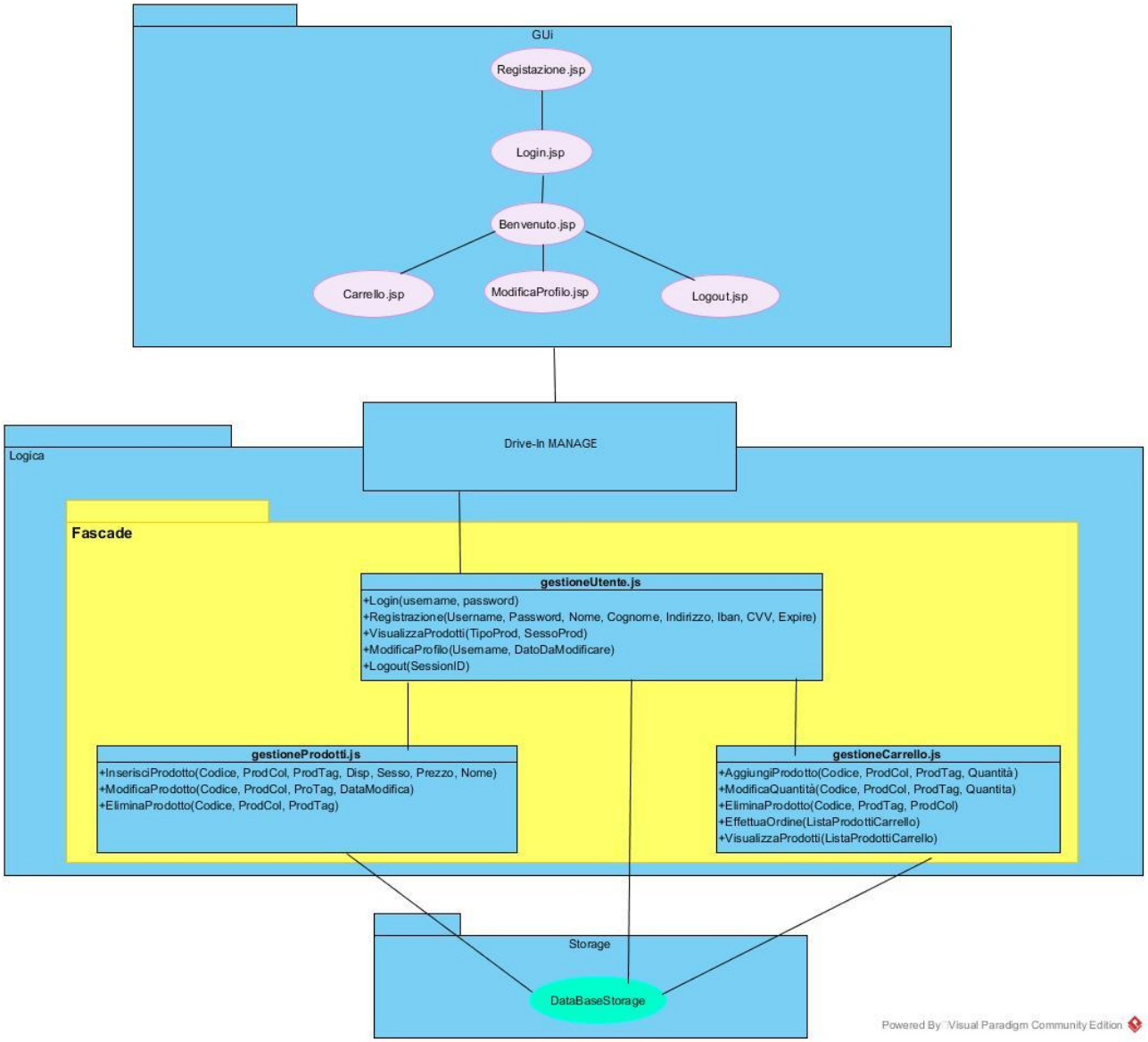
1.1.2 **HTML**: Linguaggio di mark-up per pagine web.

1.1.3 **JavaScript**: linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso.

Riferimenti

- Drive-In_SDD_V2.0
- Bernd Bruegge & Allen H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns and Java*, (3rd edition), Prentice Hall

Packages


GUi
Registazione.jsp
Login.jsp
Benvenuto.jsp
Carrello.jsp
ModificaProfilo.jsp
Logout.jsp
Drive-In MANAGE
Logica
Fascade
gestioneUtente.js
+Login(username, password)
+Registrazione(Username, Password, Nome, Cognome, Indirizzo, Iban, CVV, Expire)
+VisualizzaProdotti(TipoProd, SessoProd)
+ModificaProfilo(Username, DatoDaModificare)
+Logout(SessionID)
gestioneProdotti.js
+InserisciProdotto(Codice, ProdCol, ProdTag, Disp, Sesso, Prezzo, Nome)
+ModificaProdotto(Codice, ProdCol, ProTag, DataModifica)
+EliminaProdotto(Codice, ProdCol, ProdTag)
gestioneCarrello.js
+AggiungiProdotto(Codice, ProdCol, ProdTag, Quantità)
+ModificaQuantità(Codice, ProdCol, ProdTag, Quantita)
+EliminaProdotto(Codice, ProdTag, ProdCol)
+EffettuaOrdine(ListaProdottiCarrello)
+VisualizzaProdotti(ListaProdottiCarrello)
Storage
DataBaseStorage
Powered By Visual Paradigm Community Edition

Interfacce

Nome classe	GestioneUtente
Descrizione	Rappresenta il gestore delle funzionalità dedicate alla creazione (Registrazione) e accesso(Login) al programma.
Pre-condizione	• **context** GestioneUtente:login(Username,Password): **pre:** Username!=null && Password!=null • **context** GestioneUtente: Regstrazione(Username, Password,Nome,Cognome,Indirizzo,Iban,CVV,Expire): **pre:((**Username, Password,Nome,Cognome,Indirizzo,Iban,CVV,Expire) != null) • **context** GestioneUtente: VisualizzaProdotti (TipoProd,SessoProd): **pre:** TipoProd != null, SessoProd != null • **context** GestioneUtente:ModificaProfilo(Username, DataModifica): **pre:** Username != null , DataModifica != null • **context** GestioneUtente: Logout (SessionID): **pre:** SessionID != null
Post-condizione	
Invarianti	

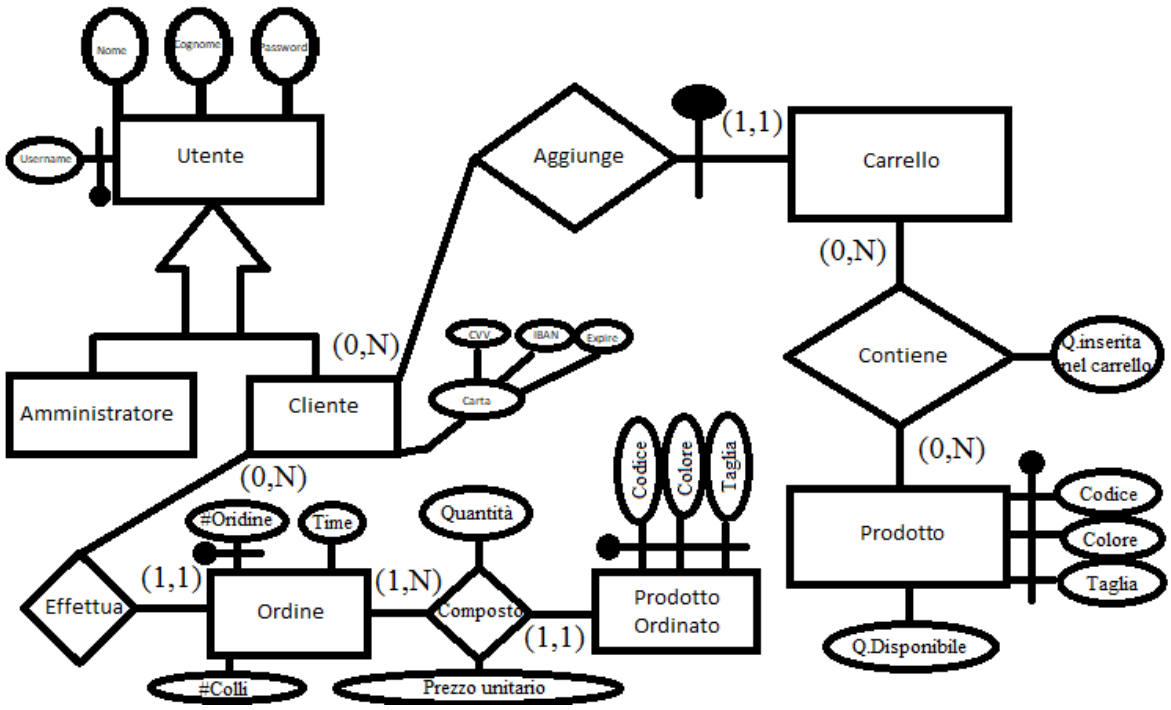
Nome classe	GestioneProdotti
Descrizione	Rappresenta il gestore delle funzionalità gestite dall Aministratore dei Prodotti presenti all'interno del catalogo
Pre-condizione	• **context** GestioneProdotti::InserisciProdotto(Codice, ProdCol, ProdTag,Disp, Sesso, Prezzo, Nome) **pre:** ((Codice, ProdCol, ProdTag,Disp, Sesso, Prezzo, Nome) != null) • **context** GestioneProdotti: ModificaProdotto (Codice, ProdCol, ProdTag): **pre**: ((Codice, ProdCol, ProdTag !=null))

