DigitalMenu

OBJECT DESIGN DOCUMENT

Presentato da:
Antonio Della Monica
Alberto Cuomo
Gerardo De Lorenzo

Sommario

1. Introduzione

1.1.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. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorare la leggibilità; tuttavia questo richiede una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.

Interfaccia vs Usabilità:

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

Manutenibilità vs Performance:

Il sistema sarà progettato in modo da avere un'ottima manutenibilità garantita dal minimo accoppiamento possibile. Per ottenere ciò, bisognerà implementare un codice modulare per minimizzare la coesione, a discapito però delle performance del sistema;

1.2. Tool

React: è una libreria Javascript open source sviluppata da Facebook per creare UI (User Interface) interattive. Lo scopo principale della libreria è proprio la semplificazione della realizzazione di interfacce UI dinamiche che possono reagire ai cambiamenti di dati in maniera autonoma, attraverso opportuni componenti.

Ant Design: è un insieme di component UI per React, in pratica una serie di widget open source e liberamente utilizzabile.

Selenium: E' un framework per il testing di applicazioni web.

JUnit: E' un framework per il testing di unità per il linguaggio di programmazione Java.

1.3.Linee Guida per la Documentazione delle Interfacce

Naming Convention:

Alla luce di quanto analizzato, sarà necessario utilizzare nomi:

- Descrittivi;
- Pronunciabili;
- Di uso comune;
- Di lunghezza medio-corta;
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9);
- Senza prefissi o suffissi.

Variabili:

- I nomi delle variabili dovranno iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola.
- Le variabili locali non saranno dichiarate all'inizio del blocco che le contiene,ma verranno dichiarate in corrispondenza del punto in cui vengono usate per la prima volta: il tutto è volto alla limitazione del loro scope.
- In ogni riga dovrà esserci un'unica variabile dichiarata, eventualmente allineata con quelle del blocco dichiarativo.

- In determinati casi, è possibile utilizzare il carattere underscore "_": il caso principale previsto è quello relativo alla dichiarazione di costanti oppure di proprietà statiche.

Metodi:

- I nomi dei metodi dovranno iniziare con la lettera minuscola e le parole successive con la lettera maiuscola.
- Il nome del metodo sarà costituito da un verbo che ne identifica l'azione seguito da un sostantivo, eventualmente aggettivato.
- Il nome dei metodi accessori e modificatori seguirà, rispettivamente, i pattern getNomeVariabile e setNomeVariabile.
- Nel caso in cui vengano utilizzati costrutti "if", "else", "for", "do" e "while" nei metodi dovranno essere utilizzate le parentesi graffe, anche se essi constano di una sola istruzione.

Classi Java:

- I nomi delle classi e delle pagine dovranno iniziare con la lettera maiuscola, così come le parole successive all'interno del nome.
- I nomi delle classi e delle pagine dovranno corrispondere alle informazioni e le funzioni fornite da quest'ultime.
- Ogni file sorgente .class dovrà contenere una singola classe e dev'essere strutturato come segue.
- Le classi saranno strutturate prevedendo rispettivamente:
    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.
- Nel caso in cui una classe avesse costruttori o metodi multipli con lo stesso nome, questi saranno posizionati sequenzialmente senza codice posto fra essi.

Packages:

- Non saranno ammessi caratteri speciali.
- Import statici e non statici in saranno specificati in blocchi singoli e separati;
- Gli import statici non saranno usati per le classi annidate ma verranno importate con import tradizionali

Database:

Per le tabelle del Database vanno seguite le seguenti regole:

- Il nome di una tabella inizia con lettera minuscola e se è formata da più parole, le seguenti parole inizieranno con lettera maiuscola. Tra le parole non ci sarà lo spazio.
- Il nome dovrà essere un sostantivo singolare.

Per gli attributi delle tabelle del Database vanno eseguite le seguenti regole:

- Devono essere costruiti da sole lettere minuscole.
- Se l'attributo è costruito da più parole, le due parole vengolo separate da un underscore. Esempio: nome_campo. Il nome deve essere un sostantivo singolare.

1.4.Definizioni, acronimi e abbreviazioni

Al momento non sono presenti definizioni, acronimi e abbreviazioni.

