Corso di Ingegneria del Software

PharmaÉlite

Odd - Object Design Document

Versione 2.7

Partecipanti:

Nome	Matricola
Bozzoli Luigi	0512105477
Martucci Antonio	0512105612
Squitieri Lucio	0512105180

Indice

Object Design Document

1. Introduzione

1.1 Object design trade-offs

Durante la fase di Object Design sono sorti diversi compromessi di progettazione. Di seguito sono riportati i trade-off:

- **Leggibilità vs tempo**: L'obiettivo sarà scrivere codice che rispetti lo standard proposto da google per la programmazione con il linguaggio java, con l'aaggiunta di commenti per eventuali chiarimenti. Questo favorirà la leggibilità ed agevolare il processo di mantenimento e modifica del codice per gli sviluppatori che non avranno lavorato al progetto fin dall'inizio. Tuttavia, questo vantaggio aumenterà il tempo necessario per lo sviluppo e la realizzazione dell'intero sistema.
- **Affidabilità vs Tempo di risposta**: Il sistema sarà implementato in modo tale da preferire l'affidabilità al tempo di risposta in quanto si garantirà un controllo accurato dei dati in input per minimizzare errori.
- **Usabilità vs Funzionalità:** Il sistema sarà facilmente usufruibile dall'utente a discapito delle funzionalità offerte fa quest' ultimo.

1.2 Componenti off-the-shelf

Per l'implementazione del sistema utilizzeremo componenti self ossia componenti software già disponibili e utilizzati per facilitare la creazione del software. Il framework che utilizzeremo per la realizzazione delle interfacce grafiche delle pagine web è Bootstrap.

Per permettere all'interfaccia di rispondere alle azioni dell'utente e quindi velocizzare il sistema aggiungendo logica applicativa lato client si utilizzeranno JQuery.

Le pagine web visualizzate dall'utente si baseranno sui mock-UPS realizzati e consultabili nel punto 3.4.5 del Requirements analysis Document.

1.3 Documentazione dell'interfacce

Nell'implementazione del sistema i programmatori dovranno attenersi alle linee guida definite di seguito.

1.3.1 Classi e interfacce Java

Nella scrittura di codice per le classi Java ci si atterrà allo standard di Google relativo alla programmazione in Java consultabile al seguente link:

http://google.github.io/styleguide/javaguide.html#s4.6-whitespace

In questo standard ogni metodo ed ogni file possono non essere preceduti da un commento, inoltre, potranno esserci commenti in merito a particolari decisioni o calcoli.

La convenzione utilizzata per i nomi delle variabili è la seguente:

camelCase.

Essa consiste nello scrivere parole composte in modo che ogni parola al centro della frase inizia con una lettera maiuscola.

Quando si codificano classi e interfacce Java, si dovrebbero rispettare le seguenti regole di formattazione:

1. Non si devono inserire spazi tra il nome del metodo e la parentesi tonda "(" che apre la lista dei parametri.

2. La parentesi graffa aperta "{" si deve trovare sulla stessa linea dell'istruzione di dichiarazione.

3. La parentesi graffa chiusa "}" si troverà su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia.

I nomi delle classi devono essere sostantivi con l'iniziale maiuscola. I nomi delle classi dovrebbero essere semplici, descrittivi e inerenti al dominio applicativo. Non devono essere usati underscore per legare nomi.

I nomi dei metodi iniziano con una lettera minuscola, non sono consentiti caratteri speciali e dovranno essere semplici, descrittivi e inerenti al dominio applicativo.

