Universitatea Tehnică ”Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare
Domeniul Calculatoare și Tehnologia Informației
Specializarea Calculatoare Tehnologia Informației

BAZE DE DATE

Numele și prenumele, grupa **Mesina Maria 1307A**

Profesor indrumator **Cătălin Mironeanu**

1. Introducere

Proiectul realizat implementeaza functionalitata unui calculator caloric. Acesta ofera facilitati precum stocarea inregistrarilor care contin alimentele consumate, cu posibilitatea ulterioara de modificare / stergere, afisarea inregistrarilor dupa zi, calculul numarului necesar de calorii si calculul sumei de calorii efectiv consumate, adaugarea produselor noi.

La rularea aplicatiei, utilizatorul este intampinat de pagina de logare, unde se poate conecta cu un account deja existent sau poate crea unul nou. In cazul logarii reusite, acesta este redirectionat pe o alta pagina, unde poate vizualiza, edita si sterge datele inregistrare. In coltul stanga-sus este dispus un widget calendar, pentru a permite accesul la inregistrarile din alta zi decat cea curenta. Se permite accesul la date calendaristice in intervalul [*data crearii accountului, data curenta*]. In dreapta acestui widget se afla 2 casete: una contine numarul de calorii recomandat pe zi, iar cealalta - numarul actual consumat, conform inregistrarilor din ziua respectiva. Valoarea recomandata de calorii este calculata cu formula Mifflin St Jeor:

- Men BMR = *(10 x weight in kg) + (6.25 x height in cm) - (5 x age in years) + 5*
- Women BMR = *(10 x weight in kg) + (6.25 x height in cm) - (5 x age in years) – 161,*

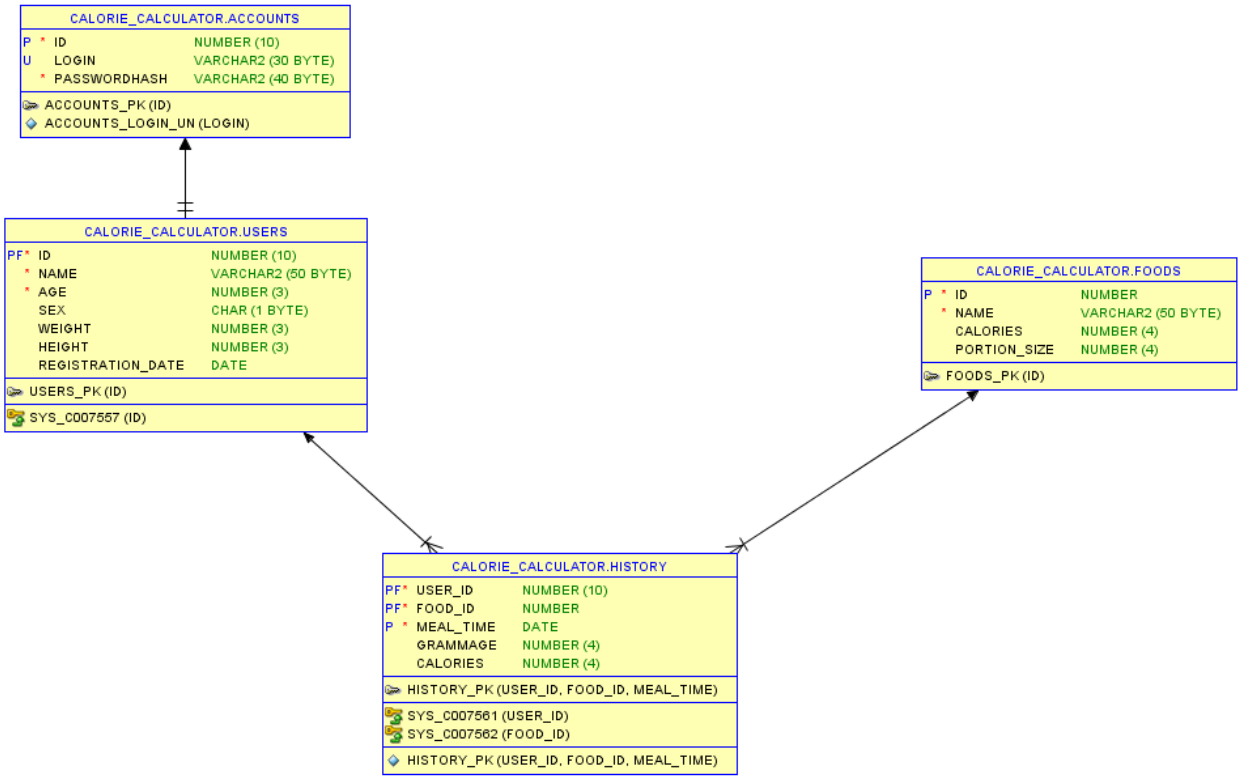
2.Tehnologii utilizate

Aplicatia este scrisa in limbajul de programare python, in spate datele sunt stocate intr-o baza de date Oracle, iar interfata grafica pusa la disponibilitatea utilizatorului este creata prin intermediului toolkit-ului Qt. De asemenea, a fost folosita aplicatia Qt Designer pentru deisignul mai usor (prin drag-and-drop) a ferestrelor. Aceasta aplicatie genereza fisiere .ui, care sunt incarcate in program prin intermediul unui loader specializat, pus la dispozitie de biblioteca PySide2.

Clasa principala a programului, GUI, este derivata din QStackedWidget, care permite crearea unui stive de widgeturi, denumite si pagini, dintre care, la un moment dat de timp, este vizibila una singura. Celelalte pagini sunt instante ale claselor QWidget si QDialog. In acest mod, a fost posibila includerea in proiect a 3 pagini si 2 dialoguri cu utilizatorul. In plus, a fost definita o implementare a clasei abstracte QabstractTableModel care impreuna cu un QTableView permite manipularea inregistrarilor utilizatorului curent.

Implementarea functionalitatii butoanelor/line edit-urilor se realizeaza cu ajutorul API-ului pus la dispozitie de Qt. Astfel, la fiecare actiune a utilizatorului, buton/edit-ul respectiv emite un semnal, precum clicked sau finishedEditing. Acestora li se poate asigna o functie de callback, numita si Slot. Acest model se dovedeste a fi unul de succes, deoarece se poate adapta necesitatilor aplicatiilor.

3. Structura si inter-relationarea tabelelor



- Tabelul ACCOUNTS contine datele necesare pentru logarea utilizatorului: loginul ales si parola criptata.
- Tabelul USERS contine informatii suplimentare despre utilizator necesare pentru calculul valorii recomandate de calorii, precum si data inregistrarii pentru a limita introducerea inregistrarilor anterioare acestei dati.
- Tabelul FOODS contine numele produsului, valoarea energetica exprimata in kCal si greutatea medie a unei portii/cantitati. Aceasta din urma este folosita pentru a facilita introduce inregistrarilor din interfata grafica
- Tabelul HISTORY stocheaza inregistrarile tuturor userilor. O inregistrare este formata din id-ul userului, id-ul alimentului consumat, ora mesei, cantitatea consumata (in grame) si numarul de calorii (calculat automat in interfata, in baza id-ului produsului si informatiile sale din tabelul FOODS)
- Intre tabelele ACCOUNTS – USERS exita o legutura ONE-TO-ONE
- Intre tabelele USERS – HISTORY exista o legutra MANY-TO-MANY
- Intre tabelele FOODS – HISTORY exista o legatura MANY-TO-MANY
- Se stabileste o legatura indirecta USERS – FOODS de tip MANY-TO-MANY. Un utilizator poate consuma diverse alimente si un aliment poate fi consumat de mai multi utilizatori distincti.

4. Constrangeri utilizate

Pentru toate campurile din toate tabelele s-a impus constrangerea NOT NULL.

- Tabelul ACCOUNTS:
  - PRIMARY KEY– campul ID
  - UNIQUE – campul login (poate exista un singur utilizator cu un anumit login)
- Tabelul USERS:
  - PRIMARY KEY – campul ID
  - FOREIGN KEY – campul ID (informatiile suplimentare trebuie sa apartina unui utiliator existent)
  - CHECK – campurile age(valori intre 10-110), sex(valori F/M), weight(valori intre 40-200), height(valori 120-220).

- Tabelul FOODS:
  - PRIMARY KEY – campul ID
  - UNIQUE – campul name (un produs poate aparea o singura data in tabela, cu un anumit numar de calorii atasat)

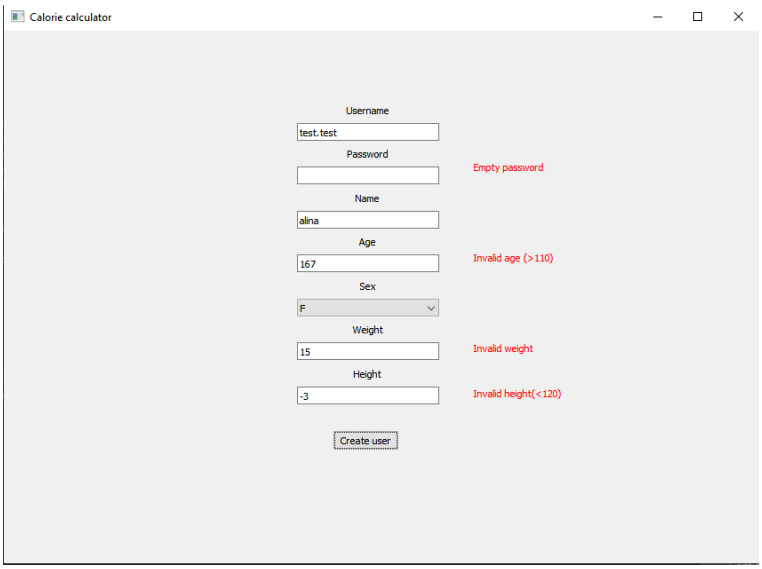
- Tabelul HISTORY:
  - PRIMARY KEY – tupla (user_id, food_id, meal_time) (in loc de a crea o noua inregistrare pentru a modifica cantitatea consumata de un anumit produs, utilizatorul poate edita acea inregistrare)
  - FOREIGN KEY – campurile user_id, food_id (atat utilizatorul, cat si produsul trebuie sa existe pentru a putea fi folosite intr-o inregistrare)

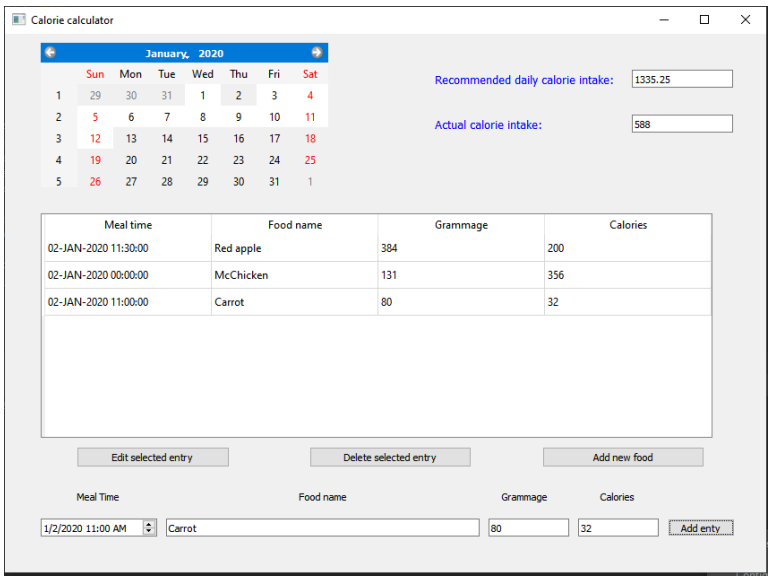
Pentru generarea id-urilor s-au folosit 2 secvente, user_id si food_id.

5. Modalitatea de conectare

Conexiunea la baza de date se realizeaza prin intermediul bibilotecii cx_Oracle. De asemenea, a fost construita o clasa singleton DBManager care faciliteaza executia si obtinerea rezultatelor generate de instructiunile SQL.

6. Screenshot-uri si exemple de cod






```sql
CREATE TABLE accounts(
    id NUMBER(10) PRIMARY KEY,
    login VARCHAR(30) NOT NULL UNIQUE,
    passwordHash VARCHAR(40) NOT NULL
    );

CREATE TABLE users(
    id NUMBER REFERENCES accounts (id) PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age NUMBER(3) NOT NULL CHECK(age BETWEEN 10 AND 110),
    sex CHAR(1) NOT NULL CHECK (sex = 'F' or sex = 'M'),
    weight NUMBER(3) NOT NULL CHECK(weight BETWEEN 40 AND 200),
    height NUMBER(3) NOT NULL CHECK(height BETWEEN 120 AND 220),
    registration_date DATE DEFAULT SYSDATE
)

CREATE TABLE foods(
    id NUMBER PRIMARY KEY,
    name VARCHAR (50) NOT NULL UNIQUE,
    calories NUMBER(4) NOT NULL,
    portion_size NUMBER(4) NOT NULL
);

CREATE TABLE history(
    user_id NUMBER REFERENCES users (id),
    food_id NUMBER REFERENCES foods (id),
    meal_time DATE NOT NUll,
    grammage NUMBER (4) NOT NULL,
    calories NUMBER(4) NOT NUlL,

    CONSTRAINT history_PK PRIMARY KEY (user_id, food_id, meal_time)
);


create sequence user_id
increment by 1;
```

GUI.py | DBManager.py | config.py | insertData.sql

```
116         # creare model pt tableView
117         self.model = TableModel(self, [])
118         self.pageUserHistory.tvHistory.setSelectionBehavior(QTableView.SelectRows)
119         self.pageUserHistory.tvHistory.setModel(self.model)
120         self.pageUserHistory.tvHistory.resizeColumnsToContents()
121         self.pageUserHistory.tvHistory.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
122
123         # setare qcompleter
124         db = DBManager.DBManager.get_instance()
125         foods_list = db.get_foods()
126         completer = QCompleter(foods_list, parent=None)
127         completer.setCaseSensitivity(Qt.CaseInsensitive)
128         self.pageUserHistory.leFoodName.setCompleter(completer)
129         self.dialogEditEntry.leFoodName.setCompleter(completer)
130
131         # conectare signal->slot
132         self.pageAuthentication.buttonNewUser.clicked.connect(self.on_buttonNewUser_clicked)
133         self.pageAuthentication.buttonLogin.clicked.connect(self.on_buttonLogin_clicked)
134
135         self.pageNewUser.buttonCreateUser.clicked.connect(self.on_buttonCreateUser_clicked)
136
137         self.pageUserHistory.leFoodName.editingFinished.connect(self.check_food_name)
138         self.pageUserHistory.leGrammage.textChanged.connect(self.check_grammage)
139         self.pageUserHistory.buttonAddEntry.clicked.connect(self.on_buttonAddEntry_clicked)
140         self.pageUserHistory.buttonDelete.clicked.connect(self.on_buttonDeleteEntry_clicked)
141         self.pageUserHistory.buttonEdit.clicked.connect(self.on_buttonEditEntry_clicked)
```

GUI.py | DBManager.py | config.py | insertData.sql

```
213             check = 0
214             self.pageNewUser.labelErrUsername.setText('Empty username')
215         else:
216             result = db.check_username(username)
217             if result:
218                 check = 0
219                 self.pageNewUser.labelErrUsername.setText('Username already exists')
220             else:
221                 if len(username) > 30:
222                     self.pageNewUser.labelErrName.setText('Username too long')
223                 else:
224                     self.pageNewUser.labelErrUsername.setText('')
225
226         password = self.pageNewUser.lePass.text()
227         if not password:
228             check = 0
229             self.pageNewUser.labelErrPass.setText('Empty password')
230         else:
231             if len(password) < 5:
232                 check = 0
233                 self.pageNewUser.labelErrPass.setText('Password too short')
234             else:
235                 if len(password) > 40:
236                     check = 0
237                     self.pageNewUser.labelErrPass.setText('Password too long')
238                 else:
```

GUI › on_buttonCreateUser_clicked() › else › else