

**OBJECT DESIGN DOCUMENT**

**Nome progetto: EsteticaMente**

**Anno accademico 2018/2019**



|  |  |
| --- | --- |
| **Partecipanti:** | **Matricola:** |
|  |  |
| Aurora Scola | 0512103834 |
| Lucia Forte | 0512103948 |
| Marco Minucci | 0512106088 |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 03/01/2019 | 1.0 | Prima stesura ODD | Minucci Marco, Scola Aurora, Forte Lucia |
|  |  |  |  |
|  |  |  |  |

**INDICE**

[1. INTRODUZIONE 4](#_Toc534969727)

[1.1. Object design trade-offs 4](#_Toc534969728)

[1.2. Linee guida per la documentazione delle interfacce 4](#_Toc534969729)

[1.3. Definizioni, acronimi e abbreviazioni 5](#_Toc534969730)

[1.4. Riferimenti 6](#_Toc534969731)

[2. PACKAGES 6](#_Toc534969732)

[2.1. Bean 6](#_Toc534969733)

[2.2. Model 6](#_Toc534969734)

[2.3. Servlet 7](#_Toc534969735)

# INTRODUZIONE

## Object design trade-offs

Dopo aver prodotto i documenti di Requirement Analysis e di System Design nei quali è stato presentato il nostro sistema tralasciando i dettagli implementativi, andiamo ora a stilare il documento di Object Design ove andremo a definire un modello capace di integrare in maniera precisa le funzionalità individuate nei documenti precedenti.

In maniera particolare si vogliono definire le interfacce delle classi, le operazioni, i tipi, gli argomenti, la signature dei sottosistemi definiti nella progettazione del sistema software, i trade-offs e le linee guida.

**Comprensibilità vs Tempo**

Il codice del nostro sistema dovrà essere di facile comprensione e lettura, quindi sarà corredato di commenti per facilitare anche la fase di testing o eventuali future modifiche. Questo, ovviamente, comporterà un aumento del tempo di sviluppo.

**Prestazioni vs Costi – eliminare e mettere da sotto in interfaccia**

Non possedendo un budget, utilizzeremo framework opensource, in particolare Bootstrap per il front-end, per poter mantenere delle buone prestazioni nel nostro sistema.

**Interfaccia vs Usabilità**

L’interfaccia è stata realizzata per rendere l’utilizzo del nostro sistema molto user-friendly così che gli utenti possano interfacciarvisi in maniera semplice ed immediata.

**Sicurezza vs Efficienza**

La sicurezza rappresenta un requisito non funzionale del nostro sistema, come riportato nel RAD. Avendo però tempi di realizzazione ristretti, ci limiteremo a realizzare un sistema di sicurezza concentrato sull’autenticazione degli utenti tramite username e password e la conseguente cifratura di queste ultime.

## Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno rispettare determinate linee guida per la stesura del codice:

**NAMING CONVENTION:**

È buona norma utilizzare nomi:

* Descrittivi
* Pronunciabili
* Di uso comune
* Di lunghezza medio-corta
* Evitando la notazione ungherese
* Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

**VARIABILI:**

* I nomi delle variabili dovranno cominciare con la lettera minuscola ed eventuali altre parole successive dovranno avere la prima lettera maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco.
* Sarà possibile usare il carattere “\_” in proprietà statiche o variabili costanti.

**METODI:**

* I nomi dei metodi dovranno cominciare con la lettera minuscola e le altre successive parole dovranno avere la prima lettera maiuscola. I nomi dei metodi dovranno essere rappresentati da verbi esemplificativi della funzionalità che andranno a svolgere. I nomi dei metodi che saranno utilizzati per accesso o modifica di variabili devono essere del tipo getNomeVariabile() e setNomeVariabile(). (Es. getNome(), setNome()).
* I metodi devono essere corredati da commenti che ne indicano la funzione, parametri e valori di ritorno.

