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

Kawaii-Comix

Object Design Document

Versione 1.0

Immagine che contiene fiore, rosa, Elementi grafici, testo

Descrizione generata automaticamente

Data: 09/01/2024

**Coordinatore del progetto**

**Partecipanti**

|  |  |
| --- | --- |
| Nome | Matricola |
| Simone D’Assisi | 0512113584 |
| Davide Del Franco Natale | 0512113233 |
| Giovanni Sicilia | 0512107458 |

|  |  |
| --- | --- |
| **Scritto da:** | Simone D’Assisi, Davide Del Franco Natale, Giovanni Sicilia |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 09/01/2024 | 1.0 | Prima stesura del documento | Simone D’Assisi |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Indice**

[1. Introduzione 4](#_Toc155720308)

[*1.1 Object design trade-offs* 4](#_Toc155720309)

[*1.1.1 Robustezza vs Tempo* 4](#_Toc155720310)

[*1.1.2 Attendibilità vs Tempo* 4](#_Toc155720311)

[*1.2 Linee guida* 4](#_Toc155720312)

[*1.3 Referenze* 4](#_Toc155720313)

[2. Directory 4](#_Toc155720314)

[*2.1* *src* 4](#_Toc155720315)

[2.2 *db* 5](#_Toc155720316)

[3. Pacchetti 5](#_Toc155720317)

[*3.1 view* 5](#_Toc155720318)

[3.1.1 view.site 5](#_Toc155720319)

[3.1.2 view.acquisto 5](#_Toc155720320)

[3.1.3 view.utente 5](#_Toc155720321)

[3.1.4 view.catalogo 5](#_Toc155720322)

[*3.2 acquistoManagement* 5](#_Toc155720323)

[*3.3 utenteManagement* 6](#_Toc155720324)

[*3.4 catalogoManagement* 6](#_Toc155720325)

[4. Interfacce di classe 7](#_Toc155720326)

# 1. Introduzione

## *1.1 Object design trade-offs*

### *1.1.1 Robustezza vs Tempo*

Il controllo dei dati in input è un aspetto importante del sistema, farlo però nella maniera più corretta e completa possibile richiederebbe un tempo di sviluppo maggiore rispetto a quello a nostra disposizione per il rilascio di una prima versione completamente funzionante. A discapito della robustezza, dunque, decidiamo di limitare i controlli sui dati in input, cosa che però sarà fatta nelle versioni successive.

### *1.1.2 Attendibilità vs Tempo*

L’attendibilità è un requisito fondamentale per il funzionamento del sistema, è infatti, ad esempio, importante gestire le transazioni in maniera corretta evitando che gli ordini possano essere effettuati senza avere la disponibilità dei prodotti richiesti. Chiaramente questo requisito necessita di maggior tempo di sviluppo per essere garantito, è infatti grazie al tempo guadagnato a discapito della robustezza che ci possiamo permettere di rispettarlo.

## *1.2 Linee guida*

Qui di seguito sono riportate alcune linee guide per la stesura del codice:

* Gli oggetto DAO devono essere nominati nomeentitàIDS.
* Gli errori devono essere gestiti tramite delle eccezioni e non dei valori di ritorno.

## *1.3 Referenze*

* **R.A.D.**
* **S.D.D.**

# 2. Directory

## *src*

* + 1. java

Questa directory deve contenere i pacchetti Java che compongono i codice del sistema.

* + 1. webapp
       1. META-INF

Questa directory deve contenere le meta-informazioni.

* + - 1. Script

Questa directory contiene i file JavaScript con numerose funzioni utilizzate dalle jsp.

* + - 1. Icons

Questa directory deve contenere le immagini utilizzate per icone o sfondi.

* + - 1. Images

Questa directory deve contenere le immagini utilizzate per le anteprime dei prodotti.

* + - 1. Styles

Questa directory deve contenere i fogli di stile.

* + - 1. Video

Questa directory deve contenere i video di presentazione.

## *db*

Questa directory deve contenere lo schema del database relazionale.

# 3. Pacchetti

In questa sezione analizzeremo la suddivisione in pacchetti di tutte le classi implementate, ecco una prima panoramica del sistema implementato:

//\*\*\* INSERIRE IMMAGINE \*\*\*/

## *3.1 view*

Questo pacchetto contiene tutti gli oggetti e le classi Java che compongono il layer di presentation del sistema.

### 3.1.1 view.site

Questo sottopacchetto contiene le classi Java adibite alla connessione al Database.

//\*\*\*INSERIRE IMMAGINE\*\*\*//

### 3.1.2 view.acquisto

Questo sottopacchetto contiene le Servlet adibite alle funzioni di acquisto del sito (e.g. carrello, pagamento etc.).

//\*\*\*INSERIRE IMMAGINE\*\*\*//

### 3.1.3 view.utente

Questo sottopacchetto contiene le Servlet adibite alle funzioni di gestione degli account nonché di autenticazione al sistema.

//\*\*\*INSERIRE IMMAGINE\*\*\*//

### 3.1.4 view.catalogo

Questo sottopacchetto contiene le Servlet e adibite alle funzioni di gestione del catalogo (e.g. visualizzazione catalogo)

//\*\*\*INSERIRE IMMAGINE\*\*\*//

## *3.2 acquistoManagement*

Questo pacchetto contiene le classi Java (Bean e DAO) adibite alle funzioni di acquisto (e.g. creazione di un nuovo ordine nel DB).

//\*\*\*INSERIRE IMMAGINE\*\*\*//

## *3.3 utenteManagement*

Questo pacchetto contiene le classi Java (Bean e DAO) adibite alle funzioni di gestione degli account e delle loro informazioni.

//\*\*\*INSERIRE IMMAGINE\*\*\*//

## *3.4 catalogoManagement*

Questo pacchetto contiene le classi Java (Bean e DAO) adibite alle funzioni di gestione del catalogo (tutte le informazioni sui prodotti, su dove sono immagazzinati etc.).

//\*\*\*INSERIRE IMMAGINE\*\*\*//

# 4. Interfacce di classe

* 1. *CarrelloIDS*

|  |  |
| --- | --- |
| **CarrelloIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità Carrello. |
| **Pre-condizione** | **context** CarrelloIDS:: doCreateCarrello(int userId)  **pre**: userId != null AND userId > 0  **context** CarrelloIDS:: doDeleteCarrello(int carrelloId)  **pre**: carrelloId!= null AND carrelloId > 0  **context** CarrelloIDS::doRetrieveCarrello(int userId);  **pre**: userId != null AND userId > 0  **context** CarrelloIDS::doRetrieveProdottiCarrello(Carrello carrello);  **pre**: carrello != null  **context** CarrelloIDS::doSvuotaCarrello(Carrello carrello);  **pre**: carrello != null AND !carrello.listaProdotti.isEmpty() |
| **Post-condizione** | **context** CarrelloIDS:: doCreateCarrello(int userId)  **post**: viene inserita una tupla nella table Carrello contenente le informazioni sul carrello e una tupla nella table CarrelloProdotto per ogni prodotto presente nel carrello all’interno del carrello  **context** CarrelloIDS:: doDeleteCarrello(int carrelloId)  **post**: vengono eliminate dal Database: la tupla corrispondente al carrello dalla table *Carrello* e le tuple che hanno carrelloId come chiave esterna dalla table *CarrelloProdotto*. Restituisce true se l’operazione è andata a buon fine, altrimenti false  **context** CarrelloIDS::doRetrieveCarrello(int userId);  **post**: restituisce il carrello appartenente all’utente con il campo id pari a userId se presente, altrimenti ne restituisce uno vuoto  **context** CarrelloIDS::doRetrieveProdottiCarrello(Carrello carrello);  **post**: restituisce un carrello con tutti i prodotti al suo interno, altrimenti restituisce un carrello vuoto  **context** CarrelloIDS::doSvuotaCarrello(Carrello carrello);  **post**: inserisce nel Database il carrello e tutti i prodotti in esso contenuti |
| **Invariante** |  |

* 1. *GestoreOrdiniIDS*

|  |  |
| --- | --- |
| **GestoreOrdiniIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità GestoreOrdini. |
| **Pre-condizione** | **context** GestoreOrdiniIDS::doSaveGestore(GestoreOrdini gestoreOrdini)  **pre**: gestoreOrdini != null AND  gestoreOrdini.nome != null AND gestoreOrdini.nome != “” AND  gestoreOrdini.cognome != null AND gestoreOrdini.cognome != “” AND  gestoreOrdini.email != null AND gestoreOrdini.email != “” AND  gestoreOrdini.password != null gestoreOrdini.password != “”  **context** GestoreOrdiniIDS::doUpdateGestore(GestoreOrdini gestoreOrdini)  **pre**: gestoreOrdini != null AND  gestoreOrdini.nome != null AND gestoreOrdini.cognome != null AND  gestoreOrdini.email != null AND gestoreOrdini.email != “” AND  gestoreOrdini.password != null AND gestoreOrdini.password != “”  **context** GestoreOrdiniIDS::doDeleteGestore(String email)  **pre**: email != null AND email != “”  **context** GestoreOrdiniIDS::doRetrieveByAuthentication(String email, String password)  **pre**: email != null AND email != “” AND password != null AND password != “” |
| **Post-condizione** | **context** GestoreOrdiniIDS::doSaveGestore(GestoreOrdini gestoreOrdini)  **post**: salva nel Database una tupla contenente le informazioni relative a un GestoreOrdini  **context** GestoreOrdiniIDS::doUpdateGestore(GestoreOrdini gestoreOrdini)  **post**: modifica le informazioni della tupla con email = gestoreOrdini.email con le nuove informazioni contenute in gestoreOrdini. Restituisce true se l’operazione è andata a buon fine, altrimenti false  **context** GestoreOrdiniIDS::doDeleteGestore(String email)  **post**: elimina la tupla contenente l’attributo email\_address = email. Restituisce true se l’operazione è andata a buon fine, altrimenti false  **context** GestoreOrdiniIDS::doRetrieveByAuthentication(String email, String password)  **post**: restituisce, se presente, il GestoreOrdini avente email\_address = email e passsword = password |
| **Invariante** |  |

* 1. *OrdineIDS*

|  |  |
| --- | --- |
| **OrdineIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità Ordine. |
| **Pre-condizione** | **context** OrdineIDS::doSaveOrdine(Ordine ordine)  **pre**: ordine != null AND ordine.id > 0 AND ordine.data != null AND  ordine.totale > 0.0 AND ordine.userId > 0 AND ordine.stato >= 1 AND  ordine.stato <= 3 AND ordine.metodoSpedizione >= 1 AND ordine.metodoSpedizione <= 3 AND !ordine.ordiniSingoli.isEmpty()  **context** OrdineIDS::doDeleteOrdine(Integer id)  **pre**: id > 0  **context** OrdineIDS:: doUpdateOrdine(Integer id, Date data, Double totale, Integer userId, Integer stato, Integer metodoSpedizione)  **pre**: id > 0 AND data != null AND totale > 0.0 AND userId > 0 AND  stato >= 1 AND stato <= 3 AND metodoSpedizione >= 1 AND  metodoSpedizione <= 3  **context** OrdineIDS::doRetrieveAllOrdini()  **pre**:  **context** OrdineIDS::doRetrieveById(Integer id)  **pre**: id > 0  **context** OrdineIDS:: doRetrieveAllOrdiniSingoli(Ordine ordine)  **pre**: ordine != null AND ordine.id > 0  **context** OrdineIDS::doSaveOrdineSingoloAssociato(OrdineSingolo ordineSingolo)  **pre**: ordineSingolo != null AND ordineSingolo.id > 0 |
| **Post-condizione** | **context** OrdineIDS::doSaveOrdine(Ordine ordine)  **post**: crea nel Database una tupla contenente le informazioni sull’ordine e su tutti gli OrdiniSingoli ad esso associati  **context** OrdineIDS::doDeleteOrdine(Integer id)  **post**: rimuove dal Database la tupla che presenta l’attributo *id* = id. Restituisce true se l’operazione è andata a buon fine, altrimenti false  **context** OrdineIDS::doUpdateOrdine(Integer id, Date data, Double totale, Integer userId, Integer stato, Integer metodoSpedizione)  **post**: aggiorna nel Database la tupla dell’ordine con attributi *id* = id. Restituisce true se l’operazione è andata a buon fine, altrimenti false  **context** OrdineIDS::doRetrieveAllOrdini()  **post**: restituisce la lista con tutti gli ordini presenti nel Database  **context** OrdineIDS::doRetrieveById(Integer id)  **post**: restituisce, se presente, l’ordine avente l’attributo  *id* = id  **context** OrdineIDS:: doRetrieveAllOrdiniSingoli(Ordine ordine)  **post**: restituisce la lista degli OrdiniSingoli associata all’ordine specifico  **context context** OrdineIDS::doSaveOrdineSingoloAssociato(OrdineSingolo ordineSingolo)  **post**: salva nel Database un singolo OrdineSingolo associato all’ordine |
| **Invariante** |  |

* 1. *OrdineSingoloIDS*
  2. *CategoriaIDS*
  3. *GenereIDS*
  4. *GestoreCatalogoIDS*

|  |  |
| --- | --- |
| **GestoreCatalogoIDS** | |
| **Descrizione** | Questa classe permette di interfacciarci al DBMS relazione e, in particolare, di accedere, modificare e interrogare l’entità GestoreCatalogo. |
| **Pre-condizione** | **context** GestoreCatalogoIDS::doSaveGestore(GestoreCatalogo gestoreCatalogo)  **pre**: gestoreCatalogo!= null AND  gestoreCatalogo.nome != null AND gestoreCatalogo.nome != “” AND  gestoreCatalogo.cognome != null AND gestoreCatalogo.cognome != “” AND  gestoreCatalogo.email != null AND gestoreCatalogo.email != “” AND  gestoreCatalogo.password != null gestoreCatalogo.password != “”  **context** GestoreCatalogoIDS::doUpdateGestore(GestoreCatalogo gestoreCatalogo)  **pre**: gestoreCatalogo!= null AND  gestoreCatalogo.nome != null AND gestoreCatalogo.cognome != null AND  gestoreCatalogo.email != null AND gestoreCatalogo.email != “” AND  gestoreCatalogo.password != null AND gestoreCatalogo.password != “”  **context** GestoreCatalogoIDS::doDeleteGestore(String email)  **pre**: email != null AND email != “”  **context** GestoreCatalogoIDS::doRetrieveByAuthentication(String email, String password)  **pre**: email != null AND email != “” AND password != null AND password != “” |
| **Post-condizione** | **context** GestoreCatalogoIDS::doSaveGestore(GestoreCatalogo gestoreOrdini)  **post**: salva nel Database una tupla contenente le informazioni relative a un GestoreOrdini  **context** GestoreCatalogoIDS::doUpdateGestore(GestoreCatalogo gestoreOrdini)  **post**: modifica le informazioni della tupla con email = gestoreCatalogo.email con le nuove informazioni contenute in gestoreCatalogo. Restituisce true se l’operazione è andata a buon fine, altrimenti false  **context** GestoreCatalogoIDS::doDeleteGestore(String email)  **post**: elimina la tupla contenente l’attributo email\_address = email. Restituisce true se l’operazione è andata a buon fine, altrimenti false  **context** GestoreCatalogoIDS::doRetrieveByAuthentication(String email, String password)  **post**: restituisce, se presente, il GestoreCatalogo avente email\_address = email e passsword = password |
| **Invariante** |  |

* 1. *ProdottoIDS*
  2. *UserIDS*