Assignment 3
Warehouse management

Student: Găvenea Radu
grupa 30292

1. Obiectivul temei

1.1. Cerinta

Proiectati si implementati o aplicatie de management al comenzilor pentru procesarea comenzilor clientilor unui depozit de produse. Pentru stocarea produselor , clientilor si a comenzilor se va folosi o baza de date relationala. De asemenea , aplicatia trebuie sa contina cel putin urmatoarele clase :

- clase ale domeniului de activitate: Oder, Customer si Product
- clase ale nivelului de Business Logic: OrderProcessing , WarehouseAdministration , ClientAdministration
- clase ale nivelului de prezentare: clase GUI
- clase de acces date : clase de acces la baza de date

1.2. Descrierea problemei

Proictul are ca scop dezvoltarea unui aplicatii ce are rolul de a simula un sistem de comert de electronic pentru un depozit de produse in care clientii pot accesa si vizualiza diferitele produse ce le contine depozitul, pot adauga produsele intr-un cos de cumparaturi virual si pot plasa comenzii cu produsele dorite. La sfarsitul cumparaturilor, clientul are la dispozitie optiunea de a plasa comanda cu toate produsele adaugate in cosul virtual sau anularea comenzii.
Totodata utilizatorul aplicatiei are posibilitatea de a genera si vizualiza rapoarte asupra produselor care sunt in depozit, si anume un raport in care sunt evidentiate toate produsele din stoc care sunt in numar mai mic de 10 si un raport cu produsele ce au pretul de cumparare cuprins intre 100 si 200 unitati monetare.
Pentru a stoca informatia necesara aplicatiei este folosita o baza de date relationala.

2. Analiza problemei

Aplicatia are rolul de a simula un sistem de comert online in care un client/utilizator poate face cumparaturi din cadrul produselor disponibile in depozit. De asemenea aplicatia trebuie sa contina si evidenta tuturor clientilor care pot face cumparaturi in cadrul sistemului si o functionalitate de generare de rapoarte. In acest sens, aplicatia este construita in asa fel incat sa ii permita utilizatorului efectuarea a 4 functionalitati diferite, si anume: Administrarea clientilor, Administrarea produselor din depozit, Efectuarea de comenzii si Generarea de rapoarte.

2.1. Functionalitati principale

2.1.1. Administrarea clientilor

Utilizatorul aplicatiei are la dispozitie posibilitatea de efectuare a operatilor de vizualizare, de adaugare, de editare si de stergere asupra clientilor inregistrati in cadrul sistemului. Aceste operatii sunt disponibile in cadrul unei ferestre separate de restul aplicatiei.

2.1.2. Administrarea produselor

Utilizatorul aplicatiei poate efectua operatii de vizualizare, de adaugare, de editare si de stergere asupra produselor aflate in stocul depozitului. In aceasi maniera ca si in cazul clientilor, manipularea produselor se va face in cadrul unei ferestre diferite in cadrul aplicatiei.

2.1.3. Efectuarea de comenzi

Efectuarea comenzilor poate fi realizata in cadrul unei ferestre specifice in cadrul aplicatiei, in care clientul poate vizualiza continutul cosului de cumparaturi. Pentru adaugarea de produse in cadrul comenzii este necesar insa sa se deschida o sesiune de cumparaturi pentru un client specific, pentru a se sti cine efectueaza comanda. Acest lucru este posibil din cadrul ferestrei de Administrarea a clientilor, prin selectarea unui client si apasarea unui buton de incepere a sesiuni de cumpararturi. Dupa ce se efectueaza inceperea sesiuni de cumparaturi, produse pot fi adugate in cadrul cosului de cumparaturi in fereastra de Administrare a produselor, unde se selecteaza produsul dorit si cantittatea, apoi se adauga in cos prin apasarea unui buton corespunzator. La sfarsitul cumparaturilor, in cadrul ferestrei Cosului de cumparaturi, acesta poate revizui comanda si are optiunea de plasa comanda sau de a o anula. In urma plasarii sau anulari comenzii, sesiunea de cumparaturi pentru clientul respectiv se termina, iar cosul de cumparaturi se goleste.

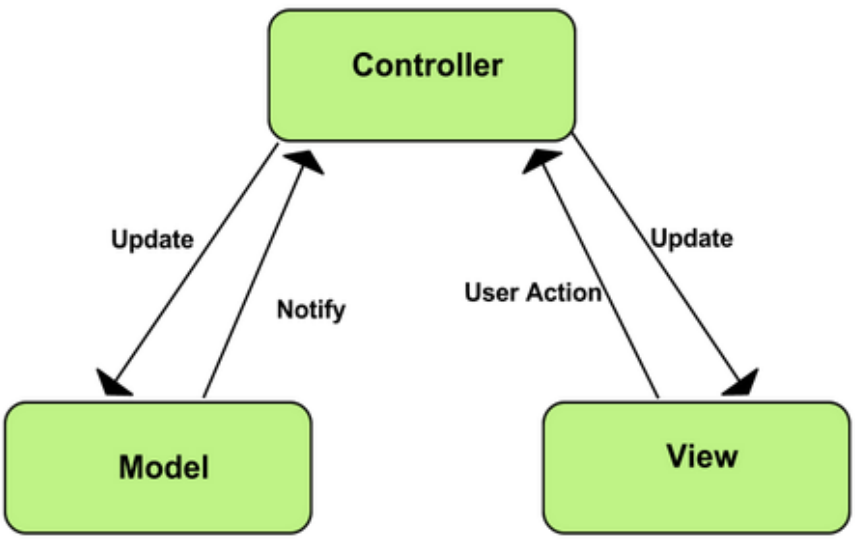
2.1.4. Generarea de rapoarte

O alta functionalitate pe care aplicatia o pune la dispozitie este aceea ca utilizatorul poate genera rapoarte cu privire la stocul produselor din depozit. Acest lucru este posibil prin accesare ferestrei de Rapoarte in care sunt disponibile doua tipuri de rapoarte ce pot fi generate, si anume: raport in care sunt afisate toate produsele din depozit ce sunt in numar mai mic de 10 si un raport in care sunt specificate toate produsele ce au pretul de cumparare cuprins intre 100 si 200 de unitati monetare. Aceste rapoarte vor fi generate in format csv.

3. Proiectarea

3.1. Arhitectura Model-View-Controller(MVC)

Aplicatia foloseste o structura de tipul mvc, care este un model arhitectural raspandit in ingineria software. Acest tip de model are la baza conceptul de izolare a logicii de business fata de partea de interfata cu utilizatorul, astfel incat aplicatia poate fi mai usor de realizat si modificat ulterior, putandu-se lucra doar la nivele dorite.



Model
In cadrul acestui nivel este realizata partea de business logic, aici sunt manipulate datele aplicatiei si tot aici se regasesc toate operatiile folosite pentru manipularea acestora. In cazul aplicatiei de fata, in cadrul efectuarii de comenzii, in partea de model sunt prezente clasele de reprezentare a entitatilor corespunzatoare tabelelor din baza de date.

View
Acest nivel face posibila interactiunea cu utilizatorul. Aici este realizata reprezentarea grafica a aplicatiei si exprimarea sub forma grafica a datelor. Aici este evidentiata informatia, inainte de a ajunge la controller. In cazul aplicatiei noastre, este folosita o interfata dezvoltata cu ajutorul graficii swing, impartita in 4 ferestre(tab-uri) diferite.

Controller
Acesta este nivelul de unde aplicatia poate fi controlata. Aici sunt manipulate rute, fisiere, metode care se afla in partea de model, si se face legatura intre partea de model si partea de view a sistemului. In cadrul aplicatiei cu polinoame, nivelul controller este implementat sub forma unor listeneri care in urma actiuni de apasare a unui buton de catre utilizator executa toate operatiile prezente in cadrul aplicatiei.

3.2. Cazuri de utilizare

Pentru realizarea unui depozit online de produse este necesar sa consideram cazurile de utilizare pentru un eventual utilizator.

Caz de utilizare: Efectuare comanda produse depozit

Actorul principal:
- utilizatorul

Preconditii:
- aplicatia trebuie pornita

Flow principal:
1. selectare a unui client din lista de clienti a magazinului
2. selectare incepere sesiune cumparaturi pentru clientul selectat
3. selectare produs pentru adaugare in cos
4. selectare cantitate numar de produse ce vor fi adaugate
5. selectare adauga in cos prin apasarea butonului corespunzator
6. reluare pasi 3-5 pentru adaugarea altor produse in cos
7. plasarea comenzii din fereastra cosului de cumparaturi prin apasarea butonui corespunzator

Flow alternativ:
5.1. sistemul returneaza mesaj eroare ca nu exista sesiune de cumparaturi pentru nici un client
5.1. se reiau pasi 1. si 2. pentru deschiderea unei noi sesiuni

Postconditii:
- sistemul afiseaza mesaj de succes cu privire la finalizarea comenzii

Diagrama UML Use-Case

In urmatoarea diagrama sunt prezentate toate cazurile de utilizare pe care un utilizator le are la dispozitie.

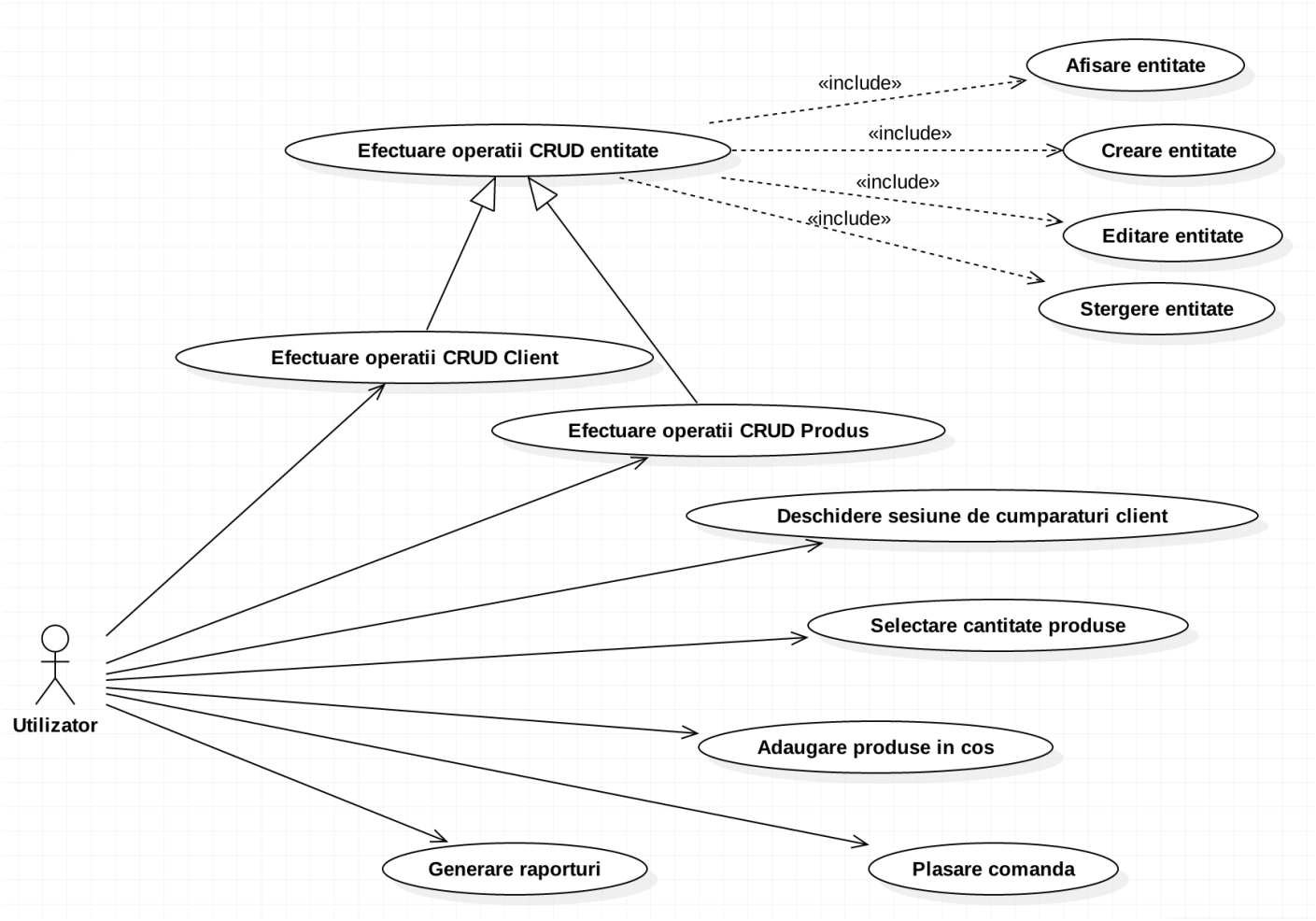


Diagrama use-case Warehouse Management Application

3.3. Stocare datelor

Pentru memorarea informatiei este folosita o baza de date MySql relationala. Pentru accesarea datelor din cadrul bazei de date este folosit driver-ul JDBC.

Mai este prezentata o imagine cu diagrama bazei de date corespunzatoare aplicatiei.

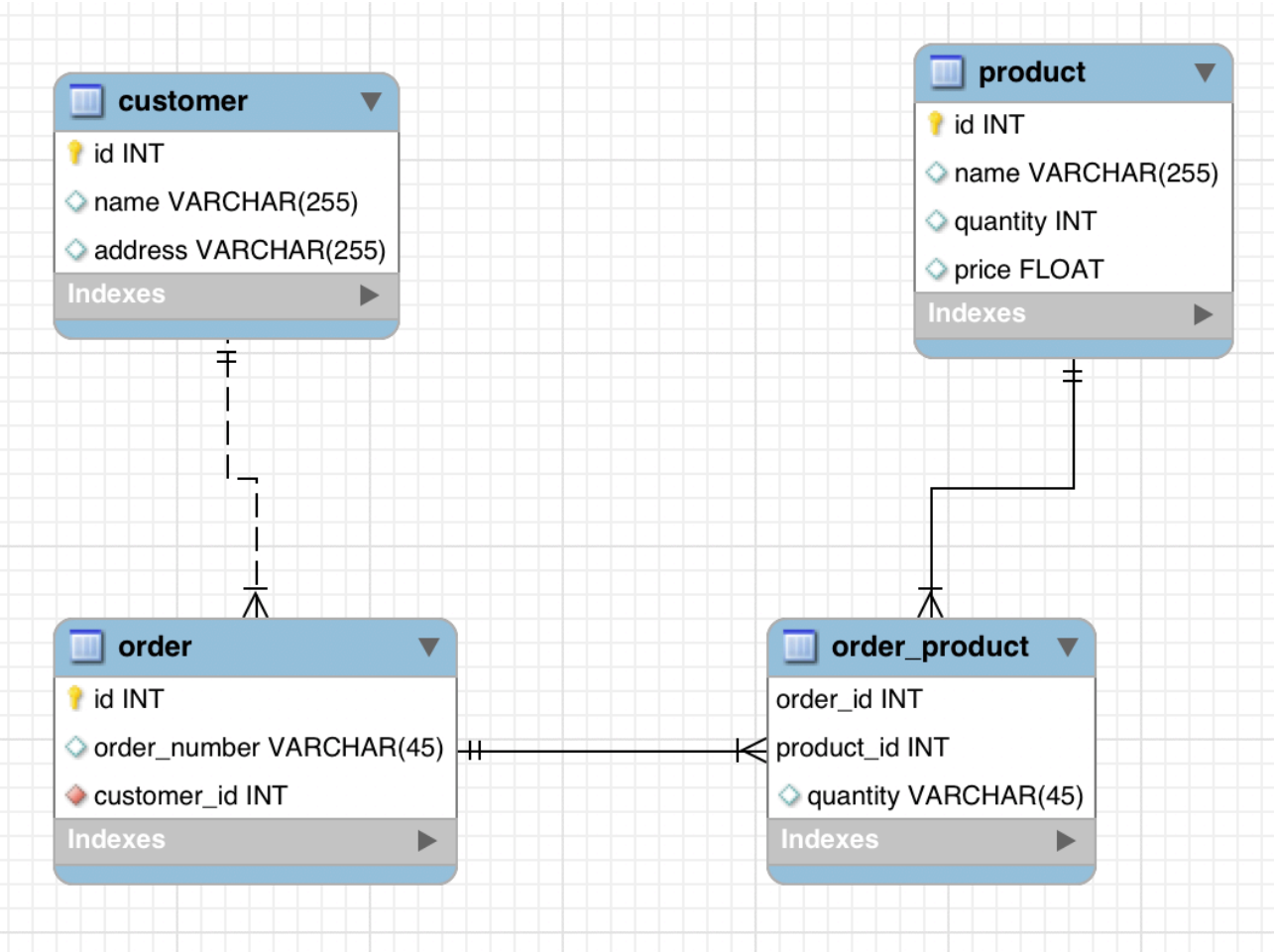


Diagrama bazei de date

4. Implementare si testare

4.1. Diagrama de clase

Din diagrama de clase prezentata in figura de mai jos se poate observa destul de usor arhitectura pe nivele folosita de tipul mvc.

Avem astel in partea de sus reprezentat nivelul de presentation.

In parte de mijloc a diagramei parcursa de sus in jos observam pe orizontala nivelul de controllere a aplicatiei, apoi un nivel mai jos este reprezentat nivelul de business logic al aplicatiei.

Iar in partea de jos a diagramei se poate observa nivelul de acces la date.

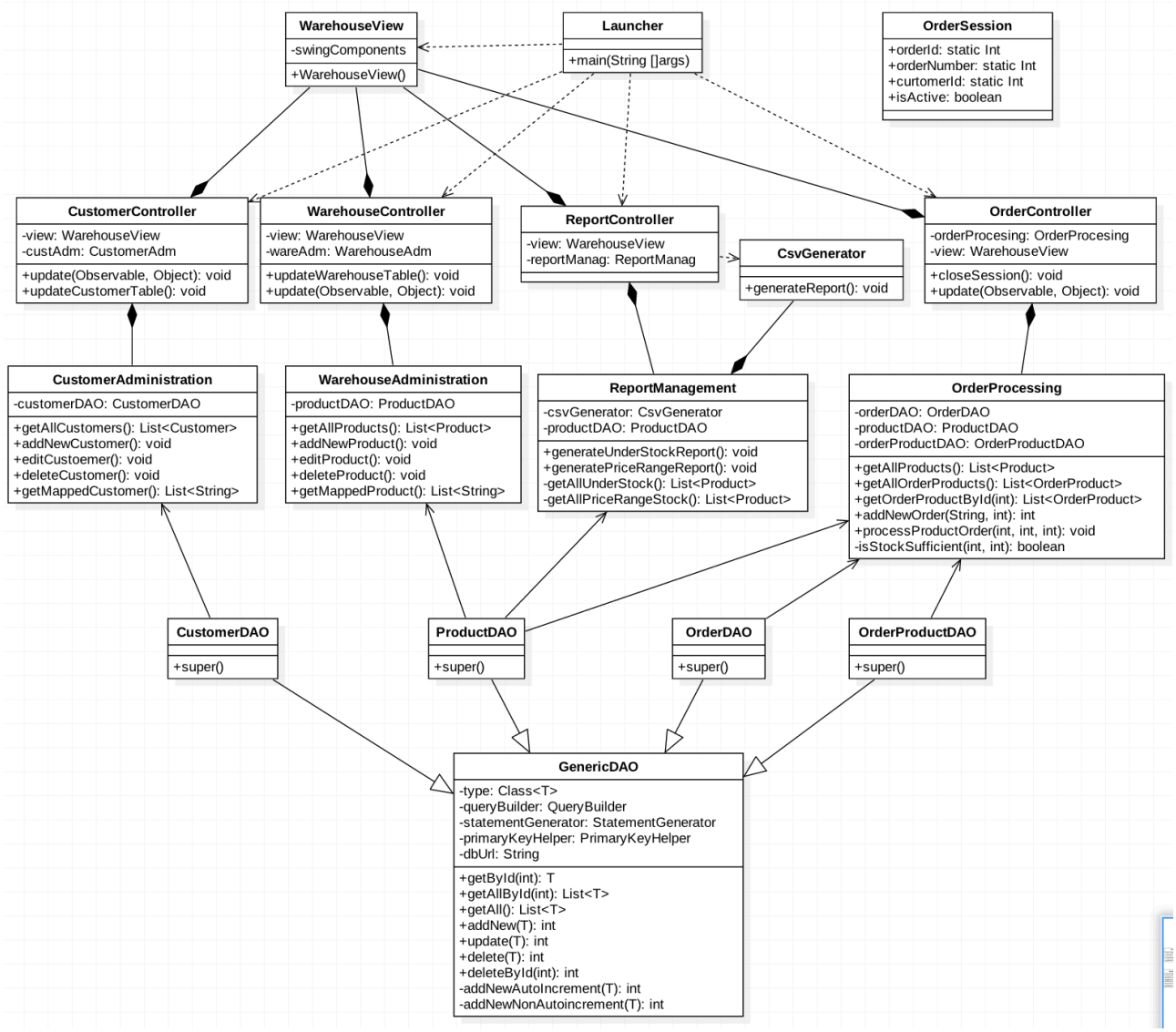


Diagrama de clase UML

4.2. Structura aplicatiei pe pachete

Dupa o analiza preliminara, am hotarat sa impartim aplicatia in urmatoarele pachete. Aceste pachete sunt alese in asa fel incat sa corespunda cat mai bine unei aplicatii pe nivele de tip MVC:

4.2.1. Pachetul <warehouse>

Acest pachet are ca obiectiv pornirea aplicatiei si mentinerea sesiunii de cumparaturi pentru un client.
Aici se regasesc clasele Launcher si OrderSession.
Clasa Launcher contine metoda main din care aplicatia porneste.
Clasa OrderSession contine niste variabile statice globale pentru memorarea sesiunii curente cu id-ul clientului si al comenzii.

4.2.2. Pachetul <connection>

Acest pachet contine clase ce permit connectarea la baza de date, precum si rularea unui script pentru crearea si initializarea bazei de date.

Clasa ConnectionUrl contine doua String-uri globale care specifica url-ul de connectare specific bazei de date utilizate pentru driver-ul JDBC. Sunt specificate ambele baze de date folosite in cadrul aplicatiei, si anume ur-ul pentru baza de date principala si un url pentru baza de date folosita pentru testare.

Clasa ConnectionFactory are rolul de creea conexiunea catre baza de date folosita pe baza url-ului specificat.

Clasa DbSqlScript este folosita pentru initializarea bazei de date folosite in cadrul blocului de testare. In cazul in care nu exista o baza de date pentru testare pentru a fi initializata se va creea automat una.

4.2.3. Pachetul <dataAccessLayer>

Acest pachet cuprinde clasele folosite pentru a facilita accesul la baza de date. Toate entitatile dorite in cadrul programului sunt reprezentate printr-o clasa specifica cu nume corespunzator tabelelor din baza de date. Pentru a nu se rescrie metode de accesare si manipulare a inregistrarilor in baza de date pentru fiecare entitate in parte a fost creata o clasa generica ce are rolul de a efectua operatiile dorite, iar clasele din acest pachet extind toate acea clasa.

4.2.4.Pachetul <dataAccessLayerUtils>

Acest pachet contine clasele necesare pentru efectuare de manipulare a bazei de date, de acces asupra informatiei din baza de date, prin specificare unei entitati de tip generic, urmand a fi extinsa de fiecare entitate in parte.
In acest sens a fost creata clasa <GenericDAO> ce contine toate metodele necesare pentru manipularea informatiilor din baza de date, cum ar fi aducere din baza de date a tuturor inregistrarilor dintr-un tabel, sau stergerea unei inregistrari pe baza id-ului etc. Aceste metode au fost implementate folosind tehnica Reflexion pe care Java il pune la dispozitie, toate operatiile sunt construite pe baza numelor entitatilor si campurilor respective a tabelelor din baza de date.
Pentru a respecta principiile Solid, au fost create urmatoarele clase ajutatoare: Clasa <PrimaryKeyHelper>, <QueryStringBuilder>, <StatementGenerator>.

Clasa <PrimaryKeyHelper> are rolul de gasi pozitia cheilor primare in cadrul unui tabel , de verificare a faptului ca o cheie primara este auto-incrementata sau nu, etc.

Clasa <QueryStringBuiler> are rolul de construi siruri de caractere corespunzatoare query-urilor ce vor fi folosite pentru accesarea bazei de date. Metodele de construire a sirurilor de caractere folosesc Reflection pentru a detecta pe baza entitatilor cum sunt numite campurile in baza de date.

Clasa <StatementGenerator> este o clasa cu scopul de acrea statement-urile necesare pentru executarea queri-urilor asupra bazei de date si de obtine Obiectele aduse din baza de date si construirea acestora pe baza Entittatii corespunzatoare. Aceste lucruri se realizeaza, de asemenea, tot cu reflexion pe baza tipul generic T.

Cu ajutorul acestui pachet se pot efectua operatii asupra bazei de date pentru toate entitatile ce extind clasa <GenericDAO> cu conditia ca se respecta numirea tabelelor la fel in baza de date cu entitatile din aplicatie, iar campurile din tabele trebuie sa aibe aceleasi denumiri, de asemenea, cu atrinutele entitatilor respective.

4.2.5. Pachetul <model>

In cadrul acestui pachet sunt pastrate toate entitatile folosite de sistem. Sunt practic niste clase ce mapeaza tabele din baza de date cu atribute corespunzatoare campurilor din baza de date. Aceste clase contin metode de get si set pentru toate atributele clasei respective.