	• **context** GestioneProdotti: EliminaProdotto(Codice, ProdCol, ProdTag): **pre: ((**Codice, ProdCol, ProdTag) != null)
Post-condizione	
Invarianti	

Nome classe	**GestioneCarrello**
Descrizione	Rappresenta il gestore delle funzionalità gestite dall Aministratore dei Prodotti presenti all'interno del catalogo
Pre-condizione	• **context** GestioneCarrello:: AggiungiProdotto(Codice, ProdCol, ProdTag, Qunatità): **pre: ((**Codice, ProdCol, ProdTag, Quantità= != null) • **context** GestioneCarrello:: ModificaQuantità(Codice, ProdCol, ProdTag, Quantità) : **pre: ((**Codice, ProdCol, ProdTag, Quantità) != null) • **context** GestioneCarrello::EliminaProdotto(Codice, ProdCol, ProdTag) : **pre: ((**Codice, ProdCol, ProdTag)!= null) • **context**GestioneCarrello ::EffettuaOrdine(ListaProdottiCarrello) : : **pre : (**ListaProdottiCarrello != null) • **context**GestioneCarrello :: VisualizzaProdotti (ListaProdottiCarrello): **pre :** ListaProdottiCarrello != null
Post-condizione	
Invarianti	

Entità

Schema ER



Dizionario dei dati

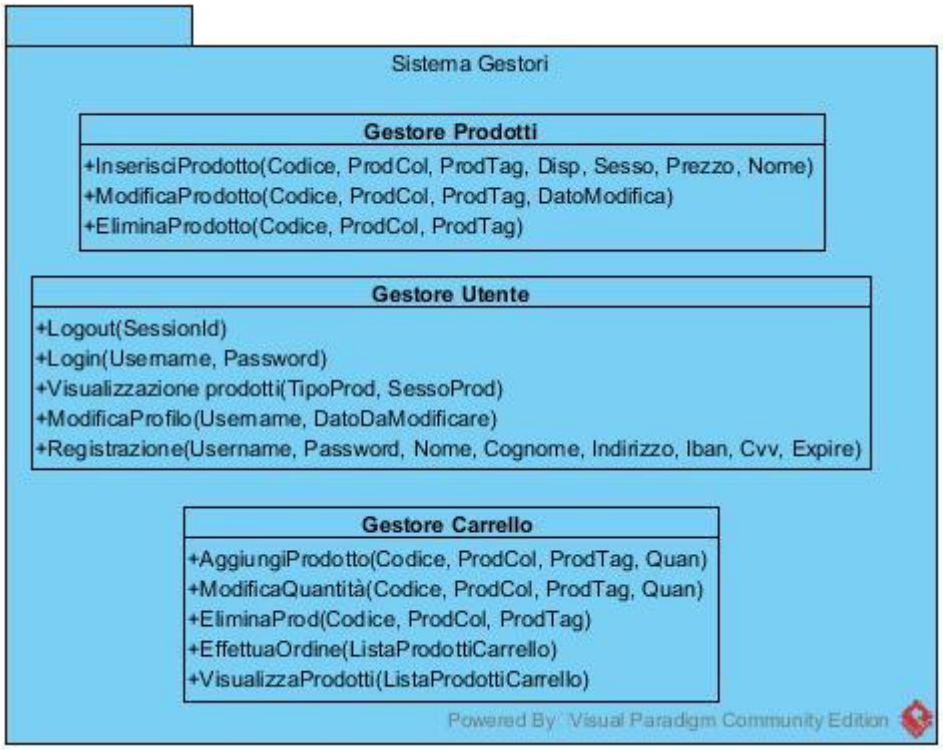
Entità	Descrizione	Attributi	Identificatore
Utente	Elenco degli utenti registrati sul sistema	Username,Nome, Cognome, Password	Username
Amministratore	Elenco degli utenti che sono amministratori	Username,Nome, Cognome, Password	Username
Cliente	Elenco degli utenti che sono clienti	Username,Nome, Cognome, Password, Carta (Cvv, Iban, Expire)	Username