**CLASSI E PAGINE:**

* I nomi delle classi e delle pagine devono cominciare con la lettera maiuscola, così come le altre parole successive. Anche i nomi delle classi devono essere evocativi circa il loro scopo.
* La dichiarazione di una classe è caratterizzata da:

1. Dichiarazione della classe pubblica
2. Dichiarazioni di costanti
3. Dichiarazioni di variabili di classe
4. Dichiarazioni di variabili d’istanza
5. Costruttore
6. Commento e dichiarazione metodi e variabili

## Definizioni, acronimi e abbreviazioni

**Acronimi**

* **RAD:** Requirement Analysis Document

## Riferimenti

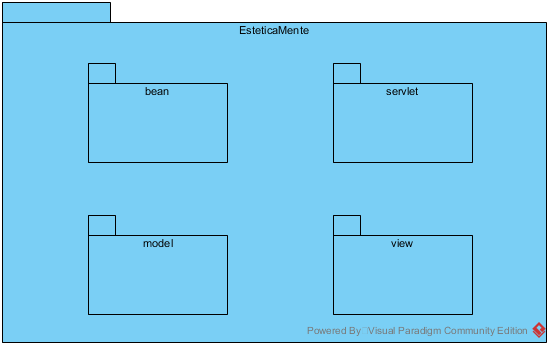
* B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML,
* Pattern and Java, Prentice Hall, 3rd edition, 2009
* Documento SDD del progetto EsteticaMente
* Documento RAD del progetto EsteticaMente

# PACKAGES

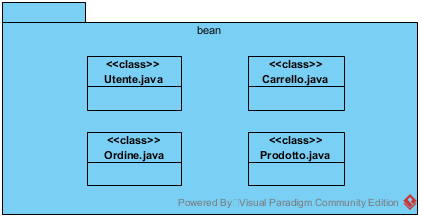
La struttura del nostro sistema è su 3 livelli (three-layer):

|  |  |
| --- | --- |
| **Presentation Layer** | Si occupa del livello di presentazione, ovvero dell’interfaccia, offrendo all’utente la possibilità di poter interagire con il sistema inviando e visualizzando i dati. |
| **Application Logic Layer** | Ha il compito di interagire con il client scegliendo di mostrare delle view o di invocare le specifiche parti dell’applicazione per eseguire determinate richieste.  Esso si occupa di gestioni come:   * Gestione Ordini * Gestione Prodotti * Gestione Utenti * Gestione Carrello * Gestione Pagamento |
| **Storage Layer** | Si occupa di memorizzare i dati del sistema in un DBMS e inoltre riceve richieste dal livello superiore, elaborandole e restituendo i dati richiesti. |

## Package Core

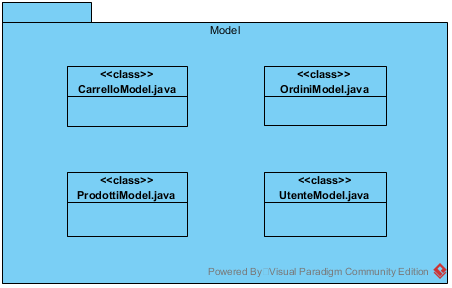


### Package bean



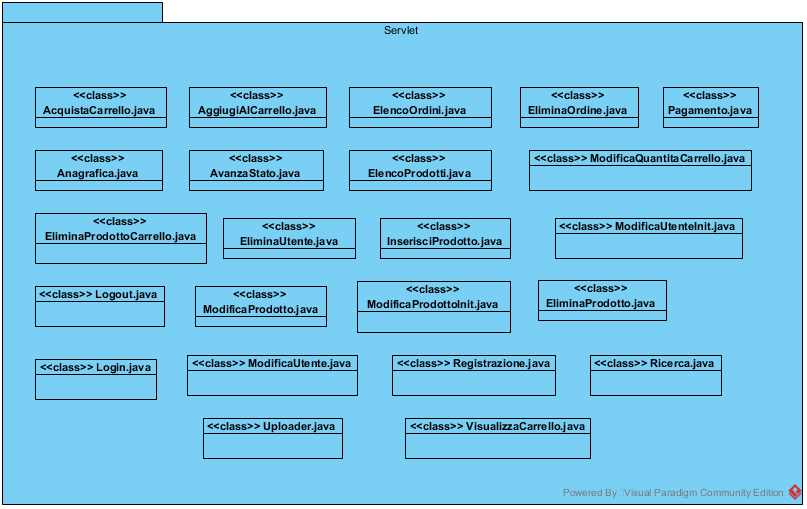
|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| Utente.java | Descrive un utente registrato del sistema. |
| Carrello.java | Descrive un carrello di un utente. |
| Prodotto.java | Descrive un prodotto del sistema. |
| Ordine.java | Descrive un ordine effettuato da un utente del sistema. |

