DOCUMENTAȚIE PROIECT

SISTEM DE MANAGERIERE AL LIVRARII DE MANCARE

Stancu Mihai-Cristian

1. Obiectivul temei

a. Obiectiv principal

i. De a crea o aplicatie care manageriaza comenzi, produse, client, admini
ii. De a crea o aplicatie de tip role-based pe baza credentialelor a 3 tipuri de useri: client, angajat si administrator.

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 sistemul aplicatiei:
   a. Trebuie sa am un sistem prin care aplicatia sa stie cum sa ma duca pe baza de login la UI-ul respectiv.
   b. Trebuie sa am o aplicatie care respecta niste principii de baza
2. Cerințele non-funcționale ale sistemul aplicatiei 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

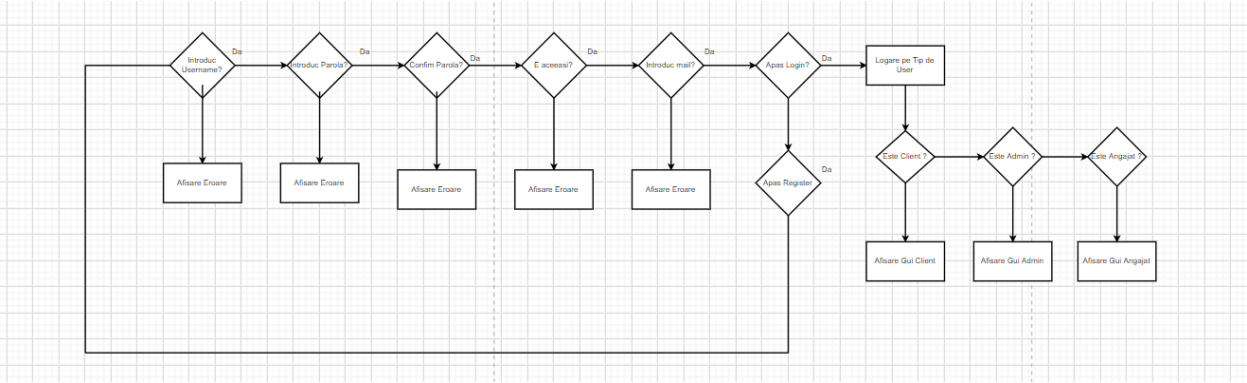
2.2 Descrierea diagramei de use-case

- Aplicatia este una role-based care putea fi facuta mult mai usor cu intermediul unei baze de date. Ma pot loga ori ca client, ori ca administrator, ori ca agajator. Fiecare tip de user are propriul lui UI, care in functie de credentiale, backendul aplicatiei se uita in ArrayList-ul de la user si vede ce tip de user am. User isi poate introduce in form numele, parola, confirmarea parolei, care este obligatoriu sa fie la fel, altfel nu va putea trece de pasul de login, si un camp de mail. Userul nu poate lasa niciun camp gol, fiind atentionat in

partea de jos a formului daca cumva a lasat ceva liber. Daca userul cumva nu are cont, se poate inregistram avand aceleasi campuri de user, parola si confirmare a parolei. Acum, din punctul de vedere al fiecarui tip de user.

- **Clientul** poate cauta alimente pe baza unor criterii precum grasime, proteina, calorii, titlu,rating, sodiu si pret. Am implementat si un motor de cautare in timp real, iar in momentul cand din choiceBox aleg un camp , in textField scriu datele si apar in timp real. Acesta poate crea order prin punerea mouselui pe un aliment, si apasarea pe buton de ADD. Dupa ce se hotaraste ca terminat, apasa pe Show Order, si i apare in zona dreapta suma totala si se creaza orderul, angajatii fiind notificati. De asemena pot forma o nota de plata sub format TXT prin ajutorul butonului BILL. Acesta poate da logout sa revina in zona de login.
- **Administratorul** poate adauga produse de tip BaseProduct sau CompositeProduct. Am 7 textFielduri in care userul poate adauga campurile pentru produs. De asemenea userul poate sterge produse pe baza unui spinner care imi indica ID-ul produsului pe care vreau sa-l sterg. Userul mai poate sterge produse pe baza unui spinner care I indica ID-ul produsului pe care vreau sa-l modifice. Acesta mai poate genera raporturi pe baza unor criterii precum : “Time interval of orders”, “Product ordered more than a specific time”, “Value of order more than a specific sum”, “Clients that ordered more than a specific number”,” Products in a day and number of times ordered”, prin intermediul unui textField in care introduce datele pentru rapoarte. Datele imi apar intr-un textArea foarte mare cand apas pe buton de generateReport. Acesta poate da logout sa revina in zona de login.
- **Angajatul** poate sa vada notificarile sale si a celorlati anjagati cand apasa pe SHOW NOTIFICATIONS. Metoda de notificari este implementata in controllerul clientului.

