

DODO.NET

ODD - Object Design Document

|  |  |
| --- | --- |
| Riferimento |  |
| Versione | 1.4 |
| Data | 26/01/2021 |
| Destinatario | Prof.ssa F. Ferrucci |
| Presentato da | Alfonso Cuomo  Simone Farina |
| Approvato da |  |

RevisionHistory

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 10/01/2020 | 1.0 | Stesura introduzione | [tutti] |
| 13/01/2021 | 1.1 | Inserimento dei Design Trade-offs | [Grazia Margarella,  Alfonso Cuomo] |
| 19/01/2021 | 1.2 | Inserimento Linee Guida per la documentazione dell’ intefaccia | [Grazia Margarella,  Alfonso Cuomo] |
| 25/01/2021 | 1.3 | Inserimento Packages , aggiunta Interfacce alle classi e prima stesura del class diagram | [Grazia Margarella,  Alfonso Cuomo] |
| 26/01/2021 | 1.4 | Class Diagram Completo | [Grazia Margarella] |

Sommario

[**1. Introduzione**](#_byalhany7p8s) **4**

[1.1 Object Design Trade-offs](#_dlp1omm4tdv7) 4

[1.1.1 Componenti off-the-shelf](#_9l0doruz0xp7) 4

[1.1.2 Design Pattern](#_bn45b5obyrp0) 4

[1.1.2.1 Singleton](#_agox5zoyku0f) 5

[1.1.2.2 Façade](#_8cw92o8cc40t) 5

[1.1.2.3 Protection Proxy](#_29efjoqggz1g) 6

[1.3 Linee guida per la documentazione dell’interfaccia](#_roxnyr9ile21) 6

[1.5 Definizioni, acronimi e abbreviazioni](#_9uoeo1br5fsg) 7

[**2. Packages**](#_4dulh1nii97k) **7**

[2.1 Control](#_n01og7jro83n) 7

[2.2 Model](#_whlt1xj8lr7i) 7

[2.3 Bean](#_gm0spcbko6mt) 8

[2.4 View](#_24iqd5m9e2vr) 8

[**3. Interfacce delle classi**](#_czeauoodpn24) **8**

[**4. Class diagram**](#_mnjcmof9xv0j) **11**

[**5. Glossario**](#_rmxso75q8ezf) **13**

# 1. Introduzione

## 1.1 Object Design Trade-offs

Nella fase dell’ Object Design bisogna stabilire quali punti rispettare e quali rendere opzionali per evitare diversi compromessi di progettazione. Per la realizzazione del sistema sono stati individuati i seguenti trade-off:

* *Prestazioni vs Affidabilità:* il sistema deve garantire una minor tempi di risposta anche a costo dell’affidabilità , se necessario.
* *Tempo di rilascio vs Funzionalità:* il sistema implementerà tutte le funzioni previste a discapito dei tempi di rilascio necessari per la realizzazione degli stessi.
* *Disponibilità vs Tolleranza ai guasti:* si preferisce garantire un sistema sempre disponibile anche se non affidabile o che non garantisce una coerenza dei dati coinvolti piuttosto che un sistema volto alla tolleranza ai guasti.
* *Buy vs Build*: la necessità di sviluppare un’applicazione web permette di accedere ad una collezione molto vasta di framework e librerie. In questo caso infatti si vuole evitare di costruire soluzioni eccessivamente primitive che vengono già fornite in vari servizi. Per questo motivo si utilizzeranno delle componenti off-the-shelf per l’implementazione delle funzioni principali.

Durante la fase di analisi, in base ai task da svolgere, ogni membro del team ha proposto uno o più framework da utilizzare secondo le proprie conoscenze e preferenze.

### 1.1.1 Componenti off-the-shelf

Il sistema sarà realizzato tramite componenti off-the-shelf (che sono componenti software già disponibili utilizzati per facilitare la creazione del software) quali:

1. *Bootstrap*: framework open source per la realizzazione di interfacce grafiche di web application. Il tool è basato su HTML e CSS ed interagisce con il documento HTML scelto definendo uno stile standard per varie classi CSS predefinite. In particolare tramite la libreria Fluent Design di MDBootstrap (riferimenti [**https://mdbootstrap.com/freebies/jquery/fluent-kit/**](https://mdbootstrap.com/freebies/jquery/fluent-kit/))
2. *jQuery*: libreria del linguaggio Javascript che permette l’interazione tra l’utente e l’interfaccia in maniera dinamica.
3. *Tomcat*: (Apache Tomcat) è un server web open source che implementa le specifiche JavaServer Pages ( JSP ) e servlet, fornendo una piattaforma software per l’ esecuzione di applicazioni web scritte in Java. Esso viene utilizzato per avere un sistema di connection pool, per gestire le connessioni. ( riferimenti: [**http://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html**](http://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html) ).

Tutte le componenti selezionate sono gratuite e open source.

### 1.1.2 Design Pattern

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo dell’applicativo Dodo.net. Per ogni pattern si darà:

• Una brevissima introduzione teorica.

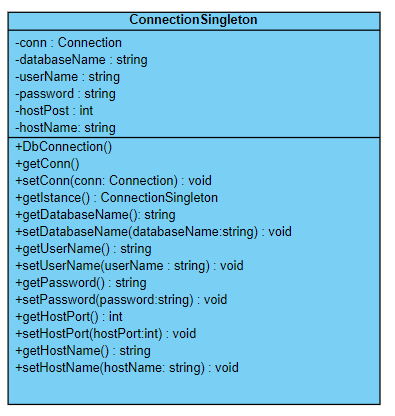
• Il problema che doveva risolvere all’interno di Dodo.net.

• Un grafico della struttura delle classi che implementano il pattern.

#### 1.1.2.1 Singleton

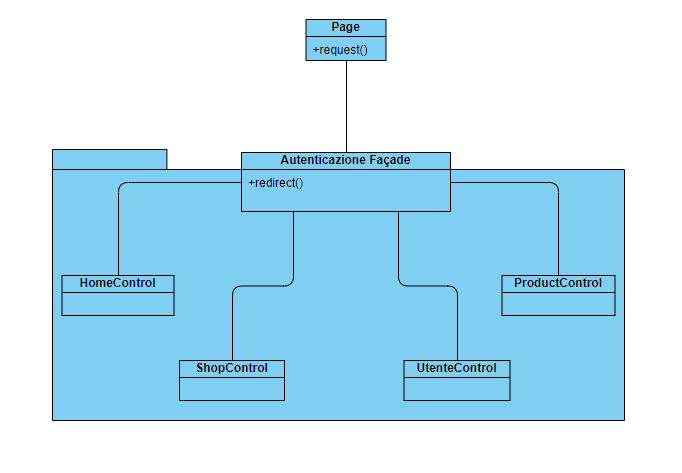
Il singleton è un [design pattern creazionale](https://it.wikipedia.org/wiki/Design_pattern#Pattern_creazionali) che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Abbiamo utilizzato questo design pattern per gestire una classe (ConnectionSingleton) che consente di effettuare tutte le operazioni con il database.



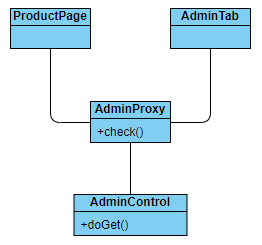
#### 1.1.2.2 Façade

Il Facade Pattern è un design pattern di tipo Strutturale che nasconde la complessità di un sistema e offre una interfacca ai client che vogliono accedere ad esso. In italiano si traduce con “Facciata” perchè è proprio questo il suo obiettivo: semplificare l'accesso ad un sistema complesso mostrando una facciata. E’ comunemente utilizzato nella programmazione orientata agli oggetti, e spesso nelle librerie di Java. L'utilizzo del Façade Pattern ci permette di rendere l’accesso ai servizi di un sottosistema complesso più semplici senza preoccuparsi del funzionamento interno dello stesso. Intendiamo quindi utilizzare quindi questo design pattern per semplificare l’accesso ai servizi del sottosistema che si occupa dei log-in e log-out e della visualizzazione delle categorie e dei prodotti.



#### 1.1.2.3 Protection Proxy

Il Protection Proxy è una particolare tipologia di design pattern di tipo Strutturale che regola e gestisce gli accessi ai servizi di un sottosistema che necessita controlli sulle richieste ai servizi assicurandone l’accesso solo agli aventi diritto. Il proxy sarà dunque implementato come unica classe accessibile per effettuare richieste all’eventuale sottosistema da proteggere. Nel nostro sistema verrà implementato per regolare gli accessi alle funzionalità degli admin garantendone l’accesso solo agli stessi.



#### 1.3 Linee guida per la documentazione dell’interfaccia

Gli sviluppatori devono seguire delle linee guida definite per rendere il progetto coerente e facilmente comprensibile da tutto il team.   
Ogni metodo e ogni classe deve essere precedute da una documentazione ad hoc che descrive sia cosa si intendeva fare con quel metodo sia quali sono le scelte intraprese, anche con commenti all’interno del codice accanto a calcoli o scelte non ovvie. Ogni file deve contenere il nome dell’autore o degli autori.  
La convenzione che deve essere adottata per i nomi delle variabili e delle classi è la Camel Notation (ossia bisogna scrivere le parole in minuscolo tranne per la prima lettera delle successive che andrà in maiuscolo).  
I nomi delle variabili e dei metodi devono essere esplicativi della funzione svolta. Stessa cosa per le eccezioni o altre componenti del codice.  
Il codice Java deve essere indentato in maniera appropriata e l’apertura di un blocco di codice deve avvenire nella stessa riga in cui è specificato il nome della classe o la firma del metodo che quel blocco definisce.  
Il codice HTML deve essere scritto secondo la formattazione richiesta dal linguaggio di markup, con i singoli tag indentati in maniera coerente e leggibile.  
Ogni file deve essere diviso in base alla categoria di appartenenza in package e cartelle specifiche. Tutte le classi che realizzano un sottosistema devono essere racchiuse nello stesso pacchetto Java. Tutte le componenti che realizzano l’interfaccia grafica devono essere collocate nella directory “/WEB-INF/<tag>/” a seconda della tipologia di componente che realizzano. Tutte le risorse statiche (come immagini o altri script) devono essere collocate nella directory “resources”.

Gli script in JavaScript devono essere collocati in file dedicati se svolgono funzioni distinte dal mero rendering e il codice JavaScript deve seguire le stesse convenzioni per il layout e i nomi del codice Java. Inoltre le funzioni JavaScript devono essere documentate come per i metodi Java e infine gli oggetti JavaScript saranno commentati seguendo lo stile di JavaDoc.

I fogli di stile CSS devono essere iniziati da commenti analoghi ai file Java. Gli stili non inline devono essere collocati in fogli di stile separati. Le regole CSS devono essere formattate indicando i selettori a livello 0 dell’indentazione, uno per riga; i selettori devono essere seguiti da una parentesi graffa aperta; le proprietà sono elencate una per riga e indentate rispetto ai selettori; la regola deve terminare con una parentesi graffa chiusa su una riga vuota.

Per quanto riguarda il Database SQL i nomi delle tabelle devono essere sostantivi singolari tratti dal dominio del problema. I nomi degli attributi devono essere sostantivi singolari tratti dal dominio applicativo e esplicativi del contenuto. Se i nomi delle colonne sono costituiti da più parole è previsto l’utilizzo dell’underscore (\_). Anche il codice SQL deve essere commentato brevemente per indicare le funzioni delle query descritte.

## 1.5 Definizioni, acronimi e abbreviazioni

**CSS:** Linguaggio usato per definire la formattazione di pagine web.  
**HTML:** Linguaggio di mark-up per pagine web.  
**Off-The-Shelf:** Servizi esterni di cui viene fatto utilizzo da terzi.

**MVC:** acronimo di Model-view-controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software.

**Bootstrap**: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web.

**JavaScript**: JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

**Servlet**: i servlet sono oggetti scritti in linguaggio Java che operano all'interno di un server web.

**JSP**: acronimo di Java Scripting Preprocessor, è una tecnologia di programmazione web in java per lo sviluppo della logica di presentazione (tipicamente secondo il pattern MVC) di applicazioni web.

# 2. Packages

In questa sezione si provvederà a definire in maniera dettagliata la divisione in pacchetti dei singoli sottosistemi.

## 2.1 Control

Il package Control contiene al suo interno le servlet necessarie per eseguire la logica di business richiesta dall’utente tramite la view. Le servlet contenute sono:

* HomeControl
* LoginControl
* LogoutControl
* ShopControl
* CarrelloControl
* GetCopertinaControl
* AdminControl
* AcquistoControl

## 2.2 Model

Il package Model fornisce l’accesso ai dati persistenti nel database. Tra queste classi ci si occupa della connessione con ConnectionSingleton e delle singole tabelle con:

* ProdottoModelDM
* UtenteModelDM
* OrdineModelDM
* ComposizioneModelDM

## 2.3 Bean

Il package Bean ha al suo interno i JavaBeans ed è una sottoparte del model che rappresentano le entità presenti nel database. Queste sono:

* ProdottoBean che descrive i prodotti
* UtenteBean che descrive l’utente
* OrdineBean che descrive l’ordine legandolo all’utente
* ComposizioneBean che descrive quali prodotti appartengono all’ordine

Inoltre è presente la classe Carrello che descrive la composizione del carrello ma che non è presente nel database in quanto è gestita tramite la sessione.

