UNIVERSITATEA TEHNICĂ “GH ASACHI” IAŞI

FACULTATEA AUTOMATICĂ ŞI CALCULATOARE

SPECIALIZAREA

CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

DISCIPLINA: BAZE DE DATE

Rezervare bilete avion

Profesor coordonator,

Sorin Avram

Student,

Grosu Gabriela

Grupa 1311A

Iași,2022

Titlul proiectului

Rezervare bilete avion

Analiza, proiectarea și implementarea unei baze de date care să modeleze un sistem de rezervare a biletelor în cadrul unui aeroport.

Descrierea proiectului

Prin această baza de date s-a dorit modelarea într-un mod minimal a unei baza de date din cadrul unui aeroport. În cadrul acesteia se va ține evidența aeroporturilor, zborurilor, pasagerilor și a biletelor rezervate.

Pentru a implementa această bază de date au fost necesare următoarele informații cu privire la: aeroporturile existente, zborurile disponibile, informații despre clienți și pașaportul acestora, dar și despre rezervările făcute de aceștia.

Tehnologii folosite pentru partea de Front-end si Back-end

Pentru partea de Back-end

Am utilizat Oracle SQL Developer, Oracle SQL Data Modeler pentru crearea tabelelor, realizarea modelului logic și modelului relațional.

Pentru partea de Front-end

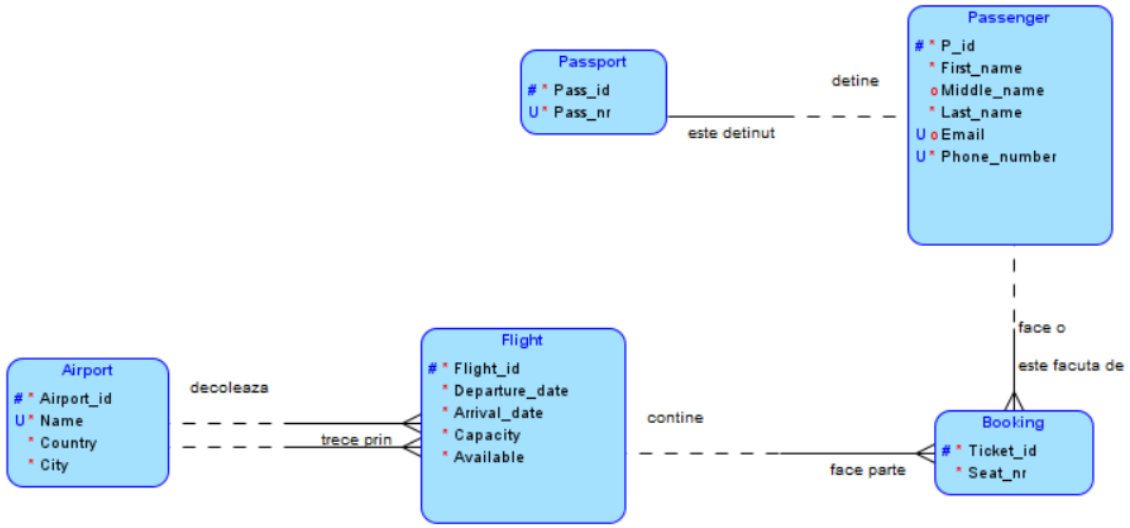
Am utilizat libraria cx_Oracle pentru a realiza conexiunea cu baza de date și următoarele limbaje de programare : Python, HTML, CSS și JavaScript pentru a valida introducerea unor valori în comparație cu valorile deja aflate în tabelă (ex. unicitate, departure_airport!=arrival_airport)

Structura tabelelor

Tabelele din această aplicație sunt:

- Airport
- Flight
- Passport
- Passenger
- Booking

Modelul logic



Pentru a putea face o rezervare vom avea nevoie mai întâi de tabelele Airport care conține aeroporturile, Flight pentru zborurile disponibile. Fiecare aeroport va fi identificat prin : **id**, **nume**, **țara** si **orașul** în care este situat. Zborurile au următoarele atribute : **id**, **data**, **ora de plecare si sosire, capacitate, locuri disponibile** și prin cu ajutorul unor foreign key-uri vor fi două id-uri de aeroporturi, unul pentru plecare și unul pentru sosire pentru că un zbor trece minim prin două aeroporturi.

