*Object Design*

*Document*

*Progetto*

*OctoPlus*



**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 06/12/2023 | **0.1** | **Prima stesura** | **Tutto il team** |
| 07/12/2023 | **0.2** | **Aggiunta sezione package + revisione documento** | **Donnarumma Salvatore** |

# 1. Introduzione

## 1.1 Object design trade-offs

### 1.1.1 Robustezza vs Tempo

Nei primi tempi non sono previsti picchi di utenza elevati, pertanto si è deciso di ritardare l’implementazione di funzionalità avanzate che renderebbero il sistema più robusto e capace di gestire uno stress maggiore, questo per velocizzare il lancio della prima versione.

### 1.1.2 Sicurezza vs Tempo

## Si ritiene che la sicurezza dei dati degli utenti registrati e l’attendibilità della piattaforma siano caratteristiche necessarie che la piattaforma dovrebbe avere fin dalla prima versione al fine di tutelare i clienti. Il tutto, naturalmente, richiede tempo per lo sviluppo il quale sarà sottratto alla robustezza.

## 1.2 Linee guida

Qui di seguito sono riportate alcune linee guide per la stesura del codice:

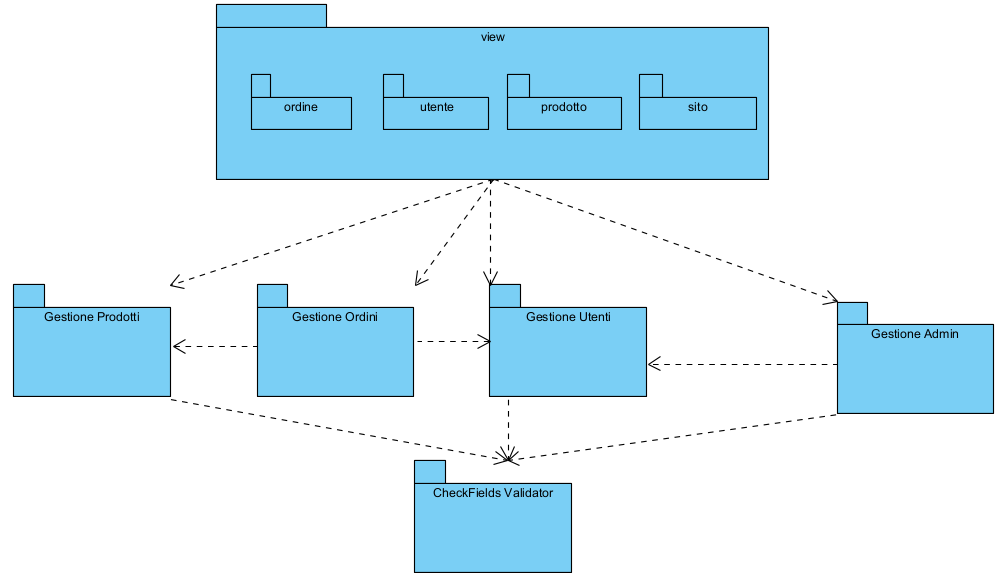
* Gli oggetti Dao dovranno avere il suffisso DaoDataSource.
* I nomi dei file jsp devono essere totalmente in minuscolo mentre i nomi delle Servlet e classi Bean, così come i DAO dovranno iniziare con la prima lettera in maiuscolo.

## 1.3 Referenze

* **R.A.D.**
* **S.D.D.**

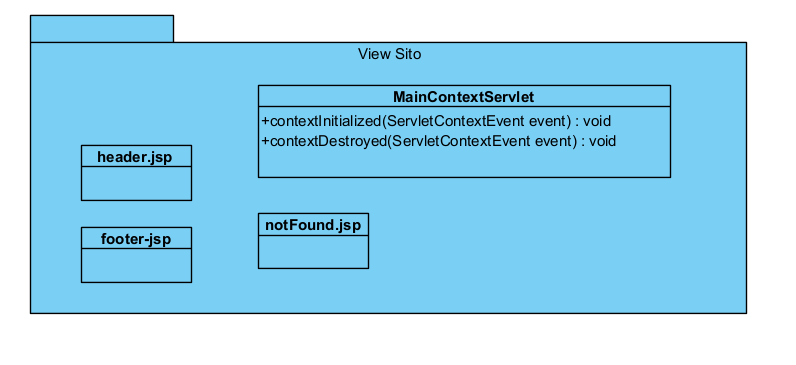
**2. Packages**

Nella sezione presente vengono mostrati nel dettaglio i packages implementati nel nostro sistema.



