**Universitatea Tehnica din Cluj-Napoca**

Calculatoare, anul 2

**LUCRAREA 2**

**ORDER MANAGEMENT SYSTEM**

**Bogdan Stupariu, grupa 30425**

**1. Introducere**

**1.1. Obiectivul lucrării**

Lucrarea urmărește implementarea unui sistem de administrare a unui magazin. Pentru aceasta este necesară îndeplinirea a două obiective. Primul dintre ele, legat de mecanismele din spate, presupune creearea unei baze de date în interiorul programului care să memoreze și să administreze informațiile. Al doilea obiectiv, este implementarea unei interfețe duale, una pentru administrare a bazei de date, iar cealaltă pentru procesarea comenzilor obișnuite într-un magazin. Obiectivul general al lucrării este dezvoltarea aptitudinilor de programare în contextul lucrării cu date.

**1.2. Abordarea lucrării**

Această lucrare prezintă o metodă de rezolvare a problemei enunțate, incluzând o serie de acțiuni ce pot fi executate atât de către utilizator cât și de către administrator. Problema a fost abordată din în special astfel încât, produsul final să fie unul atractiv pentru utilizator.

**2. Descrierea problemei**

**2.1. Analiza problemei**

Cerința inițială pentru acest proiect a dezvăluit structura programului. Fiind oferite clasele esențiale, dificultatea analizării problemei la nivel conceptual, al design-ului de clase, a fost redusă, însă a crescut dificultatea implementarii per se.

Pe langa clasele necesare ce trebuie să fie conținute de către proiect, cerința a mai edificat și metoda de implementare a memorării de date, respectiv folosind Binery Search Trees. Doua clase erau folosite ca structuri de date, Warehouse-ul, a cărui scop este acela de a administra si memora toate produsele ce se afla în magazin, respectiv OPDept (Order Processing Department), a cărei scop este acela de a memora și administra toate comenzile făcute de oricare client.

Proiectul va avea nevoie de o interfață dinamică, ce să își actualizeze opțiunile în funție de persoana care folosește programul. Pentru aceasta am ales să separ implementarea pentru administrator de cea pentru un utilizator ubișnuit. Cea mai bună soluție pentru aceasta pe care am gasit-o a fost folosirea CardLayout, iar interfața de start am ales să o fac sub forma unei platforme de logare și inregistrare.

Problema a cerut ca pentru fiecare scenariu, iar când vorbim de scenarii, în acest caz, vorbim de utilizare ca administrator sau utilizare ca utilizator, să existe un set minim de operații. Administratorul trebuie să aibă posibilitatea de a adauga produse, elimina produse și de a actualiza produse. Utilizatorul, odată logat, trebuie să poată cauta produse dintre cele existente, să poată selecta produsul pe care dorește să îl comande, să stabilească cantitatea și, mai departe, să poată plasa comanda. De asemenea, utilizatorul trebuie, conform cerinței, să poată vedea produsele comandate până la momentul actual.

**2.2. Modelare**

Odata analizată problema, următorul pas din proiect este modelarea, conform principiilor de OOP. Definirea claselor nu a reprezentat o problema, structura fiind una clara, oferită de cerință. Proiectul are nevoie de 4 categorii principale: una, categoric, pentru funționaitate, pentru clasa main. O a doua categorie este interfața grafică, modul în care proiectul se arată utilizatorului. O altă categorie este cea ce conține clasele oferite prin cerință, clasele ce implementează funcționalitatea magazinului și chiar a programlului în sine. O ultima categorie ce este necesară este cea ce definește și administrarea utilizatorii care folosesc programul. La nivel de structuri de date, pe lânga folosirea Binary Search Tree, necesară pentru implementarea administrării, am ales folosirea implementarea grafică cu ajutorul Jtables. Necesitatea respectării principiilor OOP a făcut mult mai ușoară alegerea structurilor de date și a metodelor de lucru.

**2.3. Scenarii**

Scenariile de utilizare ale proiectului sunt bazate pe inputul celui ce folosește programul. Ceea ce s-a dorit, practic, a fost ca pentru fiecare user, în parte, să existe un program personalizat, ce să afișeze pe interfață doar aspectele ce țin de utilizatorul respectiv.

În cadrul interfeței se regăsește opțiunea de logare. În cazul în care utilizatorul este administrator al sistemului și dorește să se înregistreze cu privilegiile de administrator, acesta se va loga cu unul din conturile ce dețin privilegiu de acces la administrare. Conturile de administrator sunt definite din interiorul programului și nu din interfață. Astfel încât să nu existe posibilitatea ca cineva să intre în administrarea programului fără să aibă dreptul. Din motive de simplitate, am ales ca pentru administrator să existe un cont unic, cu numele de amin.

În cazul în care admin-ul se loghează, interfața se va schimba, iar programul va afișa cadrul prin care administratorul poate face schimbările. Acesta poate adauga produse unice sau în cantitate mai mare, poate selecta produse pentru a le șterge din depozit sau poate selecta produse pentru a le actualiza numele, categorie, prețul sau cantitatea. Odată finalizate operațiunile, administratorul se poatw intoarce la interfața de logare, acolo unde poate utiliza programul ca client administrator, sau se poate loga sub formă de utilizator cu un alt cont.

Pentru un utilizator simplu, pașii ce trebuie urmați sunt următorii: în primul rând, utilizatorul trebuie să își creeze un cont pentru utilizarea programului. Odata creat contul sau în cazul în care utilizatorul și-a creat un cont într-o sesiune de utilizare anterioară, utilizatorul se poate loga prin introducerea numelui și a parolei. Logarea va oferi un mesaj de Loggin succes sau de Login failed în funție de existența sau nu a contului, respectiv de introducerea corectă sau nu a parolei.

În cazul în care utlizatorul este logat cu succes, acesta poate intra în magazin prin apăsarea butonului „search for products”. Odată apăsat acest buton, interfața se schimbă, iar pe ecran sunt afișate urmatoarele: un tabel cu toate produsele aflate în stoc, un tabel cu toate comenziile anterioare făcute de catre utilizatorul respectiv, opțiuni de plasare a comenzii și un scurt rezumat al acțiunilor făcute până acum de către utilizator. Acesta de asemenea poate căuta în tabeul cu produsele aflate în stop un anumit produs după nume, o întreagă categorie (să spunem ca, într-o bază de date se află fructe, legume, peruci și tablete. Utilizatorul poate căuta doata tablete, iar în acel caz doar tabletele vor fi afișate în tabel), un preț sau chiar o cantitate în tabel. Plasarea comenzii se face prin selectarea unui produs din tabel, selectarea cantității dorite și apoi prin apăsare butonului ”Place order”. Odată acceptată comanda, aceasta va apărea în tabelul cu comenzi.

**2.4. Cazuri de utilizare**

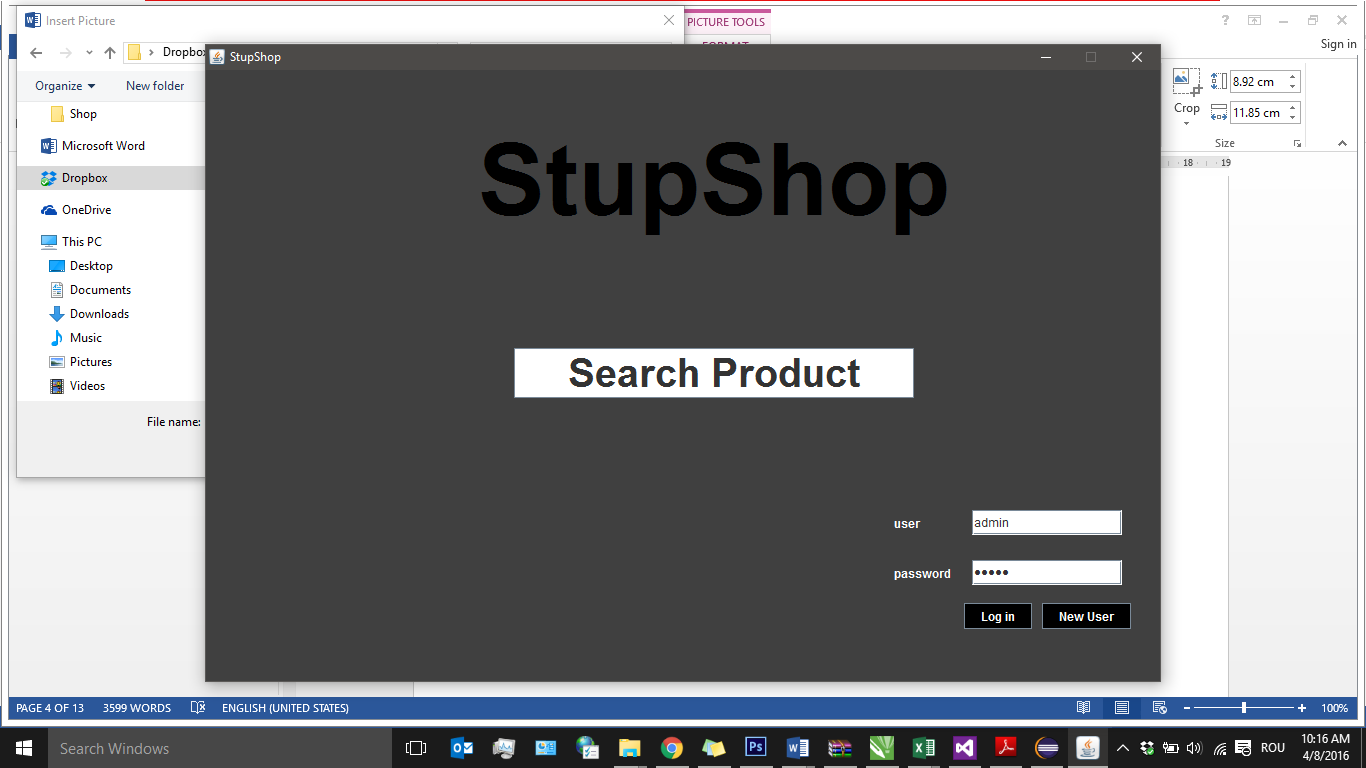
La acest capitol ne referim la modul în care relaționează user-ul cu programul. Pentru a îmi da seama cum să implementez această relație, am stabilit următoarele obiective:

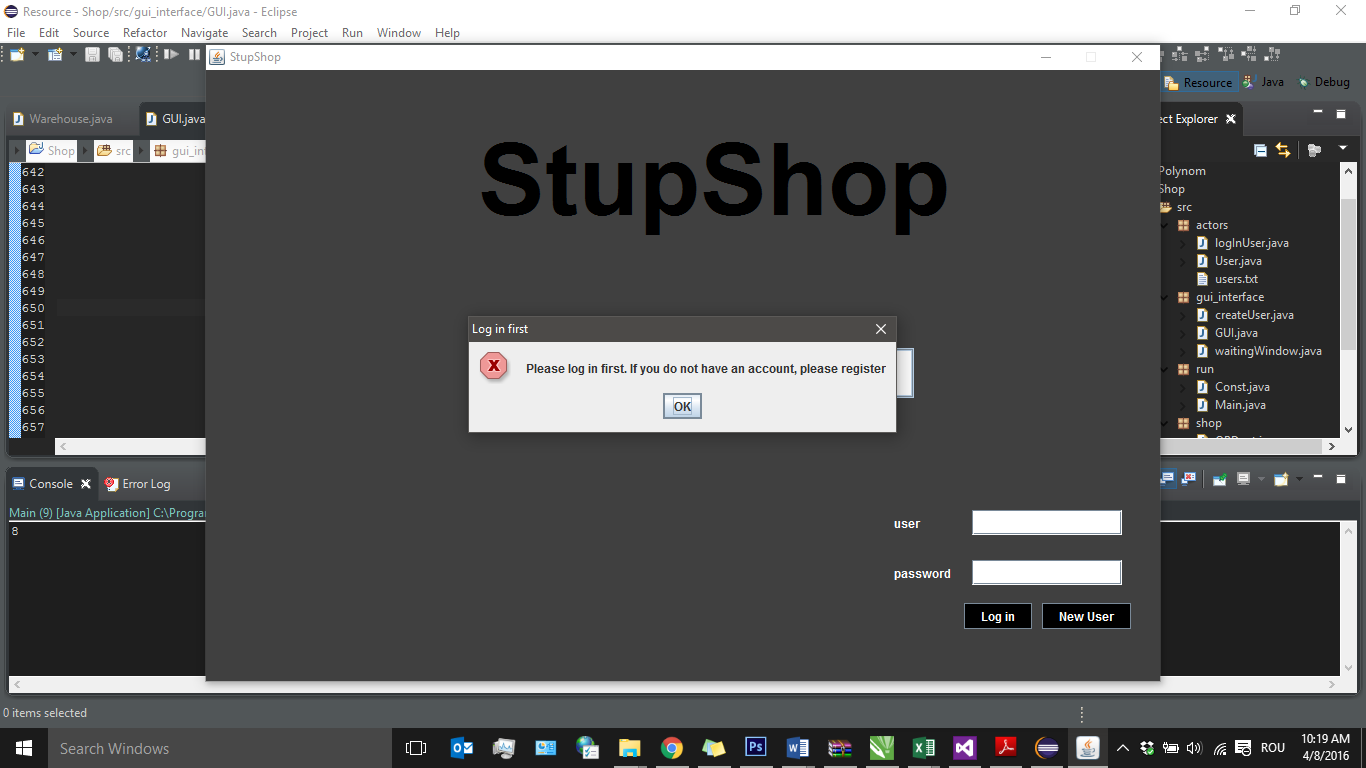
a) Interfața trebuie să fie operațională

b) interfața trebuie să fie contruită pe principiul „user-friendly”

c) interfața trebuie să conțină informații referitoare la modul în care user-ul trebuie să interacționeze

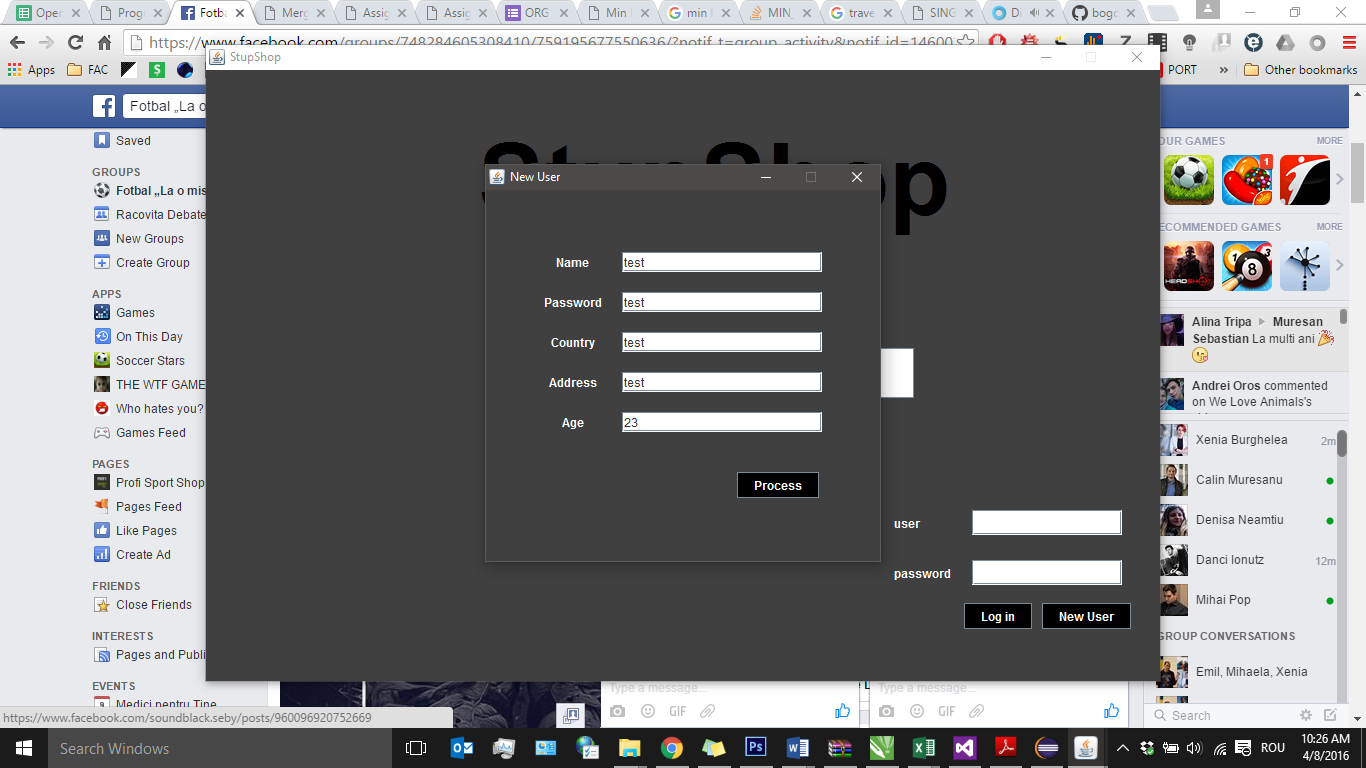
d) interfața trebuie să prezinte un aspect plăcut.

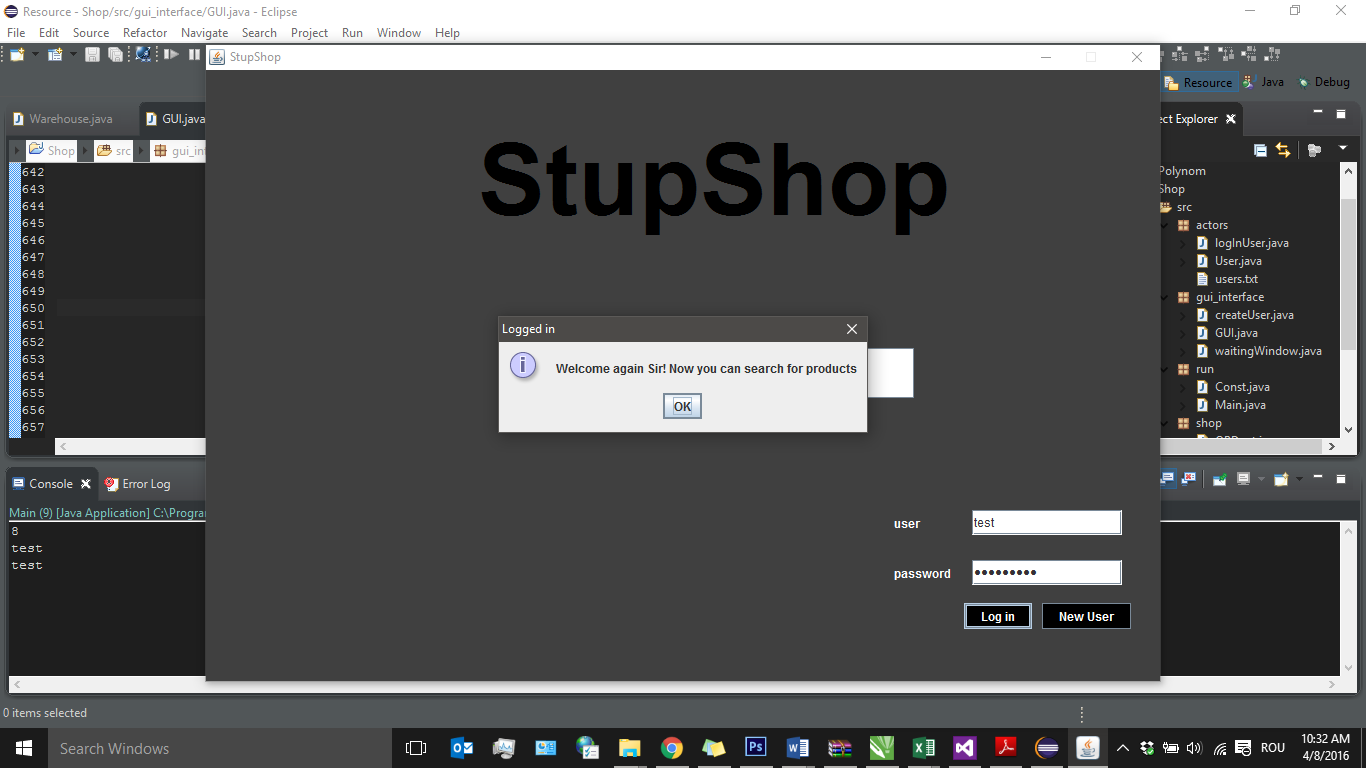
2.4.1. Interfața de logare.

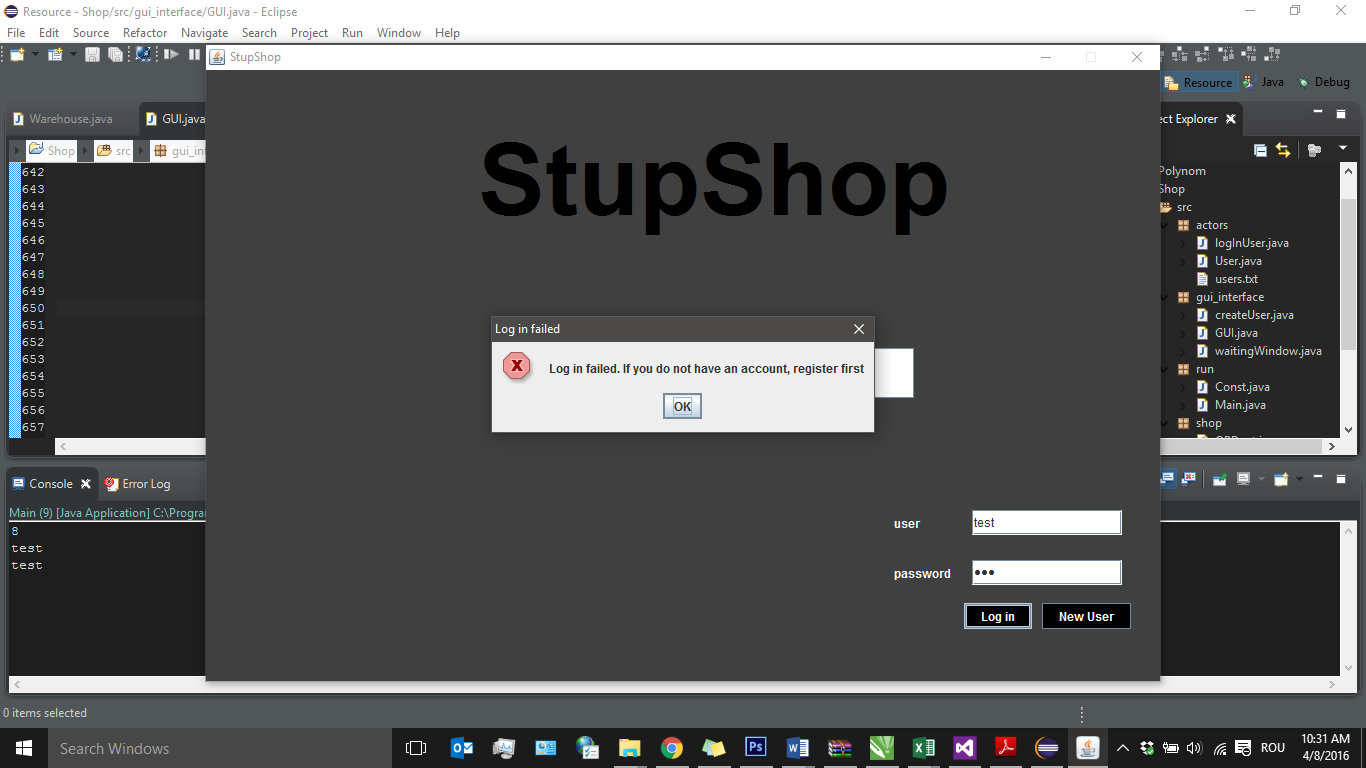
Prima interfață care va fi afișată utilizatorului, practic punctul de start al programului, este interfața de logare. În cadrul acesteia, utilizatorul trebuie să se logheze. În cazul în care utilizatorul încearcă logarea intrarea în programul de plasare de comenzi fără a fi logat, acesta va primi un mesaj de eroare prin care va fi informat că este necesară logarea pentru a putea folosi programul, precum în figura urmatoare:

Imagine 1. Interfața grafica - GUI

În acel moment, utilizatorul își va aminti că trebuie să se loghze cu contul existent sau, daca nu a mai folosit programul, va accesa opțiune New User și își va crea un cont nou. Interfața pentru crearea contului este, practic, un Jframe nou cu JtextField-uri, a căror inputul vor crea o nouă instanță a clasei User. Fiecare intanță este salvată intr-un fișier, iar la fiecare nouă logare, ceea ce se întamplă, este faptul că programul caută acea instanță în fișier.

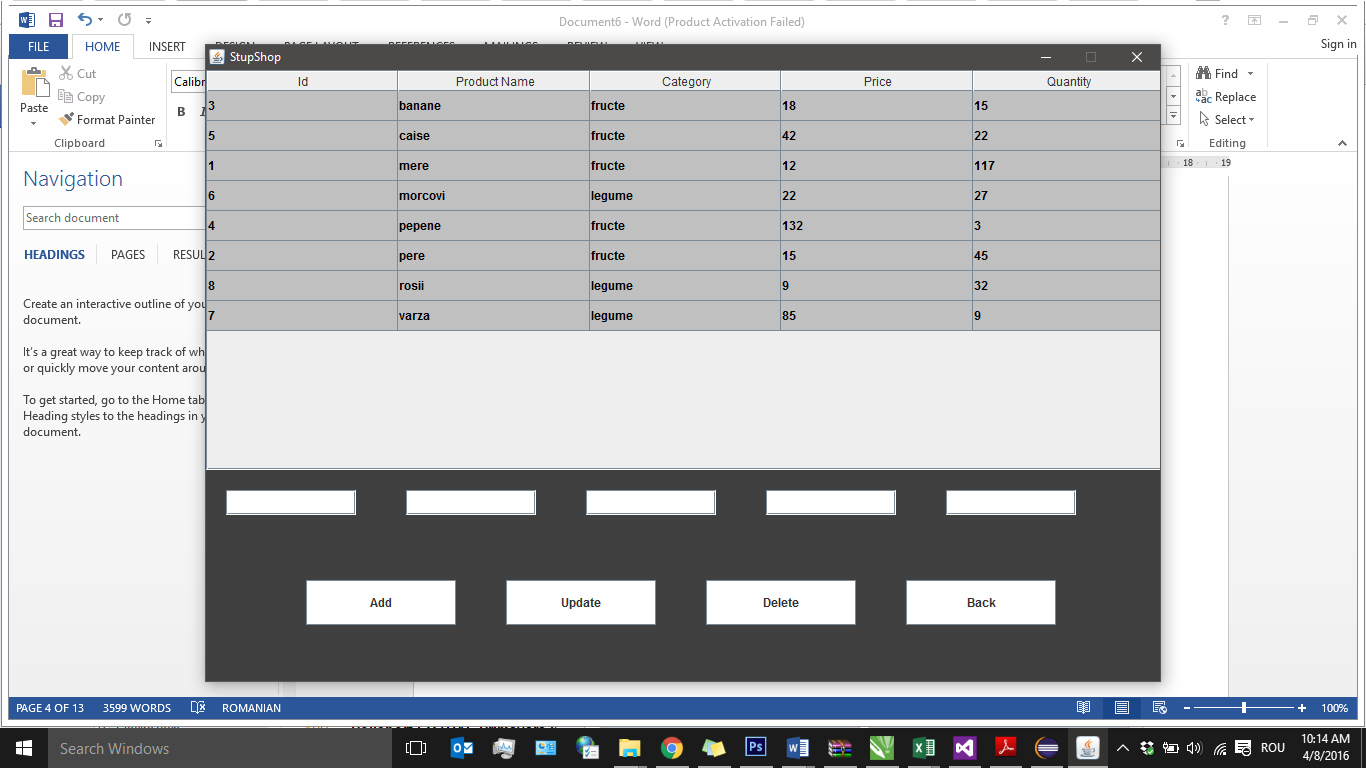
Pentru a crea un nou cont am ales să fie nevoie de introducerea unor informații pe care le-am considerat de bază pentru un cont avut pe o platformă de magazin online.

Odata creat un cont nou, utilizatorul își poate folosi numele și parola pentru a se loga. Logarea se va face instant, iar în funție de corectitudinea datelor scrise, va fi afișat unul dintre mesaje: de succes sau de eșec al logării. În cazul eșecului, utilizatorul va fi nevoit să reintroducă datele, corect, iar în cazul succesului, acesta va putea intra în sistemul de comenzi al magazinului.



În cele două imagini anterioare pot fi văzute cele două cazuri. În stanga, cazul de eșec, iar în drepta, cazul de succes al logării.

2.4.2 Interfața de administrator

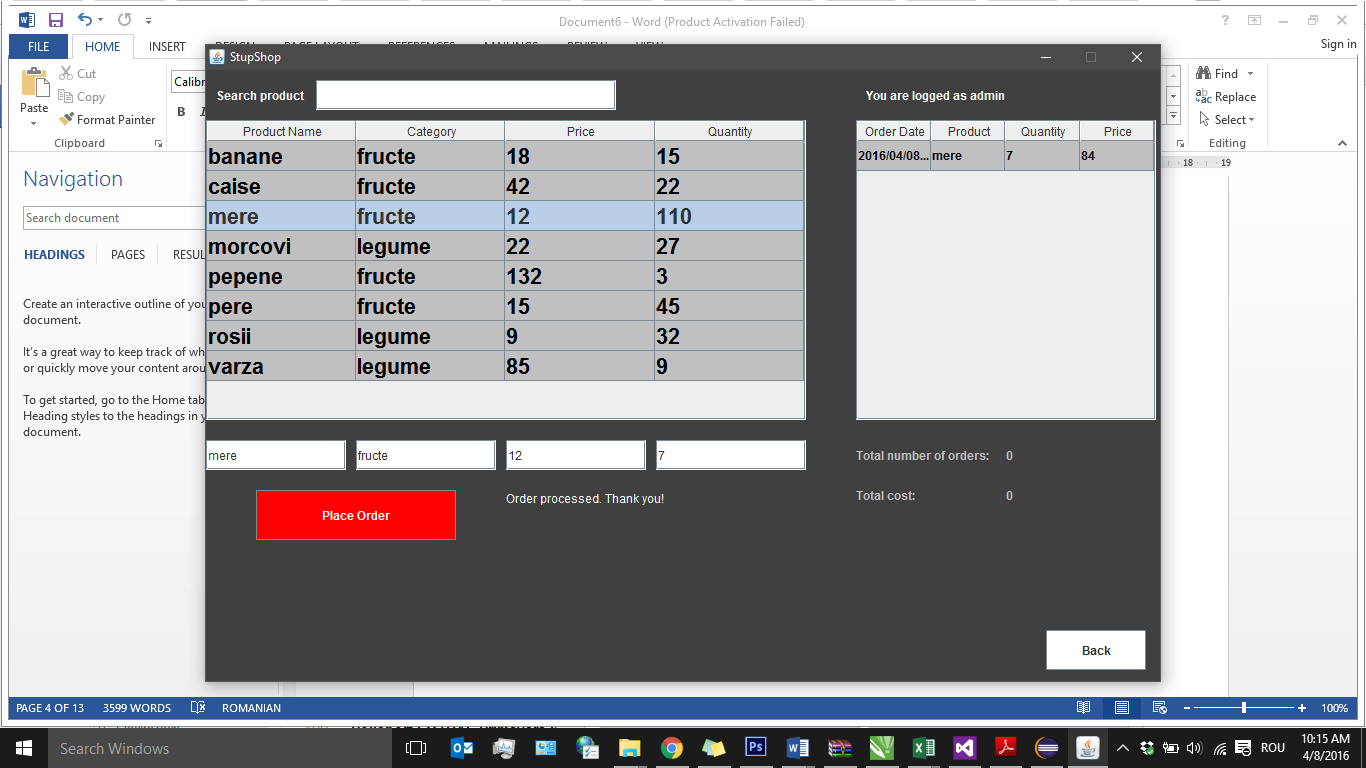
Odată logat ca admin, interfața se va schimba, iar progamul va afișa opțiunile relevante pentru un administrator, cu drepturile și privilegiile corespunzătoare. Doar din această utilizare / vizualizare pot fi modificate informațiile reținute în depozitul de produse (cu excepția cantității, acolo unde se actualizează automat după fiecare comdandă făcută ) .

Odată selectat un produs, câmpurile JtextField de sub tabel vor lua valorile rândului selectat. Prin această opțiune, practic se percepe, pentru un utilizator admin care nu știe funcționalitatea internă a programului, faptul că este încărcat un produs spre modificare/actualizare. Salvarea datelor va fi făcută prin apăsarea butonului update. Ceea ce se întâmplă de fapt, este că, prin adăugarea unui produs, este introdus în warehouse o instantă de tip Product. Warehouse-ul este practic, un binary search tree. Astfel, ceea ce se întamplă de fapt, este că prin adăugarea unui produs nou este creat un nod nou de tip product în Binary Search Tree-ul Warehouse.

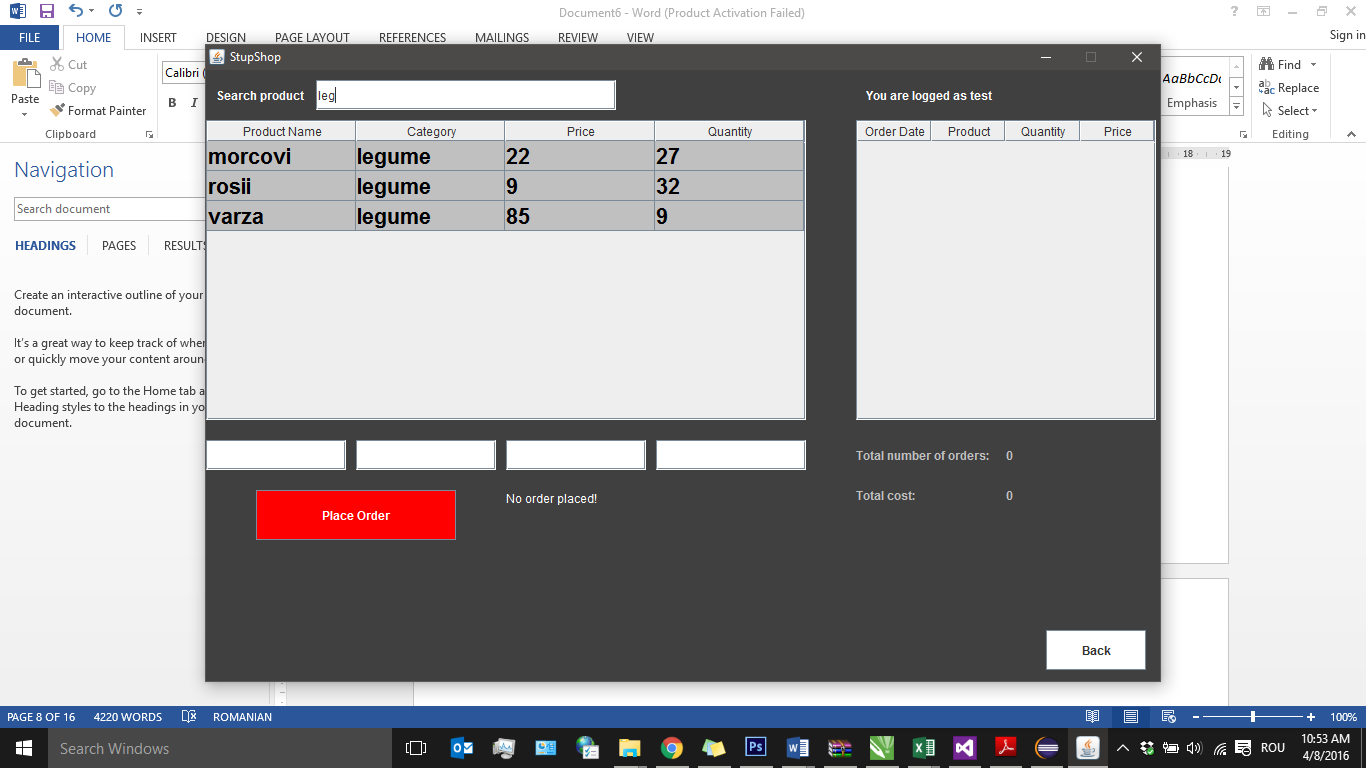
Prin selectarea unui produs, modificarea lui în JtextField și ulterior apăsarea butonului update, programul va căuta nodul de tip produs cu acel nume în arborele implementat de depozit, iar apoi va performa modificările necesare pe nodul existent. Opțiunea delete este asemănătoare, în ideea în care este căutat nodul selectat, iar apoi eliminat din arbore într-un mod convenient astfel încât structura arborelui să rămână intacta.

Apăsarea butonului back va permite administratorului să se întoarcă la interfața de logare.

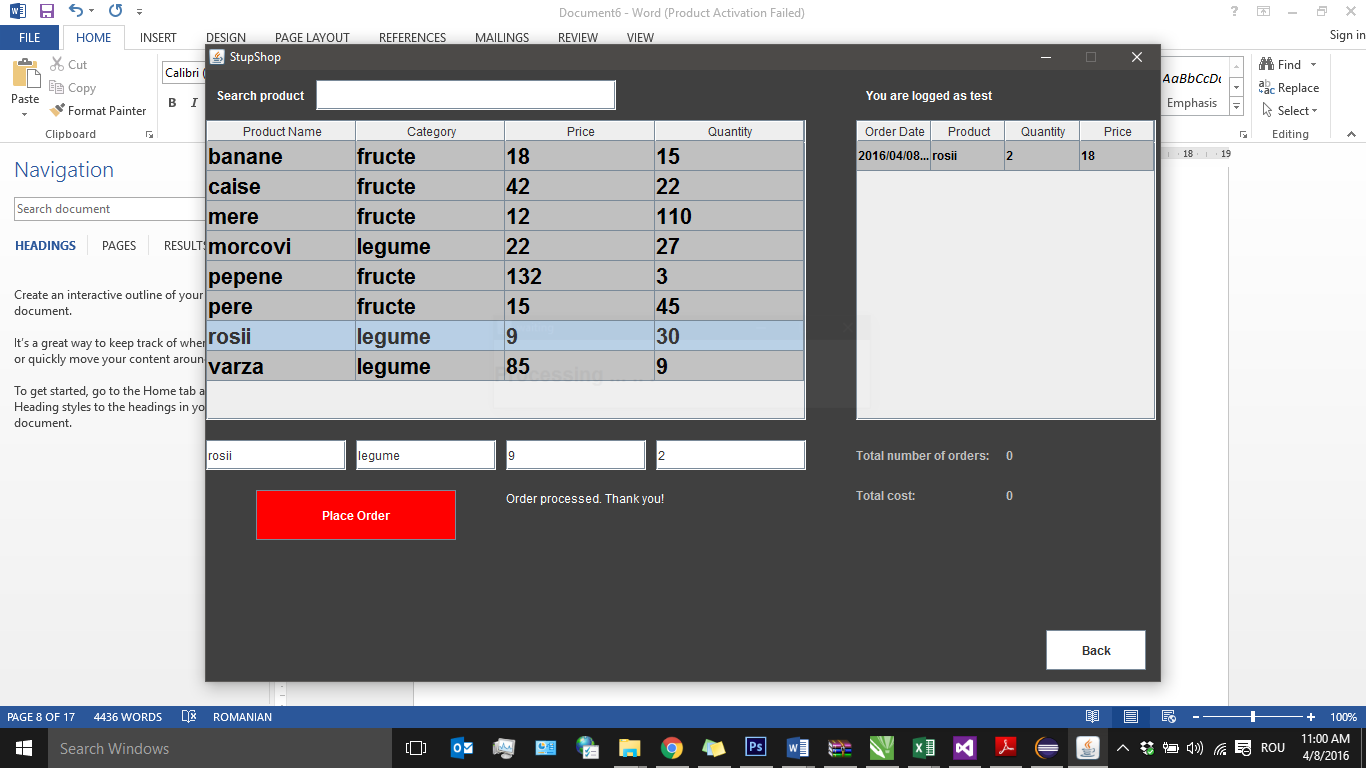
2.4.3 Interfața untilizatorului

Odată logat în sistem, utilizatorul poate accesa magazinul prin apăsarea butonului mare, din centrul ecranului, ”search for products”. Interfața se va schimba, iar astfel va apărea pe ecran o interfață pe care utilizatorul poate să o folosească.

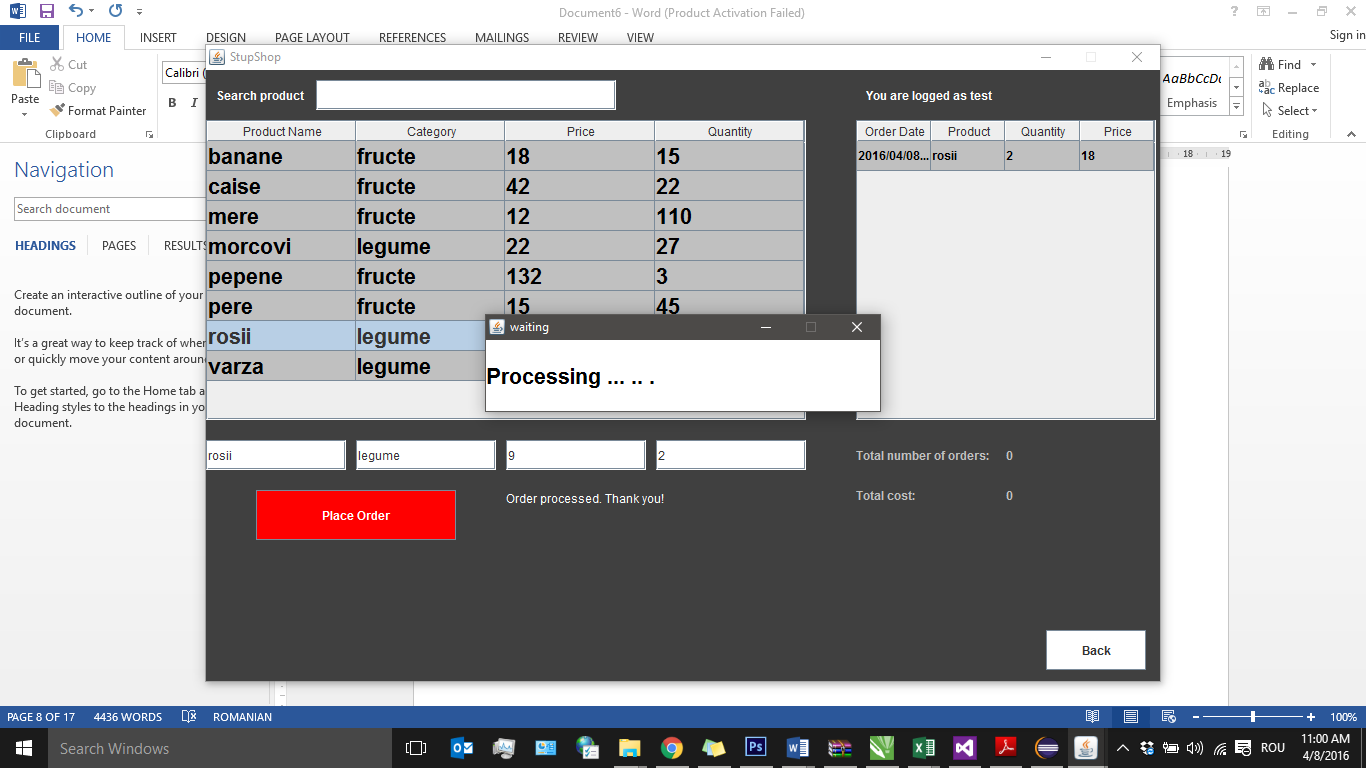
Așa cum am dezvoltat anterior, există o serie de operații și acțiuni pe care utilizatorul le poate performa în activitatea sa pe magazin. Cazurile cele mai importante de precizat, sunt exemplificarea căutării de produse și a utilității acesteia ( necesară fiind o cerință a proiectului, înă dezvoltată de către programator astfel încât să folosească cât mai mult utilizatorului ), plasarea comenzii și cazurile ce pot să apară și respectiv vizualizarea produselor deja comandate.

Opțiunea de căutare a unui produs este foarte utilă deoarece permite și căutarea după alte criterii decât cel de nume al produsului. Se observă în imaginea alăturată cum prin tastarea în JtextField doar a literelor ”leg”, sunt filtrate produsele, iar în tabel apar, acum, doar produsele din categoria legume. Practic, marele avataj, așa cum am spus, este că aceată filtrare se poate face după orice criteriu, iar citeriile pot fi multiple în același moment. Spre exemplu, dacă ar exista un produs cu numele ”legume” dar din categorie ”vegetabile”, atunci și acel prosus ar fi afișat. Consider această posibilitate una dintre reușitele acestui proiect la nivel de implementare. Un alt mare avantaj în cazul acestei utilizări, este acela al faptului că actualizare / filtrarea tabelului se produce instant, nefiind nevoie de apelarea unui buton sau a unei acțiuni pentru filtrare. Acest fapt reduce timpul de lucru și face totul mult mai accesibil și dinamic pentru utilizator.

Următorul caz de analizat, este cel al plasării comenzilor. Pentru a ilustra cât mai bine acest aspect, voi analiza fiecare pas în parte. În primul rând, este necesară selectarea unui produs și stabilirea cantității produsului respectiv ce se dorește a fi comandată. Acesta se face prin selectarea produsului, încărcarea acestuia în JtextField-urile de sub tabel, și modificarea manuală a cantității create.

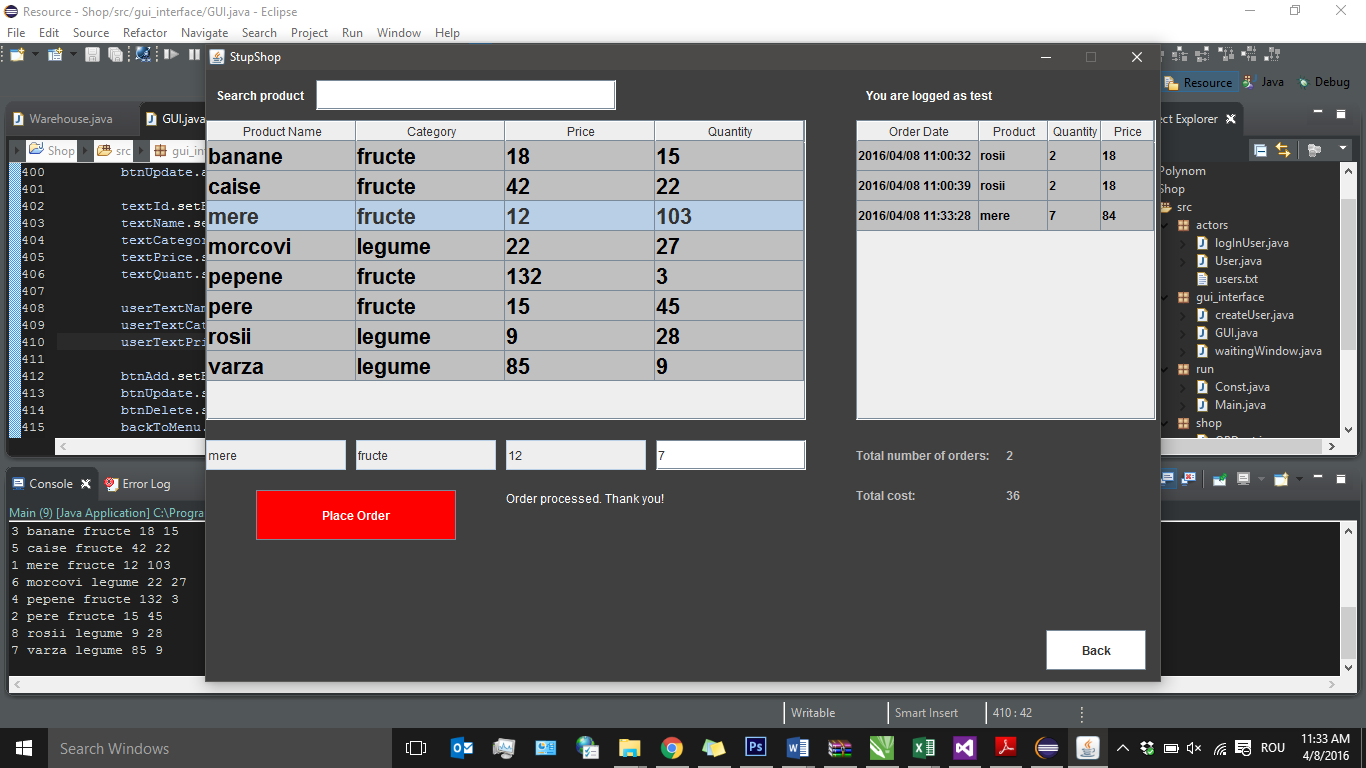
Se observă cum prosului rosii este selectat, cantitatea cerută este modificată (din 30 este modificată în 2), iar apoi este posibilă plasarea comenzii prin apăsarea butonului roșu.

Următorul pas, odată cu plasarea comenzii, este apariția unui nou frame ce arată utilizatorului că se procesează comanda. Chiar dacă procesarea ar fi fost instantă, am ales să folosesc această implementare deoarece : 1. S-a cerut ca metodă de îmbunătățire 2. Este o metodă utilă e a relaționa cu utilizatorul.

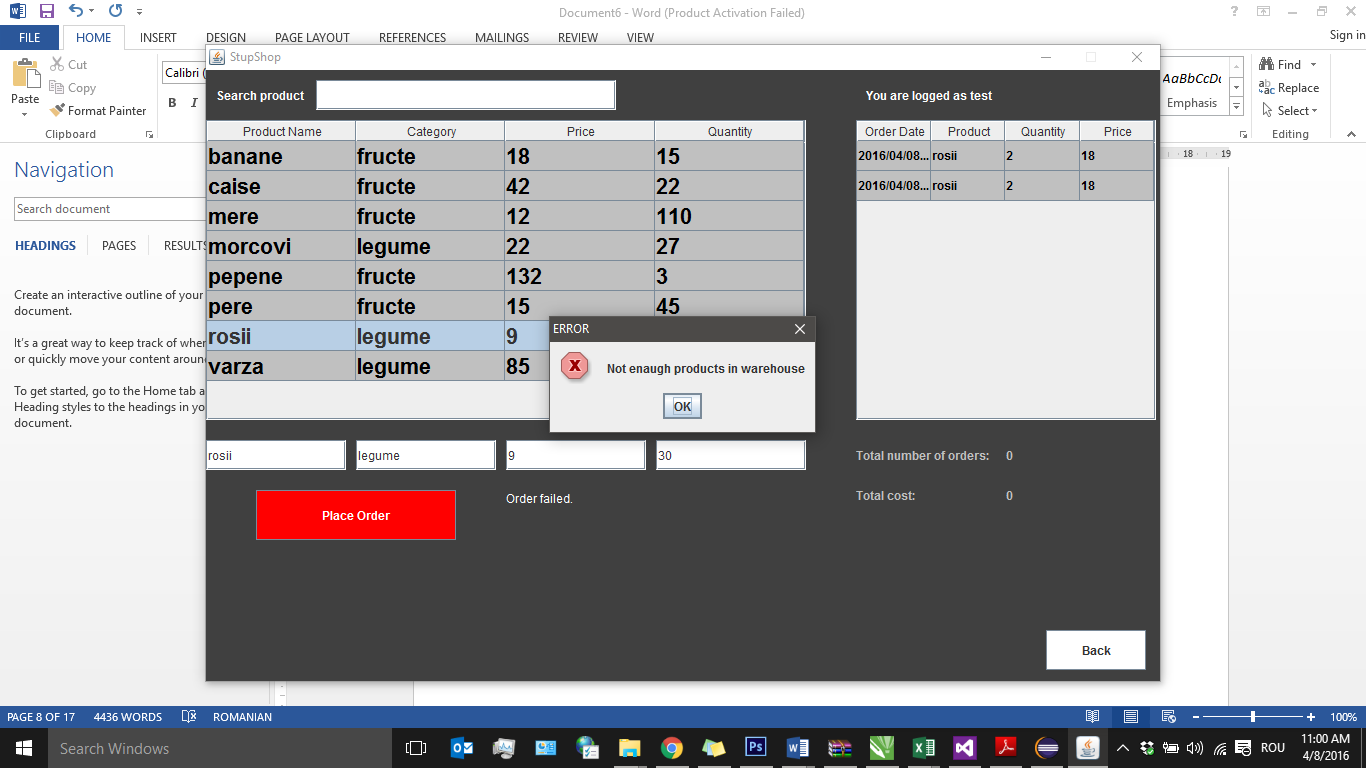
Acest mesaj de tip processing, induce practic o stare de așteptare a programului. Mesajul va apărea timp de 3-5 secunde, fiind generată aleator în acest interval durata exacta a așteptării.

Cât timp procesarea este derulată, utilizatorul nu va mai putea să opereze nicio acțiune, fiind nevoit să aștepte finalizarea procesului.

Odată finalizat procesul, au fost tratate două cazuri: primul caz, cazul în care comanda este corectă, iar cantitatea cerută nu depășește cantitatea aflată în stoc. În acest caz, după modelul relației Add product in warehouse, va fi creată o nouă instanță a clasei order cu atributele introduse și va fi adaugată în OPDept, care este, la fel ca și warehouse, o clasă ce implementează un binary search tree.

Se obsrvă cum apare mesajul langă butonul de plasare a comenzii ”Order processed. Thank you!”. De asemenea, se observă cum în tabelul ce conține comenziile facute, apare comanda efectuată la ora exactă. Comanda este introdusă imediat dupa procesarea cu succes a acesteia în tabel.

Al doilea caz, cazul în care utilizatorul comanda mai multe produse decât există pe stoc. În acest caz, comanda nu va fi efectuată, nu va fi instanțiată clasa de tip Order și tot ce se va întampla va fi faptul că, utilizatorul va fi atenționat asupra greșelii comise.

După cum se observă, este afișat mesajul ”order failed” lângă butonul de plasare a comenzii. Comanda nu este validată, deci nu este introdusă în sistem.

De asemenea, un mesaj de tip ERROR apare pentru a îl atenționa pe utilizator.

Ultimul lucru de sudiat, este legat de partea dreaptă a interfeței ce conține informația despre utilizator, tabelul cu comenziile făcute și scurte informații privitoare la totalitatea comenzilor. Butonul de back, este identic cu cel descris la interfața pentru admnistratori.

Informația afișată în partea de sus a interfeței, ”you are logged as ...” are scopul de a arăta utilizatorului că procesează comenzile de pe contul potrivit. Pot apărea cazuri în care utilizatorul precedent nu a închis aplicația, iar utilizatorul actual performează comenzi cu datele utilizatorului precedent. În acest caz, într-un caz real, comanda va ajunge la utilizatorul greșit, iar factura de plată va fi făcută pe numele utilizatorului greșit

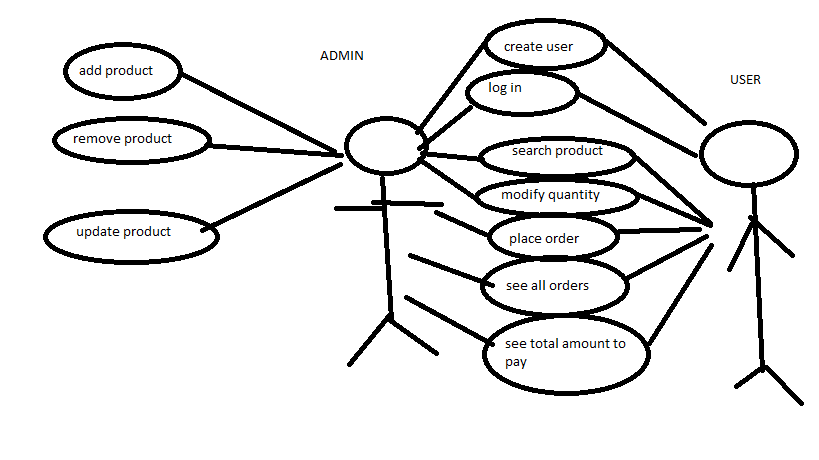
Tabelul cu comenzile făcute este imediat actualizabil, iar user-ul nu poate decât să privească lista de comenzi făcute. Nu există nicio operație ce poate fi efectuată asupra sa de către utilziator. Pentru dezvoltare ulterioară, am inclus introducerea opțiunilor de statut comandă, unde, odată plasată comanda, administratorul să poată semnala atunci când comanda a fost trimisă. Utilizatorul să vadă faptul ca aceasta a fost trimisă, iar atunci când comanda a ajuns, să poată plasa comanda ca fiind primită. De asemenea, în intervalul în care comanda este plasată și se așteaptă expediereaz ei, utilizatorul să poată să anuleze comanda. Toate aceste îmbunătățiri vor fi făcute în cazul în care varianta aceasta de program va fi și ea, în sine, îmbunătățită.

Raportul general al comenzilor făcute, respectiv numărul total al comenzilor și prețul total achitat este actualizabil doar după revenirea în interfață.

**3. Proiectare**

**3.1. Diagrama UML**

**3.1. Use Case Diagram**



Use Case Diagram relevă relația dintre user și program. Fiind două scenarii posibile, au fost ilustrate accestea. Admin-ul este, practic, un user ce are și drepturi de add product, modify product și delete product. Acțiunile pe care ei pot să le facă sunt prezentată în dragramă în modul corespunzător

**3.1.2. Class Diagram**

**3.1.3. Sequence Diagram**

**3.2. Structuri de date**

Structurile de date cele mai importante folosite sunt Binary Search Trees-urile ce implementează Warehouse și OPDept. Practic, fără aceastea proiectul nu ar funcționa deloc. Alegerea acesui mod de a stoca datele a fost impusă, însă alegerea în detaliu a fost la nivel de implementare punctuală a unui Binary Search Tree sau folosirea structurii de date Tree Map deja existentă în libraria java. Am ales folosirea implementării punctuale din două motive: 1. Am dorit ca acest proiect să mă ajute să înteleg mai bine utilizarea acestor structuri de date. 2. Chiar dacă am știut de existența Tree Map în java, am crezut că utilizarea lor nu este acceptată, fiind practic doar o metodă de a simplifica mult proiectarea programului.

**3.3. Proiectarea Claselor**

Acest punct se referă la modul în care a fost modelat proiectul din punct de vedere al claselor, bazat pe principiile OOP. Se iau în vedere împărțirea pe pachete, clasele conținute de fiecare pachet sau interfețele utilizate. La acest nivel am avut cele mai multe probleme, iar acest lucru nu deoarece nu am știut cum să impart clasele într-un mod corespunzător, ci a faptului că am dorit să le împart prea mult, iar de fiecare dată când o făceam o parte din proiect nu mai funcționa corespunzător. Din acest motiv, am fost nevoit să refac de 2 ori proiectul aproape de la zero, iar timpul alocat proiectului a fost 70% irosit pentru construirea de clase ce, utlerior, au fost șterse.

**3.3.1. Pachetul run**

Pachetul run conține două clase, Main și Const.

* Clasa Main este clasa principală a programului și apelează clasa GUI pentru a lansa programul.
* Clasa Const este o clasă de constante ce conține toate constantele programului. Am decis utilizarea acestei clase/metode de programare pentru a putea seta un set de caracteristici pe care să le pot folosi oriunde în proiect. Spre exemplu, fonturileși dimensiunile frame-urilor. Consider că, în orice proiect OOP, este foarte utilă construirea unei clase ce conține constantele din proiect, pentru a facilita accesul la modificarea lor. De asemenea, consider și un avantaj din punct de vedere al securității datelor.

**3.3.2. Pachetul gui**

Pachetul gui conține trei clase. Gui, WaitingWindow și CreateUser. Clasa principală este Gui, restul fiind apelate din cadrul cacestei clase.

* Clasa GUI este clasa ce creează și implementează în totalitate interfața grafică și modul în care aceasta funcționază. Voi dezvolta mai mult modalitatea de proiectare în cele ce urmează. Metodele principale utilizate sunt buildAdminPanel, ce creează interfața pentru administrator și buildUserPanel, ce creează interfața pentru utilizator. Clasa mai conține gettere și settere pentru username, astfel încât acesta să poată fi cunoscut în oricare dintre opțiunile de interfață. Clasa implementează actionListener și KeyListener. Implementarea ActionListener este necesară pentru a creeea acțiuni odată cu apăsarea butoanelor, selectarea tabelului etc. În cadrul acestei metode se petrec toate acțiunile declanșate de utilizarea butoanerlor. Implementarea KeyListener a fost opțională și deservește la posibilitatea logării odată completate datele de logare prin tastarea tastei enter, în loc să fie nevoie de apăsarea butonului de log in.
* Clasa Create user este definită separat pentru a asigura o cât mai bună lizibilitate a codului. Este apelată doar din GUI și deservește la crearea unui Jframe nou ce are ca scop introducerea datelor pentru un nou utilizator. Odată apăsat butonul process, utilizara acestei clase va înceta, Jframe-ul va fi închis, până la o nouă apelare a ei.
* Clasa WaitingMessage a fost implementată cu același scop de a oferi o bună lizibilitate. Aceasta este creată odată ce este apăsat butonul placeOrder, iar după ce timpul de așteptare este finalizat, Frame-ul este disposed, iar clasa va fi inactivă până la o nouă instanțiere sau apelare.

**3.3.3. Pachetul actor**

Pachetul actors conține clasele ce implementează utilizarea bazei de date de utilizatori. Aceasta conține două clase, logInUser și User, respectiv un fișier în care sunt notate conturile folosite pentru program.

* Clasa user este o clasă ce definește obiectul de tip user, cu caracteristicile sale. Acesta conține metode doar de tip getter și setter.
* Clasa logInUser este clasa ce se ocupă de funționalitatea logării, căutând în fișier existența contului și returnând un răspuns corespunzător.

**3.3.4. Pachetul shop**

Pachetul shop conține cele 4 clase oferite din cerință și care au fost necesare de implementat. De specificat este că, nu a fost nevoie de nicio altă clasă suplimentare pentru implementarea magazinului propriuzis. Clasele sunt serializabile, Warehouse și OPDept conținând metode de scriere în fișier respectiv citire conform regulilor de serializare.

* Clasa product este o clasă ce creează obiectul de tip produs, conținând doar gettere și settere.
* Clasa order este o clasă ce creează obiectul de tip order, conținând doar gettere și settere.
* Clasa Warehouse este o clasă ce conține multiple obiecte de tip product și le administrează/reține cu ajutorul Binary Search Tree. Aceasta conține metodele: addNode, ce adaugă un nou nod, findNode, ce caută un nod în BST după nume, remove, ce elimină un nod din BST după cheia produsului – toate acestea implementând BST-ul sau operațiile pe acesta. Metodele inorderTraversal, getRoot, toArray sunt folosite pentru parcurgerea BST-ului și convertirea datelor într-un format ușor accesibil din afara clasei.
* Clasa OPDept este asemănătoare cu Warehouse, având diferența că aceasta conține noduri de tip Order în locul celor de tip Product.

**3.4. Algoritmi**

Cei mai importanți algoritmi sunt cei ce implementează Binary Search Tree-ul. Aceștia sunt algoritmi clasici, astfel înât nu voi dezvolta implementarea lor.

Un algoritm, sau mai precis o metodă pe care aș dori să o dezvolt este cea ce asigură starea de Waiting odată plasată o comandă. Acest algoritm este important pentru implementare mai mult la nivel personal al dezvoltatorului, deoarece găsirea unei metode prin care să poată fi introdusă starea de așteptare fără a fi înghețată în sine toată procesarea GUI, a fost destul de dificilă.

|  |
| --- |
| *waitingW = new waitingWindow();*  *SwingWorker<Void, String> worker = new SwingWorker<Void, String>() {*  *@Override*  *protected Void doInBackground() throws Exception {*  *setRandomNumberBeetween3and5 = y*  *Thread.sleep(y\*1000);*  *waitingW.close();*  *Order order = new Order(orderNumber, orderDate, user, quantityOrdered, product, price);*  *quantTable = quantity of a product in table*  *if (quantTable < quantityOrdered) {*  *statusOrder.setText("Order failed.");*  *showErrorMessageNotEnoughProducts*  *} else {*  *statusOrder.setText("Order processed. Thank you!");*  *addOrderToTable*  *updateProductsQuantity*  *serializare*  *}}}* |

**3.5. Interfața grafică**

Interfața grafică se bazează, în mare, pe două structuri principale. Cele trei tipuri de interfață care sunt ecomutate între ele în funcție de alegerile făcute de către utilziator/administrator, sunt implementate sub forma de cardLayout. Primul layout afișat este cel al logării. Daca administratorul se loghează, cardLayout-ul va afișa interfața corespunzătoare administrării. Prin butonul de back, se revine practic, de oriunde ar fi apelat, la interfața de logare. Prin logarea unui user urmată de apelarea butonului ”search for products”, cardLayout-ul afișează interfața folosită de către utilizator pentru a plasa o comandă.

Cealaltă structură importantă folosită este Jtable-ul. Afișarea produselor sau a comenzilor este făcută în totalitate cu ajutorul Jtable.

Alte structuri importante folosite sunt JtextField, folosite peste tot unde este necesară introducerea sau selectarea/afisarea de date și Jbutton, toate din libraria swing.

**4. Implementare și testare**

Proiectul a fost implementat și testat doar in eclipse. Acesta ar trebui să fie funțional și în alte medii de testare, de asemenea. Sa-u testat și situații diverse prin care utilizatorul ar putea compromite utilizarea programului și s-au implementat remedieri corespunzătoare.

**5. Rezultat**

Rezultatul este o aplicație ce creează un sistem de Order Management și administrare de magazin. Aplicația poate fi dezvoltată astfel încât să fie chiar mai mult decât un simplu proiect pentru facultate.

**6. Concluzie**

Acest proiect a reprezentat, în primul rând, o lungă serie de nervi și timp irosit. Al doilea proiect în care observ că după ce am petrecut ore, multe, zeci chiar, scriind cot și testând, apare o problemă din cauza căreia nu poate fi funcțional, iar implementarea a trebuit reluată. Voi detalia unele situații apărute:

* Utilizarea de frame-uri noi pentru fiecare tip de acțiune. Să spunem că, nu exista o interfață de unde utilizatorul putea să facă totul, ci avea o interfață de unde alegea acțiunea. În urma apelării acelei acțiuni, un noi frame era creat cu datele specifice sau CardLayout-ul schimba interfața spre cea corespunzătoare. Dupa doua zile intense în care nu întelegeam unde era problema, am descoperit că, aparent, atunci când exista un action listener într-un frame nou creat/într-o interfață noua de card layout, actionListener-ul este apelat dublu. Odată atunci când este, in sine, apelat și odată când frame-ul este inchis sau, după caz atunci când cardLayout-ul schimbă interfața. Două zile (sau mai bine spus nopți întregi) în care am căutat motivul pentru care produsele introduse erau dublate, s-au redus la această problemă identificată cu mare dificultate. Evident, a fost necesară practic realuarea întregului proiect.
* Am încercat spargerea clasei GUI, mult prea mare pentru gusturile mele. Problema a apărut atunci când a fost necesară utilizarea butonului de back/sau orice altă acțiune ce avea loc în altă clasă (clasa gui de obicei). Nu am găsit o metodă de pasa actionEvent-uri în alte clase. M-a depășit situația, însă am pierdut cam o zi făcând acest lucru.
* O alta problemă a apărut la serializare. Am încercat totul posibil, ca dupa 2-3 zile de chinuri și cercetări indelungi să aflu că, problema se afla la nivelul apelurilor, respectiv a faptului ca serializarea era necesară după fiecare schimbare și nu la închiderea programului. Fapt minor, însă a consumat mulți nervi și energie. Evident, rezultatul era acela că nu se salva decât ultima modificare făcută.
* O altă problemă, dar mai mult la nivel de înțelegere a OOP-ului, am întâlnit-o când am dorit să creez o clasă care să performeze acțiunile. Am renunțat, de nervi, la a mai încerca.

Au mai apărut multe situații, însă rezultatul important este acela că , din aproximativ 70-80 de ore petrecute pentru a dezvolta proiectul, doar vreo 15-20 au fost în sine, constructive. Acum, mi-ar lua aproximativ 10 ore pentru implementare.

Un lucru foarte util ar fi, eventual, așa cum la proiectul trecut era minunat să avem o prezentare asupra RegEx, ar fi fost minunat să avem o prezentare asupra unor instrumente ce ne-ar fi ajutat, precum serializarea, Jtable (pe care nu le-am mai folosit niciodata), cardLayout și ce limitări aduc.

Cu siguranță, internetul este o soluție, dar una foarte costisitoare ca timp. Iar stackoverflow e extraordinar, însă pentru cazul în care apar probleme și rezolvarea lor, nu pentru crearea unei strategii de implementare.

7. Bibliografie

a) <http://stackoverflow.com/>

b) <http://www.tutorialspoint.com/>

c) <http://youtube.com/> - pentru tutoriale