UNIVERSITÀ DEGLI STUDI DI SALERNO

Music-Store

Object Design Document

Versione 1.0

Partecipanti

Nome	Cognome	Matricola
Vittorio	Ardolino	0512103772
Emanuele	Galati	0512105340
Nicola Antonio	Buzzo	0512105304

Revisioni del documento

Data	Versione	Cambiamenti	Autori
11/01/2020	0.1	Introduzione	Emanuele Galati Vittorio Ardolino Nicola Buzzo
16/01/2020	0.2	Design Pattern	Emanuele Galati Vittorio Ardolino
20/01/2020	0.3	Packages	Emanuele Galati Vittorio Ardolino
22/01/2020	0.4	Interfaccia delle classi	Emanuele Galati Vittorio Ardolino
18/02/2020	1.0	Revisione totale e correzione	Emanuele Galati Vittorio Ardolino Nicola Buzzo

Sommario

1. Introduzione

1.1. Object Design Trade-offs

Terminata la realizzazione dei documenti RAD e SDD abbiamo descritto gli obiettivi che ci aspettiamo ottenere dal sistema e, per grosse linee, tralasciando l'aspetto implementativo, anche come esso sarà formato.

Questo documento nasce dall'esigenza di realizzare in grado di mostrare in maniera precisa e coerente le funzionalità descritte nelle fasi subito precedenti.

Comprendibilità vs Tempo

Il codice deve essere scritto nella maniera più chiara e comprensibile possibile, dove ogni implementazione è accompagnata da un opportuno commento che porterà dei rallentamenti durante l'implementazione e la fase di testing, ma ci saranno notevoli vantaggi sulla comprensione globale del sistema.

Prestazioni vs Costi

Non disponendo di sovvenzioni esterne al progetto, il sistema sarà realizzato sfruttando le tecnologie open-source che garantiranno la sua funzionalità anche in maniera totalmente gratuita. Più nello specifico, sarà utilizzato un database relazionale per gestire i dati del sistema ed un web server monilitico per l'interazione con gli utenti.

Interfaccia vs Usabilità

L'interfaccia sarà realizzata rispettando i canoni dello "user-friendly": il più limpida e pulita possibili, con l'impiego di bottoni e form di facile comprensione per l'utente.

Sicurezza vs Efficienza

Il sistema garantirà sicurezza agli utenti per evitare accessi non autorizzati per proteggere informazioni personali e dati sensibili quali emails, password e dati relativi ai metodi di pagamento.

1.2. Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice:

Naming Convention

I nomi utilizzati per identificare i concetti principali, le funzionalità e le componenti generiche del sistema dovranno rispettare tali condizioni:

1. Appartenenti alla lingua italiana dove possibile
2. Di lunghezza medio-breve
3. Non sostituiti da acronimi o abbreviazioni
4. Composti da caratteri compresi tra [0-9, a-z, A-Z]

Le variabili devono:

1. Rispettare la Camel Notation
2. Iniziare con carattere minuscolo
3. Essere composti da caratteri compresi tra [0-9, a-z, A-Z]

Esempio: zipCode

Le classi e le interfacce devono:

1. Rispettare la Camel Notation
2. Iniziare con carattere Maiuscolo
3. Concludersi con il tipo di elemento che rappresentano

```
1.  public class User {
2.
3.          private int id;
4.          private String name;
5.          private String surname;
6.          private String email;
7.          private int type;
8.
9.          public User() {
10.         }
11.
12.         public User(int id, String name, String surname, String email, int type) {
13.
14.                 this.id=id;
15.                 this.name=name;
16.                 this.surname=surname;
17.                 this.email=email;
18.                 this.type=type;
19.
20.         }
21.
22.         public int getId() {
23.                 return id;
```

```
24.             }
25.             public void setId(int  id) {
26.                     this.id = id;
27.             }
28.             public String getName() {
29.                     return name;
30.             }
31.             public void setName(String name) {
32.                     this.name = name;
33.             }
34.             public String getSurname() {
35.                     return surname;
36.             }
37.             public void setSurname(String surname) {
38.                     this.surname = surname;
39.             }
40.             public String getEmail() {
41.                     return email;
42.             }
43.             public void setEmail(String email) {
44.                     this.email = email;
45.             }
46.             public int getType() {
47.                     return type;
48.             }
49.             public void setType(int  type) {
50.                     this.type = type;
51.             }
52.
53.  }
```

I pacchetti devono:

1. Contenere solamente caratteri minuscoli
2. Contenere solamente caratteri compresi tra a-z

I metodi devono:

1. Iniziare con carattere minuscolo
2. Rispettare la Camel Notation
3. Evitare di cominciare con GET o SET se non si tratta di metodi getting o setting corrispondenti
4. Contenere solamente caratteri [a-z, A-Z]

esempio. getNomeVariabile() e setNomeVariabile().

Le pagine JSP:

1. Contengono solamente caratteri minuscoli
2. Sono di lunghezza medio-breve
3. Hanno nomi composti da una singola parola in inglese o in italiano

Gestione formattazione del codice

Per rimanere conformi a specifiche definite nella sezione precedente, bisogna approcciarsi con una certa attenzione all'indentazione del codice ed alla formattazione di alcuni elementi.

Gestione codice HTML e XML

```
<html>
        <head>
        <head>
    <body>
            <div>
                Contenuto DIV
            <div>
    <body>
<html>
```

Il codice HTML deve essere indentato in maniera tale da far corrispondere sulla stessa colonna il tag di apertura e chiusura , inoltre, il contenuto di un tag si distanzierà dalle clausole dell'elemento in cui è contenuto di una distanza pari ad 1 TAB.

