



UNIVERSITÀ DEGLI STUDI DI SALERNO

**Corso di Ingegneria del Software
Object Design Document
(ODD)**

Quattro⁶³hi.it

Data: 10/01/2018

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Coordinatori del progetto:

Nome
De Lucia Andrea
Francese Rita

Partecipanti:

Nome	Matricola
Palmiero Alberto	0512103454
Piccolo Luigi	0512103964
Reccia Luca	0512103736
Volpe Nicola	0512103556

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Sommario

1. INTRODUZIONE	5
1.1. OBJECT DESIGN TRADE-OFFS	5
1.2. LINEE GUIDA DOCUMENTAZIONE INTERFACCIA	6
1.3. COMPONENTI OFF – THE – SHELF	7
1.4. DEFINIZIONI, ACRONIMI E ABBREVIAZIONI	7
1.5. RIFERIMENTI	7
2. PACKAGES	8
2.1. CONTROL	9
2.1.1. <i>Gestione articoli</i>	10
2.1.2. <i>Gestione ordini</i>	11
2.1.3. <i>Gestione Utenti</i>	12
2.2. MODEL	13
2.3. ENTITY	14
2.4. WEB PAGES	15
2.5. VIEW	16
2.6. JS	17
2.7. CSS	18
3. INTERFACCE DELLE CLASSI	19
3.1. GESTIONE ARTICOLI	19
3.1.1. <i>Visualizza prodotto</i>	19
3.1.2. <i>Ricerca prodotto</i>	20
3.1.3. <i>Ricerca prodotto avanzata</i>	21
3.1.4. <i>Visualizza catalogo</i>	22
3.1.5. <i>Ricerca prodotto da pagina esterna</i>	22
3.1.6. <i>Ricerca tutti</i>	23
3.2. GESTIONE ORDINI	24
3.2.1. <i>Visualizza carrello</i>	24
3.2.2. <i>Visualizza checkout</i>	24
3.2.3. <i>Visualizza ordini</i>	25
3.2.4. <i>Visualizza storico ordini</i>	25
3.2.5. <i>Aggiungi prodotto al carrello</i>	26
3.2.6. <i>Inserisci Dati Di Spedizione</i>	27
3.2.7. <i>Checkout</i>	28
3.2.8. <i>Gestione ordine</i>	29
3.2.9. <i>Modifica quantità in carrello</i>	30
3.2.10. <i>Rimuovi prodotto dal carrello</i>	31
3.3. GESTIONE UTENTI	32
3.3.1. <i>Login</i>	32
3.3.2. <i>Logout</i>	33
3.3.3. <i>Rimuovere Carta</i>	33
3.3.4. <i>Rimuovere indirizzo</i>	34
3.3.5. <i>Visualizza login</i>	34

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.3.6.	<i>Visualizza ordine</i>	35
3.3.7.	<i>Visualizza Profilo</i>	35
3.3.8.	<i>Welcome</i>	36
3.3.9.	<i>Inserire carta di credito</i>	36
3.3.10.	<i>Inserire indirizzo di spedizione</i>	37
3.4.	ARTICOLOINORDERMODEL	38
3.5.	ARTICOLOINSTOCKMODEL	40
3.6.	ACQUIRENTEMODEL	42
3.7.	GESTOREORDINIMODEL	44
3.8.	ORDERMODEL	45
3.9.	SHIPPINGADDRESSMODEL	47
3.10.	CREDITCARDMODEL	49
3.11.	ACQUIRENTE	51
3.12.	ARTICOLO IN STOCK	52
3.13.	ARTICOLO IN ORDER	53
3.14.	CART	54
3.15.	GESTORE ORDINI	55
3.16.	ORDER	56
3.17.	SHIPPING ADDRESS	57
3.18.	CREDIT CARD	58
4.	CLASS DIAGRAM FINALE	59
5.	SEQUENCE DIAGRAM FINALI	60

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

1. Introduzione

1.1. Object design trade-offs

[Costi vs Mantenimento]

È stato deciso di aggiungere documentazione al codice, aumentando di conseguenza i costi per la documentazione, per poterlo rendere comprensibile e poco ignoto anche alle persone che non partecipano al progetto o che non hanno partecipato a quella parte. Ciò viene fatto per aumentare la comprensibilità, il mantenimento e il processo di modifica.

[Memory space vs response time]

Per velocizzare il software, e quindi i tempi di risposta, è stato deciso di aggiungere una maggiore ridondanza e quindi un incremento della memoria utilizzata; anche grazie all'uso del database, che ha aumentato in maniera impercettibile la latenza poiché il sito non effettua molte elaborazioni / richieste con il database.

[Interfaccia vs easy – use]

L'interfaccia di Quattrocchi.it sarà minimale, intuitiva e non complessa al fine di garantire una facilità d'uso anche ad utenti meno esperti, così da non richiedere grandi skills per utilizzare il sistema. Il sistema userà font di una grandezza tale da facilitare la lettura ad un vasto insieme di utenti, la navigazione è agevole e concreta. Inoltre l'interfaccia si adatta a qualsiasi dispositivo l'utente usi per accedere al sito.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

1.2. Linee guida documentazione interfaccia

Prima dell'implementazione vera e propria, risulta opportuno delineare delle linee guida o "regole" che si devono rispettare per la scrittura del codice.

[Spostamento di linee]

Quando un'espressione supera la lunghezza della linea, occorre spezzarla secondo i seguenti principi generali:

- Interrompere la linea dopo una virgola;
- Interrompere la linea prima di un operatore;
- Allineare la nuova linea con l'inizio dell'espressione nella linea precedente.

[Indentazione]

Viene utilizzata l'indentazione del codice, soprattutto nei cicli FOR e WHILE, nel blocco di istruzioni delle condizioni IF, nel codice HTML se qualche tag è incluso in un altro tag.

Es1.

```
For (i=0; i<10; i++) {
    istruzione;
}
```

Es2.

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

[Parentesi]

Qualora dovesse esserci anche una sola istruzione, che viene eseguita in seguito al soddisfacimento di una condizione IF o all'interno di un ciclo, deve essere racchiusa tra parentesi graffe.

[Nomi delle classi]

Le classi saranno nominate con nomi al singolare. Ad esempio, la classe che rappresenta i gestori del catalogo avrà come nome *Gestore Catalogo*

[Nomi delle variabili]

La convenzione che sarà adottata per quanto riguarda i nomi delle variabili è la Camel Notation.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

1.3. Componenti off – the – shelf

Per il progetto software che si vuole realizzare facciamo uso di componenti **off-the-shelf**, ossia componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti.

Un componente che andremo ad utilizzare sarà Bootstrap, un Toolkit open source per lo sviluppo con HTML, CSS e JS. Risulta molto efficiente usare questo framework data la vasta gamma di librerie che mette a disposizione per la creazione dell'interfaccia utente.

1.4. Definizioni, acronimi e abbreviazioni

RAD: Requirements Analysis Document.

SDD: System Design Document.

ODD: Object Design Document.

DB: Database.

Off the shelf: componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti.

Gestore catalogo: utente del sito che gestisce gli articoli presenti nel catalogo.

Gestore ordini: utente del sito che gestisce la spedizione degli ordini effettuati.

Amministratore: amministratore del sito. Gestisce tutte le login registrate.

HTML: Linguaggio di Mark-up per pagine web.

Bootstrap: Framework che fornisce un insieme di elementi grafici, stilistici, di impaginazione e Javascript pronti all'uso.

Quattrocchi.it: il sito di e-commerce di occhiali da sole che si intende sviluppare.

1.5. Riferimenti

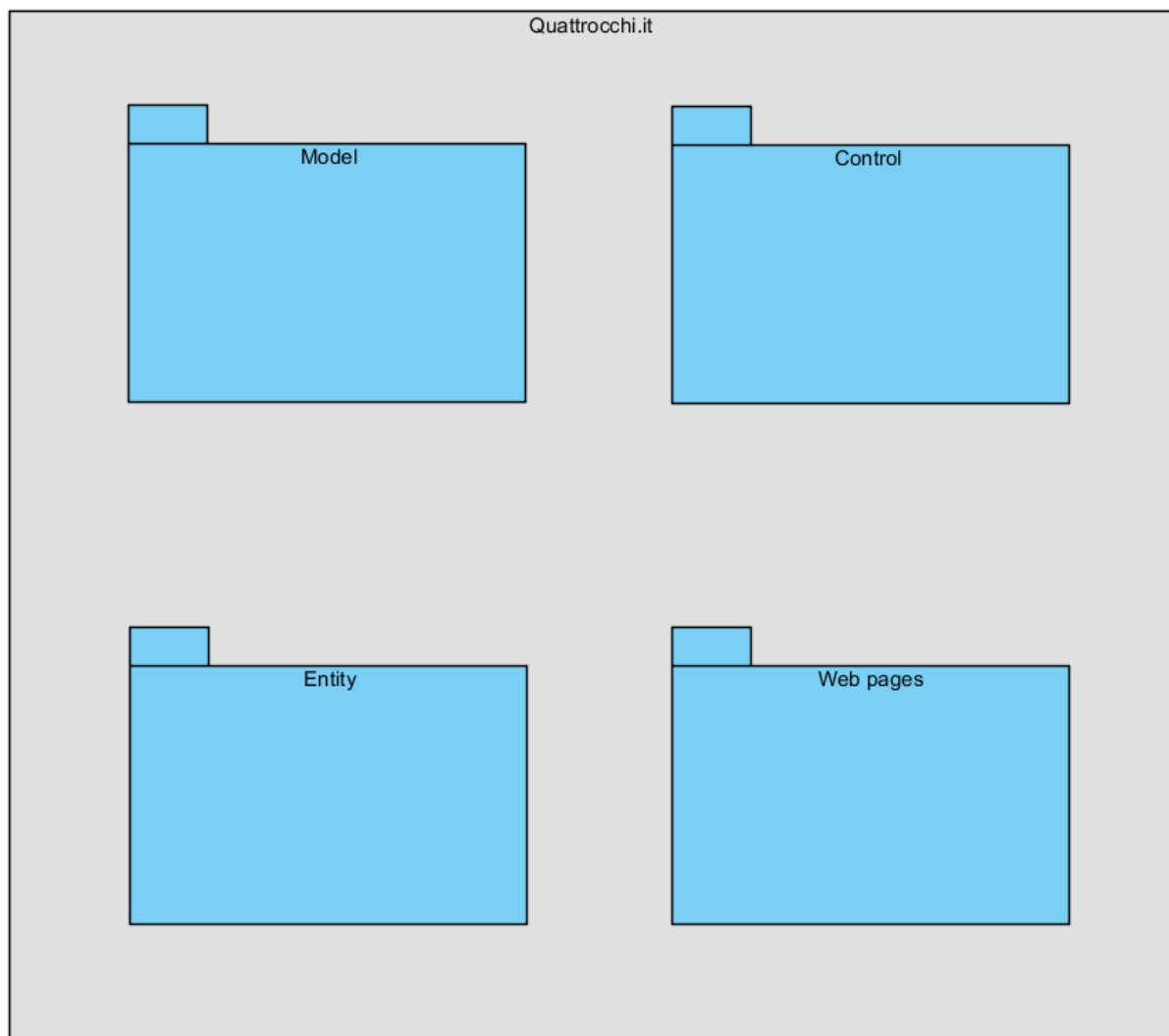
Bernd Bruegge & Allen H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns and Java*, (2nd edition), Prentice-Hall, 2003.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2. Packages

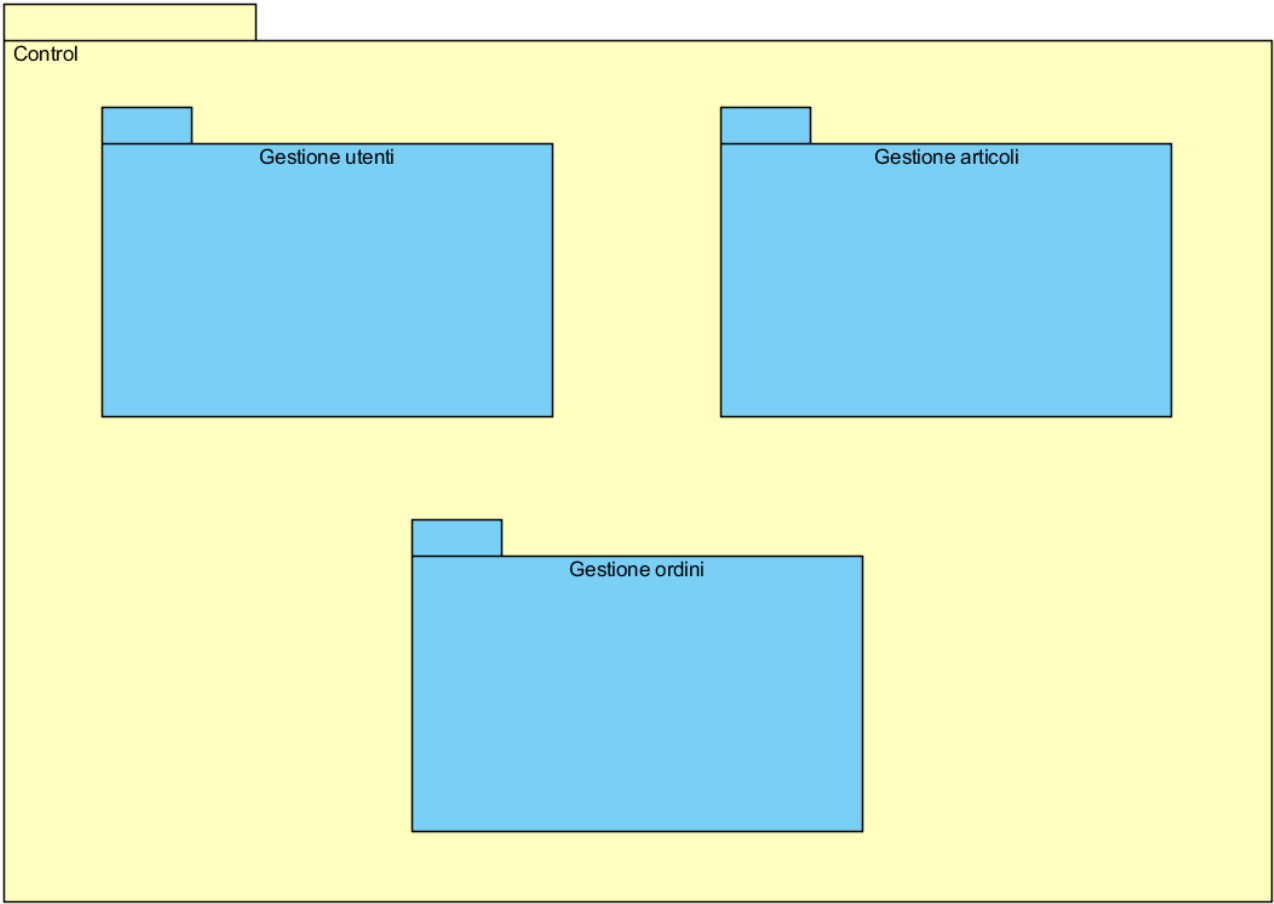
In questa sezione viene rappresentata la suddivisione in package del sistema in base all'architettura scelta. I packages principali sono:

- Control
- Model
- Entity
- Web pages



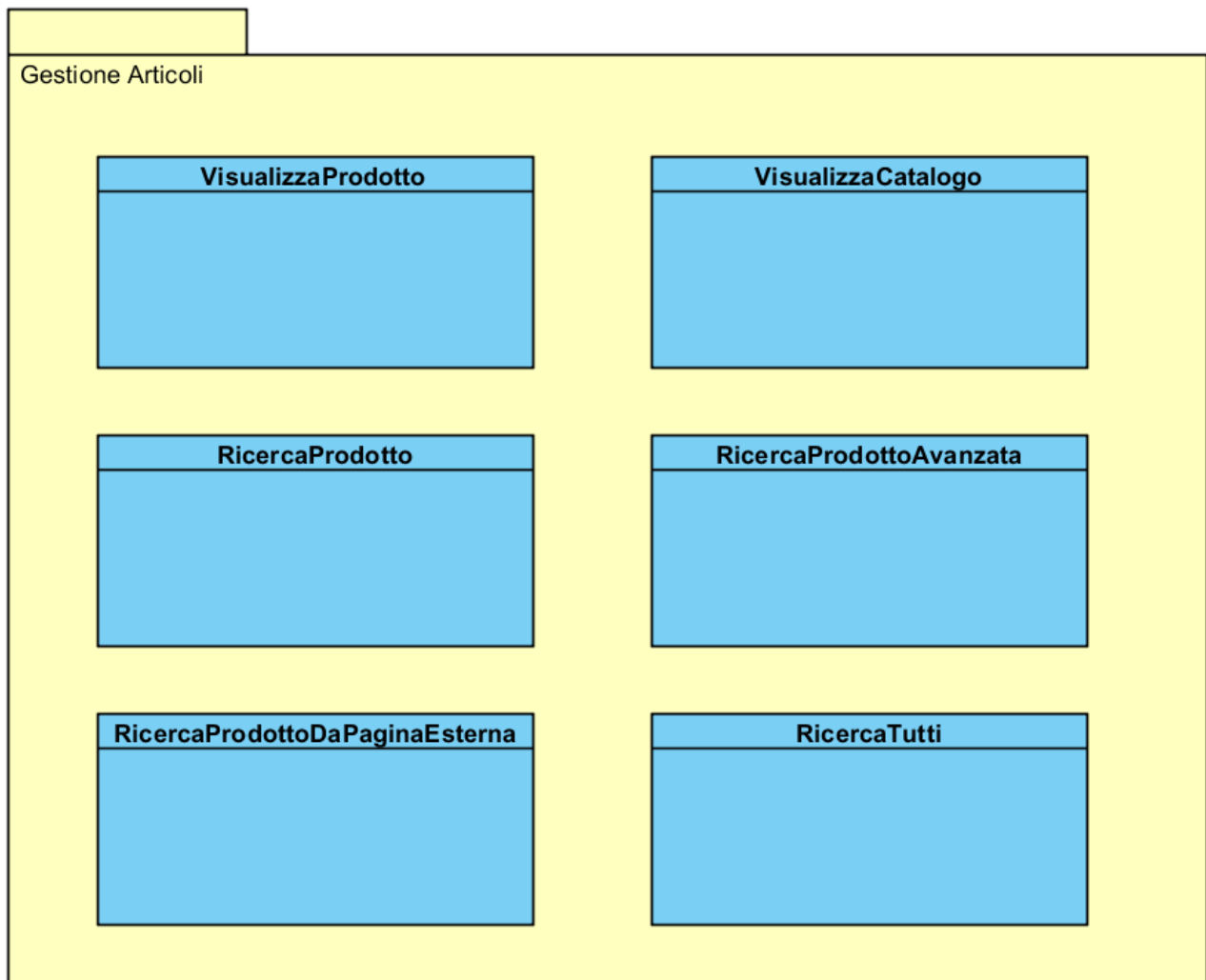
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.1. Control



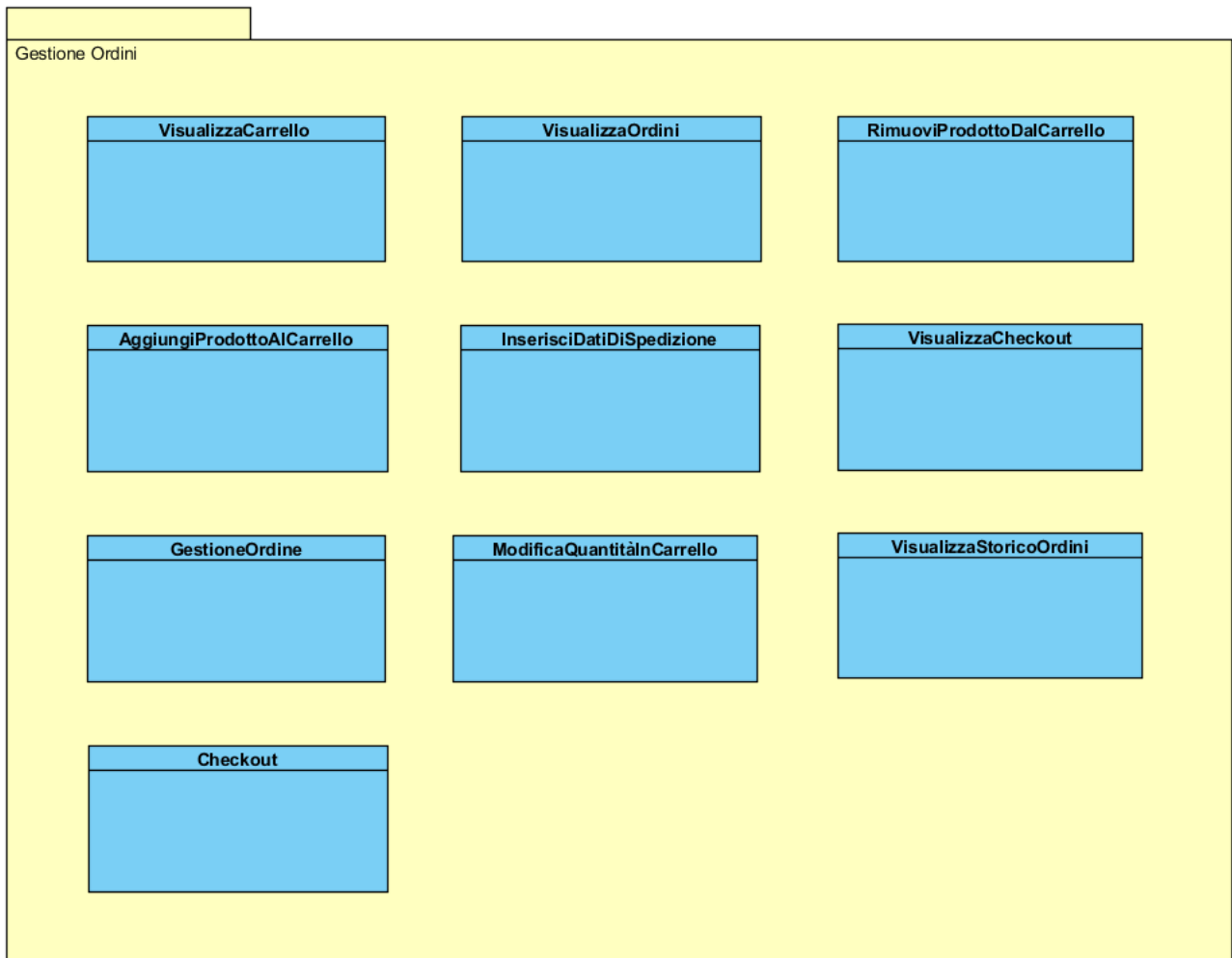
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.1.1. Gestione articoli



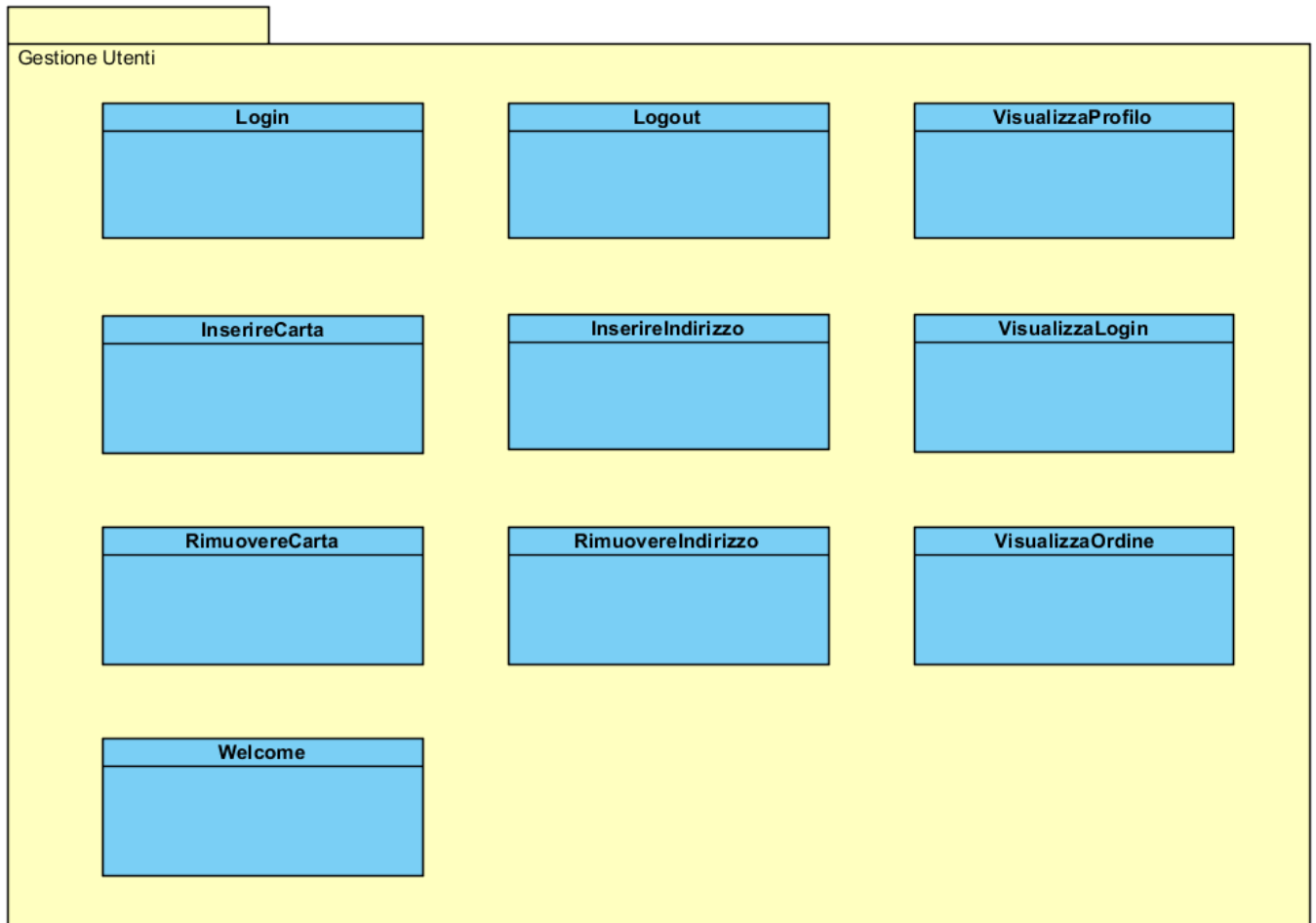
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.1.2. Gestione ordini



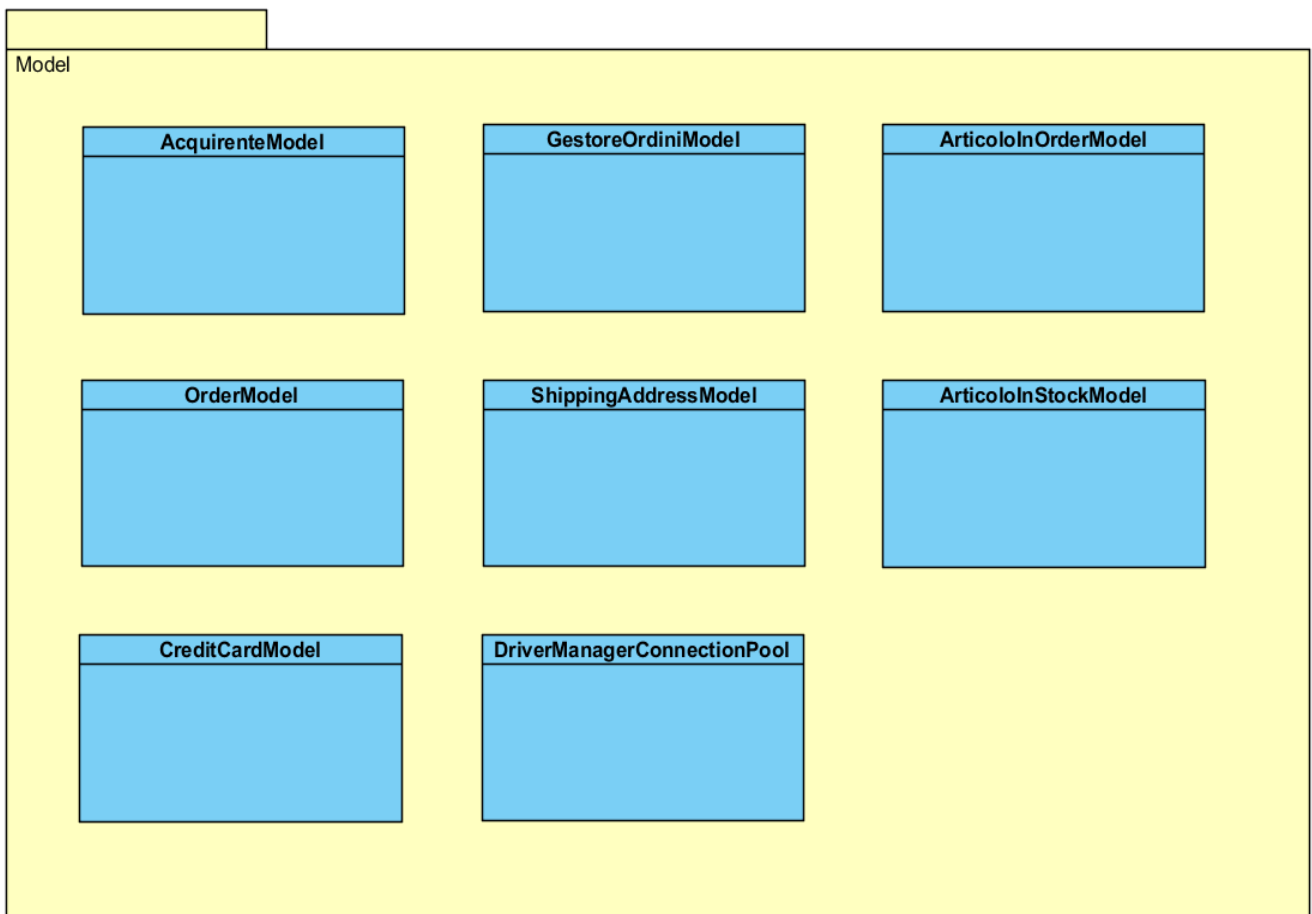
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.1.3. Gestione Utenti



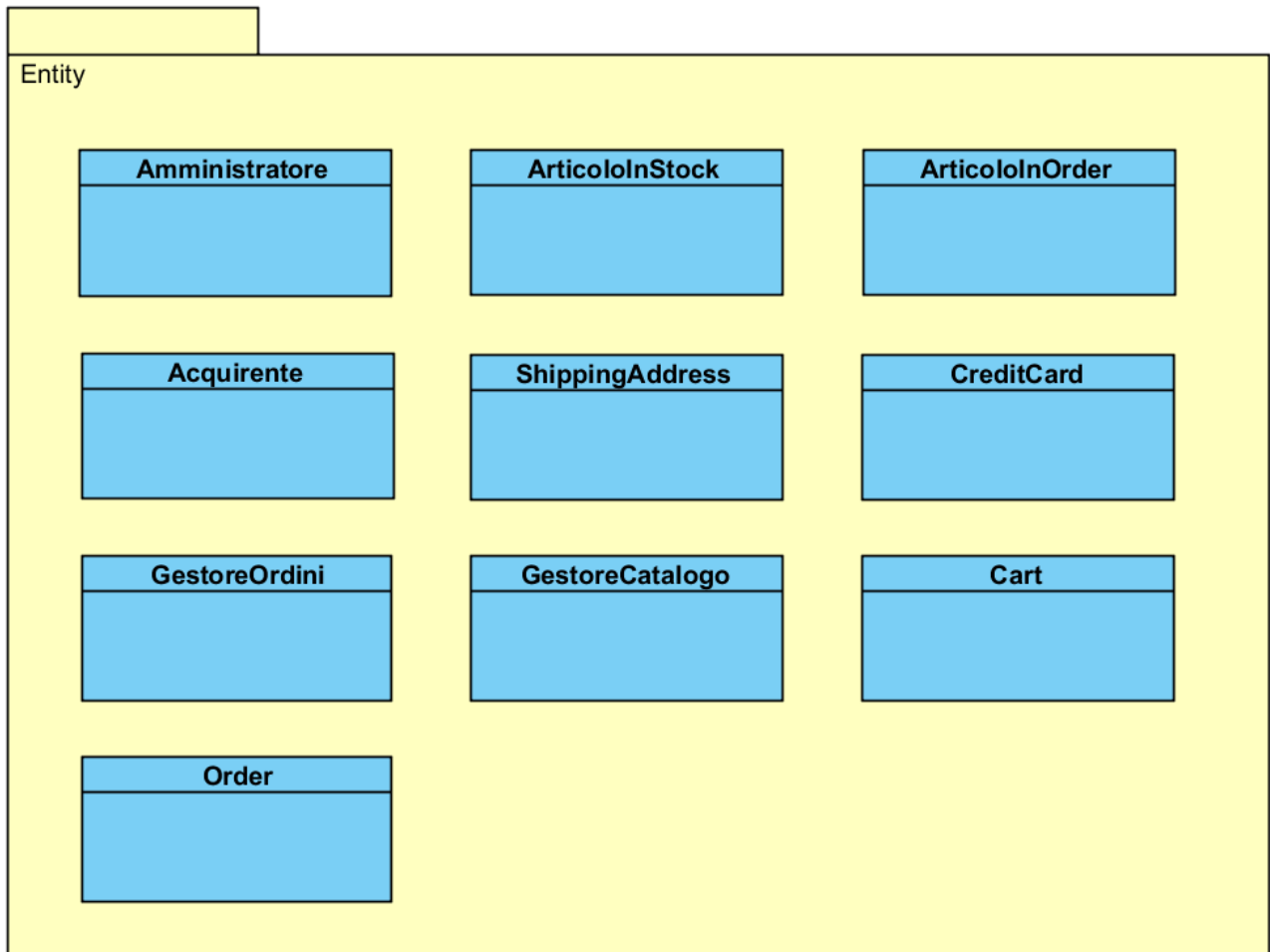
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.2. Model