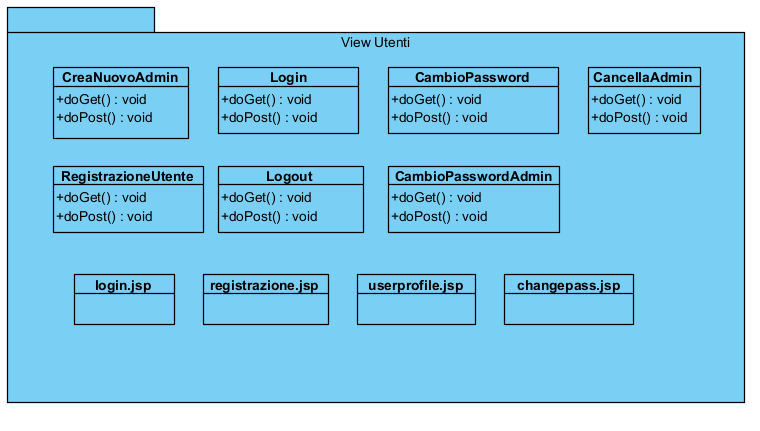
**2.1 View Sito**

Questo sottopacchetto è composto da Servlet e file jsp adibite alle funzioni generali del sito come footer, header, etc.



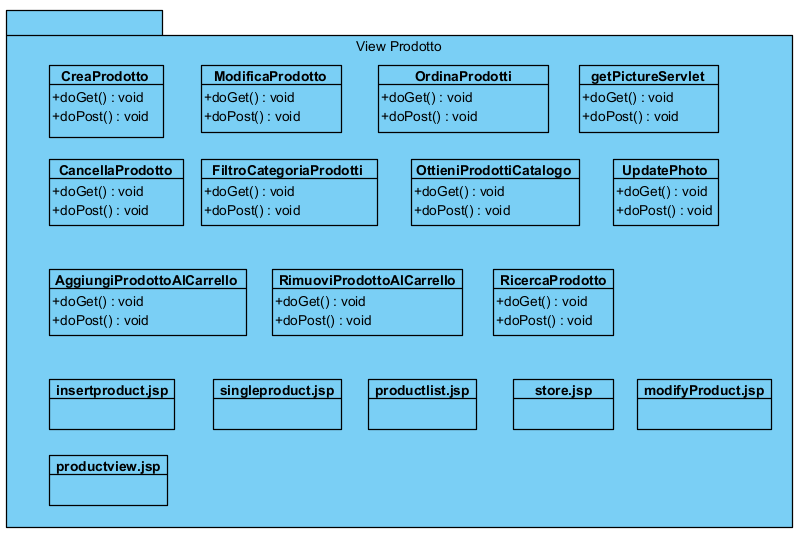
**2.2 View Utente**

Questo pacchetto è composto da Servlet e file jsp relativi alle funzionalità per la gestione degli utenti, come autenticazione, registrazione, cambio password, etc.



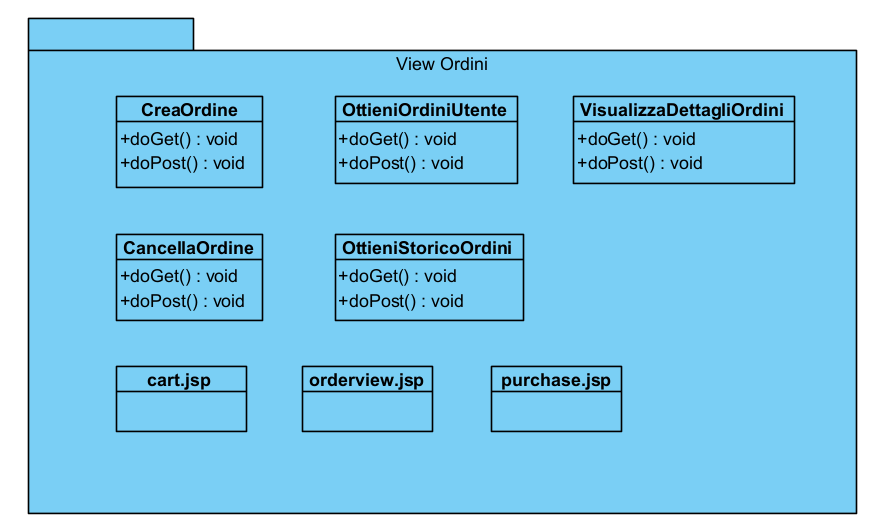
**2.3 View Prodotti**

Questo pacchetto è composto da Servlet e file jsp relativi alle funzionalità per la gestione dei prodotti come rimozione degli stessi dal catalogo, aggiunta, modifica, etc.



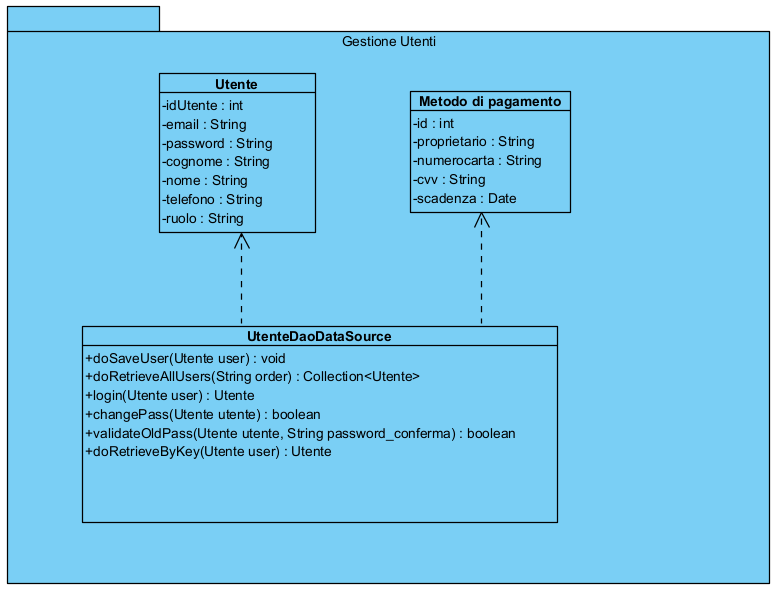
**2.4 View Ordini**

Questo pacchetto è composto da Servlet e file jsp relativi alle funzionalità per la gestione degli ordini come rimozione degli ordini, creazione dell’ordine (dunque acquisto di prodotti), visualizzazione, etc.



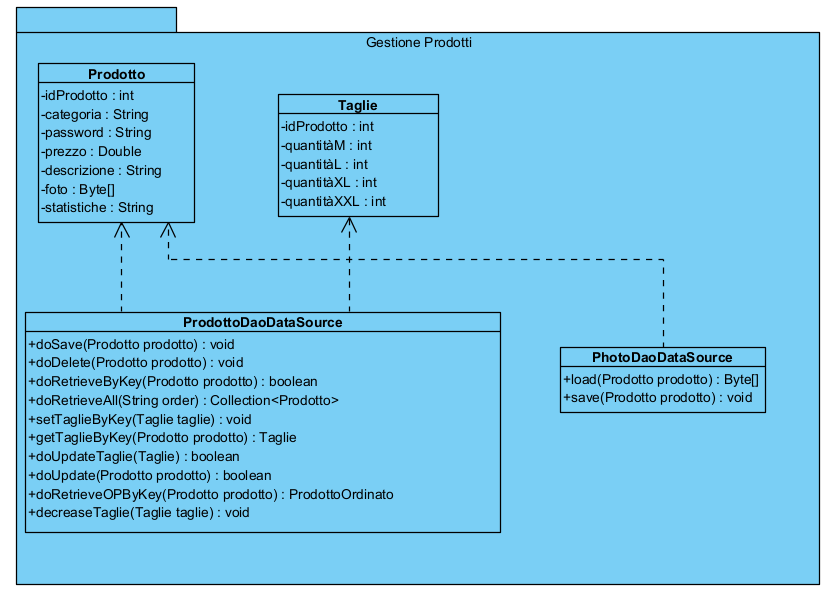
**2.5 Gestione Utenti**

Questo pacchetto contiene le classi Java (Bean e DAO) dedicati alle funzionalità per la gestione degli utenti e informazioni relative ad essi.



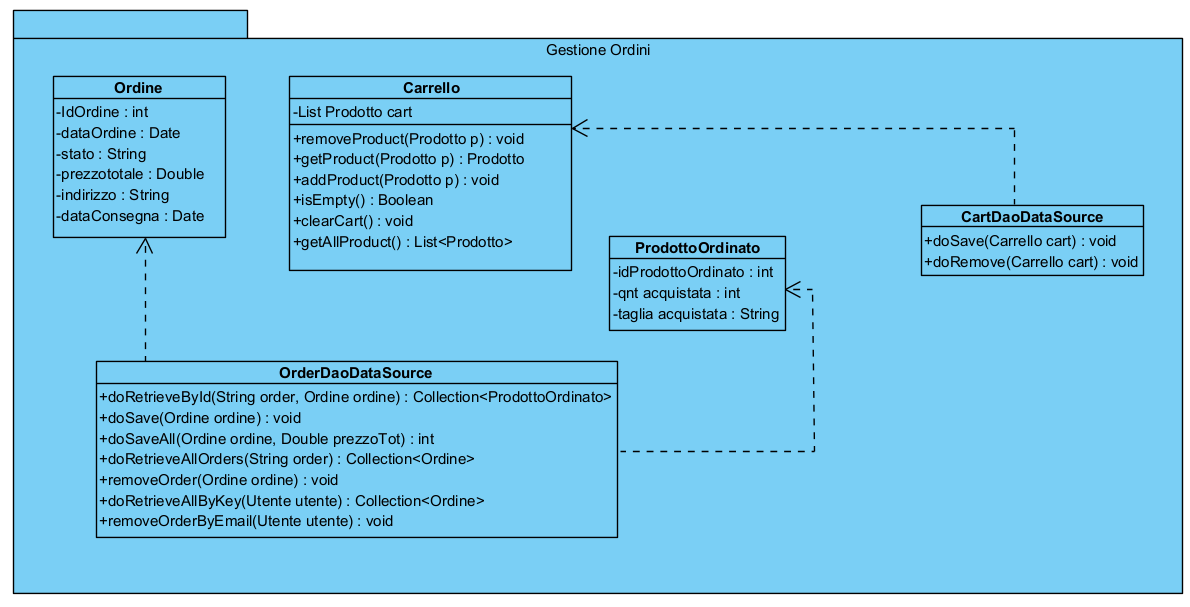
**2.6 Gestione Prodotti**

Questo pacchetto contiene le classi Java (Bean e DAO) dedicati alle funzionalità per la gestione dei prodotti e informazioni relative ad essi.

****

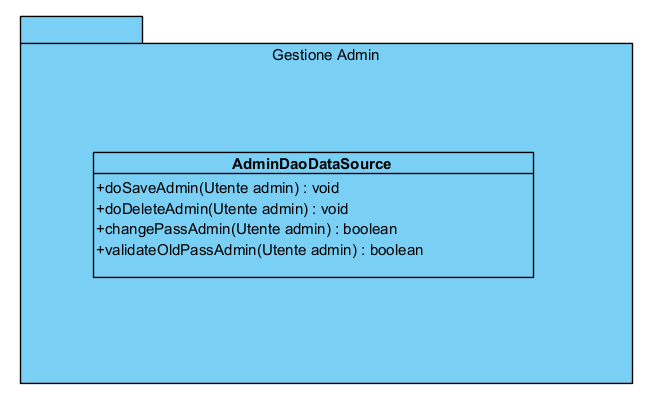
**2.7 Gestione Ordini**

Questo pacchetto contiene le classi Java (Bean e DAO) dedicati alle funzionalità per la gestione degli ordini e informazioni relative ad essi.

****

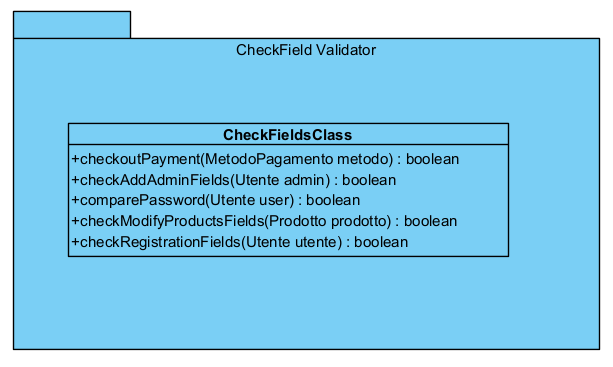
**2.8 Gestione Admin**

Questo pacchetto contiene le classi Java (Bean e DAO) dedicati alle funzionalità per la gestione degli admin.



**2.9 Checking Fields**

Questo pacchetto contiene la classe Java dedicata al controllo dei formati dei dati inseriti nei vari form presenti all’interno del sito.



# 3. Interfacce di classe

## 3.1 UtenteDaoDataSource

|  |  |
| --- | --- |
| UtenteDaoDataSource | Questa classe permette di interfacciarci al DBMS relazione modificando e interrogando l’entità Utente. |
| Precondizione | **PostCondizione** |
| context *UtenteDaoDataSource::doRetrieveByKey(Utente utente)*  pre: utente.email != null | **context** *UtenteDaoDataSource::doRetrieveByKey(Utente utente)*  **post**: return Utente, se la tupla non esiste nel DB l’oggetto restituito ha tutti i valori settati a null, altrimenti contiene le informazioni ricavate dalla tupla. |
| context *UtenteDaoDataSource::doSaveUser(Utente utente)*  pre: *utente.email != null and utente.email != “”*  *and utente.email è formattata correttamente*  *and DB not contains utente.email*  *and utente.password è formattata correttamente and password conferma== utente.password*  *and utente.nome != null and utente.nome != “”*  *and utente.cognome != null and utente.cognome != “”*  *and utente.numerotelefono è formattato correttamente* | **context** *UtenteDaoDataSource::doSaveUser(Utente utente)*  **post**: viene inserita nel DB una tupla contenente le informazioni di account |
| context *UtenteDaoDataSource::doRetrieveAllUsers(String order)*  pre: order == null or order == “Email” or order ==”Cognome” | **context** *UtenteDaoDataSource::doRetrieveAllUsers(String order)*  **post**: return List<Utente> contenente tutte le tuple della tabella |
| context *UtenteDaoDataSource::login(Utente utente)*  pre: utente.email != null and utente.email != “” and utente.password != null and utente.password != “” | **context** *UtenteDaoDataSource::login(Utente utente)*  **post**: return Utente, se la tupla non esiste nel DB l’oggetto restituito ha tutti i valori settati a null. |
| context *UtenteDaoDataSource::changePass(Utente utente, conferma\_password)*  pre: utente.email != null and utente.email != “” and utente.password != null and utente.password != “” and conferma\_password != null and conferma\_password != “” and conferma\_password == utente.password and utente.email è formattata correttamente and utente.password è formattata correttamente | **context** *UtenteDaoDataSource::changePass(Utente utente, conferma\_password)*  **post:** return Boolean.true or Boolean.false in base a sé la password è stata correttamente modificata oppure no. Viene aggiornata quindi la tupla corrispondente all’Utente interessato. |
| context *UtenteDaoDataSource::validateOldPass(Utente utente)*  pre: utente.password != null and utente.password != “” | **context** *UtenteDaoDataSource::validateOldPass(Utente utente)*  **post:** return Boolean.true or Boolean.false se la password inserita corrisponde con quella presente all’interno della tupla relativa a quell’Utente. |