1.3 Definizione, acronimi e abbreviazioni

Acronimi:

- **SDD:** System Design Document
- **ODD:** Object Design Document
- **RAD:** Requirements Analysis Document

Abbreviazioni:

- **DB:** Database
- **DBMS:** Database Management System

Definizioni:

- **Servlet:** Classi ed Oggetti Java per la gestione di operazioni su un Web Server

1.4 Riferimenti

Il contesto descritto all'interno di questo documento è ripreso dal **RAD** e dall' **SDD** del progetto **Music-Store**.

2. Packages

L'architettura del nostro sistema è three-tier, quindi suddivisa in tre livelli:

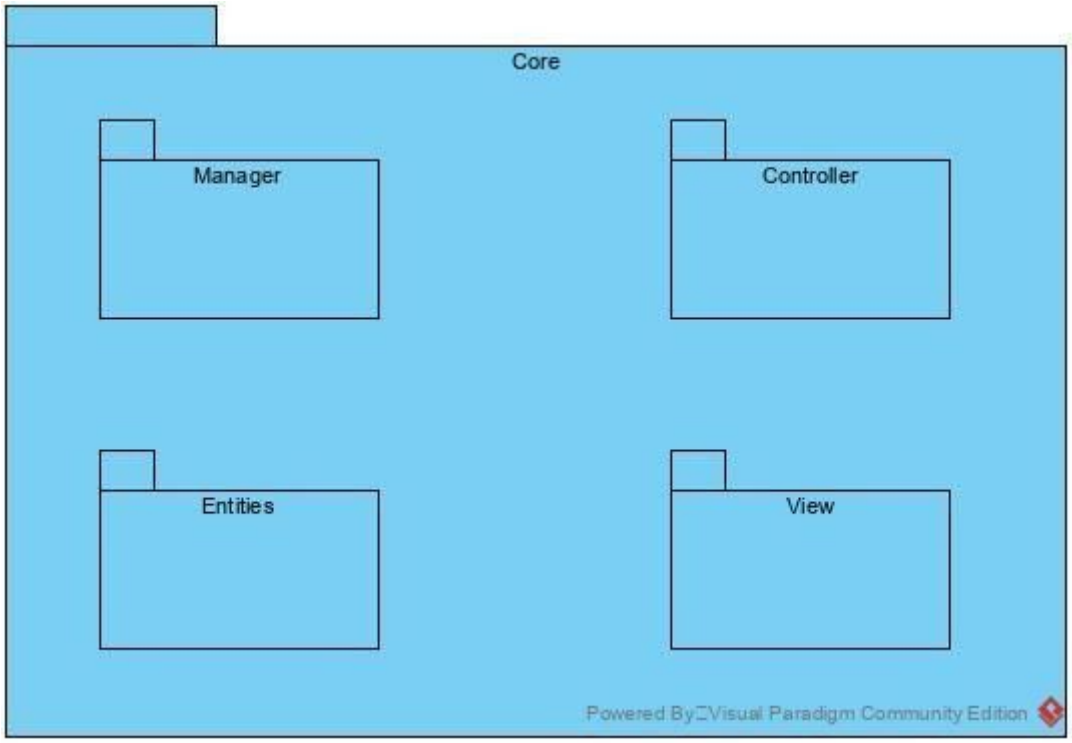
- Presentation layer
- Application layer
- Storage layer

Presentation Layer	Include tutte le interfacce grafiche, quindi i boundary object, come i form con cui l'utente interagisce. L'interfaccia verso l'utente è rappresentata da un Web Server e da altri eventuali contenuti statici.

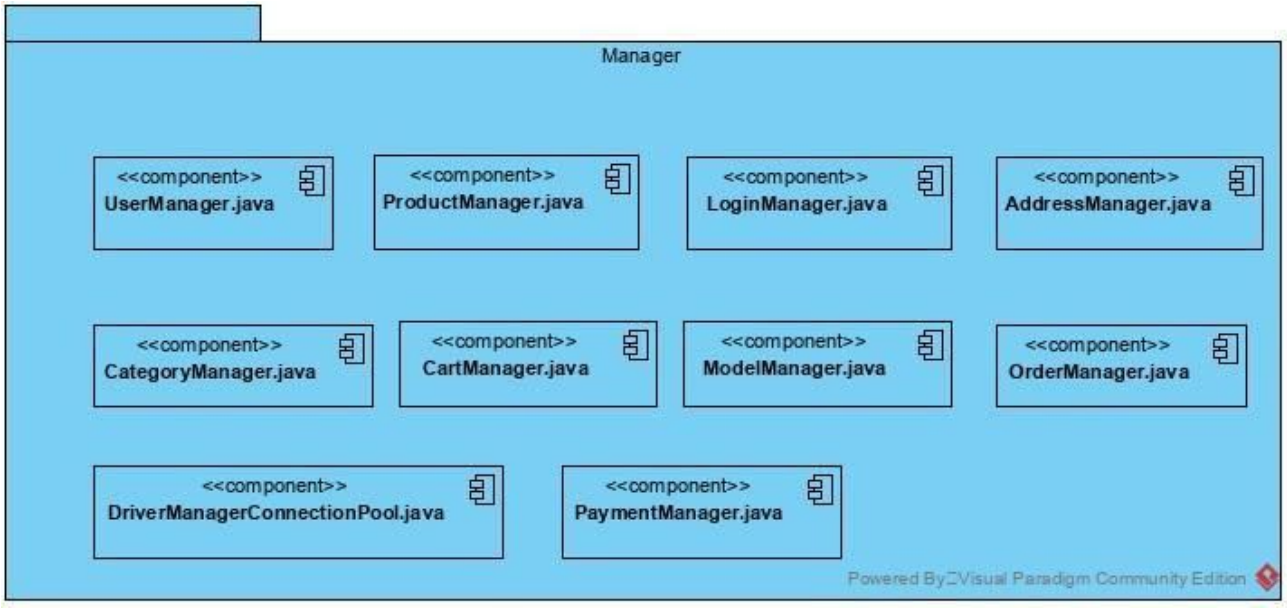
Application Layer	Include tutti gli oggetti relativi al controllo e all'elaborazione dei dati, attraverso l'interrogazione di un database tramite lo Storage Layer per accedere ai dati persistenti e generare contenuti dinamici. Si occupa di gestire: ● Gestione autenticazione ● Gestione utenti ● Gestione ordini ● Gestione prodotti ● Gestione metodi di pagamento ● Gestione indirizzo ● Gestione categoria ● Gestione registrazione ● Gestione carrello

Storage Layer	Memorizza, recupera e interroga gli oggetti persistenti. I dati che possono essere acceduti dall' Application Layer, sono organizzati, tramite DBMS, in maniera persistente.

2.1.0 Package core

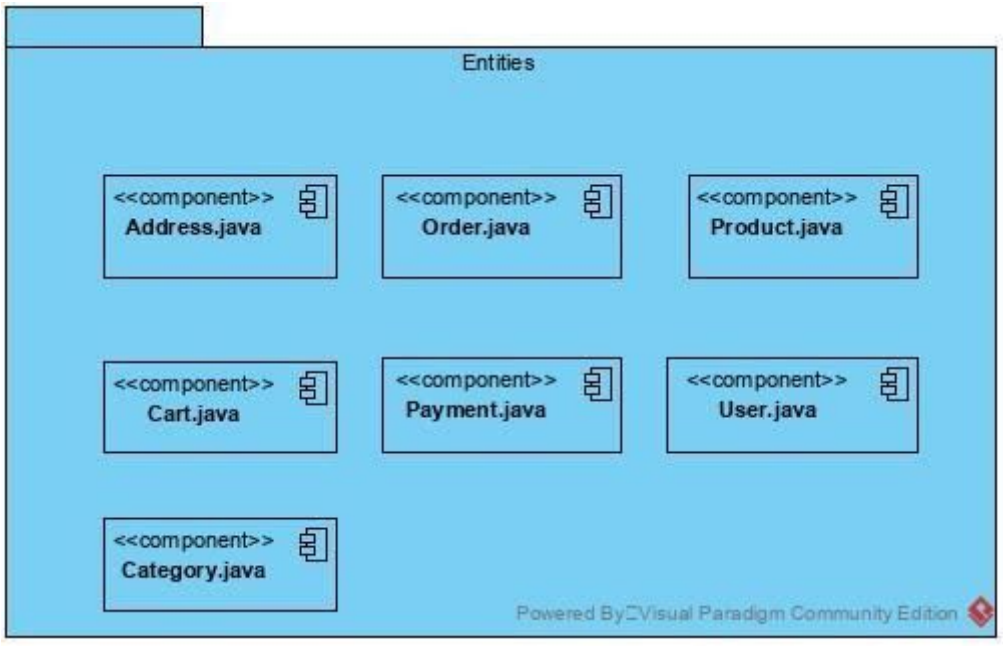


2.1.1 Package Manager



Classe	Descrizione
LoginManager.java	Gestisce l'autenticazione del sistema.
UserManager.java	Permette di interagire con la tabella User all'interno del database
ProductManager.java	Permette di interagire con la tabella Product all'interno del database
AddressManager.java	Permette di interagire con la tabella Address all'interno del database
CategoryManager.java	Permette di interagire con la tabella Category all'interno del database
CartManager.java	Permette di interagire con la tabella Cart all'interno del database
OrderManager.java	Permette di interagire con la tabella Order all'interno del database
PayementManager.java	Permette di interagire con la tabella Payment all'interno del database
ModelManager.java	Interfaccia che dichiara i metodi CRUD
DriverManagerConnectionPool	Permette la connesione al DB.

