**DOCUMENTATIE TEMA 4**

**RESTAURANT MANAGEMENT**

**SYSTEM**

Cuprins:

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Rezlutate
6. Concluzii
7. Bibliografie

**1. Obiectivul temei**

Obiectivul acestei teme este de a crea o aplicatie care sa gestioneze activitatiile care se desfasoare intr-un restaurant. Un restaurant este condus de un manager, are angajati mai multi membrii care fac parte din staff si care servesc clientii, fiecare restaurant are de asemenea si un meniu.

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

**2.1 Analiza problemei**

Rezolvarea acestei probleme se poate imparti in mai multe etape :

- Accesul la comenzile restaurantului pentru fiecare membru al personalului

- Vizualizarea meniului si a listei de preturi

- Procesarea comenzilor de catre manager

- Afisarea rezultatelor procesate cu ajutorul interfetei grafice cu utilizatorul

De aici rezulta ca aplicatia noastra va trebui sa mentina aceasta structura segmentata.

Un alt motiv pentru care am dori sa mentinem aceasta structura este nevoie de securizare a datelor. Astfel fiecare membru al personalului are un ID si o parola pentru a se putea loga in sistem.

**2.2 Modelare**

Informatiile despre produse, personal si meniu vor fi stocate in fisiere text. Fiecare membru al personalui poate sa faca doar anumite actiuni:

-Managerul poate sa modifice meniul, preturile si sa vizualizeze comenzile efectuate de membrii personalului. Acesta poate de asemenea genera si facture pentru comenzile deja incheiate. Factura va fi generata automat in format .txt in folderol dataFiles🡪reports

-Personalul poate prelua comenzile efectuate de clientii restaurantului si incheia nota de plata

**2.3 Scenarii**

In cazul acestei aplicatii exista urmatoarele scenarii :

- Managerul are posibilitatea de a vedea meniul restarantului si poate sa il editeze, poate adauga, modifica sau sterge membrii staffului, si poate accesa numarul de vanzari si platiile care au avut loc.

Parolele pentru manager se gasesc in documentul manager.txt din dataFiles si sunt urmatoarele:

* **7,password,you,manager**
* **1000,java,Hello,Java**

**-**Membrii din staff pot accesa meniul restaurantului si efectua comenzi, acestia sunt cei care incheie si comenzile pentru a putea fi generata factura in formatul .txt

Membrii din staff si parolele pentru fiecare se gasesc in documentul staff.txt fon dataFiles si sunt urmatoarele:

* **1111,password,Teo,Test**
* **1212,password2,Teofil,Chira**
* **1234,password3,Dan,Hera**
* **3333,david,David,Gatta**
* **4321,pass,Ioana,Goia**
* **4322,test,test,test**
* **4444,hogehoge,Horea,Nem**

**3. Proiectare**

Din interfata grafica utilizatorul poate avea acces foarte usor la fiecare operatiune pe care doreste sa o faca folosind datele de logare specificate mai sus.

Aplicatia este usor de inteles si de utilizat de catre oricine.

**4.Implementare**

Pentru a implementa aplicatia am folosit urmatoarele clase importante:

-**Controller\_GUI** aici sunt implementate modalitatile de logare in sistem precum si editarea meniului restaurantului

**public** **boolean** loginCheck( **int** inputID, String inputPassword, **boolean** isManager)

{

String searchClassName;

//---------search user----------

Staff rStaff = cDatabase.findStaffByID(inputID);

**if**(isManager) searchClassName = "Manager";

**else** searchClassName = "Employee";

**if**( rStaff != **null**)//User data is found

{

//Search only particular target(Manager or Employee)

**if**( rStaff.getClass().getName().equalsIgnoreCase(searchClassName))

{

**if**(rStaff.getPassword().equals(inputPassword))

{

**if**(rStaff.getWorkState() == 0) //Not clocked in yet

{

rStaff.clockIn();

}

**if**(isManager)

{

userType = ***USER\_MANAGER***;

cView.changeMode(cView.***MODE\_MANAGER***);

}

**else**

{

userType = ***USER\_EMPLOYEE***;

cView.changeMode(cView.***MODE\_EMPLOYEE***);

}

currentUserID = inputID;

currentUserName = rStaff.getFullName();

cView.setLoginUserName(currentUserName); //show user name on the view

**return** **true**; //Login success

}

**else**

{

setErrorMessage("Password unmatch.");

//printErrorMessageToView("Password unmatching.");

**return** **false**;

}

}

**else** //ID is found but type(Manager or Employee) is umnatching

{

setErrorMessage("Not found.");

//printErrorMessageToView("Not found.");

**return** **false**;

}

}

**else**

{

setErrorMessage("Not found.");

**return** **false**;

}

}

**public** **boolean** addNewMenuItem(**int** newID, String newName, **double** newPrice, **byte** menuType)

{

MenuItem rMenuItem = cDatabase.findMenuItemByID(newID);

**if**(rMenuItem != **null**)

{

setErrorMessage("ID:" + newID + " is arleady used by " + rMenuItem.getName());

**return** **false**;

}

**try**

{

cDatabase.addMenuItem(newID, newName, newPrice, menuType);

**return** **true**;

}

**catch**(DatabaseException de)

{

setErrorMessage(de.getErrMessage());

**return** **false**;

}

}

**public** **boolean** updateMenuItem(**int** id, String newName, **double** newPrice, **byte** menuType)

{

**try**

{

cDatabase.editMenuItemData(id, newName, newPrice, menuType);

**return** **true**;

}

**catch**(DatabaseException de)

{

setErrorMessage(de.getErrMessage());

**return** **false**;

}

}

**public** **boolean** deleteMenuItem(**int** id)

{

MenuItem rMenuItem= cDatabase.findMenuItemByID(id);

**if**(rMenuItem == **null**)

{

setErrorMessage("Menu item ID:" + id + " is not found.");

**return** **false**;

}

**try**

{

cDatabase.deleteMenuItem(rMenuItem);

}

**catch**(DatabaseException de)

{

setErrorMessage(de.getErrMessage());

**return** **false**;

}

**return** **true**;

}

**public** MenuItem getMenuItemData(**int** menuItemID)

{

**return** cDatabase.findMenuItemByID(menuItemID);

}

-**UserInterface\_GUI** aici sunt implementate propriu-zis toate metodele care pot fii vizibile de catre utilizator in interfata grafica

**public** UserInterface\_GUI(Controller\_GUI rController)

{

**this**.rcController = rController;

**this**.con = getContentPane();

// Set frame

setTitle("Restaurant Management System");

setBounds(***WINDOW\_X***, ***WINDOW\_Y***, ***WINDOW\_WIDTH***, ***WINDOW\_HEIGHT***);

setResizable(**false**);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

createMasterPanelConpornents();

currentUserName = "";

setLoginUserName(currentUserName);

//------- Create main content panels

//Home panel

homePanel = **new** JPanel();

homeImage = **new** JLabel();

//Random generator = new Random();

**int** i = **new** Random().nextInt(4)+1;

homeImage.setHorizontalAlignment(SwingConstants.***CENTER***);

homeImage.setVerticalAlignment(SwingConstants.***CENTER***);

homeImage.setIcon(**new** ImageIcon("images/home" + i + ".jpg"));

homePanel.add(homeImage);

homePanel.setBackground(Color.***WHITE***);

mainPanel.add("Home", homePanel);

cLoginPanel = **new** LoginPanel();

mainPanel.add("Login", cLoginPanel);

cMenuListPanel = **new** MenuListPanel();

mainPanel.add("MenuList", cMenuListPanel);

cOrderListPanel = **new** OrderListPanel();

mainPanel.add("OrderList", cOrderListPanel);

cOrderDetailPanel = **new** OrderDetailPanel();

mainPanel.add("OrderDetail", cOrderDetailPanel);

cEmployeeListPanel = **new** EmployeeListPanel();

mainPanel.add("EmployeeList", cEmployeeListPanel);

cEditEmployeePanel = **new** EditEmployeePanel();

mainPanel.add("EditEmployee", cEditEmployeePanel);

cMenuManagementPanel = **new** MenuManagementPanel();

mainPanel.add("MenuManagement", cMenuManagementPanel);

cEditMenuItemPanel = **new** EditMenuItemPanel();

mainPanel.add("EditMenuItem", cEditMenuItemPanel);

cTotalSalesPanel = **new** TotalSalesPanel();

mainPanel.add("TotalSalesPanel", cTotalSalesPanel);

cPaymentPanel = **new** PaymentPanel();

mainPanel.add("PaymentPanel", cPaymentPanel);

changeMode(***MODE\_ANONYMOUS***);

}

**private** **class** EditMenuItemPanel **extends** JPanel **implements** ActionListener

{

**private** JLabel lblMenuItemID;

**private** JTextField tbMenuItemID;

**private** JLabel lblName;

**private** JTextField tbName;

**private** JLabel lblPrice;

**private** JTextField tbPrice;

**private** JLabel lblType;

**private** JComboBox cbType;

**private** JButton btnOK;

**private** **boolean** isUpdate;

**public** EditMenuItemPanel()

{

GridBagLayout gbLayout = **new** GridBagLayout();

**this**.setLayout( gbLayout);

GridBagConstraints gbc = **new** GridBagConstraints();

lblMenuItemID = **new** JLabel("Menu item ID:");

lblMenuItemID.setPreferredSize(**new** Dimension(100, 30));

gbc.gridx = 0;

gbc.gridy = 0;

gbc.anchor = GridBagConstraints.***WEST***;

gbLayout.setConstraints(lblMenuItemID, gbc);

**this**.add(lblMenuItemID);

tbMenuItemID = **new** JTextField(4);

tbMenuItemID.setInputVerifier(**new** IntegerInputVerifier(1,10000));

gbc.gridx = 1;

gbc.gridy = 0;

gbLayout.setConstraints(tbMenuItemID, gbc);

**this**.add(tbMenuItemID);

lblName = **new** JLabel("Menu item name:");

lblName.setPreferredSize(**new** Dimension(100, 30));

gbc.gridx = 0;

gbc.gridy = 1;

gbLayout.setConstraints(lblName, gbc);

**this**.add(lblName);

tbName = **new** JTextField(20);

gbc.gridx = 1;

gbc.gridy = 1;

gbLayout.setConstraints(tbName, gbc);

**this**.add(tbName);

lblPrice = **new** JLabel("Menu item price:");

lblPrice.setPreferredSize(**new** Dimension(100, 30));

gbc.gridx = 0;

gbc.gridy = 2;

gbLayout.setConstraints(lblPrice, gbc);

**this**.add(lblPrice);

tbPrice = **new** JTextField(10);

tbPrice.setInputVerifier(**new** DoubleInputVerifier(1,10000));

gbc.gridx = 1;

gbc.gridy = 2;

gbLayout.setConstraints(tbPrice, gbc);

**this**.add(tbPrice);

lblType = **new** JLabel("Menu item type:");

lblType.setPreferredSize(**new** Dimension(100, 30));

gbc.gridx = 0;

gbc.gridy = 3;

gbLayout.setConstraints(lblType, gbc);

**this**.add(lblType);

String[] combodata = {"Main", "Drink", "Alcohol", "Dessert"};

cbType = **new** JComboBox(combodata);

gbc.gridx = 1;

gbc.gridy = 3;

gbLayout.setConstraints(cbType, gbc);

**this**.add(cbType);

btnOK = **new** JButton("OK");

btnOK.addActionListener(**this**);

gbc.gridx = 0;

gbc.gridy = 4;

gbc.gridwidth = 2;

gbLayout.setConstraints(btnOK, gbc);

**this**.add(btnOK);

}

-**Staff** este clasa care creeaza membrii personalului

**public** **abstract** **class** Staff

{

**private** **int** ID;

**private** String lastName;

**private** String firstName;

**private** String password;

**private** **byte** state;

**private** Order[] orderList;

//protected byte workState; //0:not active 1:active (on wark) 2:finish work

**protected** Date startWorkTime;

**protected** Date finishWorkTime;

**protected** **double** wageRate;

//------------------------------------------------------------

// constructor

//------------------------------------------------------------

**public** Staff()

{

ID = 0;

lastName="";

firstName="";

startWorkTime = **null**;

finishWorkTime = **null**;

state = 0;

}

**public** Staff( **int** newID, String newLastName, String newFirstName, String newPassword)

{

setID( newID);

setLastName(newLastName);

setFirstName(newFirstName);

setPassword( newPassword);

startWorkTime = **null**;

finishWorkTime = **null**;

state = 0;

//workState = 0;

}

//------------------------------------------------------------

// setter

//------------------------------------------------------------

**protected** **void** setID( **int** newID)

{

**this**.ID = newID;

}

**protected** **void** setLastName(String newLastName)

{

**this**.lastName = newLastName;

}

**protected** **void** setFirstName(String newFirstName)

{

**this**.firstName = newFirstName;

}

**protected** **void** setPassword(String newPassword)

{

**this**.password = newPassword;

}

**protected** **void** setWorkState(**byte** newState)

{

**this**.state = newState;

}

-**Manager** clasain care este creat managerul restaurantului

**public** **class** Manager **extends** Staff

{

**private** **static** **final** **double** ***MINIMUM\_RATE*** = 100.0;

**public** Manager()

{

**super**();

}

**public** Manager( **int** newID, String newLastName, String newFirstName, String newPassward)

{

**super**(newID, newLastName, newFirstName, newPassward);

wageRate = ***MINIMUM\_RATE***;

}

**public** **void** setWageRate(**double** newRate)

{

**if**(wageRate < ***MINIMUM\_RATE***)

newRate = ***MINIMUM\_RATE***;

wageRate = newRate;

}

**public** **double** culculateWages()

{

**if**(getWorkState() != ***WORKSTATE\_FINISH***)

**return** 0;

**return** **this**.wageRate;

}

}

-**Employee** clasa care are la baza fiecare angajat

**public** **class** Employee **extends** Staff

{

**private** **static** **final** **double** ***MINIMUM\_RATE*** = 13.5;

**public** Employee()

{

**super**();

}

**public** Employee( **int** newID, String newLastName, String newFirstName, String newPassward)

{

**super**(newID, newLastName, newFirstName, newPassward);

wageRate = ***MINIMUM\_RATE***;

}

**public** **void** setWageRate(**double** newRate)

{

**if**( newRate < ***MINIMUM\_RATE***)

wageRate = ***MINIMUM\_RATE***;

**else**

wageRate = newRate;

}

**public** **double** culculateWages()

{

**return** wageRate \* culculateWorkTime();

}

}

-**Database** este clasa care manipuleaza datele din sistem precum stergere, adaugare sau modificarea membrilor din staff sau a meniului

**public** **void** editStaffData(**int** staffID, String newPassword, String newFirstName, String newLastName) **throws** DatabaseException

{

Staff rStaff = findStaffByID(staffID);

rStaff.setPassword(newPassword);

rStaff.setLastName(newLastName);

rStaff.setFirstName(newFirstName);

**try**

{

**if**(rStaff **instanceof** Manager)

//if(rStaff.getClass().getName().equalsIgnoreCase("Manager"))

updateStaffFile(**true**);//update manager file

**else**

updateStaffFile(**false**);//update employee file

}

**catch**(DatabaseException dbe)

{

**throw** dbe;

}

}

**public** **void** editStaffData(Staff rStaff, **int** which, String newData) **throws** DatabaseException

{

**switch**(which)

{

**case** ***EDIT\_LAST\_NAME***:

rStaff.setLastName(newData);

**break**;

**case** ***EDIT\_FIRST\_NAME***:

rStaff.setFirstName(newData);

**break**;

**case** ***EDIT\_PASSWORD***:

rStaff.setPassword(newData);

**break**;

**default**:

**break**;

}

**try**

{

**if**(rStaff **instanceof** Manager)

//if(rStaff.getClass().getName().equalsIgnoreCase("Manager"))

updateStaffFile(**true**);//update manager file

**else**

updateStaffFile(**false**);//update employee file

}

**catch**(DatabaseException dbe)

{

**throw** dbe;

}

}

**public** **void** deleteStaff(Staff rStaff) **throws** DatabaseException

{

**boolean** isManager = **false**;

staffList.remove(rStaff);

//if(rStaff.getClass().getName().equalsIgnoreCase("Manager"))

**if**(rStaff **instanceof** Manager)

isManager = **true**;

**try**

{

updateStaffFile(isManager);

}

**catch**(DatabaseException dbe)

{

**throw** dbe;

}

}

**public** **void** addStaff(**int** newID, String newPassward, String newFirstName, String newLastName, **boolean** isManager) **throws** DatabaseException

{

Staff newStaff;

**if**(isManager)

newStaff = **new** Manager(newID, newLastName, newFirstName, newPassward);

**else**

newStaff = **new** Employee(newID, newLastName, newFirstName, newPassward);

staffList.add(newStaff);

**if**(newStaff **instanceof** Manager)

//if(newStaff.getClass().getName().equalsIgnoreCase("Manager"))

isManager = **true**;

**try**

{

updateStaffFile(isManager);

}

**catch**(DatabaseException dbe)

{

**throw** dbe;

}

}

**public** **void** editMenuItemData(MenuItem rMenuItem, **int** which, String newData) **throws** DatabaseException

{

**try**

{

**switch**(which)

{

**case** ***EDIT\_ITEM\_NAME***:

rMenuItem.setName(newData);

**break**;

**case** ***EDIT\_ITEM\_PRICE***:

**double** newPrice = Double.*parseDouble*(newData);

**if**(newPrice < 0)

**throw** **new** DatabaseException("Price must be positive number");

**else**

rMenuItem.setPrice(newPrice);

**break**;

**case** ***EDIT\_ITEM\_TYPE***:

**byte** newType = Byte.*parseByte*(newData);

**if**(newType < MenuItem.***MAIN*** || MenuItem.***DESSERT*** < newType)

**throw** **new** DatabaseException("Type must be between " + MenuItem.***MAIN***

+ " and " + MenuItem.***DESSERT*** + ")");

**else**

rMenuItem.setType(Byte.*parseByte*(newData));

**break**;

**default**:

**break**;

}

}

**catch**(DatabaseException e)

{

**throw** e;

}

**catch**(Exception e)

{

**throw** **new** DatabaseException(e.getMessage());

}

}

**public** **void** setMenuItemAsPromotionItem(MenuItem rMenuItem, **double** price)

{

rMenuItem.setState(MenuItem.***PROMOTION\_ITEM***, price);

}

**public** **void** resetMenuState(MenuItem rMenuItem)

{

rMenuItem.resetState();

}

**public** **void** deleteMenuItem(MenuItem rMenuItem) **throws** DatabaseException

{

menuList.remove(rMenuItem);

/\*try

{

updateMenuFile();

}

catch(DatabaseException dbe)

{

throw dbe;

}\*/

}

**public** **void** addMenuItem(**int** newID, String newName, **double** newPrice, **byte** newType) **throws** DatabaseException

{

MenuItem newMenuItem = **new** MenuItem(newID, newName,newPrice, newType);

menuList.add(newMenuItem);

Collections.*sort*(menuList, **new** MenuItemComparator());

/\*try

{

updateMenuFile();

}

catch(DatabaseException dbe)

{

throw dbe;

}\*/

}

//---------------------------------------------------------------

// Order

//---------------------------------------------------------------

**public** **int** addOrder(**int** staffID, String staffName)

{

**int** newOrderID = ++todaysOrderCounts;

Order newOrder = **new** Order(staffID, staffName);

newOrder.setOrderID( newOrderID);

orderList.add(newOrder);

**return** newOrderID;

}

**public** **void** addOrderItem(**int** orderID, MenuItem rItem, **byte** quantity)

{

Order rOrder = findOrderByID(orderID);

rOrder.addItem(rItem, quantity);

}

**public** **boolean** deleteOrderItem(**int** orderID, **int** index)

{

Order rOrder = findOrderByID(orderID);

**if**(rOrder == **null**)

**return** **false**;

**return** rOrder.deleteItem(index);

}

Utilizatorul poate folosi o multime de butoane pentru a selecta operatia pe care doreste sa o efectueze. Dupa ce a este logat in sistem utilizatorul are acces la mai multe operatii in sitem, in functie de membrul personalului care este logat.

**5. Rezultate**

Ca rezultat final am obtinut o aplicatie care gestioneaza activitatea unui restaurant si poate extrage, insera, updata si sterge orice tip de date in functie de cerintele utilizatorului.

Din punctul de vedere al utilizatorului, aplicatia este usor de folosit avand o serie de butoane cu notatie explicita din punctul de vedere al functionarii si cu celalalte campuri folosite pentru introducerea datelor , la fel de explicit evidentiate. Rezultatele sunt usor de interpretat, ele fiind afisate chiar in interfata grafica sau poti fi vizualizate prin generarea facturii.

Din punctul de vedere al unui programator, aplicatia este usor de mentinut fiin posibila detectarea unei probleme direct de la sursa si rezolvarea acesteia fara modificari aditionale aduse altor clase sau pachete. Totodata, aplicatia este usor de adaptat nevoilor oricarui mediu prin simpla adaugare a unor metode specifice in clasele de. Interfata grafica simpla este usor de adaptat la aceste schimbari.

**6. Concluzii, dezvoltari ulterioare**

In concluzie, dupa terminarea acestei teme pot spune ca am invatat bazele folosirii limbajului Java in gestionarea unui restaurant. Prin folosire se intelege:

-am invatat cum sa fac panouri diferite pentru logare in functie de persoana care doreste sa acceseze informatiile

-procesarea acestor date si modelarea lor dupa o serie de obiecte model cu ajutorul carora se va crea mai departe logica aplicatiei.

Motivatia pentru folosirea acestui design este evidenta luand in considerare usurinta mentinerii functionarii optime a programului si a depanarii in cazul erorilor sau a generarii de rezultate neasteptate.

Se pot remarca o gama larga de dezvoltari ulterioare, aceasta aplicatie reprezentand doar un schelet al unei aplicatii apropiate de o folosinta comerciala. Totusi, aducand imbunatatiri interfetei cu utilizatorul astfel facilitand o manipulare mai rapida a datelor si crescand numarul de moduri in care pot fi manipulate datele, aceasta aplicatie ar putea fi folosita de orice restaurant care se ocupa cu gestionarea unui numar mare produse, clienti si comenzi, o centralizare a datelor fiind evident necesara. Pe langa acestea, adaugand suport pentru manipulari simultane a datelor folosind threaduri putem facilita folosirea aplicatiei de un numar mare de utilizatori simultan, singura restrictie fiind capacitatea serverelor care gazduiesc datele. De exemplu, pornind de la ideea de baza a temei, gestionarea unui restaurant, putem facilita folosirea aplicatiei de catre fiecare angajat al restaurantului simultan, astfel permitand o vizualizare in timp real al produselor, a clientilor si adaugarea si executarea comenzilor.

**7.Bibliografie**

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf>

<https://limetray.com/blog/restaurant-management-system/>