Sia i nomi delle classi che i nomi dei metodi dovranno seguire la notazione camelCase descritta sopra.

```
public class ClienteBeanDAOImpl implements ClienteBeanDAO{

  public void doSave(ClienteBean c){
    Connection con = null;
    PreparedStatement ps = null;

    try {
      con = DriverManagerConnectionPool.getConnection();
      ps = con.prepareStatement("INSERT INTO PROGETTOTSW.Cliente value(?,?,?)");

      ps.setString(1, c.getEmail());
      ps.setString(2, c.getPassword());
      ps.setBoolean(3, c.getTipo());

      ps.execute();
    } catch (Exception e) {
      e.printStackTrace();
    }finally{

      try {

        ps.close();
        DriverManagerConnectionPool.releaseConnection(con);

      } catch (SQLException e) {

        e.printStackTrace();
      }

    }
  }
```

1.3.2 Pagine lato Server (JSP)

Le pagine JSP quando eseguite dovranno produrre un documento conforme allo standard HTML 5.

Il codice Java presente nelle JSP deve aderire alle convenzioni descritte nel punti 1.3.1, con le seguenti specifiche:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;

2. Il tag di chiusura (%>) si trova all'inizio di una riga;

3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione (<%= %>).

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="model.beans.DatiAnagraficiBean"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
  href="https://use.fontawesome.com/releases/v5.8.2/css/all.css">
<link rel="stylesheet"
  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
<link rel="stylesheet"
  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
<script
  src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
<script
  src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>


<link href="stylesheets/registerSuccess.css" rel="stylesheet"/>

<title>Pharmaélite</title>
</head>
<body>
<%
DatiAnagraficiBean dati = (DatiAnagraficiBean) request.getAttribute("datiAnagrafici");
%>

  <div class="container">
    <div id="mainDiv" class="row text-center">
          <div class="col-sm-6 col-sm-offset-3">
          <br><br> <h2 class="text-success">Success</h2>
          <h3><%= dati.getNome()%> <%= dati.getCognome() %></h3>
          <p class="text-muted" style="font-size:20px;">
            Grazie per la registrazione. Effettua l'accesso per usufruire dei servizi. </p>
          <a href="Login.html" class="btn btn-success"> Login</a>
          </div>

    </div>
  </div>
</body>
</html>
```

1.3.3 Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML 5 e il codice deve essere identanto, per facilitare la lettura, secondo le seguenti regole:

1. Un'indentazione consiste in una tabulazione;

2. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;

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

4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
    href="https://use.fontawesome.com/releases/v5.8.2/css/all.css">
<link rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
<link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
<script
    src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
<script
    src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>


<link href="stylesheets/registerSuccess.css" rel="stylesheet"/>

<title>Pharmaélite</title>
</head>
<body>
  <div class="container" style="align:center">
    <div id="mainDiv" class="row text-center">
          <div class="col-sm-6 col-sm-offset-3">
          <br><br> <h1 class="text-success">Success</h1>
          <p class="text-muted" style="font-size:25px;">
            Ordine eseguito con successo. </p>
          <a href="home.html" class="btn btn-success">Torna alla Home</a>
          </div>

    </div>
  </div>
</body>
</html>

```

1.3.4 Script Javascript

```javascript
function mostraDiv(x){
    $(x).fadeIn();
}

function modificaPrezzosped(id, prezzo){
    $("#"+id).text("EUR "+prezzo);
}

function prezzoSped(metodoDiSpedizione, totOrdine){
    if(totOrdine >= 49.99){
        modificaPrezzosped("prezzoSped", 0);
        return 0;
    }


    if(metodoDiSpedizione == "gratuita"){
        modificaPrezzosped("prezzoSped", 0);
        return 0;
    }
    if(metodoDiSpedizione == "standard"){
        modificaPrezzosped("prezzoSped", 4);
        return 4;
    }
    if(metodoDiSpedizione == "rapida"){
        modificaPrezzosped("prezzoSped", 5);

        return 5;
    }
}

function aggiornaPrezzo(){
    var metodoDiSpedizione=$("select[name=spedizione]").val();


    var totOrdine = $("#prezzoProdotti").text();
    totOrdine = totOrdine.replace("EUR", "");
    totOrdine = parseFloat(totOrdine);

    var prezzo = prezzoSped(metodoDiSpedizione, totOrdine);

    totOrdine += prezzo;

    $("#totOrdine").text("EUR "+totOrdine);
```

Gli Script in Javascript devono rispettare le seguenti convenzioni:

1. Gli script che svolgono funzioni distinte dal rendering della pagina dovrebbero essere collocati in file dedicati.

2. Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.

3. Le funzioni Javascript devono essere documentate in modo analogo ai metodi Java.

1.3.5 Fogli di stile CSS

I fogli di stile (CSS) devono seguire la seguente convenzione:

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

Inoltre ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si devono trovare nella stessa riga;

2. L'ultimo selettore della regola è seguito da 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 terminata da una parentesi graffa chiusa "}" collocata da sola su una riga;

```css
body {
    padding-top: 80px;
    padding-bottom: 80px;
}

.col-sm-3 {
    text-align: center;
}

.col-sm-6 {
    text-align: center;
}

.indirizzoSpedizione, .metodoPagamento {
    padding-top: 20px;
    display: none;
}

.indirizzoSpedizione {
    paddind-bottom: 20px;
}

#prezzoProdotti {
    font-weight: bold;
}

#numeroCarta, #ind {
    display: none;
    color: red;
}

#totOrdine {
    color: green;
}

@media screen and (max-width: 768px) {
    .col-sm-6 {
        padding-bottom: 15px;
    }
}
```

1.3.6 Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere maiuscole;

2. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

I nomi dei campi delle tabelle devono seguire le seguenti regole:

1.1 Il nome deve rispettare la convenzione camelCase.

1.2 Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

1.4 Design Pattern

1.4.1 MVC

Il design pattern MVC consente la suddivisione del sistema in tre blocchi principali: Model, View e Controller. Il Model modella i dati del dominio applicativo e fornisce i metodi di accesso ai dati persistenti, il View si occupa della presentazione dei dati all'utente e di ricevere da quest'ultimo gli input, infine il Controller riceve i comandi dell'utente attraverso il View e modifica lo stato di quest'ultimo e del Model.

1.4.2 DAO (Data Access Object) Pattern

Il DAO pattern è utilizzato per il mantenimento di una rigida separazione tra le componenti del Model e il Controller in un'applicazione basata sul paradigma MVC.

Il pattern sarà composta da:

•**Data Access Object Interfaccia:** interfaccia che definisce le operazioni che saranno performante sugli oggetti del model.

•**Data Access Object classe concreta:** classe che implementa l'interfaccia descritta sopra.

•**Model Object:** POJO contenenti i metodi get/set per salvare i dati ritirati attraverso le classi DAO.

1.5 Definizioni, Acronimi e abbreviazioni

JSP: acronimo di JavaServer Pages (talvolta detto Java Scripting Preprocessor), è una tecnologia di programmazione web in java per lo sviluppo della logica di presentazione (tipicamente secondo il pattern MVC) di applicazioni web.

MVC: acronimo di Model-view-controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software.

Off-The-Shelf: Servizi esterni al sistema di cui viene fatto utilizzo.

Bootstrap: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web.

HTML: Linguaggio di markup utilizzato per lo sviluppo di pagine Web.

CSS: acronimo di Cascading Style Sheets è un linguaggio usato per definire la formattazione delle pagine

Web.

JavaScript: JavaScript è un linguaggio di scripting utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

JQuery: JQuery è una libreria JavaScript per applicazioni web.

AJAX: AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la

realizzazione di applicazioni web interattive.

camelCase: È una tecnica di naming delle variabili adottata dallo standard

Google Java. Essa consiste nello scrivere più parole insieme delimitando l’inizio di una nuova parola con una lettera maiuscola.

Servlet: i servlet sono oggetti scritti in linguaggio Java che operano all'interno di un server web.

