DOCUMENTAȚIE PROIECT

MANAGEMENT DE COMENZI

Stancu Mihai-Cristian

1. Obiectivul temei

a. Obiectiv principal

i. De a crea o aplicatie care manageriaza comenzile

b. Obiective Secundare

- Analizarea problemei și identificarea cerințelor (Capitolul 2)
- Proiectarea aplicatiei (Capitolul 4)
  - descrierea tuturor pachetelor
  - descrierea tuturor claselor
- Implementarea aplicatiei (Capitolul 3)
- Testarea aplicatiei a aplicației si a functionalitatii(Capitolul 5)

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

2.1 Cerințele funcționale si non-funcționale

Dupa ce am citit problema si am analizat textul, pot indentifica principalele cerinte:

1. functionale pentru calculatorul polinomial:
   - Trebuie sa am un camp mare pentru logul de evenimente in care o sa aratat clientii, produsele, comenzile
   - Trebuie sa am textfielduri si spinere pentru ca utilizatorul sa poata introduce date
   - Trebuie sa am o metoda de comunicare cu baza de date prin care UI-ul va prelua date
2. Cerințele non-funcționale ale calculatorului care sunt logice cand vine vorba despre ajutarea utilizatorului:
   - Trebuie sa fie intuitiv
   - Trebuie sa aibe validari:
     - Sa fie mail-ul corect
     - Sa nu fie campuri goale

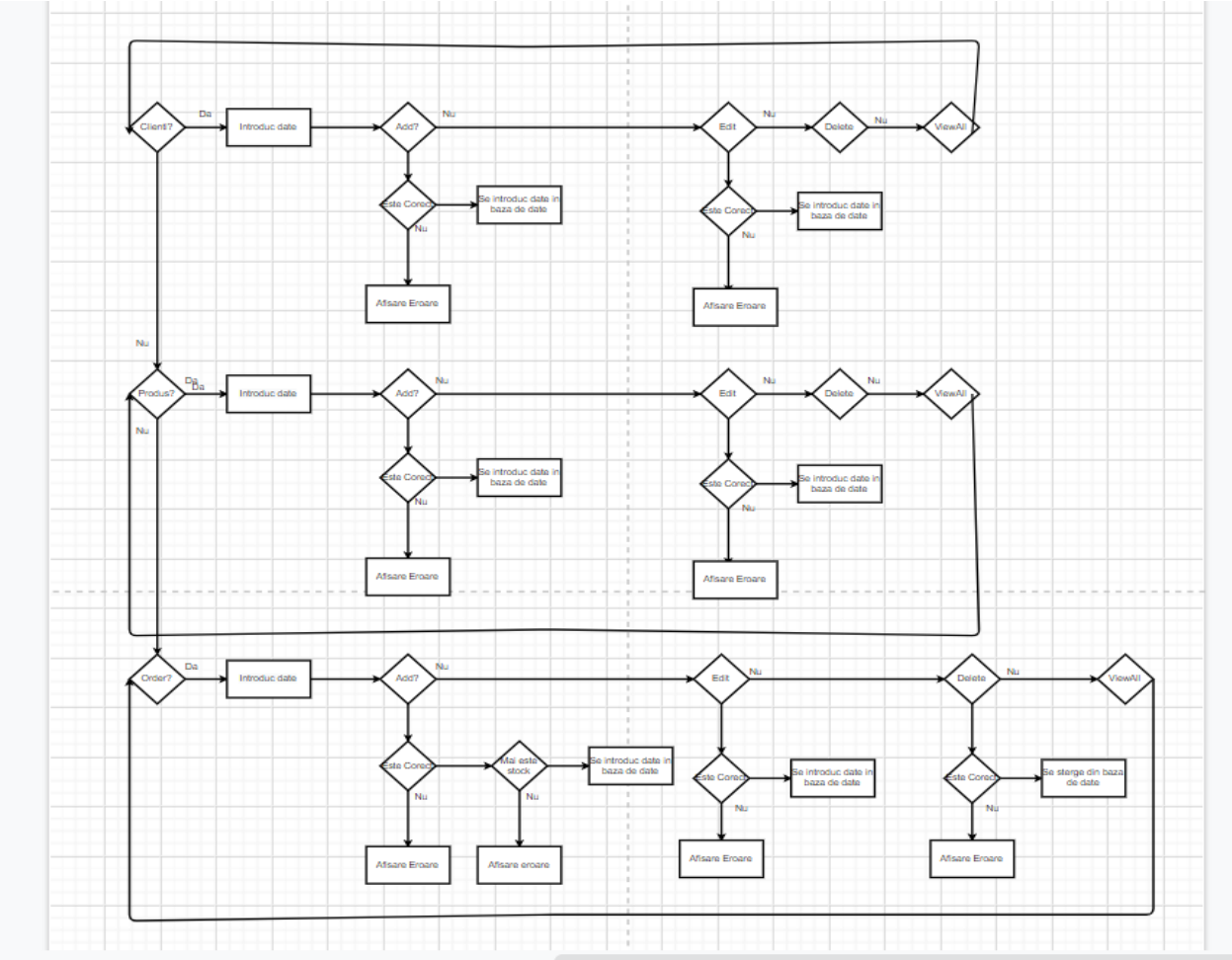
2.2 Descrierea diagramei de use-case

- Aplicatia are un numar destul de redus de use casuri, insa pentru o aplicatie micuta, poate fi privita ca fiind destul de complexa. Utilizatorul poate alege prin intermediul UI si al butoanelor fereastra in care vrea sa stea, anume Clients, Products, Orders. De asemenea, pentru fiecare fereastra, vor fi 4 butoane care reprezinta operatii CRUD, si anume Add, Edit,Delete,View All. De asemenea, in fiecare window vor fi zone in care se vor putea introduce datele, cum ar fi:
  - La client : nume, email,adresa,varsta si id.
  - La comanda: id-ul clientului, id-ul produsului, cantitea si id-ul

- La produs: numele, pretul, cantitatea, id-ul

- La fiecare window am zona de messages in care primesc mesaje cand s-a introduc un client, sau daca s-a editat,sters etc. La orders am butonul finish purchase care imi ia comanda pentru clientul cu id-ul luat din ui si imi face un pdf cu chitanta care contine numele clientului si idul, numele produselor si suma totala. De asemenea pentru order am un buton de purchase care imi decrementeaza cantitatea produsului pe care clientul cu id-ul id incearca sa-l cumpere, insa daca trece peste stocul respectiv, va fi informat ca nu mai este stoc. La adaugarea de client nou am validari pe mail, nume , varsta si adresa.

2.3 Descrierea use-case-urilor



3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

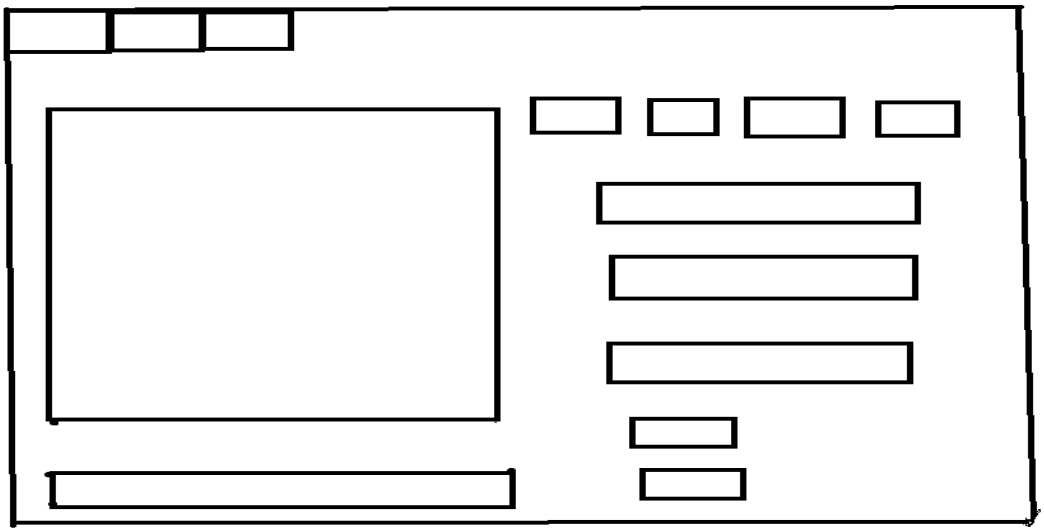
3.1 Decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete

In etapa 3, dupa ce am analizat celelealte 2 etape ce ce parte functionala si non-functionala, mi-am creat un mock in Paint pentru a incerca sa vizualizez partea de interfata grafica si si sa incerc s-o duc in arhitectura MVC pentru a avea o structura clasa si concisa pentru program.

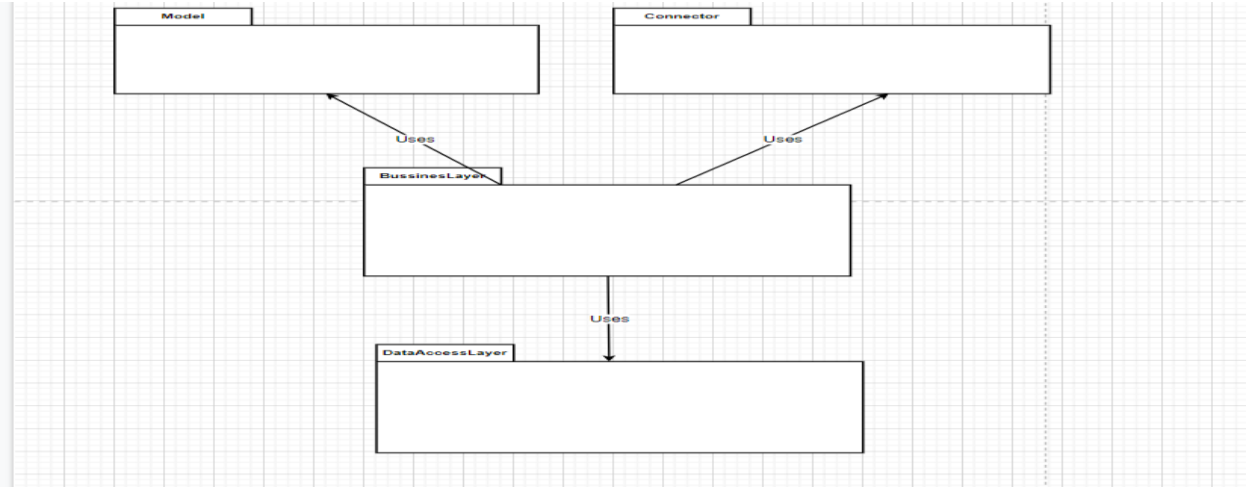
Am 5 pachete, model,connector,dataAccessLayer , businessLayer si presentation:

- *Model*: In model am clasele Pojo care vor fi mapate in baza de date. Aici nu este logica implicate, ci doar contruirea de clasa Boiler precum getter si si setter si toString al claselor Product, Order si ProductOrder.
- *Connector* contine partea de de conectivitate catre baza de date prin intermediul atributelor aferente
- *DataAccessLayer* contine clasa Reflection de operatii Crud care fac inserarea, stergerea sau update-ul in baza de date
- *Presentation* contine partea de controller care imi ia datele din UI si mi le prelucreaza / valideadeaza pentru a putea fi date mai departe catre Service si catre Dao si in final sa ajunga datele in database.
- Partea de GUI o am in fxml, 4 la numar, astfel am rupt logica aplicatiei de UI pentru a le putea manevra cu o mai mare usurinta si pentru a putea avea dezvoltari ulterioarea mult mai concise.