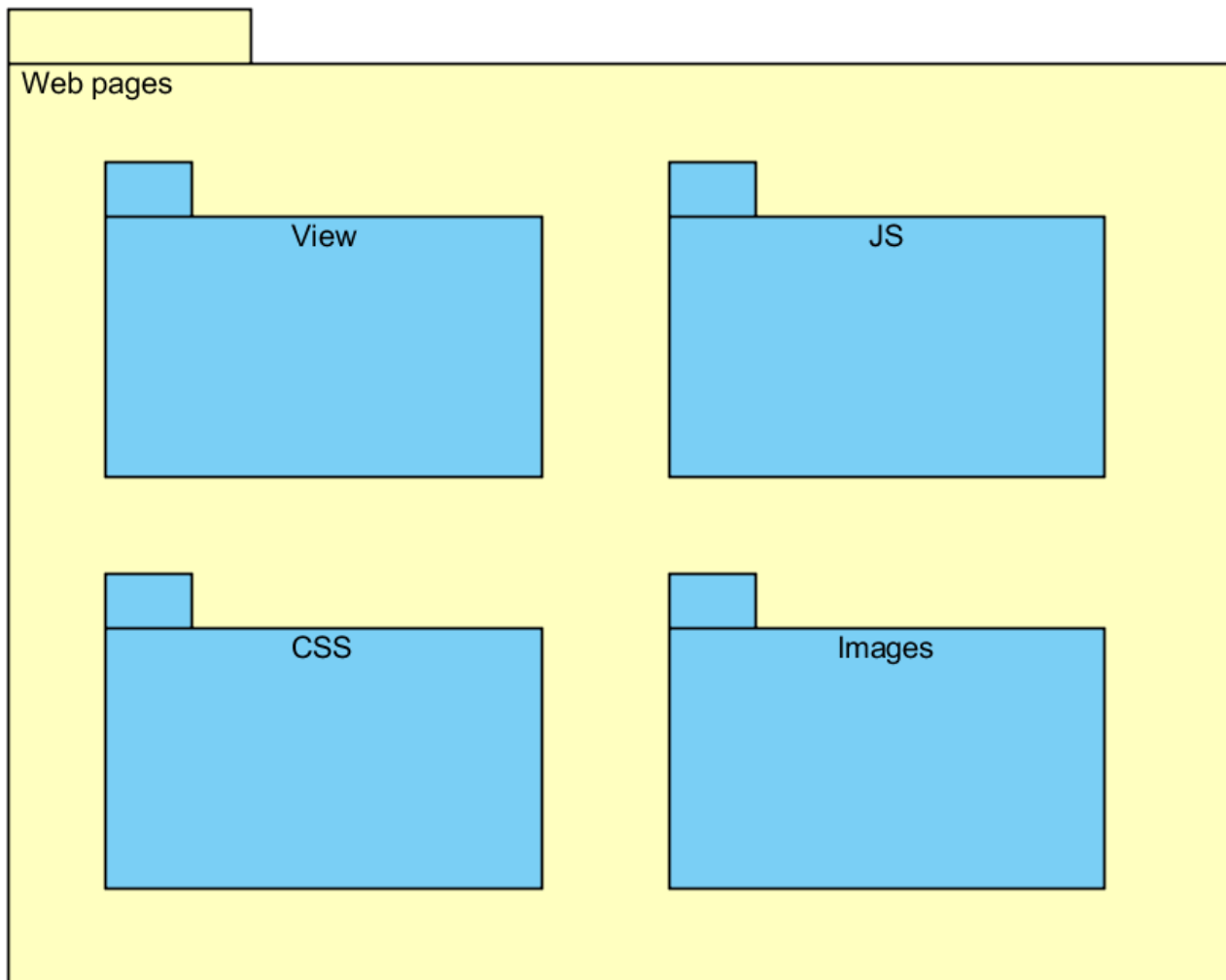
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.3. Entity



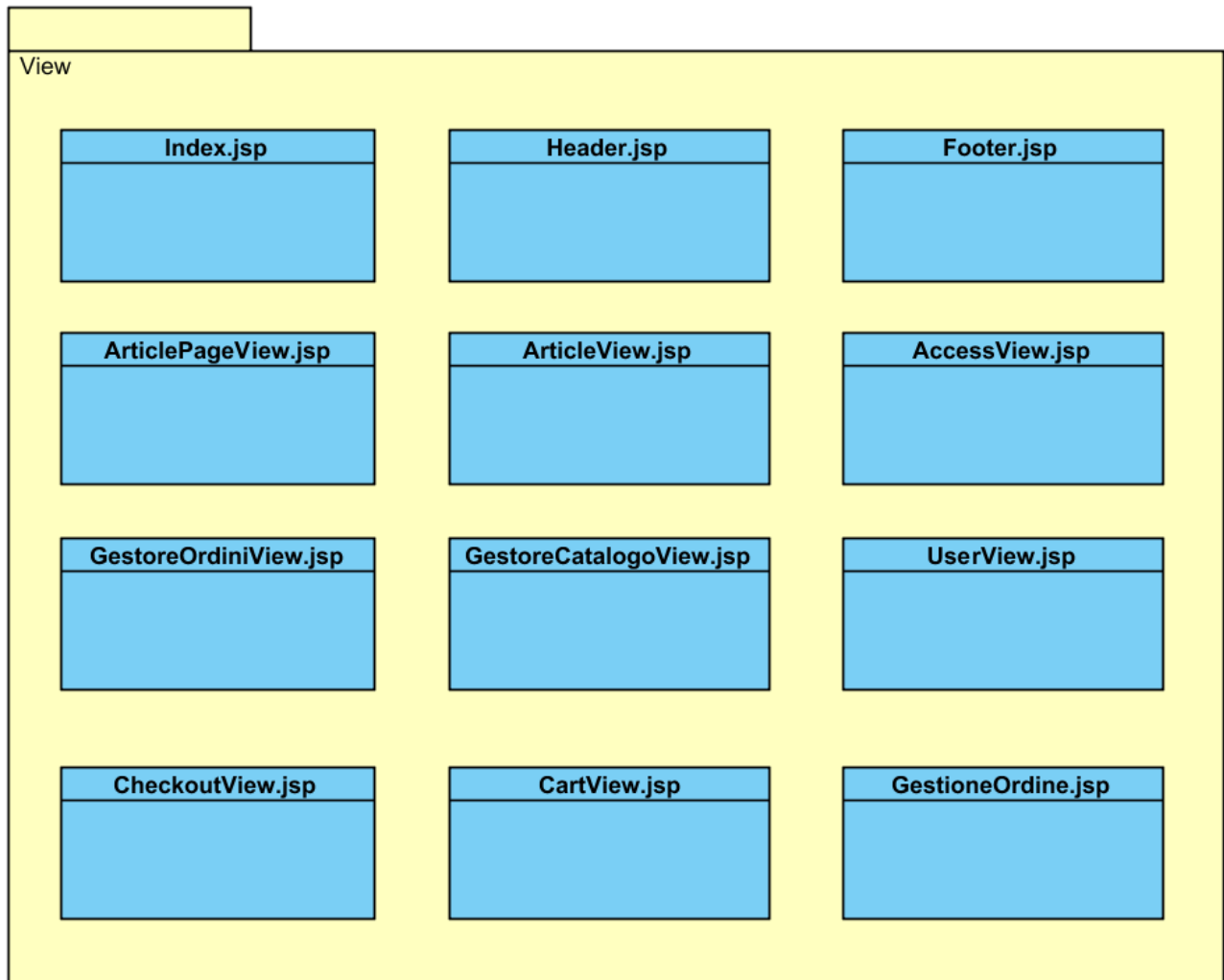
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.4. Web pages



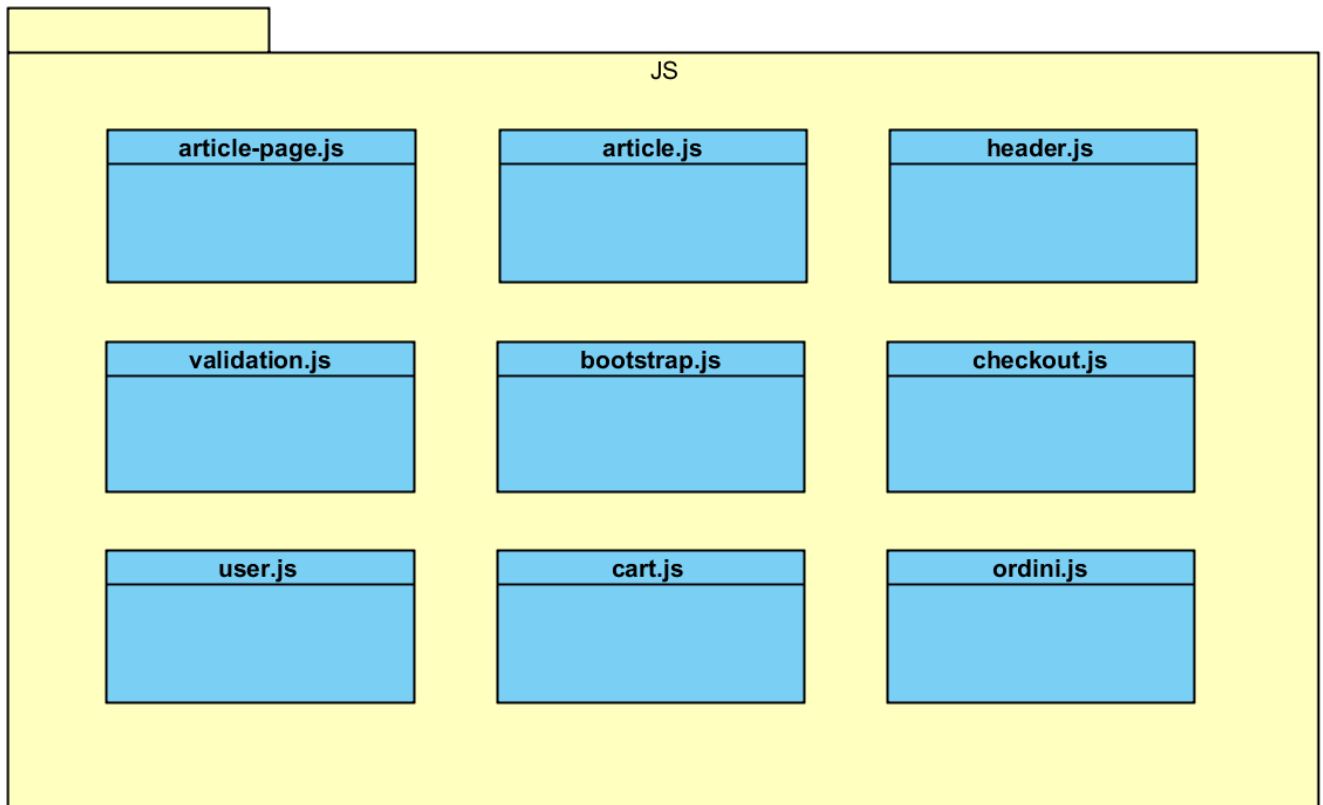
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.5. View