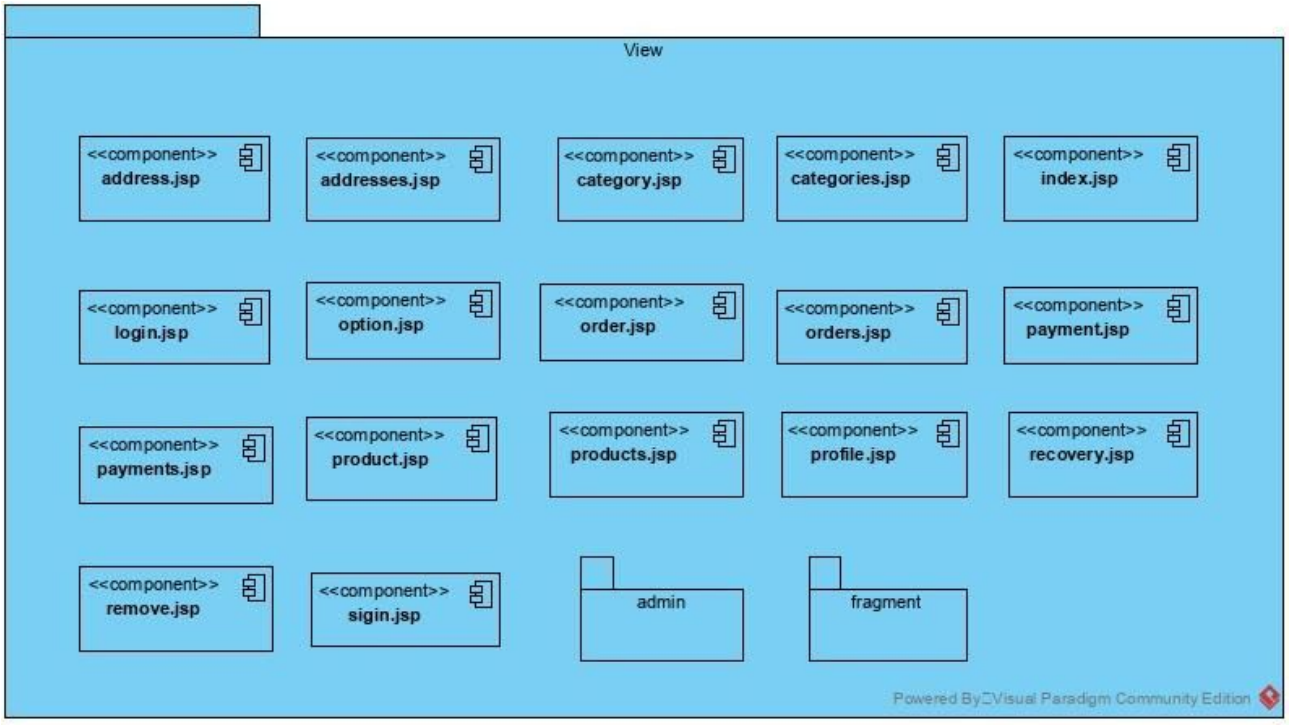
2.1.2 Package Entities



Classe	Descrizione
User.java	Descrive un utente nel sistema.
Address.java	Descrive un indirizzo nel sistema.
Order.java	Descrive un ordine nel sistema.
Product.java	Descrive un prodotto nel sistema.
Cart.java	Descrive un carrello nel sistema.

Payment.java	Descrive un metodo di pagamento nel sistema.
Category.java	Descrive una categoria nel sistema.

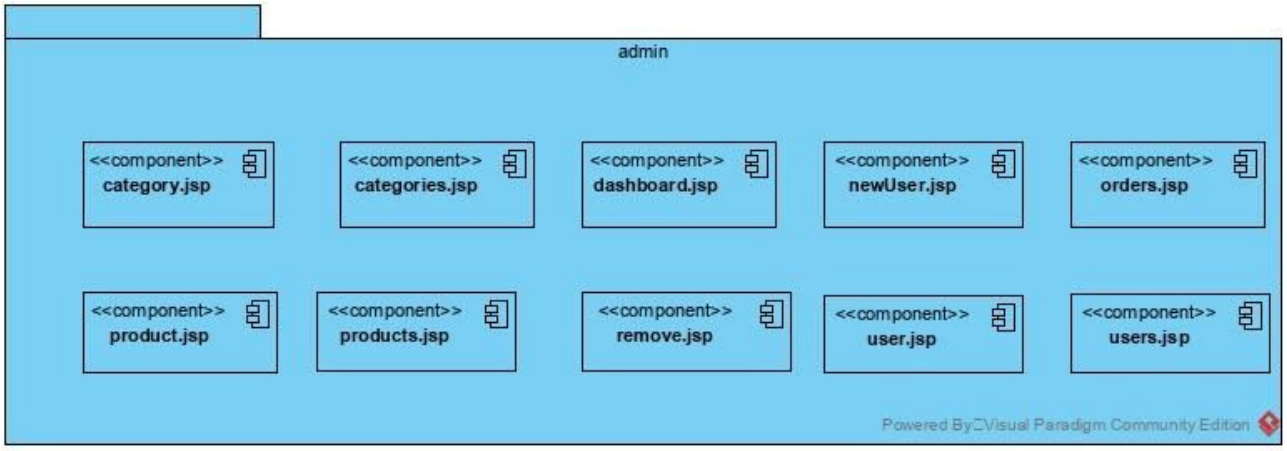
2.1.3 Package View



Classe	**Descrizione**
address.jsp	Visualizza un form per creare o modificare un nuovo ordine
addresses.jsp	Visualizza una lista degli indirizzi di un utente
categories.jsp	Visualizza una lista delle categorie accessibili dall'utente

category.jsp	Visualizza i dettagli di una categoria
index.jsp	Visualizza la pagina principale del sistema senza aver effettuato l'accesso.
login.jsp	Visualizza la pagina di login.
order.jsp	Visualizza il form per modificare o creare un nuovo ordine
orders.jsp	Visualizza una pagina con tutti gli ordini dell'utente
payment.jsp	Visualizza un form per l'inserimento di una nuova carta di pagamento
payments.jsp	Visualizza la lista dei metodi di pagamento di un utente
product.jsp	Visualizza una pagina con i dettagli del prodotto
products.jsp	Visualizza una pagina con la lista dei prodotti
profile.jsp	visualizza un form con i dati dell'utente con la possibilità di modificarli
recovery.jsp	Mostra un form in cui è possibile richiedere nuovi dati di accesso
remove.jsp	Visualizza una pagina in cui viene chiesta la conferma di rimozione di un indrizzo, la propria utenza o un metodo pagamento
sigin.jsp	Visualizza la pagina di registrazione al sistema da parte di un cliente.
checkout.jsp	Visualizza i dettagli di un acquisto che si intende effettuare
cart.jsp	Visualizza il carrello dell'utente

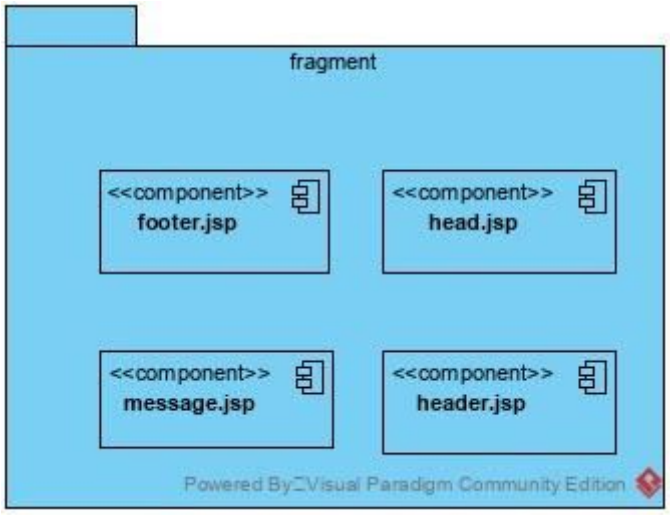
2.1.3.1 Package admin



Classe	Descrizione
categories.jsp	Visualizza un elenco di tutte le categorie
category.jsp	Visualizza un form per creare o modificare una categoria
dashboard.jsp	Visualizza la home page di un amministratore con le relative opzioni in base alle funzionalità
newUser.jsp	Visualizza un form per la creazione di un nuovo utente
orders.jsp	Visualizza una lista di tutti gli ordini
product.jsp	Visualizza un form per creare o modificare un ordine
products.jsp	Visualizza una lista con tutti i prodotti presenti nel sistema
remove.jsp	Visualizza una pagina in cui viene

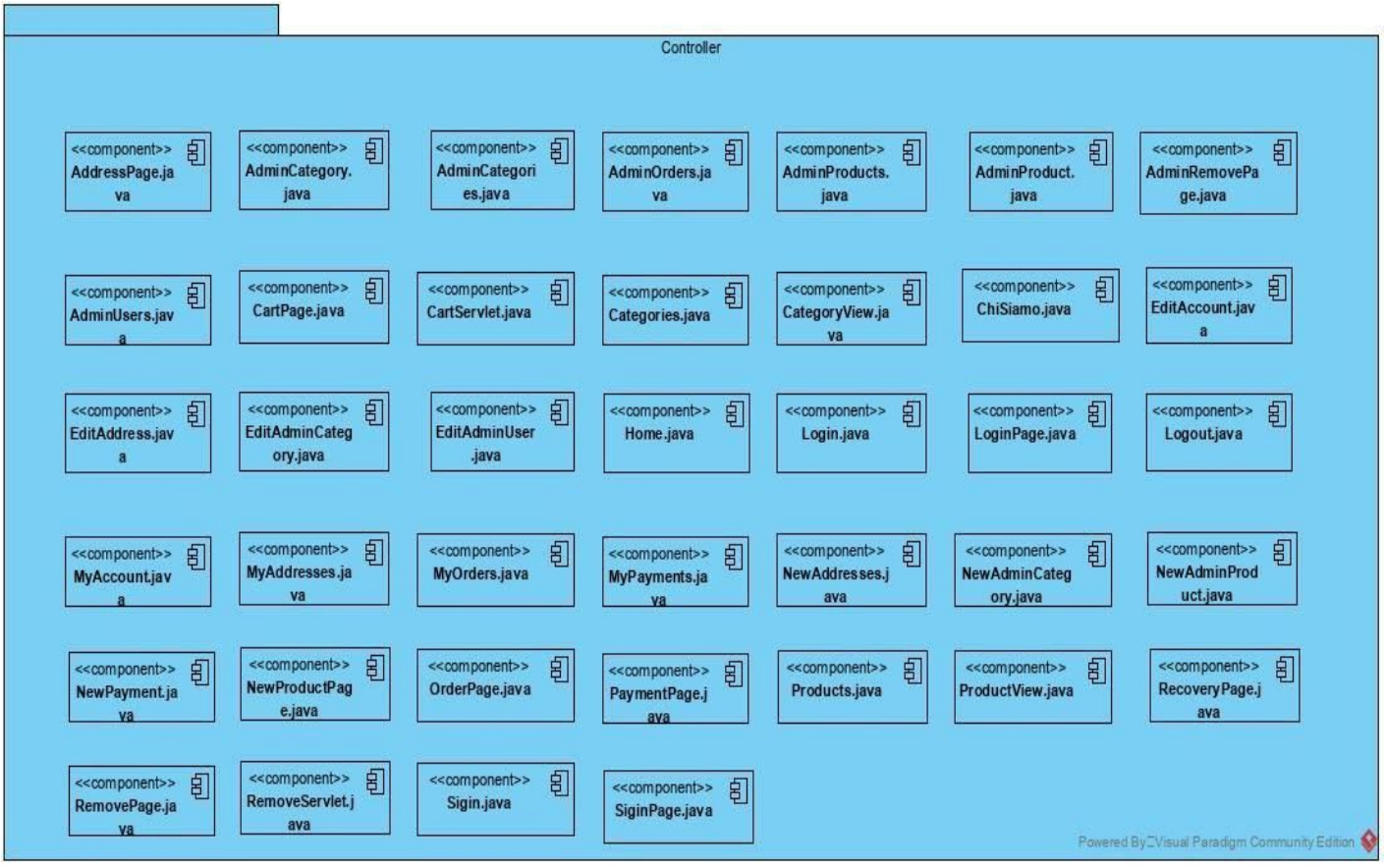
	chiesta la conferma di rimozione di un prodotto, un ordine, un utente o una categoria dal sistema
user.jsp	Visualizza un form per creare o rimuovere un utente dal sistema
users.jsp	Visualizza un elenco di tutti gli utenti presenti nel sistema

2.1.3.2 Package fragment



Classe	**Descrizione**
footer.jsp	Contiene il footer del sistema.
head.jsp	Contiene l'intestazione delle pagine
header.jsp	Contiene l'header e la navbar del sito
message.jsp	Contiene una sezione dedicata ai messaggi all'interno delle pagine

2.1.4 Package Controller



Classe	Descrizione

AddressPage.java	Ottiene i dettagli di un indirizzo e reindirizza alla pagina dell'indirizzo per mostrarne i dettagli oppure alla pagina per crearne uno nuovo.
AdminCategory.java	Se presenti, inoltra i dettagli di una categoria alla relativa pagina di modifica in caso contrario inoltra alla pagina di creazione della categoria amministrativa
AdminCategories.java	Carica un elenco di tutte le Categorie presenti nel sistema e le inoltra alla relativa jsp amministrava
AdminOrders.java	Carica un elenco di tutti gli ordini presenti nel sistema e li inoltra alla relativa jsp amministrativa
AdminProducts.java	Carica un elenco di tutti gli ordini presenti nel sistema e li inoltra alla relativa jsp amministrativa
AdminProduct.java	Se vi sono dati relativi al prodotto presenti reindirizza alla pagina di modifica del prodotto, in caso contrario alla pagina di creazione per un nuovo prodotto
AdminRemovePage.java	Questa pagina seleziona l'oggetto da rimuovere e lo reindirizza alla pagina di conferma rimozione
AdminUsers.java	Carica i dati di tutti gli utenti presenti nel sistema e li reindirizza alla relativa jsp amministrativa
CartPage.java	Reindirizza i dati del carrello presenti in sessione alla relativa pagina jsp
CartOperation.java	Seleziona l'operazione da effettuare relativa al carrello e la esegue indirizzando alla pagina che mostra il carrello

Categories.java	Ottiene tutti i dati delle categorie e li invia alla jsp che li mostra
CategoryView.java	Ottiene tutti i dati di una categoria e li indirizza alla jsp che ne mostra i dettagli
EditAccount.java	Ottiene i dati dell'account immessi da una form e li inserisce nel sistema
EditAddress.java	Ottiene i dati dell'indirizzo immessi da una form e li aggiorna con quelli presenti nel sistema
EditAdminCategory.java	Ottiene i dati della categoria immessa in un form e li aggiorna con quelli presenti sistema
EditAdminUser.java	Ottiene i dati dell'utente immessi in una form e li aggiorna con quelli presenti nel sistema
Home.java	Indirizza alle varie home page esistenti in base ai dettagli del tipo dell'utente
Login.java	Prende i dati dell'utente e li inserisce in sessione e reindirizza alla home page
LoginPage.java	Visualizza la pagina del login
Logout.java	Rimuove tutti i dettagli della sessione e reindirizza alla home page non loggata
MyAccount.java	Carica i dettagli dell'utente in sessione e li indirizza alla pagina di profilo
MyAddresses.java	Carica i dettagli degli indirizzi e li invia ad una pagina che li mostra
MyOrders.java	Carica i dettagli degli ordini e li invia ad una pagina che li mostra
MyPayments.java	Carica i dettagli dei pagamenti e li invia ad una pagina che li mostra

NewAddresses.java	Inserisce nel sistema i dettagli dell'indirizzo che riceve
NewAdminCategory.java	Inserisce nel sistema i dettagli della categoria che riceve
NewAdminProduct.java	Inserisce nel sistema i dettagli del prodotto che riceve
NewPayment.java	Inserisce nel sistema i dettagli del pagamento che riceve
NewProductPage.java	Inserisce nel sistema i dettagli del prodotto che riceve
OrderPage.java	Se riceve i dati di un ordine li reindirizza alla pagina di modifica in caso contrario a quella di creazione di un nuovo ordine
PaymentPage.java	Se riceve i dati di un ordine li reindirizza alla pagina di modifica in caso contrario
Products.java	Carica i dettagli degli ordini del sistema e li reindirizza alla pagina che li mostra
ProductView.java	Carica i dettagli degli ordini dal sistema e li reindirizza alla pagina che li mostra
RecoveryPage.java	Reindirizza alla pagina di recupero dei dati di accesso
RemovePage.java	Reindirizza alla pagina di conferma di rimozione oggetto
RemoveServlet.java	Rimuove l'oggetto che riceve dal sistema
Sigin.java	Inserisci un nuovo utente di tipo cliente nel sistema
SiginPage.java	Indirizza alla pagina di registrazione al sito
Checkout.java	Reindirizza alla pagina di checkout

Success.java	Effettua il checkout e crea un nuovo ordine con i dati presenti nel checkout

3. Interfaccia delle classi

Package Manager

3.1.0 Manager Autenticazione

Classe che funge da interfaccia del sottosistema per gestire l'autenticazione.

- Public boolean checkClient()

Questo metodo permette di verificare se l'utente è un cliente. Restituisce true se lo è altrimenti false.

- Precondizione: user != null
- Postcondizione: il metodo ritorna true se un utente è un cliente e quindi il tipo è 0 altrimenti false
- Invariante: il database deve essere funzionante

- Public boolean `checkOrderAdmin()`

Questo metodo permette di verificare se l'utente è un Gestore degli ordini. Restituisce true se lo è altrimenti false.

- Precondizione: user != null
- Postcondizione: : il metodo ritorna true se un utente è un Gestore Ordini e quindi il tipo è uguale a 1 altrimenti false
- Invariante: il database deve essere funzionante

- Public boolean `checkCatalogAdmin()`

Questo metodo permette di verificare se l'utente è un Gestore catalogo. Restituisce true se lo è altrimenti false.

- Precondizione: user != null
- Postcondizione: : il metodo ritorna true se un utente è un Gestore Catalogo e quindi il tipo è uguale a 2 altrimenti false
- Invariante: il database deve essere funzionante

- Public boolean `checkUserAdmin()`

Questo metodo permette di verificare se l'utente è un Gestore degli utenti. Restituisce true se lo è altrimenti false.

- Precondizione: user != null
- Postcondizione: : il metodo ritorna true se un utente è un Gestore Utenti e quindi il tipo è uguale a 3 altrimenti false
- Invariante: il database deve essere funzionante

- Public boolean `checkAdmin()`

Questo metodo permette di verificare se l'utente è un Gestore degli utenti. Restituisce true se lo è altrimenti false.