A doua parte se axează pe clienți, cei care vor face rezervarea, și vom avea nevoie de tabelele **Passport** și **Passenger**. Pașaportul de identifica prin id si număr ,care va fi unic și id-ul pasagerului care va fi un foreign key. Tabela Passenger va conține toate informațiile legate de client. În această tabelă avem informații cu privire la nume, prenume, email și număr de telefon.

Iar ultima parte va fi cea de rezervare propriu-zisă. După ce un client s-a inserat cu succes in baza de date în tabela Passenger, acesta va putea rezerva unul sau mai multe bilete pentru diferite zboruri, iar de acest lucru se va ocupa tabela Booking. Fiecare rezervare se va identifica prin: id, numărul de locuri rezervate, id_client pentru a ști cine a făcut rezervarea, id_zbor pentru a avea informații și despre zborul ales (ultimele trei atribute fiind foreign key-uri).

Normalizare

La această bază de date s-a aplicat normalizarea pentru a reduce redundanța datelor, pentru a nu apărea anomalii la operațiile de update, insert sau delete.

Caracteristicile pentru fiecare tip de normalizare sunt:

-FN1

-un atribut conține valori atomice din domeniul lui

-nu conține grupuri care se repetă

-FN2

-este FN1

-dacă toate atributele non-cheie depind în totalitate de toate cheile candidat

-FN3

-este FN2

-toate atributele non-cheie sunt direct dependente de cheile candidat

-FN Boyce Codd

-este FN3

-pentru fiecare dependenta functionala x->y cel putin una din afirmatii e adevarata: x->y dependenta functionala banala si x super cheie

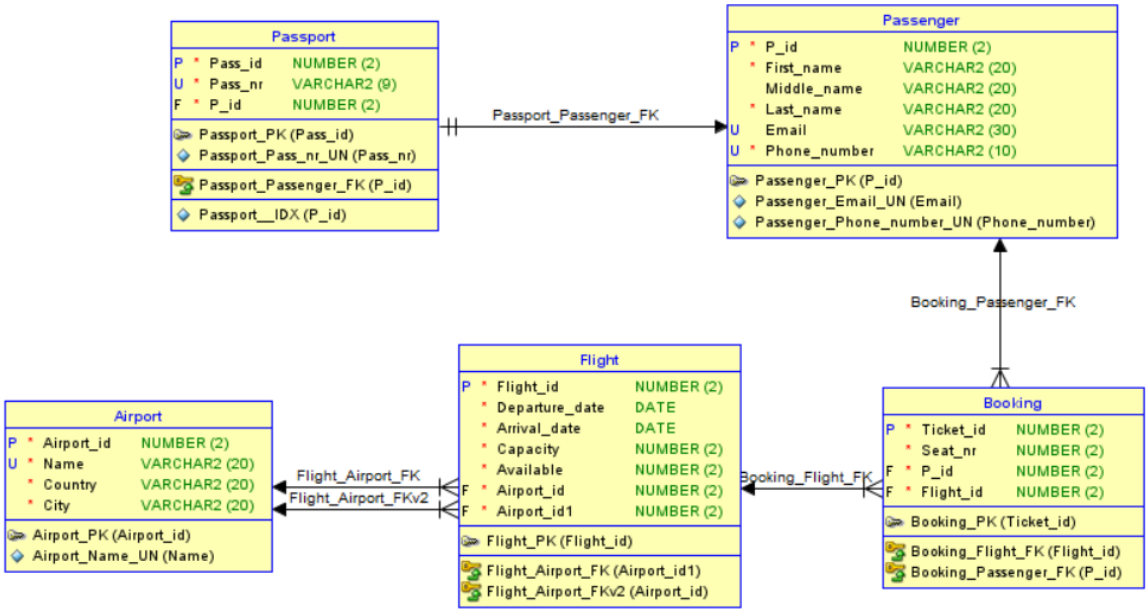
Tabela **Flight** nu este FN2, deoarece există atribut care nu depinde în totalitate de cheia candidat(flight_id) și acela este available care depinde de capacity, dar este FN1.