Essi quindi permettono un ottimo incapsulamento del codice, peraltro riutilizzabile. Al programmatore sarà pressochè invisibile la sezione di codice puro, sostituito da richiami ai metodi delle classi incluse.

## 2.4 View

Il package View ha al suo interno sia le risorse statiche (nella cartella “resources”) che le JSP HomePage che permette la visualizzazione della pagina iniziale, ShopPage che permette di visualizzare la lista dei prodotti offerti, ProductView che permette di visualizzare un singolo prodotto in particolare, LoginPage che permette all’utente di effettuare il login, RegistrationPage permette la registrazione, LogoutTab che permette agli utenti di eseguire il logout e AdminTab che permette al solo admin di aggiungere nuovi prodotti, CartPage che permette di visualizzare il carrello del cliente e OrderPage che permette di effettuare la procedura di acquisto. Una parte di JSP riutilizzata dalle altre pagine è header che si trova nella sottocartella “tools”.

# 3. Interfacce delle classi

|  |  |
| --- | --- |
| Nome Classe | Index |
| Descrizione | Questa classe visualizza la pagina iniziale del sito e reindirizza a Login, ShopPage, UserPage e CarrelloPage |
| Metodi e Attributi |  |
| Pre-Condizione |  |
| Post-Condizione |  |
| Invariante |  |

|  |  |
| --- | --- |
| Nome Classe | Login |
| Descrizione | Questa classe permette di gestire la procedura di login o la procedura di registrazione di un utente |
| Metodi e Attributi |  |
| Pre-Condizione | context Login: validate(email,Password) 0;  pre:email!=null && password!=null  email.equals(DB.email) &&password.equals(DB.password)  context Registrazione: newCliente(email,Password) 0;  pre:email!=null && password!=null && nome !=null && cognome!=null  !email.equals(DB.email) |
| Post-Condizione |  |
| Invariante |  |

|  |  |
| --- | --- |
| Nome Classe | ShopPage |
| Descrizione | Questa classe gestisce la visualizzazione e l’interazione con i prodotti presenti nel DataBase. |
| Metodi e Attributi |  |
| Pre-Condizione |  |
| Post-Condizione |  |
| Invariante |  |

|  |  |
| --- | --- |
| Nome Classe | CarrelloPage |
| Descrizione | Questa classe visualizza il carrello di un utente |
| Metodi e Attributi |  |
| Pre-condizione | context CarrelloPage:  Utente.isLogged |
| Post-Condizione |  |
| Invariante |  |

|  |  |
| --- | --- |
| Nome Classe | AcquistoPage |
| Descrizione | Questa classe visualizza la pagina per la procedura di acquisto di un prodotto |
| Metodi e Attributi |  |
| Pre-Condizione | context AcquistoPage:  Utente.isLogged && !Carrello.isEmpty |
| Post-Condizione |  |
| Invariante |  |

|  |  |
| --- | --- |
| Nome Classe | LogoutTab |
| Descrizione | Questa classe permette di eseguire la procedura di logout da parte di un utente o admin dal sito |
| Metodi e Attributi |  |
| Pre-Condizione | context Logout: logout(User)  pre: User.isLogged |
| Post-Condizione |  |
| Invariante |  |

|  |  |
| --- | --- |
| Nome Classe | AdminTab |
| Descrizione | Questa classe permette l’inserimento di nuovi prodotti nello store e quindi inserendolo nel DataBase |
| Metodi e Attributi |  |
| Pre-Condizione | context UserPage:  Utente.isLogged && Utente.isAdmin |
| Post-Condizione |  |
| Invariante |  |

# 

# 

# 4. Class diagram

# 5. Glossario

**CSS:** Linguaggio per la definizione degli stili delle pagine web.  
**Framework**: Software di supporto allo sviluppo. **Componenti off-the-shelf**: prodotti software sviluppati da terzi riutilizzabili.  
**JavaScript**: Linguaggio di scripting nato per dare dinamicità alle pagine HTML.  
**JavaServer Pages**: Tecnologia che facilita lo sviluppo di pagine web dinamiche fornendo un’interfaccia Java tag-oriented con il back end.  
**Javadoc**: Documentazione generata automaticamente a partire dai commenti scritti nei sorgenti Java.