4.2.6. Pachetul <businessLayer>

Acest pachet are ca scop incapsularea logicii aplicatiei. In cadrul claselor acestui pachet se efcectueaza toate operatiile de manipulare si procesare a datelor cu care utilizatorul aplicatiei lucreaza. Este o colectie de clase de servicii pentru toate operatiile efectuate in cadrul sistemului. Acest pachet este un intermediar intre nivelul de acces direct asupra informatiilor din baza de date si nivelul de control a utilizarii aplicatiei prin cadrul nivelului de presentare a aplicatiei(interfata grafica).

Clasa <WarehouseAdministration> contine toate metodele de servici disponibile utilizatorului cu privire la produsele din depozit. In acest sens sunt disponibile metode ce apeleaza la randul lor metode din pachetul de acces la date, insa si metode de procesare si efectuare de operatii asupra entitatilor de tip Produs.

Clasa <CustomerAdministration> contine toate metodele de servici disponibile utilizatorului cu privire la clientii ce sunt inregistratii in cadrul aplicatiei. In acest sens sunt disponibile metode ce apeleaza la randul lor metode din pachetul de acces la date, insa si metode de procesare si efectuare de operatii asupra entitatilor de tip Client.

Clasa <OrderProcesing> este o clasa ce are ca scop procesare comenzilor. In cadrul acestei clase sunt metode de adaugare si manipulare a comenzilor, precum si metode de verificare a stocului disponibil si altele ajutatoarea in acest sens.

Clasa <ReportManagement> este clasa folosita pentru generarea de rapoarte in cadrul aplicatiei cu privire la informatii despre disponibilitatea stocului sau a preturilor diferitelor produse pe care depozitul le pune spre vanzare. Aici sunt prezente metode de generare a diferitelor rapoarte, precum si metode necesare pentru colectarea datelor despre produse corespunzatoare cu necesitatile de afisare a fiecarui tip de raport.

Clasa <CsvGenerator> este o clasa folosita pentru generarea rapoartelor sub format de tip csv. Aceasta clasa contine o metoda de generare a unui raport pe baza unei liste de produse ce vor fi continute in cadrul raportului. In cadrul acestei metode se creeaza fisierul .csv cu datele necesare ce va avea nume data si ora actuala. De fiecare data cand se creeaza un nou raport, un nou fisier este generat.

Clasa <CsvGeneratorUtils> este o clasa ajutatoare pentru clasa <CsvGenerator>. In cadrul acestei clase sunt defintite metode de scriere in fisier pentru a fi respectate normele formatului .csv.

4.2.7. Pachetul <controllers>

Acest pachet contine controlere pentru cele mai importate 4 functionalitati a sistemului. Asadar au fost creaate urmatoarele clase:

Clasa <WarehouseController> este folosita pentru furnizarea controlul pentru Obiectele de tip Produs. Aceasta clasa face legatura intre nivelul de View, in care utilizatorul interactioneaza cu aplicatia si nivelul de Model, unde sunt prezente toate serviciile pe care aplicatia le pune la dispozitie pentru manipularea obiectelor de tip Produs. Aceasta clasa are in cadrul ei subclase de

listeneri ce au rolul de asculta apasarea de butoane a utilizatorului in cadrul interfetei grafice si efectuarea operatiilor corespunzatoare pe baza interpretarii listener-ilor respectivi. De asemenea, exista si listeneri pe tabel pentru identificarea produsului selectat si efectuarea operatilor corespunzatoare pentru produsul respectiv.

Clasa <CustomerController> este o clasa ce are rolul de furniza controlul pentru Obiectele de tip Customer. Aceasta clasa face legatura intre nivelul de View, in care utilizatorul interactioneaza cu reprezentarea grafica a clientilor aplicatiei si nivelul de Model, unde sunt prezente toate serviciile pe care aplicatia le pune la dispozitie pentru manipularea obiectelor de tip Customer. Aceasta clasa are in cadrul ei subclase de listeneri ce au rolul de asculta apasarea de butoane a utilizatorului in cadrul interfetei grafice si efectuarea operatiilor corespunzatoare pe baza interpretarii listener-ilor respectivi. De asemenea, exista si listeneri pe tabelul de afisare a clientilor pentru identificarea produsuli selectat si efectuarea operatilor corespunzatoare pentru clientul respectiv.

Clasa <OrderController> are ca scop de furnizarea controlul pentru Obiectele de tip Order. Aceasta clasa face legatura intre nivelul de View, in care utilizatorul interactioneaza aplicatia si nivelul de Model, unde sunt prezente toate serviciile pe care aplicatia le pune la dispozitie pentru manipularea obiectelor de tip Order si procesarea comenzilor. Aceasta clasa are in cadrul ei subclase de listeneri ce au rolul de asculta apasarea de butoane a utilizatorului in cadrul interfetei grafice si efectuarea operatiilor corespunzatoare pe baza interpretarii listener-ilor respectivi. Tot in cadrul acestei clase se fac si plasarile de comanda si adaugarea de produse noi in cosul de cumparaturi.

Clasa <ReportController> este o clasa folosita pentru a efectua legatura dintre listener-ii butoanelor de generarea a rapoartelor din cadrul interfetei grafice si apelarea servicilor de generare a rapoartelor corespunzatoare. Aceasta clasa face legatura intre nivelul de View, in care utilizatorul interactioneaza cu aplicatia si nivelul de Model, unde sunt prezente toate serviciile pe care aplicatia le pune la dispozitie pentru efectuarea rapoartelor.

4.2.8. Pachetul <presentation>

Acest pachet ca scop prezentarea sub forma grafica a aplicatiei. Aici este creata interfata grafica disponibila pentru ca utilizatorul sa poata interactiona cu sistemul.

Acest pachet contine o singura clasa <WarehouseView> ce are rolul de furniza interfata utilizator pentru aplicatie. Acest lucru este realizat cu ajutorul framework-ului Swing pus la dispozitie de limbajul Java.

4.2.9. Pachetul <presentationTableUtils>

Acest pachet este folosit pentru realizarea unei functionalitati de generare a tabelelor din cadrul interfetei grafice pe baza numelor atributelor entitatilor prin metoda de reflexie in mod automat. In acest sens toate tabelele din aplicatie vor avea ca si coloane numele atributelor corespunzatoare.

Pentru acest lucru a fost creata o clasa generica <GenericTableModel<T>> care extinde clasa <DefaultTableModel> si suprascrie unele din metodele aceteia. De asemenea, aceasta clasa contine metodele necesare pentru realizarea tabelelor prin metode ce citesc prin reflexie numele coloanelor, si primesc ca parametrii de intrare Vectori generici de tip T.

4.3. Testare

Functionalitatea aplicatiei este testa printr-un bloc de teste de tip unit test. Aceste teste verifica toate operatiile de vizualizare, de creare, de editare si de stergere care se pot efectua asupra entitatilor din cadrul aplicatiei. Astefl au fost create 4 module de teste pentru fiecare entitate in parte, si anume: <CustomerCRUDTests>, <OrderCRUDTests>, <OrderProductCRUDTests>, <ProductCRUDTests>.

Pentru testare este folosita o baza de date separata ce are ca scop doar rularea acestor teste. Toate aceste clase de teste contin in faza de initialiare aplelarea unei metode de initializare a bazei de date corespunzatoare testarii. Aceasta metoda are rolul de a ire-initializa baza de date cu date corecte pentru procesarea testelor. In cazul in care nu exista baza de date corespunzatoare testarii aceasta este creata automat in cadrul procesului.

5. Rezultate

Aplicatia functioneaza conform asteptarilor/ cerintelor. Utilizatorul acesteia poate efectua o comanda prin adaugarea produselor in cos si finalizarea comenzii.
In cazul in care o comanda a fost finalizata cu succes, un mesaj corespunzator este afisat de catre sistem pentru a confirma acest lucru.

Iar in cazul in care erori de folosire a aplicatiei apar, de asemenea, sunt generate de sistem mesaje corespunzatoare pentru a informa utilizatorul ce trebuie sa faca sau ce pasi sa urmeze. Un exemplu in acest sens este cazul in care utilizatorul nu deschide o sesiune noua de cumparaturi pentru efectuarea comenzii. Sistemul raspunde cu mesaj de eroare si redirectioneaza utilizatorul spre faptul ca trebuie sa porneasca o sesiune noua.

6. Concluzii

Aplicatia este una pur didactica si nu are aplicabilitate in viata reala, fiind doar un mic prototip pentru o aplicatie ce ar necesita mai multe resurse de dezvoltare.

Dezvoltari ulterioare pentru aceasta aplicatie pot fi reprezentate prin introducerea urmatoarelor functionalitati: adaugarea unui sistem de autentificare pentru fiecare client in parte, posibilitatea de modificare a comenzii inainte de plasarea acesteia, implementarea unui sistem de generare factura si trimiterea acesteia prin e-mail clientului corespunzator etc.

7. Bibliografie

3. http://www.wikipedia.org/
4. http://stackoverflow.com/