**Object Design document**

1. **Introduzione**

**Object Design Trade-offs**

Ci apprestiamo a definire gli aspetti implementativi del nostro sistema, aspetti che nei documenti precedenti (e.g. “RAD”, “SSD” etc.) sono stati tralasciati per definire in modo chiaro gli obiettivi del sistema. Con il seguente documento puntiamo a creare un modello capace di descrivere in modo preciso tutte le funzionalità individuate nei modelli precedenti.

Nel particolare, definiremo le interfacce delle classi, le operazioni, i tipi e gli argomenti e le “signature” dei sottosistemi definiti nel System Design.

Sono necessari però definire una serie di trade-off e di linee guida:

*Comprensibilità vs Tempo:*

Un codice comprensibile e commentato sarà sempre preferito rispetto ad un codice scritto rapidamente; questo perché è importante che il codice rimanga di facile fruizione per tutti i collaboratori del progetto, anche se potrà rallentarci più avanti nello sviluppo.

*Prestazioni vs Costi:*

Il progetto è portato avanti senza alcuna sovvenzione economica, quindi si farà affidamento a tecnologie “Open Source” e a strumenti free online, si farà il possibile per evitare intaccare il meno possibile le prestazioni del sistema per abbassare il più possibile i costi dell’intero sistema.

*Interfaccia vs Usabilità:*

L’interfaccia fa dell’usabilità il suo punto di forza. Fin dal primo momento la comodità di utilizzo è stata messa in primo piano durante il design, scelta che si rispecchia fin dai primi mock-up.

*Sicurezza vs Efficienza:*

Il poco tempo a disposizione non ci permette di rendere il sistema resistente a possibili attacchi esterni. Metteremo comunque il minimo di sicurezza possibile nel progetto, assicurandoci che il login/registrazione e tutte le altre interazioni con il database siano sicure e quanto più efficienti possibili.

*1.2 Linea guida per la documentazione delle interfacce*

Per creare un progetto chiaro, pulito e univocamente compreso, gli sviluppatori dovranno sottostare ad alcune linee guida per la stesura del codice:

*Code style:*

È buona norma rispettare gli standard di Java circa la formattazione del codice, nello specifico:

• Tutto il codice sorgente deve essere indentato con tabs

• Il codice deve essere formattato secondo lo stile vigente dell’IDE utilizzato (si consiglia caldamente Eclipse)

Esempio:

for(Object x: listaElementi){

System.out.println(x.getNome());

}

*Naming Convention:*

È buona norma utilizzare nomi che siano:

• Descrittivi

• Pronunciabili

• Di uso comune

• Non abbreviati (se non per variabili momentanee)

• Utilizzando solo i caratteri consentiti (A-Z, a-z, 0-9)

*Variabili:*

È buona norma rispettare gli standard di Java circa la nomenclatura delle variabili, nello specifico:

• Devono iniziare per lettera minuscola e ogni parola seguente deve iniziare con una maiuscola. Esempio: nomeDellaVariabile

• Nel caso di variabili costanti, il nome deve essere interamente in maiuscola ed eventuali spazi devono essere sostituiti da underscore (‘\_’). Esempio: ORE\_IN\_UN\_GIORNO

*Metodi:*

È buona norma rispettare gli standard di Java circa la nomenclatura dei metodi, nello specifico:

• Devono iniziare per lettera minuscola e ogni parola seguente deve iniziare con una maiuscola. Esempio: nomeDelMetodo()

• Eventuali metodi per l’accesso e la modifica delle variabili d’istanza di una classe devono essere del tipo getNomeVariabile() e setNomeVariabile() Esempio: getOrario(), setOrario()

• I commenti devono essere raggruppati in base alla loro funzionalità, devono essere situati prima della dichiarazione del metodo. È obbligatorio usare il sistema “JavaDocs” per poter avere una documentazione completa ed uniforme.

*Classi:*

È buona norma rispettare gli standard di Java circa la nomenclatura delle classi, nello specifico:

• Devono iniziare per lettera maiuscola e ogni parola seguente deve iniziare con una maiuscola. Esempio: NomeDellaClasse

• Devono avere un nome autodescrittivo.

*1.3 Definizioni, Acronimi, Abbreviazioni e Riferimenti*

**Acronimi:**

• RAD: Requirements Analysis Document

• SDD: System Design Document

• ODD: Object Design Document

**Abbreviazioni:**

• DB: Database

**Riferimenti:**

• Libro di testo: Bruegge, A.H. Dutoit, Object Oriented Software Engineering.

• Slide fornite dal professore, reperibili su e-learning.

