UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” IAȘI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

GESTIONAREA CENTRELOR DE RECOLTARE DE SÂNGE

(pentru analize)

Proiect la disciplina

BAZE DE DATE

Coordonator,
Cristian-Nicolae Buțincu

Student,
Mihalache Nicoleta-Ecaterina
Grupa: 1309A

Iași, 2020

Titlu proiect: Gestionarea centrelor de recoltare de sânge (pentru analize)

Analiza, proiectarea și implementarea unei baze de date, precum și implementarea vizuală a funcționalităților principalelor funcții de interogare a unei baze de date. Astfel putem accesa/ modifica o bază de date cu ușurință datorită bibliotecilor din python folosite și menționate mai sus.
Principalele funcții folosite sunt:

- Select
- Insert
- Update
- Delete

Descrierea cerințelor și modul de organizare al proiectului

Volumul mare de informații existente în cazul unui centru de recoltare de sânge cu numeroși pacienți determină necesitatea fluidizării fluxurilor de date, gestiunea acestora fiind o adevărată provocare. Activitatea de gestiune a unui centru de recoltare de sânge implică o muncă intensă în ceea ce privește numeroasele documente ce țin de înregistrarea pacienților, a analizelor pe care aceștia le fac, de personalul medical, precum și de realizarea analizelor în urma cărora cabinetul medical oferă pacienților rezultatul acestora.
Am ales ca în aplicația realizată să nu implic toate tabelele din baza de date în interfață. Voi urmări ca prin această aplicație să se pună accentul pe un singur cabinet unde vor fi anagajați mai multe asistente și mai mulți medici, unde vor veni să-și facă analize mai mulți pacienți. Voi urmări afișarea pacienților cu toate detaliile, adăugarea, modificarea și ștergerea lor; afișarea analizelor cu detaliile aferente acestora, agăugarea, modificarea și ștergerea lor; afișarea asistentelor și a medicilor precum și adăugarea și ștergerea lor.

Informațiile de care avem nevoie sunt cele legate de:

-**cabinete**: ne interesează să identificăm cabinetul ce va gestiona datele (de exemplu, posibilitatea pacienților de a alege un anume cabinet unde își vor face analizele medicale). Acesta va avea o denumire, o adresă și un număr de telefon.

-**contracte**: ne interesează să identificăm contractele semnate de fiecare cabinet cu angajații săi, ce vor avea o data când au fost semnate, o dată de final și un număr de luni ce reprezintă durata acestora.

-**pacienți**: datele referitoare la pacienți se află în tabela Pacienti și ne interesează numele și prenumele pacientului, data nașterii, CNP-ul, vârsta, genul și adresa acestuia.

-**analize**: în cazul informațiilor legate de analize, informațiile se vor stoca în două tabele: Analize și Detalii. Astfel, în tabela Analize vom stoca informații legate de tipul analizei, data când a fost recoltată și data rezultatului, urmând ca în tabela Detalii să fie stocate informații legate de numele analizei ce face parte dintr-o anumită categorie data de tipul analizei din tabela Analize, valoarea minimă și valoarea maximă a acesteia.

-**angajatii**: în cazul informațiilor legate de angajați, informațiile se vor stoca în două tabele: Asistente și Medici_lab. Astfel, în tabela Asistente vom stoca informații legate de asistentele angajate care se vor ocupa de recoltarea sângelui de la pacienți pentru analize, numele și prenumele lor, iar în tabela Medici_lab vom stoca informațiile legate de medicii de laborator, cei care realizează rezultatul analizei, numele și prenumele acestora.

Descrierea funcțională a aplicației

Principalele funcții care se pot întâlni într-un centru de recoltare de sânge sunt:

- Evidența pacienților
- Evidența angajaților
- Evidența analizelor efectuate

Descrierea detaliată a entităților și a relațiilor dintre tabele Tabelele

Tabelele din această aplicație sunt:

- ✓ Cabinete;
- ✓ Contracte;
- ✓ Analize;
- ✓ Detalii;
- ✓ Pacienți;
- ✓ Asistente;
- ✓ Medici_lab;
- ✓ Pacienti_Analize;
- ✓ Pacienti_Detalii;

În proiectarea acestei baze de date s-au identificat tipurile de **relații** 1:1, 1:n, n:n.

