



**CORSO DI INGEGNERIA DEL SOFTWARE PROF. A. DE LUCIA**

**PROGETTO EMC**

**OBJECT DESIGN DOCUMENT**

**2020/2021**

|  |  |
| --- | --- |
| PARTECIPANTI | MATRICOLA |
| ALESSANDRA POTESTA’ | **06188** |
| ROSARIO ANNUNZIATA | **05810** |
| GIOVANNI TAVOLO | **05912** |

Sommario

[1 – Introduzione 4](#_Toc80020232)

[1.1 Object design trade offs 4](#_Toc80020233)

[1.2 Componenti off-the-shelf 4](#_Toc80020234)

[1.3 Linee guida 5](#_Toc80020235)

[1.4 Riferimenti 9](#_Toc80020236)

[2 - Packages 10](#_Toc80020237)

[View 10](#_Toc80020238)

[Model 12](#_Toc80020239)

[Servlets 12](#_Toc80020240)

[Test 14](#_Toc80020241)

[3 - Interfacce di classe 15](#_Toc80020242)

[4 - Diagrammi 20](#_Toc80020243)

# **1 – Introduzione**

# **Object design trade offs**

**Comprensibilità vs Tempo:**

Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare.

**Costi vs Estensibilità**:

Il rispetto dei costi stabiliti prevarrà sull’estensibilità. Quindi, al fine di rispettare i tempi di rilascio, probabilmente non saranno presenti nella prima release le funzionalità di sistema associate a priorità più basse.

**Interfaccia vs Usabilità:**

Verrà realizzata un’interfaccia chiara e user friendly, usando form e pulsanti predefiniti che hanno lo scopo di rendere semplice l’utilizzo del sistema da parte dell’utente finale.

**Sicurezza vs Efficienza:**

La sicurezza per via di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su filtri, e password criptate in SHA-56.

# **Componenti off-the-shelf**

Per il progetto software che si intende realizzare ci serviremo di diversi componenti off-the-shelf ovvero componenti software già sviluppate, ottimizzate, pronte all’uso. Nello specifico, useremo diverse tecnologie, sia il lato back-end e il lato front-end, che ci permetteranno di implementare al meglio la nostra WebApp. In particolar modo:

* jQuery, libreria JavaScript che facilita la scrittura di scrip rendendo semplice la selezione e la manipolazione di elementi del DOM in pagine HTML.
* Sarà utilizzato AJAX come strumento di sviluppo software, per gestire i messaggi asincroni all’interno delle pagine. Consente l’aggiornamento dinamico di una pagina web senza caricamento esplicito da parte dell’utente. Verrà utilizzato insieme all’estensione JSON.
* JSON, formato di dati adatto allo scambio di informazioni in applicativi client/server. Usato in AJAX tramite l’API XHRHttp
* Come web server la scelta, invece, è ricaduta su Tomcat
* Il linguaggio di programmazione principalmente utilizzato sarà Java. Tramite JSP (JavaServerPages) e Servlet sarà possibile far comunicare il lato back-end e il lato front-end, permettendo così di avere una pagina dinamica e interattiva
* Selenium, una suite di tool utilizzati per automatizzare i test di sistema eseguendoli sul web browser.
* JUnit, un framework di programmazione Java che viene utilizzato per implementare itest di unità

Tutte le componenti selezionate ed utilizzate sono gratuite ed open source e rispettano i requisiti di costo

# **Linee guida**

Nella fase di implementazione del sistema, gli sviluppatori si dovranno attenere alle linee guida descritte nel seguente paragrafo:

**Package**

Il progetto verrà sviluppato con L’IDE IntellijIDEA e sarà strutturato come segue:

Il progetto avrà 4 package: model, servlet, test e jsp, i quali contengono i corrispettivi sub-package con le rispettive classi.

