

**TEHNICI DE PROGRAMARE**

**APLICAȚIE PENTRU GESTIUNEA UNUI**

**RESTAURANT**

**TEMA IV**

Student:

**Mihaela- Cristina Palamaru**

**Grupa 30228**

Prof. Indrumator:

**Dl Prof. Dan Mitrea**

1. **Obiectivul temei date**

Aceasta lucrare are ca scop insusirea cunostiintelor mai avansate de Java, imbinand interfetele grafice, cu noi cunostiinte de prelucrare si salvare a datelor cum ar fi serializarea. Studentul incepe sa inteleaga care ar fi modul de proiectare al unei aplicatii la nivel de business si nu doar din punct de vedere didactic precum in proiectele de pana acum.

Acesta trebuie sa inteleaga care ar fi nevoile unui restaurant sis a faca apel la creativitate pentru a reusi sa construiasca o aplicatie cu o interfata creative pe care actorii principali dintr- un restaurant ar trebui sa o inteleaga cu usurinta sis a o foloseasca pentru a le face munca mai usoara!

Adevarul e ca acest tip de exercitiu nu necesita doar cunostiinte de cod, ci multa rabdare in a intelege care sunt elementele constituent intr- o asemenea aplicatie si in ce fel trebuie sa iti organizezi datele astfel incat orice persoana care si- ar dori sa dezvolte in continuare aplicatia, sa o poata face cu usurinta!

Recunosc ca am rescris codul de cateva ori pentru a fi inteligibil si usor de automatizat!

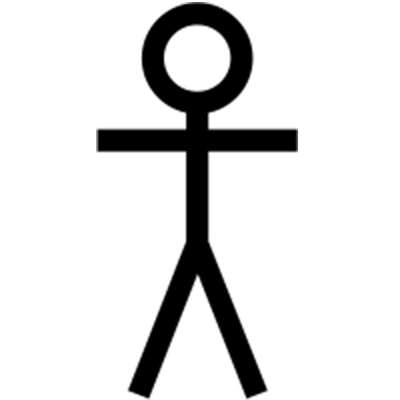
1. **Analiza problemei**

Pentru a fi mai usor de urmarit modul in care aplicatia a fost proiectata, am atasat mai jos un tabel de tip USE-CASE in care se vor regasi functionalitatile si cerintele principale ale aplicatiei.

|  |  |
| --- | --- |
| **Use case** | **Simulator cozi** |
| Actorii | Administratorul unui restaurant, chelnerul care serveste la acel restaurant si bucatarul acestor |
| Descriere | Aplicatia are rol de sistem de gestiune al unui restaurant, oferind urmatoarele functionalitati pentru utilizatorii sai:  Administrator:  - adauga un nou item la meniu  - vizualizeaza meniul existent  - editeaza itemi din meniu  - sterge itemi din meniu  Chelner:  - adauga o noua comanda in urma interactiunii cu clientii de la o anumita masa  - vizualizeaza toate comenzile active  - genereaza “nota” pentru mesele care isi doresc sa paraseasca restaurantul  Bucatar: - este notificat si are o lista cu toate comenzile care necesita preparare |
| Preconditii | Nefiind inca o aplicatie intocmai portabila, utilizatorul trebuie sa ruleze urmatoarea comanda in terminalul sau din directorul in care a salvat aplicatia:  **java -jar PT2020\_30228\_Mihaela\_Palamaru\_Assignment\_4.jar Restaurant.ser** |
| Postconditii | Utilizatorul, in general, ospatarul primeste nota de plata sub forma unui fisier intitutal “bill no. #id”, id- ul fiind generat automat la realizarea unei noi comenzi. |
| Curs normal de executie | In mod normal, dupa rularea comenzii initiale din terminal ar trebui sa se deschida cele trei ferestre specifice pentru actorii descrisi mai sus. Functionalitatile se vor manevra din butoane sau, pentru a selecta mai multi itemi care urmeaza a fi actualizati, prin apasarea casutelor de text indicate de etichete specifice la fiecare comanda!  Implementarea asigura caracteristica “ user- friendly” datorita modului foarte explicit de a indica manevrantului sau care sunt comenzile disponibile! |
| Curs alternative si exceptii/ Observatii | Exista functionalitati precum editarea itemilor de baza dintr- un produs compus care nu au fost implementate inca! In acel moment se va genera o caseta de dialog care sa anunte ca nu exista inca o implementare!  Am evitat destul de mult sa lucrez cu introducerea manuala a datelor si m- am bazat mai mult pe selectie! Singura problema care ar putea sa apara ar fi la introducerea preturilor pentru itemi sau numarul mesei la care se afla client! In aceste situatii vor aparea instantaneu casete de dialog prin care utilizatorul este atentionat ca a introdus gresit datele. |
| Triggere | Principalele triggere se afla in asculatorii care vor declansa actiuni pentru butoane.  In cazul Bucatarului, dupa adaugarea unei comenzi cu produse compuse se va adauga un mesaj special de notificare, urmand ca noua comanda sa fie adaugata in casuta acestuia. |
| Frecventa utilizare | Acest tip de aplicatie este extrem de util in orice business de tip HORECA eficientizand fluidizarea comunicarii dintre actorii unei asemenea institutii. Nu doar ca se eficientizeaza comunicarea personalului, dar creste si satisfactia clientiilor cand sunt serviti mai rapid! |

Am incercat sa expun cat mai bine idea aplicatiei prin urmatorul scenariu de business, dispus in diagrame diferite pentru cei 3 actori principali ai business- ului ( administrator, chelner, bucatar).