Între tabelele **Cabinete** și **Contracte** este o relație **one-to-many** deoarece un cabinet poate încheia mai multe contracte. Reciproca însă nu este valabilă, deoarece un contract poate fi semnat doar de un singur cabinet. Legătura dintre cele două este realizată prin câmpul **Cabinete_id_cabinet**.

Între tabelele **Cabinete** și **Analize** este o relație **one-to-many** deoarece un cabinet poate realiza un tip de analize la un anumit pret, prețul fiind un atribut de pe această relație.Reciproca însă nu este valabilă, deoarece o analiză poate fi realizată doar de un singur cabinet la un anumit preț. Prețul analizelor poate să difere de la un cabinet la altul. Legătura dintre cele două este realizată prin câmpul **Cabinete_id_cabinet.**

Între tabelele **Analize** și **Detalii** este o relație **one-to-one** deoarece unei analize îi corespunde un set de detalii, iar un set de detalii este deținut de o singură analiză. Tabela Detalii este concepută în special pentru a descrie intervalul de valori pe care le poate avea o analiză. Legătura dintre cele două este data de câmpul **Analize_id_analiza**.

Între tabelele **Analize** și **Pacienti** este o relație **many-to-many** deoarece o analiză pot fi făcută de unul sau mai mulți pacienți, iar un pacient poate face una sau mai multe analize. Pentru ca tabela să se afle în 3FN această relație se va sparge în două, rezultând două relații **1:n** și legătura dintre cele două se va realiza cu ajutorul unei alte tabele **Pacienti_Analize** care va conține cheia primară a fiecărei tabele. Altfel spus legătura se face prin două câmpuri **Pacienti_id_pacient** și **Analize_id_analiza** reunite într-o tabelă comună.

Între tabelele **Pacienți** și **Detalii** este o relație **many-to-many** deoarece un pacient pot avea una sau mai multe detalii ale analizelor, iar un detaliu este corespondent unui sau mai multor pacienți. Pentru ca tabela să se afle în 3FN această relație se va sparge în două, rezultând două relații **1:n** și legătura dintre cele două se va realiza cu ajutorul unei alte tabele **Pacienti_Detalii** care va conține cheia primară a fiecărei tabele. Altfel spus legătura se face prin două câmpuri **Detalii_id_detalii** și **Pacienti_id_pacient** reunite într-o tabelă comună. Această tabelă va conține și atributul **rezultat**, fiecare pacient având un rezultat unic la analize.

Între tabelele **Analize** și **Medici_lab** este o relație **one-to-many** deoarece o analiză poate fi realizată de unul sau mai mulți medici. Reciproca însă nu este valabilă, deoarece un medic poate realiza o singură analiză. Legătura dintre cele două este data de câmpul **Medici_id_medic**.

Între tabelele **Pacienți** și **Asistente** este o relație **one-to-many** deoarece unui pacient i se poate recolta sânge de către una sau mai multe asistente. Reciproca însă nu este valabilă, deoarece o asistentă poate recolta sânge de la un singur pacient. Legătura dintre cele două este data de câmpul **Asistente_id_asistentă**.

Între tabelele **Contracte** și **Medici_lab** este o relație **one-to-one** deoarece un contract este semnat cu un medic, iar un medic poate semna un singur contract. Legătura dintre cele două este data de câmpul **Contracte_id_contract**.

Între tabelele **Contracte** și **Asistente** este o relație **one-to-one** deoarece un contract poate fi semnat cu o asistentă, iar o asistentă poate semna un singur contract. Legătura dintre cele două este data de câmpul **Contracte_id_contract**.

În proiectarea acestei baze de date s-au identificat următoarele tipuri de **constrângeri de integritate referențială**: primary key și foreign key, precum și **alte tipuri** de constrângeri: check, unique, not null.

Primary key-urile sunt generate de baza de date prin mecanismul de tip **autoincrement** și sunt atribuite tuturor id-urilor din tabele, identificând în mod unic fiecare înregistrare dintr-un tabel. Ele trebuie să conțină doar valori unice și nu pot avea valori NULL, iar un tabel poate avea doar o singură cheie primară. Aici avem totodată și constrângerile de tip **unique** și **not null**.

Constrângerea **unique** este atașată atributului CNP din tabela Pacienti, deoarece nu pot exista doi pacienți cu același cnp. Punând această constrângere impunem ca toate CNP-urile introduse în baza de date să fie unice.

Foreign key-urile sunt utilizate pentru a lega două tabele împreună, fiind utilizate în toate relațiile dintre tabelele bazei de date. Acestea sunt câmpuri din tabele care se referă la Primary key-ul dintr-un anumit tabel. Un exemplu este Cabinete_id_cabinet ce este foreign key și leagă tabela copil Contracte de tabela părinte Cabinete. Pot exista și mai multe foreign key-uri și un exemplu este în cazul în care avem o relație many-to-many. Aceasta este spartă în alte două relații one-to-many rezultând astfel 2 foregn key-uri. Un exemplu este apariția tabelei copil Pacienti_Analize între tabelele Pacienti și Analize, având 2 foreign key-uri Pacienti_id_pacient și Analize_id_analiză.

Constrângerea de tip **check** este utilizată pentru atributul nr_telefon din tabela Cabinete deoarece se impune un anumit format pentru numarul de telefon pe care îl poate avea acel cabinet (să conțină 10 cifre, prima cifră să fie 0 și a doua cifră să fie 7, 2 sau 3). Mai este utilizată pentru atributele val_min și val_max din tabela Detalii pentru ca val_min să nu conțină valori mai mici decât 0 și pentru

val_max să conțină valori mai mari decât valorile pe care le conține val_min, pentru atributul CNP din tabela Pacienți ca lungimea lui să fie egală cu 13.

Constrângerea **check** este utilizată pentru a impune unei valori să aparțină unei liste de valori existente. Un exemplu este dat de către atributul gen din tabela Pacienți a cărei valoare trebuie să existe în lista de valori data, dacă este femeie gen=F și dacă este bărbat gen=B. Ea este utilizată și pentru atributele tip și nume_d din tabelele Analize și Detalii pentru a specifica categoria și numele analizei.

Instalare aplicații necesare

Această aplicație este alcătuită din două părți: partea de front-end și cea de back-end.

1.Partea de front-end este realizată în HTML și CSS.

2.Partea de back-end este realizată in Python.
Se instalează python versiunea 3.8 care pote fi descarcată de pe site-ul official https://www.python.org/downloads/ .

Aplicația are nevoie de următoarele biblioteci:

1. Click versiunea 6.7

Click este un pachet Phyton folosit pentru a crea interfețe în linia de comandă, într-un mod ușor de înțeles, eficient și ușor de utilizat, cu cât mai puțin cod posibil.

Există mai multe astfel de pachete Python. Însă de ce folosim Click?
Sunt câteva motive pentru care se folosește Click-ul:

- este ușor de compilat și nu are restricții;
- este complet îmbricat;
- generează automat o pagina help;

Comanda de instalare:

```
pip install click == 6.7
```

Un exemplu simplu de folosire a bibliotecii click:

```
import click


@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name',
              help='The person to greet.')
```

```
def hello(count, name):
    """Simple program that greets NAME for a total of COUNT times."""
    for x in range(count):
        click.echo('Hello %s!' % name)

if __name__ == '__main__':
    hello()
```

2. Flask versiunea 0.12.2

Flask este un API (Application Programming Interface) Python ce permite construirea de aplicații web. A fost dezvoltat de Armin Ronacher (un programator austriac și un speaker cunoscut la conferințele în domeniul software).
Acest framework este mult mai ușor de înțeles decât framework-ul Django's și mult mai ușor de învățat deoarece are mai puțin cod de implementat în cazul unei aplicații web.

