Aplicatie pentru facilitarea livrarii de produse alimentare

Aplicatie realizata de

Bursuc Andrei

Grupa 30234

Contents

[1. Requirements Analysis 3](#_Toc510046037)

[1.1 Assignment Specification 3](#_Toc510046038)

[1.2 Functional requirements 3](#_Toc510046039)

[1.3 Non-functional Requirements 3](#_Toc510046040)

[2. Use-Case 4](#_Toc510046041)

[3. System Architectual Design 5](#_Toc510046042)

[3.1 Architectural Pattern Description 5](#_Toc510046043)

[3.2 Diagram 5](#_Toc510046044)

[4. Class Design 5](#_Toc510046045)

[4.1 Design Patterns Description 5](#_Toc510046046)

[4.2 UML class diagram 6](#_Toc510046047)

[5. Data Model 10](#_Toc510046048)

[5.1 Database diagram 11](#_Toc510046049)

[6. Bibliografie 12](#_Toc510046050)

# Requirements Analysis

## Assignment Specification

Acesta tema urmareste realizarea unei aplicatii ce ajuta la comandarea si livrarea de produse alimentare. Aplicatia trebuie sa aiba doua tipuri de utilizatori: administratorul, ce poate vedea si modifica datele clientilor, dar si modifica si adauga produse in lista de produse, si utilizatorul obisnuit ce isi poate crea un cont nou ale carui date le poate edita cand doreste, de asemenea acesta in meniu isi creaza un cos nou in care adauga produse, putand efectua plata la finalizarea comenzii fie cu cardul sau cash.

## 

## 1.2 Functional requirements

Datele vor fi stocate intr-o baza de date. Foloseste modelul de design Layers pentru organizarea aplicatiei. Utilizati un moodel logic de domeniu (script de tranzactie sau model de domeniu) / un model hibrid de sursa de date (modul de tabel, inregistrare activa) si un model pur se sursa de date (gateway de date tabel, gateway de date rand, mapper de date).

## 1.3 Non-functional Requirements

Disponibilitate: aceasta apicatie desktop poate rula pe orice sistem modern ce dispune de un minim de resurse, ea neavand nevoie de o conexiune la internet

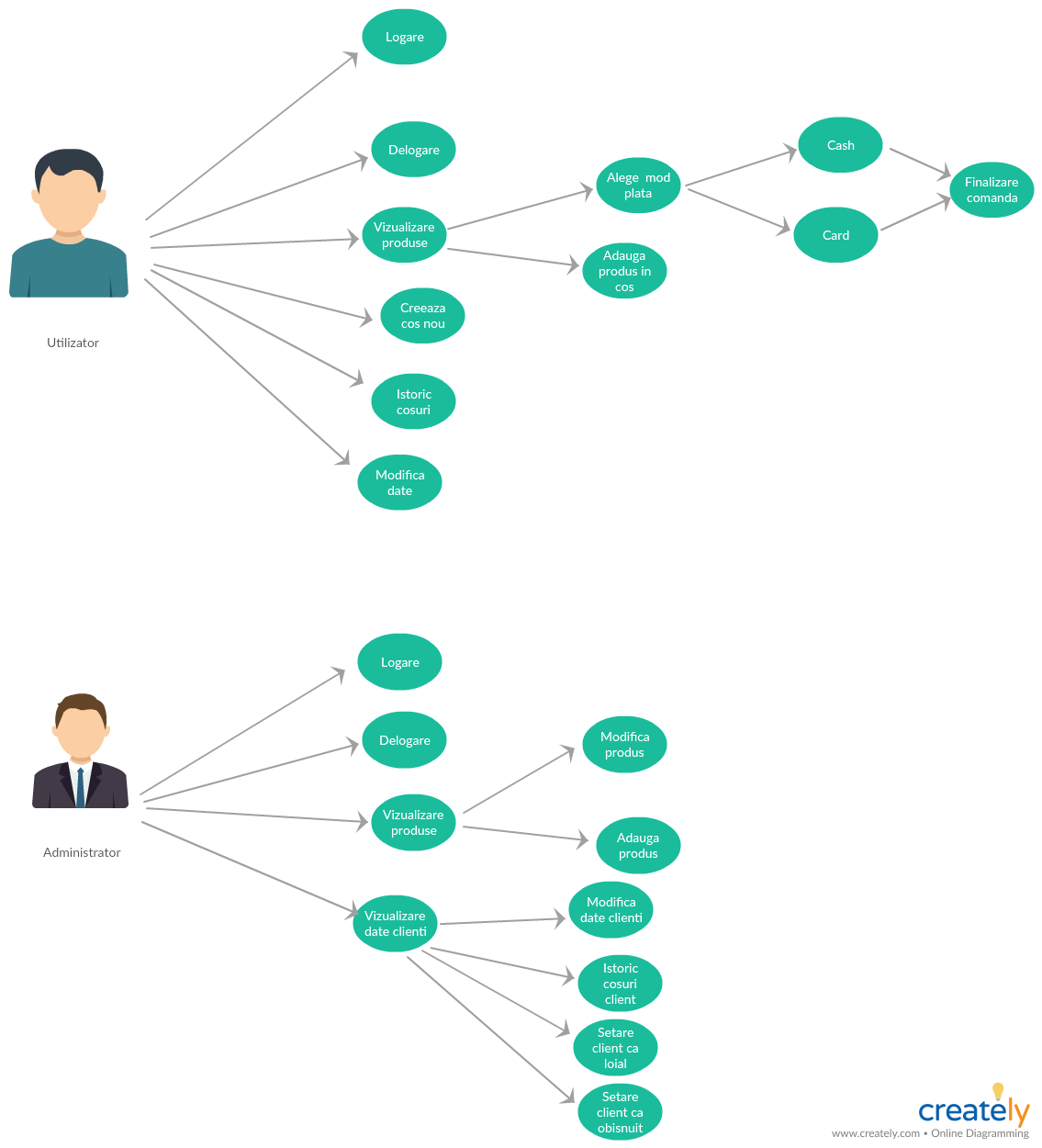
Extensibialitate: aplicatia poate fi extinsa cu alte functionalitati cum ar fi o descriere a produselor, o functie ce notifica clientii cu diferite oferte etc.

Mentenanta: imbunatatirea aplicatiei se poate face relativ usor, codul fiind bine structura in layere, la o imbunatatire majora nefiind nevoie sa fie modificat tot codul, ci doar din anumite layere.

Portabilitate: aplicatia ruleaza pe orice sistem de operare atata timp cat exista un mediu Java Runtime Environment instalat. Se recomanda utilizarea celei mai noi verisuni de JRE.

Securitate: Conexiunea la baze de date este una securizata nefiind accesul la ea de catre utilizatorii neautorizati, iar parola este criptata in baza de date cu metoda SHA256.

# Use-Case

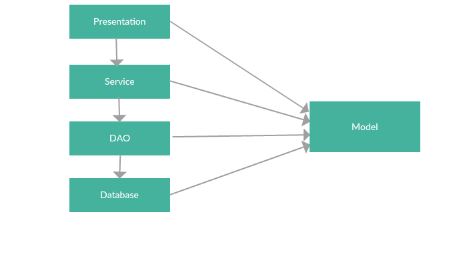


# System Architectual Design

## Architectural Pattern Description

Cea mai cunoscuta forma de architectura este cea de tip layered, cunoscuta ca si “n-tier architecture pattern”. Componentele dintr-o arhitectura de tip layered sunt organizate in layere orizontale, fiecare layer avand un rol specific in cadrul aplicatiei. Cu toate ca arhitecturie de tip layer nu specifica numarul sau tipul de layere ce trebuie sa existe in arhitectura, in cele mai multe exista patru mari layere: presentation, business, persistance si database. Fiecare layer are un rol specific si o responsabilitate in cadrul aplicatiei. Fiecare layer din arhitectura formeaza o abstractizare a lucrurilor ce trebuie facute pentru a satisface o anumita cerinta.

## Diagram



# Class Design

## 4.1 Design Patterns Description

Pentru realizarea aplicatiei am optat pentru utilizarea arhitecturii stratificate care imparte aplicatia in mai multe straturi:

„Presentation Layer” contine clasele folosite la definirea interfetei utilizatorului. Rolul său este de a evidenția informația obținută prin intermediul „Data Acces Layer”. Interfața a fost realizată cu ajutorul bibliotecii Javax.Swing utilizând plugin-ul special din Intelij, ce permite realizarea mai ușoară a unei interfețe atractive pentru utilizator și ușor de folosit.

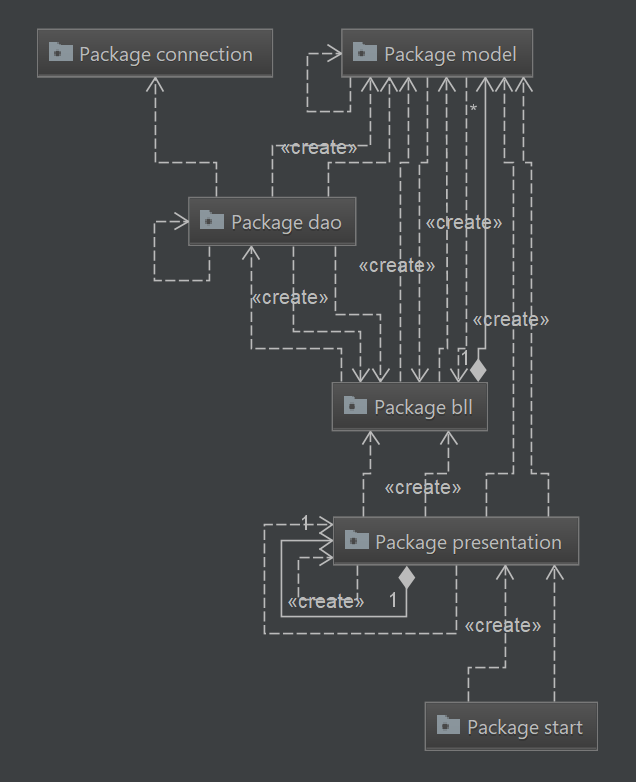
„Business Layer” contine clasele care incapsuleaza logica aplicatiei

„Data Acces Layer” contine clasele ce contin interogarile pentru baza de date si practic in cazul de fata definesc functiile prin care programul interactioneaza cu baza de date;

In plus in cadrul acestei architecturi mai avem „Modelul” ce contine clase care modeleaza fiecare tabel in parte din baza de date cu care programul interactioneaza.

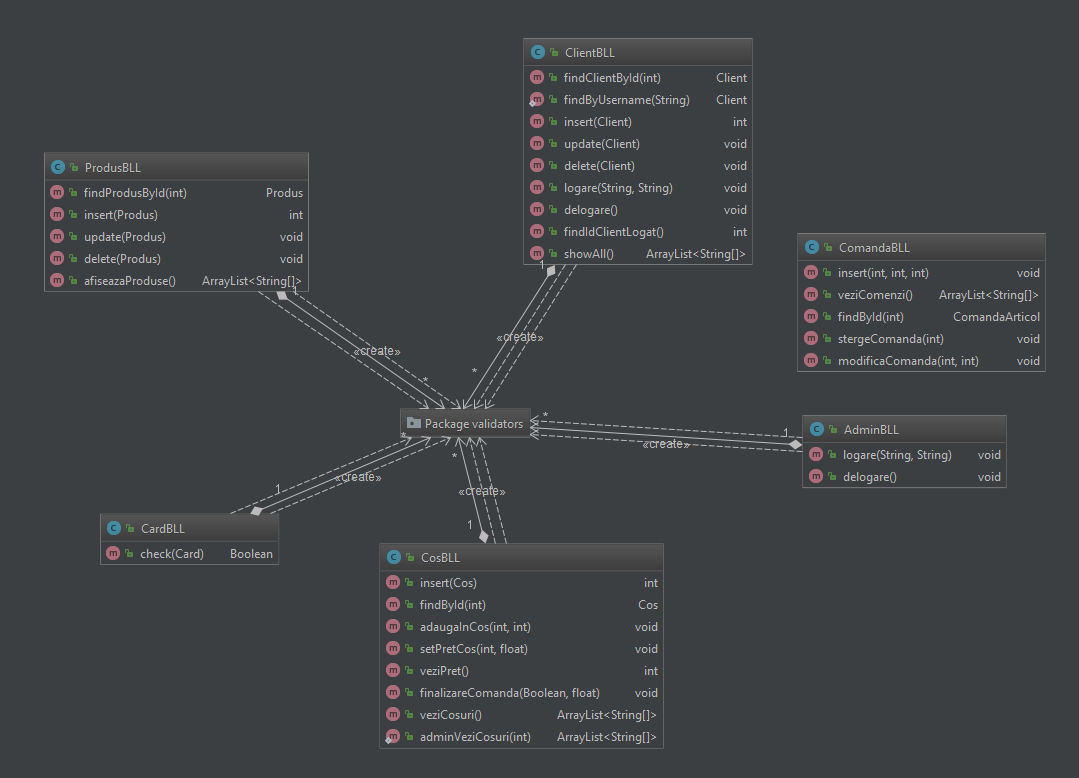
## 4.2 UML class diagram

In continuare este prezentata diagrama UML a pachetelor folosite in program iar mai apoi diagrama UML a fiecarui pachet in parte:



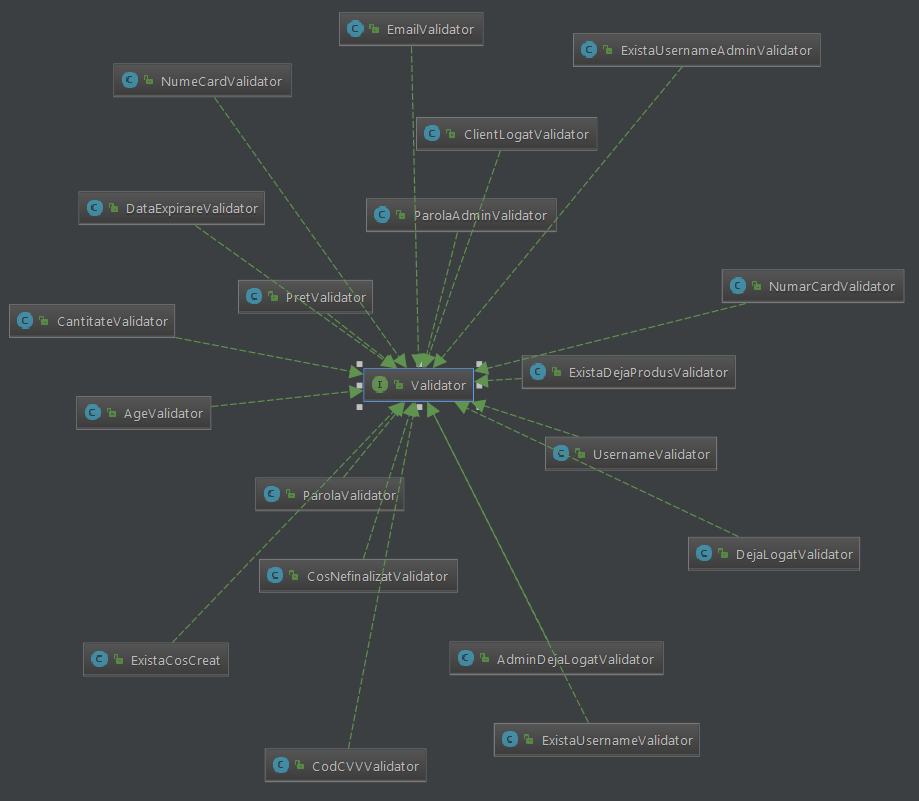
Proiectul fiind alcatuit din mai multe pachete vom prezenta unele dintre pachetele cele mai importante cu diagrama UML corespunzatoare.

**Package BLL ( Business Logic )**



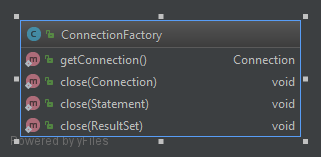
Acest pachet contine clasele ce verifica functiile din pachetul „Data Acces” aplicand pe ele validatorii ce spun daca interogarea pe baza de date se poate sau nu executa. Un exemplu ar fi clasa ClientBLL.

**Package bll.validators**



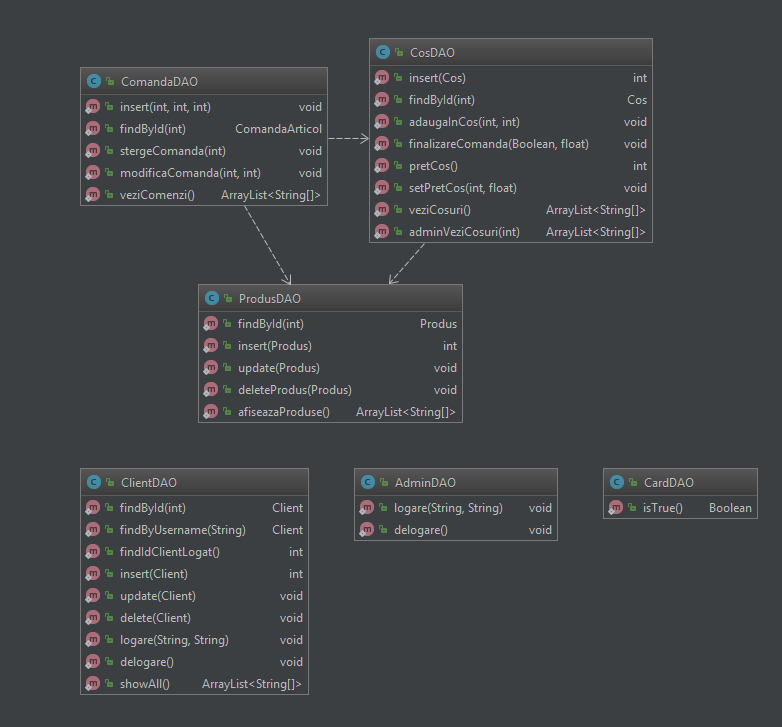
Scopul claselor din acest pachet este sa verifice daca datele introduse de utilizator sunt corecte, iar in cazul in care acestea sunt gresite metoda validate din clasa va arunca o exceptie.

**Package Connection**



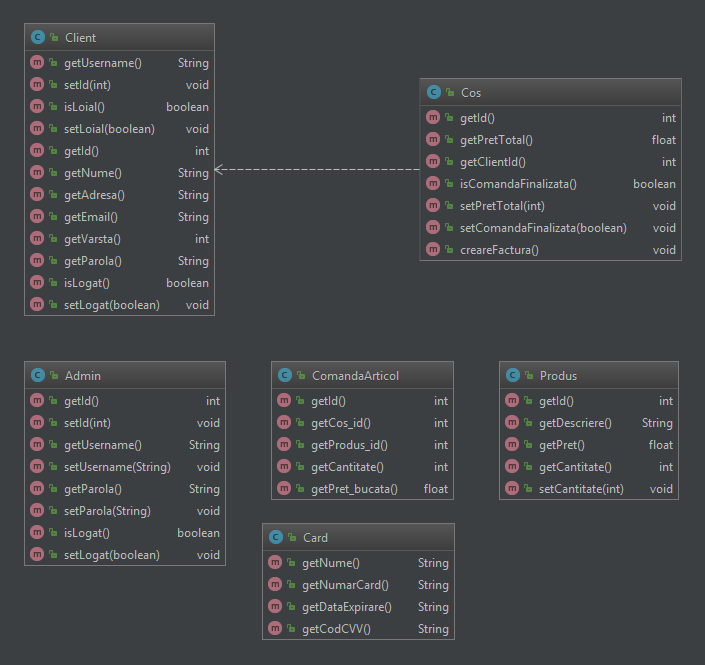
Acest pachet este alcatuit dintr-o singura clasa in care se foloseste pattern-ul singleton ce permite crearea doar a unei singure instante a clasei ConnectionFactory, acest lucru usurandu-ne modul in care realizam conexiunea cu baza de date inainte de apelarea unei interogari.

**Package dao**



Aici sunt definite clasele care contin metodele prin care interactionam cu baza de date si putem vedea sau modifica continutul ei.

**Package model**

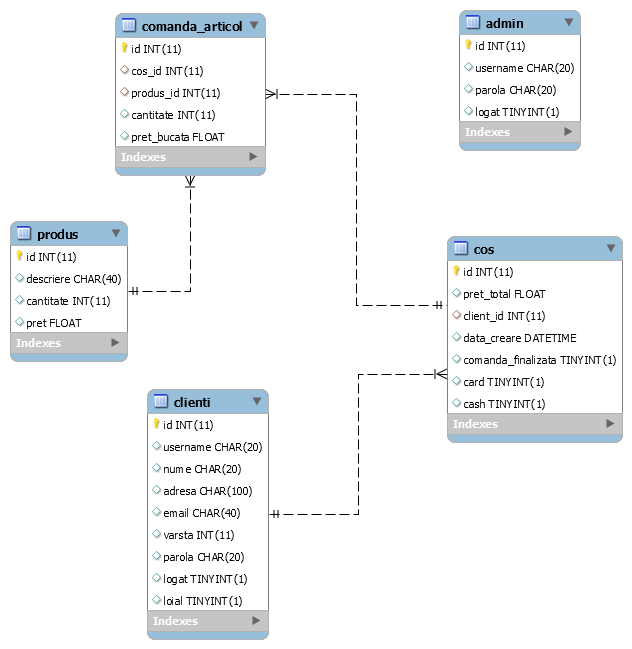


Clasele din acest pachet reprezinta corespondentul tabelelor din baza de date (cu exceptia clasei Card creata pentru a facilita verificarea formularului pentru card).

# Data Model

Baza de date este alcatuita din 5 tabele: Admin, Clienti, Produs, Comanda\_Articol si Cos. Tabelul admin este definit de campurile: id, username, parola, logat (camp boolean setat pe true cand unul din administratori este logat) Tabelul clienti este definit de campurile: id, username, nume, adresa, email, varsta, parola, logat (camp de tip boolean care confirma daca este sau nu logat clientul) si loial (camp boolean ce poate fi activat de admin daca el considera dupa vizualizarea comenzilor clientului ca el este loial, daca acest camp este activat clientul beneficiaza de o reducere de 5%), acest tabel ne ajuta sa stocam datele despre client si ne spune daca un client este logat sau nu. Tabelul Produs cu campurile: id, descriere, cantitate, pret stocheaza datele despre produse; Tabelul cos definit de: id, pret\_total, client\_id, data\_creare, comanda\_finalizata, cash, card ne ajuta sa facem legatura intre comenzi si fiecare client in parte, un client putand sa aiba mai multe comenzi ce au acelasi cos\_id, de asemenea campurile boleene cash si card ne ajuta sa identificam modul in care clientul a efectuat plata (cash sau cu cardul), tabelul comanda\_articol este definit de: id, cos\_id, produs\_id, cantitate, pret\_bucata.

## 5.1 Database diagram



# Bibliografie

1. <http://docs.oracle.com/javase/>
2. <http://stackoverflow.com/>
3. Kathy Sierra, Bert Bates. SCJP Sun Java Certified Programmer for Java 6. McGraw Hill, USA 2008
4. <https://dzone.com/articles/layers-standard-enterprise>