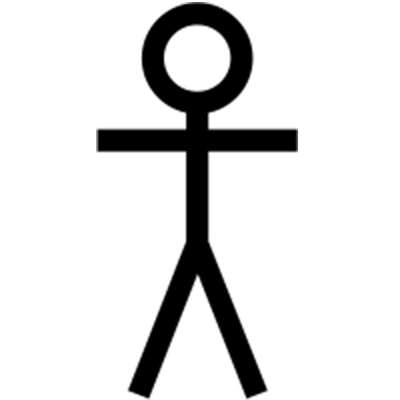
Adauga la Meniu

****

Editeaza Meniu

SISTEM

Sterge din meniu

****

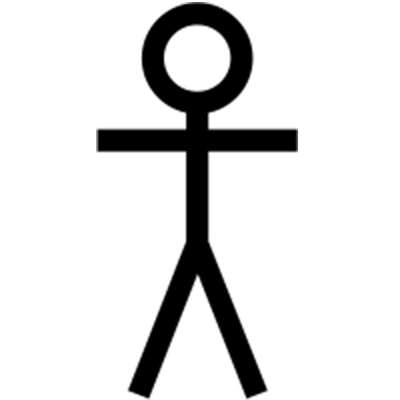
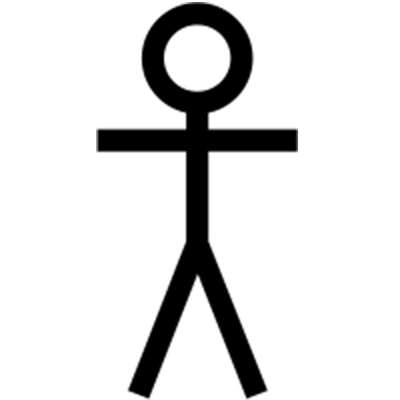
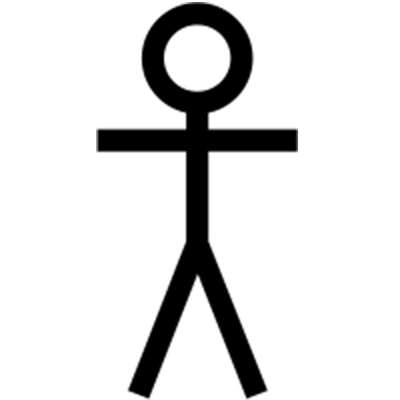
Vizualizeaza meniu

Genereaza casete si tabele prin care administratorul poate observa in timp real actiunile dorite, descrise mai sus!

ADMINISTRATOR

Adauga comanda

Notifica si afiseaza noile comenzi generate

****

Vizualizeaza comenzi active

Genereaza factura

Confirma primirea noilor comenzi

Genereaza casete si tabele prin care chelnerul poate observa in timp real actiunile dorite, descrise mai sus!

BUCATAR

SISTEM

CHELNER

1. **Proiectarea si descrierea aplicatiei**

## Aceasta aplicatie este extrem de utila pentru gestionarea unei afaceri de tip restaurant sau cafenea, avand functionalitati dezvoltate pentru 3 tipuri de persoane diferite:

1. Administratorul Business- ului – acesta va putea adauga noi itemi la meniu. Itemii pto fi de tip Base ( de exemplu, putem avea “side orders” precum chips- uri sau un anumit tip de sos sau condiment, acestea nu necesita preparare”) sau Composite care este format din ingredient de tip Base ( de exemplu, o ciorba poate fi alcatuita din morcovi, sare, apa, carne de pui, etc.. – itemi adaugati in prealabil ca pret).

**Foarte important! In specificatia temei ni se cere sa implementam metoda ComputePrice() in itemii din meniul unui restaurant! Am decis ca aceasta sa aiba rol de PriceGetter deoarece, in practica, nu imi doresc ca pretul unui CompositeItem sa fie neaparat egal cu suma base item- urilor pe care le contine. Am vrut sa ma apropii cat mai mult de realitate, pentru ca la o posibila dezvoltare, sa pot adauga gramaje din anumite cantitati.**

De asemenea, administratorul poate edita numele sau pretul pentru orice tip de produs, dar nu si ingredientele sale. Cand Administratorul decide ca sa elimine din meniu un produs de baza, toate produsele compuse vor fi de asemenea sterse!

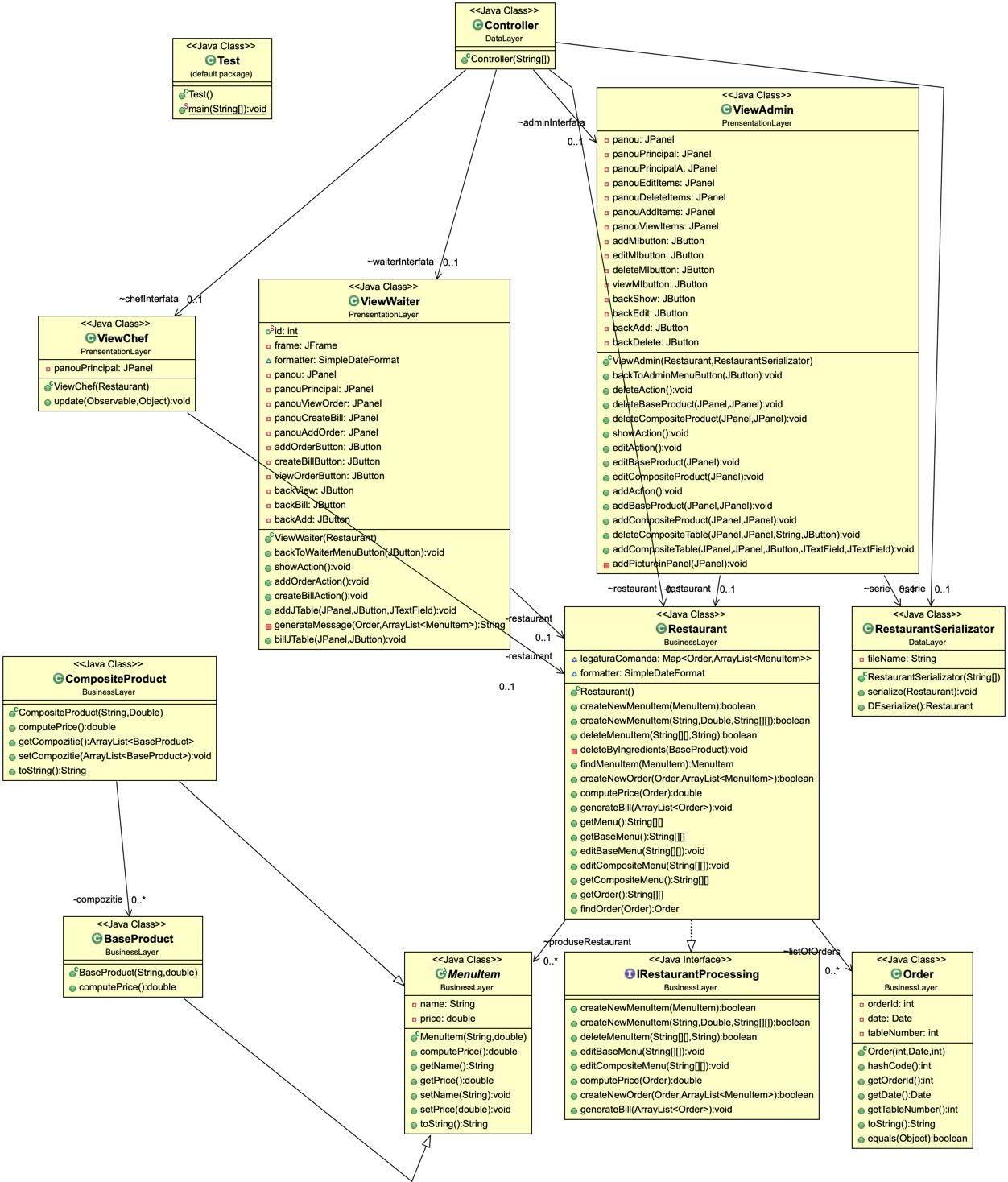
1. Chelnerul- acesta preia noile comenzi pentru o anumita masa. Itemii dintr- o comanda sunt cu usurinta alesi dintr- o lista pusa la dispozitie de administrator!

Chelnerul poate sa isi vizualizeze comenzile si, de asemenea sa emita o facture text cu idComenzii!

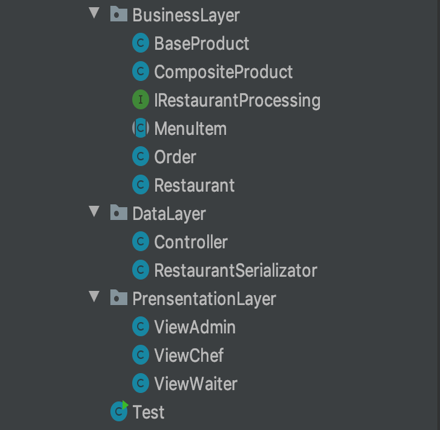
1. Bucatarul/ Barista- acesta primeste o notificare de fiecare data cand chelnerul adauga o noua comanda care necesita preparare. A necesita preparare inseamna ca respective comanda sa contina Composite Products din meniu

## 3.2.Diagrama de clase

De aceasta data am dorit sa imi generez diagrama de clase pentru a vedea o abordare mult mai colorata din partea software- urilor de dezvoltare! Aceasta varianta pare mult mai completa!

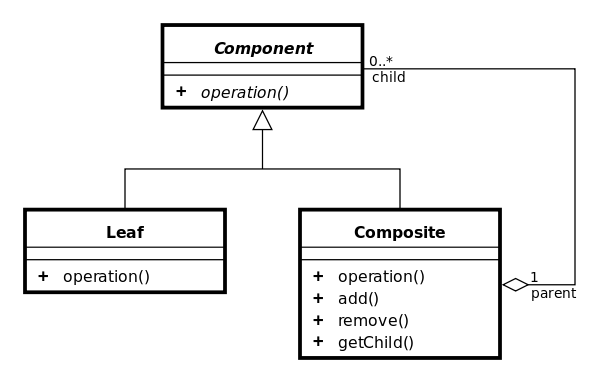


1. **Detalii de implementare & algoritmi**



De la tema trecuta am vazut ca e mult mai bine cand lucrez un pic mai organizat pe pachete, asa ca am incercat aplic acest lucru si pentru aceasta tema. Desi cerintele temei nu o cereau, m- am impotmolit un pic intre panel- urile de la interfata grafica, in incercarea de a oferi aplicatiei un aspect cat mai placut!

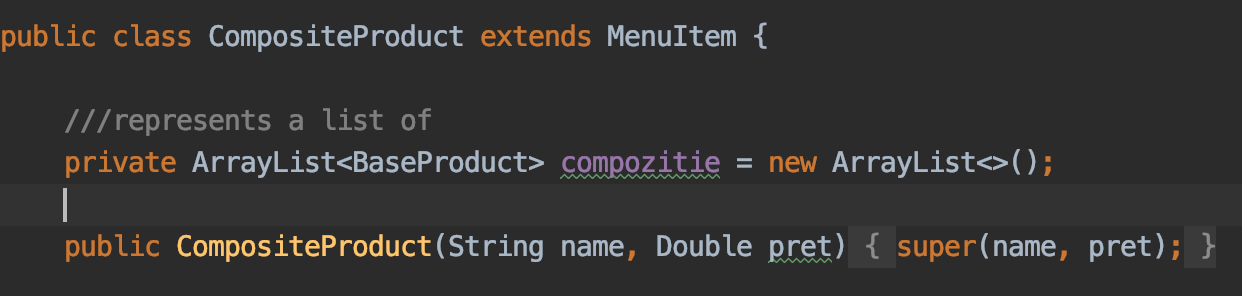
Am incercat sa comentez fiecare metoda de la ViewAdmin pentru ca unui posibil dezvoltator sa ii fie usor sa manevreze meniul!

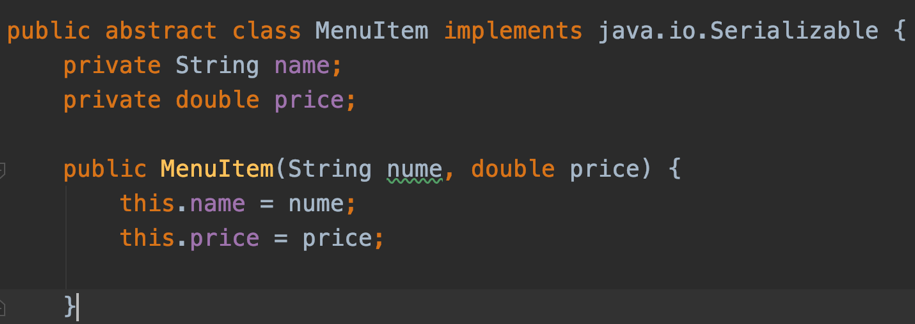
**4.1 Composite Design Pattern**

Meniul restaurantului este alcatuit din produse de baza ( side orders) si produse compuse ( din produse de baza) care necesita sa fie preparate.

In implementare am incercat sa manevrez pe cat posibil un obiect generic “MenuItem” pe care in Composite Design Pattern l- am identificat ca fiind **“componenta de baza”.**

**MenuItem** este o clasa abstracta pe care atat BaseProduct si CompositeProduct o mostnesc! Conform documentatiei, componenta de baza poate fi atat interfata, cat si clasa abstracta!



****

Clasa compusa **CompositeProduct** care are in plus o lista de ingrediente.

Clasa “componenta” din care reiese ca orice produs, indifferent de tipul lui **concret**, trebuie sa aiba un nume si un pret.

**4.2 Folosirea interfetelor grafice prin modelul MVC**

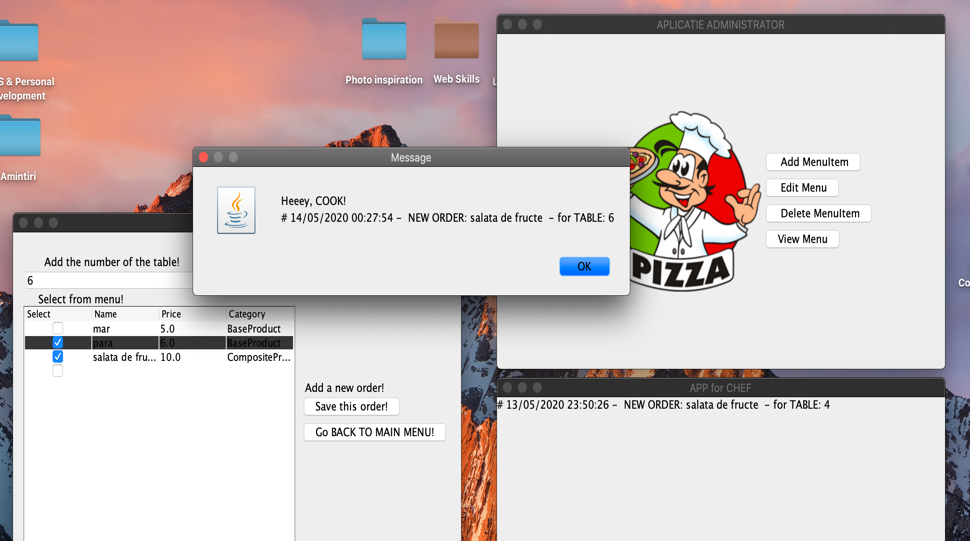
Modelul MVC reprezinta cea mai structurata cale pe care o cunoastem in present pentru a ne separa interfata grafica de functionalitatile aplicatiei in momentul de fata!

Modelul acestei aplicatii este reprezentat de clasa “Restaurant” care centralizeaza toate functionalitatile ( metodele) business- ului pentru fiecare dintre actorii sai!

Desi am lucrat cu un model centralizat, am incercat sa separ intr- o clasa separate fiecare fereastra pentru personalul restaurantului. Codul pentru interfata grafica este destul de condensate, mai ales cand se doreste o abordare mai dinamica prin crearea unor meniuri.

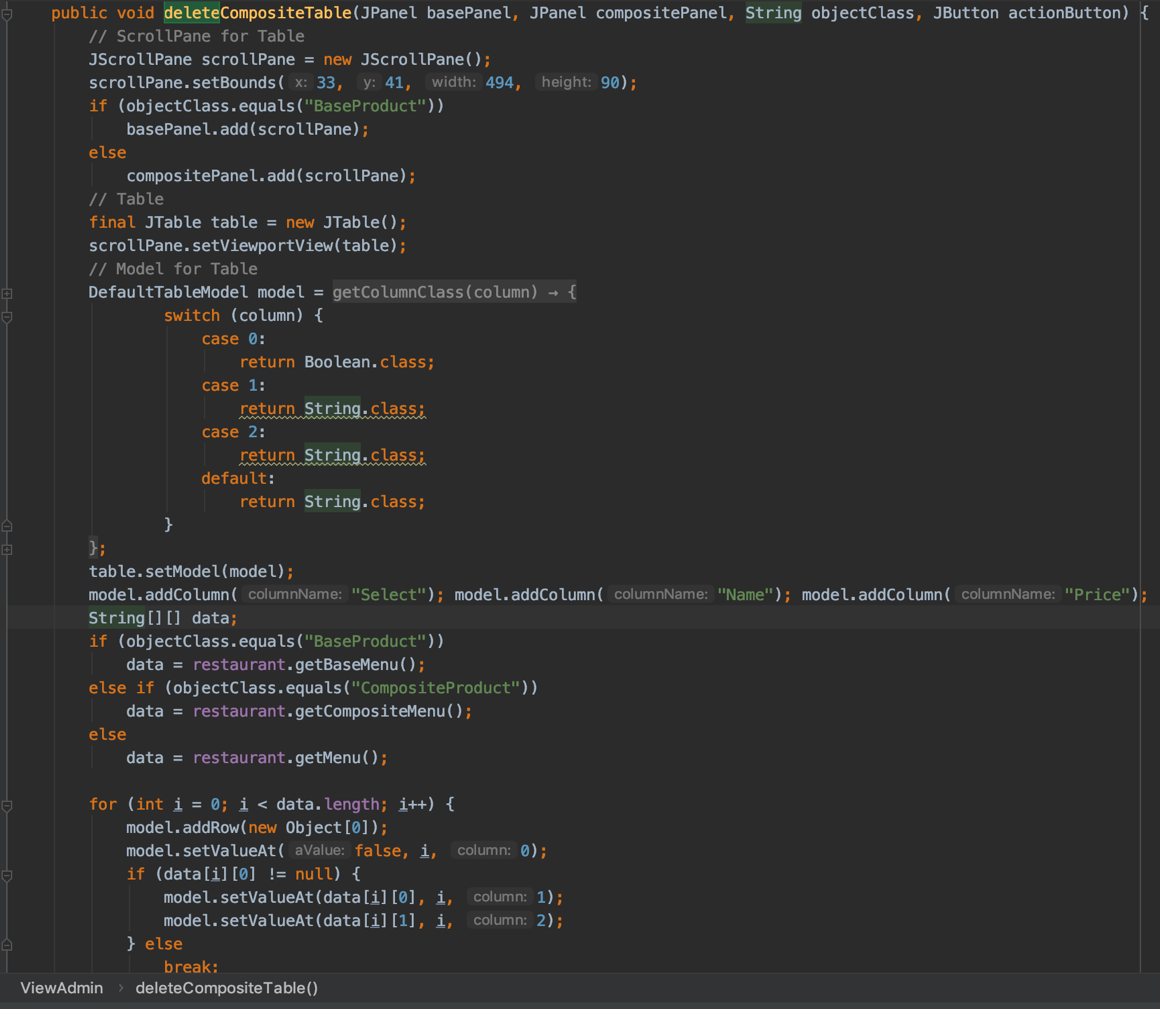
Astfel, aplicatia mea vine cu ferestre separate care se lanseaza simulatan pentru: administrator, chelner si bucatar.

Exista un singur controller in care se initializeaza toate componentele aplicatiei si care, transmite mai departe legaturle dintre clase!



Cel mai dificil de implementat element din interfetele grafice au fost tabele ce contineau casete de selectie. Am vrut sa minimizez sansele de a da de o exceptie in validarea datelor prin introducerea manuala si acesta este motivul pentru care utilizatorul o sa intalneasca foarte multe tabele in care o sa aiba posibilitatea sa selecteze itemii cu care lucreaza.

Un bun exemplu ar fi tabelul din care **administratorul** alege ce itemi vrea sa stearga, indifferent ca sunt produse de baza sau produse de baza! **M- am asigurat** ca atunci cand se sterge un produs de baza se sterg si produsele compuse aferente!



Deoarece am vrut sa pot naviga dintr- un panou in altul si datele sa se “**refresh- uiasca”** in tabele la fiecare noua actualizare, am facut ca la fiecare noua confirmare prin apasarea pe buton a unei actualizari continutul panoului de baza sa fie sters, iar utilizatorul sa fie dus in meniul parinte.

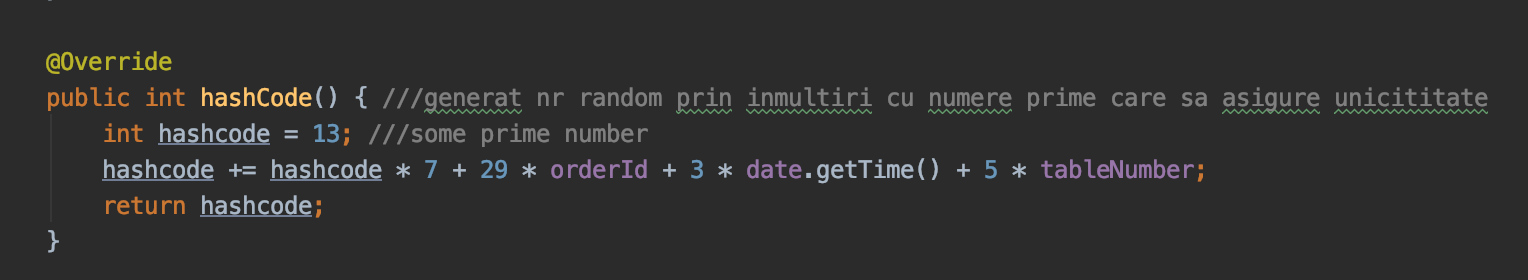
Acest lucru a fost posibil doar cu ajutorul metodelor Listener:

****

Actiunea butonului generic “**back”** duce utilizatorul la meniul principal, ascunzand panourile secundare si stergandu-le continutul pentru a le pregati de o noua posibila interactiune!

**4.3 HashMap si HashCode**

Generarea HashCode- ului am realizat- o in clasa Order prin utilizarea unei formule compusa din combinatii de numere prime ce asigura unicitatea codului.

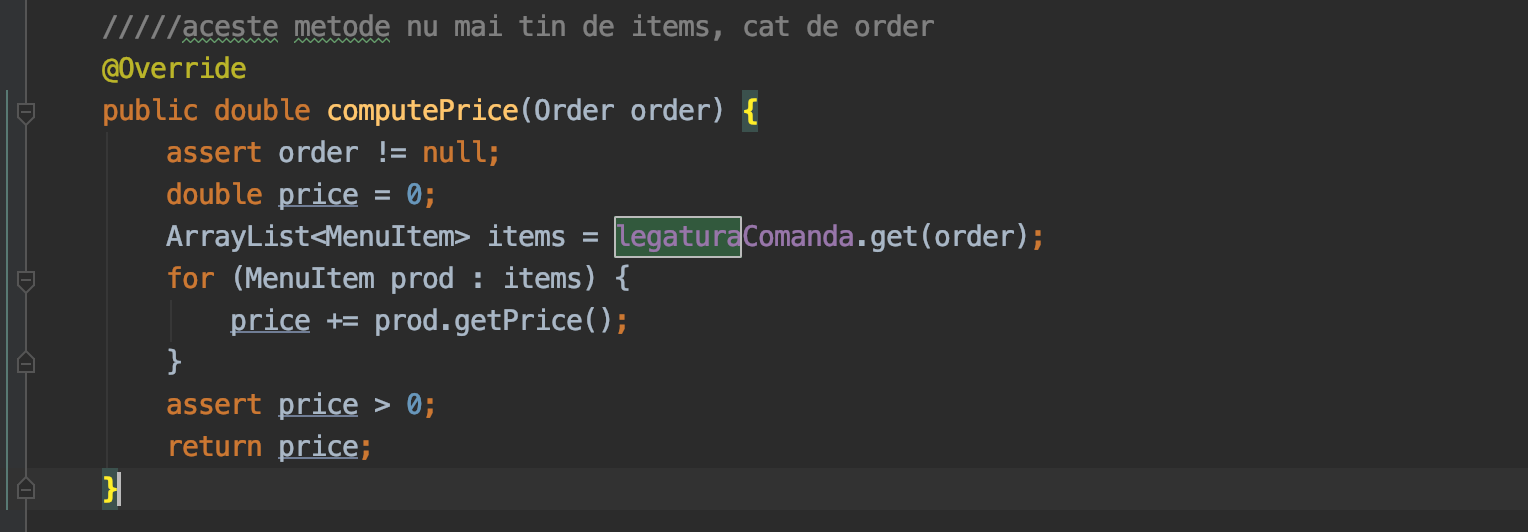


In practica, Hashmap- urile sunt folosite pentru un acces mult mai rapid la anumite date, datorita cheii sale unice ce poate fi foarte putin costisitor aflata. Daca la listele inlantuite inca ne loveam de problema cautarii intr- o lista, hash map- ul eficientizeaza foarte mult acest proces.

In cazul aplicatiei de fata, am creat un hashmap care face legatura dintre comanda continand “metadata” ( detalii fizice precum:  **numarul mesei, id- ul comenzii si timpul precis la care s-a luat comanda)** si comanda in sine   
( adica ce itemi si- au ales clientii din meniu).

O cautare secventiala ar fi extrem de costisitoare ca timp daca ar fi sa facem potrivirea dintre itemi si detaliile comenzii, dar HASHMAP- ul usureaza acest lucru.

Astfel, pentru comenzi am o lista in care pastrez datele din acestea, am un hashMap de legatura si am un ArrayList cu itemii selectati din meniu pentru fiecare comanda!

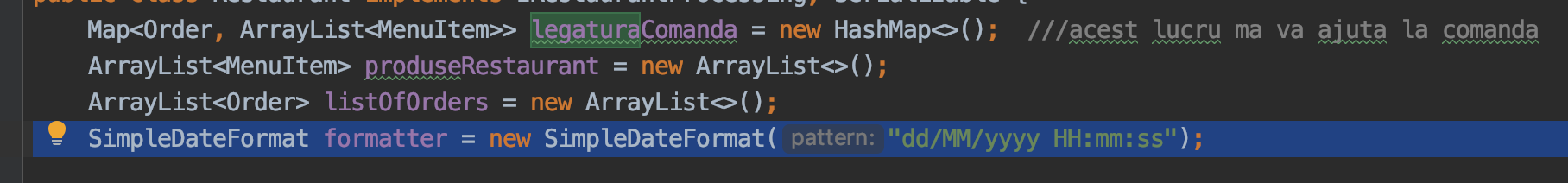


Cheia map- ului meu o reprezinta comanda, avand ca valori itemii selectati din meniu!

**4.4 Generare comanda cu ID automat si data curenta**

­La fiecare noua comanda generez un ID printr- o variabila intreaga statica care se incrementeaza! E de mentionat ca la oprirea din executie a aplicatiei id- urile o sa se reseteze la 1, considerand ca un nou chelner va executa o noua ordine.

Data unei comenzi este mereu cea curenta, pe care am putut sa o formatez cu ajutorul unui Pattern special din Java:



**4.5 Observable si Observer**

Aceste 2 concepte functioneaza impreuna, avand intotdeauana o clasa care este Observable ( in cazul nostru chelenerul) si un Observer ( care este bucatarul)

Observerul are o metoda speciala **update** pe care o suprascriem din interfata apartinand pachetului java.util:

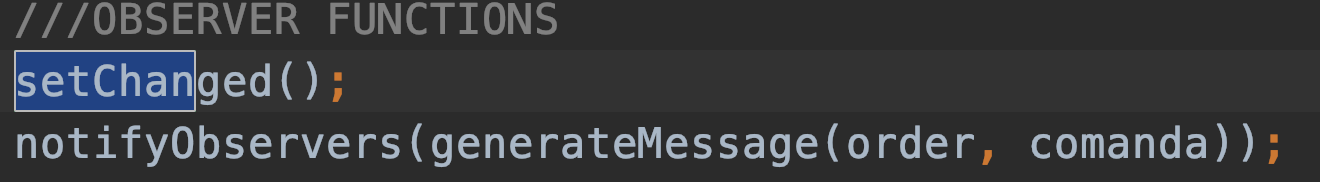


Acesta este modul prin care in fereastra bucatarului va aparea un mesaj compus de ospatar cu noua comanda introdusa si cu ce contine respectiva comanda! Mereu la coada listei va fi cea mai recenta comanda!

****

Chelnerul va transmite o noua comanda cu ajutorul listener- ului de pe butonul pe care apasa cand doreste sa adauge comanda! Mai intai se genereaza un mesaj pentru acesta daca comanda a fost generate cu success, iar apoi o alta caseta pentru bucatar, de instiintare!

Cum anunta ospatarul? Observable contine 2 metode specifice prin care interactioneaza cu Observatorul sau, si anume:



**Ca observatie tehnica! Am vazut ca notificarile pentru bucatar se genereaza corespunzator din JFrame- ul acestuia, dar comanda ajunge sa se afiseze doar dupa ce o alta noua a fost notificata! Ma gandesc ca acest lucru are de a face cu faptul ca Observable a fost considerate deprecated dupa versiunea de Java 7, considerandu- se ca nu acopera foarte bine nevoia de “dinamism” intr- o aplicatie.**

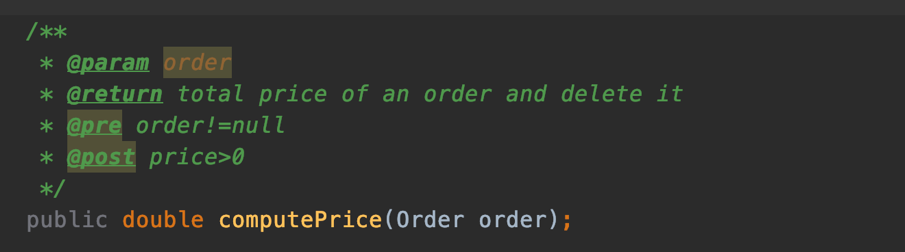
**4.6 Design by Contract: Pre si Post conditii**

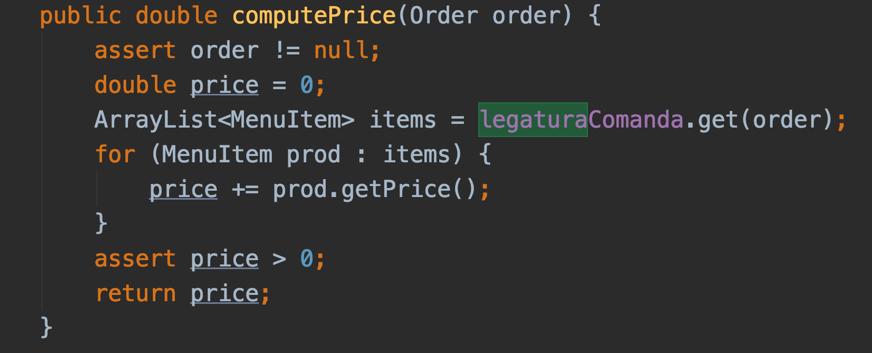
Inavriantul clasei reprezinta cumulul de conditii de inceput si de sfarsit pe care trebuie sa le contina o metoda in executia sa pentru a fi valida!

In momentul in care se proiecteaza modul in care va arata o restaurantul cu ajutorul interfetei **IRestaurantProcessing,** definescsi conditiile initiale si cele finale pe care metoda mea trebuie sa le contina!

De exemplu, in cazul **computePrice** ( metoda folosita pentru calcularea pretului total pe comanda inainte de generarea unei facturi), trebuie sa ma asigur ca comenzile trimite pentru a le fi calculate pretul **e diferit de null,** iar la final, sa ma asigur ca **pretul este mai mare ca zero.**

Acest lucru se realizeaza pe 2 planuri: initial in interfata, prin parametrii genereci, iar apoi in clasa instanta, prin **asertiuni.**

****



Prezentare pre si post conditii in interfata restaurantului

Prezentare pre si post conditii clasa prin asertiuni

**4.7 Folosirea interfetei Serializable**

Prin interfata serializable ni se ofera sansa de a salva in mod automat toate modificarile pe care dorim sa le executam in proiectul nostru. Acestea se vor pastra intr- un fisier numit: “Restaurant.ser” care va fi incarcat ca parametru o data cu incarcarea claselor.

Initial fisierul va fi “deserializat”, adica datele din acesta vor fi injectate in meniul restaurantului meu, urmand ca la fiecare modificare facuta de administrator asupra meniului sa pot imi pastrez datele( apeland metoda **serialize** in listenerii butoanelor de actualizare din AdminView), nefiind nevoit ca la fiecare rulare sa le introduc manual!



Metode in clasa RestaurantSerializator

**­­As dori sa mentionez ca in specificatia temei spune sa salvez datele din Restaurant si asa am facut**, dar am considerat ca doar datele salvate de admin sunt necesare, daca ar fi sa folosim aplicatia nu doar pentru uz didactic! Ma gandeam ca vreau ca meniul din restaurant sa fie permanenent, dar la fiecare noua rulare, un chelner nou care isi incepe tura sa aiba o zona noua de comenzi! Am considerat ca sistemul ar functiona pe zile si ca acesta s- ar inchide la finalul unei zile si atunci, clar nu arm ai fi comenzi active ale carui istoric am dori sa pastram!

**4.8 Generare fisier text**

Pentru a genera o facture este initial necesar sa obtin cu ajutorul HashMap-ului legatura dintre metadatele comenzii si produsele pe care aceasta le contine. Urmeaza sa le calculez pretul, sa generez un String in care insirui toate produsele alese la o anumita masa cu preturile aferente si mai apoi sa le transmit ca “nota de plata” sub forma unui fisier text de iesire cu ajutorul **FileWriter,** dupa cum urmeaza:

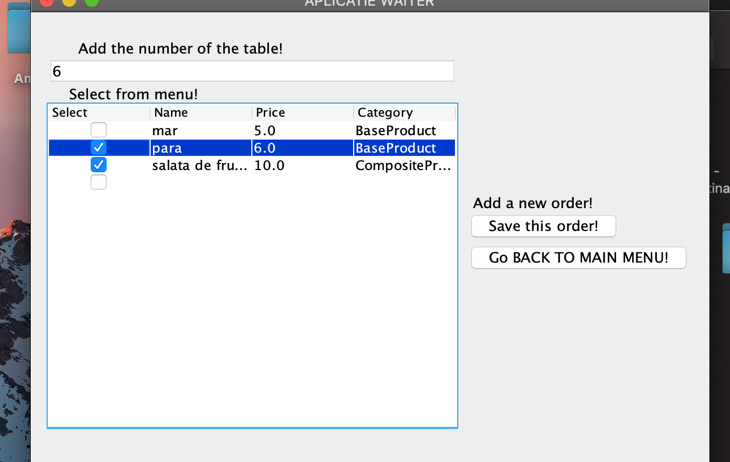


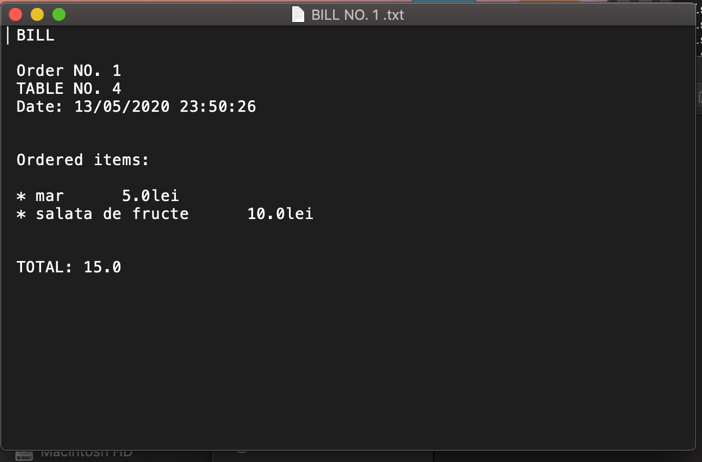
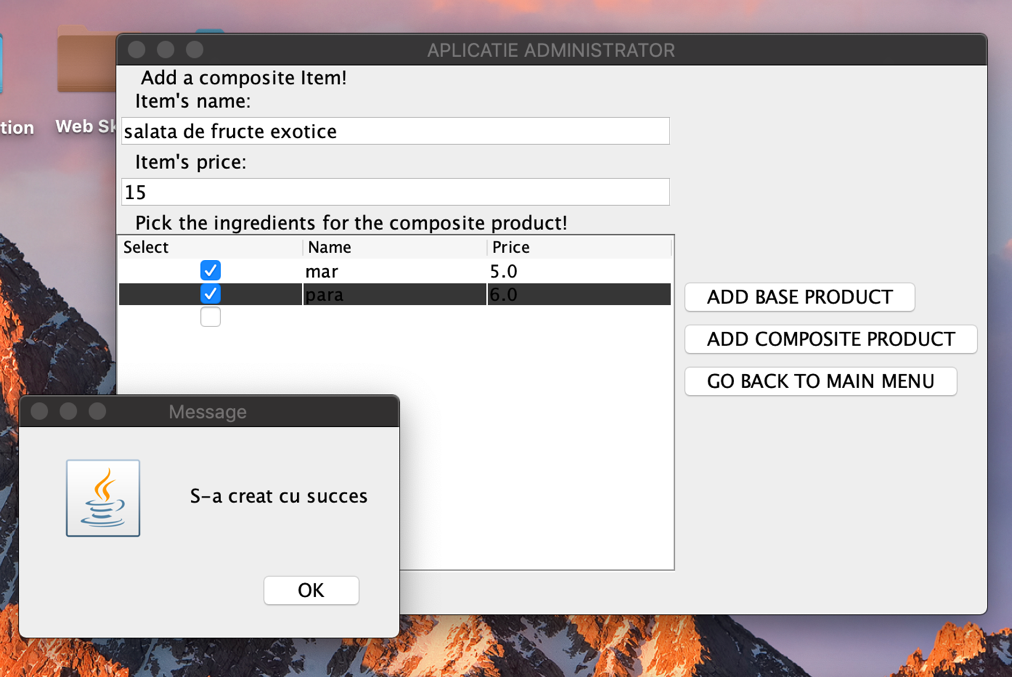
Am tinut cont ca comanda sa fie eliminata din lista dupa ce i- a fost generate factuta

1. **Rezultate**



Rezultatele practice ale interfetei reies in mod direct si dinamic din interfata grafica! Atunci cand administratorul face o adaugare de produs, automat aceasta va fi transmisa tabelelor de vizualizare si selectie a meniului sau a comenzilor pentru ferestrele aferente administratorului si ale chelenerului! Am incercat sa evit foarte mult introducerea manuala a comenzilor sau editarea produselor!

Astfel, aproape la fiecare meniu cu care unul dintre actorii business- ului interactioneaza, va da de un tabel de selectie precum in exemplul de mai jos:

****

Exista un buton specific pentru fiecare comanda care urmeaza sa se execute si, de asemenea, un fisier text in care, pentru fiecare comanda se va genera o facture pe care clientul o preia de la chelner spre casier in format “txt”.

1. **Observatii si posibilitati de dezvoltare**

La acest moment sunt multumita de modul de functionare al aplicatiei si am adaugat o imagine mai draguta pentru pagina administratorului, insa nu am gasit metodele potrivite pentru a face interfata din Swing mai atractiva! Ar trebui lucrat intr- un “Design Builder” mai actual care se ocupa de interfete interactive!

In momentul in care lucrezi ca ospatar trebuie sa fii foarte rapid si sa ai grija sa nu incurci comenzile clientilor tai! Gandind astfel, imi doresc ca pe viitor sa adaug operatiuni multiple prin care o comanda poate fi editata o data ce a fost preluata! Daca, de exemplu, o masa si- ar dori sa adauge sosuri sau bauturi la comanda lor, sau daca vor s ail atentioneze pe bucatar legat de anumite restrictii ( de exemplu, sa elimine anumite ingredient din felul de mancare comandat).

Pentru administrator as incerca ca atunci cand adauga un produs sa poata adauga si o fotografie pentru acesta pentru a fi mai usor manevrat atunci chelnerul si ospatarul vor avea de a face cu el la comenzi.

Mai mult decat atat, pentru fiecare comanda as vrea sa adaug un camp pentru timpul estimative al comenzii, astfel chelnerul o sa preia cat mai rapid. De asemenea, as vrea un buton pentru bucatar care sa anunte ca o anumita comanda este gata si poate fi preluata sau o optiune de “Intreaba chelenerul” in cazul in care un produs din meniu a fost personalizat la cererea clientului!

1. **Bibliografie**

- <https://www.youtube.com/watch?v=j442WG8YzM4>

- <https://www.baeldung.com/>

- <https://www.daniweb.com/>

- <http://www.coned.utcluj.ro/~salomie/PT_Lic/>