### Package model



|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| CarrelloModel.java | Il model che effettua le operazioni riguardanti il carrello, interfacciandosi al DB a cui è connesso. |
| ProdottiModel.java | Il model che effettua le operazioni riguardanti i prodotti del nostro sistema, interfacciandosi al DB al quale è connesso. |
| OrdiniModel.java | Il model che esegue le operazioni legate agli ordini del sistema, interfacciandosi al DB a cui è connesso. |
| UtenteModel.java | Il model che esegue le operazioni legate agli utenti del nostro sistema, interfacciandosi al DB a cui è connesso. |

### Package servlet



|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| AcquistaCarrello.java | Controller che interviene quando viene confermato l’acquisto del carrello. |
| AggiungiAlCarrello.java | Controller che Interviene quando viene aggiunto un prodotto ad un carrello. |
| Anagrafica.java | Controller che si occupa di far visualizzare i dati anagrafici dell’utente. |
| AvanzaStato.java | Controller che interviene quando l’amministratore cambia lo stato di un ordine. |
| ElencoOrdini.java | Controller che si attiva quando viene richiesta la visualizzazione degli ordini effettuati. |
| ElencoProdotti.java | Controller che si attiva quando viene richiesta la visualizzazione dei prodotti del sistema. |
| EliminaOrdine.java | Controller invocato quando l’utente decide di eliminare un ordine. |
| EliminaProdotto.java | Controller che permette di eliminare un prodotto dal sistema da parte dell’amministratore. |
| EliminaProdottoCarrello.java | Controller che permette all’utente di rimuovere un prodotto dal carrello. |
| EliminaUtente.java | Controller che permette ad un utente di eliminare il proprio account dal sistema. |
| InserisciProdotto.java | Controller che permette all’amministratore di inserire un prodotto nel sistema. |
| Login.java | Controller che permette ad un utente di effettuare il login al sistema. |
| Logout.java | Controller che permette ad un utente di effettuare il logout al sistema. |
| ModificaProdotto.java | Controller che permette all’amministratore di modificare i dati di un prodotto presente nel sistema. |
| ModificaProdottoInit.java | Controller che interviene per visualizzare i campi del prodotto che si vuole modificare. |
| ModificaQuantitaCarrello.java | Controller che permette di modificare la quantità di un prodotto nel carrello. |
| ModificaUtente.java | Controller che permette all’utente di modificare i propri dati dell’account. |
| ModificaUtenteInit.java | Controller che mostra all’utente la sua pagina di modifica dei dati in base al proprio ruolo nel sistema. |
| Pagamento.java | Controller che perette all’utente di effettuare il pagamento del proprio ordine. |
| Registrazione.java | Controller che permette la gestione della registrazione. |
| Ricerca.java | Controller che permette ad un utente di effettuare la ricerca dei prodotti presenti nel sistema. |
| Uploader.java | Controller che permette di effettuare l’upload dell’immagine di un prodotto. |
| VisualizzaCarrello.java | Controller che permette all’utente di visualizzare il carrello. |

### Package view

* AnagraficaAmministratore.jsp
* AnagraficaErrore.jsp
* AnagraficaUtente.jsp
* Carrello.jsp
* Catalogo.jsp
* ChiSiamo.jsp
* Database.jsp
* DatabaseErrore.jsp
* EliminazionePSuc.jsp
* EliminazioneUtente.jsp
* EliminazioneUtenteSucc.jsp
* ErroreGenerale.jsp
* Index.jsp
* InserimentoPFallita.jsp
* InserimentoPSuccesso.jsp
* InserisciProdotto.jsp
* Login.jsp
* LoginErrore.jsp
* LoginNotifica.jsp
* ModificaProdotto.jsp
* ModificaProdottoS.jsp
* ModificaUtente.jsp
* ModificaUtenteE.jsp
* ModificaUtenteS.jsp
* ModificaUtenteUserE.jsp
* Occasioni.jsp
* OrdineAmminsitratore.jsp
* OrdineCliente.jsp
* Pagamento.jsp
* Registrazione.jsp
* RegistrzioneEsistente.jsp
* RegistrazioneFallita.jsp
* RegistrazioneSuccesso.jsp