Carrello	Elenco dei carrelli all'interno del sistema		Username (identificatore esterno

			realizzato dall'entità utente)
Prodotto	Elenco dei prodotti	Codice,Colore,Taglia,Q.Disponibile	Codice,Colore,Taglia
Ordine	Elenco degli ordini realizzati e registrati	#Ordine, Time, #Colli	#Ordine
Prodotto Ordinato	Elenco dei prodotti ordinati	Codice,Colore,Taglia	Codice,Colore,Taglia

Tavola dei volumi

Concetto	Costrutto	Volume
Utente	Entità	2000
Amministratore	Entità	20
Cliente	Entità	2000
Carrello	Entità	2000
Prodotto	Entità	200
Ordine	Entità	500
Prodotto Ordinato	Entità	2000
Aggiunge	Relazione	10000
Contiene	Relazione	500
Effettua	Relazione	200
Composto	Relazione	800

Gestori



Nome	Gestore Prodotti
Descrizione	Questa classe rappresenta il gestore dei prodotti (o amministratore dei prodotti) e i vari metodi che quest'ultimo può sfruttare
Signature dei metodi	EliminaProdotto(Codice, ProdCol, ProdTag) InserisciProdotto(Codice, ProdCol, ProdTag, Disp, Sesso, Prezzo, Nome) ModificaProdotto(Codice, ProdCol, ProdTag, DatoModifica)

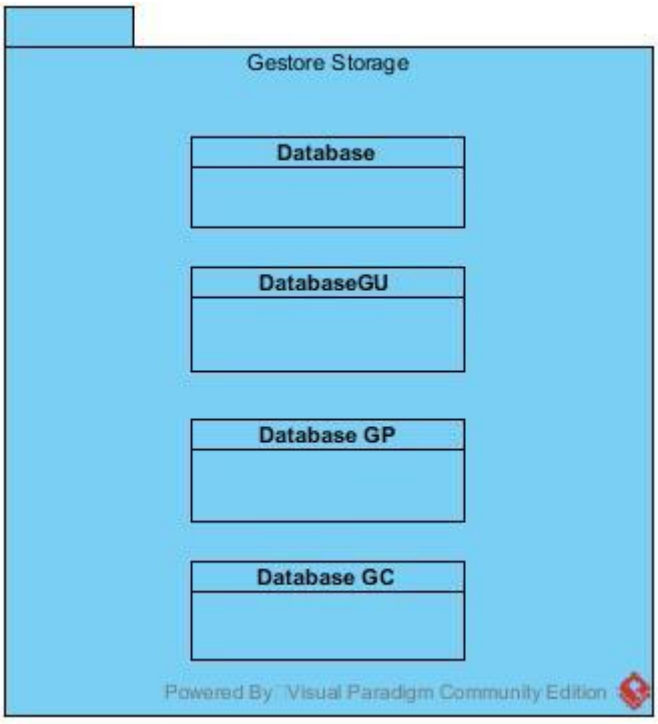
Nome	Gestore Carrello
Descrizione	Questa classe rappresenta il gestore del carrello(o cliente), con i vari metodi che quest'ultimo può utilizzare.
Signature dei metodi	AggiungiProdotto(Codice, ProdCol, ProdTag, Quan) ModificaQuantità(Codice, ProdCol, ProdTag, Quan)

	EliminaProd(Codice, ProdCol, ProdTag) EffettuaOrdine(ListaProdottiCarrello) VisualizzaProdotti(ListaProdottiCarrello)

Nome	Gestore Utente
Descrizione	Questa classe rappresenta il gestore degli utenti con i vari metodi che quest'ultimo può utilizzare.
Signature dei metodi	Logout(SessionId) Login(Username, Password) Visualizzazione prodotti(TipoProd, SessoProd) ModificaProfilo(Username, DatoDaModificare) Registrazione(Username, Password, Nome, Cognome, Indirizzo, Iban, Cvv, Expire)

Storage

Il package storage è formto da quattro classi principali che si occuperanno di interfacciarsi al database e di passare le varie query.



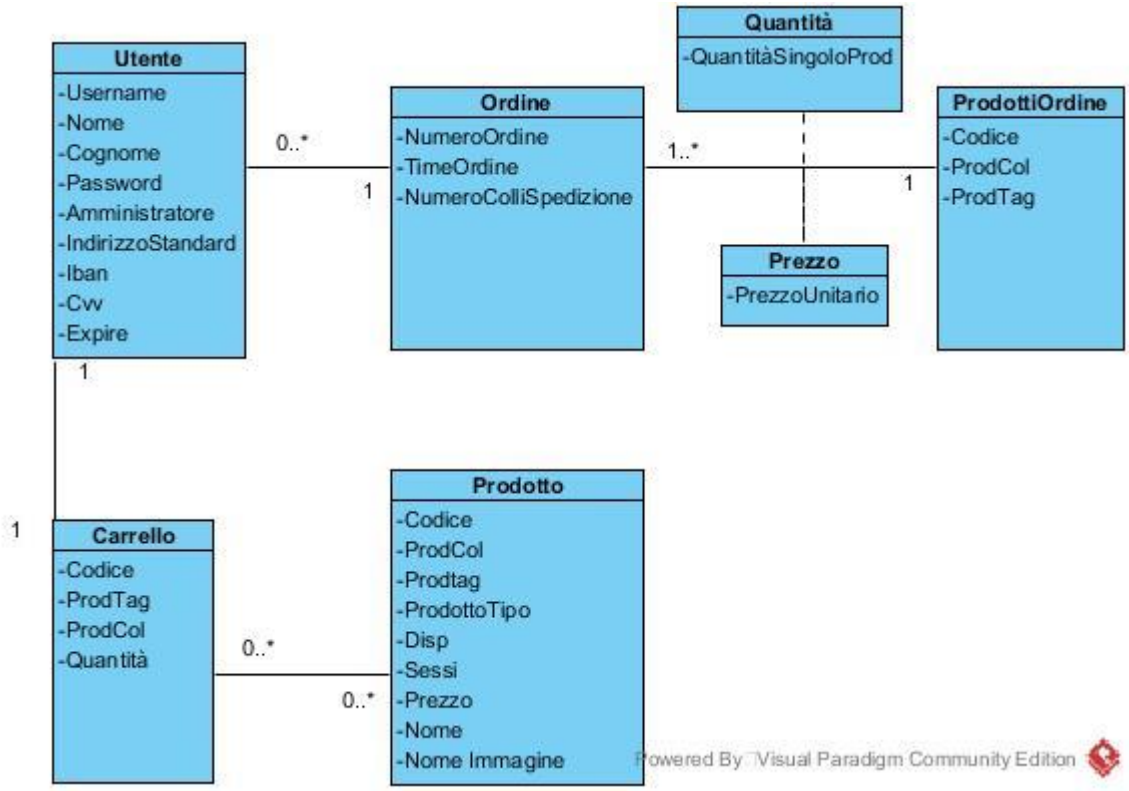
Database – Rappresenta il pool di connessioni con il database.

