A picture containing icon

Description automatically generated

**UNIVERSITATEA TEHNICĂ “GH ASACHI” IAŞI**

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**SPECIALIZAREA: CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI**

**ACHIZIȚIA ȘI PRELUCRAREA DATELOR - PROIECT**

**Gestiunea flotei de mașini a unei companii IT**

**Student**

**Marian Tudor-Constantin**

**Grupa 1310B**

**Coordonator,**

**Ș.l Mironeanu Cătălin**

**Iași, 2024**

* Titlu

Gestiunea flotei de mașini a unei companii IT

Proiectul va cuprinde analiza, proiectarea și implementarea unei baze de date care să modeleze gestiunea activității mașinilor în cadrul unei companii de IT.

* Descriere

Pentru a realiza baza de date sunt necesare mai multe informații, printre care le regăsim pe următoarele: utilizatori (angajați), tipul de autovehicul deținut de firmă, caracteristicile pe care acesta trebuie să le îndeplinească, diferite detalii referitoare la angajați.

Un client este identificat printr-un ID, nume, prenume, data nașterii, email și data la care acesta se înregistrează.

Partea cea mai importantă o reprezintă autovehiculele pe care firma le pune la dispoziția angajaților . Fiecare autovehicul va fi identificat printr-un ID. Acesta are, de asemenea, un număr prin care este identificat pe platformă, dar modelul este câmpul care delimitează autovehiculele în categorii. Totodată, o mașină deține particularități precum: marcă, culoare, model, an de fabricare.

Pentru a beneficia de oricare dintre autoturisme, angajații trebuie să realizeze o legătură între ei înșiși (users) și mașini (vehicles). Această tabelă de legătură conține doar ID-ul angajatului și ID-ul mașinii.

* Tehnologii utilizate pentru Front-end și Back-end

Am utilizat tutorialul disponibil pe moodle.

Pentru partea de Front-end am utilizat JavaScript, CSS și HTML pentru interfața aplicației cu utilizatorul.

Pentru partea de Back-end am utilizat Oracle SQL Developer, Flask pentru crearea tabelelor, modelului logic și modelului relațional.

* Descrierea funcțională a platformei

Principalele funcții care se pot întâlni sunt:

* Evidența angajaților care dețin permis de conducere
* Evidența utilizatorilor
* Evidența autovehiculelor din firmă
* Descrierea detaliată a entităților și relațiilor dintre ele

Tabelele din aplicație sunt:

* USERS
* VEHICLES
* USER\_VEHICLES
* DRIVER\_LICENSES
* MODELS
* BRANDS

În proiectarea bazei de date s-au identificat următoarele tipuri de relații:

1:1 (one-to-one)

1:n (one-to-many)

n:n (many to many)

Între tabela **USERS** și tabela **DRIVER\_LICENSES** se stabilește o relație de 1:1. Un angajat are propriul sau carnet de conducere, nu pot exista aceiași angajați cu același permis de conducere. Legătura între cele două tabele se face prin câmpul *User\_Id*.

Între tabela **VEHICLES** și tabela **MODELS** se realizează o relație de 1:n. Un model poate fi asociat mai multor mașini. Legătura dintre cele două tabele se face prin câmpul **Model\_ID**. Prin această legătură, fiecare înregistrare din tabela **VEHICLES** este asociată cu un model specific din tabela **MODELS**. În același timp, un model din tabela **MODELS** poate fi asociat cu mai multe vehicule din tabela **VEHICLES**.

Între tabela **MODELS** și tabela **BRANDS** se realizează o relație tot de 1:n, întrucât un brand specific poate avea mai multe modele de mașini. Legărtura se realizează prin intermediul coloanei *Brand\_Id* din tabela **MODELS**. Atât tabela **MODELS** cât și tabela **BRANDS** sunt extensii ale tabele **VEHICLES**, utilizate pentru o mai bună descriere a acesteia.

Tabela **USER\_VEHICLES** apare ca o necesitate, întrucât între tabelele principale **USERS** și **VEHICLES** apare o relatie de n:n: n users pot avea n masini. În acest caz, relația many to many a fost împărțită în două relații 1:n.

