

**Università degli Studi di Salerno**  
Corso di Ingegneria del Software

**CoinVerter**  
**Object Design Document**  
**Versione 1.1**



Data: 16/12/2024

Progetto: CoinVerter	Versione: 1.1
Documento: Object Design Document	Data: 16/12/2024

**Coordinatore del progetto:**

Nome	Matricola

**Partecipanti:**

Nome	Matricola
Pastore Alfredo	0512108925
Perna Alessandro	0512118963

<b>Scritto da:</b>	Pastore Alfredo
--------------------	-----------------

**Revision History**

Data	Versione	Descrizione	Autore
15/12/2024	1.0	Prima versione del documento	Pastore Alfredo
16/12/2024	1.1	Completata sezione interfacce di sistema	Pastore Alfredo

# Indice

1.	INTRODUZIONE .....	4
1.1.	Design Trade-off .....	4
1.1.1.	Gestione della sessione .....	4
1.1.2.	Gestione portafoglio.....	4
1.1.3.	Gestione storico ordini .....	4
1.2.	Linee guida per la documentazione delle Interfacce.....	4
2.	Struttura progetto .....	5
3.	Interfacce classi .....	5
3.1.	Registro utenti .....	5
3.2.	Carrello.....	6
3.3.	Catalogo .....	6
3.4.	Ordine.....	7

# 1. INTRODUZIONE

## *1.1.Design Trade-off*

### **1.1.1. Gestione della sessione**

Il client web dev'essere in grado di ricordare e riconoscere un utente loggato, per fare questo fa uso dell'oggetto legato all'utente. Il primo trade off di questo approccio è quello di carico vs responsivness.

Avendo scelto di caricare e mantenere più informazioni all'interno della sessione, stiamo andando ad aggiungere carico sulla macchina dell'utente risparmiando però diverse comunicazioni con il server che, a seconda dello stato della connessione, poteva dare la sensazione di un sito lento e poco ottimizzato.

### **1.1.2. Gestione portafoglio**

Il portafoglio corrisponde all'insieme di tutte le valute digitali mai acquistate dall'utente e viene caricato insieme al profilo dello stesso al login. La scelta da effettuare qui è legata all'aggiornamento all'interno del database.

Per garantire quanto più possibile la consistenza dei dati, si è deciso di aggiornare le informazioni relative al portafoglio dopo ogni acquisto; questa decisione comporta un carico di lavoro maggiore sul lato server dato che un utente può effettuare più acquisti in una singola sessione, ma la soluzione con l'aggiornamento al logout è stata riputata troppo rischiosa, anche se più efficiente

### **1.1.3. Gestione storico ordini**

La gestione dello storico porta con sé due problemi di ottimizzazione, tempi di caricamento e spazio occupato, entrambi ricadono nel trade-off tra completezza vs performance e sono stati gestiti come segue:

Un gestore autenticato è in grado di accedere allo storico degli ordini effettuati, ma il sistema carica effettivamente i record dal database solo al momento dell'accesso alla sezione di "gestione ordini" evitando così che la macchina del gestore venga caricata inutilmente ed evitando anche rallentamenti.

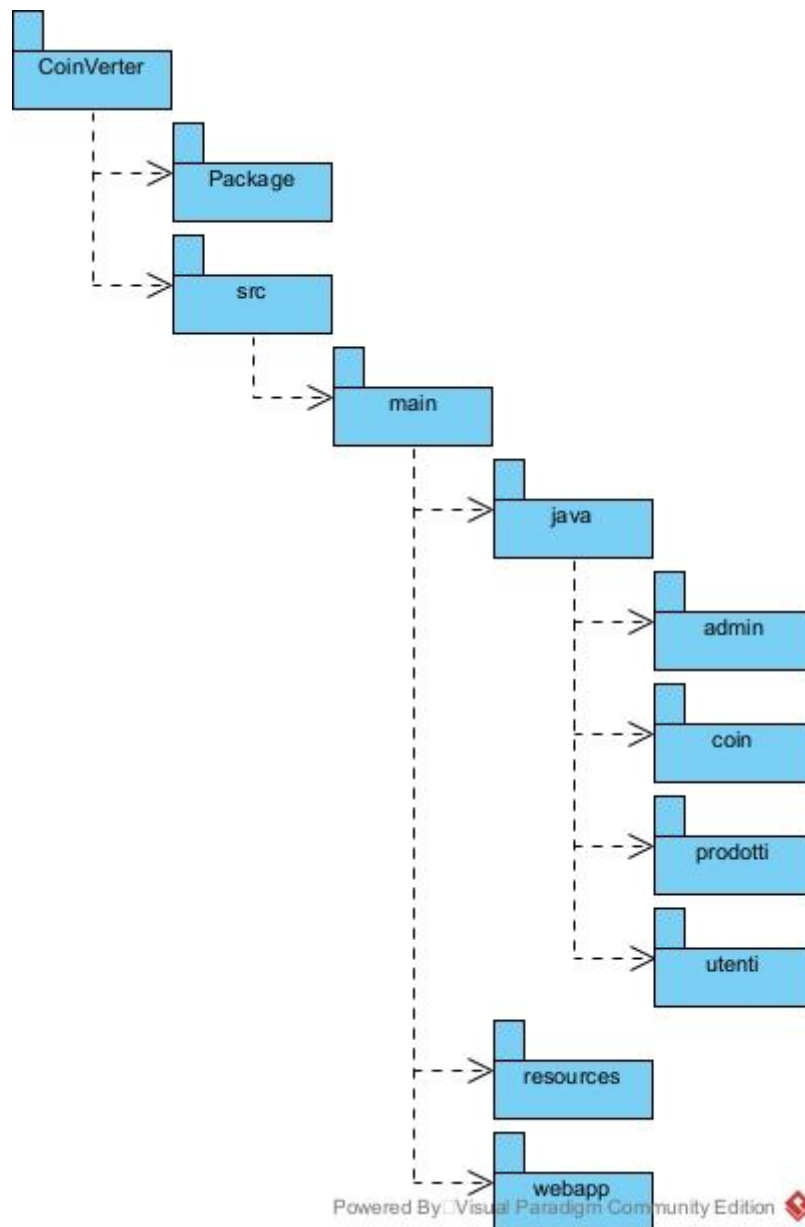
Dovendo accedere agli ordini esclusivamente per scopi di inventario si è deciso di limitare il numero di record disponibili sulla pagina a quelli di un singolo mese, permettendo così al gestore di svolgere il suo compito senza intaccare troppo le performance

## *1.2.Linee guida per la documentazione delle Interfacce*

Per garantire una buona leggibilità del codice e mantenere consistenza con le best practices Java:

- I nomi delle classi composti da 2 o più parole vanno inseriti in PascalCase
- I nomi delle variabili e dei parametri composti da 2 o più parole vanno inseriti in camelCase
- Gli errori vanno gestiti tramite eccezioni
  - Fanno esclusione gli errori web(EG 404,500,ecc...)

## 2. Struttura progetto



## 3. Interfacce classi

### 3.1. Registro utenti

Metodo	Contratti
+GetUser(email,password)	context RegistroUtenti::GetUser(Email:String,password:String):Utente  pre: self.GetUser(self.Email,self.password) = null  post: self.GetUser(self.Email,self.password) = self

### 3.2. Carrello

Metodo	Contratti
+addProduct(product)	context Carrello::addProduct(product:Prodotto):void  inv: self. elenco prodotti ->size() >= 0  post: self. elenco prodotti ->contains(product) AND self. elenco prodotti ->size()= @pre(self. elenco prodotti ->size())+1
+removeProduct(product)	context Carrello::removeProduct(product:Prodotto):void  inv: self.elenco prodotti->size() >= 0  pre:self.elenco prodotti-> contains(product)  post: self.elenco prodotti->size()= @pre(self. elenco prodotti ->size())-1
+getProducts()	context Carrello::getProducts():Arraylist
+acquista()	context Carrello::acquista():void  pre: self.elenco prodotti->size()>0  post self.elenco prodotti->size()=0

### 3.3. Catalogo

Metodo	Contratti
+addProduct(product)	Context Catalogo::addProduct(product:Prodotto):void  pre: self. elenco prodotti ->excludes(product)  post: self. elenco prodotti ->contains(product) AND self. elenco prodotti ->size()= @pre(self. elenco prodotti ->size())+1

+deleteProduct(product)	Context Catalogo::deleteProduct(product:Prodotto):void  pre: self.elenco prodotti->contains(product)  post: self.elenco prodotti ->excludes(product) AND self. elenco prodotti ->size()= @pre(self. elenco prodotti ->size())-1
+GetProducts()	Context Catalogo::getProducts():Arraylist
+modifyProduct(product)	Context Catalogo::modifyProduct(product:Prodotto):void  pre: self.elenco prodotti->contains(product)

### 3.4. Ordine

Metodo	Contratti
+addOrder(ordine)	Context Ordine::addOrder(ordine:Order):void  pre: self.elementi->excludes(product)  post: self. elementi ->contains(product) AND self. elementi ->size()= @pre(self. elementi ->size())+1
+removeOrder(ordine)	Context Ordine::removeOrder(ordine:Order):void  pre: self.elementi->contains(product)  post: self. elementi ->excludes(product) AND self. elementi ->size()= @pre(self. elementi ->size())-1