Comanda de instalare Flask:

```
pip install Flask
```

Un exemplu de folosire a Flask-ului:

```
from flask import Flask
app = Flask(__name__) # Flask constructor
# A decorator used to tells the application
# which URL is associated function
@app.route('/')
def hello():
  return 'HELLO'
if __name__=='__main__':
app.run()
```

2.1. Django

Django este un framework web bazat pe Python, care vă permite să creați rapid aplicații web fără toate problemele de instalare sau de dependență pe care le veți găsi în mod normal. Când construiți un site web, aveți întotdeauna nevoie de un set similar de componente: o modalitate de a gestiona autentificarea utilizatorilor (înscrierea, conectarea, deconectarea), un panou de gestionare pentru site-ul dvs. web, formulare, o modalitate de a încărca fișiere etc. Django vă oferă componente gata de

utilizat.

De ce Django?

1. Este foarte ușor să schimbați baza de date în cadrul Django.
2. Acesta a construit în interfața de administrare care face ușor de a lucra cu ea.
3. Django este un cadru complet funcțional care nu necesită altceva.
4. Are mii de pachete suplimentare disponibile.
5. Este foarte scalabil.

2.2.Werkzeug

Werkzeug este o bibliotecă utilitară pentru limbajul de programare Python , cu alte cuvinte un set de instrumente pentru aplicații WSGI (Web Server Gateway Interface) și este licențiat sub licență BSD . Werkzeug poate realiza obiecte software pentru funcții de solicitare, răspuns și funcții utilitare.

2.3. Jinja2

Jinja2 este o librărie pentru Python, creata pentru a fi flexibila, rapidă și sigură.
Poți instala ultimele versiuni de Jinja2 folosind easy_install sau pip.

```
easy_install Jinja2
pip install Jinja2
```

Caracteristici:

-Conține server de dezvoltare și depanator;
-Suport integrat pentru testarea unităților;
-Utilizează Jinja2 templating;
-Suport pentru cookie-urile securizate;
-Compatibilitatea Google App Engine.

```
from flask import Flask
app = Flask ( __name__ )


  @app.route ( "/" )
  def hello ():
    return "Hello World!"


  if __name__ == "__main__" :
     app .  run ()
```

3. cx_Oracle

cx_Oracle este un modul de extensie Python care permite accesul la Oracle Database.
Conformă cu specificația API 2.0 a bazei de date Python, cu un număr considerabil de adăugiri și câteva excluderi.

Pentru a utiliza cx_Oracle 7 cu Python și Oracle Database aveți nevoie de:

-Python 2.7 sau 3.5 și mai mult. Versiunile mai vechi ale cx_Oracle pot funcționa cu versiuni mai vechi ale Python.

-Bibliotecile client Oracle. Acestea pot fi de la clientul Oracle Instant Client gratuit sau de la cele incluse în Oracle Database dacă Python se află pe aceeași mașină ca și baza de date. Bibliotecile clienților Oracle 19, 18, 12 și 11.2 sunt acceptate pe Linux, Windows și MacOS. Utilizatorii au raportat, de asemenea, succesul cu alte platforme.

-O bază de date Oracle. Standardul de interoperabilitate Oracle standard pentru client-server permite cx_Oracle să se conecteze la bazele de date mai vechi și mai noi.