DatabaseGU – Rappresenta il gestore del database per "Gestione Utente"

DatabaseGP – Rappresenta il gestore del database per "Gestione Prodotti

DatabaseGC – Rappresenta il gestore del database per "Gestoione Carrello"

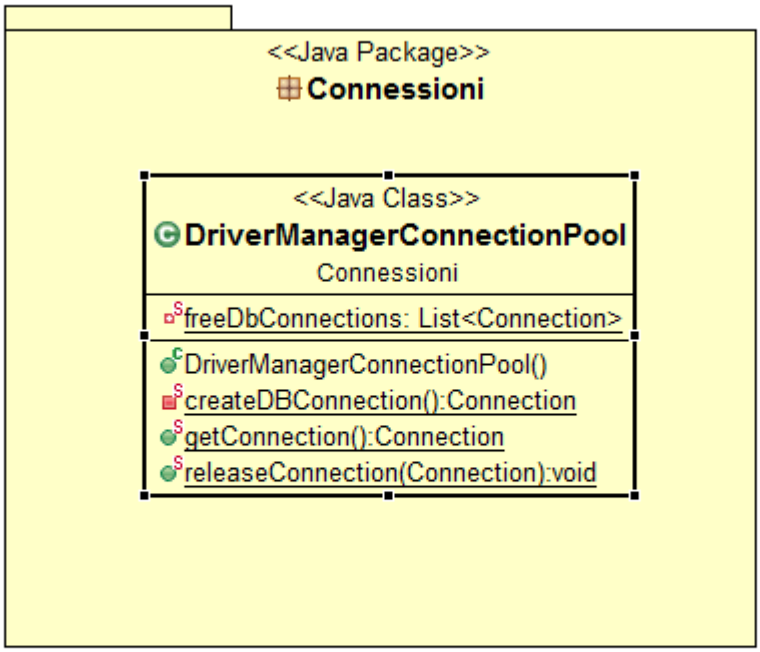
Modello logico



Interfaccia delle classi

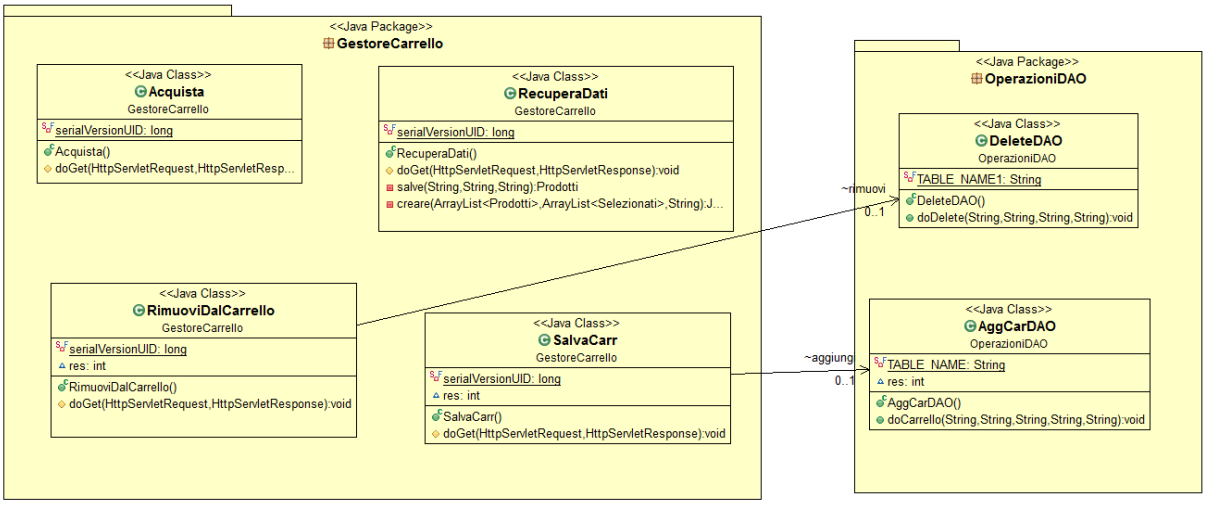
I seguenti diagrammi dei packages sono stati realizzati utilizzando ObjectAID, un plugin del framework di Eclipse, dopo aver completato lo sviluppo.

Connessioni

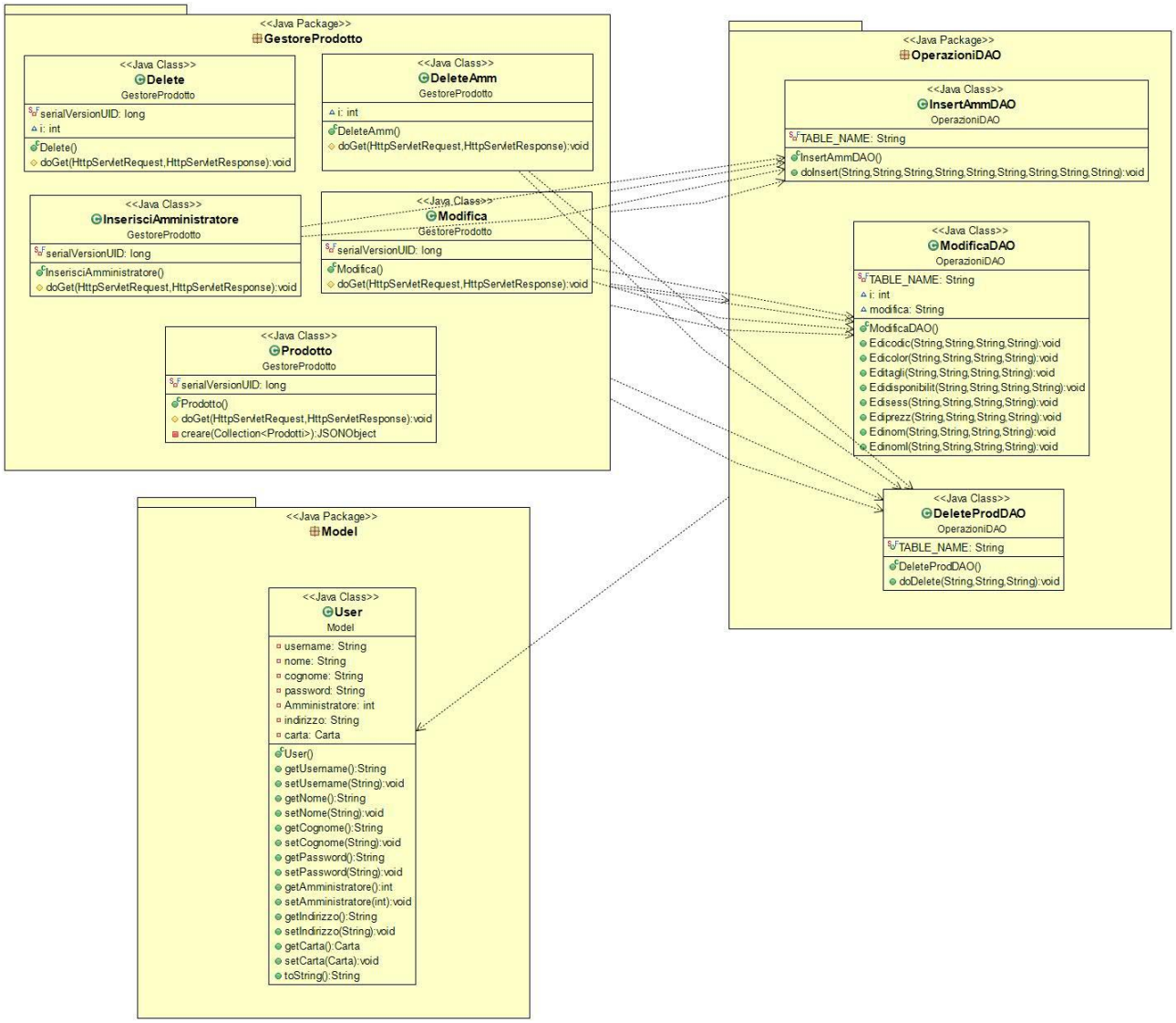


Classe per la gestione del pool di connessioni sulla quale si poggiano tutte le classi DAO per ottenere una connessione al database.

