TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA
ROMANIA

FACULTATEA: Automatică și Calculatoare
SPECIALIZAREA: Calculatoare și Tehnologia Informației
DISCIPLINA: Tehici de programare
Grupa 30226 | An 2 semestrul 2

Student:

Galiș George-Laurențiu

Cuprins

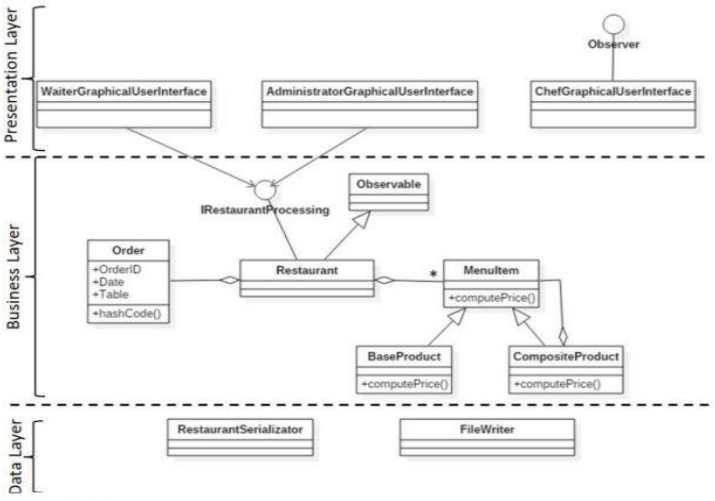
1. Obiectivul temei

Obiectivul acestei teme pentru laborator este acela de a implementa un program care ajuta la gestionarea proceselor ce au loc intr-un restaurant intr-un mod cat mai simplu posibil. Aceasta aplicatie foloseste o interfata grafica pentru a fi utilizata de oricine, si care poate fi inbunatatita pe viitor.

2. Analiza problemei

Aceasta tema poate fi folosita foarte bine si in lumea reala deoarece este foarte usor de folosit si are o intrebuinta destul de mare. Mai exact, aceasta aplicatie se adreseaza la doua tipuri de utilizatori: Administrator si Waiter(ex: client). Fiecare tip de utilizator are anumite procese pe care le poate face. De exemplu, Administratorul poate crea si informa bucatarul cand este vorba de o noua reteta, Waiter-ul poate crea comenzi pe baza retetelor create de Administrator si asa mai departe.

Crearea unui astfel de aplucatie nu e foarte dificila dar, necesita creearea unei diagrame dupa care programatorul se poate informa. De exemplu se poate lua diagrama de mai jos impartita pe mai multe Layer-e.



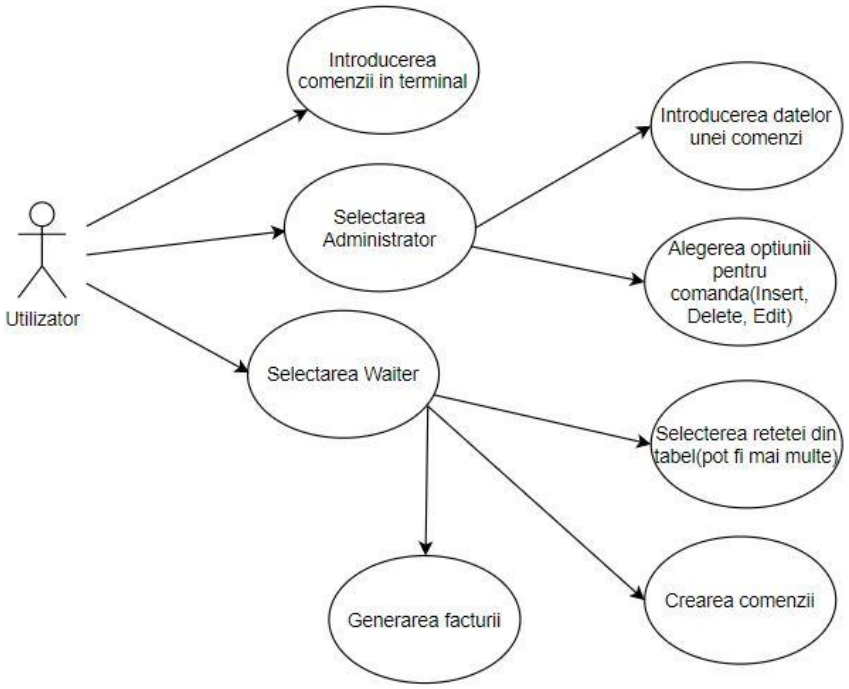
Mai jos este reprezentată diagramă use-case care ajută la înțelegerea folosirii programului și opțiunile pe care le are de ales utilizatorul. Mai întâi utilizatorul trebuie să introducă comanda pentru rularea fisierului .jar in terminal, mai exact: java -jar dupa care numele fisierului executabil. Comanda ar arata in felul urmator:

java -jar PT2020_30226_Galis_Laurentiu_Assignment_4.jar

Dupa care, utilizatorul poate alege ce tip de utilizator sa fie(Administrator sau Waiter). In functie de alegerea facuta, aplicatia va deschide ferestre diferite in care se pot face operatiile dorite pe tabele.

Daca utilizatorul va alege optiunea de Administrator, acesta va avea urmatoarele optiuni: crearea unei retete introducand de la tastatura numarul, numele si ingredientele retetei separate prin virgula. In functie de informatiile introduse de la tastatura, acesta poate introduce o reteta noua in tabel, sterge si editarea unuia sau mai multor retete din tabel prin selectarea lor cu mouse-ul.

Daca utilizatorul va alege optiunea de Waiter, acesta va avea urmatoarele optiuni: creara unei noi comenzi prin selectarea mai multor retete din tabelul creat de Administrator, genererea unei facturi la o anumita comanda in format pdf si vizualizarea tuturor comenzilor create



3. Proiectare

Cand vine vorba de implementarea propriu-zisa a acestui program, trebuie sa se tina cont de mai multi factori si sa punem urmatoarele intrebari: Ce operatii pot face fiecare tip de utilizator?, Ce tipuri de date se vor introduce in tabele?, Cum vor fi introduse datele in tabele?, Cum poate fi implemantat in Programarea Orientata pe Obiect?.

Implementarea acestei poate aveal ba baza o tabela de dispersie in care se stocheaza comenzile facute de utilizator. O tabela de dispersie este o structura eficienta de date pentru implementarea dictionarelor . In cazul cel mai defavorabil cautarea unui element intr-o tabela de dispersie poate necesita la fel de mult timp ca si cautarea unui element într-o listă inlantuita - O(n). In anumite ipoteze rezonabile, timpul necesar căutarii unui element într-o tabela de dispersie este O(1). Reprezinta o generalizare a noţiunii de tablou. Adresarea directă într-un tablou foloseste abilitatea de a examina o poziţie arbitrara în tablou într-un timp O(1). Adresarea directa este aplicabila in cazuliîn care este posibila alocarea in tablou a câte unei pozitii pentru fiecare cheie posibila. Tabela de dispersie reprezintă o alternativa eficienta la adresarea directă intr-un tablou cand numărul cheilor memorate efectiv este relativ mic faţă de numărul total de chei posibile (foloseşte în mod normal un tablou de marime proportionala cu numarul de chei memorate efectiv). Cheia nu este folosită ca indice în tablou ci indicele este calcudat pe baza cheii. Dispersia reprezinta o tehnică extrem de eficienta si practica; operaţiile de bază pentru dicţionare necesită, în medie, doar un timp O(1).

La proiectarea acestei probleme m-am folosit de tehnica „Layered Architecture”. Aceasta face mult mai usor de inteles si proiectat pasii pentru aplicatia in sine deoarece desparte problema in mai multe „straturi”. Am proiectat aceste „straturi” sub forma de pachete .Eu m-am folosit de urmatoarele:

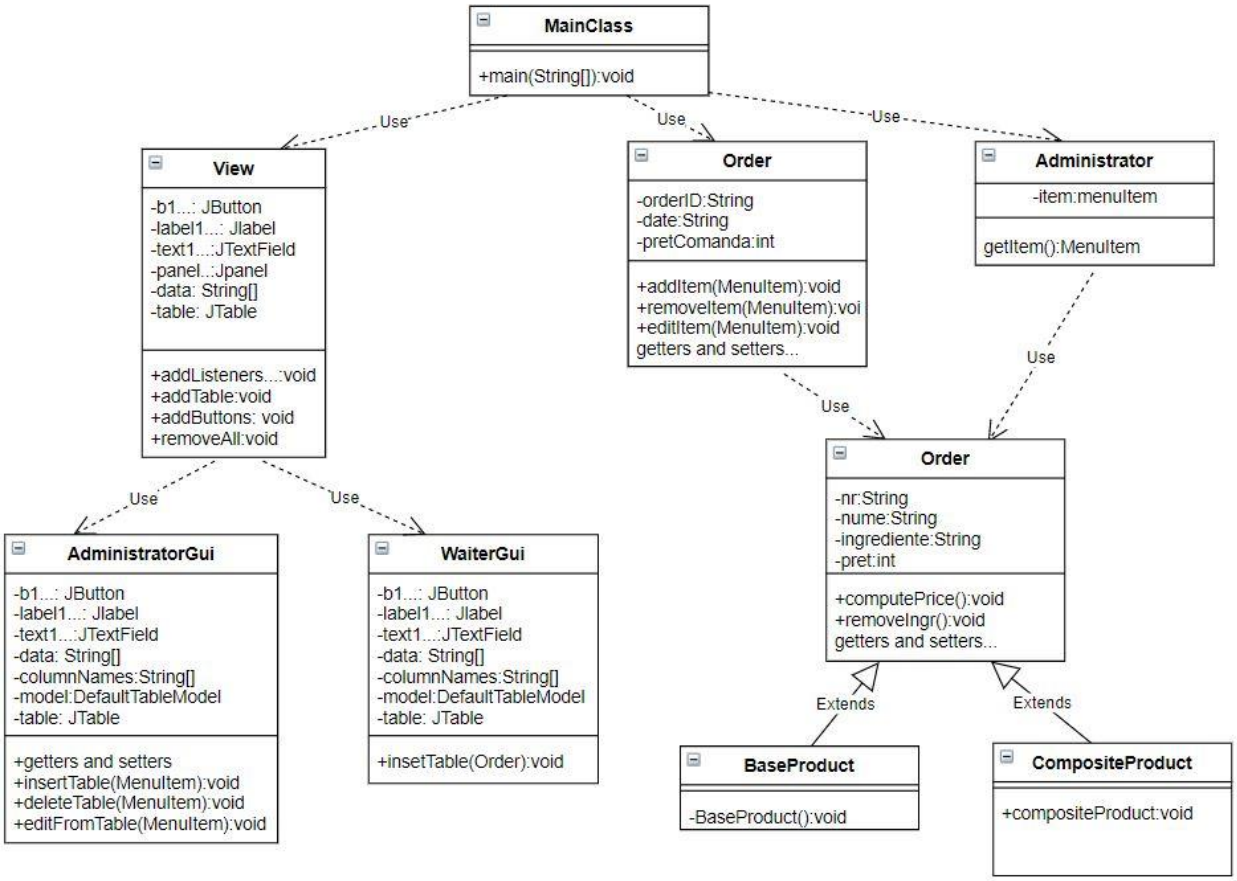
-buisnessLayer in care am pus clasa MainClass in care are loc rularea aplicatiei, BaseProduct in care se creaza retetele de baza, MenuItem care se defineste o reteta, Order care defineste o comanda si Restaurant in care se stocheaza comenzile, mai exact intr-o tabela de dispersie.

-modelLayer in care am pus clasele corespunzatoare fiecarui tabel. Acest pachet ma ajuta sa pot manipula tabelele si datele din fiecare in programul Java.

-presentationLayer in care am pus clasele care creeaza interfata grafica: AdministratorGUI care afiseaza interfata graica petntru Administrator, WaiterGUI care afiseaza interfata grafica penreu Waiter si clasa View in care se afla optiunile pentru selectarea utilizatorului;

- **Diagrama UML**

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.



4. Implementare

- **Pachetele:**

Pachetele din aceasta clasa sunt: buisnessLayer,, modelLayer, presentationLayer. Scopul fiecarui pachet l-am explicat mai sus. Acestea sunt formate pentru a impartii aplicatia in mai multe ramuri, fiecare cu un scop bine definit

- **Clasele:**

Clasa MainClass:

Este clasa în care se implementează apeleaza clasa Controller in care are loc executia efectiva a programului. Functia ‚public static void main' are ca argument un vectori de tipul String(String[] args) unde args sunt argumentele ce vor fi citite.

Clasa BaseProduct:

Aceasta clasa este formata dn constructorul care genereaza automat in tabel diferite retede default

Clasa MenuItem:

Aceasta clasa are metode care creaza o reteta si genereaza preturi pentru fiecare reteta.

Clasa Order:

Aceasta clasa are metode care creaza o comanda si genereaza operatii pe comenzile facute: addItem, removeItem, editItem

Clasa Restaurant:

Aceasta clasa este formata din tabela de dispersie in care se stocheaza comenzile facute

Clasa AdministratorGUI:

Aceasta clasa contine metode care genereaza interfata grafica pentru administrator. In aceasta clasa se creaza butoanele, text field-urile si tabelul pentru operatuiile facute de Administrator: create nre item, delete item, delete item.

Clasa View:

Aceasta clasa ajuta la citirea si transformarea datelor din fisierul .txt. transformarea se face cu ajutorul metodelor parseInt din clasa Integer. Pe langa acest lucru, aceasta clasa foloseste clasa Operations pentru a face operatii pe baza de date conform instructiunii citite din fiseirul.txt

Clasa WaiterGUI:

Aceasta clasa contine metode care genereaza interfata grafica pentru waiter. In aceasta clasa se creaza butoanele, si tabelul pentru operatiile facute de Waiter: create new order, generate bill, view menu, view item

- **Metodele utilizate în clasa Order:**

Aceasttea sunt metodele care fac operatiile pe retetele create de administrator. Operatiile sunt facute pe os structura de date numita ArrayList<MenuItem> items

```
public static void addItem(MenuItem item) {
        items.add(item);
}
public static void removeItem(MenuItem item) {
        int j = 0;
        for (MenuItem i : items) {
                if (i.nr == item.nr) {
                        items.remove(j);
                        break;
                }
                j++;
        }
}
public static void editItem(MenuItem item, MenuItem item1) {
        int j = 0;
        for (MenuItem i : items) {
                if (i.nr == item1.nr) {
                        items.set(j, item);
                        break;
                }
                j++;
        }
}
```

```
public int computePrice() {
        Random rand = new Random();

        int min=10;
        int max=100;
        pret=rand.nextInt((max - min) + 1) + min;
        Order.pretComanda+=pret;
        return pret;

    }
```

- **Metoda utilizata in clasa Restaurant**

Aceasta clasa are doar o metoda care introduce datele intr-o tabela de dispersie in care cheia este un obiect de tipul Order si datele sunt de tipul ArrayList<MenuItem>

```
public static void addOrder(Order o, ArrayList<MenuItem> i) {
    order.put(o, i);
}
```

- **Metodele utilizate în clasa AdministratorGUI:**

Mai jos se poat obeva operatiile ce au loc la apasarea butonului de inserare a unei retete si metodele specifice

```
b1.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String nr = AdministratorGUI.getText1();
            String nume = AdministratorGUI.getText2();
            String ingrediente = AdministratorGUI.getText3();

            Administrator a = new Administrator(nr, nume, ingrediente);
            AdministratorGUI.insertTable(a.getItem());
            a.getItem().ingrediente.clear();
        }
    });
public static void insertTable(MenuItem item) {

    data[0] = item.nr;
    data[1] = item.nume;

    for (String i : item.ingrediente) {
        data[2] += i + ", ";
    }
```

```
                model.addRow(new Object[] { data[0], data[1], data[2] });
                Order.addItem(item);
                data[0] = "";
                data[1] = "";
                data[2] = "";
        }

public static void deleteFromTable() {

                MenuItem item = new MenuItem();

                item.nr = (String) table.getValueAt(table.getSelectedRow(), 0);

                int numRows = table.getSelectedRows().length;
                for (int i = 0; i < numRows; i++) {
                        model.removeRow(table.getSelectedRow());
                }
                Order.removeItem(item);
        }
```

- **Metodele utilizate în clasa WaiterGui:**

Mai jos se poat obeva operatiile ce au loc la apasarea butonului de inserare a unei comenzi si metodele specifice

```
b1.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {

                    id++;
                    DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
                    LocalDateTime now = LocalDateTime.now();
                    String data = dtf.format(now).toString();
                    Order order = new Order();
                    order.setDate(data);
                    order.setOrderID(String.valueOf(id));
                    WaiterGUI.insertTable(order);
                    JOptionPane.showMessageDialog(new JFrame(), "Ai de
preparat o noua comanda", "Bucatar",

        JOptionPane.INFORMATION_MESSAGE);
                }
            });



public static void insertTable(Order order) {

            data[0] = order.getOrderID();
```

```
            data[1] = order.getDate();

            int numRows =
AdministratorGUI.getTable().getSelectedRows().length;
            for (int i = 0; i < numRows; i++) {
                  Order.pretComanda+=MenuItem.getPret();
                  Restaurant.addOrder(order, Order.getItems());
            }

            model.addRow(new Object[] { data[0], data[1] });

            data[0] = "";
            data[1] = "";

      }
```

- **Metodele utilizate în clasa View:**

  Aceste metode ajuta la introducerea elementelor in freme-ul principal din clasa View. Mai exact panel-urile, butoanele, text field-urile si tabelele

```
            void addAdministratorListener(ActionListener a) {
                        b2.setSelected(false);
```

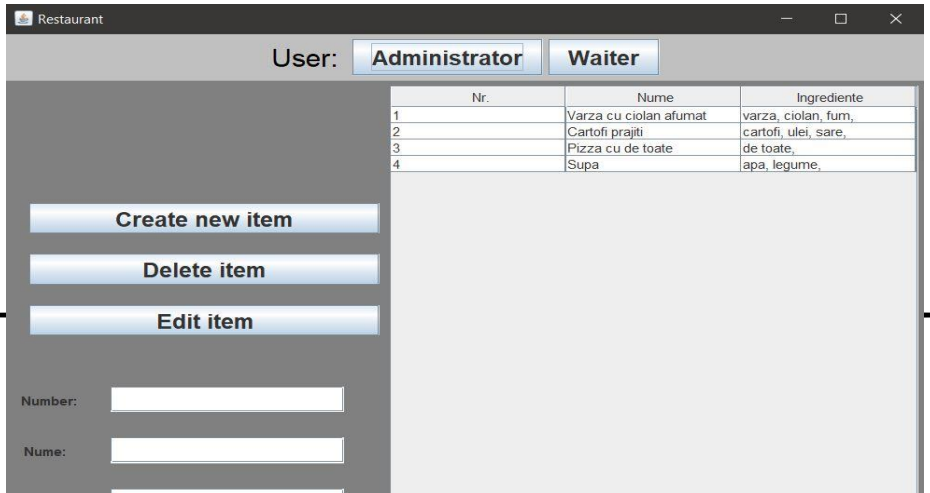
```
        b1.addActionListener(a);
}

void addWaiterListener(ActionListener a) {
        b2.setSelected(false);
        b2.addActionListener(a);
}

void addTable(Component a) {
        panel3.removeAll();
        frame.repaint();
        frame.setSize(802, 700);
        frame.setSize(801, 700);
        panel3.add(a);
}

void addButtons(Component a) {
        frame.repaint();
        frame.setSize(801, 700);
        panel2.add(a);
}

void removeAll() {
        panel2.removeAll();
        //panel3.removeAll();
```

5. Rezultate

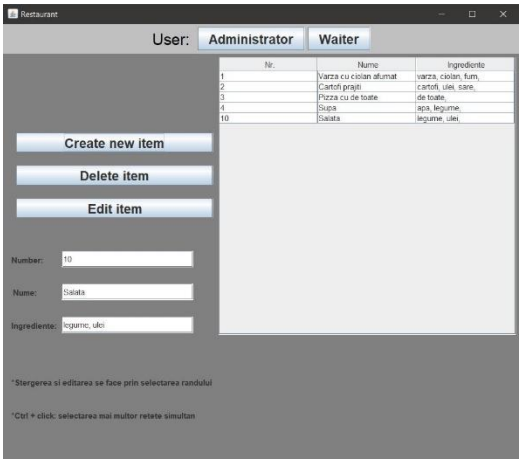
Dupa rularea comenzii in consolo se va deschide urmatoarea fereastra ce reprezinta interfata grafica a aplicatiei

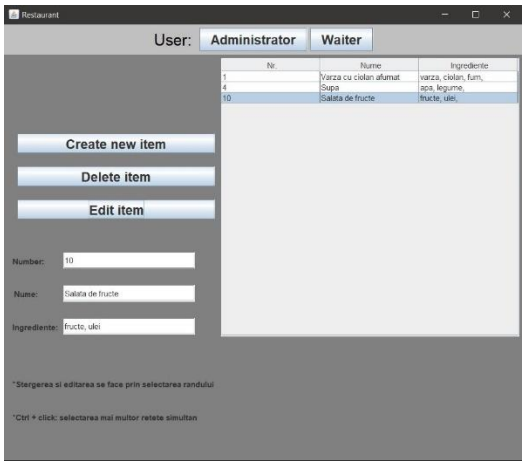
Aici se poate face algerea de catre utilizator dupa care se va deschide fereastra specifica fiecaruia.

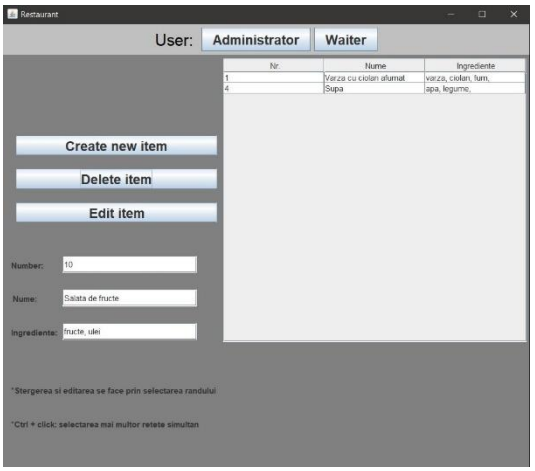
Interfata grafica pentru Administrator este urmatoarea:



Operatiile facute pe retete sunt reprezentate mai jos

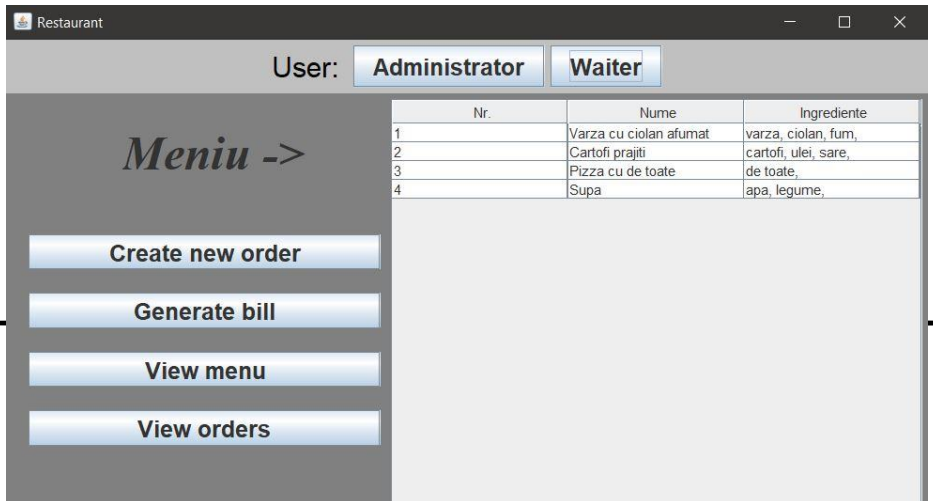




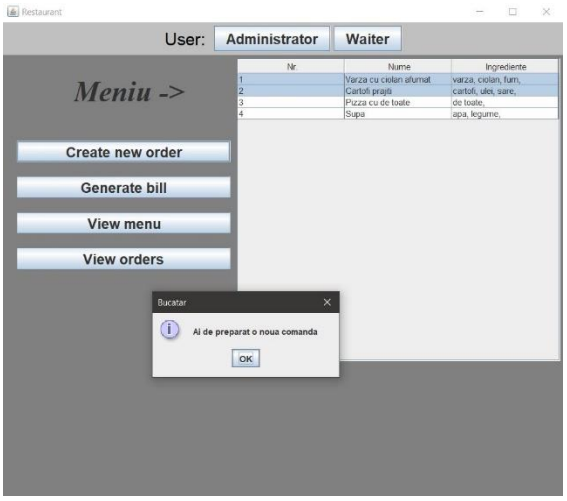


Adaugare Stergere Editare

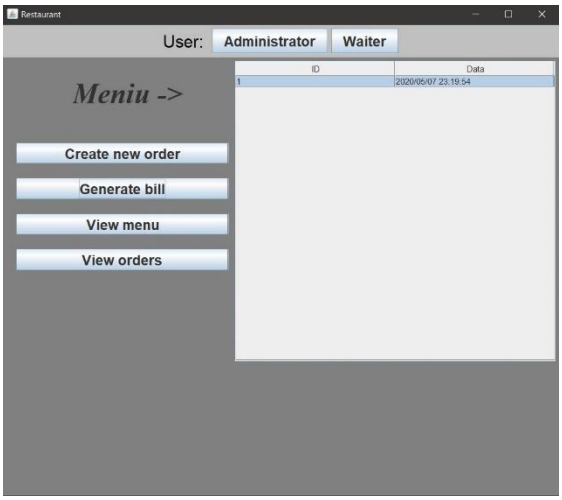
Interfata grafica pentru Waiter este urmatoarea:



Operatiile pe comenzi sunt urmatoarele

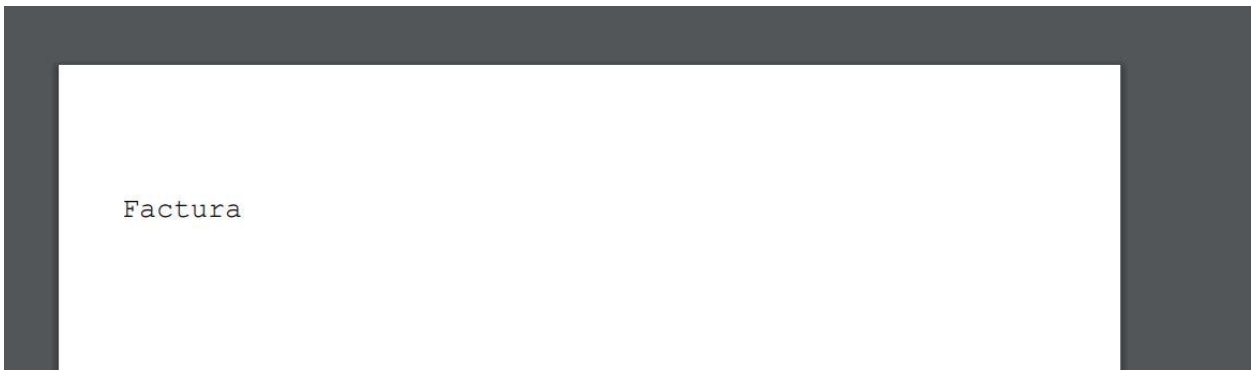


Crearea unei noi comenzi



Vizualizarea comenzilor

Factura generata:



6. Concluzie

În concluzie, acest proiect a avut scopul de a aprofunda cunoștiințele în tot ce înseamnă limbajul Java, implementarea paradigmelor OOP, folosirea si

intelegerea bazelr de date si citirea din fisier. Mai mult de atât cred ca a avut și scopul de a reaminti tehnicile de programare învățate semestrul trecut despre tot ce înseamnă sintaxa si aranjarea unui cod ca să poată fi înțeles foarte ușor și de un alt programator.

7. Webografie

1. http://youtube.com
2. https://www.wikipedia.org/
3. https://www.w3schools.com/
4. https://www.geeksforgeeks.org/