2.3 Descrierea use-case-urilor



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

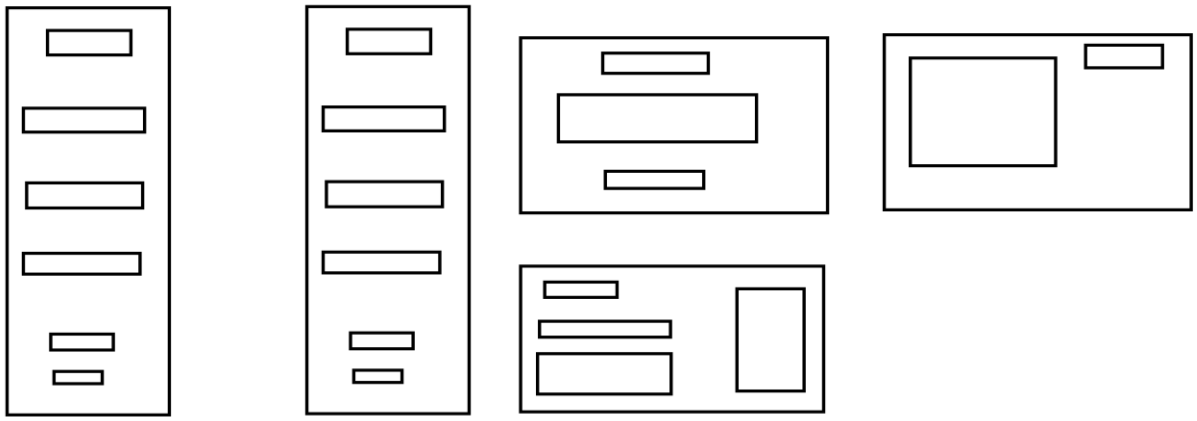
3.1 Decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete

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.

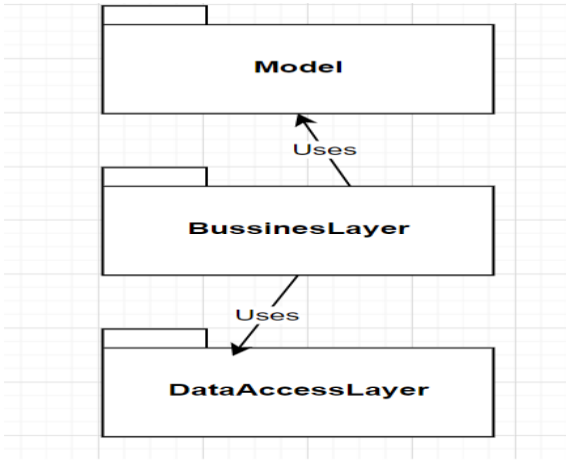
Am 4 pachete: model ,dataAccessLayer , businessLayer si presentation:

- *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.
- *DataAccessLayer* contine clasele cu operatii Crud care fac inserarea, stergerea sau update-ul in fisierele serializate.
- *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 fisierele serializate..
- 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.

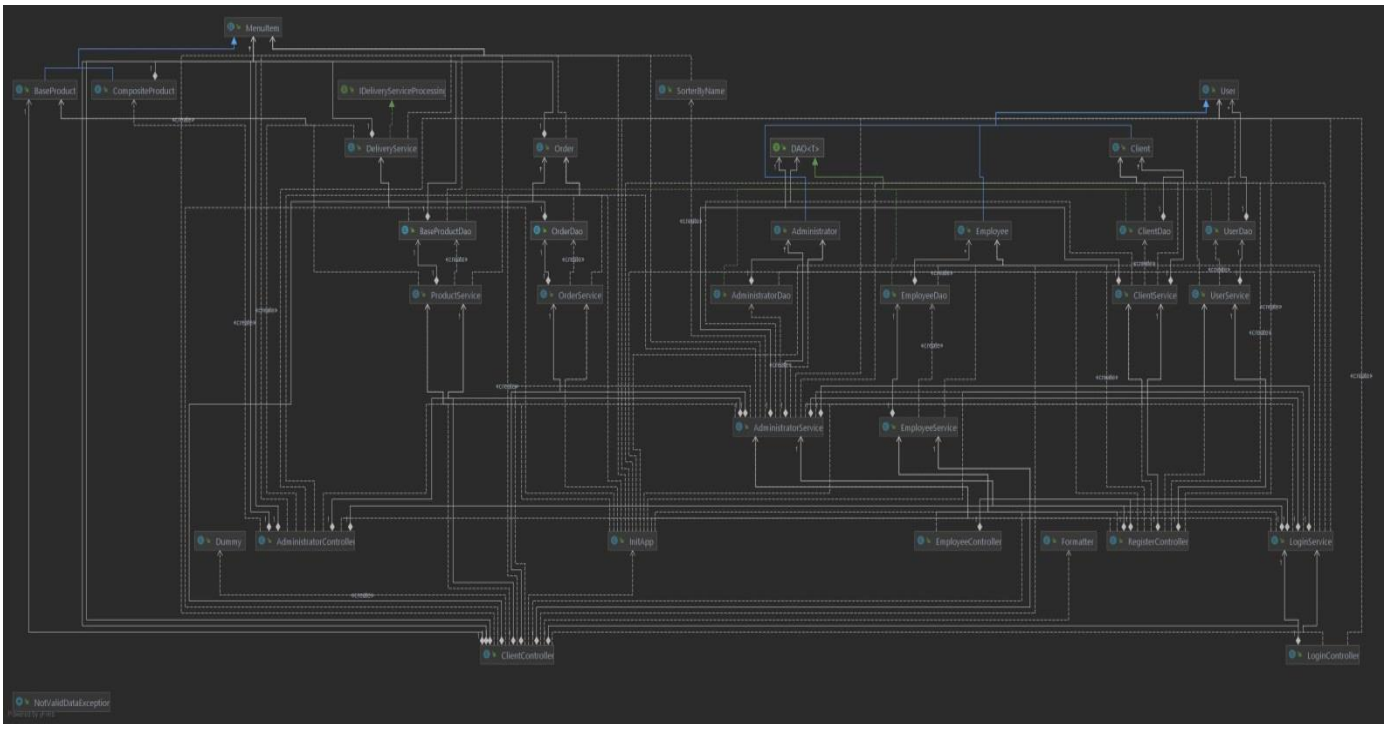
Mock



3.2 Diagrama de pachete



3.3 Diagrama de clase



4. Implementare

La implementarea temei am inceput prin analizarea a a detaliilor despre SERIALIZare, despre patternuri , 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 :
    - **LoginController:**
        - Acesta contine form style loginul aplicatiei. Este una dintre bazele aplicatiilor de backend, idea de role-based authentification este foarte importanta la orice tip de aplicatie. Am niste fielduri care sunt validate la final daca sunt ocupate sau nu. In caz de parola si confirm password nu coincid, arunc eroare. Am un arrayList de useri in clasa USER, clientii, adminii si employee extinzand aceasta clasa, iar in momentul cand dau

login, imi cauta in acea lista userul respectiv si imi da mai departe un id(“idGrasper”) si imi arata ui-ul corect.

- **RegisterController**: In cazul in care userul nu este inregisrat, acesta trebuie sa treaca prin aceasta etapa de autentificare. In cazul in care apasa din greseala pe register, poate reveni inapoi la fereastra de login printr-un buton care imi “App.setRoot(”Login”)”. In momentul in care dau register, un user imi este creat si este serializat in fiserul meu, useri pe care Il deserializez in momentul in care pornesc aplicatia. De asemena se serializeaza si in fiserele de admin sau de employee sau de client iar la fel, Il deserializez in momentul in care pornesc aplicatia. Am un combobox care contine tipul de user, iar in functie de ea, intr-un switch fac serializarea. Dupa de dau register, ma duce aplicatia inapoi in fereastra de login

- **ClientController.** Aici relatia gui-logica a clientului. In metoda de initTable(), imi initalizez tabelul cu alimentele prin PropertyValueFactory. In caz de fac operatii pe table, imi deserialieze alimentele din fisierul de alimente serializat, altfel ma folosesc de delivery service in care cu ajutorul streamurilor imi costruiesc lista de menuItem. Tot aici imi am referinta la constructSearchEngine(), astfel, am un listener la textFieldul in care scriu si care asculta la schimbarile/tastarea mea in functie de criteriul pe care l-am ales in comboBox ("title", "rating", "calories", "protein", "fat", "sodium", "price")). In metoda addProductToFinalOrder, imi adaug la order obiectul pe care am dat click din table, informatii sumare despre aliment punadu-se in textArea-ul din dreapta. In metoda calculateOrder() imi calculez suma totala a alimentelor orderului si adaug in textArea aceasta suma si ora si timpul la care este inregistrata. Dupa asta, notific fiecare angajat sa fie gata pentru order in functie de orderId , de timp si de ora. In metoda generateBill() imi creez txt-ul , luand textul din textArea. In metoda sendDataToAdministrator() imi pun orderul in order.txt prin serializare, setand date pe care le pot folosi la administrator cand generez rapoarte.

- **EmployeeController.** Aici relatia gui-logica a angajatului. Doar am un textArea in care afisez notificarile fiecarui employee in metoda showNotifications(), unde dau append.

- **AdministratorController.** Aici am relatia gui-logica a adminului. Am logica de tip database in care pot sa modific, sa adaug sau sa sterg produse, date care vor ajunge in tableViewul din clientController. Am 7 textFielduri in care introduce datele pentru produs, id-ul generanduse in functie de lastId al produselor(ultimul id). In “”choose Criteria” criterii precum : “Time interval of orders”, “Product ordered more than a specific time”, “Value of order more than a specific sum”, “Clients that ordered more than a specific number”,” Products in a day and number of times ordered”, cu ajutorul carora pot genera rapoarte. Am si spinerul id care este init cu 1 si editabil si care imi da support la operatiile crud care se fac pe baza de id(delete si modify).

2) PACKAGE BUSINESS LAYER

- Aici am serviceurile. Logica aplicatiei este implementata aici, adica totul mai putin CRUD. Am clasele de AdminService, unde am metode de generare a rapoartelor. Am folosit hashseturi si hashmapuri pentru optimizari. Am folosit si streamuri pentru filtrare. In deliveryService imi mapez pathul catre products.csv si imi extrag ,dupa ce sar primul rand, toate produsele pe care mi le pun intr-un hashset care are override metodele de equals si hashcode pe baza la nume prin intermidul unor streamuri care se duc in 7 directii, pe atribut. In loginService am logica de backed a logarii care se uita In lista de useri si imi ia in functie de id tipul de user(client, admin, angajat). Userul il iau in functie de mail, ea fiind cheia mea primara pe form. Am 3 foruri care parseaza si imi dau return la string-ul care reprezinta tipul de user. Am si validari in functia de checkCondition care verifica ca parola sa fie la fel in ambele fielduri de parole si sa nu am campuri lasate goale, altfel generez mesaj si returnez null. 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.

3) PACKAGE BUSINESS LAYER

- Am clasa DAO<T>. Este o clasa care imi creeaza endpointul aplicatiei catre fisierele serializate. Practic aici am toata logica de CRUD a aplicatiei, atingand polimorfism prin ajutorul interfetei.
- Am metodele:
  - insert este metoda care imi genereaza obiectele pe care le pun in fisierul serializat.
  - 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.
  - 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 .

4) PACKAGE UTIL

- Am clasa Formatter, care imi face tipul datei si al timpului pe care le folosesc la bill si notificari.
- Am initApp care imi face deserializarea la inceputul programului, le folosec in main.
- Am sorterByName in care imi sortez datele pentru eficientizare la adminService cand fac un raport

5. Rezultate

- Pentru testare am facut cateva printuri care sa mi apara in consola in momentul cand pornesc aplicatia sau cand apas pe butoane care fac anumite lucruri cu datele.

6. Concluzii

1) Dezvoltari :
   a. Sa am ceva mai multe validari
2) Ce am invatat:
   a. am invatat JavaFx
   b. am invatat DataBinding
   c. mi-am creat o idee asupra urmatoarelor proiecte
   d. refactoring

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=wiQdrH2YpT4&t=159s – Observer Pattern Derek Banas
2) https://www.youtube.com/watch?v=2HUnoKyC9l0 – Composite Pattern Derek Banas