Progetto **MYTHOS**

Object Design Document

Partecipanti
Egidio Mario 0512105122
Pagliaro Vincenzo 0512105546

Project Manager
De Lucia Andrea
Pecorelli Fabiano

Sommario

1. Introduzione

1.1. Object Design Trade-offs

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolare, definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.

- Comprensibilità vs Tempo: Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.
- Prestazioni vs Costi: Essendo il progetto sprovvisto di budget, al fine di mantenere prestazioni elevate, per alcune funzionalità verranno utilizzati dei template open source esterni, in particolare Bootstrap.
- Interfaccia vs Usabilità: L'interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa. Fa uso di form e pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale.
- Sicurezza vs Efficienza: Dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

1.2. Linee Guida per la Documentazione delle Interfacce

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

- Organizzazione dei file
  - Ogni file deve essere sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un'unica funzionalità che persegue. Ogni pagina (login, visualizzazione, etc.) deve essere implementata in file separati o divisa in più file, se raggiunge una lunghezza tale da divenire difficile da leggere e comprendere.
  - Organizzare in una cartella i file delle librerie usate e le altre risorse scaricate necessarie per lo sviluppo del progetto.
- Indentazione
  - L'indentazione deve essere effettuata con un TAB e qualunque sia il linguaggio usato per la produzione di codice, ogni istruzione deve essere opportunamente indentata.
- Naming Convention
  - È buona norma utilizzare nomi descrittivi, pronunciabili, di uso comune, lunghezza medio-corta, non abbreviati utilizzando solo caratteri consentiti (a-z, A-Z, 0-9,_).
  - Verrà utilizzata la lingua inglese per dare nomi a variabili, classi, pagine e metodi.
- Variabili
  - I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la maiuscola. È inoltre possibile utilizzare il carattere underscore "_".
- Metodi:
  - I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica un'azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitarne la leggibilità. Esistono però casi particolari come ad esempio nell'implementazione dei model, dove viene utilizzata l'interfaccia CRUD.
  - I commenti dei metodi devono essere in lingua italiana e devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e

deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno e, se applicabile, sulle eccezioni.

- Classi e pagine:
  - I nomi delle classi devono cominciare con una lettera maiuscola e quelli delle pagine con una minuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di queste ultime devono fornire informazioni sul loro scopo. I nomi delle servlet sono analoghi a quelli delle classi.
  - Ogni file sorgente java contiene una singola classe e deve essere strutturato in un determinato modo:
  - Una breve introduzione alla classe (l'autore, la versione e la data).
    /**
    *@author nome dell'autore
    *@version numero di versione della classe
    *@since data dell'implementazione
    */

3. Dichiarazione della classe pubblica, dichiarazioni di costanti, dichiarazioni di variabili di classe, dichiarazioni di variabili d'istanza, costruttore, commento e dichiarazione metodi.

1.3. Definizioni, acronimi e abbreviazioni

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- CRUD: Create Read Update Delete
- DB: DataBase

2. Packages

La gestione del nostro sistema è suddivisa in tre livelli (three-tier):

- Presentation layer
- Application layer
- Storage layer

Il package Mythos contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

Presentation layer
Include tutte le interfacce grafiche e in generale i boundary objects e oggetti relativi alla logica di controllo, come le form con cui interagisce l'utente. L'interfaccia verso l'utente è rappresentata da un Web server e da eventuali contenuti statici (es. pagine HTML).

Application layer
Include tutti gli oggetti utili all'elaborazione dei dati. Questo avviene interrogando il database tramite lo storage layer per generare contenuti dinamici e accedere a dati persistenti.
Si occupa di varie gestioni quali:

- Gestione Utenti
- Gestione Prenotazioni Tavoli
- Gestione Liste Clienti p.r.
- Gestione p.r.
- Gestione Magazzino

- Gestione Ordinazioni

Storage layer

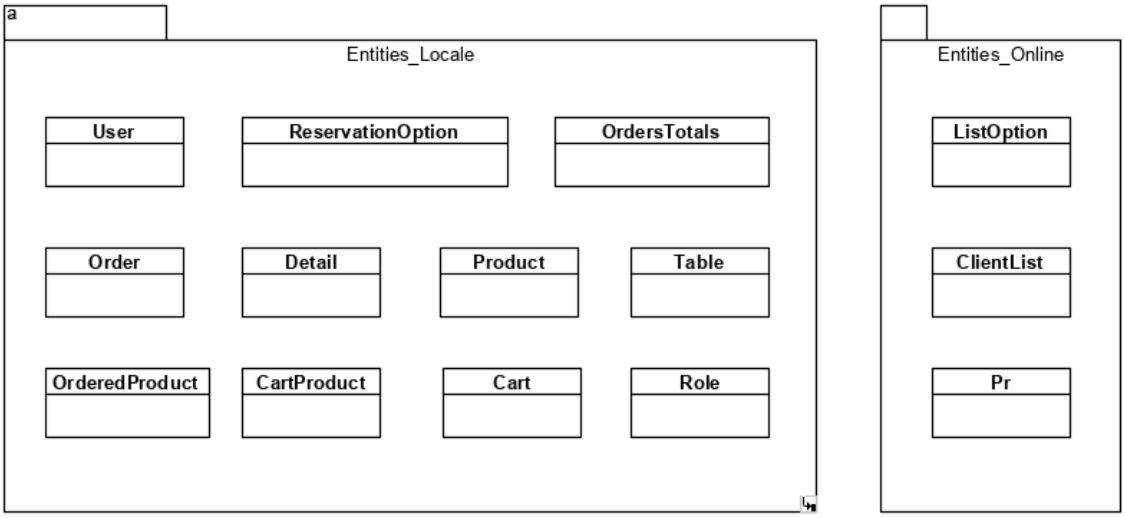
Ha il compito di effettuare memorizzazione, il recupero e l'interrogazione degli oggetti persistenti. I dati, i quali possono essere acceduti dall'application layer, sono depositati in maniera persistente su un database tramite DBMS.

2.1. Packages core

Ogni package verrà suddiviso tra package per la piattaforma locale ("_Locale") e package per la piattaforma online ("_Online").

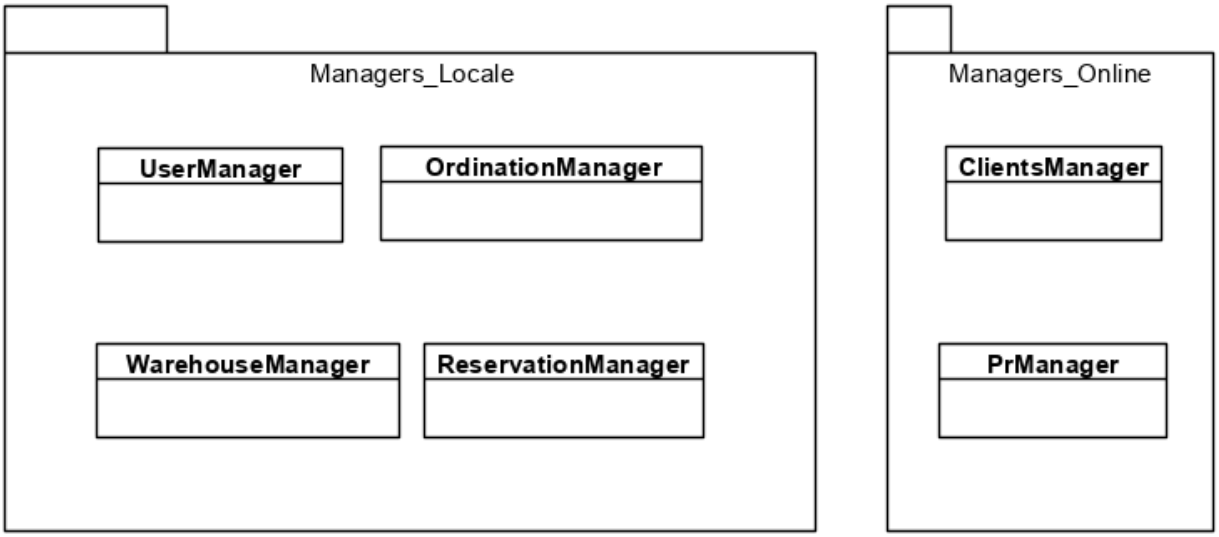
I package verranno illustrati con nomi al plurale ma nell'implementazione il nome dei package sarà al singolare.

2.1.1. Packages Entities



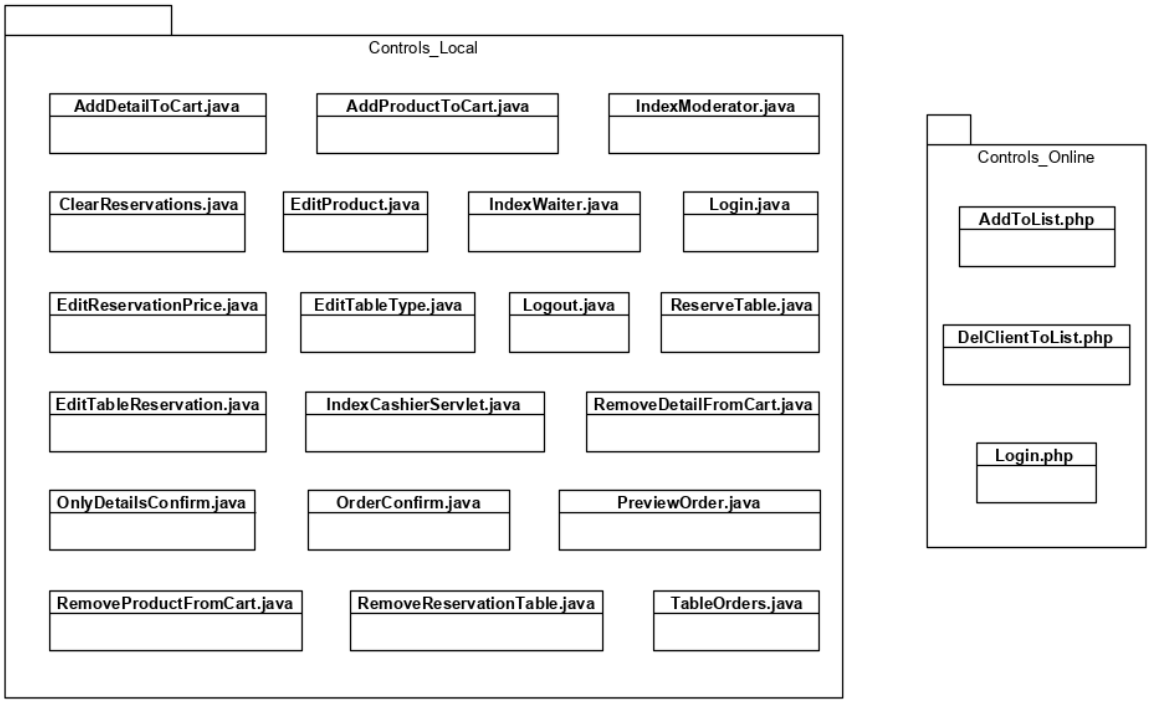
Entità	Descrizione
User.java	Classe per memorizzare i dati riguardanti gli utenti.
ReservationOption.java	Classe per memorizzare i dati riguardanti le i costi e le opzioni di prenotazione.
Role.java	Classe per memorizzare i dati riguardanti il ruolo degli Utenti.
Table.java	Classe per memorizzare i dati riguardanti i tavoli e le prenotazioni.
Order.java	Classe per memorizzare i dati riguardanti gli ordini.
OrderedProduct.java	Classe per memorizzare i dati riguardanti i prodotti ordinati.
Detail.java	Classe per memorizzare i dati riguardanti i dettagli.
OrderTotals.java	Classe per memorizzare i dati riguardanti il rendiconto dei tavoli.
Cart.java	Classe per memorizzare il carrello.
CartProduct.java	Classe per memorizzare i dati riguardanti i prodotti del carrello.
Product.java	Classe per memorizzare i dati riguardanti i prodotti.
ListOption.php	Classe per memorizzare le opzioni della lista clienti.
ClientList.php	Classe per memorizzare i dati dei clienti in lista.
Pr.php	Classe per memorizzare i dati riguardanti i Pr.

2.1.2. Packages Managers



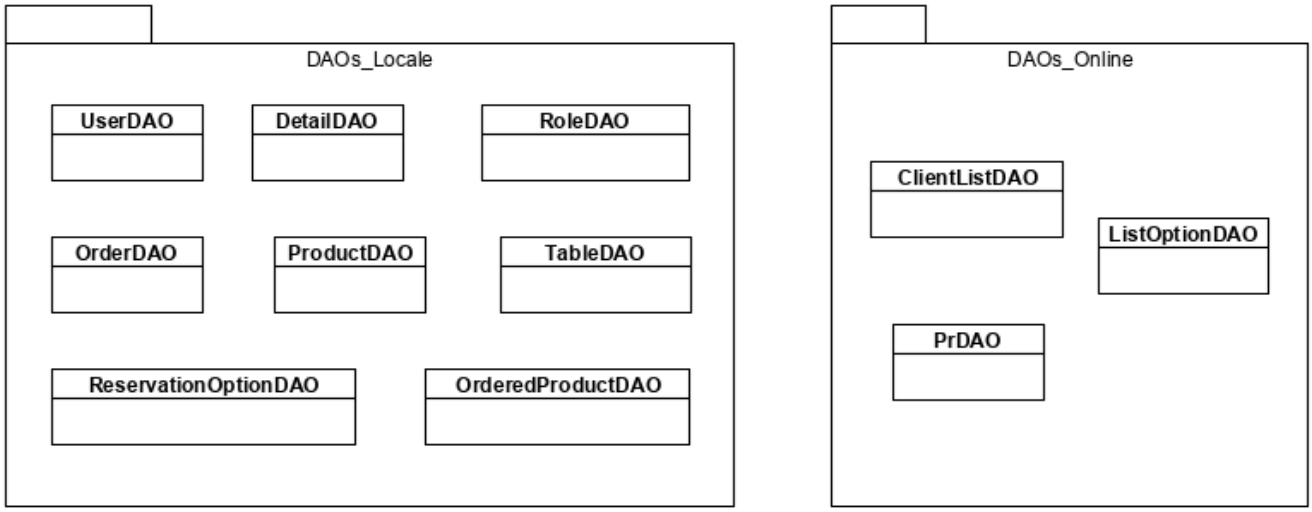
Gestore	Descrizione
UserManager	Classe gestore che offre servizi riguardanti gli utenti.
OrdinationManager	Classe gestore che offre servizi riguardanti le ordinazioni.
WarehouseManager	Classe gestore che offre servizi riguardanti i prodotti in magazzino.
ReservationManager	Classe gestore che offre servizi riguardanti le prenotazioni.
ClientsManager	Pagina gestore che offre servizi riguardanti la lista clienti.
PrManager	Pagina gestore che offre servizi riguardanti i Pr.

2.1.3. Packages Controls



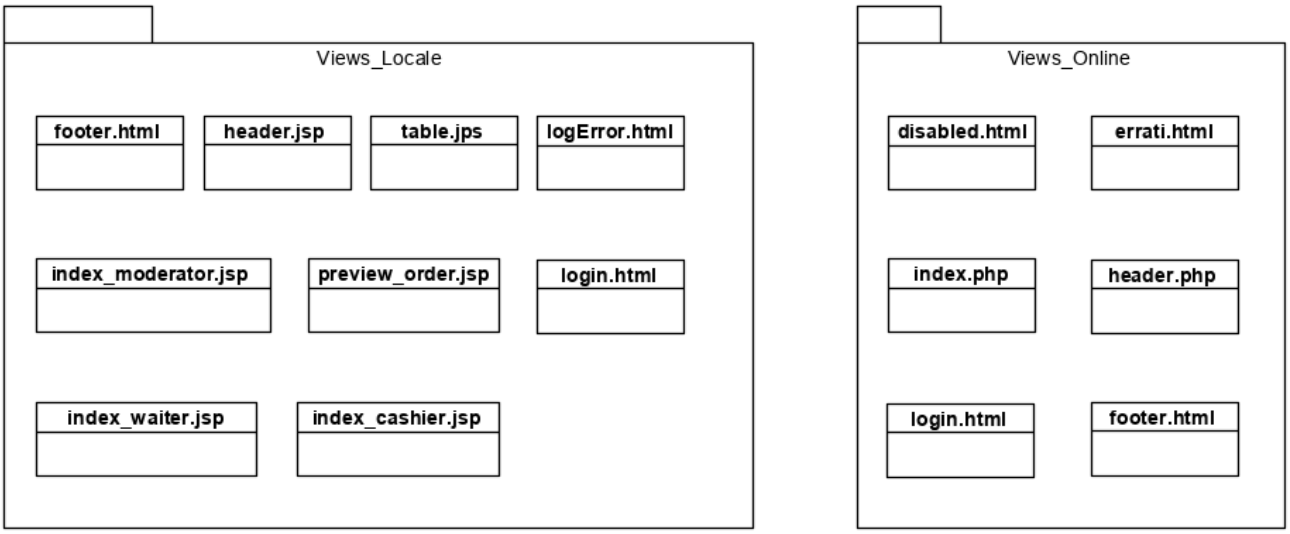
Control	Descrizione
AddDetailToCart.java	Servlet che permette di aggiungere dettagli al carrello.
AddProductToCart.java	Servlet che permette di aggiungere prodotti al carrello.
IndexModerator.java	Servlet che permette di prelevare i dati per la pagina iniziale del moderatore.
ClearReservations.java	Servlet che permette di azzerare tutte le prenotazioni dei tavoli con relativi ordini.
EditProduct.java	Servlet che permette la modifica dei prodotti.
IndexWaiter.java	Servlet che permette di prelevare i dati per la pagina iniziale del cameriere.
Login.java	Servlet che permette di effettuare il login.
EditReservationPrice.java	Servlet che permette la modifica del costo delle prenotazioni.
EditTableType.java	Servlet che permette la modifica del tipo di tavolo.
Logout.java	Servlet che permette di effettuare il logout.
ReserveTable.java	Servlet che permette di effettuare una prenotazione ad un tavolo.
EditTableReservation.java	Servlet che permette la modifica delle prenotazioni.
IndexCashier.java	Servlet che permette di prelevare i dati per la pagina iniziale del cassiere.
RemoveDetailFromCart.java	Servlet che permette di rimuovere dettagli dal carrello.
RemoveProductFromCart.java	Servlet che permette di rimuovere prodotti dal carrello.
OnlyDetailsConfirm.java	Servlet che permette l'invio di ordini con soli dettagli.
OrderConfirm.java	Servlet che permette l'invio degli ordini.
PreviewOrder.java	Servlet che permette di prelevare i dati per la pagina preview_order.jsp.
RemoveReservationTable.java	Servlet che permette di cancellare una prenotazione.
TableOrders.java	Servlet che permette di prelevare il rendiconto dei tavoli.
AddToList.php	Pagina php che permette ad un Pr di aggiungere un cliente in lista.
DelClientToList.php	Pagina php che permette ad un Pr di rimuovere un cliente dalla lista.
Login.php	Pagina php che permette di effettuare il login.

2.1.4. Packages DAOs



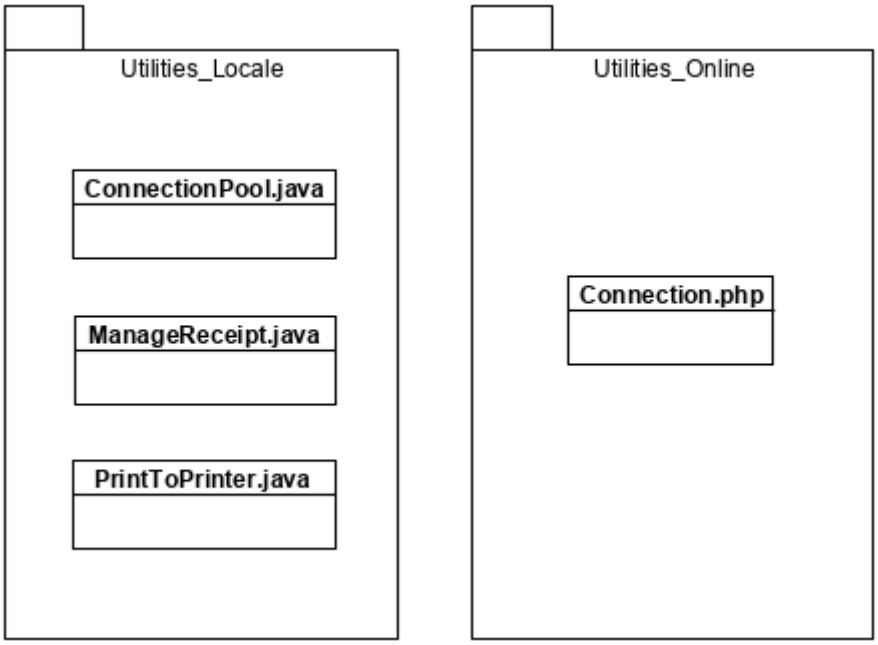
DAO	Descrizione
UserDAO	Classe che si occupa di prelevare i dati degli utenti dallo storage.
DetailDAO	Classe che si occupa di prelevare i dati dei dettagli dallo storage.
RoleDAO	Classe che si occupa di prelevare i dati dei ruoli degli utenti dallo storage.
OrderDAO	Classe che si occupa di prelevare i dati degli ordini dallo storage.
ProductDAO	Classe che si occupa di prelevare i dati dei prodotti dallo storage.
TableDAO	Classe che si occupa di prelevare i dati delle prenotazioni e dei tavoli dallo storage.
ReservationOptionDAO	Classe che si occupa di prelevare i dati i costi e le opzioni di prenotazione dallo storage.
OrderedProductDAO	Classe che si occupa di prelevare i dati dei prodotti ordinati dallo storage.
ClientListDAO	Classe che si occupa di prelevare i dati dei clienti in ista dallo storage.
ListOptionDAO	Classe che si occupa di prelevare i dati delle opzioni della lista dallo storage.
PrDAO	Classe che si occupa di prelevare i dati dei Pr dallo storage.

2.1.5. Packages Views



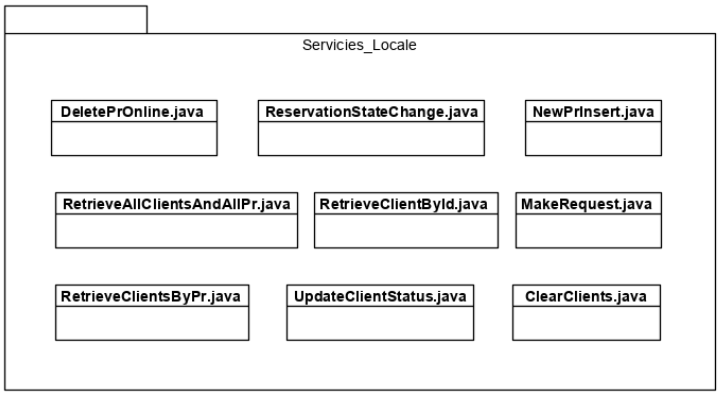
View	Descrizione
header.jsp	Pagina che rappresenta l'header. Verrà inclusa nelle altre pagine della piattaforma in locale.
footer.html	Pagina che rappresenta il footer. Verrà inclusa nelle altre pagine della piattaforma in locale.
table.jsp	Pagina che rappresenta la mappa dei tavoli. Verrà inclusa nelle altre pagine della piattaforma in locale.
logError.html	Pagina di errore della piattaforma in locale.
index_moderator.jsp	Pagina iniziale del moderatore, permette al moderatore di interagire con la piattaforma in locale.
preview_order.jsp	Pagina che permette al cameriere di interagire con la piattaforma in locale per effettuare gli ordini per un determinato tavolo.
index_waiter.jsp	Pagina iniziale del cameriere, permette al cameriere di interagire con la piattaforma in locale.
index_cashier.jsp	Pagina iniziale del cassiere, permette al cassiere di interagire con la piattaforma in locale.
login.html	Pagina iniziale della piattaforma in locale, permette agli utenti di autenticarsi.
disabled.html	Pagina della piattaforma online mostrata quando le liste sono disabilitate.
errati.html	Pagina di errore della piattaforma online.
index.php	Pagina iniziale del Pr, permette al Pr di interagire con la piattaforma online.
login.html	Pagina iniziale della piattaforma in online, permette ai Pr di autenticarsi.
footer.html	Pagina che rappresenta il footer. Verrà inclusa nelle altre pagine della piattaforma online.
header.php	Pagina che rappresenta l'header. Verrà inclusa nelle altre pagine della piattaforma online.

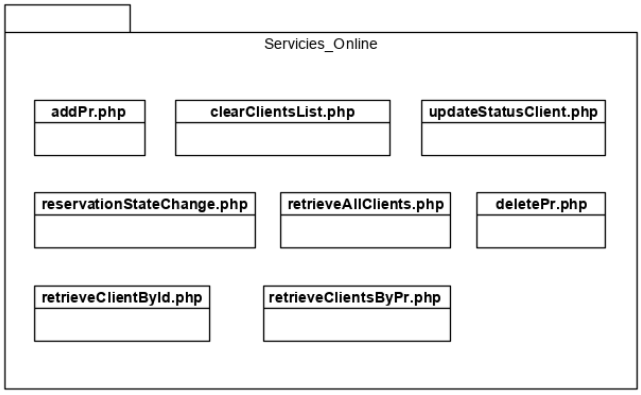
2.1.6. Packages Utilities



Utility	**Descrizione**
ConnectionPool.java	Classe che si occupa di gestire le connessioni al db in locale.
ManageReceipt.java	Classe che si occupa di gestire la creazione del documento da stampare.
PrintToPrinter.java	Classe che si occupa di gestire la stampa.
Connection.php	Pagina che si occupa di gestire le connessioni al database online.

2.1.7. Packages Services





Service	Descrizione
DeletePrOnline.java	Servlet che interagisce con il servizio di rimozione di un Pr offerto dalla piattaforma online.
ReservationStateChange.java	Servlet che interagisce con il servizio di modifica dello stato delle prenotazioni offerto dalla piattaforma online.
NewPrInsert.java	Servlet che interagisce con il servizio di inserimento di un Pr offerto dalla piattaforma online.
RetrieveAllClientsAndAllPr.java	Servlet che interagisce con il servizio offerto dalla piattaforma online che restituisce i Pr e i clienti.
RetrieveClientsById.java	Servlet che interagisce con il servizio offerto dalla piattaforma online che restituisce un cliente conoscendo l'id.
MakeRequest.java	Servlet che si occupa di effettuare le richieste alla piattaforma online.
RetrieveClientsByPr.java	Servlet che interagisce con il servizio offerto dalla piattaforma online che restituisce un cliente conoscendo il Pr.
UpdateClientStatus.java	Servlet che interagisce con il servizio di modifica dello stato di un cliente offerto dalla piattaforma online.
ClearClients.java	Servlet che interagisce con il servizio di azzeramento della lista clienti offerto dalla piattaforma online.
addPr.php	Pagina php che offre il servizio di inserimento di un Pr.
clearClientsList.php	Pagina php che offre il servizio di azzeramento della lista clienti.
updateStatusClient.php	Pagina php che offre il servizio di modifica dello stato di un cliente.
reservationStateChange.php	Pagina php che offre il servizio di modifica dello stato delle prenotazioni.
retrieveAllClients.php	Pagina php che offre il servizio che restituisce i Pr e i clienti.
deletePr.php	Pagina php che offre il servizio di rimozione di un Pr.
retrieveClientById.php	Pagina php che offre il servizio che restituisce un cliente conoscendo l'id.
retrieveClientsByPr.php	Pagina php che offre il servizio che restituisce un cliente conoscendo il Pr.

3. Interfaccia delle classi

Interfacce delle classi della piattaforma in locale

3.1. Package DAO

DetailDAO.java

```
package DAO;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import entity.Detail;
import utility.ConnectionPool;
/**
 * Classe che si occupa di prelevare i dati dei dettagli dallo storage.
 */
public class DetailDAO {
	/**
	 * Metodo che restituisce l'elenco dei dettagli.
	 * @return {@link ArrayList} elenco dettagli
	 */
	public ArrayList<Detail> doRetrieveAll() ;

	/**
	 * Metodo che restituisce un dettaglio conoscendone l'id.
	 * @param id {@link Integer} indica l'id del dettaglio.
	 * @return {@link Detail}
	 */
	public Detail doRetrieveById(int id) ;

}
```

OrderDAO.java

```
package DAO;
import java.sql.*;
import java.util.ArrayList;
import entity.Order;
import entity.User;
import utility.ConnectionPool;
/**
 * Classe che si occupa di prelevare i dati degli ordini dallo storage.
 */
public class OrderDAO {
	/**
	 * Metodo che si occupa di memorizzare un ordine conoscendo utente e id del
tavolo.
	 * @param u {@link User} u!=null indica l'utente(cameriere) che effettua l'ordine.
	 * @param idTable {@link Integer} idTable!=null && idTable>0 indica il tavolo per
cui fare l'ordine.
	 * @return {@link Order} ordine memorizzato.
	 */
	public synchronized Order makeNewOrderByUserAndTableId(User u, int idTable) ;
```

```
	/**
	 * Metodo che restituisce l'ultimo ordine di un cameriere.
	 * @param u {@link User} u!=null indica il cameriere.
	 * @return {@link Integer} numero ordine.
	 */
	public int doRetrieveMaxIdByUser(User u) ;

	/**
	 * Metodo usato per aggiornare dei valori di un ordine già memorizzato.
	 * @param o {@link Order} o!=null indica i valori del nuovo ordine.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doUpdate(Order o) ;

	/**
	 * Metodo che restituisce un'ordine dal suo id.
	 * @param id {@link Integer} id!=null && id>0 indica l'id dell'ordine.
	 * @return {@link Order} ordine cercato.
	 */
	public Order doRetrieveById(int id) ;

	/**
	 * Metodo che restituisce tutti gli ordini effettuati ad un tavolo.
	 * @param tn {@link Integer} tn!=null && tn>0 indica il numero del tavolo
	 * @return {@link ArrayList} elenco ordini del tavolo richiesto.
	 */
	public ArrayList<Order> doRetrieveByTableNumber(int tn) ;

	/**
	 * Metotdo per azzerare le tabella orders.
	 */
	public void doClearTable() ;

	/**
	 * Metodo che restituisce i pagamenti extra degli ordini confermati per un tavolo.
	 * @param table_id {@link Integer} table_id!=null && table_id>0 indica il numero
	 *        del tavolo.
	 * @return {@link Float} totale pagamento extra ricavato dalla somma di tutti gli
	 *         extraPagamenti degli ordini confermati al tavolo passato come parametro.
	 */   public float doRetrieveExtraPaymentByTable(int table_id);

	/**
	 * Metodo per eliminare un ordine dallo storage.
	 * @param o {@link Order} o!= null indica l'ordine da eliminare.
	 */
	public void doDeleteByOrder(Order o) ;
}
```

OrderedProductDAO.java

```
package DAO;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import entity.OrderedProduct;
import entity.OrdersTotals;
import utility.ConnectionPool;
```

```
/**
 * Classe che si occupa di prelevare i dati dei prodotti ordinati dallo storage.
 */
public class OrderedProductDAO {

	/**
	 * Metodo che permette di memorizzare un prodotto ordinato.
	 * @param op {@link OrderedProduct} op!=null indica il prodotto da memorizzare.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doSave(OrderedProduct op);

	/**
	 * Metodo che restituisce tutti i prodotti ordinati.
	 * @return {@link ArrayList} elenco prodotti ordinati.
	 */
	public ArrayList<OrderedProduct> doRetrieveAll();

	/**
	 * Metodo che restituisce tutti i prodotti ordinati di un determinato ordine.
	 * @param order {@link Order} order!=null indica l'ordine.
	 * @return {@link ArrayList} elenco prodotti ordinati.
	 */
	public ArrayList<OrderedProduct> doRetrieveByOrder(int order) ;

	/**
	 * Metodo per azzerare la tabella ordered_products.
	 */
	public void doClearTable() ;

	/**
	 * Metodo che restituisce i totali per tavolo.
	 * @return {@link ArrayList} elenco tavoli con relativi totali.
	 */
	public ArrayList<OrdersTotals> doRetrieveTableTot() ;


}
```

ProductDAO.java

```
package DAO;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import entity.Product;
import utility.ConnectionPool;

/**
 * Classe che si occupa di prelevare i dati dei prodotti dallo storage.
 */
public class ProductDAO {

	/**
	 * Metodo che restituisce l'elenco di tutti i prodotti.
	 * @return {@link ArrayList} elenco prodotti.
	 */
	public ArrayList<Product> doRetrieveAll() ;
```

```
	/**
	 * Metodo che restituisce l'elenco di tutti i prodotti in cambusa.
	 * @return {@link ArrayList} elenco prodotti in cambusa.
	 */
	public ArrayList<Product> doRetrieveGalleyAll();

	/**
	 * Metodo che restituisce l'elenco un prodotto conoscendo l'id.
	 * @param id {@link Integer} id!=null && id>0 indica l'id del prodotto.
	 * @return {@link Product} prodotto cercato.
	 */
	public Product doRetrieveById(int id) ;

	/**
	 * Metodo usato per aggiornare dei vaolri di un prodotto già memorizzato.
	 * @param p {@link Product} p!=null indica il prodotto.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doUpdate(Product p) ;

	/**
	 * Metodo per verificare se la quantità richiesta di un prodotto è disponibile.
	 * @param prod {@link Product} prod!=null indica il prodotto.
	 * @param requestedQta {@link Integer} requestedQta!=null && requestedQta>0 indica
la qta richiesta.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doCheckAvailabilityProduct(Product prod, int requestedQta) ;



	/**
	 * Metodo che restituisce l'elenco di tutti i prodotti nel bar.
	 * @return {@link ArrayList} elenco prodotti nel bar.
	 */
	public ArrayList<Product> doRetrieveBarAll() ;

	/**
	 * Metodo per ripristinare le quantità in cambusa dal carrello.
	 * @param prod {@link Product} prod!=null indica il prodotto.
	 * @param qtaToAdd {@link Integer} qtaToAdd!=null && qtaToAdd>0 indica la quantità
da ripristinare.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doRestoreQuantity(Product prod, int qtaToAdd) ;

	/**
	 * Metodo per bloccare la tabella products.
	 */
	public static void lockTable() ;

	/**
	 * Metodo per sbloccare la tabella products.
	 */
	public static void unlockTable();

}
```

ReservationOptionDAO.java

```
package DAO;
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import entity.ReservationOption;
import utility.ConnectionPool;
/**
 * Classe che si occupa di prelevare i dati i costi e le opzioni di prenotazione dallo
storage.
 */
public class ReservationOptionDAO {
	/**
	 * Metodo che restituisce le opzioni della prenoazione.
	 * @return {@link ReservationOption} opzioni prenotazione.
	 */
	public ReservationOption doRetrieveConfig() ;

	/**
	 * Metodo per aggiornare le opzioni di prenotazione.
	 * @param ro {@link ReservationOption} ro!=null nuovi valori per le opzioni di
prenotazione.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doUpdate(ReservationOption ro);
}
```

RoleDAO.java

```
package DAO;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import entity.Role;
import utility.ConnectionPool;

/**
 * Classe che si occupa di prelevare i dati dei ruoli degli utenti dallo storage.
 */
public class RoleDAO {
	/**
	 * Metodo che restisuisce il ruolo dall'id.
	 * @param idRole {@link Integer} idRole!=null && idRole>0 indica l'id del ruolo.
	 * @return {@link Role} ruolo.
	 */
	public Role retrieveRoleByID(int idRole);
}
```

TableDAO.java

```
package DAO;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import entity.Table;
import utility.ConnectionPool;
/**
```

```
 * Classe che si occupa di prelevare i dati delle prenotazioni e dei tavoli dallo
storage.
 */
public class TableDAO {

	/**
	 * Metodo che restituisce l'elenco dei tavoli.
	 * @return {@link ArrayList} elenco tavoli.
	 */
	public ArrayList<Table> doRetrieveAll() ;

	/**
	 * Metodo che restituisce l'elenco dei tavoli disponibili.
	 * @return {@link ArrayList} elenco tavoli disponibili.
	 */
	public ArrayList<Table> doRetrieveNotAssigned();

	/**
	 * Metodo che restituisce i dati del tavolo sapendo il numero del tavolo.
	 * @param idTable {@link Integer} idTable!=null && idTable>0.
	 * @return {@link Table} tavolo.
	 */
	public Table doRetrieveById(int idTable);
	/**
	 * Metodo che restituisce l'elenco dei tavoli non disponibili.
	 * @return {@link ArrayList} elenco tavoli non disponibili.
	 */
	public ArrayList<Table> doRetrieveAssigned() ;

	/**
	 * Metodo usato per aggiornare dei valori di un tavolo già memorizzato.
	 * @param t {@link Table} t!=null indica i nuovi valori del tavolo.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doUpdate(Table t) ;

	/**
	 * Metodo che setta un tavolo come "disponibile"
	 * @param table {@link Table} table!=null indica il tavolo.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doSetNotAssignedById(int table) ;

	/**
	 * Metodo per modificare una prenotazione
	 * @param id_table {@link Integer} id_table!=null && id_table>0 indica il numero
del tavolo.
	 * @param name {@link String} name!=null indica il nome della prenotazione.
	 * @param people {@link Integer} people!=null && people>0 indica il numero di
persone.
	 * @param budget {@link Float} budget!=null && budget>0 indica il budget.
	 * @return
	 */
	public boolean doEditReservation(int id_table,String name,int people, float
budget) ;

	/**
	 * Metodo per azzerare i campi della tabella tables.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean doClear() ;
}
```

UserDAO.java

```
package DAO;
import java.sql.*;
import entity.User;
import utility.ConnectionPool;

/**
 * Classe che si occupa di prelevare i dati degli utenti dallo storage.
 */

public class UserDAO {


	/**
	 * Metodo per verificare le credenziali di un utente.
	 * @param username {@link String} username!=null indica l'username.
	 * @param password {@link String} password!=null indica la password.
	 * @return {@link User} utente.
	 */
	public User doRetrieveByUsernamePassword(String username, String password) ;

	/**
	 * Metodo che restituisce un utente dall'username.
	 * @param username {@link String} username!=null indica l'username.
	 * @return User {@link User} utente.
	 */
	public User doRetrieveByUsername(String username);

}
```

3.2. Package entity

Cart.java

```
package entity;
import java.util.HashMap;
/**
 * Classe per memorizzare il carrello.
 */
public class Cart {

	/**
	 * Costruttore.
	 */
	public Cart() ;

	/**
	 * Metodo per aggiungere un prodotto al carrello.
	 * @param cart_product {@link CartProduct} cart_product!=null indica il prodotto
	   da aggiungere.
	 * @return {@link Boolean} esito operazione.
	 */
	public boolean addProductToCart(CartProduct cart_product) ;

	/**
	 * Metodo per rimuovere un prodotto dal carrello.
	 * @param productId {@link Integer} productId!=null && productId>0 indica l'id del
	   prodotto.
```

```
     * @return {@link Boolean} esito operazione.
     */
    public boolean removeProductFromCart(int productId) ;

    /**
     * Metodo per ottenere il numero di articoli nel carrello.
     * @return {@link Integer} numero articoli.
     */
    public int getProductCount() ;
    /**
     * Metodo per ottenere il numero di dettagli nel carrello.
     * @return {@link Integer} numero dettagli.
     */
    public int getDetailCount() ;

    /**
     * Metodo per verificare se un dettaglio è presente nel carrello.
     * @param d {@link Detail} d!=null indica il dettaglio da cercare.
     * @return {@link Boolean} vero=trovato, altrimenti false.
     */
    public boolean containsDetail(Detail d) ;

    /**
     * Metodo per aggiungere un dettaglio al carrello.
     * @param detail {@link Detail} detail!=null indica il dettaglio.
     * @return {@link Boolean} esito operazione.
     */
    public boolean addDetailToCart(Detail detail) ;


    /**
     * Metodo per rimuovere un dettaglio dal carrello.
     * @param id {@link Integer} id!=null && id>0 indica l'id del dettaglio.
     * @return {@link Boolean} esito operazione.
     */
    public boolean removeDetailFromCart(int id);

    /**
     * Metodo per restituire la lista prodotti nel carrelo.
     * @return {@link HashMap} lista prodotti.
     */
    public HashMap<Integer, CartProduct> getProductList() ;

    /**
     * Metodo per restituire la lista dettagli nel carrelo.
     * @return {@link HashMap} lista dettagli.
     */
    public HashMap<Integer, Detail> getDetailsList() ;

    /**
     * lista prodotti.
     */
    private HashMap<Integer, CartProduct> productList;
    /**
     * lista dettagli.
     */
    private HashMap<Integer, Detail> detailsList;

}
```

CartProduct.java

```
package entity;

import java.io.Serializable;

public class CartProduct implements Serializable {

	/**
	 * Classe per memorizzare i dati riguardanti i prodotti del carrello.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * Metodo che restituisce il prodotto.
	 * @return {@link Product} prodotto nel carrello.
	 */
	public Product getProduct() ;

	/**
	 * Metodo per aggiornare il prodotto.
	 * @param product {@link Product} indica il nuovo prodotto.
	 */
	public void setProduct(Product product) ;

	/**
	 * Metodo che restituisce la quantità ordinata.
	 * @return {@link Integer} qta ordinata.
	 */
	public int getQuantity() ;

	/**
	 * Metodo per settare la quantità ordinata
	 * @param quantity {@link Integer} indica la nuova quantità.
	 */
	public void setQuantity(int quantity) ;

	/**
	 * Prodotto.
	 */
	private Product;
	/**
	 * qta.
	 */
	private int quantity;
}
```

Detail.java

```
package entity;
import java.io.Serializable;

public class Detail implements Serializable {

	/**
	 * Classe per memorizzare i dati riguardanti i dettagli.
	 */
	private static final long serialVersionUID = 1L;
	/**
	 * Metodo che restituisce l'id del dettaglio.
	 * @return {@link Integer} id dettaglio.
	 */
	public int getIdDetail() ;
	/**
	 * Metodo per settare l'id del dettaglio.
```

```
	 * @param idDetail {@link Integer} indica il nuovo id.
	 */
	public void setIdDetail(int idDetail) ;

	/**
	 * Metodo che restituisce la descrizione del dettaglio.
	 * @return {@link String} descrizione del dettaglio.
	 */
	public String getDescription() ;

	/**
	 * Metodo che setta la nuova descrizione del dettaglio.
	 * @param description {@link String} indica la nuova descrizione.
	 */
	public void setDescription(String description) ;

	/**
	 * Metodo to string.
	 */
	public String toString();

	/**
	 * id dettaglio.
	 */
	private int idDetail;
	/**
	 * descrizione dettaglio.
	 */
	private String description;
}
```

Order.java

```
package entity;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * Classe per memorizzare i dati riguardanti gli ordini.
 */
public class Order implements Serializable {

	private static final long serialVersionUID = 1L;

	/**
	 * Metodo che restituisce l'id dell'ordine.
	 * @return {@link Integer} id ordine.
	 */
	public int getIdOrder() ;

	/**
	 * Metodo per settare l'id del dettaglio.
	 * @param idOrder {@link Integer} indica il nuovo id dell'ordine.
	 */
	public void setIdOrder(int idOrder) ;
	/**
	 * Metodo che restituisce la data dell'ordine.
	 * @return {@link String} data ordine.
	 */
	public String getDateOrder() ;
	/**
	 * Metodo per settare la data dell'ordine.
```

```
 * @param dateOrder {@link String} data ordine.
 */
public void setDateOrder(String dateOrder) ;

/**
 * Metodo che restituisce lo stato dell'ordine.
 * @return {@link Integer} stato ordine.
 */
public int getStatus() ;

/**
 * Metodo per settare lo stato dell'ordine.
 * @param status {@link Integer} indica lo stato dell'ordine.
 */
public void setStatus(int status) ;

/**
 * Metodo che restituisce l'elenco dei dettagli.
 * @return {@link String} elenco dettagli.
 */
public String getDetails() ;

/**
 * Metodo per settare l'elenco dei dettagli.
 * @param details {@link String} elenco dettagli.
 */
public void setDetails(String details) ;

/**
 * Metodo che restituisce l'utente che ha fatto l'ordine.
 * @return {@link User} utente che ha fatto l'ordine
 */
public User getUser() ;

/**
 * Metodo per settare l'utente che ha fatto l'ordine.
 * @param user {@link User} utente che ha fatto l'ordine
 */
public void setUser(User user) ;

/**
 * Metodo che restituisce il tavolo a cui appartiene l'ordine.
 * @return {@link Table} tavolo a cui appartiene l'ordine
 */
public Table getTable() ;

/**
 * Metodo per settare il tavolo a cui appartiene l'ordine.
 * @param table {@link Table} tavolo a cui appartiene l'ordine
 */
public void setTable(Table table) ;

/**
 * Metodo per restiruire l'elenco dei prodotti ordinati.
 * @return {@link ArryList} elenco dei prodotti ordinati.
 */
public ArrayList<OrderedProduct> getOrderDetails() ;




/**
 * Metodo per settare l'elenco dei prodotti ordinati.
```

```
	 * @param orderDetails {@link ArryList} elenco dei prodotti ordinati.
	 */
	public void setOrderDetails(ArrayList<OrderedProduct> orderDetails) ;

	/**
	 * Metodo che restituisce il metodo di pagamento.
	 * @return {@link String} metodo di pagamento.
	 */
	public String getMethodPay() ;

	/**
	 * Metodo per settare il metodo di pagamento.
	 * @param methodPay {@link String} metodo di pagamento.
	 */
	public void setMethodPay(String methodPay) ;

	/**
	 * Metodo che restituisce il totale di pagamento extra.
	 * @return {@link Float} totale di pagamento extra.
	 */
	public float getExtraPayments() ;

	/**
	 * Metodo per settare il totale di pagamento extra.
	 * @param extraPayments {@link Float} totale di pagamento extra.
	 */
	public void setExtraPayments(float extraPayments) ;

	/**
	 * Metodo to string.
	 */
	@Override
	public String toString() ;


	private int idOrder;
	private String dateOrder;
	private int status;
	private String details = "";
	private User;
	private Table;
	private ArrayList<OrderedProduct> orderDetails;
	private String methodPay;
	private float extraPayments;
}
```

OrderedPrdouct.java

```
package entity;

import java.io.Serializable;

/**
 * Classe per memorizzare i dati riguardanti i prodotti ordinati.
 */
public class OrderedProduct implements Serializable {


	private static final long serialVersionUID = 1L;
	/**
	 * Metodo per restituire la quantità ordinata.
	 * @return {@link Integer} quantità ordinata.
```

```
 */
public int getQuantity() ;

/**
 * Metodo per settare la quantità ordinata.
 * @param quantity {@link Integer} quantità ordinata.
 */
public void setQuantity(int quantity) ;

/**
 * Metodo per restituire il prezzo d'acquisto.
 * @return {@link Float} prezzo d'acquisto.
 */
public float getPurchaseUnitPrice() ;

/**
 * Metodo per settare il prezzo d'acquisto.
 * @param purchaseUnitPrice {@link Float} prezzo d'acquisto.
 */
public void setPurchaseUnitPrice(float purchaseUnitPrice) ;

/**
 * Metodo per restituire l'ordine.
 * @return {@link Order} ordine.
 */
public Order getOrder() ;

/**
 * Metodo per settare l'ordine.
 * @param order {@link Order} ordine.
 */
public void setOrder(Order order) ;

/**
 * Metodo che restituire il prodotto.
 * @return {@link Product} prodotto.
 */
public Product getProduct() ;

/**
 * Metodo per settare il prodotto.
 * @param product {@link Product} prodotto.
 */
public void setProduct(Product product) ;

/**
 * Metodo to string.
 */
@Override
public String toString() ;

/**
 * quantità.
 */
private int quantity;
/**
 * prezzo d'acquisto.
 */
private float purchaseUnitPrice;
/**
 * ordine.
 */
private Order;

/**
 * prodotto.
 */
```

```
	private Product;
}
```

OrderTotals.java

```
package entity;

/**
 * Classe per memorizzare i dati riguardanti il rendiconto dei tavoli.
 */
public class OrdersTotals {

	/**
	 * Metodo per restituire il tavolo.
	 * @return {@link Table} tavolo.
	 */
	public Table getTab() ;

	/**
	 * Metodo per settare il nuovo tavolo.
	 * @param tab {@link Table} tavolo.
	 */
	public void setTab(Table tab) ;

	/**
	 * Metodo per restituire il totale del tavolo.
	 * @return {@link Float} totale del tavolo.
	 */
	public float getTotal() ;

	/**
	 * Metodo per settare il totale del tavolo.
	 * @param total {@link Float} totale del tavolo.
	 */
	public void setTotal(float total) ;

	/**
	 * Metodo per restituire il totale extra del tavolo.
	 * @return {@link Float} totale extra del tavolo.
	 */
	public float getExtraPayments() ;

	/**
	 * Metodo per settare il totale extra del tavolo.
	 * @param extraPayments {@link Float} totale extra del tavolo.
	 */
	public void setExtraPayments(float extraPayments) ;

	/**
	 * tavolo.
	 */
	private Table tab;
	/**
	 * pagamento extra.
	 */
	private float extraPayments;
	/**
	 * totale tavolo.
	 */
	private float total;

}
```

Cart.java

```java
package entity;

/**
 * Classe per memorizzare i dati riguardanti i prodotti.
 */
public class Product {

	/**
	 * Metodo per restituire l'id del prodotto.
	 * @return {@link Integer} id prodotto.
	 */
	public int getIdProduct() ;

	/**
	 * Metodo per settare l'id del prodotto.
	 * @param idProduct {@link Integer} id prodotto.
	 */
	public void setIdProduct(int idProduct);

	/**
	 * Metodo per restituire il nome del prodotto.
	 * @return {@link String} nome del prodotto.
	 */
	public String getName() ;

	/**
	 * Metodo per settare il nome del prodotto.
	 * @param name {@link String} nome del prodotto.
	 */
	public void setName(String name) ;

	/**
	 * Metodo per restituire la descrizione del prodotto.
	 * @return {@link String} descrizione del prodotto.
	 */
	public String getDescription() ;

	/**
	 * Metodo per settare la descrizione del prodotto.
	 * @param description {@link String} descrizione del prodotto.
	 */
	public void setDescription(String description) ;

	/**
	 * Metodo per restituire il prezzo del prodotto.
	 * @return {@link Float} prezzo del prodotto.
	 */
	public float getPrice() ;

	/**
	 * Metodo per settare il prezzo del prodotto.
	 * @param price {@link Float} prezzo del prodotto.
	 */
	public void setPrice(float price);

	/**
	 * Metodo per restituire lo stato di eliminazione del prodotto.
	 * @return {@link Boolean} stato di eliminazione del prodotto.
	 */
	public boolean isDeleted() ;
	/**
	 * Metodo per settare lo stato di eliminazione del prodotto.
	 * @param deleted {@link Boolean} stato di eliminazione del prodotto.
	 */
	public void setDeleted(boolean deleted) ;
	/**
	 * Metodo per restituire la qta in magazzino del prodotto.
```

```
     * @return {@link Integer} qta in magazzino del prodotto.
     */
    public int getQuantityWarehouse() ;

    /**
     * Metodo per settare la qta in magazzino del prodotto.
     * @param quantityWarehouse {@link Integer} qta in magazzino del prodotto.
     */
    public void setQuantityWarehouse(int quantityWarehouse) ;

    /**
     * Metodo per restituire la qta in cambusa del prodotto.
     * @return {@link Integer} qta in cambusa del prodotto.
     */
    public int getQuantityGalley() ;

    /**
     * Metodo per settare la qta in cambusa del prodotto.
     * @param quantityGalley {@link Integer} qta in cambusa del prodotto.
     */
    public void setQuantityGalley(int quantityGalley) ;

    /**
     * Metodo per restituire la qta in bar del prodotto.
     * @return {@link Integer} qta in bar del prodotto.
     */
    public int getQuantityBar() ;

    /**
     * Metodo per settare la qta in bar del prodotto.
     * @param quantityBar {@link Integer} qta in bar del prodotto.
     */
    public void setQuantityBar(int quantityBar) ;

    /**
     * Metodo per restituire il tipo del prodotto.
     * @return {@link Integer} tipo del prodotto.
     */
    public int getFlagType() ;

    /**
     * Metodo per settare il tipo del prodotto.
     * @param flagType {@link Integer} tipo del prodotto.
     */
    public void setFlagType(int flagType) ;

    /**
     * Override di equals.
     * @param obj
     * @return {@link Boolean}
     */
    public boolean equals(Object obj) ;

    /**
     * id prodotto.
     */
    private int idProduct;
    /**
     * nome prodotto.
     */
    private String name;
    /**
     * descrizione prodotto.
     */
    private String description;
    /**
     * prezzo prodotto.
```

```
	 */
	private float price;
	/**
	 * quantita in magazzino.
	 */
	private int quantityWarehouse;
	/**
	 * stato eliminazione prodotto.
	 */
	private boolean deleted;
	/**
	 * qta in cambusa.
	 */
	private int quantityGalley;
	/**
	 * qta in bar.
	 */
	private int quantityBar;
	/**
	 * flag tipo.
	 */
	private int flagType;
}
```

ReservatioOption.java

```
package entity;
/**
 * Classe per memorizzare i dati riguardanti le i costi e le opzioni di prenotazione.
 */
public class ReservationOption {
	/**
	 * Metodo per restituire il prezzo della prenotaione ai tavoli.
	 * @return {@link Float} prezzo prenotazione.
	 */
	public float getTablePrice() ;

	/**
	 * Metodo per settare il prezzo della prenotaione ai tavoli.
	 * @param tablePrice {@link Float} prezzo prenotazione.
	 */
	public void setTablePrice(float tablePrice) ;

	/**
	 * Metodo per restituire il prezzo della prenotaione luxuus ai tavoli.
	 * @return {@link Float} prezzo prenotazione.
	 */
	public float getLuxusTablePrice() ;

	/**
	 * Metodo per settare il prezzo della prenotaione luxus ai tavoli.
	 * @param luxusTablePrice {@link Float} prezzo prenotazione.
	 */
	public void setLuxusTablePrice(float luxusTablePrice) ;

	/**
	 * prezzo prenotazione normale.
	 */
	private float tablePrice;
	/**
	 * prezzo prenotazione luxus.
	 */
	private float luxusTablePrice;
}
```

Role.java

```
package entity;

import java.io.Serializable;
/**
 * Classe per memorizzare i dati riguardanti il ruolo degli Utenti.
 */
public class Role implements Serializable{

	private static final long serialVersionUID = 1L;

	/**
	 * Metodo per restituire l'id del ruolo.
	 * @return {@link Integer} id ruolo.
	 */
	public int getIdRole() ;

	/**
	 * Metodo per settare l'id del ruolo.
	 * @param idRole {@link Integer} id ruolo.
	 */
	public void setIdRole(int idRole) ;

	/**
	 * Metodo per restituire il nome del ruolo.
	 * @return {@link String} nome ruolo.
	 */
	public String getRoleName() ;

	/**
	 * Metodo per settare il nome del ruolo.
	 * @param roleName {@link String} nome ruolo.
	 */
	public void setRoleName(String roleName) ;

	/**
	 * id ruolo.
	 */
	private int idRole;
	/**
	 * nome ruolo.
	 */
	private String roleName;
}
```

Table.java

```
package entity;

import java.io.Serializable;
/**
 * Classe per memorizzare i dati riguardanti i tavoli e le prenotazioni.
 */
public class Table implements Serializable{

	private static final long serialVersionUID = 1L;

	/**
	 * Metodo per restituire il numero del tavolo.
	 * @return {@link Integer} numero tavolo.
	 */
	public int getTableNumber() ;


	/**
```

```
     * Metodo per settare il numero del tavolo.
     * @param tableNumber {@link Integer} numero tavolo.
     */
    public void setTableNumber(int tableNumber) ;

    /**
     * Metodo per restituire il nome del tavolo.
     * @return {@link String} nome tavolo.
     */
    public String getTableName() ;

    /**
     * Metodo per settare il nome del tavolo.
     * @param tableName {@link String} nome tavolo.
     */
    public void setTableName(String tableName) ;

    /**
     * Metodo per restituire la capacità del tavolo.
     * @return {@link Integer} capacità tavolo.
     */
    public int getCapacity() ;

    /**
     * Metodo per settare la capacità del tavolo.
     * @param capacity {@link Integer} capacità tavolo.
     */
    public void setCapacity(int capacity) ;

    /**
     * Metodo per restituire lo stato del tavolo (assegnato o no).
     * @return {@link Boolean} stato tavolo.
     */
    public boolean isAssigned() ;

    /**
     * Metodo per settare lo stato del tavolo (assegnato o no).
     * @param isAssigned {@link Boolean} stato tavolo.
     */
    public void setAssigned(boolean isAssigned) ;

    /**
     * Metodo per restituire il nome della prenotazione al tavolo.
     * @return {@link String} nome prenotazione.
     */
    public String getReservationName() ;

    /**
     * Metodo per settare il nome della prenotazione al tavolo.
     * @param reservationName {@link String} nome prenotazione.
     */
    public void setReservationName(String reservationName) ;

    /**
     * Metodo per restituire il numero di persone della prenotazione al tavolo.
     * @return {@link Integer} numero persone della prenotazione.
     */
    public int getPeopleNumber() ;

    /**
     * Metodo per settare il numero di persone della prenotazione al tavolo.
     * @param peopleNumber {@link Integer} numero persone della prenotazione.
     */
    public void setPeopleNumber(int peopleNumber) ;


    /**
```

```
     * Metodo per restituire il budget della prenotazione al tavolo.
     * @return {@link Float} budget della prenotazione.
     */
    public float getBudget() ;

    /**
     * Metodo per settare il budget della prenotazione al tavolo.
     * @param budget {@link Float} budget della prenotazione.
     */
    public void setBudget(float budget) ;

    /**
     * Metodo per restituire il tipo di tavolo.
     * @return {@link Boolean} tipo di tavolo.
     */
    public boolean isLuxus() ;

    /**
     * Metodo per settare il tipo di tavolo.
     * @param isLuxus {@link Boolean} tipo di tavolo.
     */
    public void setLuxus(boolean isLuxus);

    /**
     * Metodo to string.
     */
    public String toString() ;

    /**
     * numero del tavolo.
     */
    private int tableNumber;
    /**
     * nome del tavolo.
     */
    private String tableName;
    /**
     * capacità tavolo.
     */
    private int capacity;
    /**
     * stato tavolo.
     */
    private boolean isAssigned;
    /**
     * nome della prenotazione.
     */
    private String reservationName;
    /**
     * numero persone della prenotazione.
     */
    private int peopleNumber;
    /**
     * budget tavolo.
     */
    private float budget;
    /**
     * tipo prenotazione.
     */
    private boolean isLuxus;
}
```

User.java

```java
package entity;

import java.io.Serializable;

/**
 * Classe per memorizzare i dati riguardanti gli utenti.
 */
public class User implements Serializable{

	private static final long serialVersionUID = 1L;

	/**
	 * Metodo per restituire l'username dell'utente.
	 * @return {@link String} username.
	 */
	public String getUsername();

	/**
	 * Metodo per settare l'username dell'utente.
	 * @param username {@link String} username.
	 */
	public void setUsername(String username) ;

	/**
	 * Metodo per restituire la password dell'utente.
	 * @return {@link String} password.
	 */
	public String getPassword() ;

	/**
	 * Metodo per settare la password dell'utente.
	 * @param password {@link String} password.
	 */
	public void setPassword(String password) ;

	/**
	 * Metodo per restituire il ruolo dell'utente.
	 * @return {@link Role} ruolo.
	 */
	public Role getRole() ;

	/**
	 * Metodo per settare il ruolo dell'utente.
	 * @param role {@link Role} ruolo.
	 */
	public void setRole(Role) ;

	/**
	 * Metodo per restituire il nome dell'utente.
	 * @return {@link String} nome.
	 */
	public String getName() ;

	/**
	 * Metodo per settare il nome dell'utente.
	 * @param name {@link String} nome.
	 */
	public void setName(String name) ;

	/**
	 * username.
	 */
	private String username;
	/**
	 * password.
```

```
	 */
	private String password;
	/**
	 * nome.
	 */
	private String name;
	/**
	 * ruolo.
	 */
	private Role;
}
```

3.3. Package manager

OrdinationManager.java

```
package manager;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import DAO.DetailDAO;
import DAO.OrderDAO;
import DAO.OrderedProductDAO;
import DAO.ProductDAO;
import DAO.TableDAO;
import entity.Cart;
import entity.CartProduct;
import entity.Detail;
import entity.Order;
import entity.OrderedProduct;
import entity.Product;
import entity.Table;
import entity.User;
import utility.ManageReceipt;

/**
 * Classe gestore che offre servizi riguardanti le ordinazioni.
 */

public class OrdinationManager {

	/**
	  * Oggetto usato per eseguire le operazioni di "verifica qta prodotto e conferma
	 ordine" come una sequenza atomica.
	 */
	private static Object lock = new Object();

	/**
	 * Metodo per aggiungere un dettaglio al carrello.
	 * @param id {@link Integer} id!=null & id>0 indica l'id del dettaglio.
	 * @param cartList {@link Cart} cartList!=null indica il carrello.
	 * @return {@link Boolean} esito operazione.
	 */
	public static boolean addDetailToCart(int id, Cart cartList) ;



	/**
```

```
     * Metodo per rimuovere un dettaglio dal carrello.
     * @param id {@link Integer} id!=null & id>0 indica l'id del dettaglio.
     * @param cartList {@link Cart} cartList!=null indica il carrello.
     * @return {@link Boolean} esito operazione.
     */
    public static boolean removeDetailFromCart(int id, Cart cartList) ;

    /**
     * Metodo per aggiungere un prodotto al carrello.
     * @param idProd {@link Integer} idProd!=null & idProd>0 indica l'id del prodotto.
     * @param quant {@link Integer} quant!=null & quant>0 indica la qta del prodotto.
     * @param {@link Cart} cartList!=null indica il carrello.
     * @return {@link Boolean} esito operazione.
     */
    public static boolean addProductToCart(int idProd, int quant, Cart cartList) ;

    /**
     * Metodo per rimuovere un prodotto dal carrello.
     * @param id {@link Integer} id!=null & id>0 indica l'id del prodotto.
     * @param cartList {@link Cart} cartList!=null indica il carrello.
     * @return {@link Boolean} esito operazione.
     */
    public static boolean removeProductFromCart(int id, Cart cartList) ;

    /**
     * Metodo per eliminare l'ordine dallo storage.
     * @param o {@link Order} o!=null indica l'ordine.
     */
    public static void deleteOrderFromStorage(Order o) ;

    /**
     * Metodo per creare o aggiornare un'ordinde se già esiste.
     * @param o {@link Order} o!=null indica l'ordine.
     * @param u {@link User} u!=null indica l'utente che effettua l'ordine.
     * @param table_id {@link Integer} table_id!=null & table_id>0 indica l'id del
       tavolo.
     * @return
     */
    public static Order doCreateOrUpdateOrderByUser(Order o, User u, int table_id) ;
    /**
     * Metodo che restituisce l'elenco degli ordini di un tavolo.
     * @param id {@link Integer} id!=null & id>0 indica l'id del tavolo.
     * @return {@link ArrayList} elenco ordini.
     */
    public static ArrayList<Order> tableOrders(int id) ;
    /**
     * Metodo per confermare un'ordine di dettagli.
     * @param cartList {@link Cart} cartList!=null indica il carrello.
     * @param o {@link Order} o!=null indica l'ordine.
     * @return {@link Boolean} esito operazione.
     */
    public static boolean detailsConfirm(Cart cartList, Order o) ;
    /**
     * Metodo per confermare un'ordine.
     * @param o {@link Order} o!=null indica l'ordine.
     * @param cart {@link Cart} cart!=null indica il carrello.
     * @param totalOrder {@link Float} totalOrder!=null & totalOrder>0 indica il
       totale dell'ordine.
     * @param methodPay {@link String} methodPay!=null indica il metodo di pagamneto.
     * @return {@link Integer} esito operazione.
     */
    public static int orderConfirm(Order o, Cart, float totalOrder, String methodPay)
;
```

```
	/**
	 * Metodo per ripristinare le qta dei prodotti se l'ordine non va a buon fine.
	 * @param o {@link Order} o!=null indica l'ordine.
	 * @param cart {@link Cart} cart!=null indica il carrello.
	 */
	private static void restoreProducts(Order o, Cart) ;

	/**
	 * Metodo per verificare la disponibilità e scalare la qta di un prodotto.
	 * @param cart {@link Cart} cart!=null indica il carrello.
	 * @param o {@link Order} o!=null indica l'ordine.
	 * @return {@link Integer} esito operazione.
	 */
	private static int checkAvailabilityAndScaleQuantity(Cart cart, Order o) ;

	/**
	 * Metodo che restituisce un prodotto se la qta richiesta è disponibile.
	 * @param idProd {@link Integer} idProd!=null & idProd>0 indica l'id del prodotto.
	 * @param qta {@link Integer} qta!=null & qta>0 indica la qta richiesta.
	 * @return {@link Product} prodotto.
	 */
	private static Product getProductIfAvailable(int idProd, int qta) ;

	/**
	 * Metodo per inviare la stampa di un'ordine di soli dettagli.
	 * @param cartList {@link Cart} cartList!=null indica il carrello.
	 * @param o {@link Order} o!=null indica l'ordine.
	 * @return {@link Boolean} esito operazione.
	 */
	private static boolean printDetailHandler(Cart cartList, Order o) ;

	/**
	 * Metodo per inviare la stampa di un'ordine.
	 * @param cartList {@link Cart} cartList!=null indica il carrello.
	 * @param o {@link Order} o!=null indica l'ordine.
	 * @return {@link Boolean} esito operazione.
	 */
	private static boolean printOrderlHandler(Cart cartList, Order o) ;

}
```

ReservationManager.java

```
package manager;

import java.util.ArrayList;

import DAO.OrderDAO;
import DAO.OrderedProductDAO;
import DAO.ReservationOptionDAO;
import entity.Table;
import DAO.TableDAO;
import entity.OrdersTotals;
import entity.ReservationOption;


/**
 * Classe gestore che offre servizi riguardanti le prenotazioni.
 */
public class ReservationManager {
```

```
	/**
	 * Metodo che restituisce i costi delle prenotazioni.
	 * @return {@link ReservationOption} costi prenotazioni.
	 */
	public static ReservationOption getReservationsCost() ;

	/**
	 * Metodo che restituisce l'elenco dei tavoli disponibili.
	 * @return {@link ArrayList} elenco tavoli disponibili.
	 */
	public static ArrayList<Table> availableTable() ;

	/**
	 * Metodo che restituisce l'elenco dei tavoli non disponibili.
	 * @return {@link ArrayList} elenco tavoli prenotati.
	 */
	public static ArrayList<Table> reservationsList();

	/**
	 * Metodo che restituisce l'elenco dei tavoli.
	 * @return {@link ArrayList} elenco tavoli.
	 */
	public static ArrayList<Table> tablesList() ;

	/**
	 * Metodo che restituisce le informazioni di un tavolo.
	 * @param idTable {@link Integer} idTable!=null & idTable>0 indica l'id del
	   tavolo.
	 * @return {@link Table} tavolo con le informazioni.
	 */
	public static Table reservationInfo(int idTable);

	/**
	 * Metodo che restituisce i totali di ogni tavolo.
	 * @return {@link ArrayList} totali tavoli.
	 */
	public static ArrayList<OrdersTotals> tablesReport();

	/**
	 * Metodo per effettuare un prenotazione ad un tavolo.
	 * @param idTable {@link Integer} idTable!=null & idTable>0 indica l'id del
	   tavolo.
	 * @param reservationName {@link String} reservationName!=null indica il nome di
	   chi prenota il tavolo.
	 * @param peopleNumber {@link Integer} peopleNumber!=null & peopleNumber>0 indica
	   il numero di persone.
	 * @param budg {@link Float} budg!=null & budg>0 indica il budget.
	 * @param ro {@link ReservationOption} ro!=null indica i costi delle prenotazioni.
	 */
	public static void newReservation(int idTable, String reservationName, int
peopleNumber, float budg,ReservationOption ro) ;

	/**
	 * Metodo per modificare una prenotazione.
	 * @param id_table {@link Integer} id_table!=null & id_table>0 indica l'id del
	   tavolo.
	 * @param people {@link Integer} people!=null & people>0 indica il numero di
	   persone.
	 * @param name {@link String} name!=null indica il nome di chi prenota il tavolo.
	 * @param ro {@link ReservationOption} ro!=null indica i costi delle prenotazioni.
	 */
	public static void editReservation(int id_table, int people, String name,
ReservationOption ro) ;
```

```
	/**
	 * Metodo per modificare il tipo di tavolo.
	 * @param id {@link Integer} id!=null & id>0 indica l'id del tavolo.
	 * @param isLuxus {@link Boolean} isLuxus!=null indica se il tavolo è di lusso.
	 */
	public static void editTableType(int id, boolean isLuxus) ;

	/**
	 * Metodo per modificare i costi delle prenotazioni.
	 * @param normalTablePrice {@link Float} normalTablePrice!=null &
	    normalTablePrice>0 indica il costo dei tavoli normali.
	 * @param luxusTablePrice {@link Float} luxusTablePrice!=null & luxusTablePrice>0
	   indica il costo dei tavoli di lusso.
	 */
	public static void editReservationPrice(float normalTablePrice, float
luxusTablePrice) ;

	/**
	 * Metodo per cancellare una prenotazione.
	 * @param id_table {@link Integer} id_table!=null & id_table>0 indica l'id del
	   tavolo.
	 * @return {@link Boolean} esito operazione.
	 */
	public static boolean removeReservation(int id_table) ;

	/**
	 * Metodo per azzerare le prenotazioni.
	 */
	public static void clearReservations() ;

}
```

UserManager.java

```
package manager;
import DAO.UserDAO;
import entity.User;

/**
 * Classe gestore che offre servizi riguardanti gli utenti.
 */
public class UserManager {

	/**
	 * Metodo per verificare le credenziali di un utente.
	 * @param username {@link String} username!=null indica l'username.
	 * @param password {@link String} password!=null indica la password.
	 * @return {@link User} l'utente cercato se le credenziali sono corrette,
	   altrimenti null.
	 */
	public static User checkUser(String username, String password) ;

}
```

WarehouseManager.java

```
package manager;
import java.util.ArrayList;
import DAO.DetailDAO;
```

```
import DAO.ProductDAO;
import entity.Detail;
import entity.Product;

/**
 * Classe gestore che offre servizi riguardanti i prodotti in magazzino.
 */
public class WarehouseManager {

	/**
	 * Metodo che restituisce i prodotti della cambusa.
	 * @return {@link ArrayList} prodotti cambusa.
	 */
	public static ArrayList<Product> showOnlyGalleyProducts() ;

	/**
	 * Metodo che restituisce i prodotti del bar.
	 * @return {@link ArrayList} prodotti bar.
	 */
	public static ArrayList<Product> showOnlyBarProducts() ;

	/**
	 * Metodo che restituisce tutti i prodotti.
	 * @return {@link ArrayList} elenco prodotti.
	 */
	public static ArrayList<Product> showAllProducts() ;

	/**
	 * Metodo che restituisce tutti i dettagli.
	 * @return {@link ArrayList} elenco dettagli.
	 */
	public static ArrayList<Detail> showOrderDetails() ;

	/**
	 * Metodo che permette la modifica di un prodotto.
	 * @param id {@link Integer} id!=null & id>0 indica l'id del prodotto
	 * @param location {@link Integer} location!=null & location>0 indica la
	   locazione.
	 * @param price {@link Float} price!=null & price>0 indica il prezzo.
	 * @param galley {@link Integer} galley!=null & galley>0 indica la qta del
	   prodotto in cambusa.
	 * @param bar {@link Integer} bar!=null & bar>0 indica la qta del prodotto in bar.
	 * @param warehouse {@link Integer} warehouse!=null & warehouse>0 indica la qta
	   del prodotto in magazzino.
	 */
	public static void editProduct(int id, int location, float price, int galley, int
bar, int warehouse, String flag) ;


}
```

3.4. Package utility

ConnectionPool.java

```
package utility;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.TimeZone;

import org.apache.tomcat.jdbc.pool.DataSource;
import org.apache.tomcat.jdbc.pool.PoolProperties;

/**
 * Classe che si occupa di gestire le connessioni al db in locale.
 */
public class ConnectionPool {

	/**
	 * Metodo per ottenere la connessione del database.
	 * @return {@link Connection} connessione al db.
	 */
	public static Connection getConnection();

	/**
	 * datasource.
	 */
	private static DataSource datasource;
	/**
	 * ultima chiamata al db.
	 */
	private static long lastCall;

}
```

ManageReceipt.java

```
package utility;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;
import javax.print.PrintService;
import javax.print.event.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.printing.PDFPageable;
import com.itextpdf.text.*;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Font;
import com.itextpdf.text.List;
import com.itextpdf.text.ListItem;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Rectangle;
import com.itextpdf.text.pdf.BaseFont;
import com.itextpdf.text.pdf.PdfPCell;
```

```
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.InputStream;
import javax.print.Doc;
import javax.print.DocFlavor;
import javax.print.DocPrintJob;
import javax.print.PrintException;
import javax.print.SimpleDoc;

import entity.*;

/**
 * Classe che si occupa di gestire la creazione del documento da stampare.
 */
@WebServlet("/TestPrint")
public class ManageReceipt {

	private String filename;

	/**
	 * Metodo che crea il documento.
	 * @param cart {@link Cart} cart!=null carrello.
	 * @param order {@link Order} order!=null ordine.
	 * @param filename {@link String} filename!=null nome file.
	 */
	public void createDocument(Cart cart, Order, String filename);

	/**
	 * Metodo per inviare la stampa.
	 * @return {@link Boolean} esito stampa.
	 */
	public boolean printToPrinter();

	/**
	 * Metodo che crea il documento per i dettagli in un ordine.
	 * @param cart {@link Cart} cart!=null carrello.
	 * @param filename {@link String} filename!=null nome file.
	 * @param tableNumber {@link Integer} tableNumber!=null && tableNumber>0 numero
	   tavolo.
	 */
	public void createDetailsDocument(Cart cart, String filename, int tableNumber);
}
```

PrintToPrinter.java

```
package utility;
import java.awt.print.PrinterJob;
import javax.print.*;
/**
 * Classe che si occupa di gestire la stampa.
 */
public class  PrintToPrinter {
	/**
	 * Restituisce il servizio di stampa specificato (null se non lo trova).
	 * @param printerName {@link String} printerName!=null indica il nome della
	   stampante.
	 * @return {@link PrintService}
	 */
	public static PrintService findPrintService(String printerName) ;
```

```
	/**
	 * Restituisce un'istanza di printer job con il printer service specificato.
	 * @param printerName {@link String} printerName!=null indica il nome della
	   stampante.
	 * @return {@link PrinterJob}
	 */
	public static PrinterJob findPrinterJob(String printerName) throws Exception ;

	/**
	 * L'elenco delle stampanti non si aggiorna necessariamente se si modifica
	   l'elenco di
	 * stampanti all'interno dell'O / S; eseguire per aggiornare se necessario.
	 */
	public static void refreshSystemPrinterList() ;
}
```

3.5. Package service

MakeRequest.java

```
package service;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
/**
 * Servlet che si occupa di effettuare le richieste alla piattaforma online.
 */
public class MakeRequest {
	private final static String USER_AGENT = "Mozilla/5.0";
	private final static String HOST="http://localhost:80/mythos_pr/src/webservices/";

	/**
	 * Metodo che invia la richiesta all'host dove è situata la piattaforma per i p.r.
	 * @param resource {@link String} resource!=null indica la risorsa(nel nostro caso
	   il link con i parametri GET).
	 * @return {@link String} la risposta del server.
	 */
	protected static String sendGet(String resource);
}
```

Interfacce delle classi della piattaforma online

Essendo sviluppato in PHP la piattaforma online, non ci si riferisce soltanto alle interfacce delle classi ma anche alle pagine che faranno poi da manager. Per ogni pagina che non conterrà una classe, in questa sezione verrà specificato soltanto la signature delle funzioni che poi verranno richiamate.

3.6. Package DAO

ClientListDAO.php

```php
<?php

    /**
     * Classe che si occupa di prelevare i dati dei clienti in ista dallo storage.
     */

    class clientListDAO{

        /**
         * @var $connessione mysqli viene memorizzarato lo stato della connessione.
         */
        private $connessione;

        /**
         * Metodo per aprire la connessione con il database.
         */
        private function openConnection();

        /**
         * Metodo per chiudere la connessione con il database.
         */

        private function closeConnection();



        /**
         * Metodo che dato l'id di un p.r. restituisce tutti i suoi clienti in lista.
         * @param $id string indica l'id del p.r. -> $id!= null
         * @return array restituisce l'elenco dei clienti in lista del p.r.
           con id uguale a $id.
         */
        public function doRetrieveByPrId($id);

        /**
         * Metodo che dato l'id di un cliente restituisce tutti i suoi dati.
         * @param $id int indica l'id del cliente -> $id!= null && $id>0
         * @return ClientList restituisce il cliente con id uguale a $id.
         */
        public function doRetrieveByClientId($id);

        /**
         * Metodo che restituisce tutti i clienti in lista ordinati per Cognome,Nome.
         * @return array restituisce la lista clienti.
         */
        public function doRetrieveAll();
        /**
          * Metodo che restituisce l'elenco clienti in base al nome e al cognome del
            cliente o p.r.
         * @param $name string $name!=null indica il nome del cliente o p.r.
         * @param $lastname string $lastname!=null indica il cognome del cliente o p.r.
         * @return array restituisce la lista clienti.
         */
        public function doRetrieveByNameAndLastname($name,$lastname);
```

```
        /**
         * Metodo che restituisce l'ultimo cliente inserito da un p.r.
         * @param $prUsername string $prUsername !=null indica l'id del p.r.
         * @return int restituisce l'id dell'ultimo cliente inserito (-1 se non va a
            buon fine la query).
         */
        public function doRetrieveLastClientIdByPrId($prUsername);

        /**
         * Metodo per salvare un cliente nella lista.
         * @param $client ClientList $client!=null indica i dati del cliente, compreso
           il p.r. che lo ha inserito.
         * @return bool restituisce l'esito dell'inserimento.
         */
        public function doSave($client);

        /**
         * Matodo per aggiornare i campi di un cliente
         * @param $client ClientList $client!=null indica i dati del cliente, compreso
           il p.r. che lo ha inserito.
         * @return bool restituisce l'esito della modifica
         */
        public function doUpdate($client);

        /**
         * Metodo per eliminare un cliente dalla lista conoscendo l'id.
         * @param $id int indica l'id del cliente -> $id!= null && $id>0.
         */
        public function doDeleteById($id);

        /**
         * Metotdo per azzerare la tabella client_list.
         * @return bool restituisce l'esisto della cancellazione.
         */
        public function doClear();
    }
?>
```

ListOptionDAO.php

```
<?php

    /**
      * Classe che si occupa di prelevare i dati delle opzioni della lista dallo
        storage.
      */

        class ListOptionDAO{

        /**
         * @var $connessione mysqli viene memorizzarato lo stato della connessione.
         */
                private $connessione;

        /**
         * Metodo per aprire la connessione con il database.
         */
        private function openConnection();

        /**
         * Metodo per chiudere la connessione con il database.
         */
        private function closeConnection();
```

```
        /**
         * Metodo che restiruisce le opioni della lista (abilitata/disabilitata).
         * @return ListOption|null
         */
        public function doRetrieveAll();

        /**
         * Matodo per aggiornare i campi delle opzioni della lista.
         * @param $option ListOption indica le opzioni della lista.
         * @return bool restituisce l'esito della modifica.
         */
        public function doUpdate($option);

        }
?>
```

PrDAO.php

```
<?php

    /**
     * Classe che si occupa di prelevare i dati dei Pr dallo storage.
     */

    class PrDAO {

        /**
         * @var $connessione mysqli viene memorizzarato lo stato della connessione.
         */
        private $connessione;

        /**
         * Metodo per aprire la connessione con il database.
         */
        private function openConnection();



        /**
         * Metodo per chiudere la connessione con il database.
         */
        private function closeConnection();

        /**
         * Metodo che restituisce un p.r. avendo username e password.
         * @param $usern string  $usern!=null indica l'username del p.r.
         * @param $pass string $pass!=null indica la password del p.r.
         * @return Pr|null restituisce il p.r. se esiste oppure null se non esiste.
         */
        public function doRetrieveByUserAndPassword($usern, $pass);

        /**
         * Medoto che restituisce tutti i p.r.
         * @return array restituisce l'elenco dei p.r.
         */
        public function doRetrieveAll();

        /**
         * Metodo che restituisce un p.r. conoscendo l'username.
         * @param $id string $id=!null indica l'username del p.r.
         * @return Pr|null restituisce il p.r. con tutti i suoi dati.
         */
        public function doRetrieveById($id);
```

```
        /**
         * Metodo per salvare un p.r.
         * @param $pr Pr indica il p.r. da memorizzare.
         * @return bool restituisce l'esito dell'inserimento.
         */
        public function doSave($pr);

        /**
         * Metodo per eliminare un p.r. conoscendo l'id.
         * @param $pr string indica l'username del p.r.
         * @return bool restituisce l'esito dlla cancellazione.
         */
        public function doDeleteById($pr);
    }

?>
```

3.7. Package entity

ClientList.php

```
<?php

    /**
     * Classe per memorizzare i dati dei clienti in lista.
     * Per tutte le istanze di ClienList $client_id>0
     */
     class ClientList implements JsonSerializable {

        /**
         * @return array|mixed
         */
        public function jsonSerialize();

        /**
         * Metodo che restituisce l'id del cliente.
         * @return int
         */
        public function getClient_id() ;
        /**
         * Metodo che restituisce il nome del cliente.
         * @return string
         */
        public function getName();

        /**
         * Metodo che restituisce il cognome del cliente.
         * @return string
         */
        public function getSurname();

        /**
         * Metodo che restituisce lo stato del cliente.
         * @return bool
         */
        public function isEntered();

        /**
         * Metodo che restituisce la data di inserimento in lista del cliente.
         * @return string
         */
        public function getAdded_date() ;
```

```
        /**
         * Metodo che restituisce il p.r. che ha inserito in lista il cliente.
         * @return Pr
         */
        public function getPr() ;

        /**
         * Metodo che setta l'id del cliente.
         * @param $client_id int indica il nuovo id del cliente.
         */
        public function setClient_id($client_id) ;

        /**
         * Metodo che setta il nome del cliente.
         * @param $name string indica il nuovo nome del cliente.
         */
        public function setName($name) ;

        /**
         * Metodo che setta il cognome del cliente.
         * @param $surname string indica il nuovo cognome del cliente.
         */
        public function setSurname($surname) ;

        /**
         * Metodo che setta lo stato del cliente del cliente.
         * @param $entered bool indica il nuovo stato del cliente.
         */
        public function setEntered($entered) ;

        /**
         * Metodo che setta la data di inserimento in lista del cliente.
         * @param $added_date string indica la nuova data.
         */
        public function setAdded_date($added_date) ;

        /**
         * Metodo che setta il p.r. che ha inserito in lista il cliente.
         * @param $pr Pr indica il nuovo p.r.
         */
        public function setPr($pr) ;
        /**
         * @var $client_id int indica l'id del cliente.
         */
        private $client_id;
        /**
         * @var $name string indica il nome del cliente.
         */
        private $name;
        /**
         * @var $surname indica il cognome del cliente.
         */
        private $surname;
        /**
         * @var $entered bool  indica lo stato del cliente.
         */
        private $entered;
        /**
         * @var $added_date string  indica la data di inserimento in lista del cliente.
         */
        private $added_date;
        /**
         * @var $pr Pr  indica il p.r. che ha inserito in lista il cliente.
         */
        private $pr;
    }
?>
```

ListOption.php

```php
<?php

    /**
     * Classe per memorizzare le opzioni della lista clienti.
     */
     class ListOption implements JsonSerializable {
        /**
         * @return array|mixed
         */
        public function jsonSerialize();

        /**
         * Metodo che restituisce lo stato delle liste.
         * @return bool
         */
        public function getFlagList() ;

        /**
         * Metodo che setta lo stato delle liste.
         * @param $flagList bool indica il nuovo stato delle liste.
         */
        public function setFlagList($flagList) ;

        /**
         * @var $flagList bool indica lo stato delle liste.
         */
        private $flagList;
      }
?>
```

Pr.php

```php
<?php

    /**
     * Classe per memorizzare i dati riguardanti i Pr.
     */
     class Pr implements JsonSerializable {
        /**
         * @return array|mixed
         */
        public function jsonSerialize();

        /**
         * Metodo che restituisce l'username di un p.r.
         * @return string
         */
        public function getUsername() ;

        /**
         * Metodo che restituisce la password del p.r.
         * @return string
         */
        public function getPassword() ;
```

```
        /**
         * Metodo che restituisce il nome del p.r.
         * @return string
         */
        public function getName() ;

        /**
         * Metodo che restituisce il cognome del p.r.
         * @return string
         */
        public function getLastname() ;

        /**
         * Metodo che setta l'username del p.r.
         * @param $username string indica il nuovo username del p.r.
         */
        public function setUsername($username) ;

        /**
         * Metodo che setta la password del p.r.
         * @param $password string indica la nuova password del p.r.
         */
        public function setPassword($password) ;

        /**
         * Metodo che setta il nome del p.r.
         * @param $name string indica il nuovo nome del p.r.
         */
        public function setName($name);

        /**
         * Metodo che setta il cognnome del p.r.
         * @param $lastname string indica il  nuovo cognome del p.r.
         */
        public function setLastname($lastname);

        /**
         * @var $username string indica l'usenrame del p.r.
         */
        private $username;


        /**
         * @var $password string indica la password del p.r.
         */
        private $password;
        /**
         * @var $name string indica il nome del p.r.
         */
        private $name;
        /**
         * @var $lastname string indica il cognome del p.r.
         */
        private $lastname;
    }
?>
```

3.8. Package manager

ClientsManager.php

```
<?php

/**
 * Pagina gestore che offre servizi riguardanti la lista clienti.
 */

    /**
     * Servizio per abilitare o disabilitare le liste.
     * Abilita le liste se sono disabilitate.
     * Disabilita le liste se sono abilitate.
     * @return bool restituisce l'esito dell'operazione.
     */
    function enableOrDisableList();

    /**
     * Servizio per azzerare le liste.
     * @return bool restituisce l'esito dell'operazione.
     */
    function clearOrders();

    /**
     * Servizio per modificare lo stato di un cliente.
     * @param $clientId int $clientId=!null && $clientId>0 indica l'id del cliente
     * @return bool restituisce l'esito della modifica.
     */
    function editClientListStatus($clientId);

    /**
     * Servizio che restituisce la lista di clienti avente nome e cognome o del p.r. o
       del cliente.
     * @param $prName string indica il nome.
     * @param $prLastname string indica il cognome.
     * @return array
     */
    function clientListByPr($prName,$prLastname);

    /**
     * Servizio per inserire un cliente in lista.
     * @param $name string indica il nome del cliente.
     * @param $lastname string indica il cognome del cliente.
     * @param $prSession Pr incica il p.r.
     * @return int indica l'id del cliente inserito. (-1 se l'inserimento non è andato a
       buon fine).
     */
    function addClientToList($name,$lastname,$prSession);

    /**
     * Servizio per rimuovere un cliente dalla lista.
     * @param $id int indica l'id del cliente.
     */
    function removeClientFromList($id);

    /**
     * Servizio per ottere la lista clienti.
     * @return array
     */
    function clientList();
```

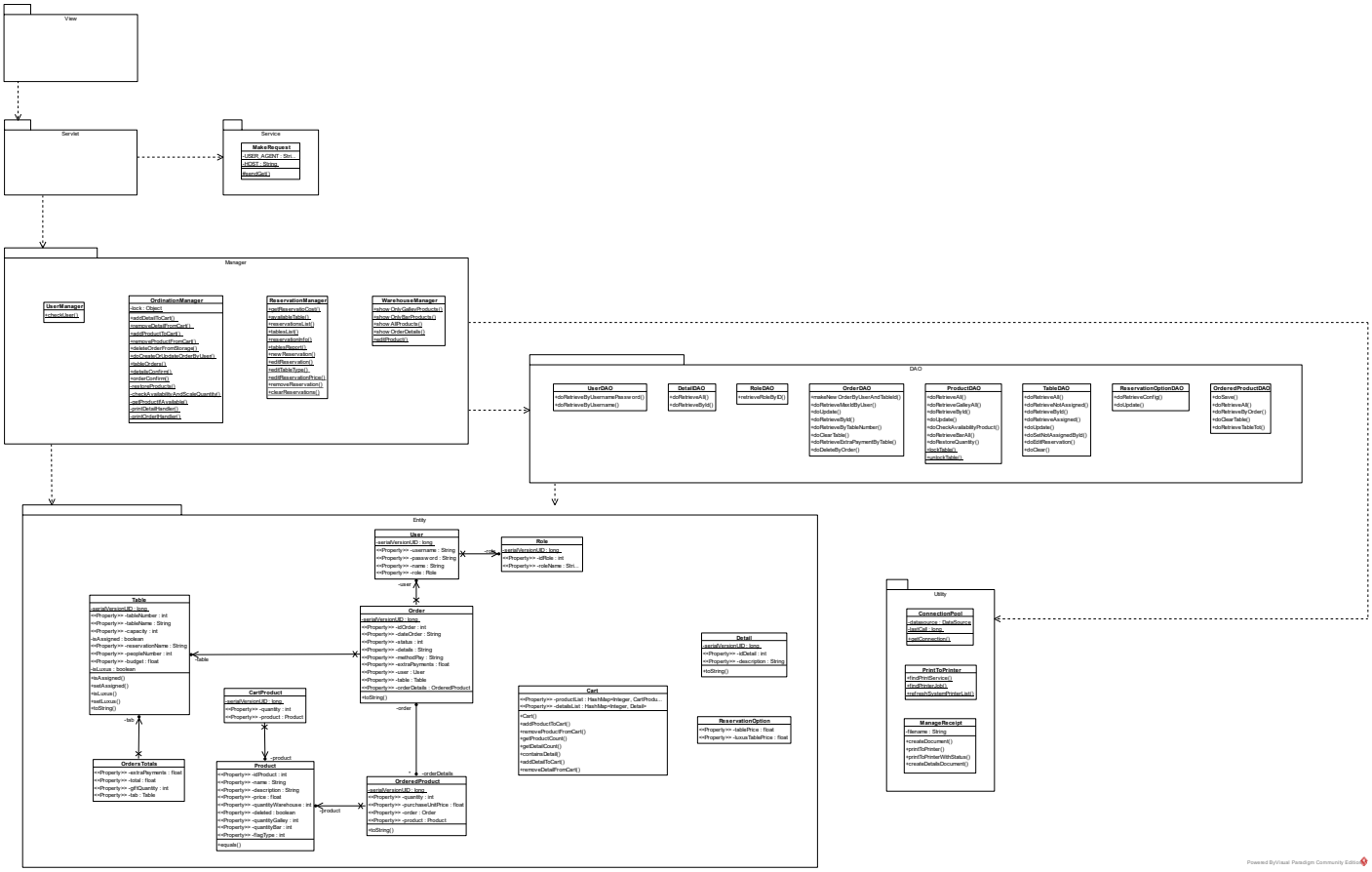
```
    /**
     * Servizio per cercare un cliente in lista.
     * @param $id int indica l'id del cliente.
     * @return ClientList
     */
    function searchClient($id);

    /**
     * Servizio per ottenre lo stato delle liste.
     * @return ListOption|null
     */
    function listStatus();

?>
```

PrManager.php

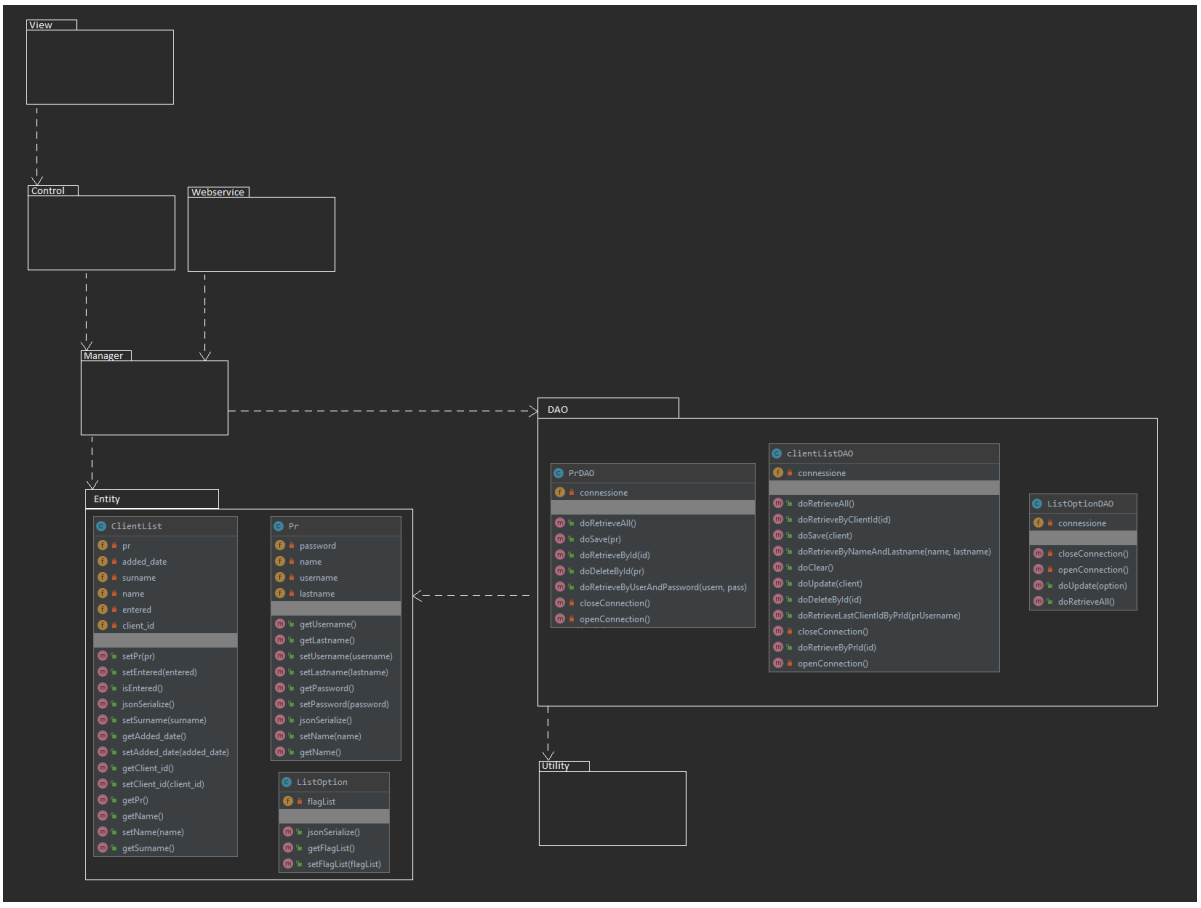
```
<?php
   /**
    * Pagina gestore che offre servizi riguardanti i Pr.
    */

    /**
     * Servizio per aggiungere un p.r.
     * @param $username string indica l'username del p.r.
     * @param $password string indica la password del p.r.
     * @param $name string indica il nome del p.r.
     * @param $lastname string indica il cognome del p.r.
     * @return bool restituisce l'esito dell'inserimento.
     */
    function addPR($username,$password,$name,$lastname);

    /**
     * Servizio per eliminare un p.r.
     * @param $username string indica l'username del p.r.
     * @return bool restituisce l'esito dell'eliminazione.
     */
    function deletePR($username);

    /**
     * Servizio che restituisce la lista di p.r.
     * @return array
     */
    function prList();

    /**
     * Servizio per verificare le credenziali di un p.r.
     * @param $user string indica l'username del p.r.
     * @param $pass string indica la password del p.r.
     * @return Pr|null
     */
    function checkPr($user, $pass);
?>
```

4. Class Diagram

4.1. Class diagram della piattaforma in locale.



4.2. Class diagram della piattaforma online.

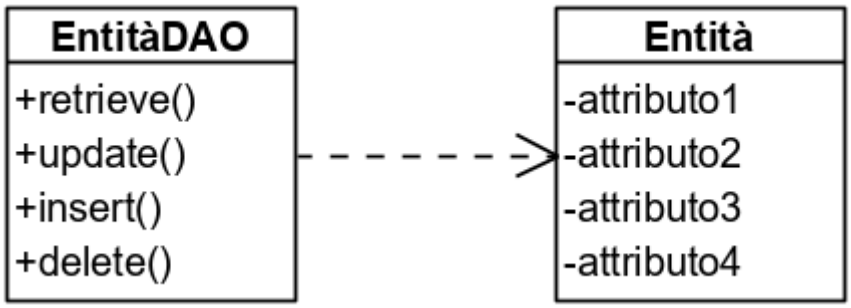


5. Design Pattern

Per la realizzazione del sistema Mythos sono stati utilizzati i seguenti design pattern:

5.1. DAO Pattern

Per l'accesso al database è stato impiegato il pattern DAO (Data Access Object). Il pattern è usato per separare le API, di basso livello, di accesso ai dati, dalla logica di business di alto livello. Per ogni tabella presente nel DB esisterà una sua corrispondente classe DAO che possiederà metodi per effettuare le operazioni CRUD. Ogni classe DAO utilizzerà esclusivamente la classe Entity corrispondente alla tabella su cui effettua le operazioni. Di seguito è presentato un modello di utilizzo del pattern:



5.2. Façade Pattern

Per la realizzazione dei servizi dei sottosistemi è stato usato il pattern Façade. Il pattern si basa sull'utilizzo di un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

Nel nostro caso, ogni sottosistema, identificato nell'SDD, possiede una classe chiamata manager. Questa implementa dei metodi che corrispondono ai servizi offerti dal sottosistema. Tali metodi, nella loro implementazione, utilizzeranno le classi entity e le classi DAO per eseguire il servizio da essi offerto. In output presenteranno un oggetto che servirà alla servlet per la presentazione del risultato dell'operazione. In questo modo si svincola completamente la logica di business, implementata dai manager, e la logica di controllo, implementata dalla servlet. Viene qui presentato un modello di utilizzo del pattern: