<Food delivery application>

Student: Man Silvana

**Group: 30234**

Table of contents

[1.Requirements Analysis 3](#_Toc510557503)

[1.1 Assignment Specification 3](#_Toc510557504)

[1.2 Functional Requirements 3](#_Toc510557505)

[1.3 Non-functional Requirements 3](#_Toc510557506)

[2.Use-Case Model 4](#_Toc510557507)

[3.System Architectural Design 5](#_Toc510557508)

[3.1 Architectural Pattern Description 5](#_Toc510557509)

[3.2 Diagrams 6](#_Toc510557510)

[4.UML Sequence Diagrams 8](#_Toc510557511)

[5.Class Design 8](#_Toc510557512)

[5.1 Design Patterns Description 8](#_Toc510557513)

[5.2 UML Class Diagram 10](#_Toc510557514)

[6.Data Model 11](#_Toc510557515)

[7.Presentation 11](#_Toc510557516)

[8.Bibliography 16](#_Toc510557517)

# 1. Requirements Analysis

## Assignment Specification

Use JAVA/C# API to design and implement a food delivery application. The application should have two types of users (a regular user represented by the customer and an administrator user) which have to provide a username and a password in order to use the application.

The customer can perform the following operations:

* Add/update/view own information (name, identity card number, personal numerical code, address, etc.).
* Create/update/delete/view shopping cart (items,amount of money, date and hour, delivery address etc.).
* Choose payment type.
* Visualization of historical data.

The administrator user can perform the following operations:

* CRUD on customers’ information.
* CRUD on menu
* Generate reports for a particular period containing the activities performed by a customer and set the customer as loyal (5% discount).

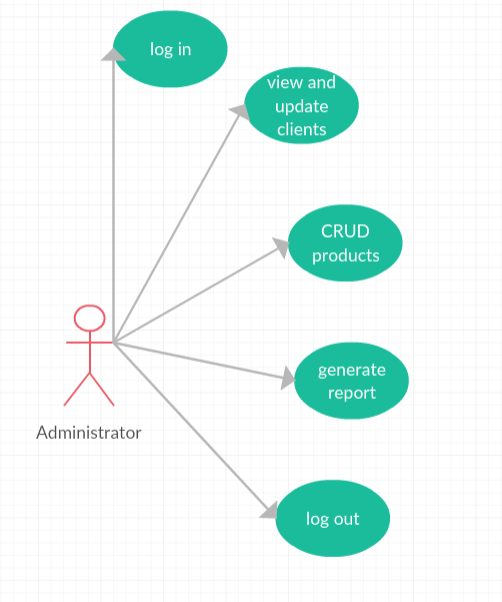
## Functional Requirements

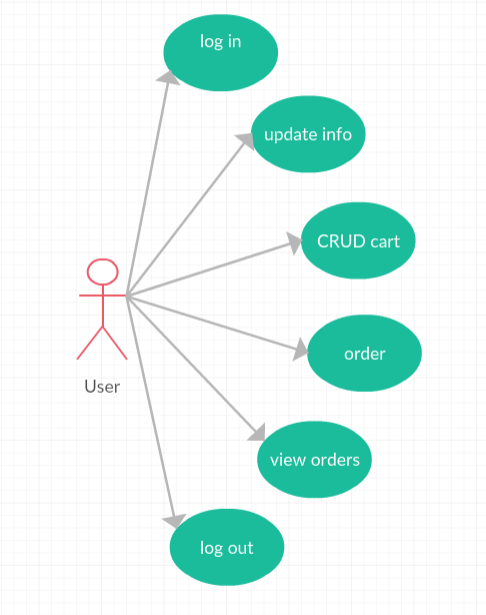
* Administratorul trebuie sa gestioneze produsele disponibile pe site, sa vada ce persoane isi fac cont pe site si sa poate oferi un discount pentru loialitate. De asemenea, el trebuie sa genereze un raport la o anumita perioada de timp cu activitatea unui anumit client.
* Clientii, pe de alta parte, trebuie sa isi creeze cont pe site , sa poata efectua o comanda, sa isi vada cosul de cumparaturi si sa il poata edita. De asemenea el are posibilitatea de a plati online, cu cardul sau ramburs.