- Precondizione: user != null
- Postcondizione: : il metodo ritorna true se un utente è un Gestore Utenti, un Gestore Catalogo o Gestore Utenti, quindi il tipo uguale a 1, 2 o 3
- Invariante: il database deve essere funzionante

3.1.2 Manager Utenti

Classe che funge da interfaccia del sottosistema per gestire degli utenti.

- Public User doRetriveByKey(String key)

Questo metodo permette di cercare un utente tramite il suo id (key)

- Precondizione: key != null
- Postcondizione: il metodo ritorna un User, se presente nel database
- Invariante: il database deve essere funzionante

- Public User `doRetriveByEmailandPassword` (String email, String password)

Questo metodo permette di cercare un utente tramite la sua email e la sua password

- Precondizione: email != null
- Postcondizione: il metodo ritorna un User, se presente nel database
- Invariante: il database deve essere funzionante

- Public User `getUserPassword` (int userId)

Questo metodo permette di ottenere la password dell' utente

- Precondizione: userId > -1
- Postcondizione: il metodo ritorna la password, se presente nel database
- Invariante: il database deve essere funzionante

- Public synchronized void doSave(User user, String password)

Questo metodo permette di aggiungere un nuovo amministratore

- Precondizione: user != null
- Postcondizione: l'utente deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

- Public Collection<User> doRetriveAll(String orderBy)

Questo metodo permette di visualizzare l'elenco di tutti gli utenti

- Precondizione: orderBy != null
- Postcondizione: il metodo ritorna una lista di utenti presenti nel database
- Invariante: il database deve essere funzionante

- Public void doDelete(User user)

Questo metodo permette di eliminare un utente dal database

- Precondizione: user != null
- Postcondizione: l'utente viene eliminato se esiste nel database
- Invariante: il database deve essere funzionante

- Public void doUpdate(User user)

Questo metodo permette di modificare un utente dal database

- Precondizione: user != null
- Postcondizione: l'utente viene modificato se esiste nel database

- Invariante: il database deve essere funzionante

3.1.3 Manager Ordini

Classe che funge da interfaccia del sottosistema per gestire gli ordini.

- Public Order doRetriveByKey(String key)

Questo metodo permette di cercare un ordine tramite il suo id (key)

- Precondizione: key != null
- Postcondizione: il metodo ritorna un Order, se presente nel database
- Invariante: il database deve essere funzionante

- Public Collection<Order> doRetriveAll(String orderBy)

Questo metodo permette di visualizzare l'elenco di tutti gli ordini

- Precondizione: orderBy != null
- Postcondizione: il metodo ritorna una lista di ordini presenti nel database
- Invariante: il database deve essere funzionante

- Public synchronized void doSave(Order order)

Questo metodo permette di aggiungere un nuovo ordine

- Precondizione: order != null
- Postcondizione: l'ordine deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

- Public void doUpdate(Order order)

Questo metodo permette di modificare un ordine dal database

- Precondizione: order != null
- Postcondizione: l'ordine viene modificato se esiste nel database

- Invariante: il database deve essere funzionante

- Public void doDelete(Order order)

Questo metodo permette di eliminare un ordine dal database

- Precondizione: order != null
- Postcondizione: l'ordine viene eliminato se esiste nel database
- Invariante: il database deve essere funzionante

- Public Collection<Order> doRetriveAllByUser(User user)

Questo metodo permette di cercare tutti gli ordini di un utente

- Precondizione: user != null
- Postcondizione: il metodo restituisce la lista di ordini legati ad un utente
- Invariante: il database deve essere funzionante

3.1.4 Manager Prodotti

Classe che funge da interfaccia del sottosistema per gestire i prodotti.

- Public Product doRetriveByKey(String key)

Questo metodo permette di cercare un prodotto tramite il suo id (key)

- Precondizione: key != null
- Postcondizione: il metodo ritorna un Product, se presente nel database
- Invariante: il database deve essere funzionante

- Public Collection<Product> doRetriveAll(String orderBy)

Questo metodo permette di visualizzare l'elenco di tutti i prodotti

- Precondizione: orderBy != null
- Postcondizione: il metodo ritorna una lista di prodotti presenti nel database

- Invariante: il database deve essere funzionante

- Public synchronized void doSave(Product product)

Questo metodo permette di aggiungere un nuovo prodotto

- Precondizione: product != null
- Postcondizione: il prodotto deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

- Public void doUpdate(Product product)

Questo metodo permette di modificare un prodotto dal database

- Precondizione: product != null
- Postcondizione: il prodotto viene modificato se esiste nel database
- Invariante: il database deve essere funzionante

- Public void doDelete(Product product)

Questo metodo permette di eliminare un prodotto dal database

- Precondizione: product != null
- Postcondizione: il prodotto viene eliminato se esiste nel database
- Invariante: il database deve essere funzionante

- Public Collection<Product> getAllByCategory(Category category)

Questo metodo permette di ottenere una lista di prodotti legati ad una categoria

- Precondizione: category != null
- Postcondizione: il metodo ritorna la lista di prodotti della categoria
- Invariante: il database deve essere funzionante

- Public Collection<Product> doRetriveAllByCart(Cart cart)

Questo metodo permette di visualizzare una lista di prodotti legati ad un carrello

- Precondizione: cart != null
- Postcondizione: il metodo ritorna la lista di prodotti del carrello
- Invariante: il database deve essere funzionante

3.1.5 Manager Pagamento

Classe che funge da interfaccia del sottosistema per gestire i metodi di pagamento.