1. **Packages**

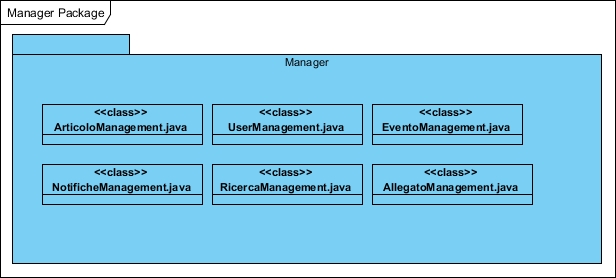
*Package core*

*Bean package*

*Controller package*

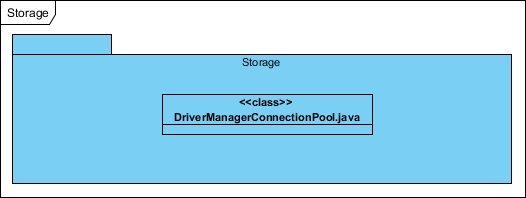
*Exception package*

*Model package*



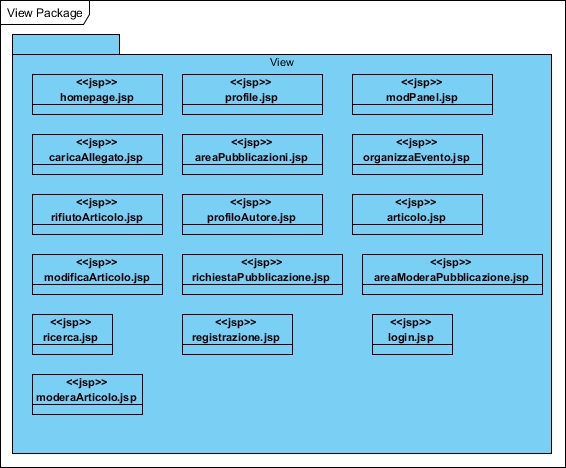
|  |  |
| --- | --- |
| Classe: | Descrizione: |
| ArticoloManagement.java | Gestisce tutte le funzionalità per la pubblicazione/moderazione/modifica dell’articolo. |
| UserManagement.java | Gestisce tutte le funzionalità per la gestione delle informazioni utente. |
| EventoManagement.java | Gestisce tutte le funzionalità per la gestione degli eventi. |
| NotificheManagament.java | Gestisce tutte le funzionalità per gestire le notifiche. |
| RicercaManagement.java | Gestisce tutte le funzionalità per la ricerca. |
| AllegatoManagement.java | Gestisce tutte le funzionalità per per la gestione degli allegati. |

*Storage package*



|  |  |
| --- | --- |
| Classe: | Descrizione: |
| DriverManagerConnectionPool.java | Si occupa di gestire il pool delle connessioni al database. |

*View package*



|  |  |
| --- | --- |
| Classe: | Descrizione: |
| homepage.jsp | Gestisce la visualizzazione della Homepage |
| profile.jsp | Gestisce la visualizzazione delle informazioni del profilo utente |
| modPanel.jsp | Gestisce la visualizzazione del pannello di controllo del moderatore |
| caricaAllegato.jsp | Gestisce la visualizzazione della pagina per caricare gli allegati |
| areaPubblicazioni.jsp | Gestisce la visualizzazione degli articoli pubblicati da parte di un autore |
| organizzaEvento.jsp | Gestisce la visualizzazione della pagina per creare un evento |
| rifiutoArticolo.jsp | Gestisce la visualizzazione della pagina per rifiutare un articolo |
| profiloAutore.jsp | Gestisce la visualizzazione della pagina autore |
| articolo.jsp | Gestisce la visualizzazione della pagina articolo |
| modificaArticolo.jsp | Gestisce la visualizzazione della pagina per la modifica di un articolo |
| richiestaPubblicazione.jsp | Gestisce la visualizzazione della pagina per inviare la pubblicazione di un articolo. |
| areaModeraPubblicazione.jsp | Gestisce la visualizzazione della pagina per iniziare la moderazione di un articolo presenti nella lista degli articoli in attesa di moderazione |
| Ricerca.jsp | Gestisce la visualizzazione della pagina per mostrare gli articoli o autori trovati |
| Registrazione.jsp | Gestisce la visualizzazione della pagina per registrare un’utente nel sistema |
| login.jsp | Gestisce la visualizzazione della pagina per permettere l’accesso ad un utente |
| moderaArticolo.jsp | Gestisce la visualizzazione della pagina per permettere la moderazione di un articolo |

3. **Interfaccia delle Classi**

Ogni metodo e ogni classe sarò opportunamente descritta nella documentazione “javadoc” allegata.

4. **Design Patterns**

DAO (Data Access Object) è un pattern per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un database, usata principalmente in applicazioni web sia di tipo Java EE sia di tipo EJB, per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe). Nel nostro sistema abbiamo una serie di classi DAO che ci permettono di comunicare con il database (locale e non) senza doverci preoccupare di gestire le connessioni, le request http e i dati ricevuti da queste ultime.