## Non-functional Requirements

* Vizualizarea datelor din baza de date cu ajutorul tabelelor JTable; de asemenea, fiecare modificare adusa datelor trebuie sa fie vizibila in timp real
* Aplicarea pattern-ului architectural Layer
* Validarea datelor impotriva intrarilor nepotrivite

# Use-Case Model





Use case: autentificare ca si administrator

Primary actor: administrator

Main success scenario:

* Se introduce username-ul si parola administratorului
* Se apasa butonul “log in”
* Se deschide fereastra in care se vizualizeaza clientii si produsele
* Se executa functii pe client sau produse

Use case: plasare comanda

Primary actor: user

Main success scenario:

* Se introduce username-ul si parola
* Se apasa butonul “log in”
* Se deschide fereastra in care se pot vizualiza produsele
* Se adauga produse in cos
* Se alege o metoda de plata
* Se valideaza datele
* Se creeaza comanda

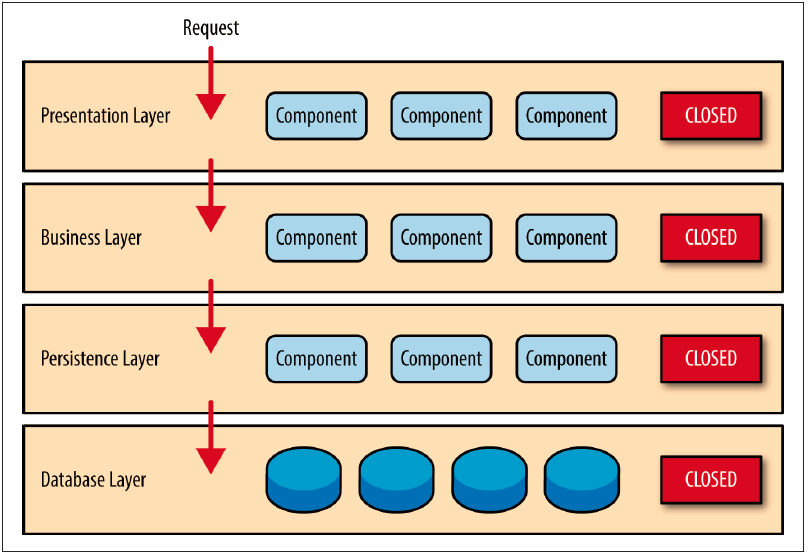
# System Architectural Design

## Architectural Pattern Description

Am folosit arhitectura de tip Layer. Este cunoscuta ca fiind o arhitectura n-tier, pe mai multe nivele. Aplicatia se structureaza in nivele de abstractizare. Fiecare layer produce servicii pentru urmatorul layer. Intr-o arhitectura relaxata, se poate sari peste un layer. In general se folosesc 4 layers pentru un sistem:

* Presentation layer (UI layer)
* Application layer (service layer)
* Business logic layer (domain layer)
* Data access layer (persistence layer)

Se foloseste pentru aplicatii desktop sau web.



## Diagrams

**Database Layer**- este ultimul nivel, aici se tin datele introduse de cele 2 tipuri de utilizator, dupa ce acestea sunt validate

**DAO Layer**- fiecare tabel din baza de date are o clasa specifica in care sunt implementate metodele care efectueaza CRUD pe acesta; este singurul layer care comunica direct cu baza de date si pentru implementarea acetuia se foloseste patternul Table Data Gateway

**Service Layer**- are rolul de a valida datele introduse in interfata de catre utilizator si de a le trimite mai departe la DAO; foloseste Table Model Pattern

**Presentation Layer**- contine partea de interfata a proiectului si ii permite utilizatorului sa interactioneze cu aplicatia

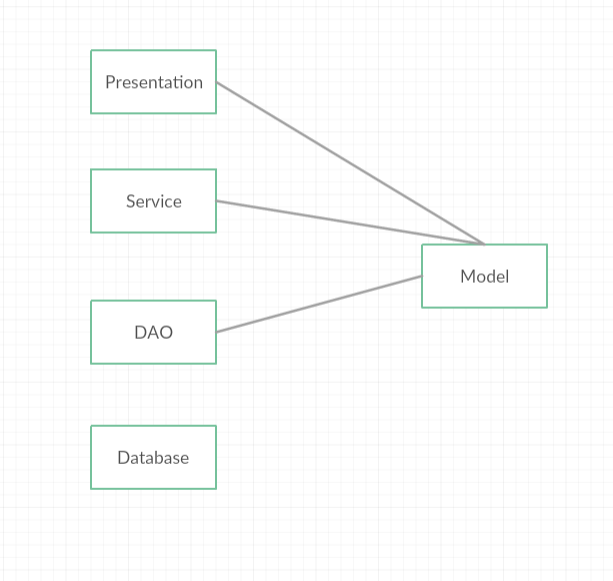


Diagrama de pachete:

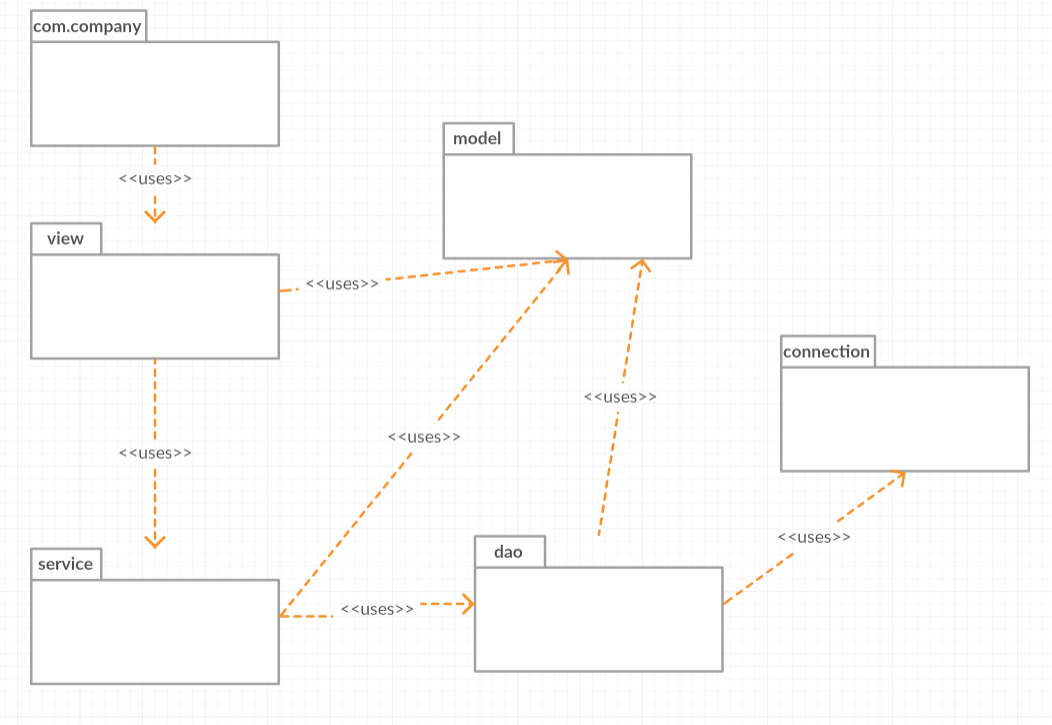
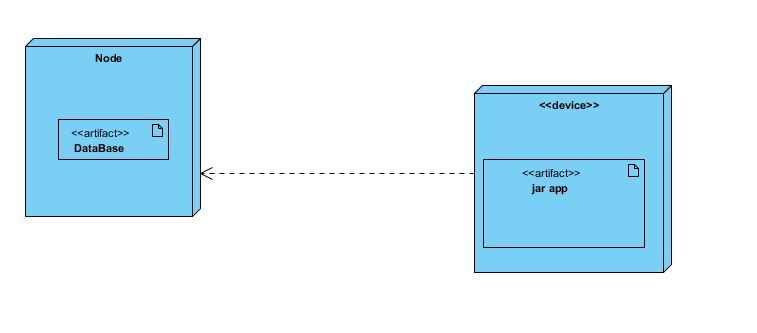
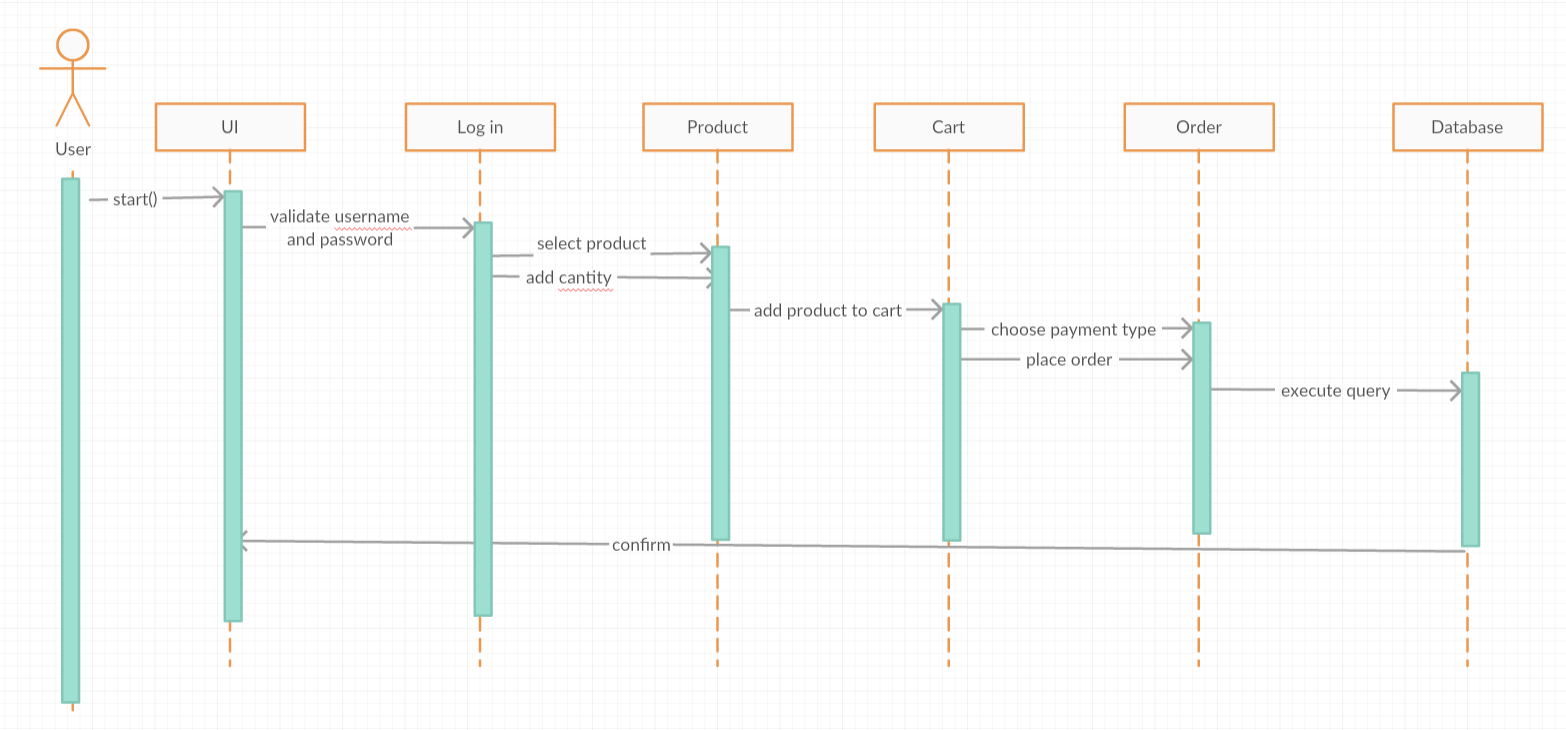


Diagrama de deployment:



# UML Sequence Diagrams

Diagrama de secventa pentru plasarea unei comenzi de catre un user:

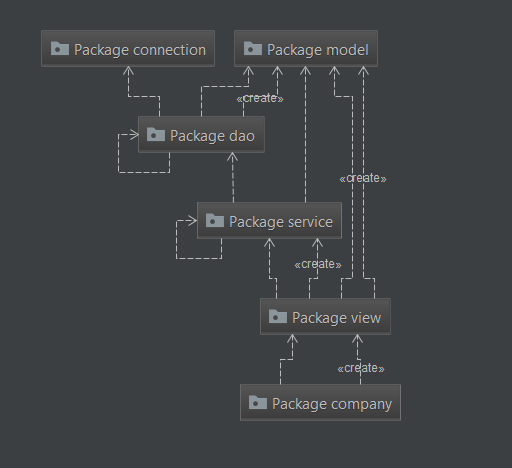
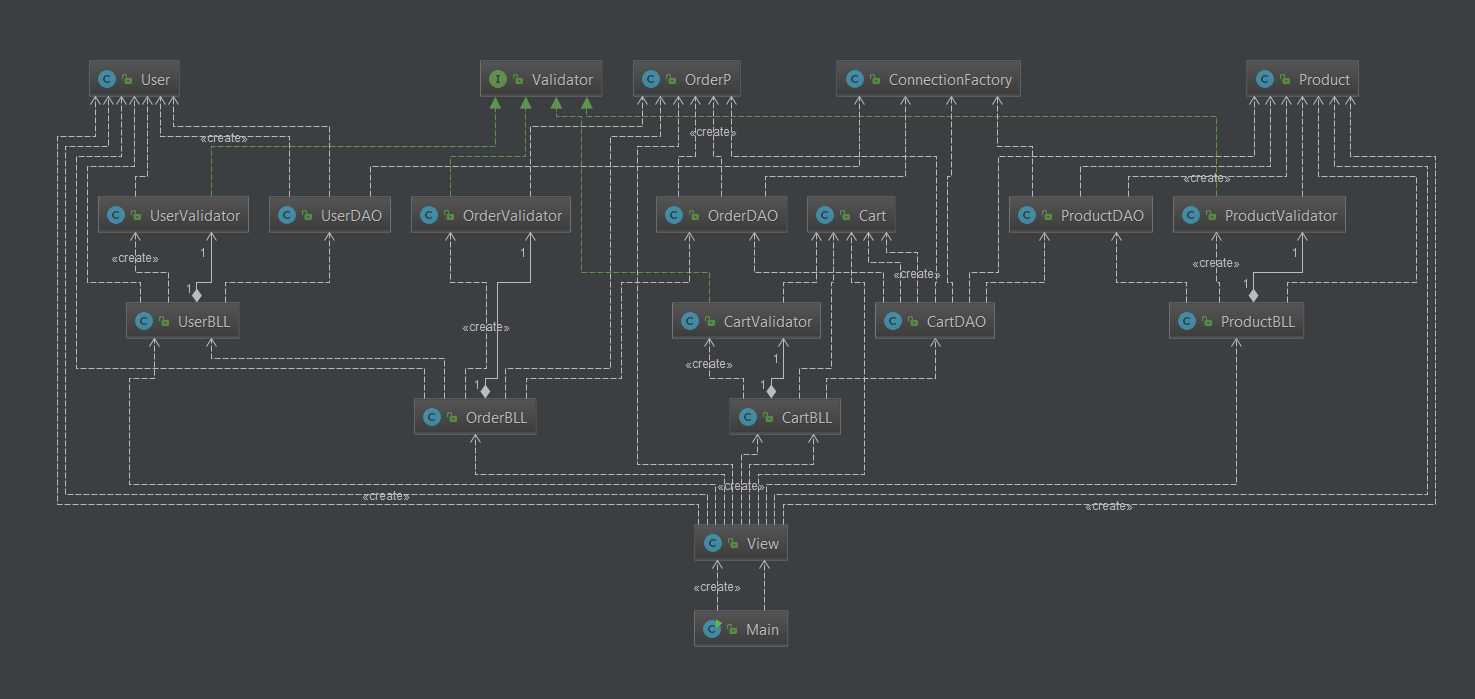


# Class Design

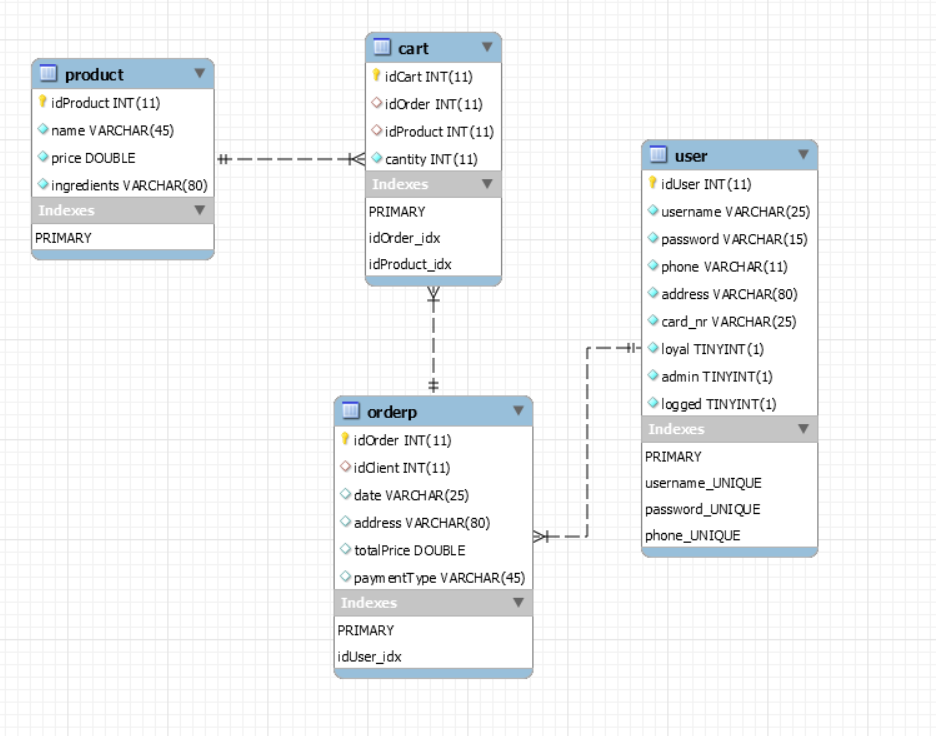
## Design Patterns Description

* **Singleton** – am folosit acest design pattern in clasa care face conexiunea cu baza de date. Astfel, numai o singura instanta a acestei clase va fi folosita in toate operatiile care au legatura cu baza de date.
* **Domain module** –pattern util pentru validarea datelor de intrare, folosit in partea de logica a proiectului.
* **Data mapper** – este un pattern folosit in partea de DAO a proiectului, care consta intr-o clasa folosita pentru a mapa datele in baza de date cu ajutorul unui string, un query.

## UML Class Diagram



# Data Model

**

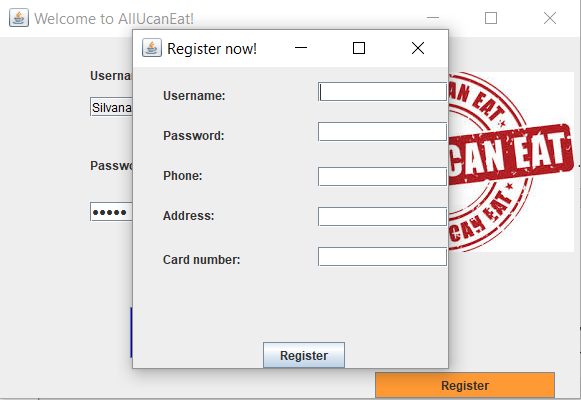
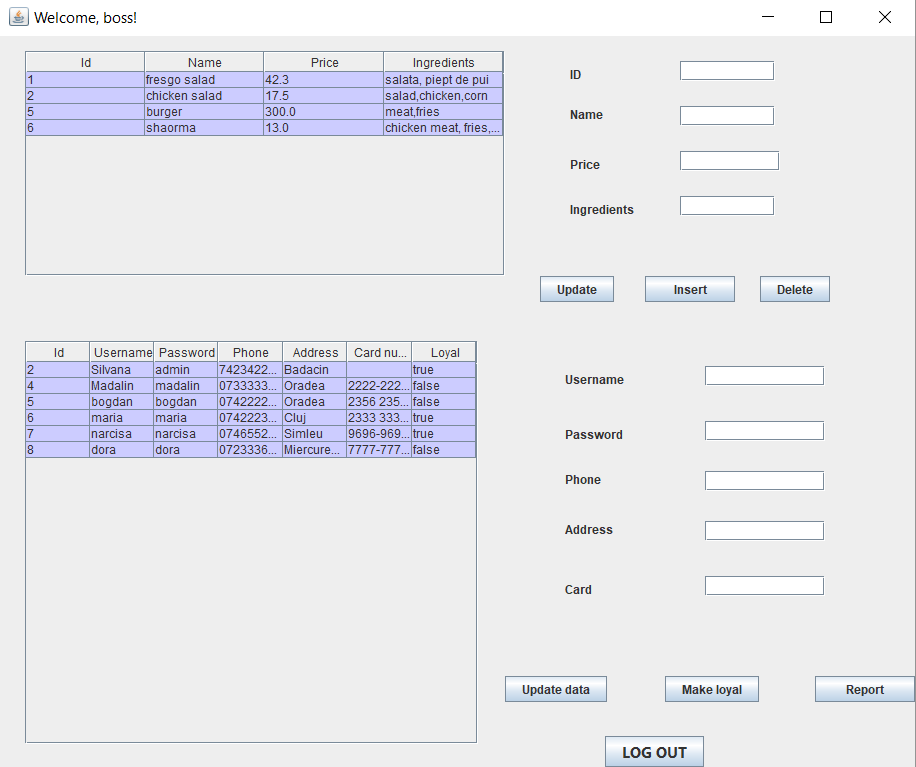
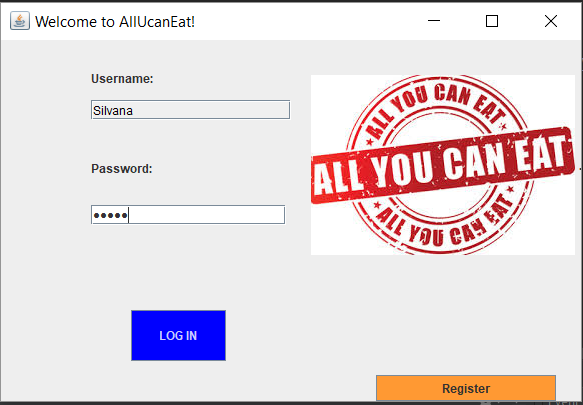
Baza mea de date are 4 tabele.

Intre tabela “product” si “cart” este o relatie 1:n, deoarece intr-un cos pot exista mai multe produse.

Intre tabela “cart” si “order” este o relatie 1:n pentru ca intr-o comanda pot sa apara mai multe produse adaugate in cos.

Intre tabela “order” si “user” exista o relatie 1:n, pentru ca un user poate avea mai multe comenzi.

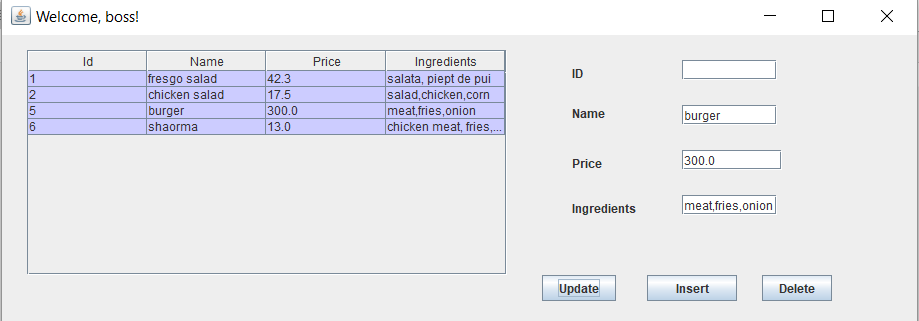
# 7. Presentation

La rularea aplicatiei se deschide fereasta de autentificare. Tot de aici, prin apasarea butonului “register”, un client nou isi poate crea cont. Aplicatia are un singur administrator [*username=Silvana, parola=admin*]. Daca aceste date sunt introduce, se deschide fereasta specifica administratorului. 

Administratorul poate gestiona produsele si, partial, conturile clientilor.

Pentru produse:

-**butonul UPDATE**, modifica datele pe care le stim ale unui produs, cum ar fi pretul sau ingredientele. Se introduce numele produsului ales, apoi noul pret si ingredientele.



-**butonul INSERT** introduce un produs nou in baza de date. Este necesara completarea field-urilor: name, price si ingredients.

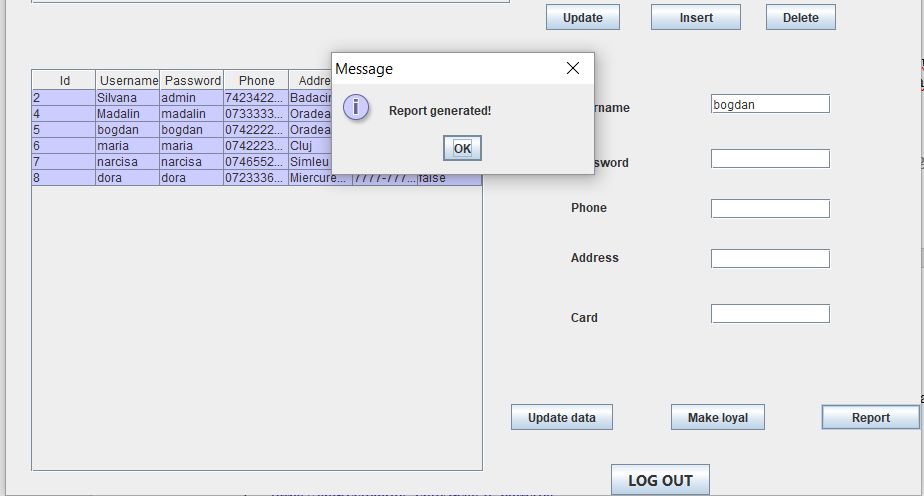
-**butonul DELETE** sterge un produs din baza de date. Se introduce doar ID-ul corespunzator.

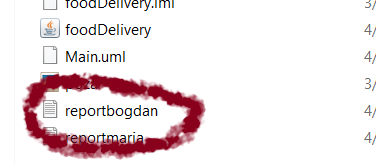
Pentru clienti:

-**butonul UPDATE DATA** modifica datele clientilor. Aceasta modificare se face dupa username, care este unic.

-pentru a face un client loial, se introduce username-ul sau si se selecteaza **butonul MAKE LOYAL**. Acum, clientul va beneficia de 5% discount la fiecare comanda.

-**butonul REPORT**, va crea un fisier text la selectarea lui, unde se va vedea ce comenzi a efectuat clientul, al carui unsername se introduce.





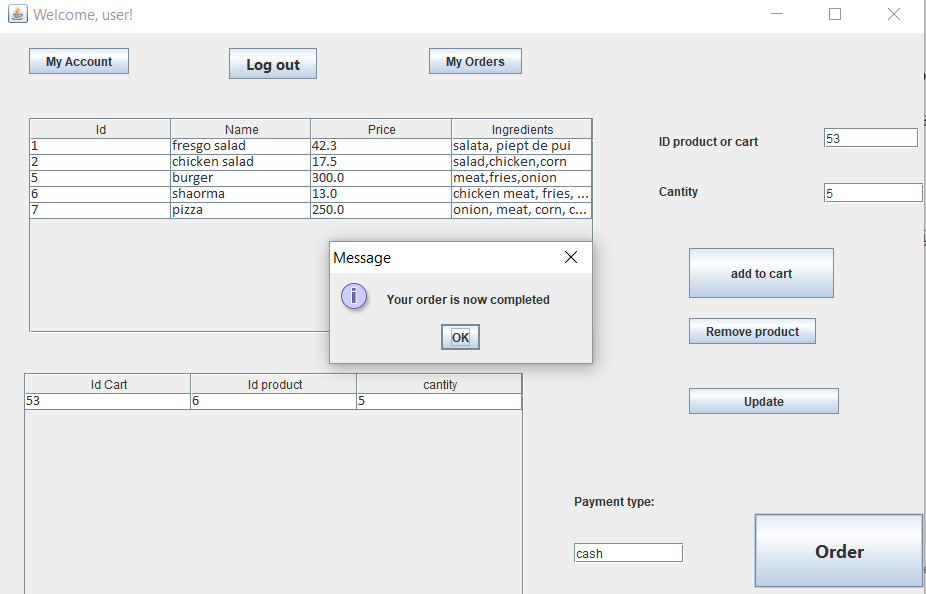
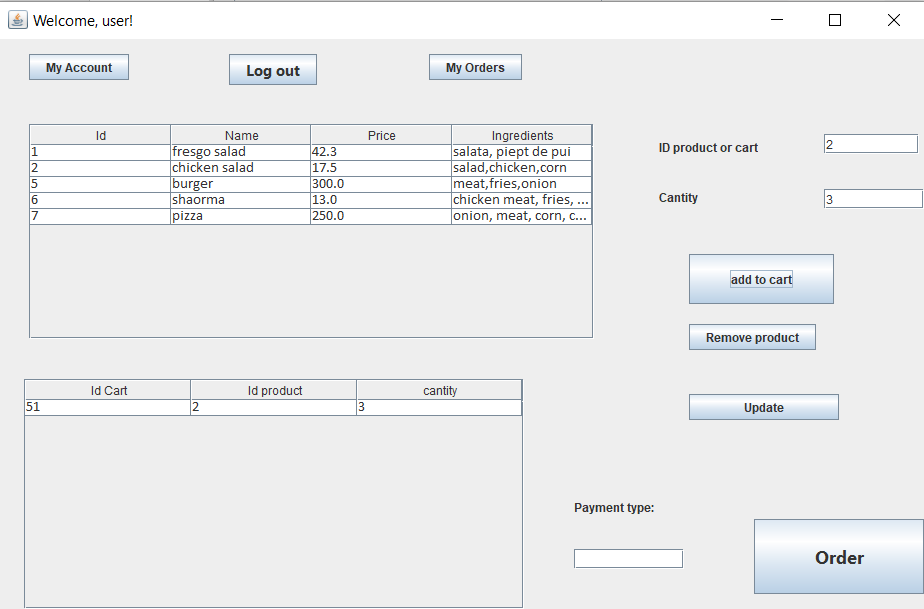
Pentru a ne intoarce la pagina principala, administratorul trebuie sa se delogheze, selectand **butonul LOG OUT**

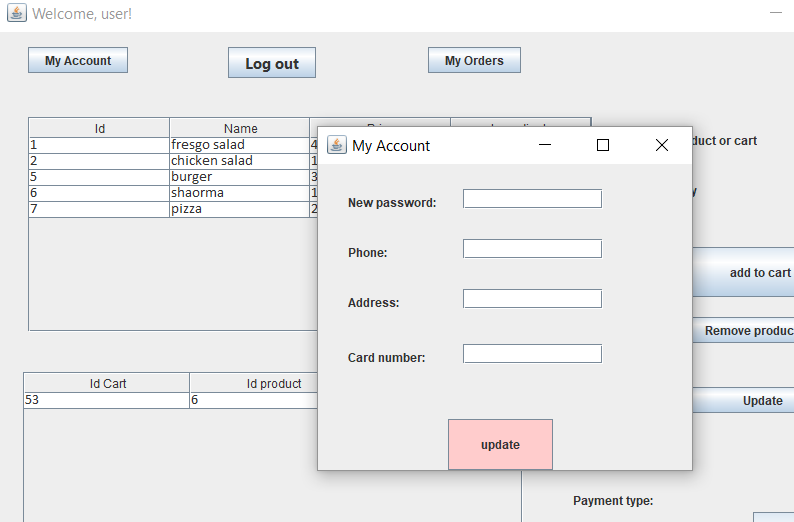
Daca ne logam ca si user, vom avea alte optiuni. In primul rand putem vizualiza produsele disponibile si adauga in cosul nostru, produse, impreuna cu cantitatea dorita. Se introduce id-ul produsului dorit, cantitatea si se selecteaza **butonul ADD TO CART**.

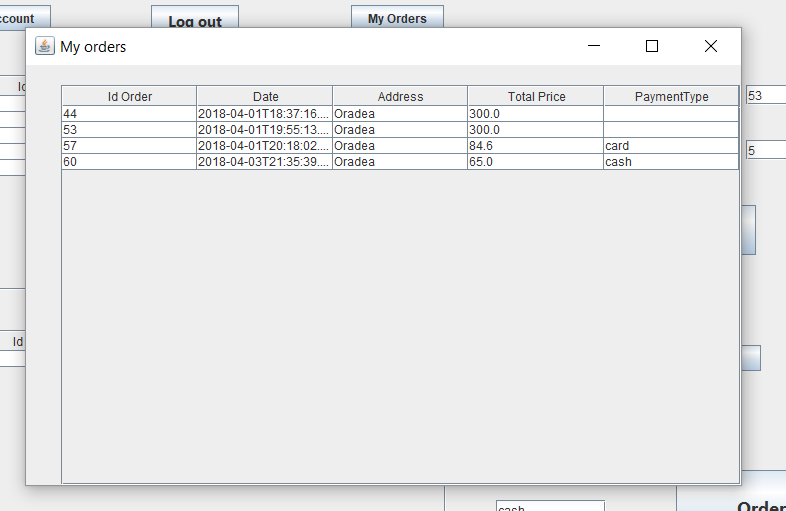
Pentru a putea sterge un produs din cos (**REMOVE PRODUCT**) sau a schimba cantitatea pe care am ales-o initial (**UPDATE**), introducem **ID-ul cosului**.

Dupa ce am introdus produsele dorite in cos, trebuie sa introducem modul de plata: CASH sau CARD, apoi sa selectam **butonul ORDER**. Acum, comanda a fost plasata .

In sectiunea **MY ACCOUNT**, ne putem modifica datele personale, iar **butonul MY ORDERS**, ne permite sa vizualizam istoricul comenzilor noastre.







# 8.Bibliography

1. <https://sourcemaking.com/design_patterns>
2. <https://stackoverflow.com/questions/11496700/how-to-use-printwriter-and-file-classes-in-java>
3. <https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html>