- Public Payement doRetriveByKey(String code)

Questo metodo permette di cercare un prodotto tramite il suo id (code)

- Precondizione: code != null
- Postcondizione: il metodo ritorna un Payement, se presente nel database
- Invariante: il database deve essere funzionante

- Public Collection<Payement> doRetriveAll(String orderBy)

Questo metodo permette di visualizzare l'elenco di tutti i pagamenti

- Precondizione: orderBy != null
- Postcondizione: il metodo ritorna una lista di pagamenti presenti nel database
- Invariante: il database deve essere funzionante

- Public synchronized void doSave(Payement payement)

Questo metodo permette di aggiungere un metodo di pagamento

- Precondizione: payement!= null
- Postcondizione: il metodo di pagamento deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

- Public void doDelete(Payement payement)

Questo metodo permette di eliminare un metodo di pagamento dal database

  - Precondizione: payement!= null
  - Postcondizione: il metodo di pagamento viene eliminato, se esiste nel database
  - Invariante: il database deve essere funzionante

- Public Collection<Payement> doRetriveByUser(User user) DA VEDERE

Questo metodo permette di visualizzare la lista di metodi di pagamento legati ad un utente

  - Precondizione: user != null
  - Postcondizione: il metodo visualizza il metodi di pagamento legati ad un utente
  - Invariante: il database deve essere funzionante

3.1.6 Manager Indirizzo

Classe che funge da interfaccia del sottosistema per gestire gli indirizzi.

- Public Address doRetriveByKey(String key)

Questo metodo permette di cercare un prodotto tramite il suo id (key)

  - Precondizione: key != null
  - Postcondizione: il metodo ritorna un Address, se presente nel database
  - Invariante: il database deve essere funzionante

- Public Collection< Address > doRetriveAll(String orderBy)

Questo metodo permette di visualizzare l'elenco di tutti i indirizzi

  - Precondizione: orderBy != null
  - Postcondizione: il metodo ritorna una lista di indirizzi presenti nel database
  - Invariante: il database deve essere funzionante

- Public synchronized void doSave(Address address)

Questo metodo permette di aggiungere un nuovo indirizzo

- Precondizione: address != null
- Postcondizione: il nuovo indirizzo deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

- Public void doUpdate(Address address)

Questo metodo permette di modificare un indirizzo dal database

- Precondizione: address != null
- Postcondizione: l'indirizzo viene modificato se esiste nel database
- Invariante: il database deve essere funzionante

- Public void doDelete(Address address)

Questo metodo permette di eliminare un indirizzo dal database

- Precondizione: address != null
- Postcondizione: l'indirizzo viene eliminato, se esiste nel database
- Invariante: il database deve essere funzionante

- Public Collection< Address > doRetriveByUser(User user) DA VEDERE

Questo metodo permette di visualizzare la lista di indirizzi legati ad un utente

- Precondizione: user != null
- Postcondizione: il metodo visualizza gli indirizzi legati ad un utente
- Invariante: il database deve essere funzionante

3.1.7 Manager Categoria

Classe che funge da interfaccia del sottosistema per gestire le categorie.

- Public Category doRetriveByKey(String key)

Questo metodo permette di cercare una categoria tramite il suo nome (key)

- Precondizione: key != null
- Postcondizione: il metodo ritorna un Category, se presente nel database
- Invariante: il database deve essere funzionante

- Public Collection<Category> doRetriveAll(String orderBy)

Questo metodo permette di visualizzare l'elenco di tutte le categorie

- Precondizione: orderBy != null
- Postcondizione: il metodo ritorna una lista di categorie presenti nel database
- Invariante: il database deve essere funzionante

- Public synchronized void doSave(Category category)

Questo metodo permette di aggiungere una nuova categoria

- Precondizione: category != null
- Postcondizione: la categoria deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

- Public void doUpdate(Category category)

Questo metodo permette di modificare una categoria dal database

- Precondizione: category != null

- Postcondizione: la categoria viene modificata, se esiste nel database
- Invariante: il database deve essere funzionante

- Public void doDelete(Category category)

Questo metodo permette di eliminare una categoria dal database

- Precondizione: category!= null
- Postcondizione: la categoria viene eliminata, se esiste nel database
- Invariante: il database deve essere funzionante

- Public ArrayList<Category> doRetriveAllByProduct(Product product)

Questo metodo permette di visualizzare una lista di categorie legate ad un prodotto

- Precondizione: product!= null
- Postcondizione: il metodo, visualizza le categorie di un prodotto
- Invariante: il database deve essere funzionante

3.1.8 Manager Registrazione

Classe che funge da interfaccia del sottosistema per gestire le registrazioni.

- Public synchronized void doSave(String email, String password)

Questo metodo permette di aggiungere un nuovo utente

- Precondizione: user != null
- Postcondizione: l'utente deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

3.1.9 Manager Carrello

Classe che funge da interfaccia del sottosistema per gestire il carrello.

- Public Cart doRetriveByKey(String key)

Questo metodo permette di cercare carrello tramite id(key)

- Precondizione: key != null
- Postcondizione: il metodo ritorna un Cart, se presente nel database

- Public synchronized void doSave(Cart cart)

Questo metodo permette di aggiungere una nuovo carrello

- Precondizione: cart != null
- Postcondizione: il carrello deve essere salvato nel database correttamente
- Invariante: il database deve essere funzionante

- Public void doDelete(Cart cart)

Questo metodo permette di eliminare un carrello dal database

- Precondizione: cart != null
- Postcondizione: il carrello viene eliminato, se esiste nel database
- Invariante: il database deve essere funzionante

4. Design Pattern

4.1 Façade Pattern

Il Façade Pattern è stato utilizzato per la realizzazione in sottosistemi. Si basa, attraverso un'interfaccia più semplice, di accedere a sottosistemi che utilizzano interfacce più complesse.

Rappresentazione modello: