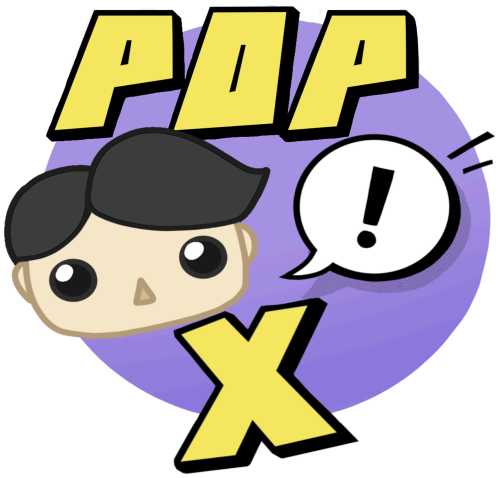
Università degli Studi di Salerno
Corso di Ingegneria del Software

Pop!x
Object Design Document
Versione 1.1



Data: 15/12/2024

Coordinatore del progetto:

Nome	Matricola
Scaparra Daniele Pio	0512116260

Partecipanti:

Nome	Matricola
Scaparra Daniele Pio	0512116260
Bonagura Grazia	0512116167
Nappi Antonio	0512117391
Nardiello Raffaele	0512118666

Scritto da:	Scaparra Daniele Pio, Bonagura Grazia, Nappi Antonio, Nardiello Raffaele

Revision History

Data	Versione	Descrizione	Autore
15/12/2024	1.0	Prima versione dell'Object Design Document	Scaparra Daniele Pio, Bonagura Grazia, Nappi Antonio, Nardiello Raffaele
26/12/2024	1.1	Ristrutturato l'Object Design Document	Scaparra Daniele Pio, Bonagura Grazia, Nappi Antonio, Nardiello Raffaele

Indice

1 Introduzione..... 4
1.1 Object Design Trade-offs..... 4
1.2 Linee guida per la documentazione dell'interfaccia..... 4
1.3 Definizioni, acronimi, abbreviazioni..... 4
1.4 Riferimenti..... 5
2 Packages..... 5
2.1 Struttura dei pacchetti..... 5
2.2 Struttura grafica dei packages nel sistema..... 5
3 Interfacce delle classi..... 7
Metodo: createUser..... 7
Metodo: modifyUser..... 7
Metodo: addProduct..... 7
Metodo: modifyProduct..... 8
Metodo: deleteProduct..... 8
Metodo: createOrder..... 9
Metodo: viewOrder..... 9
Metodo: manageOrder..... 9
Metodo: addToCart..... 10
Metodo: removeFromCart..... 10
Metodo: viewCart..... 10

1 Introduzione

1.1 Object Design Trade-offs

Il design di Pop!x si basa sul pattern MVC (Model-View-Controller) per garantire la separazione delle responsabilità.

- ***Buy vs Build***: Sono stati utilizzati framework e librerie open-source, come Bootstrap per il front-end e Tomcat per il server web, riducendo i tempi di sviluppo.
- ***Memory Space vs Response Time***: Sono stati ottimizzati i DAO per minimizzare le query al database e migliorare i tempi di risposta, accettando un maggiore utilizzo della memoria per memorizzare cache locali.

1.2 Linee guida per la documentazione dell'interfaccia

Le seguenti convenzioni sono state adottate per uniformare il design delle interfacce:

- **Naming:**
  - Classi con nomi al singolare e descrittivi (es. Prodotto, Ordine).
  - Metodi con frasi verbali che indicano l'azione svolta (es. aggiungiProdotto, validaPagamento).
  - Campi e parametri con nomi sostantivi (es. prezzo, idUtente).
- **Gestione degli errori:**
  - Gli errori sono segnalati tramite eccezioni.
  - Le operazioni su collezioni restituiscono oggetti di tipo List o Map robusti rispetto alla modifica degli elementi.
- **Conformità:** Tutte le interfacce devono essere documentate con JavaDoc e seguire il principio di interfacce minime.

1.3 Definizioni, acronimi, abbreviazioni

- **DAO:** Data Access Object, utilizzato per separare la logica di accesso ai dati.
- **DTO:** Data Transfer Object, per trasferire dati tra layer.
- **MVC:** Model-View-Controller, un pattern architetturale.

1.4 Riferimenti

- Riferimento al Requirements Analysis Document (RAD).
- Riferimento al System Design Document (SDD).
- Riferimento al Problem Statement Document.

2 Packages

2.1 Struttura dei pacchetti

Il sistema è suddiviso nei seguenti pacchetti:

1. **Presentation Layer:**
    - Pacchetto: com.popx.presentazione
    - Componenti principali: VistaUtente, VistaCatalogo, VistaCarrello, VistaOrdine.
2. **Service Layer:**
    - Pacchetto: com.popx.servizio
    - Componenti principali: ServizioAutenticazione, ServizioCheckout, ServizioProdotti.
3. **Persistence Layer:**
    - Pacchetto: com.popx.persistenza
    - Componenti principali: UtenteDAO, ProdottoDAO, OrdineDAO.

Ogni pacchetto è strutturato per ridurre le dipendenze e facilitare il riuso del codice.

2.2 Struttura grafica dei packages nel sistema

In questa sezione è descritta la struttura organizzativa dei files all'interno del progetto.
Si tratta di un'organizzazione di un progetto Java Web che utilizza Maven come gestore delle dipendenze.
Si può notare come il modello della suddivisione dei sottosistemi individuata nel documento di System Design viene pienamente rispettata andando ad essere implementata dai packages presentazione (Persistence Layer), servizio (Service Layer), persistenza (Persistence Layer).
All'interno di ognuno di questo package verranno realizzati i componenti indicati nel component diagram e rispetteranno la funzionalità loro assegnata.
Abbiamo anche i file e directory di Maven, come quello di target, il pom.xml e vari file di

configurazioni, esattamente come dovrebbe essere in un progetto che usa le tecnologie descritte (Java servlets, JSP, Bootstrap, Maven, HTML, CSS)

```
project-root/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── popx/
│   │   │           ├── presentazione/       // Controller e View
│   │   │           ├── servizio/            // Logica applicativa
│   │   │           ├── persistenza/         // DAO e repository
│   │   │           ├── modello/             // Classi di dominio (es. Prodotto, Ordine)
│   │   │           └── utils/               // Classi delle utilities
│   │   ├── resources/
│   │   │   ├── application.properties       // Configurazioni del progetto
│   │   │   └── templates/                   // Template JSP/HTML
│   │   └── webapp/
│   │       ├── WEB-INF/
│   │       │   ├── jsp/                     // JSP
│   │       │   └── web.xml                  // Configurazione servlet
│   └── test/
│       └── java/                        // Test unitari e di integrazione
├── target/                              // Directory generata da Maven
│   ├── classes/                         // Classi compilate
│   ├── generated-sources/               // Sorgenti generati
│   ├── test-classes/                    // Classi di test compilate
│   ├── popx-ecommerce-1.0-SNAPSHOT.jar  // Artefatto generato
└── pom.xml                              // File Maven principale
```

3 Interfacce delle classi

Metodo: createUser

Campo	Descrizione
Precondizioni	Il nome deve essere non vuoto (name <> ""), (email.matches("[^@]+@[^\\.]+\\..+")), (password.size() >= 8)
Postcondizioni	(result = true) implies User.allInstances()->exists(u | u.name = name and u.email = email)
Invarianti	Gli utenti devono avere univocità su email: User.allInstances()->isUnique(u | u.email)

Metodo: modifyUser

Campo	Descrizione
Precondizioni	Deve esistere un utente con l'identificativo fornito: User.allInstances()->exists(u | u.id = userId).
Postcondizioni	Se l'operazione ha successo (result = true), l'utente deve avere il nuovo nome e la nuova email: User.allInstances()->exists(u | u.id = userId and u.name = newName and u.email = newEmail).
Invarianti	Gli ordini devono rimanere associati correttamente agli utenti: Order.allInstances()->forAll(o | o.user.id = u.id).

Metodo: addProduct

Campo	Descrizione

Precondizioni	Il nome del prodotto non deve essere vuoto (name <> ""), il prezzo deve essere maggiore di zero (price > 0) lo stock deve essere maggiore/uguale a zero (stock >= 0).
Postcondizioni	Se l'operazione ha successo (result = true), il prodotto deve essere aggiunto al catalogo con i dettagli forniti: Catalog.allProducts->exists(p | p.name = name and p.price = price and p.stock = stock)
Invarianti	Catalog.allProducts->forAll(p | p.stock >= 0)

Metodo: modifyProduct

Campo	**Descrizione**
Precondizioni	Deve esistere un prodotto con l'identificativo fornito: Catalog.allProducts->exists(p | p.id = productId)
Postcondizioni	Se l'operazione ha successo (result = true), il prodotto deve avere i nuovi dettagli forniti: Catalog.allProducts->exists(p | p.id = productId and p.name = newName and p.price = newPrice and p.stock = newStock)
Invarianti	Order.allInstances()->forAll(o | o.items->excludes(p) implies p.id <> productId)

Metodo: deleteProduct

Campo	**Descrizione**
Precondizioni	Deve esistere un prodotto con l'identificativo fornito: Catalog.allProducts->exists(p | p.id = productId)
Postcondizioni	Se l'operazione ha successo (result = true), il prodotto deve essere rimosso dal catalogo: not Catalog.allProducts->exists(p | p.id = productId)

Invarianti	Cart.allCarts->forAll(c | c.items->excludes(p) implies p.id <> productId)

Metodo: createOrder

Campo	**Descrizione**
Precondizioni	(User.allInstances()->exists(u | u.id = userId)), ((cartItems->notEmpty())
Postcondizioni	(result = true) implies Order.allInstances()->exists(o | o.user.id = userId and o.items = cartItems)
Invarianti	Order.allInstances()->forAll(o | o.items->notEmpty())

Metodo: viewOrder

Campo	**Descrizione**
Precondizioni	Order.allInstances()->exists(o | o.id = orderId)
Postcondizioni	result = Order.allInstances()->any(o | o.id = orderId)
Invarianti	Order.allInstances()->forAll(o | User.allInstances()->exists(u | u.id = o.user.id))

Metodo: manageOrder

Campo	**Descrizione**
Precondizioni	Order.allInstances()->exists(o | o.id = orderId)
Postcondizioni	Order.allInstances()->exists(o | o.id = orderId and o.status = newStatus)
Invarianti	Order.allInstances()->forAll(o | o.status in Sequence{"Creato", "In lavorazione", "Completato"})

Metodo: addToCart

Campo	Descrizione
Precondizioni	(User.allInstances()->exists(u | u.id = userId)), (Catalog.allProducts->exists(p | p.id = productId and p.stock >= quantity))
Postcondizioni	Cart.allCarts->exists(c | c.user.id = userId and c.items->exists(i | i.product.id = productId and i.quantity = quantity))
Invarianti	Cart.allCarts->forAll(c | c.items->forAll(i | i.quantity <= i.product.stock))

Metodo: removeFromCart

Campo	Descrizione
Precondizioni	Cart.allCarts->exists(c | c.user.id = userId and c.items->exists(i | i.product.id = productId))
Postcondizioni	not Cart.allCarts->exists(c | c.user.id = userId and c.items->exists(i | i.product.id = productId))
Invarianti	Cart.allCarts->forAll(c | c.items->excludes(i | i.product.id = productId) implies i = null)

Metodo: viewCart

Campo	Descrizione
Precondizioni	User.allInstances()->exists(u | u.id = userId)
Postcondizioni	result = Cart.allCarts->any(c | c.user.id = userId).items
Invarianti	Order.allInstances()->forAll(o | o.user.id = c.user.id implies c.items->isEmpty())