|  |
| --- |
| **Università degli Studi di Salerno Corso di Ingegneria del Software** |

***SocialBook  
Documento Analisi Requisiti***

****

**Partecipanti:**

|  |  |
| --- | --- |
| Nome | Matricola |
| Barbato Alessia | 0512105858 |
| Proietto Angelica | 0512105762 |
| Russo Luca | 0512105840 |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 25/01/2021 | 1.0 | Stesura introduzione (Paragrafi 1.1, 1.2, 1.3, 1.4, 1.5) | Proietto Angelica |

Indice

1.Introduzione 4

1.1 Object Design trade-offs 4

1.2 Linee guida per la documentazione delle interfacce 8

1.3 Definizioni, acronimi, abbreviazioni 10

1.4 Design Pattern 11

1.5 Riferimenti 11

2. Packages

3. Interfacce delle classi 12

4. Glossario

1. Introduzione

Dopo aver stilato i documenti Requirements Analysis e System Design è necessario porre attenzione sugli aspetti implementativi. Questo documento ha l’obiettivo di produrre un modello che integri in modo coerente tutte le informazioni collezionate nelle fasi precedenti. In particolar modo, verranno definite le interfacce delle classi, le operazioni supportate, i tipi dei dati, i parametri delle procedure, le signatures dei sottosistemi definiti nel documento di System Design, i trade-offs e le linee guida.

* 1. Object Design trade-offs

***FUNZIONALITA’ vs AFFIDABILITA’***

L’integrazione di molteplici funzionalità potrebbe portare con maggiore facilità a malfunzionamenti di componenti software e quindi comprometterne drasticamente l’affidabilità.

***COMPRENSIBILITA’ vs TEMPO DI SVILUPPO***

Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorarne la leggibilità; tuttavia questo richiederà una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.

**TEMPO DI RISPOSTA vs AFFIDABILITA’**

Per garantire che il sistema non si blocchi in caso di picchi di carico elevati, si preferisce rinunciare a un ottimo tempo di risposta. Si è deciso di tollerare questa problematica al fine di offrire agli utenti un sistema per l’appunto affidabile.

* 1. Linee guida per la documentazione delle interfacce

In questa sezione vengono definite le linee guida che ogni sviluppatore rispetterà per la leggibilità del codice. Per la formattazione dei file XML, HTML, CSS e JS si userà il formatter di IntelliJ di default, mentre per i file Java si seguiranno le convenzioni della Sun di Java . Tra cui evidenziamo alcuni aspetti fondamentali:

1.2.1 Java Naming Convention

I nomi delle classi Java devono:

● Essere singolari;

● Essere descrittivi;

● Utilizzare solo caratteri consentiti (a-z, A-Z, 0-9);

● Iniziare con la lettera maiuscola, così come le parole successive all’interno del nome.

Esempio: *NomeClasse.java*

I nomi delle variabili devono:

● Iniziare con la lettera minuscola

- Se sono variabili locali composte da più nomi, le parole successive devono iniziare con la lettera maiuscola;

Esempio: *nomeVariabileLocale*

- Se sono variabili d’istanza (ad esempio, nelle classi bean) composte da più nomi, le parole sono divise dal simbolo \_ (underscore);

Esempio: nome\_variabile\_d\_istanza

● Descrivere il significato della variabile in questione;

● Essere scritti in inglese.

I nomi dei metodi devono:

● Iniziare con la lettera minuscola e le parole successive all’interno del nome con la lettera maiuscola;

Esempio: *nomeMetodo()*

● Essere chiari e descrivere l’azione che il metodo eseguirà;

● Rispettare il pattern getNomeVariabile e setNomeVariabile nel caso in cui si tratti rispettivamente di metodi getter o setter.

Dichiarazioni:

● variabile locale

1) dichiarata e inizializzata nel blocco di codice in cui è necessaria;

2) dichiarata all’esterno di un blocco di codice, nel caso in cui ci sia il bisogno di inizializzarla in un secondo momento e di riutilizzarla anche in blocchi differenti.

● variabile d’istanza

1) dichiarata in un rigo e inizializzata successivamente nel costruttore della classe oppure tramite il metodo setter associato a quella variabile;

2) una sola dichiarazione di una variabile per ogni riga.

1.2.2 HTML Convention

Le pagine HTML, sia in forma statica che dinamica (ovvero JSP), devono essere conformi allo standard HTML 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

● Un’indentazione consiste in una tabulazione;

● Ogni tag deve avere un’indentazione maggior del tag che lo contiene;

● Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;

● I tag di commento, se presenti, devono seguire le stesse regole che si applicano ai tag normali.

1.2.3 CSS Convention

I fogli di stile (CSS) devono seguire le seguenti convenzioni:

● Tutti gli stili non inline devono essere collocati in fogli di stile separati;

● Ogni regola CSS deve essere formattata come segue:

1 – I selettori della regola si trovano a livello 0 di indentazione, separati da una virgola;

2 – L’ultimo selettore della riga è seguito da una parentesi graffa aperta ({);

3 – Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;

4 – La regola è determinata dalla parentesi graffa chiusa (}), collocata sulla riga successiva all’ultima proprietà elencata.

1.2.4 Database SQL Convention

I nomi delle tabelle devono seguire le seguenti regole:

● Devono essere costituiti di sole lettere;

● Devono iniziare con una lettera minuscola e se si tratta di nomi composti, le parole successive devono iniziare con una lettera maiuscola;

● Devono essere sostantivi singolari tratti dal dominio del problema ed esplicativi del contenuto.

I nomi dei campi devono seguire le seguenti regole:

● Devono essere costituiti di sole lettere minuscole;

● Se si tratta di nomi composti, le parole devono essere separate dal simbolo \_ (underscore).

* 1. Definizioni, acronimi, abbreviazioni

**RAD:** Requirements Analysis Document.

**SDD:** System Design Document.

**HTTPS:** HyperText Transfer Protocol Secure.

**JSP:** Java Server Page.

**HTML:** Hyper Text Markup Language.

**CSS:** Cascading Style Sheets.

**JS:** JavaScript.

**SQL:** Structured Query Language.

* 1. Design Pattern

1.4.1 Singleton Pattern

Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, così da fornire un punto di accesso globale a tale istanza. Abbiamo progettato una classe Singleton (**ConPool**) per evitare la perdita di efficienza dovuta alla creazione di più istanze di questa classe.

Per realizzare il singleton pattern occorre avere:

● Una variabile privata statica della classe, nel nostro caso dataSource, che rappresenta l’unica istanza creata;

● Un metodo statico e pubblico getConnection(), che restituisce l’istanza in questione.

Il suo scopo è:

● Avere un accesso controllato all’unica istanza della classe;

● Centralizzare informazioni e comportamenti in un’unica entità condivisa dagli utilizzatori.

Il principale vantaggio è la mutua esclusione.

* 1. Riferimenti

● Object-Oriented Software Engineering Using UML, Patterns, and Java, Bernd Bruegge & Allen H. Dutoit.

● SocialBook RAD.

● SocialBook SDD.

1. Packages