Tabela **Airport** este FN1. Verificam FN2 care se confirma,deoarece tabela este deja in forma FN1, iar atributele non-cheie depind in totalitate de cheile candidat. In cazul nostru atributele non-cheie sunt Country si City,iar cheile candidat sunt airport_id si name(impreuna sau separat) si se poate

observa ca aceste atribute non-cheie depinde in totalitate de chei. Daca FN2 este stabilit, continuam cu FN3, si dupa un rationament asemanator cu cel anterior rezulta ca tabela Booking este in FN3.

In urma unui rationament asemanator cu cel din tabela Airport, am dedus ca si tabelele **Passenger** este FN3, **Passport** si **Booking** sunt in FN Boyce Codd.

Modelul relațional



Relațiile dintre tabele

În proiectarea acestei baze de date s-au folosit relații de tipul:

1:1 (one-to-one) și 1:n (one-to-many)

Între tabela **Passport** și **Passenger** se crează o relație de 1:1 deoarece fiecare pasager are propriul pașaport, și nu putem avea doi pasageri cu același pașaport. Legătura dintre două tabele se face prin câmpul **p_id.**

Între tabela **Passenger** și **Booking** se creează o relație de 1:n => un pasager poate face mai multe rezervări, dar o rezervare aparține doar unui pasager. Legătura dintre aceste tabele se face prin câmpul **p_id**.

Între tabela **Aiport** și **Flight** avem două relații de 1:n => într-un aeroport decolează mai multe zboruri și, de asemenea, un zbor trece minim prin două aeroporturi, unul de plecare și unul de sosire. Această legătură se face prin intermediul unor două câmpuri de tip **airport_id**.

Între tabela **Flight** și **Booking** este o relație de 1:n => într-un zbor sunt mai multe rezervări, dar o rezervare poate face parte numai dintr-un zbor. Legătura se face prin câmpul **flight_id.**

Între tabela **Passenger** și **Flight** este o relație ipotetică de tip m:n =>un zbor conține rezervările de la mai mulți pasageri, iar un pasager poate rezerva bilet la mai multe zboruri. Aceasta relație se realizează prin intermediul tabelei **Booking**.

Descrierea constrângerilor

Constrângerile de tip *check* sunt numeroase și se găsesc în toate tabelele.

Prin aceste constrângeri s-a verificat:

→dimensiunea valorilor introduse, dar și formatul introdus

De exemplu, pentru atributele: **First_name, Middle_name, Last_name, Country, City, Name** se va verifica ca lungimea cuvântului să fie de cel puțin două caractere, și cu ajutorul unei funcții de tip regexp_like se va verifica dacă nu au fost introduse anumite caractere nepermise (în acest caz cifre), este verificat formatul **numărului de telefon** pentru a se introduce doar cifre, iar prima cifra să fie egala cu 0, iar a doua să fie 7/2/3 (numere de fix sau mobil). S-a verificat și formatul **email**-ului care poate conține litere sau cifre, @ sau .