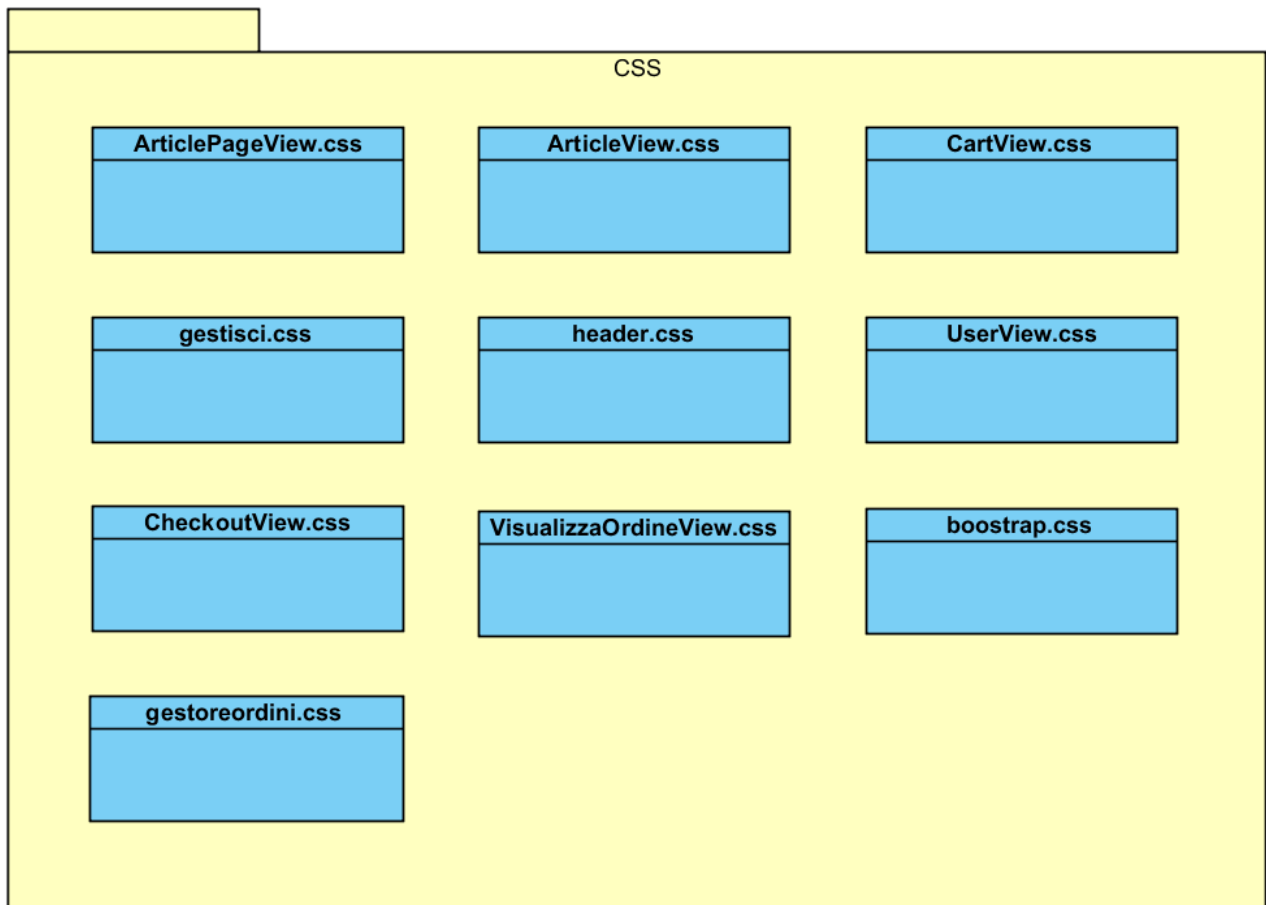
Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.6. JS



Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

2.7. CSS



Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3. Interfacce delle classi

3.1. Gestione articoli

3.1.1. Visualizza prodotto

Nome della classe	VisualizzaProdotto
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione della scheda di un prodotto.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di far visualizzare la scheda di un prodotto. Prende il parametro "idProdotto" di tipo String dalla request, chiama il metodo doRetrieveByIdInStock per controllare se il prodotto è presente nel database ed esegue il dispatch della pagina.
Pre – condizione	Context: VisualizzaProdotto :: doGet(request, response) Pre: request.getParameter("idProdotto") != null && Formato idProdotto [A-Za-z0-9]{10} && Esiste un prodotto con idProdotto nel Database
Post – condizione	Context: VisualizzaProdotto :: doGet(request,response) Post: request.getAttribute("articolo") != null && viene inserito un articolo nella response.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.1.2. Ricerca prodotto

Nome della classe	RicercaProdotto
Descrizione	Questa classe è una servlet che si occupa di gestire la ricerca di prodotti dal catalogo.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la ricerca di prodotti all'interno del catalogo utilizzando la stringa inserita dall'utente nell'apposita barra di ricerca.
Pre – condizione	Context: RicercaProdotto :: doGet(request, response) Pre: Formato stringaRicerca [A-Za-z0-9] {1,20} Esiste almeno un articolo nel database che corrisponde alla ricerca
Post – condizione	Context: RicercaProdotto :: doGet(request, response) Post: se il Model restituisce almeno 1 prodotto : request.getAttribute("prodotti") != null se il Model non restituisce nessun prodotto che matcha con la stringa: request.getAttribute("prodotti") == null e viene scritto nella response una lista di articoli non vuota.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.1.3. Ricerca prodotto avanzata

Nome della classe	RicercaProdottoAvanzata
Descrizione	Questa classe è una servlet che si occupa di gestire la ricerca avanzata di prodotti dal catalogo.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la ricerca avanzata di prodotti all'interno del catalogo utilizzando i vari parametri inseriti dall'utente.
Pre – condizione	Context: RicercaProdottoAvanzata :: doGet(request, response) Pre: request.getParameter("stringaRicerca") != null && Formato stringaRicerca [A-Za-z0-9]{1,20} Marca != null !marca.matches("[A-Za-z0-9]{1,20}") prezzoMinS != null !prezzoMinS.matches("[0-9]{1,5}") prezzoMaxS != null !prezzoMaxS.matches("[0-9]{1,5}") prezzoMinS != && prezzoMaxS != null) Integer.parseInt(prezzoMinS) <= Integer.parseInt(prezzoMaxS) Esiste almeno un articolo nel database che corrisponde ai parametri non vuoti.
Post – condizione	Context: RicercaProdottoAvanzata :: doGet(request, response) Post: se il Model restituisce almeno 1 prodotto : request.getAttribute("prodotti") != null se il Model non restituisce nessun prodotto che matcha con la stringa: request.getAttribute("prodotti") == null Viene scritto in response una lista di articoli non vuota.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.1.4. Visualizza catalogo

Nome della classe	VisualizzaCatalogo
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione del catalogo dei prodotti.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione del catalogo dei prodotti.
Pre – condizione	Context: VisualizzaCatalogo :: doGet(request, response)
Post – condizione	//

3.1.5. Ricerca prodotto da pagina esterna

Nome della classe	RicercaProdottoDaPaginaEsterna
Descrizione	Questa classe è una servlet che si occupa di gestire la ricerca da una qualsiasi pagina.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la ricerca da una qualsiasi pagina.
Pre – condizione	Context: RicercaProdottoDaPaginaEsterna :: doGet(request, response) Pre: toSearch != null && toSearch.matches("[A-Za-z0-9]{1,20}")
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.1.6. Ricerca tutti

Nome della classe	RicercaTutti
Descrizione	Questa classe è una servlet che si occupa di gestire la ricerca di tutti i prodotti.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione dell'intero catalogo dei prodotti
Pre – condizione	Context: RicercaTutti :: doGet(request, response)
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2. Gestione ordini

3.2.1. Visualizza carrello

Nome della classe	VisualizzaCarrello
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione del carrello.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione del carrello.
Pre – condizione	Context: VisualizzaCarrello :: doGet(request, response) Pre: request.getParameter("Cart") != null
Post – condizione	//

3.2.2. Visualizza checkout

Nome della classe	VisualizzaCheckout
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione della pagina di checkout.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione della pagina di checkout..
Pre – condizione	Context: VisualizzaCheckout :: doGet(request, response) Pre: L'acquirente è loggato && il carrello non è vuoto.
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2.3. Visualizza ordini

Nome della classe	VisualizzaOrdini
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione degli ordini.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione degli ordini.
Pre – condizione	Context: VisualizzaOrdini :: doGet(request, response) Pre: L'utente loggato è un gestore degli ordini.
Post – condizione	//

3.2.4. Visualizza storico ordini

Nome della classe	VisualizzaStoricoOrdini
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione dello storico degli ordini.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione dello storico degli ordini.
Pre – condizione	Context: VisualizzaStoricoOrdini :: doGet(request, response) Pre: L'acquirente è loggato.
Post – condizione	Post: ordini contiene tutti gli ordini sottomessi da parte dell'acquirente.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2.5. Aggiungi prodotto al carrello

Nome della classe	AggiungiProdottoAlCarrello
Descrizione	Questa classe è una servlet che si occupa di gestire l'aggiunta di un prodotto al carrello.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare l'aggiunta di un prodotto al carrello.
Pre – condizione	Context: AggiungiProdottoAlCarrello :: doGet(request, response) Pre: request.getParameter("idProdotto") != null && Formato idProdotto [A-Za-z0-9]{10} request.getParameter("Cart") != null
Post – condizione	Context: AggiungiProdottoAlCarrello :: doGet(request, response) Post: (se il prodotto aggiunto era già presente nel carrello: viene aumentata la quantità di quel prodotto se il prodotto aggiunto non era presente nel carrello: cart.getArticles().size() = cart.getArticles().size() + 1) && cart.getTotal() = cart.getTotal() + article.getPrice()

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2.6. InserisciDatiDiSpedizione

Nome della classe	InserisciDatiDiSpedizione
Descrizione	Questa classe è una servlet che si occupa di gestire l'aggiunta del numero di tracking e la data della consegna relativa all'ordine da gestire.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare l'aggiunta del numero di tracking e la data della consegna relativa all'ordine da gestire.
Pre – condizione	Context: InserisciDatiDiSpedizione :: doGet(request, response) Pre: request.getParameter("numDiTracking") != null && request.getParameter("numDiTracking") != " " && Formato numDiTracking [A-Za-z0-9]{5,15} && request.getParameter("corriere") != null && request.getParameter("corriere") != " " && Formato corriere [A-Za-z]{3,10} && request.getParameter("dataConsegna") != null && parsed != null && parsed.matches("[0-9]{4}-[1]{0-9}{2}-[1]{0-9}{2}!")
Post – condizione	Context: InserisciDatiDiSpedizione :: doGet(request, response) Post: order.getNumTracking() != null && !(order.getNumTracking().equals("")) && order.getCorriere() != null && !(order.getCorriere().equals("")) && order.getDataConsegna() != null && order.getStato().equals("in corso") order viene aggiornato e le modifiche vengono propagate al database.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2.7. Checkout

Nome della classe	Checkout
Descrizione	Questa classe è una servlet che si occupa di gestire la procedura di checkout.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la procedura di checkout.
Pre – condizione	Context: Checkout :: doGet(request, response) Pre: request.getParameter("Cart") != null && request.getParameter("Cart").getArticles().size() > 0 && request.getParameter("cartaCredito") != null && request.getParameter("indirizzoSpedizione") != null && request.getSession().getAttribute("user") != null L'indirizzo di spedizione e la carta di credito devono corrispondere all'Acquirente.
Post – condizione	Context: Checkout :: doGet(request, response) Post: cart.getArticles().isEmpty() usr.getCart().size() == 0 e il numero degli ordini di use è incrementato di uno. La disponibilità di tutti gli articoli in ordine viene decrementata per i relativi articoli in stock.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2.8. Gestione ordine

Nome della classe	GestioneOrdine
Descrizione	Questa classe è una servlet che si occupa di cambiare lo stato di un ordine da "Da spedire" ad "In corso" oppure a "Consegnato" e viceversa.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di cambiare lo stato di un ordine da "da spedire" ad "in corso" o a "Consegnato" e viceversa.
Pre – condizione	Context: GestioneOrdine :: doGet(request, response) Pre: request.getParameter("idOrdine") != null && Formato idOrdine [A-Za-z0-9]{10} L'utente connesso è un gestore degli ordini e idOrdine corrisponde veramente ad un id di un ordine nel database.
Post – condizione	Context: GestioneOrdine :: doGet(request, response) Post: order.getStato().equals("consegnato")

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2.9. Modifica quantità in carrello

Nome della classe	ModificaQuantitàInCarrello
Descrizione	Questa classe è una servlet che si occupa di modificare la quantità degli articoli presenti nel carrello.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di modificare la quantità degli articoli presenti nel carrello.
Pre – condizione	Context: ModificaQuantitàInCarrello :: doGet(request, response) Pre: request.getParameter("articoloId") != null && Formato articoloId [A-Za-z0-9]{10} && request.getParameter("quantità") != null && request.getSession().getAttribute("acquirente").getCart() != null
Post – condizione	Context: ModificaQuantitàInCarrello :: doGet(request, response) Post: if (cart.aumentaQuantità(articoloId) articolo.quantità += 1 if(cart.diminuisceQuantità(articoloId) articolo.quantità -= 1 L'articolo corrispondente all'articoloId presente nel carrello ha una quantità pari a "quantità".

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.2.10. Rimuovi prodotto dal carrello

Nome della classe	RimuoviProdottoDalCarrello
Descrizione	Questa classe è una servlet che si occupa di rimuovere un articolo presente nel carrello.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di rimuovere un articolo dal carrello.
Pre – condizione	Context: RimuoviProdottoDalCarrello :: doGet(request, response) Pre: request.getParameter("articoloId") != null && Formato articoloId [A-Za-z0-9]{10} &&
Post – condizione	Context: ModificaQuantitàInCarrello :: doGet(request, response) Post: L'articolo non è presente nel carrello.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.3. Gestione utenti

3.3.1. Login

Nome della classe	Login
Descrizione	Questa classe è una servlet che si occupa di gestire la procedura di login.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la procedura di login.
Pre – condizione	Context: Login :: doGet(request, response) Pre: request.getParameter("Username") != null && request.getParameter("Username") != " " && formato Username [A-Za-z0-9]{5,15} && request.getParameter("Password") != null && request.getParameter("Password") != " " && formato Password [A-Za-z0-9]{5,15}
Post – condizione	Context: Login :: doGet(request, response) Post: se il login viene effettuato da un user : request.getSession().getAttribute("user") != null se il login viene effettuato dal gestore ordini: request.getSession().getAttribute("gestoreOrdini") != null

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.3.2. Logout

Nome della classe	Logout
Descrizione	Questa classe è una servlet che si occupa di gestire la procedura di logout.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la procedura di logout.
Pre – condizione	Context: Logout :: doGet(request, response)
Post – condizione	//

3.3.3. Rimuovere Carta

Nome della classe	RimuovereCarta
Descrizione	Questa classe è una servlet che si occupa di gestire la rimozione di una carta di credito.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la procedura di rimozione di una carta di credito.
Pre – condizione	Context: RimuovereCarta :: doGet(request, response) Pre: L'acquirente è loggato e l'id corrisponde ad una sua carta
Post – condizione	Context: RimuovereCarta :: doGet(request, response) Post: L'id non corrisponde ad una carta dell'acquirente perché è stata rimossa. usr.getCarte().size -= 1

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.3.4. Rimuovere indirizzo

Nome della classe	RimuovereIndirizzo
Descrizione	Questa classe è una servlet che si occupa di gestire la rimozione di un indirizzo di spedizione.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la procedura di rimozione di un indirizzo di spedizione.
Pre – condizione	Context: RimuovereIndirizzo :: doGet(request, response) Pre: L'acquirente è loggato e l'id corrisponde ad un suo indirizzo di spedizione.
Post – condizione	Context: RimuovereIndirizzo :: doGet(request, response) Post: L'id non corrisponde ad un indirizzo di spedizione dell'acquirente perché è stato rimosso. usr.getIndirizzi().size -= 1

3.3.5. Visualizza login

Nome della classe	VisualizzaLogin
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione della pagina di login.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione della pagina di login.
Pre – condizione	Context: VisualizzaLogin :: doGet(request, response) Pre: L'utente non è loggato.
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.3.6. Visualizza ordine

Nome della classe	VisualizzaOrdine
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione di un ordine.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione di un ordine.
Pre – condizione	Context: VisualizzaOrdine :: doGet(request, response) Pre: orderId != null && corrisponde ad un ordine nel database. L'acquirente è loggato && l'ordine è associato ad esso o l'utente è un gestore degli ordini.
Post – condizione	//

3.3.7. Visualizza Profilo

Nome della classe	VisualizzaProfilo
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione del profilo di un'acquirente.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione del profilo di un'acquirente.
Pre – condizione	Context: VisualizzaProfilo :: doGet(request, response) Pre: L'acquirente è loggato.
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.3.8. Welcome

Nome della classe	Welcome
Descrizione	Questa classe è una servlet che si occupa di gestire la visualizzazione della pagina di welcome.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la visualizzazione della pagina di welcome.
Pre – condizione	Context: Welcome :: doGet(request, response)
Post – condizione	//

3.3.9. Inserire carta di credito

Nome della classe	InserireCarta
Descrizione	Questa classe è una servlet che si occupa di gestire l’inserimento di una carta di credito.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare l’inserimento di una carta di credito.
Pre – condizione	Context: InserireCarta :: doGet(request, response) Pre: L’acquirente è loggato. numcc != null && numcc.matches("[0-9]{16}") intestatario != null && intestatario.matches("[A-Za-z0-9]{4,40}") circuito != null && circuito.matches("[A-Za-z]{4,20}") scadenza != null && scadenza.matches("[0-9]{2}/[1]{1}[0-9]{4}") cvv != null && cvv.matches("[0-9]{3}")
Post – condizione	Post: usr.getCarte().size() += 1

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.3.10. Inserire indirizzo di spedizione

Nome della classe	InserireIndirizzo
Descrizione	Questa classe è una servlet che si occupa di gestire l'inserimento di un indirizzo di spedizione.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare l'inserimento di un indirizzo di spedizione.
Pre – condizione	Context: InserireIndirizzo :: doGet(request, response) Pre: L'acquirente è loggato. indirizzo != null && indirizzo.matches("[A-Za-z0-9]{5,40}") civico != null && civico.matches("[0-9]{1,4}") cap != null && cap.matches("[0-9]{5}") provincia != null && provincia.matches("[A-Z]{2}") stato != null && stato.matches("[A-Za-z]{5,30}")
Post – condizione	Post: usr.getIndirizzi().size() += 1

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.4. ArticoloInOrderModel

Nome della classe	ArticoloInOrderModel
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti gli articoli in ordine.
Metodi	+doRetrieveByIdInOrder (String codiceProdotto) : ArticoloInOrder articolo +doRetrieveAllInOrder () : List<ArticoloInOrder> articoli +restituisceArticoliAssociatiAdUnOrdine (String codiceOrdine) : List<ArticoloInOrder> articoli +addArticle(ArticoloInOrder a, Order o) : void

Nome Metodo	+doRetrieveByIdInOrder (String codiceProdotto) : ArticoloInOrder articolo
Descrizione	Questo metodo si occupa di verificare se nel database è presente un articolo in ordine tramite un codice specifico preso in input.
Pre – condizione	Context: ArticoloInOrderModel :: doRetrieveByIdInOrder (codiceProdotto) Pre: codiceProdotto != null
Post – condizione	//

Nome Metodo	+doRetrieveAllInOrder() : List<ArticoloInOrder> articoli
Descrizione	Questo metodo si occupa di effettuare la ricerca di tutti gli articoli in ordine presenti nel database.
Pre – condizione	//
Post – condizione	//

Nome Metodo	+restituisceArticoliAssociatiAdUnOrdine (String codiceOrdine) : List<ArticoloInOrder> articoli
Descrizione	Questo metodo si occupa di effettuare la ricerca di tutti gli articoli in ordine associati ad un determinato ordine presenti nel database.
Pre – condizione	Context: ArticoloInOrderModel :: restituisciArticoliAssociatiAdUnOrdine (codiceOrdine) Pre: codiceOrdine != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Nome Metodo	+addArticle (ArticoloInOrder a, Order o) : void
Descrizione	Questo metodo si occupa di aggiungere un articolo ad un ordine.
Pre – condizione	Context: ArticoloInOrderModel :: addArticle(a,o) Pre: a != null && o != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.5. ArticoloInStockModel

Nome della classe	ArticoloInStockModel
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti gli articoli in stock.
Metodi	+doRetrieveByIdInStock (String codiceProdotto) : ArticoloInStock articolo +doRetrieveAllInStock () : List<ArticoloInStock> articoli +doRetrieveSimpleSearch (String daCercare) : List<ArticoloInStock> articoli +doRetrieveAdvancedSearch (String daCercare, String marca, double minPrice, double maxPrice, String colore) : List<ArticoloInStock> articoli +updateDisponibilità (ArticoloInStock toSave) : void

Nome Metodo	+doRetrieveByIdInStock (String codiceProdotto) : ArticoloInStock articolo
Descrizione	Questo metodo si occupa di verificare se nel database è presente un articolo in stock tramite un codice specifico preso in input.
Pre – condizione	Context: ArticoloInStockModel :: doRetrieveByIdInStock (codiceProdotto) Pre: codiceProdotto != null
Post – condizione	//

Nome Metodo	+doRetrieveAllInStock () : List<ArticoloInStock> articoli
Descrizione	Questo metodo si occupa di effettuare la ricerca di tutti gli articoli in stock presenti nel database.
Pre – condizione	//
Post – condizione	//

Nome Metodo	+doRetrieveSimpleSearch (String daCercare) : List<ArticleInStock> articoli
Descrizione	Questo metodo si occupa di effettuare la ricerca di prodotti per stringa immessa dall'utente come parametro di ricerca.
Pre – condizione	Context: ArticoloInStockModel :: doRetrieveSimpleSearch (daCercare) Pre: daCercare != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Nome Metodo	+doRetrieveAllInStock () : List<ArticoloInStock> articoli
Descrizione	Questo metodo si occupa di effettuare l'aggiornamento della disponibilità di un articolo in stock.
Pre – condizione	Context: ArticoloInStockModel :: updateDisponibilità (toSave) Pre: toSave corrisponde ad un articolo in stock presente nel database && toSave != null
Post – condizione	Context: ArticoloInStockModel :: updateDisponibilità (toSave) Post: aggiornata disponibilità correttamente dell'articolo toSave

Nome Metodo	+doRetrieveAdvancedSearch (String daCercare, String marca, double minPrice, double maxPrice, String colore) : List<ArticoloInStock> articoli
Descrizione	Questo metodo si occupa di effettuare la ricerca di prodotti che matchano con i vari parametri immessi dall'utente nell'apposito form.
Pre – condizione	Context: ArticoloInStockModel :: doRetrieveAdvancedSearch (daCercare, marca, minPrice, maxPrice, colore) Pre: daCercare != null marca != null minPrice != null maxPrice != null colore != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.6. AcquirenteModel

Nome della classe	UserModel
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti l'acquirente ed il carrello ad esso associato.
Metodi	+doRetrieveById (String userName): Acquirente acquirente +checkLogin (String username, String password): Acquirente acquirente +doRetrieveCartByUser (String userName): Cart cart +updateAcquirente (Acquirente toUpdate): void +updateCart (Acquirente acquirente): void +dropCart (Acquirente acquirente): void

Nome Metodo	+doRetrieveById (String username): Acquirente acquirente
Descrizione	Questo metodo si occupa di verificare se nel database è presente un Acquirente tramite una username specifica presa in input.
Pre – condizione	Context: AcquirenteModel :: doRetrieveById (username) Pre: username != null
Post – condizione	//

Nome Metodo	+checkLogin (String username, String password): Acquirente acquirente
Descrizione	Questo metodo si occupa di verificare se i dati immessi dall'utente per effettuare il login sono presenti nel database.
Pre – condizione	Context: AcquirenteModel :: checkLogin(username, password) Pre: username != null && password != null
Post – condizione	//

Nome Metodo	+doRetrieveCartByUser (String username): Cart cart
Descrizione	Questo metodo si occupa di verificare se il carrello associato ad un acquirente è presente nel database.
Pre – condizione	Context: AcquirenteModel :: doRetrieveCartByUser (username) Pre: username != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Nome Metodo	+updateAcquirente (Acquirente toUpdate): void
Descrizione	Questo metodo si occupa di aggiornare i dati di un Acquirente presente nel database.
Pre – condizione	Context: AcquirenteModel :: updateAcquirente (toUpdate) Pre: toUpdate != null
Post – condizione	//

Nome Metodo	+updateCart (Acquirente acquirente): void
Descrizione	Questo metodo si occupa di aggiornare la lista degli articoli di un carrello associato ad un Acquirente presente nel database.
Pre – condizione	Context: AcquirenteModel :: updateCart (acquirente) Pre: acquirente != null
Post – condizione	//

Nome Metodo	+dropCart (Acquirente acquirente): void
Descrizione	Questo metodo si occupa di svuotare un carrello associato ad un acquirente presente nel database.
Pre – condizione	Context: AcquirenteModel :: dropCart (acquirente) Pre: acquirente != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.7. GestoreOrdiniModel

Nome della classe	GestoreOrdiniModel
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti i gestori degli ordini.
Metodi	+doRetrieveById (String userName): GestoreOrdini gestore +checkLogin (String username, String password): GestoreOrdini gestore

Nome Metodo	+doRetrieveById (String username): GestoreOrdini gestore
Descrizione	Questo metodo si occupa di verificare se nel database è presente un GestoreOrdini tramite una username specifica presa in input.
Pre – condizione	Context: GestoreOrdiniModel :: doRetrieveById (username) Pre: username != null
Post – condizione	//

Nome Metodo	+checkLogin (String username, String password): GestoreOrdini gestore
Descrizione	Questo metodo si occupa di verificare se i dati immessi dal gestore ordini per effettuare il login sono presenti nel database.
Pre – condizione	Context: GestoreOrdiniModel :: checkLogin (username, password) Pre: username != null && password != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.8. OrderModel

Nome della classe	OrderModel
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti gli ordini.
Metodi	+doRetrieveById (String idOrdine) : Order ordine +doRetrieveAll () : List<Order> ordini +createOrder (Order toCreate): void +updateOrder (Order toUpdate) : void +doRetrieveByAcquirente (Acquirente acquirente) : List<Order> ordini

Nome Metodo	+doRetrieveById(String idOrdine) : Order ordine
Descrizione	Questo metodo si occupa di effettuare la ricerca di un ordine per id.
Pre – condizione	Context: OrderModel :: doRetrieveById (idOrder) Pre: idOrder != null
Post – condizione	//

Nome Metodo	+doRetrieveAll () : List<Order> ordini
Descrizione	Questo metodo si occupa di effettuare la ricerca di tutti gli ordini.
Pre – condizione	//
Post – condizione	//

Nome Metodo	+createOrder (Order toCreate) : void
Descrizione	Questo metodo si occupa di rendere persistente un nuovo ordine.
Pre – condizione	Context: OrderModel :: createOrder (toCreate) Pre: toCreate != null
Post – condizione	//

Nome Metodo	+updateOrder (Order toUpdate) : void
Descrizione	Questo metodo si occupa di effettuare l'aggiornamento di un ordine.
Pre – condizione	Context: OrderModel :: updateOrder (toUpdate) Pre: toUpdate != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Nome Metodo	+doRetrieveByAcquirente (Acquirente acquirente) : List<Order> ordini
Descrizione	Questo metodo si occupa di ricercare tutti gli ordini effettuati da un determinato acquirente.
Pre – condizione	Context: OrderModel :: doRetrieveByAcquirente (acquirente) Pre: acquirente != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.9. ShippingAddressModel

Nome della classe	ShippingAddressModel
Descrizione	Questa classe è un manager che si occupa di interagire con il database.
Metodi	+doRetrieveById (String idShip): ShippingAddress indirizzo +doRetrieveByUser (String username): List<ShippingAddress> indirizzi +createShippingAddress (ShippingAddress toCreate) : void +updateShippingAddress (ShippingAddress toUpdate) : void +deleteShippingAddress (ShippingAddress toDelete) : void

Nome Metodo	+doRetrieveById (String idShip) : ShippingAddress indirizzo
Descrizione	Questo metodo si occupa di effettuare la ricerca di un indirizzo di spedizione per id.
Pre – condizione	Context: ShippingAddressModel :: doRetrieveById (idShip) Pre: idShip != null
Post – condizione	//

Nome Metodo	+doRetrieveByUser (String username) : List<ShippingAddress> indirizzi
Descrizione	Questo metodo si occupa di effettuare la ricerca degli indirizzi di spedizione per user.
Pre – condizione	Context: ShippingAddressModel :: doRetrieveByUser (username) Pre: username != null
Post – condizione	//

Nome Metodo	+createShippingAddress (ShippingAddress toCreate) : void
Descrizione	Questo metodo si occupa di rendere persistente un nuovo indirizzo di spedizione.
Pre – condizione	Context: ShippingAddressModel :: createShippingAddress (toCreate) Pre: toCreate != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Nome Metodo	+updateShippingAddress (ShippingAddress toUpdate) : void
Descrizione	Questo metodo si occupa di effettuare l'aggiornamento di un indirizzo di spedizione.
Pre – condizione	Context: ShippingAddressModel :: updateShippingAddress (toUpdate) Pre: toUpdate != null
Post – condizione	//

Nome Metodo	+deleteShippingAddress (ShippingAddress toDelete) : void
Descrizione	Questo metodo si occupa di effettuare la rimozione di un indirizzo di spedizione.
Pre – condizione	Context: ShippingAddressModel :: deleteShippingAddress (toDelete) Pre: toDelete != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.10. CreditCardModel

Nome della classe	CreditCardModel
Descrizione	Questa classe è un manager che si occupa di interagire con il database. Gestisce le query riguardanti le carte di credito
Metodi	+doRetrieveById (String idCarta): CreditCard cartaDiCredito +doRetrieveByUser (String username): List<CreditCard> carte +createCreditCard (CreditCard toCreate) : void +updateCreditCard (CreditCard toUpdate) : void +deleteCreditCard (CreditCard toDelete) : void

Nome Metodo	+doRetrieveById (String idCarta) : CreditCard cartaDiCredito
Descrizione	Questo metodo si occupa di effettuare la ricerca di una carta di credito per id.
Pre – condizione	Context: CreditCardModel :: doRetrieveById(idCarta) Pre: idCarta != null
Post – condizione	//

Nome Metodo	+doRetrieveByUser (String username) : List<CreditCard> carte
Descrizione	Questo metodo si occupa di effettuare la ricerca delle carte di credito per user.
Pre – condizione	Context: CreditCardModel :: doRetrieveByUser (username) Pre: username != null
Post – condizione	//

Nome Metodo	+createCreditCard (CreditCard toCreate) : void
Descrizione	Questo metodo si occupa di inserire una nuova carta di credito nel database.
Pre – condizione	Context: CreditCardModel :: createCreditCard (toCreate) Pre: toCreate != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

Nome Metodo	+updateCreditCard (CreditCard toUpdate) : void
Descrizione	Questo metodo si occupa di aggiornare una carta di credito nel database.
Pre – condizione	Context: CreditCardModel :: updateCreditCard (toUpdate) Pre: toUpdate != null
Post – condizione	//

Nome Metodo	+deleteCreditCard (CreditCard toDelete) : void
Descrizione	Questo metodo si occupa di cancellare una carta di credito nel database.
Pre – condizione	Context: CreditCardModel :: deleteCreditCard (toDelete) Pre: toDelete != null
Post – condizione	//

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.11. Acquirente

Acquirente
-Username : String -Password : String -Nome : String -Cognome : String -Email : String -DataNascita : Date -Cc : ArrayList<CreditCard> -ShipAdd : ArrayList<ShippingAddress> -Cart : Cart
+getUsername() : String +setUsername(Username : String) : void +getPassword() : String +setPassword>Password : String) : void +getNome() : String +setNome(Nome : String) : void +getCognome() : String +setCognome(Cognome : String) : void +getEmail() : String +setEmail>Email : String) : void +getDataNascita() : Date +setDataNascita(DataNascita : Date) : void +toString() : String +getCc() : ArrayList<CreditCard> +setCc(Cc : ArrayList<CreditCard>) : void +getShipAdd() : ArrayList<ShippingAddress> +setShipAdd(shipAdd : ArrayList<ShippingAddress>) : void +getCart() : Cart +setCart(Cart : Cart) : void +checkCC(int creditCardId) : CreditCard +checkSA(int shippingAddressId) : ShippingAddress +resetCart() : void +addShippingAddress(ShippingAddress sa) : void +addCreditCard(CreditCard cc) : void +removeShippingAddress(int id) : boolean +removeCreditCard(int id) : boolean

Descrizione

Questa classe rappresenta l'acquirente

checkCC (int creditCardId): CreditCard

Questo metodo controlla se la carta scelta per il pagamento esiste nella lista delle carte dell'Acquirente.

checkSA (int shippingAddressId): ShippingAddress

Questo metodo controlla se l'indirizzo scelto per il checkout esiste nella lista degli indirizzi dell'Acquirente.

resetCart (): void

Questo metodo resetta il carrello dopo aver completato un acquisto.

addShippingAddress (ShippingAddress sa): void

Questo metodo aggiunge un nuovo indirizzo di spedizione alla lista degli indirizzi dell'Acquirente.

addCreditCard (CreditCard cc): void

Questo metodo aggiunge una nuova carta di credito alla lista delle carte di credito dell'Acquirente.

removeShippingAddress (int id): boolean

Questo metodo rimuove un indirizzo di spedizione dalla lista degli indirizzi dell'acquirente.

removeCreditCard (int id): boolean

Questo metodo rimuove una carta di credito dalla lista delle carte di credito dell'acquirente.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.12. Articolo in Stock

a ArticoloInStock
-Codice : Integer -Modello : String -Marca : String -Img1 : String -Img2 : String -Img3 : String -Descrizione : String -Prezzo : double -Disponibilità : Integer
+getCodice() : Integer +setCodice(Codice : Integer) : void +getModello() : String +setModello(Modello : String) : void +getMarca() : String +setMarca(Marca : String) : void +getImg1() : String +setImg1(Img1 : String) : void +getImg2() : String +setImg2(Img2 : String) : void +getImg3() : String +setImg3(Img3 : String) : void +getDescrizione() : String +setDescrizione(Descrizione : String) : void +getPrezzo() : double +setPrezzo(Prezzo : double) : void +getDisponibilità() : Integer +setDisponibilità(Disponibilità : Integer) : void

Descrizione

Questa classe rappresenta l'articolo presente nel catalogo.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.13. Articolo in Order

ArticoloInOrder
-Codice : Integer -Marca : String -Img1 : String -Img2 : String -Img3 : String -Descrizione : String -Prezzo : double -Quantità : Integer -Order : Order
+getCodice() : Integer +setCodice(Codice : Integer) : void +getModello() : String +setModello(Modello : String) : void +getMarca() : String +setMarca(Marca : String) : void +getImg1() : String +setImg1(Img1 : String) : void +getImg2() : String +setImg2(Img2 : String) : void +getImg3() : String +setImg3(Img3 : String) : void +getDescrizione() : String +setDescrizione(Descrizione : String) : void +getPrezzo() : double +setPrezzo(Prezzo : double) : void +getQuantità() : Integer +setQuantità(Quantità : Integer) : void +getOrder() : Order +setOrder(Order : Order) : void

Descrizione

Questa classe rappresenta l'articolo presente negli ordini e quindi con le caratteristiche che aveva al momento dell'acquisto.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.14. Cart

Cart
-Prezzo : double -Articoli : Map<ArticoloInStock, Integer>
+getPrezzo() : double +setPrezzo(Prezzo : double) : void +getArticoli() : Map<ArticoloInStock, Integer> +setArticoli(Articoli : Map<ArticoloInStock, Integer>) : void +getNumArticoli() : Integer +addArticle(ArticoloInStock ais) : void +removeArticle(ArticoloInStock ais) : void +setArticle(ArticoloInStock a, Integer quantità) : void +mergeCart(Cart c) : void

Descrizione

Questa classe rappresenta il carrello.

addArticle (ArticoloInStock ais): void

Questo metodo consente l'aggiunta di un articolo al carrello.

removeArticle (ArticoloInStock ais): void

Questo metodo consente la rimozione di un articolo dal carrello.

setArticle (ArticoloInStock ais, Integer quantità): void

Questo metodo consente l'aggiornamento della quantità di un articolo nel carrello.

mergeCart (Cart c): void

Questo metodo consente ad un acquirente di effettuare il merge del carrello riempito prima del login.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.15. Gestore ordini

^a GestoreOrdini
-Username : String -Password : String -Nome : String -Cognome : String -Email : String -Matricola : String
+getUsername() : String +setUsername(Username : String) : void +getPassword() : String +setPassword>Password : String) : void +getNome() : String +setNome(Nome : String) : void +setCognome(Cognome : String) : void +getCognome() : String +getEmail() : String +setEmail>Email : String) : void +getMatricola() : String +setMatricola(Matricola : String) : void +toString() : String

Descrizione

Questa classe rappresenta il gestore degli ordini.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.16. Order

a	Order
	-Codice : String -DataEsecuzione : Date -Prezzo : double -StatoOrdine : String -DataConsegna : Date -NumeroTracking : String -Corriere : String -Acquirente : Acquirente -ShippingAddress : ShippingAddress -CreditCard : CreditCard -ListaDiArticoli : List<ArticoloInOrder>
	+getCodice() : String +setCodice(Codice : String) : void +getDataEsecuzione() : Date +setDataEsecuzione(DataEsecuzione : Date) : void +getPrezzo() : double +setPrezzo(Prezzo : double) : void +getStatoOrdine() : String +setStatoOrdine(StatoOrdine : String) : void +getDataConsegna() : Date +setDataConsegna(DataConsegna : Date) : void +getNumeroTracking() : String +setNumeroTracking(NumeroTracking : String) : void +toString() : String +getCorriere() : String +setCorriere(Corriere : String) : void +getAcquirente() : Acquirente +setAcquirente(Acquirente : Acquirente) : void +getShippingAddress() : ShippingAddress +setShippingAddress(ShippingAddress : ShippingAddress) : void +getCreditCard() : CreditCard +setCreditCard(CreditCard : CreditCard) : void +getListaDiArticoli() : List<ArticoloInOrder> +setListaDiArticoli(ListaDiArticoli : List<ArticoloInOrder>) : void

Descrizione

Questa classe rappresenta l'ordine.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.17. Shipping Address

a	ShippingAddress
-Codice : String -Stato : String -Provincia : String -CAP : int -Indirizzo : String -NumeroCivico : int -Acquirente : Acquirente	
+getCodice() : String +setCodice(Codice : String) : void +getStato() : String +setStato(Stato : String) : void +getProvincia() : String +setProvincia(Provincia : String) : void +getCAP() : int +setCAP(CAP : int) : void +getIndirizzo() : String +setIndirizzo(Indirizzo : String) : void +getNumeroCivico() : int +setNumeroCivico(NumeroCivico : int) : void +toString() : String +getAcquirente() : Acquirente +setAcquirente(Acquirente : Acquirente) : void	

Descrizione

Questa classe rappresenta l'indirizzo di spedizione.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

3.18. Credit card

a CreditCard
-Codice : String -NumeroCC : String -Intestatario : String -Circuito : String -DataScadenza : Date -CvcCvv : int -Acquirente : Acquirente -Modello : String
+getCodice() : String +setCodice(Codice : String) : void +getNumeroCC() : String +setNumeroCC(NumeroCC : String) : void +getIntestatario() : String +setIntestatario(Intestatario : String) : void +getCircuito() : String +setCircuito(Circuito : String) : void +getDataScadenza() : Date +setDataScadenza(DataScadenza : Date) : void +getCvcCvv() : int +setCvcCvv(CvcCvv : int) : void +toString() : String +getAcquirente() : Acquirente +setAcquirente(Acquirente : Acquirente) : void +getLastCC() : String

Descrizione

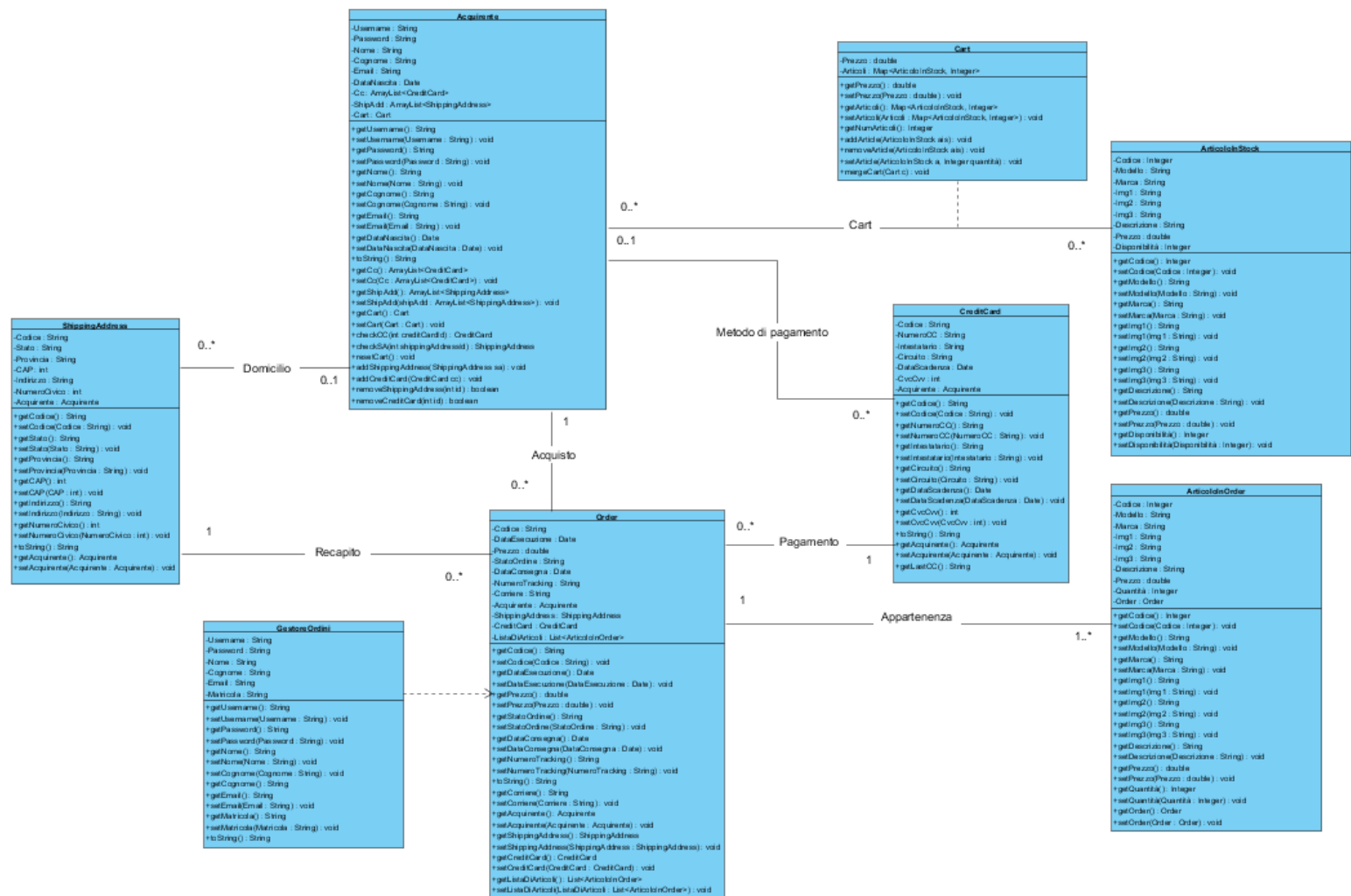
Questa classe rappresenta la carta di credito.

getLastCC(): String

Questo metodo consente di ricavare gli ultimi 4 numeri della carta di credito.

Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

4. Class diagram finale



Progetto: Quattrocchi.it	Versione: 2.2
Documento: Object Design Document	Data: 10/01/2018

5. Sequence diagram finali

I sequence diagram rappresentanti le interazioni all'interno del sistema implementato, sono riportati nel documento esterno chiamato "SEQUENCE DIAGRAM FINALI".