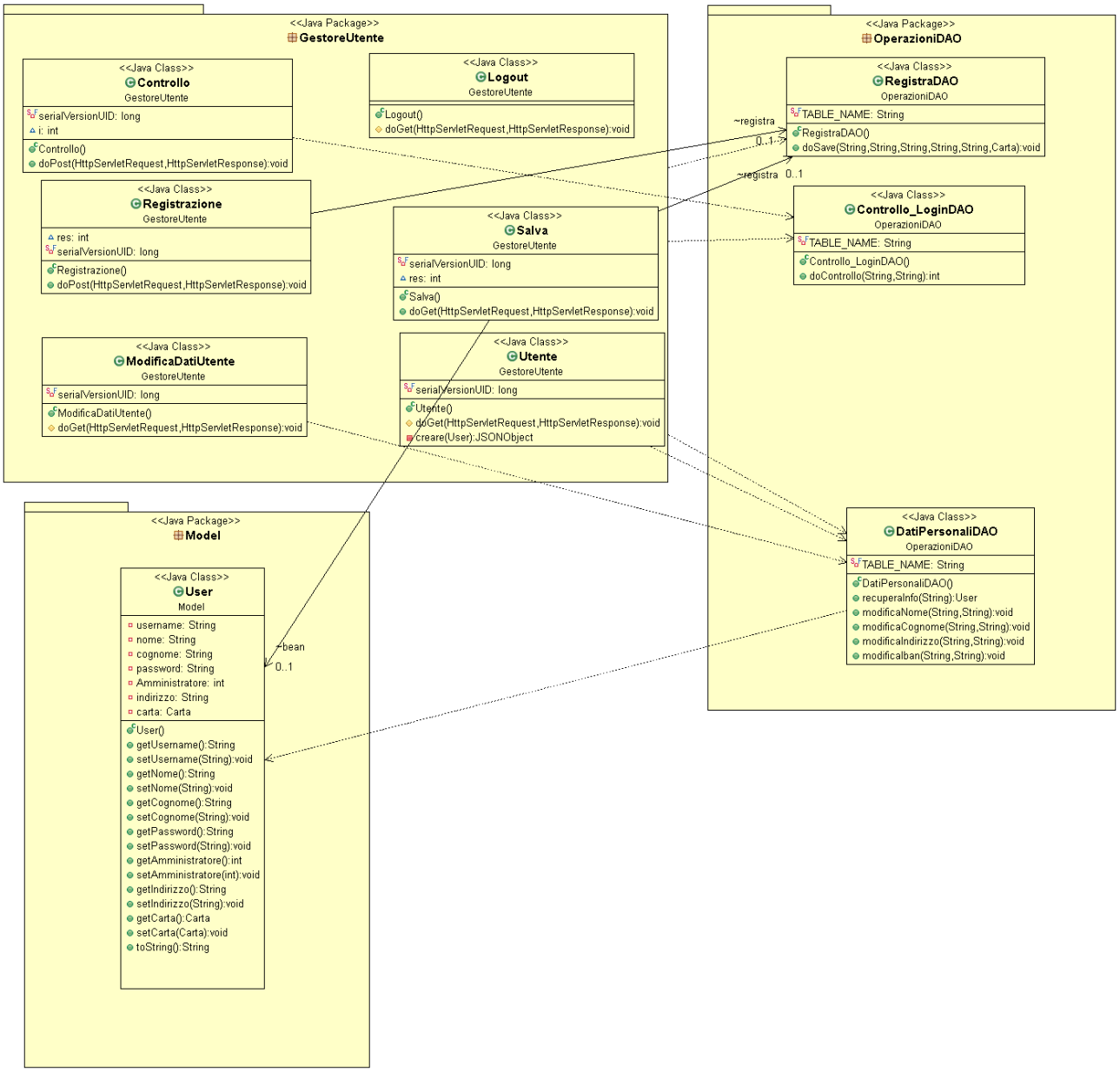
Gestore Carrello



Gestore Prodotti



Gestore Utente



Gestore Carrello