<Aplicatie magazin online de produse electronice>

Student: Man Silvana

**Group: 30234**

Table of Contents

[1. Requirements Analysis 3](#_Toc512511305)

[1.1 Assignment Specification 3](#_Toc512511306)

[1.2 Functional Requirements 3](#_Toc512511307)

[1.3 Non-functional Requirements 3](#_Toc512511308)

[2. Use-Case Model 4](#_Toc512511309)

[3. System Architectural Design 4](#_Toc512511310)

[3.1 Architectural Pattern Description 4](#_Toc512511311)

[3.2 Diagrams 5](#_Toc512511312)

[4. UML Sequence Diagrams 6](#_Toc512511313)

[5. Class Design 6](#_Toc512511314)

[5.1 Design Patterns Description 6](#_Toc512511315)

[5.2 UML Class Diagram 7](#_Toc512511316)

[6. Data Model 8](#_Toc512511317)

[7. Bibliography 8](#_Toc512511318)

# 1. Requirements Analysis

## Assignment Specification

Se va crea o aplicatie desktop implementata in limbajul de programare Java, care sa imite functionalitatea unui magazin online de produse electronice. Vor exista doua tipuri de utilizatori: administrator si user (client). Ambii sunt nevoiti sa se autentifice pentru a folosi aplicatia. Magazinul online ofera posibilitatea de a comanda produse apartinand a doua categorii: laptopuri si telefoane mobile. Totodata se poate alege metoda de plata: cash sau cu cardul.

## Functional Requirements

La deschiderea aplicatiei, cele doua tipuri de utilizatori au posibilitatea de a se loga, dar si noii vizitatori ai aplicatiei isi pot crea un cont.

Administratorii vor putea sa gestioneze produsele din magazine, cu ajutorul operatiilor CRUD, sa adauge specificatii produselor si sa faca un utilizator loial. Primind acest atribut el va avea parte de un discount de 5% la fiecare comanda plasata. Din motive de securitate, clientii au parola criptata pentru a nu fi vizibila de catre administrator.

Userii vor putea sa vada lista de produse, specificatiile lor. Va putea compara doua produse intre ele, pentru a fi mai usoara alegerea si va putea vedea istoricul comenzilor sale. Evident, ei pot adauga in cosul lor de cumparaturi produse si vor ptea plasa o comanda, alegand metoda de plata: cash sau cu cardul. La plasarea unei comenzi, administratorul va fi notificat si se va genera o facture in format text.

- Datele vor fi salvate intr-o baza de date, generata utilizand bibblioteca ORM, Hibernate.

- Se va folosi pattern-ul arhitectural MVC la proiectarea si organizarea aplicatiei.

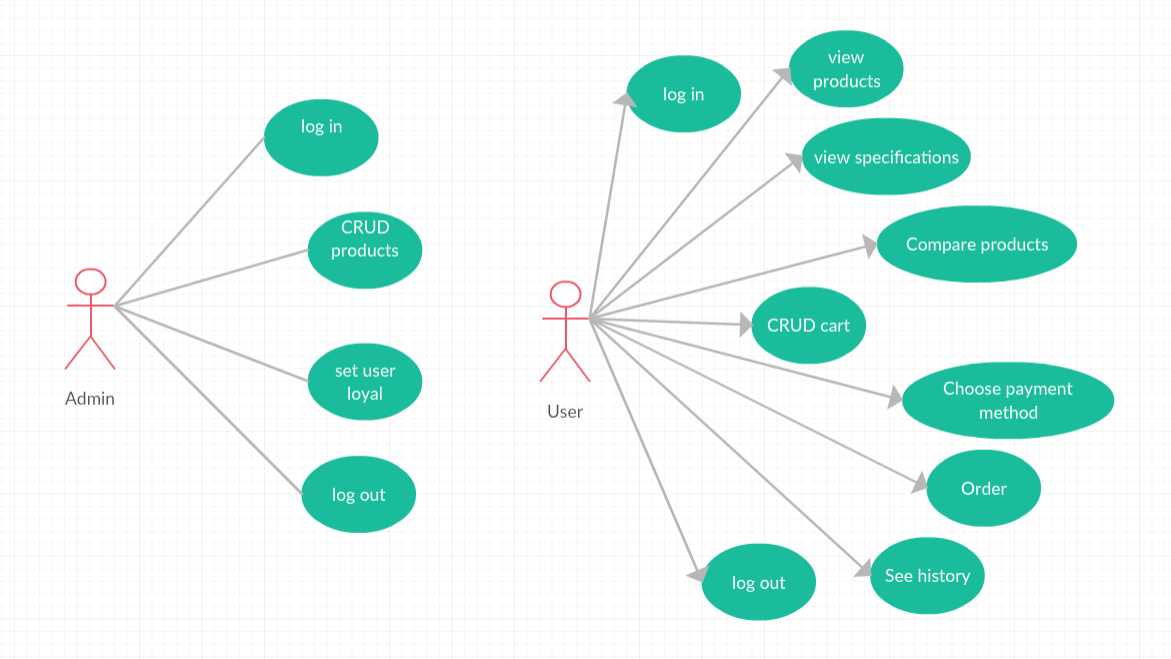
- Parolele vor fi salvate criptat in baza de date

## Non-functional Requirements

Disponibilitate: aceasta aplicatie desktop poate rula pe orice sistem modern ce dispune de un minim de resurse.

Extensibilitate: aplicatia poate fi extinsa cu alte functionalitati cum ar fi o descriere mai ampla a produselor (inclusiv poze, rating), prezenta mai multor date pentru casieri (inclusiv poza de identificare), trimiterea unui email de confirmare etc.

# Use-Case Model

**

*Use case: order product*

*Primary actor: user*

*Main success scenario:*

* Introducere username si parola
* Validate username si parola
* Log in
* Se deschide fereastra in care se pot vedea produsele disponibile
* Se introduce id-ul produsului ales si cantitatea
* Se adauga in cosul de cumparaturi
* Se selecteaza metoda de plata
* Se plaseaza comanda

*Extensions:* daca validarea esuaza se arunca o exceptie

# System Architectural Design

## Architectural Pattern Description

Pentru realizarea aplicației am folosit arhitectura MVC (Model – View - Controller). Ideea este separarea interfeței utilizator într-un Model, un View (vedere, vizualizare) (creează afişajul, interacționând cu Modelul după nevoi), şi un Controller (răspunde la cererile utilizatorului, interacționând atât cu Vizualizarea cât şi cu Controlorul).

Arhitectură:

• **Model** -Această parte a controlatorului manipulează operațiunile logice și de utilizare a informației (trimisă dinainte de către rangul său superior), în cazul acestei probleme modelul este reprezentat de clasele care vor crea baza de date- Admin, User, Product, Cart,OrderP dar si clasele care fac operatii pe baza de date- AdminOp, UserOp, ProductOp, CartOp, OrderOp.

• **View**- Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controller. Interfața a fost realizată cu ajutorul bibliotecii Swing. Din View in cazul nostru fac parte clasele: Log, AdminView si UserView, SpecificationView, MyOrders, RegisterView care definesc ferestrele programului.

• **Controller** - Cu acest element putem controla accesul la aplicația noastră. În cazul aplicației de față avem 3 clase care controleaza aplicatia, cate una pentru fiecare window: ControllerInit (partea de log in), ControllerAdmin (controleaza fereastra specifica administratorului), Controlleruser (controleaza fereastra folosita de user).

## Diagrams

Diagrama de pachete:

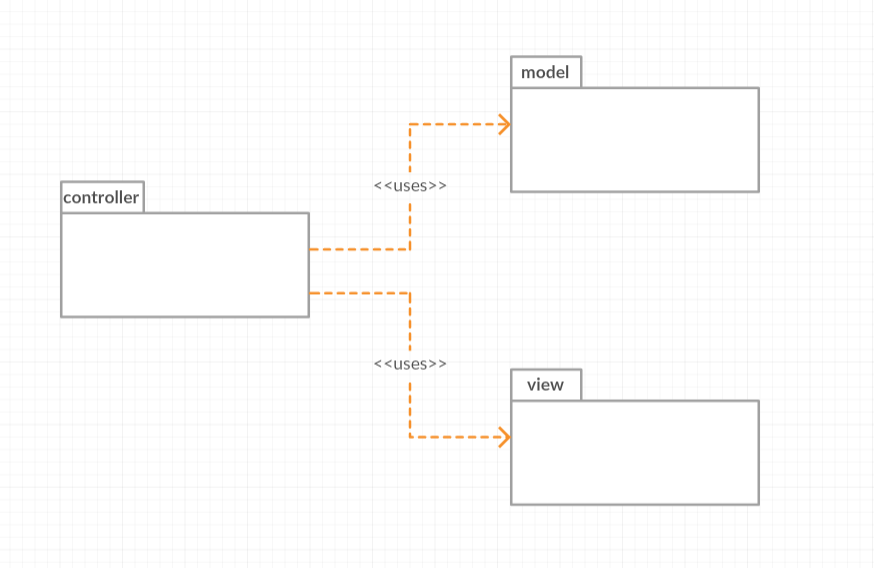
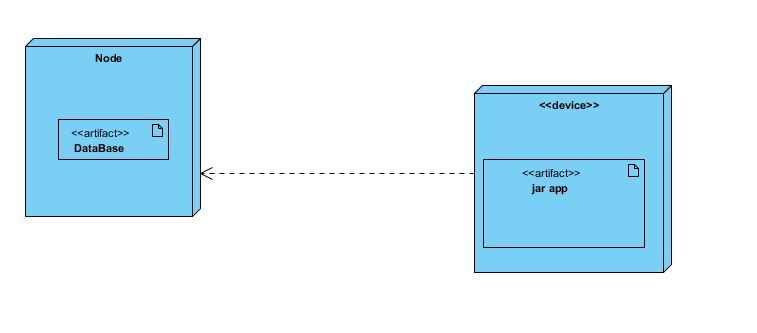
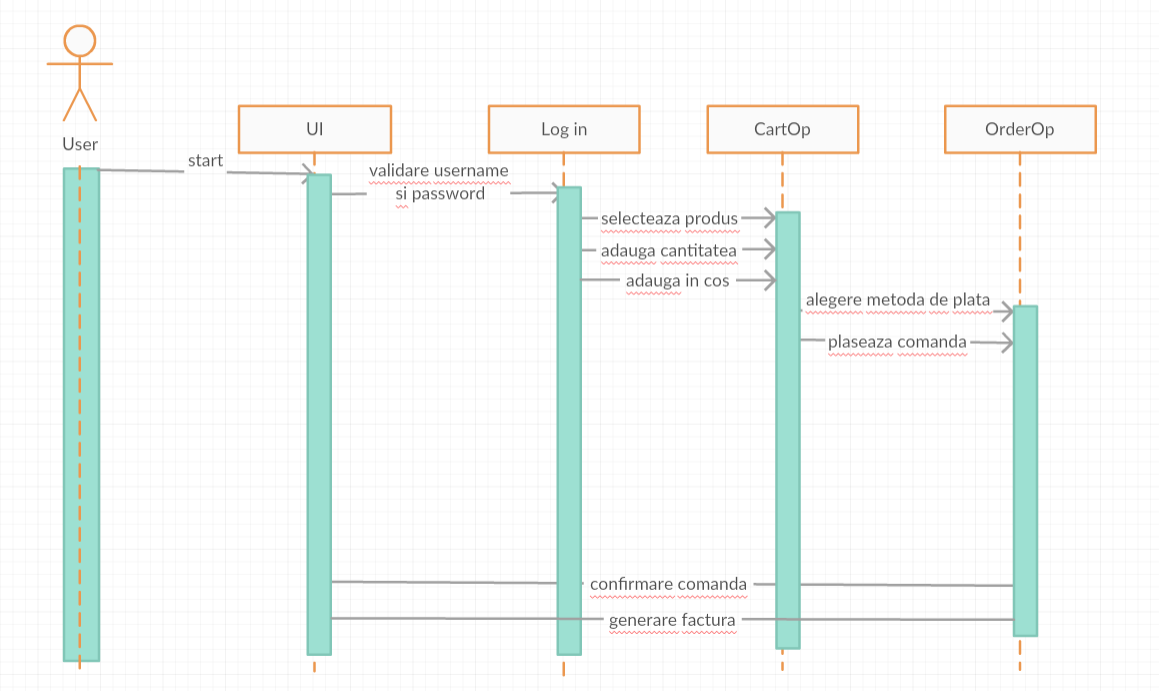
**

Diagrama de deployment:



# UML Sequence Diagrams

Diagrama de secventa pentru comandarea unui produs:



# Class Design

## Design Patterns Description

Design Pattern-ul *Observer* defineste o relatie de dependenta 1..\* intre obiecte astfel incat cand un obiect isi schimba starea, toti dependentii lui sunt notificati si actualizati automat. Implica existenta unui obiect denumit *subiect* care are asociata o lista de obiecte dependente, numite *observatori*, pe care le apeleaza automat de fiecare data cand se intampla o actiune.

Acest pattern este de tip **Behavioral** (comportamental), deorece faciliteaza mai buna organizare a comunicatiei dintre clase in functie de rolurile/comportamentul acestora.

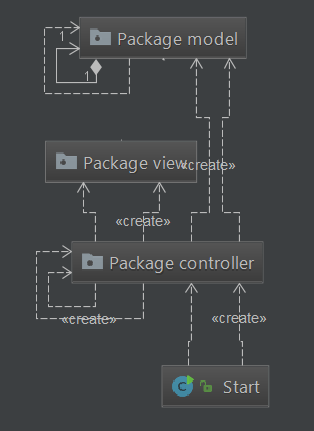
Design pattern-ul Observer se foloseste in cazul in care mai multe clase(*observatori*) depind de comportamentul unei alte clase(*subiect*), in situatii de tipul:

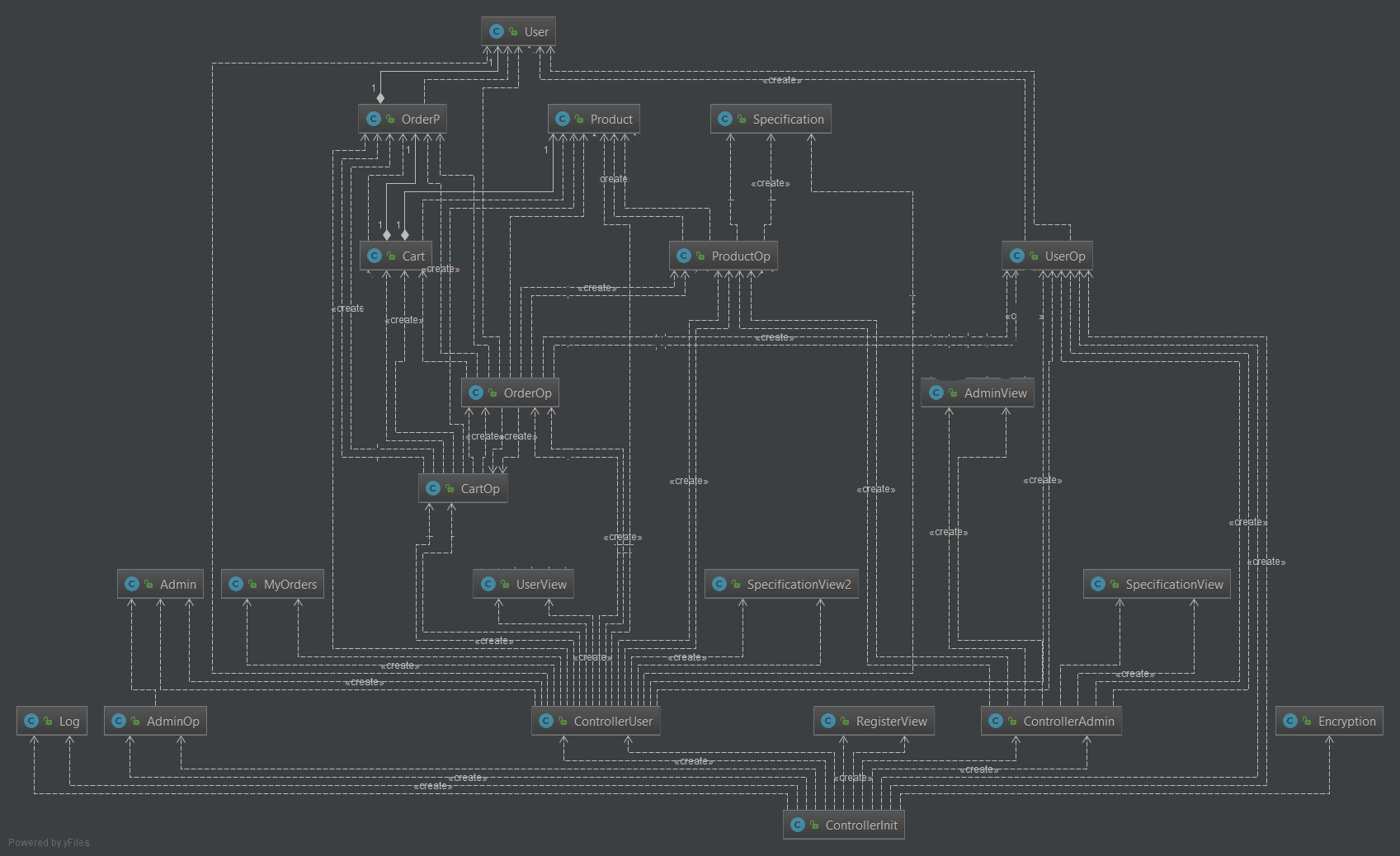
* o clasa implementeaza/reprezinta logica, componenta de baza, iar alte clase doar folosesc rezultate ale acesteia (monitorizare).
* o clasa efectueaza actiuni care apoi pot fi reprezentate in mai multe feluri de catre alte clase.

Am folosit acest design pattern pentru a notifica administratorul, in momemntul in care se plaseaza o comanda.

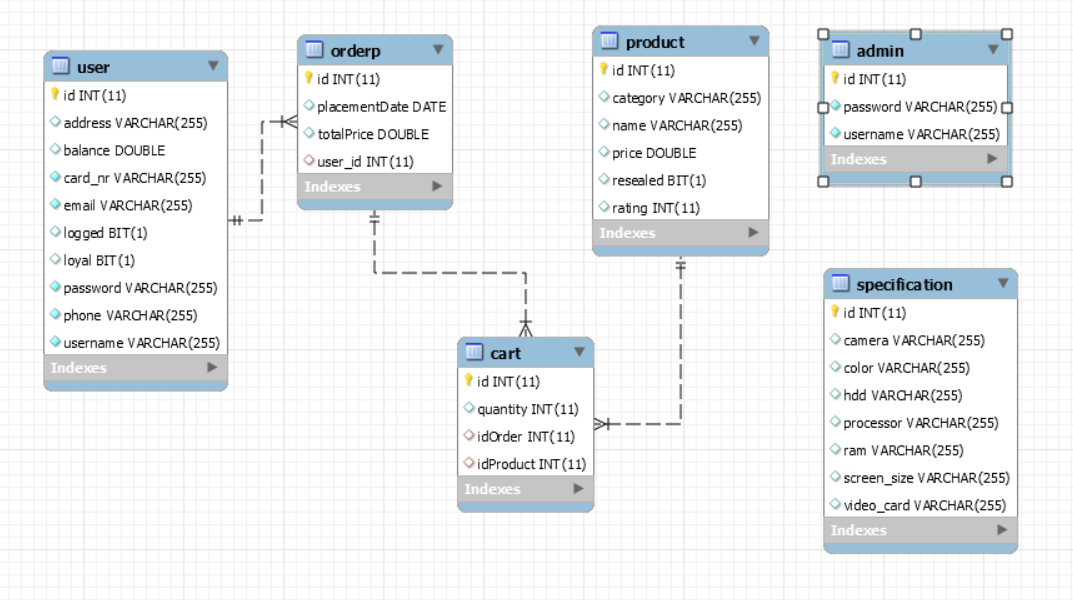
Am folosit si design pattern-ul Singleton pentru a crea o singura instanta a conexiunii cu baza de date. Aceasta se face prin crearea unui obiect Session factory.

## UML Class Diagram





# Data Model

**

# Bibliography

* <https://www.roseindia.net/hibernate/hibernate4/spring_hibernate_encrypted_password.shtml>
* <http://cursuri.cs.pub.ro/~poo/wiki/index.php/Design_Patterns_Basics>
* <https://stackoverflow.com/questions/6592010/encrypt-and-decrypt-a-password-in-java>