Instalare

```
python - m pip install cx_Oracle – upgrade
```

Data modeler

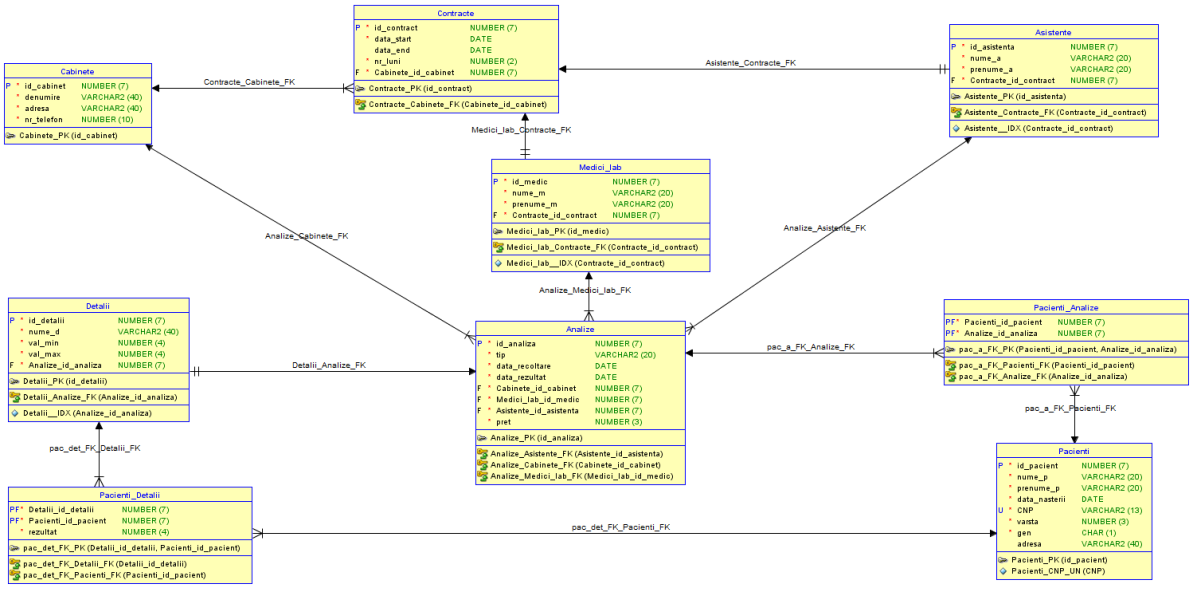

Fig.1. Diagramă relațională

Funcționalitatea aplicației

Prin această aplicație am încercat să implementez vizual funcționalitatea principalelor funcții de interogare a unei baze de date. Astfel putem accesa/ modifica o bază de date cu ușurință datorită bibliotecilor din python menționate mai sus.
Principalele funcții folosite sunt:

1.Funcția Select

Fiecare din următoarele declarații sunt valide:

```
SELECT * FROM Pacienti;

SELECT
*
FROM
Pacienti
;

SELECT *
FROM  Pacienti;
```

Este comanda cea mai utilizată. Este folosită pentru obținerea datelor din bazele de date.

Exemplu de utilizare a funcției select pentru afișarea datelor dintr-o tabelă în python.

```python
@app.route('/Pacienti')
def Pacienti():
    pacienti=[]

    cursor.execute("SELECT * FROM Pacienti")
    for result in cursor:
      pacient={}
      pacient['id_pacient']=result[0]
      pacient['nume_p']=result[1]
      pacient['prenume_p']=result[2]
      pacient['data_nasterii']=datetime.strptime(str(result[3]), '%Y-%m-%d %H:%M:%S').strftime('%d.%b.%y')
      pacient['CNP']=result[4]
      pacient['varsta']=result[5]
      pacient['gen']=result[6]
      pacient['adresa']=result[7]
      pacienti.append(pacient)
    return render_template('pages/Pacienti.html', Pacienti=pacienti)
```

Fig.2.Funcția SELECT în Python

Rezultatul acestui cod este următorul:

index	nume_p	prenume_p	data_nasterii	CNP	varsta	gen	adresa	Stergere pacient
1	MIHALACHE	NICOLETA	06.Dec.98	1234567891234	23	F	BOTOSANI	Sterge
2	ROSU	MIHAI	23.Jun.72	1234536251234	43	M	SUCEAVA	Sterge
3	POPESCU	LAVINIA	14.Sep.01	1234590791234	20	F	BOTOSANI	Sterge
4	SCRIPCARU	SEBASTIAN	27.May.99	1234567856723	19	M	BUCURESTI	Sterge
5	IONESCU	GIGI	04.Mar.98	2345676543478	25	M	IASI	Sterge
6	PUSCASU	ANDREEA	23.Jul.99	1234227891234	22	F	SIBIU	Sterge
7	DANILIUC	STEFAN	21.Mar.97	1234536251200	23	M	BRASOV	Sterge
8	APETREI	BIANCA	14.Oct.04	1234590735234	28	F	BACAU	Sterge
9	POPA	OANA	18.Jul.17	4170718070033	4	F	BOTOSANI	Sterge
10	CHELARESCU	COSMIN	12.Dec.98	1981212070055	23	M	IASI	Sterge
11	IRIMIA	COSMINA	23.May.98	1984590753424	23	F	BRASOV	Sterge

Fig.3.Afișarea in front-end a rezultatului funcției SELECT

2.Funcția Insert

Adăugare o nouă înregistrare.

Sintaxa este :

```
INSERT INTO tabela [ ( coloana [ , coloana . . . ])]
VALUES ( valoare [, valoare . . . ]);
```

Exemplu de cod pentru introducerea unor date noi în tabela:

```python
@app.route('/addPacient', methods=['POST'])
def addPacient():
    try:
        if request.method=='POST':
            pacient=0
            cursor.execute('SELECT max(id_pacient) FROM Pacienti')

            for result in cursor:
                pacient=result[0]
            pacient+=1

            values=[]
            values.append("'"+str(pacient)+"'")

            if request.form['nume_p'].isdigit():
                print("Numele pacientului contine numere!")
            else:
                print("Numele pacientului nu contine numere!")
                values.append("'" + request.form['nume_p'] + "'")

            if request.form['prenume_p'].isdigit():
                print("Prenumele pacientului contine numere!")
            else:
                print("Prenumele pacientului nu contine numere!")
                values.append("'" + request.form['prenume_p'] + "'")

            values.append("'" + datetime.strptime(str(request.form['data_nasterii']),'%Y-%m-%d').strftime('%d.%b.%y') + "'")
            values.append("'" + request.form['CNP'] + "'")
            values.append("'" + request.form['varsta'] + "'")

            if request.form['gen']=="F":
                values.append("'" + request.form['gen'] + "'")
            elif request.form['gen']=="M":
                values.append("'" + request.form['gen'] + "'")
            else:
                print("Gen invalid!")

            if request.form['adresa'].isdigit():
                print("Adresa pacientului contine numere!")
            else:
```

Fig.4.Funcția INSERT în Python

```python
            else:
                print("Adresa pacientului nu contine numere!")
                values.append("'" + request.form['adresa'] + "'")

            fields=['id_pacient','nume_p','prenume_p','data_nasterii','CNP','varsta','gen','adresa']
            query = 'INSERT INTO %s (%s) VALUES (%s)' % ('Pacienti', ', '.join(fields), ', '.join(values))
            cursor.execute(query)
            cursor.execute('commit')
        print('S-a realizat adaugarea!')
        return redirect('/Pacienti')
    except:
        print('Nu s-a realizat adaugarea! Probleme la baza de date!')
        return redirect('/Pacienti')
```

Fig.5.Funcția INSERT în Python

Pagina pentru insert este următoarea:

Fig.6. Pagina de inserare/ adăugare

3.Funcția Update

Modifică datele existente cu alte date valide.
Exemplu de cod:

```python
@app.route('/editPacient', methods=['POST'])
def editPacient():
    try:
        pacient=0

        nume_p="'"+request.form['nume_p']+"'"
        if nume_p.isdigit():
            print('Numele pacientului contine numere!')
        else:
            print('Numele pacientului nu contine numere!')

        prenume_p= "'" + request.form['prenume_p'] + "'"
        if prenume_p.isdigit():
            print('Prenumele pacientului contine numere!')
        else:
            print('Prenumele pacientului nu contine numere!')

        cursor.execute('SELECT id_pacient FROM Pacienti where nume_p='+nume_p)
        for result in cursor:
            pacient=result[0]

        data_nasterii = "'" + datetime.strptime(str(request.form['data_nasterii']),'%Y-%m-%d').strftime('%d.%b.%y')+"'"
        CNP = "'" + request.form['CNP'] + "'"
        varsta = "'" + request.form['varsta'] + "'"
        gen = "'" + request.form['gen'] + "'"
        adresa = "'" + request.form['adresa'] + "'"
        if adresa.isdigit():
            print("Adresa pacientului contine numere!")
        else:
            print("Adresa pacientului nu contine numere!")

        query='UPDATE Pacienti SET nume_p=%s , prenume_p=%s , data_nasterii=%s , CNP=%s , varsta=%s ,gen=%s , adresa=%s WHERE id_pacient=%s' % (nume_p, prenume_p, data_nasterii, CNP, varsta, gen, adresa
        cursor.execute(query)
        cursor.execute('commit')
        print('S-a realizat modificarea!')
        return redirect('/Pacienti')
    except:
        print('Nu s-a realizat modificarea! Probleme la baza de date!')
        return redirect('/Pacienti')
```

Fig.7.Funcția UPDATE în Python

La efectuarea unui update, datele existente sunt citite din tabelă și completate automat în câmpurile necesare. Pagina de update arată astfel:

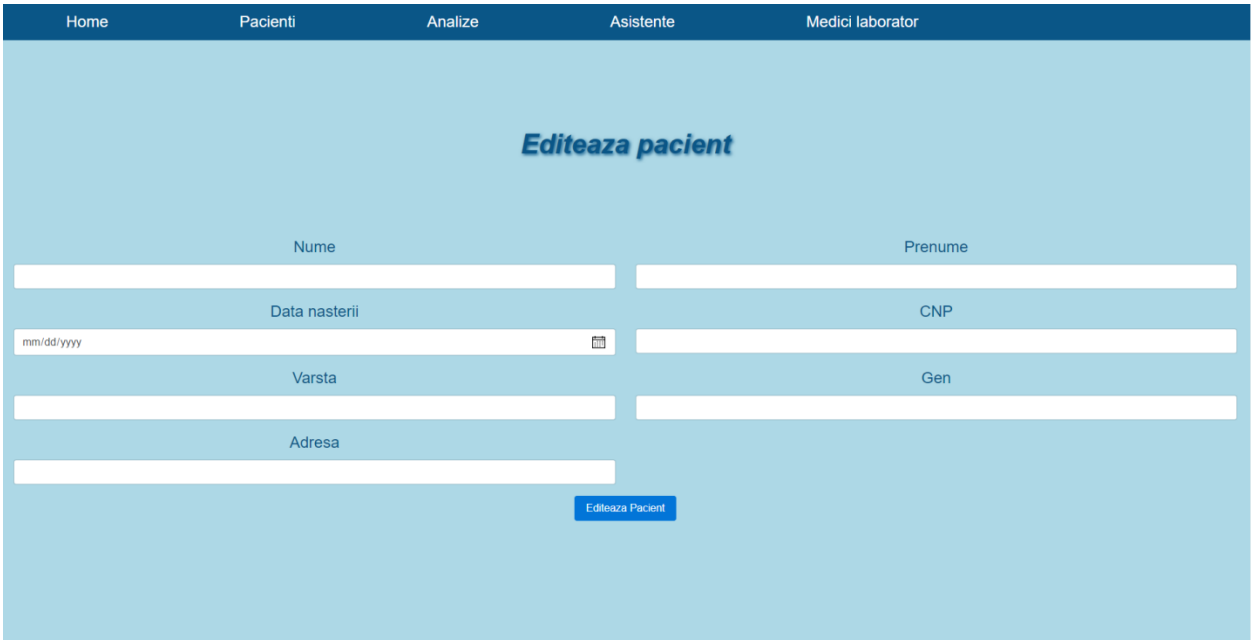

Fig.8.Pagina de editare/ modificare

4.Functia Delete

Șterge o înregistrare din tabelă.
Exemplu de cod utilizat în aplicația noastră:

```
@app.route('/delPacient', methods=['POST'])
def delete_pacient():
    try:
        pacient = request.form['id_pacient']
        cur = con.cursor()
        cur.execute('delete from Pacienti where id_pacient=' + pacient)
        cur.execute('commit')
        print('S-a realizat stergerea!')
        return redirect('/Pacienti')
    except:
        print('Nu s-a realizat stergerea! Probleme la baza de date!')
        return redirect('/Pacienti')
```

Fig.9.Funcția DELETE în Python