1.5.Design Pattern

1.4.1 Singleton Pattern

Si è scelto di utilizzare il Singleton Pattern che deve assicurarsi che esista una singola istanza della classe Singleton, che nel nostro caso si chiama DriverManagerConnectionPool, e fornire un accesso globale a tale istanza. L'istanza viene dichiarata come variabile privata statica e creata quando viene inizializzata.
Per realizzare il singleton pattern occorre avere:

- una variabile privata statica della classe che rappresenta l'unica istanza creata;
- un metodo pubblico getInstance che torna l'istanza.

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 offerto è:

- Mutua esclusione

2. PACKAGES

Come possiamo notare dal documento SDD Digital menu le componenti base che costituiscono il sistema sono raccolte in moduli a livelli.
I tre livelli rappresentano la suddivisione dettata dal modello di architettura preso in considerazione per il sistema Digital menu Tree Trier. Ciascun livello rappresenta un package contenente le componenti relative alle funzioni associate al livello.

PACKAGE SOURCE

- BEAN
    - CategoriaBean.java
    - ProdottoBean.java
    - DettagliOrdineBean.java
    - IngredienteBean.java
    - OrdineBean.java
- DAO
    - CategoriaDAO.java
    - IngredienteDAO.java
    - OrdineDAO.java
    - ProdottoDAO.java
    - DriverManagerConnection.java
- VIEW
    - VistaOrdini.js
    - VistaCarrello.js
    - VistaProdotti.js
    - VistaCategorie.js
    - ItemDetProdotto.js
    - VistaGestioneMenu.js
    - Login.js
    - Logout.js
    - VistaOrdiniNonPagati.js

- CONTROLLER
    - GestioneCategoria.java
    - GestioneCodaOrdini.java
    - GestioneOrdine.java
    - GestioneProdotto.java
    - GestionePagamento.java
    - GestoreStatoOrdine.java
    - ListaOrdine.java
    - VisualizzaCat.java

CONTROL	
Class	**Description**
GestioneCategoria.java	Gestisce la visualizzazione dei prodotti di una categoria
GestioneCodaOrdini.java	Gestisce tutti gli ordini che non sono stati completati
GestioneOrdine.java	Gestisce l'aggiunta dell'ordine
GestioneProdotto.java	Gestisce i dettagli del prodotto
GestionePagamento.java	Gestisce il pagamento
GestoreStatoOrdine.java	Gestisce lo stato dell'ordine
VisualizzaCat.java	Gestisce la visualizzazione delle categorie

DAO	
Class	**Description**
CategoriaDAO.java	Classe che modella le interazioni della categoria con il database
DriverManagerConnectionPool.java	Classe che si occupa della connessione con il database
IngredienteDAO.java	Classe che modella le interazioni dell'ingrediente con il database
OrdineDAO.java	Classe che modella le interazioni dell'ordine con il database
ProdottoDAO.java	Classe che modella le interazioni del prodotto con il database

VIEW	
Class	**Description**
VistaOrdini.js	Componente che permette di visualizzare la lista degli ordini con i relativi prodotti e stato dell'ordine
VistaOrdiniNonPagati.js	Componente che permette di visualizzare la lista degli ordini con i relativi prodotti non pagati
VistaCarrello.js	Componente per la gestione e la visualizzazione del carrello

VistaProdotti.js	Componente che permette di visualizzare la lista dei prodotti disponibili
VistaCategorie.js	Componente che permette di visualizzare la lista delle categorie disponibili
ItemDetProdotto.js	Componente che permette di visualizzare i dettagli di un prodotto
Login.js	Componente che permette di accedere alle funzioni amministrative
Logout.js	Componente che permette di uscire dalle funzioni amministrative

BEANS	
Class	**Description**
CategoriaBean.java	Classe che rappresenta le informazioni di un’entità Categoria
DettagliOrdineBean.java	Classe che rappresenta le informazioni di un’entità DettagliOrdine
IngredienteBean.java	Classe che rappresenta le informazioni di un’entità Ingrediente
OrdineBean.java	Classe che rappresenta le informazioni di un’entità Ordine
ProdottoBean.java	Classe che rappresenta le informazioni di un’entità Prodotto

3. Interfaccia Classi

Nome Classe **Context CategoriaDAO.java**			
Metodi	**Descrizione**	**Pre-Condition**	**Post-Condition**
List<ProdottoBean>getAllPiattiByCat(int idCat)	Restituisce la lista di prodotti di una categoria in base all'id	self.idCat->exist(idCat)	listaProdotti!=null
List<CategoriaBean> getAllCat()	Restituisce una lista di tutte le categorie presenti nel sistema		listaCategoria!=null
void AddCategoria(int idCat, String nome, String foto)	Aggiunge una categoria al sistema	nome != null && foto != null && idCat >0 && !self.idCat->exist(idCat)	self.idCat->exist(idCat)
void DeleteCategoria(int idCat)	Cancella una categoria del sistema	idCat!=null self.idCat->exist(idCat)	!self.categorie-> exist(idCat)
void UpdateCategoria (int idCat, String nome, String foto)	Modifica una categoria del sistema	nome != null && foto != null && idCat >0 && self.idCat->exist(idCat)	self.categorie-> exist(categoria) && self.Categorie-> exist(categoria. equals(self.Categorie))

Nome Classe **Context IngredienteDAO.java**			
Metodi	**Descrizione**	**Pre-Condition**	**Post-Condition**
List<IngredienteBean> getAllIngredientiByPro (int idProdotto)	Restituisce la lista di ingredienti di un prodotto in base all'id	idProdotto!=null && self.idProdotto->exist(idPro)	listaIngredienti !=null && self.idProdotto-> exist(idPro)
void AddIngrediente(String nome, int isRimovibile, int idIng)	Aggiunge un ingrediente al sistema	nome !=null && isRimovibile!=null && !self.idIng->exist(idIng)	self.idIng->exist(idIng)
void DeleteIngrediente(int idIng)	Cancella un ingrediente del sistema	idIng!=null && self.idIng->exist(idIng)	!self.idIng->exist(idIng)
void UpdateIngrediente (String nome, int isRimovibile, int idIng)	Modifica un ingrediente del sistema	nome !=null && isRimovibile!=null && self.idIng->exist(idIng)	self.idIng-> exist(idIng) && self.idIng. equals(idIng)

Nome Classe **Context OrdineDAO.java**			
Metodi	**Descrizione**	**Pre-Condition**	**Post-Condition**
void doSave(List<DettagliOrdineBean> p, int stato, int idO)	Salva un ordine all'interno del sistema	p!=null && p.quantita>0 && self.p.DeleteIng->exist(DeleteIng) && p.idP>0 && self.p.idP->exist(idP) && p.prezzo>0 && stato<5 || stato>=0 && idO>0 && !self.idO->exist(idO)	!self.idO->exist(idO))
int getLastId()	Restituisce l'id dell'ultimo ordine fatto		id>0
int getstato(int id)	Restituisce lo stato dell'ordine in base all'id	id>0 && self.idO->exist(id)	stato>=0 && self.idO->exist(id)
void setstato(int id, int stato)	Setta lo stato di quell'ordine in base all'id	id>0 && stato>0 && stato<3 && self.stato->equals(stato+1) && self.idO->exist(id)	self.idO->exist(id) && self.stato.equals(stato) && self.idO.equals(id)
List<OrdineBean> Visualizzaordini(int stato)	Serve a visualizzare gli ordini	stato>=0 && stato <5	listaOrdini != null

Nome Classe **Context ProdottoDAO.java**			
Metodi	**Descrizione**	**Pre-Condition**	**Post-Condition**
ProdottoBean getPrdotto (int idPro)	Restituisce un prodotto in base all'id	idPro>0 && idPro!=null && self.idPro->exist(.idPro)	self.Prodotti-> exist(Prodotto.idPro)
void AddProdotto(int idP, List<IngredienteBean> ing, String nome, float prezzo, int stato, String foto)	Serve ad inserire un prodotto nel sistema	IdP > 0 && prezzo > 0 && foto != null && nome != null && Ing != null && ing.IngredienteId > 0 && ing.IsRimovibile != null && !self.idP->exist(idPro)	self.idP-> exist(idPro)
void DeleteProdotto(int idPro)	Cancella un prodotto del sistema	idPro!=null && idPro > 0 && self.idPro->exist(idPro)	!self.idP-> exist(idPro)
void UpdateProdotto(int idP, List<IngredienteBean> ing, String nome, float	Modifica un prodotto del sistema	IdP > 0 && prezzo > 0 && foto != null && nome != null &&	self.idP-> exist(idPro)

prezzo, int stato, String foto)		Ing != null && ing.IngredienteId > 0 && ing.IsRimovibile != null && self.idP->exist(idPro)	

4. Glossario

RAD: Requirements Analysis Document.
SDD: System Design Document.
ODD: Object Design Document.
NFR: Requisiti non funzionali.
UC: Use case.
FR: Requisiti funzionali.
DG: Design Goal
DBMS: Database Management System
HTTP: HyperText Transfer Protocol
JSON: JavaScript Object Notation