Tomcat: Apache Tomcat è un web server open source. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

2. Packages

2.1 Presentation

Il package presentation è formato dal package Gestore Catalogo e dalle classi

- VisualizzaCarrello JSP
- Checkout JSP
- Userpage JSP
- ListaOrdini JSP
- ListaProdotti JSP
- RegisterSuccess JSP.

Il package Gestore Catalogo è formato dalle seguenti classi:

- FormModificaProdotto
- ListaModificaProdotto
- ListaEliminaProdotto.

2.2 Controller

Il package controller riceve, tramite il pacchetto View, i comandi dell'utente. Esso, a sua volta, è diviso in 4 pacchetti:

1. Manager utente
2. Manager catalogo
3. Manager ordine
4. Manager carrello.

Il package Manager utente è formato dalle seguenti classi:

- GestoreUtente
- Login
- Logout
- Registrazione
- Userpage.

Il package Manager catalogo è formato dalle seguenti classi:

- Gestore catalogo

- AggiungiProdottoCatalogoForm
- InserisciProdottoCatalogo
- ModificaProdottoCatalogoForm
- UpdateProdotto
- ListaModificaProdottiCatalogo
- EliminaProdottoCatalogo
- CercaProdotto
- ListaProdottiCategorie.

Il package Manager ordine è formato dalle seguenti classi:

- AggiungiIndirizzoSpedizione
- AggiungiMetodoPagamento
- Checkout
- OrdinaOra
- CronologiaOrdini.
- GestoreOrdine

Il package Manager carrello è formato dalle seguenti classi:

- AggiungiProdottoCarrello
- UpdateCarrello
- RimuoviProdottoCarrello
- Carrello.
- GestoreCarrello

2.3 Model

Il package Model contiene tutte le classi dedite alla gestione dei dati persistenti. Esso si occupa di fare da tramite tra l'applicazione e il database sottostante. Ogni classe contenuta all'interno di questo pacchetto

fornisce i metodi per accedere ai dati persistenti dell' applicazione. Le classi contenute all'interno di questo package sono:

- ClienteBean
- ComposizioneBean
- DatiAnagraficiBean
- IndirizzoDiSpedizioneBean

- MetodoDiPagamentoBean
- OrdineBean
- ProdottoBean
- ProdottoNellordineBean
- CarrelloBean
- CarrelloBeanDAOImpl
- ClienteBeanDAOImpl
- ComposizioneBeanDAOImpl
- DatiAnagraficiBeanDAOImpl
- IndirizzoDiSpedizioneBeanDAOImpl
- MetodoDiPagamentoBeanDAOImpl
- OrdineBeanDAOImpl
- ProdottoBeanDAOImpl
- ProdottoNellordineBeanDAOImpl

3. Interfaccia delle classi

VINCOLI GESTORE UTENTE
Questa classe permette il login, la registrazione e la visualizzazione della user page.
Context :: GestoreUtente: registrazione(Nome, Cognome, sesso, città, numeroCarta, tipoCarta, IndirizzoDiSpedizione); **pre:** Nome !=null && Cognome !=null && sesso =sesso selezionato && città !=null && numeroCarta !=null && tipoCarta !=null && IndirizzoDiSpedizione !=null **post:** Utente registrato e aggiunto al DB
Context:: GestoreUtente: login(Email, Password); **pre:** Email !=null && password !=null
Context:: GestoreUtente: userPage();

pre: L’utente deve essere loggato
Context:: GestoreUtente: controlloEsistenzaEmail(email); **post:** restituisce true se rispetta il formato, false altrimenti
Context:: GestoreUtente: getMetodoDiPagamento() **post:** restituisce il metodo di pagamento usato dall’utente
Context:: GestoreUtente: setMetodoDiPagamento(metodoPagamento) **post:** imposta il metodo di pagamento usato dall’utente
Context:: GestoreUtente: getIndirizzoSpedizione() **post:** restituisce l’indirizzo di spedizione dell’utente
Context:: GestoreUtente: setIndirizzoSpedizione(indirizzoSpedizione) **post:** imposta l’indirizzo di spedizione
Context:: GestoreUtente: getDatiAnagrafici() **post:** restituisce i dati anagrafici dell’utente
Context:: GestoreUtente: setDatiAnagrafici(datiAnagrafici) **post:** imposta i dati anagrafici dell’utente
Context:: GestoreUtente: checkNumeroCarta(numeroCarta); **pre:** numeroCaarta!=null

post: restituisce true se rispetta il formato, false altrimenti
Context:: GestoreUtente: InserisciMetodoPagamento() **Post:** il metodo di pagamento è stato inserito nel DB
Context:: GestoreUtente: InserisciIndirizzoSpedizione() **Post:** l'indirizzo è stato inserito nel DB

Nome Classe	GestoreCatalogo
Descrizione	Questa classe permette al gestore del catalogo di aggiungere prodotti al catalogo modificare ed eliminare quelli gia esistenti e permette atutti gli utenti dcercare un prodotto.
Pre-Condizione	**Context:** GestoreCatalogo: aggiungiProdottoCatalogo(Id, UrlImmagine, categoria, Nome, prezzo, quantita,descrizione);
	pre: Id !=null && UrlImmagine !=null && categoria !=null && Nome !=null && prezzo !=null && quantita >=0 && descrizione !=null **post:** Prodotto inserito e salvato nel DB
	Context: GestoreCatalogo: modificaProdottoCatalogo(Id, UrlImmagine, categoria, Nome, prezzo, quantita,descrizione)
	pre: : Id !=null && UrlImmagine !=null && categoria !=null && Nome !=null && prezzo !=null && quantita >=0 && descrizione !=null **post:** UrlImmagine=newUrlImmmagine && categoria=newCategoria && Nome= newNome && prezzo= newPrezzo&& quantita= newQuantita && descrizione= newDescrizione
	Context: GestoreCatalogo: cercaProdotto(Nome) **pre:** Nome !=null **post:** Elenco di tutti i prodotti con quel nome

	pre: Nome !=null **post:** Elenco di tutti i prodotti con quel nome
	Context: GestoreCatalogo EliminaProdotto(Id)
	Pre: id!=null **Post:** Il prodotto con l'Id inserito è stato eliminato dal DB
	Context: GestoreCatalogo ritiraProdotti()
	Post: Elenco di tutti i prodotti presenti nel database

Nome Classe	GestoreCarrello
Descrizione	Questa classe permette di aggiungere ed eliminare prodotti dal proprio carrello e di procedere all'ordine per l'acquisto
Pre-Condizione	**Context** GestoreCarrello: aggiungiProdottoCarrello(id, quantita)
	pre: id!=null && quantita >=1 **post:** Il prodotto selezionato viene aggiunto al carrello dell'utente
	Context: GestoreCarrello: rimuoviProdottoCarrello(id)
	pre: id!=null **post:** il prodotto con l'id corrispondente a quello inserito viene eliminato dal carrello dell'utente
	Context: GestoreCarrello: procediAllOrdine(carrello)
	pre: l'utente deve essere loggato **Post:** l'utente visualizza la pagina con il riepilogo dell'ordine
	Context: GestoreCarrello: modificaQuantitaProdottoCarrello(dProdotto, quantita, cliente, carrello)
	Pre: idProdotto !=null && quantita!= null &&cliente!=null && carrello !=null

	Post: la quantità del prodotto nel carrello risulta modificate se la quantita rimanente del prodotto è sufficiente a soddisfare la richiesta
	Context: GestoreCarrello: ritiraCarrelloUtenteLoggato(email)
	Pre: email!=null **Post:** l'utente loggato visualizza il suo carrello
	Context: GestoreCarrello: ritiraCarrello(carrello)
	Post: l'utente visualizza il suo carrello
	Context: GestoreCarrello: getListaProdotti()
	Post: restituisce la lista dei prodotti
	Context: GestoreCarrello: setListaProdotti(listaProdotti)
	Post: listaProdotti= new listaProdotti
	Context: GestoreCarrello: getPrezzo()
	Post: restituisce il prezzo

	Context: GestoreCarrello: setPrezzo(prezzo)
	Post: prezzo=new prezzo
	Context: GestoreCarrello: getCarrello()
	Post: restituisce il carrello
	Context: GestoreCarrello: setCarrello(carrello)
	Post: carrello=new carrello

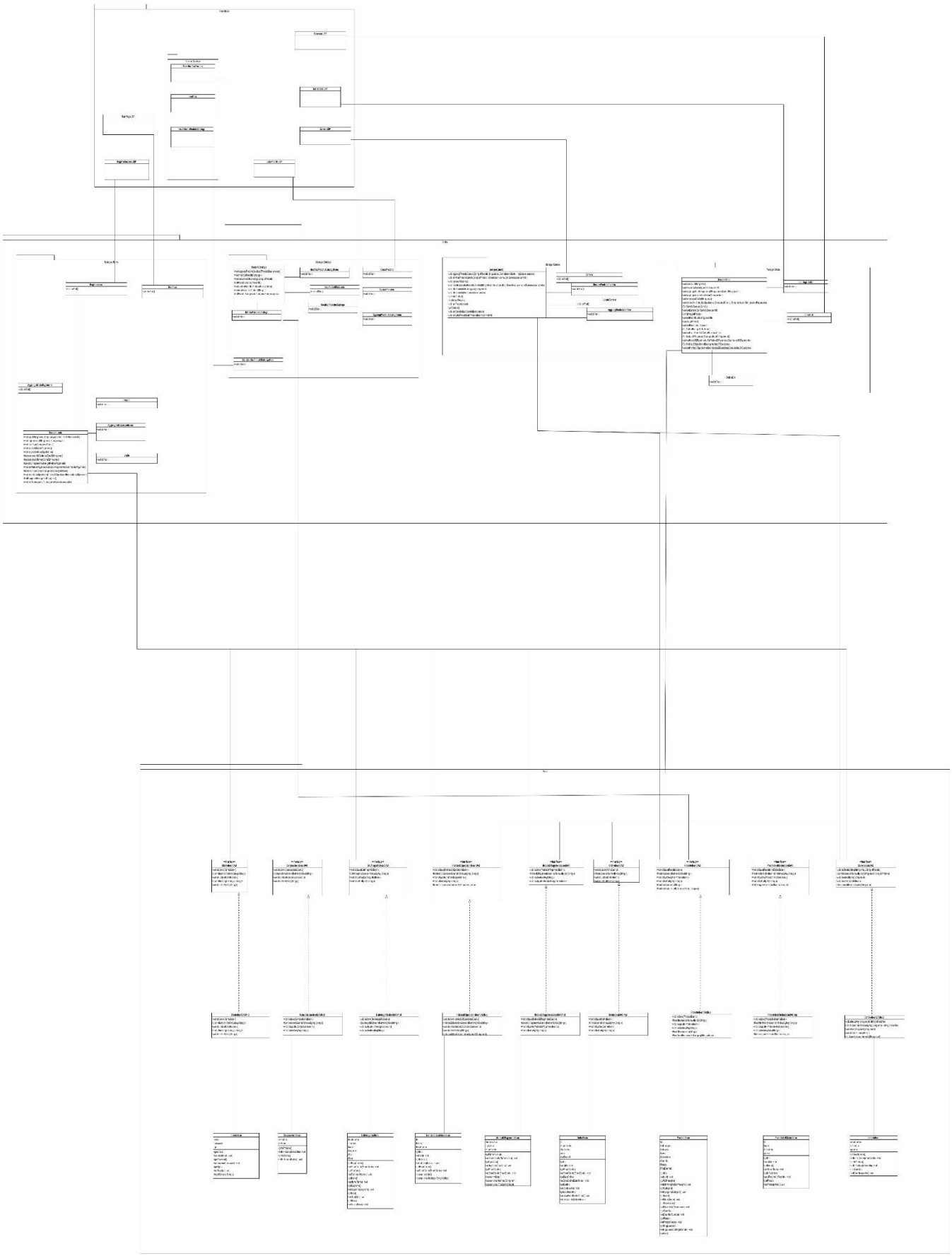
Nome Classe	GestoreOrdini
Descrizione	Questa classe permette il login, la registrazione e la visualizzazione della user page.
Pre-Condizione	**Context** GestoreOrdini: checkout(carrello)
	pre: l'utente deve essere loggato **post:** viene effettuato l'ordine e viene salvato nel DB
	Context: GestoreOrdini: visualizzaCarrello(carrello)
	post: vengono visualizzati tutti i prodotti nel carrello dell'utente
	Context: GestoreOrdini: aggiungiMetodoPagamento(numeroCarta,tipoCarta)
	pre: numeroCarta !=null && tipoCarta !=null **post:** il nuovo metodo di pagamento risulta salvato nel DB e legato all'utente
	Context: GestoreOrdini: aggiungiIndirizzoSpedizione(indirizzo);
	pre: indirizzo !=null **post:** il nuovo indirizzo di spedizione risulta salvato nel DB e legato all'utente
	Context: GestoreOrdini: cronologiaOrdini(email)

	pre: l'utente deve essere loggato **post:** vengono visualizzati i dati relativi a tutti gli ordini effettuati dall'utente
	Context: GestoreOrdini: ordinaOra(tipoSpedizione, emailCliente, indirizzo, metodoPagamento)
	Pre: tipoSpedizione !=null && emailCliente !=null && indirizzo !=null &&metodoPagamento !=null && **Post:** L'ordine effettuato dall'utente viene salvato nel DB
	Context: GestoreOrdini: getCarrello()
	Post: restituisce il carrello
	Context: GestoreOrdini: setCarrello(carrello)
	Post: carrello=new carrello
	Context: GestoreOrdini: getProdotti()
	Post: restituisce i prodotti dell'ordine

	Context: GestoreOrdini: setProdotti(prodotti)
	Post: prodotti=new prodotti
	Context: GestoreOrdini: getPrezzo()
	Post: restituisce il prezzo dell' ordine
	Context: GestoreOrdini: setPrezzo(prezzo)
	Post: prezzo=new prezzo
	Context: GestoreOrdini: getListaOrdini()
	Post: restituisce la lista dei prodotti di un utente
	Context: GestoreOrdini: setListaOrdini(listaOrdini)
	Post: listaOrdini=new listaOrdini
	Context: GestoreOrdini: getMetodiDiPagamento()
	Post: restituisce i metodi di pagamento dell'utente

	Context: GestoreOrdini: setMetodiDiPagamento(metodiDiPagamento)
	Post: aggiunge metodiDiPagamento all'insieme preesistente dei metodi di pagamento dell'utente
	Context: GestoreOrdini: getIndirizziDiSpedizione()
	Post: restituisce gli indirizzi di spedizione
	Context: GestoreOrdini: setIndirizziDiSpedizione(indirizziDiSpedizione)
	Post: aggiunge indirizzoDiSpedizione all'insieme preesistente degli indirizzi di spedizione dell'utente

4.Class Diagram



5. Glossario

Trade-off: Il Trade-off è una situazione che implica una scelta tra due possibilità, in cui la perdita di

valore di una costituisce un aumento di valore in un'altra.

POJO: acronimo di Plain Old Java Object utilizzati per indicare che un oggetto è un oggetto ordinario Java non legato ad alcuna restrizione.