→valoarea introdusă

- ➔ **Seat_nr** va fi între 1 și 10
- ➔ **Arrival_date** trebuie să fie mai mare decât **Departure_date**
- ➔ **capacity** mai mare sau egală cu 10
- ➔ **available** mai mare sau egală 0 și mai mică sau egală cu **capacity**

➔ în tabela Flight, **aiport_id!=airport_id1** pentru că nu este posibil ca un avion să plece și să sosească în același aeroport

Sunt folosite constrângeri de tip **unique** pentru atributele: **pass_nr** (pentru a nu avea două pașapoarte identice), **email**, **phone_number**(pentru a nu avea doi clienți cu același email/număr de telefon) și la **Name** din cadrul tabelei Airport, pentru a nu avea două aeroporturi identice.

Pe lângă constrângeri sunt folosite și **triggere** pentru a ține evidența locurilor disponibile pentru fiecare zbor. La inserare este folosit un trigger numit **Available_initial** care atunci când introducem un nou zbor, locurile disponibile sa fie egal cu capacitatea avionului. Apoi avem două triggere: **actualizare_rezervare** și **actualizare_anulare_rezervare** care se ocupă cu incrementarea sau decrementarea locurilor disponibile în funcție de inserarea sau ștergerea unei înregistrări. Atunci când se va insera o înregistrare se va scade din numărul de locuri disponibile locurile rezervate, iar la anulare/ștergere se va aduna în același mod.

Pentru a face actualizarea capacității avionului trebuie să avem grijă și de locurile disponibile care trebuie să se actualizeze și ele. Pentru acest lucru am folosit o variabila **old_capacitate** în care stocăm valoare de dinainte de actualizare a capacității. Această valoare ne va ajuta la actualizarea variabilei Available care se va calcula cu formula **Available=Available-(old_capacitateCapacity)**. Această formulă funcționează și pentru incrementarea, dar și pentru decrementarea capacității, singura restricție fiind că capacitatea e mai mare decât locurile disponibile.

Constrângerile de tip **not null** se găsesc pe majoritatea atributelor pentru a fi siguri că se vor completa și că vom avea informațiile necesare altor tabele.

Primary key-urile sunt generate de baza de date printr-un mecanism de autoincrement (**pass_id,p_id,ticket_id,flight_id,airport_id**).

Aceste constrângeri din baza de date s-au verificat și în cadrul programului, după cum urmează:

Validarea informațiilor introduse se face prin interfața, cu ajutorul unor verificări scris în html, dar și prin funcții scrise in javascript.

Pentru a valida datele de intrare m-am folosit de atributul **pattern** (pentru ca numele să nu conțină cifre, validare email, număr de telefon), **required** (verificare dacă atributul este obligatoriu sau nu), **minlength** (un text să aibă lungimea mai mare de 2 caractere).

Funcțiile în javascript sunt folosite pentru a determina unicitatea unui termen introdus, în comparație cu celelalte atribute deja introduse, pentru a verifica ca data de sosire a unui avion este mai mare sau egală decât data de decolare, de asemenea ca aeroportul de decolare să nu fie același cu cel de sosire și pentru a verifica când nu mai există locuri disponibile la un anumit zbor.

Descrierea modalității de conectare la baza de date din aplicație

Conectarea la baza de date s-a făcut utlizând cx_Oracle,un modul de extensie Python care ne permite accesul la Oracle Database. Acest modul s-a instalat din IDE-ul pycharm folosind comanda

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

De asemenea a fost nevoie și de fișierul instantclient_21_7 care se descarcă de la adresa Instant Client for Microsoft Windows (x64) 64-bit (oracle.com) și se dezarhivează în fișierul proiectului.

```
# conexiunea la baza de date
cx_Oracle.init_oracle_client(lib_dir=r"C:\Users\Gabriela\Desktop\BD-tema_de_casa\instantclient_21_7")
con_tns = cx_Oracle.makedsn('bd-dc.cs.tuiasi.ro', '1539', service_name='orcl')
try:
    con = cx_Oracle.connect('bd158', 'bd158', dsn=con_tns)

except Exception:
    print('Unexpected error')
else:
    print('Conexiune stabilita')
```

Funcționalitatea aplicației

Prin aceasta aplicație s-a implementat vizual principalele funcții de interogare a unei baze de date.

1. Funcția SELECT

Este folosita pentru obținerea datelor din baza de date.

Exemplu de utilizare:

```
@app.route("/")
@app.route('/Passenger')
def passenger():
    passengers = []
    cur = con.cursor()
    cur.execute('select * from passenger')
    for result in cur:
        passenger = {}
        passenger['P_id'] = result[0]
        passenger['First_name'] = result[1]
        passenger['Middle_name'] = result[2]
```

```
    passenger['Last_name'] = result[3]
    passenger['Email'] = result[4]
    passenger['Phone_number'] = result[5]
    passengers.append(passenger)
cur.close()
return render_template('Passenger.html', passengers=passengers)
```

Rezultatul codului este următorul:

Passenger id	First name	Middle name	Last name	Email	Phone number
47	Paul	None	Stan	paul@gmail.com	0726598456
57	Mihai	Alin	Diaconu	mihalin@gmail.com	0726598451
56	Laura	None	Stan	None	0756984231
30	Mihai	Andrei	Dobre	dobremi@gmail.com	0786951489
31	Maria	None	Popovici	pmaria1@gmail.com	0763956478

2. Funcția INSERT

Prin aceasta funcție se adaugă o noua înregistrare în tabelă

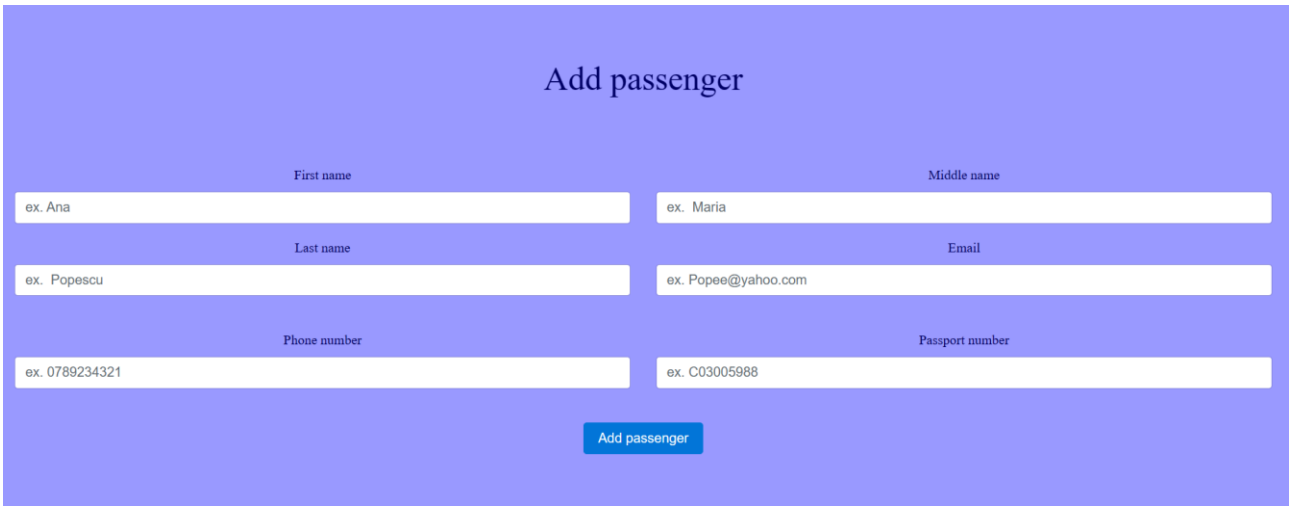
Exemplu de cod:

```
@app.route('/addPassenger', methods=['GET', 'POST'])
def addPassenger():
   if request.method == 'POST':
      cur = con.cursor()
      values = []
      id = 'NULL'
      values.append(str(id))
      first = "'" + request.form['first_name'] + "'"
      middle = "'" + request.form['middle_name'] + "'"
      last = "'" + request.form['last_name'] + "'"
      email = "'" + request.form['email'] + "'"
      phone = "'" + request.form['phone_number'] + "'"
      fields = ['P_id', 'First_name', 'Middle_name', 'Last_name', 'Email',
'Phone_number']


      pass_nr = "'" + request.form['pass_nr'] + "'"


      query = 'INSERT INTO %s (%s) VALUES
(%s,initcap(%s),initcap(%s),initcap(%s),%s,%s)' % (
      'Passenger', ', '.join(fields), id, first, middle, last, email, phone)
      cur.execute(query)
      cur.execute('commit')
      cur.close()


      cur = con.cursor()
      cur.execute('select max(p_id) from passenger')
      p_id = 0
      for result in cur:
```

```
        p_id = result[0]
    cur.close()
    cur = con.cursor()
    query1 = 'INSERT INTO PASSPORT values (NULL,UPPER(%s),%s)' % (pass_nr, p_id)
    cur.execute(query1)

    return redirect('/Passenger')
else:

    cur=con.cursor()
    pas=[]
    cur.execute('select pass_nr from passport')
    for result in cur:
        pas.append(result[0])
    cur.close()

    cur=con.cursor()
    phone=[]
    cur.execute('select phone_number from passenger')
    for result in cur:
        phone.append(result[0])
    cur.close()

    cur=con.cursor()
    email=[]
    cur.execute('select email from passenger')
    for result in cur:
        email.append(result[0])
    cur.close()
```

```
    return
render_template('addPassenger.html',Passport=pas,Phone=phone,Email=email)
```