**Classi Java**

Il linguaggio Java sarà parte del cuore del sistema, infatti verrà utilizzato per modellarne il comportamento a seguito delle operazioni effettuate dall’utente. La versione di riferimento sarà Java Standard Edition (JavaSE). Il codice Java dovrà seguire i seguenti punti:

* È buona norma utilizzare nomi che siano:
  + Descrittivi
  + Non troppo lunghi
  + Non abbreviati
  + Pronunciabili
* I nomi delle variabili devono essere scritti secondo il Camel Case: devono iniziare con lettera minuscola e le parole seguenti con la lettera maiuscola (per identificare appunto l’inizio di una nuova parola), es. myArray. Le variabili dovranno essere definite all’inizio del blocco di codice.
* Le variabili costanti seguiranno invece la notazione Macro Case: devono utilizzare soltanto lettere maiuscole, separate dal trattino basso: es. ARRAY\_SIZE.
* Anche i metodi devono essere scritti secondo il Camel Case. Questi, in genere, sono formati da verbo più nome oggetto: il verbo identifica l’azione da compiere sull’oggetto, es. GetUsername.
* Il codice deve essere provvisto di commenti per facilitarne la lettura e la comprensione. Questi dovranno descrivere la funzionalità oggetto
* I nomi delle classi e delle pagine devono invece essere scritti secondo il Capital Camel Case: devono iniziare con lettera maiuscola, così come le parole che seguiranno, es. ServletLogin.java.
* I nomi dei package devono essere scritti in Lower Case: devono utilizzare soltanto lettere minuscole: es. account.

**Pagine JSP**

I nomi delle pagine JSP dovranno seguire la notazione “camelCase” descritta per i metodi delle classi Java.

Le pagine JSP quando eseguite dovranno produrre un documento conforme allo standard HTML5. Il codice java presente nelle JSP deve aderire alle convenzioni descritte precedentemente.

● Il tag di apertura (<%) dovrà essere seguito da un invio a capo;

● Il tag di chiusura (%>) dovrà trovarsi all’inizio della riga;

● Il codice tra i tag dovrà essere indentato;

● Nel caso di singola istruzione le tre regole precedenti possono essere evitate;



**Pagine HTML**

Il codice HTML sarà presente all’interno delle View del sistema per modellare la struttura dell’interfaccia grafica. La versione di riferimento che verrà utilizzata è la versione 5. Ogni blocco di codice HTML dovrà seguire i seguenti punti:

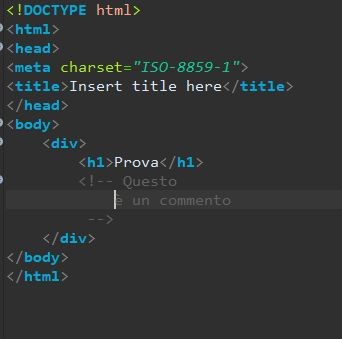
• Ogni tag di apertura deve essere necessariamente seguito dall’apposito tag di chiusura, eccezione

fatta per i tag self-closing (es. <hr>, <br/>, <img>, …).

• Il blocco di codice dev’essere opportunamente indentato.

• L’indentazione del codice deve avvenire tramite tabulazioni (tasto TAB) e non tramite i classici

spazi bianchi (tasto BARRA DI SPAZIATURA).

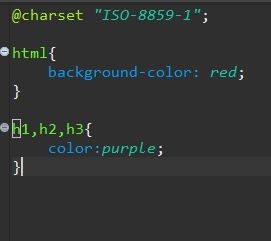
• Il codice dev’essere tutto scritto in lowercase, es. <hr> e non <HR>.• I tag <script> devono essere posizionati alla fine del file (in genere questi vanno posizionati prima del tag di chiusura </body>).

**Script**

Gli script dovranno essere scritti in JavaScript o in JQuery, dovranno essere ben indentati, di facile lettura. I nomi dei file degli script dovranno seguire la notazione “camelCase”

**Fogli di stile CSS**

I fogli di stile dovranno essere formattati come segue:

* I selettori della regola dovranno trovarsi sulla stessa riga;
* L’ultimo selettore della regola è seguito dalla parentesi graffa aperta “{“;
* Le proprietà che costituiscono la regola saranno una per riga e sono indentate rispetto ai selettori;
* La regola è terminata da una graffa chiusa “}” collocata da sola su una sola riga;

**Database SQL**

I costrutti sql devono essere scritti con sole lettere maiuscole

I nomi delle tabelle devono essere costituiti da solo lettere minuscole, le parole devono essere separate dal carattere underscore “\_”, come nel caso di tabelle generate da relazioni N-M.

I nomi devono appartenere al dominio del problema ed esplicare correttamente ciò che intendono rappresentare.

I nomi delle colonne delle tabelle devono seguire la notazione “camelCase”, anche loro devono esplicare correttamente la parte del dominio del problema che intendono rappresentare.

**Design Pattern MVC**

Il design pattern MVC consente la suddivisione del sistema in tre blocchi principali: Model, View e Controller. Il Model modella i dati del dominio applicativo e fornisce i metodi di accesso ai dati persistenti, il View si occupa della presentazione dei dati all’utente e di ricevere da quest’ultimo gli input, infine il Controller riceve i comandi dell’utente attraverso il View e modifica lo stato di quest’ultimo e del Model.

**DAO (Data Access Object) Pattern**

Il DAO pattern è utilizzato per il mantenimento di una rigida separazione tra le componenti Model e Controller, in questo tipo di applicazioni basate sul paradigma MVC.

Data-Access Object: classe che implementa i metodi degli oggetti che rappresenta.

Classi Bean: classi che contengono i getters/setters degli oggetti che rappresentano e saranno usati dai DAO

# **Riferimenti**

Documento RadEMC.docx

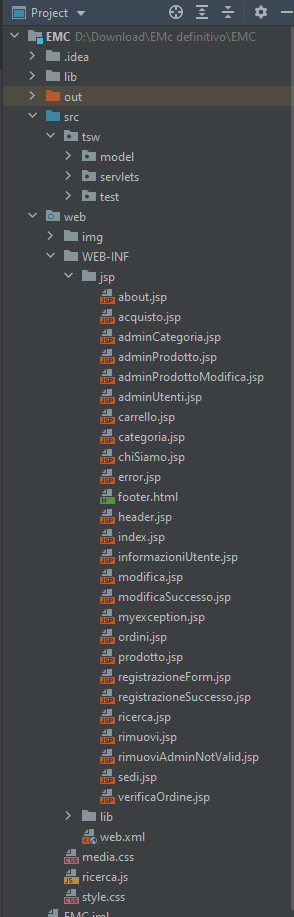
Documento SddEMC.docx

# 2 - Packages

## **View**

Nel folder Web/WEB-INF sono contenute le JSP per la visualizzazione delle pagine.

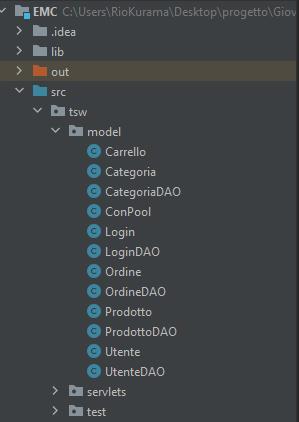
Le JSP sono così suddivise:



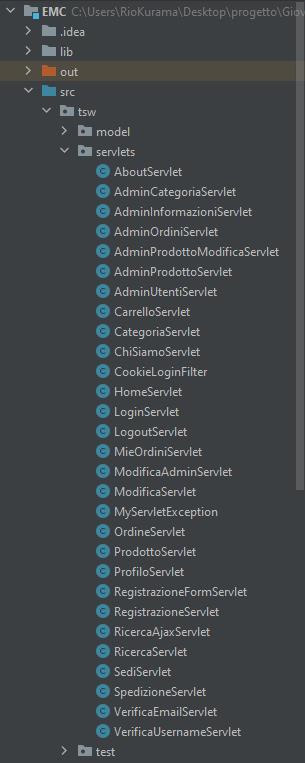
* **about.jsp,** la pagina mostra le informazioni per poter entrare in contatto con l’azienda;
* **acquisto.jsp,** la pagina mostra un form per specificare l’indirizzo di acquisto, nel caso si voglia effettuare un oridine;
* **admincategoria.jsp,** la pagina mostra, a seconda che si tratti dell’operazione di modifica o di aggiunta di un prodotto, i relativi form da compilare per la modifica/aggiunta di una categoria;**adminprodotto.jsp,** la pagina mostra i relativi form da compilare per la aggiunta di una categoria;
* **adminprodottomodifica.jsp,** la pagina mostra i relativi form da compilare per la modifica di una categoria;
* **adminutente.jsp**, la pagina mostra le informazioni relative all’utente con la relativa possibilità di modificare tali informazioni se si è un amministratore;
* **carrello.jsp,** la pagina mostra il carrello, ovvero, i prodotti che si trovano all’interno del carrello;
* **categoria.jsp,** la pagina mostra una categoria, con all’interno, tutti i prodotti di quella categoria;
* **chisiamo.jsp,** la pagina mostra le informazioni di vario genere sull’azienda;
* **error.jsp,** pagina di errore, utile quando si verifica un errrore;
* **header.jsp,** la pagina mostra il menu navigazionale che permette all’utente di spostarsi nella web app;
* **index.jsp,** è la pagina principale della webapp dove compaiono header, footer e lista delle recensioni;
* **informazioniutente.jsp,** la pagina mostra le informazioni relative all’utente che ha effettuato il login
* **modifica.jsp,** è la pagina che permette ad un amministratore di modificare le credenziali di un utente;
* **modificaSuccesso.jsp,** la pagina mostra un messaggio di avviso, con la dicitura; modifica avvenuta con successo;
* **myexception.jsp,** la pagina mostra un’eccezione;
* **ordini.jsp,** la pagina mostra tutti gli ordini effettuati da un determinato utente;
* **registazioneForm.jsp,** è la pagina che permette ad un utente non registrato di registrarsi alla piattaforma inserendo nome, cognome, email, username, password e conferma password;
* **registrazioneSuccesso.jsp,** la pagina mostra un messaggio di avviso, con la dicitura; registrazione avvenuta con successo;
* **ricerca.jsp,** la pagina mostra i prodotti che sono stati ricercati nell’apposito campo di ricerca;
* **rimuovi.jsp**, la pagina mostra un messaggio di avviso, con la dicitura; rimozione avvenuta;
* **rimuoviAdminNotValid.jsp,** la pagina mostra un messaggio di avviso, con la dicitura; Non puoi rimuovere un admin;
* **sedi.jsp,** la pagina mostra informazioni relative alle sedi della società;
* **verificaOrdine.jsp,** la pagina mostra un messaggio di avviso, con la dicitura; Grazie per aver acquistato da noi, le abbiamo fornito un id per il suo ordine! Buona Giornata.

# **Model**

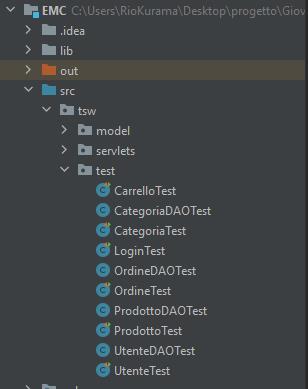
Nel folder tsw/model sono contenuti i Bean e i DAO, e sono così suddivisi:

* **Carrello**, è la classe Bean che rappresenta un carrello
* **Categoria**, è la classe Bean che rappresenta una categoria
* **CategoriaDAO**, è la classe DAO che rappresenta la classe contenente i metodi utili all’interazione con il database per ciò che riguarda la categoria;
* **ConPool**, è la classe che consente la connessione al database.
* **Login**, è la classe Bean che rappresenta il login
* **LoginDAO**, è la classe DAO che rappresenta la classe contenente i metodi utili all’interazione con il database per ciò che riguarda il login;
* **Ordine**, è la classe Bean che rappresenta un ordine
* **OrdineDAO**, è la classe DAO che rappresenta la classe contenente i metodi utili all’interazione con il database per ciò che riguarda l’ordine;
* **Prodotto**, è la classe Bean che rappresenta un prodotto
* **ProdottoDAO**, è la classe DAO che rappresenta la classe contenente i metodi utili all’interazione con il database per ciò che riguarda un prodotto;
* **Utente**, è la classe Bean che rappresenta un utente
* **UtenteDAO**, è la classe DAO che rappresenta la classe contenente i metodi utili all’interazione con il database per ciò che riguarda un utente;

# **Servlets**

* **AboutServlet,** permette a tutti gli attori di visualizzare la pagina di informazioni About;
* **AdminCategoriaServlet**, permette di modificare/aggiungere una categoria;
* **AdminProdottoModificaServlet**, permette di modificare un prodotto;
* **AdminOrdiniServlet**, permette di visualizzare le informazioni di un utente;
* **AdminProdottoServlet**, permette di aggiungere un nuovo prodotto;
* **AdminUtentiServlet**, permette di modificare le informazioni di un utente;
* **CarrelloServlet**, permette a tutti gli attori di visualizzare il carrello;
* **CategoriaServlet**, permette a tutti gli attori di visualizzare le categorie;
* **ChiSiamoServlet**, permette a tutti gli attori di visualizzare la pagina di informazioni Chi Siamo;
* **CookieLoginFilter**, permette di tener traccia di un utente;
* **HomeServlet**, permette di visualizzare la pagina iniziale;
* **LoginServlet**, permette di effettuare il login;
* **LogoutServlet**, permette a tutti gli attori di effettuare il logout;
* **MieOrdiniServlet**, permette ad un utente di poter vedere i propri ordini;
* **ModificaAdminServlet**, permette all’ amministratore di compilare i campi per modificare le informazioni di un utente;
* **ModificaServlet**, permette all’amministratore di rimuovere/modificare un utente;
* **MyServletException**, eccezione che si verifica quando un utente non registrato si trova su pagine a cui non può accedete;
* **OrdineServlet**, permette ad un utente registrato di poter compilare i campi per procedere all’ordine;
* **ProdottoServlet**, permette a tutti gli attori di visualizzare un prodotto;
* **ProfiloServlet**, permette agli utenti registrati di visualizzare il loro profilo;
* **RegistrazioneFormServlet**, permette ad un utente non registrato di compilare i campi di registrazione;
* **RegistrazioneServlet** permette di effettuare la registrazione al sistema;
* **RicercaAjaxServlet**, permette di ricercare un prodotto nel sistema;
* **RicercaServlet**, permette di visualizzare i risultsati della ricerca di un prodotto;
* **SediServlet**, permette di visualizzare la pagina di informazioni Sedi;
* **SpedizioneServlet**, permette di visualizzare la pagina con la conferma dell’acquisto;
* **VerificaEmailServlet**, permette di effettuare la verifica dell’email;
* **VerificaUsernameServlet**, permette di effettuare la verifica dell’username.

# **Test**

* **CarrelloTest,** contiene i test relativi al CarrelloBean;
* **CategoriaDAOTest,** contiene i test relativi alla CategoriaDAO;
* **CategoriaTest,** contiene i test relativi alla CategoriaBean;
* **LoginTest,** contiene i test relativi al LoginBean;
* **OrdineDAOTest,** contiene i test relativi all’OrdineDAO;
* **OrdineTest,** contiene i test relativi all’OrdineBean;
* **ProdottoDAOTest,** contiene i test relativi al ProdottoDAO;
* **ProdottoTest,** contiene i test relativi al ProdottoBean;
* **UtenteDAOTest,** contiene i test relativi all’UtenteDAO;
* **UtenteTest,** contiene i test relativi all’UtenteBean;

# 

# 3 - **Interfacce di classe**

|  |  |
| --- | --- |
| Nome Classe | AdminCategoriaServlet |
| Descrizione | Permette di aggiungere una categoria |
| Pre-condizione: | if ((nome != null && nome.trim().length() > 0))  if ((descrizione != null && descrizione.trim().length() > 0))  if (categoria == null)  if ((categoriaDAO.doRetrieveByNome(nome) != null)) |
| Post-condizione | AdminCategoriaServlet::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | AdminCategoriaServlet |
| Descrizione | Permette di modificare una categoria |
| Pre-condizione: | if ((nome != null && nome.trim().length() > 0))  if ((descrizione != null && descrizione.trim().length() > 0))  if (categoria != null)  if ((categoriaDAO.doRetrieveByNome(nome) != null) && (categoriaDAO.doRetrieveByDescrizione(descrizione) != null)) |
| Post-condizione | AdminCategoriaServlet::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | AdminCategoriaServlet |
| Descrizione | Permette di visualizzare rimuovere una categoria |
| Pre-condizione: | if (request.getParameter("rimuovi") != null) |
| Post-condizione | AdminCategoriaServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | AdminOrdiniServlet |
| Descrizione | Permette di visualizzare le informazioni di un utente |
| Pre-condizione: | if (utente != null) |
| Post-condizione | AdminOrdiniServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | AdminProddottoModificaServlet |
| Descrizione | Permette di modificare un prodotto |
| Pre-condizione: | if (rb == null)  if ((nome != null && nome.trim().length() > 0))  if ((descrizione != null && descrizione.trim().length() > 0))  if ((prezzoCent != null && prezzoCent.trim().length() > 0))  if ((iv != null && iv.trim().length() > 0))  if ((prodottoDAO.doRetrieveByNomeSingolo(nome) != null) && (prodottoDAO.doRetrieveByDescrizione(descrizione) != null) && (prodottoDAO.doRetrieveByPrezzo(Long.parseLong(prezzoCent)) != null) && (prodottoDAO.doRetrieveByIva(Integer.parseInt(iv)) != null)) |
| Post-condizione | AdminProddottoModificaServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | AdminProddottoServlet |
| Descrizione | Permette di aggiungere un prodotto |
| Pre-condizione: | if (rb == null)  if ((nome != null && nome.trim().length() > 0))  if ((descrizione != null && descrizione.trim().length() > 0))  if ((prezzoCent != null && prezzoCent.trim().length() > 0))  if ((iv != null && iv.trim().length() > 0))  if (idstr.isEmpty()  if (prodottoDAO.doRetrieveByNomeSingolo(nome) != null) |
| Post-condizione | AdminProddottoServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | AdminProddottoServlet |
| Descrizione | Permette di rimuovere un prodotto |
| Pre-condizione: | if (request.getParameter("rimuovi") != null) |
| Post-condizione | AdminProddottoServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | LoginServlet |
| Descrizione | Permette di effettuare il login al sito web |
| Pre-condizione: | if (username != null && password != null)  if (utente != null) |
| Post-condizione | LoginServlet ::doPost(request,response);  session.getAttribute(“utente”)!=null; |

|  |  |
| --- | --- |
| Nome Classe | LogoutServlet |
| Descrizione | Permette di effettuare il logout dal sito web |
| Pre-condizione: | if (cookies != null) |
| Post-condizione | LogoutServlet ::doPost(request,response);  session==null; |

|  |  |
| --- | --- |
| Nome Classe | MieiOrdiniServlet |
| Descrizione | Permette di visualizzare gli ordini di un utente registrato |
| Pre-condizione: | if (utente != null) |
| Post-condizione | MieiOrdiniServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | ModificaAdminServlet |
| Descrizione | Permette di modificare le informazioni di un utente se si è amministratori |
| Pre-condizione: | if ((username != null && username.length() >= 6 && username.matches("^[0-9a-zA-Z]+$")))  if ((nome != null && nome.trim().length() > 0 && nome.matches("^[ a-zA-Z\u00C0-\u00ff]+$")))  if (!(email != null && email.matches("^\\w+([\\.-]?\\w+)@\\w+([\\.-]?\\w+)(\\.\\w+)+$"))) |
| Post-condizione | ModificaAdminServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | OrdineServlet |
| Descrizione | Permette di procedure all’acquisto |
| Pre-condizione: | if ((indirizzo.trim().length() > 0)) |
| Post-condizione | OrdineServlet ::doPost(request,response); |

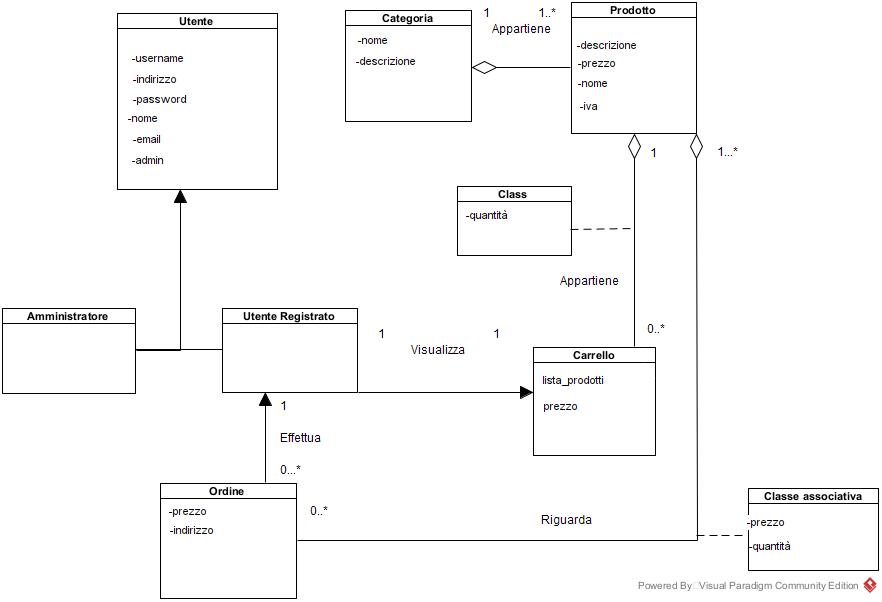
|  |  |
| --- | --- |
| Nome Classe | RegistrazioneServlet |
| Descrizione | Permette di registrarsi al sito web |
| Pre-condizione: | if ((username != null && username.length() >= 6 && username.matches("^[0-9a-zA-Z]+$")))  if ((password != null && password.length() >= 8 && !password.toUpperCase().equals(password)  if (password.equals(passwordConferma))  if ((nome != null && nome.trim().length() > 0 && nome.matches("^[ a-zA-Z\u00C0-\u00ff]+$")))  if ((email != null && email.matches("^\\w+([\\.-]?\\w+)\*@\\w+([\\.-]?\\w+)\*(\\.\\w+)+$"))) |
| Post-condizione | RegistrazioneServlet ::doPost(request,response); |

|  |  |
| --- | --- |
| Nome Classe | VerificaEmailServlet |
| Descrizione | Permette di registrarsi al sito web |
| Pre-condizione: | if (email != null && email.matches("^\\w+([\\.-?\\w+)\*@\\w+  ([\\.-]?\\w+)\*(\\.\\w+)+$") && utenteDAO.doRetrieveByEmail(email) == null) |
| Post-condizione | VerificaEmailServlet ::doPost(request,response); |

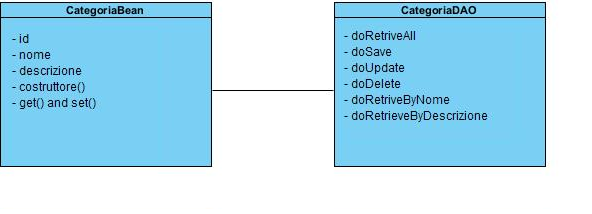
|  |  |
| --- | --- |
| Nome Classe | VerificaUsernameServlet |
| Descrizione | Permette di registrarsi al sito web |
| Pre-condizione: | if (username != null && username.length() >= 6 && username.matches("^[0-9a-zA-Z]+$") |
| Post-condizione | VerificaUsernameServlet ::doPost(request,response); |

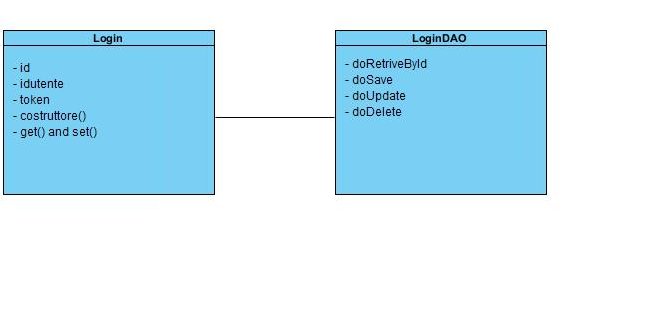
# 4 - **Diagrammi**

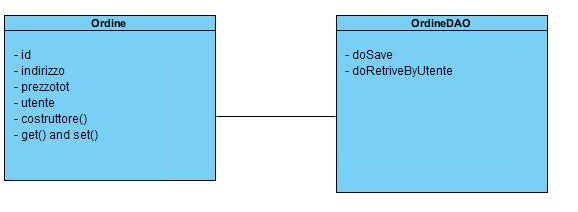
**Class Diagramm**



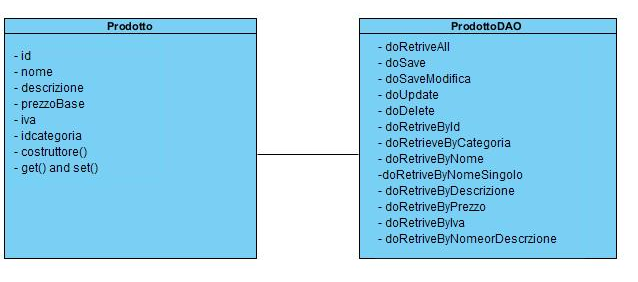
**Categoria**

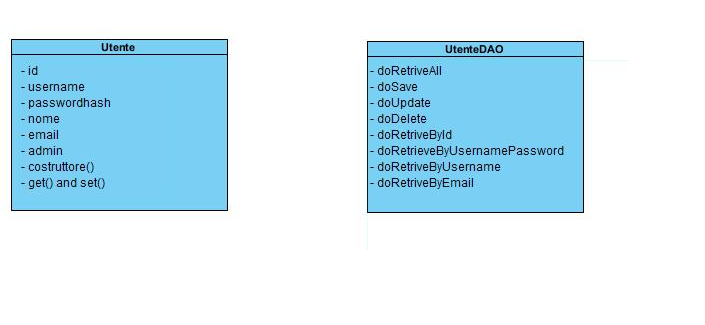


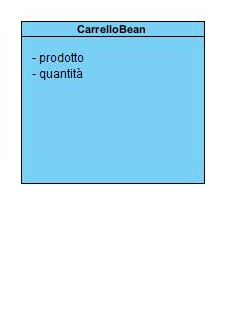
 **Login**

 **Ordine**

**Prodotto**



**Utente**

**Carrello**