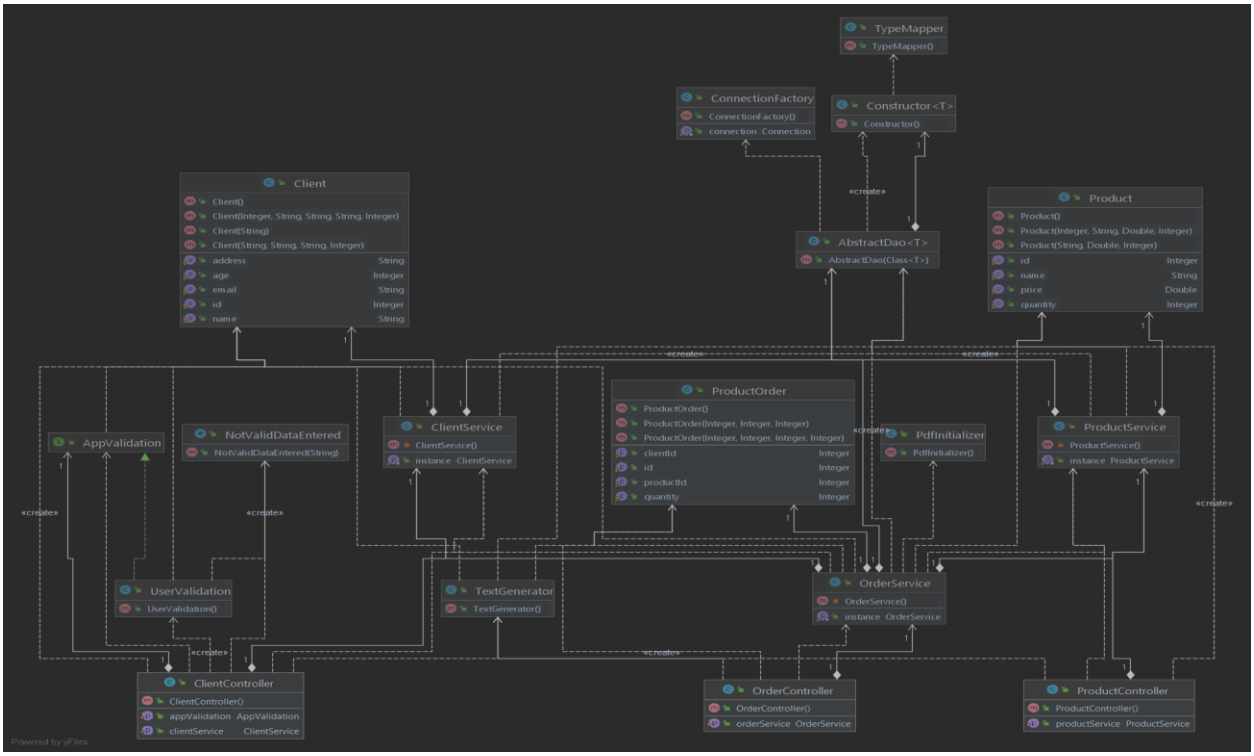
Mock



3.2 Diagrama de pachete



3.3 Diagrama de clase



4. Implementare

La implementarea temei am inceput prin analizarea a a detaliilor despre JDBC, am inceput prin a studia metodele basic de conectare si creere a operatiilor CRUD. Mi-am organizat clasele in pachete si am mi-am mapat clasele in model.

1) PACKAGE PRESENTATION

- Aici am incorporat partea de controller care imi ia datele din UI. Pentru fiecare Pojo am creat un controller. Am un :
    - TabPaneController pentru a mi putea crea celelalte 3 controller pe acelasi window mare. In tabpane nu am nimic.

    - ClientController am partea de comunicare UI-cod a ferestrei de Client. Am metodele de editare, adaugare ,stergere stergere a clientilor. De de asemenea comunic cu pachetul de validare pentru a valida campurile goale sau emailul prin intermediul unui regex. In UI am 4 butoane care au listeneri. Mai am 3 Textfielduri pentru nume, mail si adresa si 2 spinnere pentru varsta si id. Am un TextArea pentru mesaje mesaje informative cu privirea la inputul dat de utilizator. Root-ul aplicatiei este pe un ScrollPane care imi arata totii clientii prin intermediul unor butoane cu text.
    - OrderController am partea de comunicare UI-cod a ferestrei de Order. Am metodele de editare, adaugare ,stergere a comenzilor. In UI am 4 butoane care au listeneri. Am 4 spinnere pentru clientId, productId si cantitate si si Id. Am un un TextArea pentru mesaje informative cu privirea la inputul dat de utilizator. Root-ul aplicatiei este pe un ScrollPane care imi arata toate comenzile prin intermediul unor butoane cu text.

    - ProductController am partea de comunicare UI-cod a ferestrei de Product. Am metodele de editare, adaugare ,stergere a produselor. In UI am 4 butoane care au listeneri. Am 2 spinnere pentru cantitate si Id.Am si si 2 textfielduri pentru nume,pret Am un TextArea pentru mesaje informative cu privirea la inputul dat de utilizator. Root-ul aplicatiei este pe un ScrollPane care imi arata toate comenzile prin intermediul unor butoane cu text.

- Intregul UI este creat in FXML. Dupa cum se poate observa, am un tabPane, iar fiecare tab in partea reprezinta un controller, astfel pot avea mai multe windwouri cu componente de UI diferite in acelasi Window parinte. Am folosit initializare la fiecare controller pentru a initializa spinerele de la fiecare window cu valorea 0. De asemenea, am initializat fiecare service in metoda de INIT

2) PACKAGE MODEL

- Am clasele pojo Client, Product si ProductOrder cu atributele aferente care sunt mapate conform bazei de date.

3) *PACKAGE DataAccessLayer*

- Am clasa AbstractDao<T>. Este o clasa care imi creeaza endpointul aplicatiei catre database. Practic aici am toata logica de sql a aplicatiei. In contructorul clasei am un obiect type de tip T pentru polimorfism.
- Am metodele:
    - takeValues care imi parseaza fiecare field al unui obiect de tip T si imi ia adauga la o lista de tip object toate valorile din clasa clasa respective, valori pe care le iau din UI.

- createObjects este metoda care imi genereaza obiectele pe care le pun in baza de date. Prin intermediul lui resultSet, parsez toate coloanele din baza de date si procesez acele entitati.
- selectAll imi ia fiecare rezultat din tabela respectiva si mi-l afiseaza in UI.
- findById imi ia clasa respectiva pe baza unui id pe care il dau prin intermediul interfetei si al spinnerului ID.
- delete imi ia pe baza imi face update pe tabela T prin intermediul statementului si imi sterge entitatea cu id-ul id.
- create imi creeaza un nou obiect de tipul T, iar valorile din UI sunt luate prin intermediul lui takeValues, crearea se se face pe baza ultimului update din tabela, tabela avand in SQL auto-increment.
- update imi updateaza obiectul T de la id-ul id cu noul obiect model, tot de tip T. Datele pe care le voi da sunt luate din UI, iar updateul se face prin intemediul lui prepareStatement.executeUpdate(). Datele sunt luate din UI prin intermediul metodei publice takeValues

4) *PACKAGE Connector*

- Am o singura metoda importanta in clasa ConnectionFactory care prin intermediul DriverManagerului obtin conexiunea de lla a baza de date prin url local al bazei de date, usernameulului si parolei bazei de date. De asemenea mai am metodele de close care sunt suprascrise pentru ResultSet,Statement si Connection.

5) *PACKAGE businessLayer*

- Am pachetele interioare:
  - Exception:
    - Am exceptia pe care o folosesc la validari: NotValidException care primeste mesaje corespunzatoare in momentul validarii datelor din UI.
  - Util
    - Constructor: Imi construiesc listele de fielduri sau de tipuri ale fieldurilor prin intermdiul prin intermediul reflexiei. De asemnea am 2 metode pentru update si insert query in care imi formez queryurile corecte din punct de vedere al sintaxei SQL pentru a-mi prelua sau a pune datele in baza de date.
    - PdfInitializer: am 2 metode statice in care imi adaug un header la pdf cu campurile : client, products, total price. De asemenea am metoda de adaugare a unei celule in PDF.
    - TextGenerator este clasa care imi genereaza in mod corect textul care va intra in messageArea. Folosesc un atribut method care imi spune ce fel de operatie CRUD a fost facuta de catre utilizator. Am cate o metoda pentru fiecare clasa.
    - TypeMapper imi construieste tipurie de date ca sa se potriveasca cu denumirea metodelor prin metoda static a lui String: startsWith().
    - In service am partea de business a aplicatiei, unde primesc data de la ui sau de la dao si unde fac fac operatii de logica suplimentara. Pentru fiecare instantiere de clasa folosesc acelasi obiect prin intermediul In feicare fiecare clasa,mai putin order, doar trimit datele mai departe in oricare dintre cele 2 flowuri ale aplicatiei. In clasa OrderService am si logica de obtinere a sumei totale pe care o cheltuieste clientul cu id-ul id. Ea se putea calcula la fel de usor si cu niste queryu - uri in sql, dar am decis sa folosesc streamuri pentru o logica apropiata. De de asemenea am si metoda createReceipt care imi parseaza lista de productOrder si imi ia productOrderurile de la clientul cu id-ul id. De asemenea, pentru fiecare client ales prin id, incrementez un receiptCount declarant static pentru a denumi fiecare PDF in in mod intuitiv, folosidu-ma de utilul din pachetul util al PDF-ului pentru header si pentru adaugarea de noi randuri la tabelul din PDF. Serviceul este creat dupa popularul design Singleton, astfel am un constructor privat pentru fiecare clasa service, iar cand instantiez de 2 sau mai multe ori serviceul, de fapt ma refer la acelasi service. Practic imi mapez idea de Dependency Inversion, autolegand si creand un singur flow de control al aplicatiei, precum notatia @Autowired din Spring.

5. Rezultate

- Pentru testare am facut trecut prin toate operatiile din fiecare window si am testat operatiile CRUD. Mi-am creat pdf-uri multiple, am facut si cate teste pe queryuri si pe returnurile de la utils, toate reusind sa si atinga scopul. Initial mi mi-am facut cateva teste in Junit, dar era prea mult cod de scris pentru un singur test, asa ca m-am rezumat la teste in metodele pe care le voiam testate prin printf-uri sau verificare de date, sau prin simpla verificare a validitatii pe care am facut-o in packegeul validation.

6. Concluzii

1) Dezvoltari :
    a. Sa am ceva mai multe validari
    b. Imbunatarie de UI

2) Ce am invatat:
    a. am invatat JavaFx
    b. am invatat DataBinding
    c. mi-am creat o idee asupra urmatoarelor proiecte
    d. refactoring
    e. jdbc, inainte am lucrat cu cu spring si nu am folosit jdbc mai deloc
    f. reflection techniques. O metoda care poate fi uneori chiar utila pentru a face aplicatie tight coupled, si pentru a scoate din numarul mare de linii de cod prin intermediul unui DAO generic.

3) Concluzii:
    a. la proiectul urmator ar trebui sa imi fac prima oara arhitectura si sa stau mai mult sa gandesc de dinainte codul, iar abia dupa ce am gandit totul, sa incep sa implementez Este mult mai rapid si este o buna practica pentru viitor, deoarece asa imi pot tine metodele mai succinte si codul mai curat. De asemenea, cred ca voi incerca ceva nou cu UI-ul si cu validarile, deoarece nu am toate cazurile luate si pot aparea buguri. Voi face UI-ul ceva mai intuitiv.

7. Bibliografie

1) https://www.youtube.com/watch?v=agnblS47F18&t=829s -> Derek Banas Reflection Techinque Java
2) https://www.baeldung.com/javadoc
3) https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html
4) https://dzone.com/articles/layers-standard-enterprise
5) https://www.baeldung.com/java-pdf-creation