Pagina pentru funcția de insert:



3. Funcția UPDATE

Folosită pentru a modifica datele existente cu alte valori

Exemplu de cod:

```
@app.route('/editPassenger', methods=['POST'])
def editPassenger():
    p_id = "'" + request.form['p_id'] + "'"
    first = "'" + request.form['first_name'] + "'"
    middle = "'" + request.form['middle_name'] + "'"
    last = "'" + request.form['last_name'] + "'"
    email = "'" + request.form['email'] + "'"
```

```
    phone = "'" + request.form['phone_number'] + "'"
    cur = con.cursor()
    query = "UPDATE passenger SET First_name=%s,Middle_name=%s, Last_name=%s,
Email=%s, Phone_number=%s where P_id=%s" % (
        first, middle, last, email, phone, p_id)

    cur.execute(query)

    cur.execute('commit')

    return redirect('/Passenger')
```

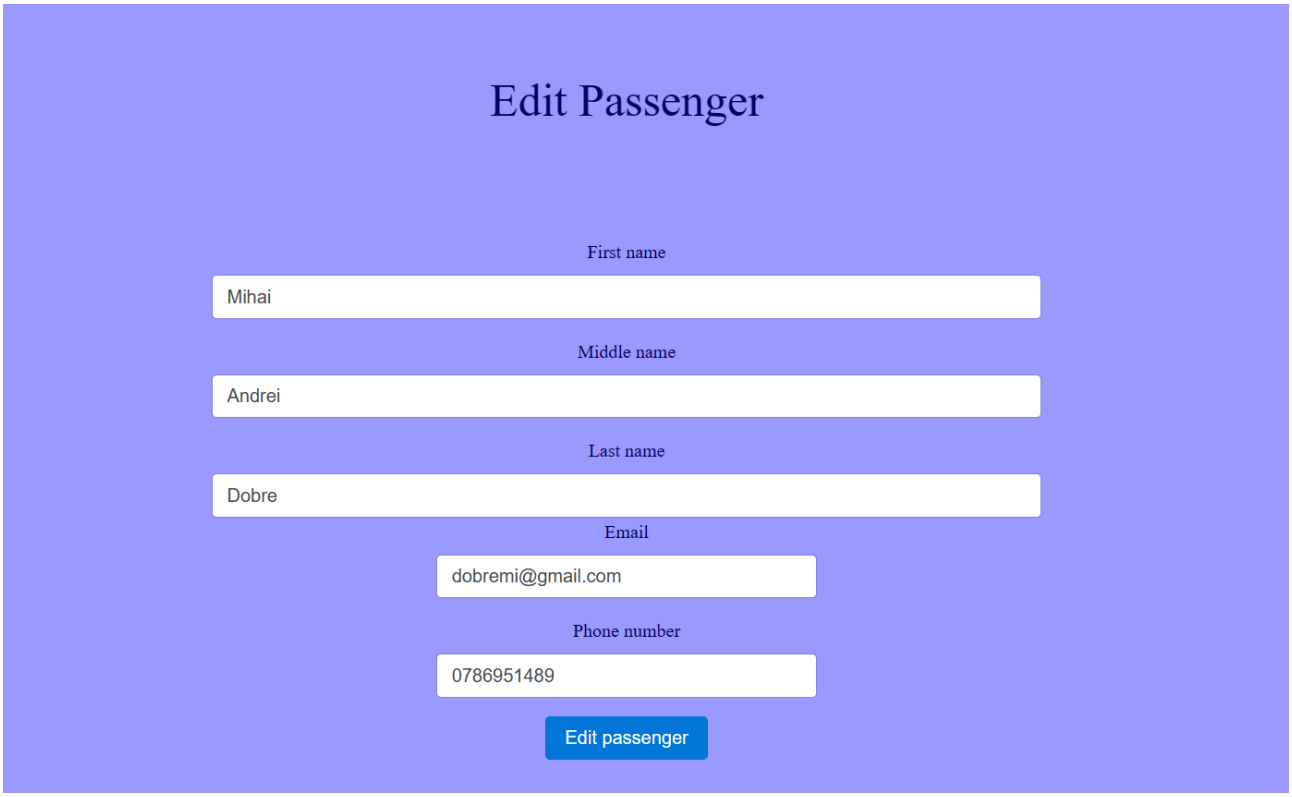
De asemenea, la această funcție de update este necesară și citirea datelor din tabelă, corespunzătoare acelei înregistrări pentru a putea modifica una sau mai multe câmpuri, restul rămânând intacte

Această funcție arată în felul următor:

```
@app.route('/getPassenger', methods=['POST'])
def getPassenger():
    id = request.form['p_id']
    cur = con.cursor()
    cur.execute('select * from passenger where p_id=' + id)

    p = cur.fetchone()
    p_id = p[0]
    first_name = p[1]
    middle_name = p[2]
    last_name = p[3]
```

```
    email = p[4]
    phone = p[5]
    cur.close()

    cur = con.cursor()
    phone_nr = []
    cur.execute('select phone_number from passenger where
phone_number!='+phone)
    for result in cur:
        phone_nr.append(result[0])
    cur.close()

    cur = con.cursor()
    mail = []
    cur.execute('select email from passenger where p_id!='+str(p_id))
    for result in cur:
        mail.append(result[0])
    cur.close()
    return render_template('editPassenger.html', P_id=p_id, First_name=first_name,
Middle_name=middle_name,
                           Last_name=last_name,
                           Email=email,
Phone_number=phone,Phone=phone_nr,Pass_email=mail)
```

Pagina pentru update arată în felul următor:



4. Funcția DELETE

Folosită pentru a șterge o înregistrare din tabelă

Exemplu de cod:

```
@app.route('/delPassenger', methods=['POST'])
def deletePassenger():
    cur = con.cursor()
    pas = request.form['p_id']
    cur.execute('delete from booking where p_id=' + str(pas))
    cur.execute('delete from passport where p_id=' + str(pas))
    cur.execute('delete from passenger where p_id=' + str(pas))
    cur.execute('commit')
    return redirect('/Passenger')
```

Restul tabelelor sunt tratate într-un mod asemănător.