* Descrierea detaliată a constrângerilor

Tipuri de constrângeri:

* CHECK

Constrângerile de tip check se găsesc în aproape toate tabelele. Se verifică dimensiunea valorilor introduse pentru nume, prenume, numar de înmatriculare, seria de șasiu (VIN), data de expirare a permisul de conducere. Toate acestea pentru a nu introduce în baza de date greșeli de scriere, apărând o eroare în cazul uneia, dar și pentru respectarea unor anumite formate în cazul VIN-ului și a numărului de înmatriculare.

Data de expirare a permisului de conducere este de zece ani de la data eliberării.

Constrângerile de tip check sunt folosite pentru a verifica formatul email-ului clientului. Email-ul poate conține litere sau cifre, @ și . .

* UNIQUE

Constrângerile de tip UNIQUE se utilizează pentru atributele Email (pentru a nu avea doi clienți cu același email).

* NOT NULL

Constrângerile de tip null se regăsesc în majoritatea tabelelor. Primary key-urile sunt id-urile din fiecare tabelă ele fiind generate automat prin *auto-increment* și*identity*. Pe lângă cheile primare, alte tabele care respectă constrangerea de NOT NULL sunt datele esențiale referitoare la vehicule (VIN, LICENSE\_PLATE, REGISTRATION\_DATE, MODEL\_ID), numele, prenumele și email-ul userului cât și întregul conținut al tabelei DRIVER\_LICENSES.

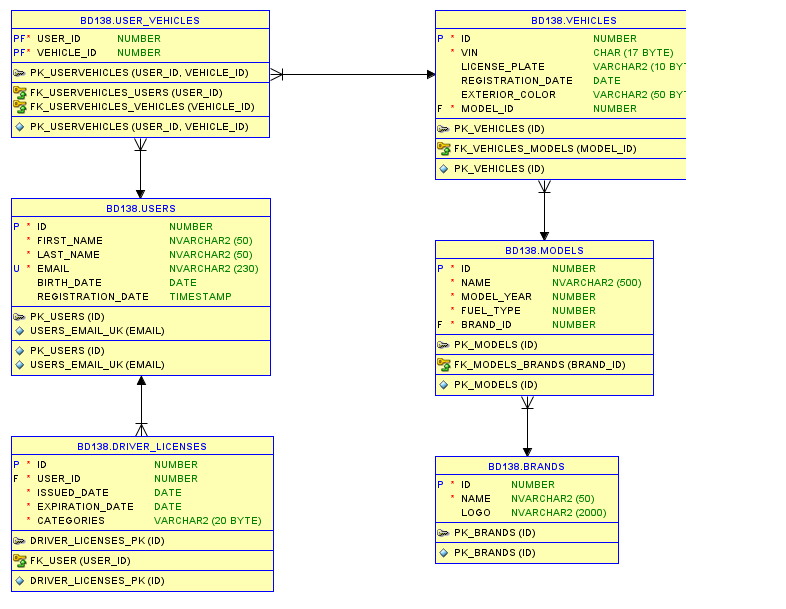
* FOREIGN KEY

Constrângerile Foreign Key au fost utilizate pentru a modela diferite tipuri de relații între tabele:

-Relații 1:n în VEHICLES, MODELS, BRANDS

-Relații 1:1 în DRIVER\_LICENSES

* Diagrama relațională

****

* Normalizări:

Entitățile USERS, USER\_VEHICLES, VEHICLES, BRANDS, MODELS, DRIVER\_LICENSES îndeplinesc condițiile primei forme normale deoarece nu există atribute cu valori multiple și grupuri de atribute care să se repete.

De asemenea, entitățile satisfac condițiile celei de-a doua forme normale deoarece atributele care nu sunt Primary Key depind funcțional de acestea.

În fiecare tabelă am introdus Primary keys pe care le-am realizat cu autoincrement, asigurând unicitatea lor. Se verifică acest lucru și la inserarea datelor în tabele, fiecare câmp acceptând o singură valoare.

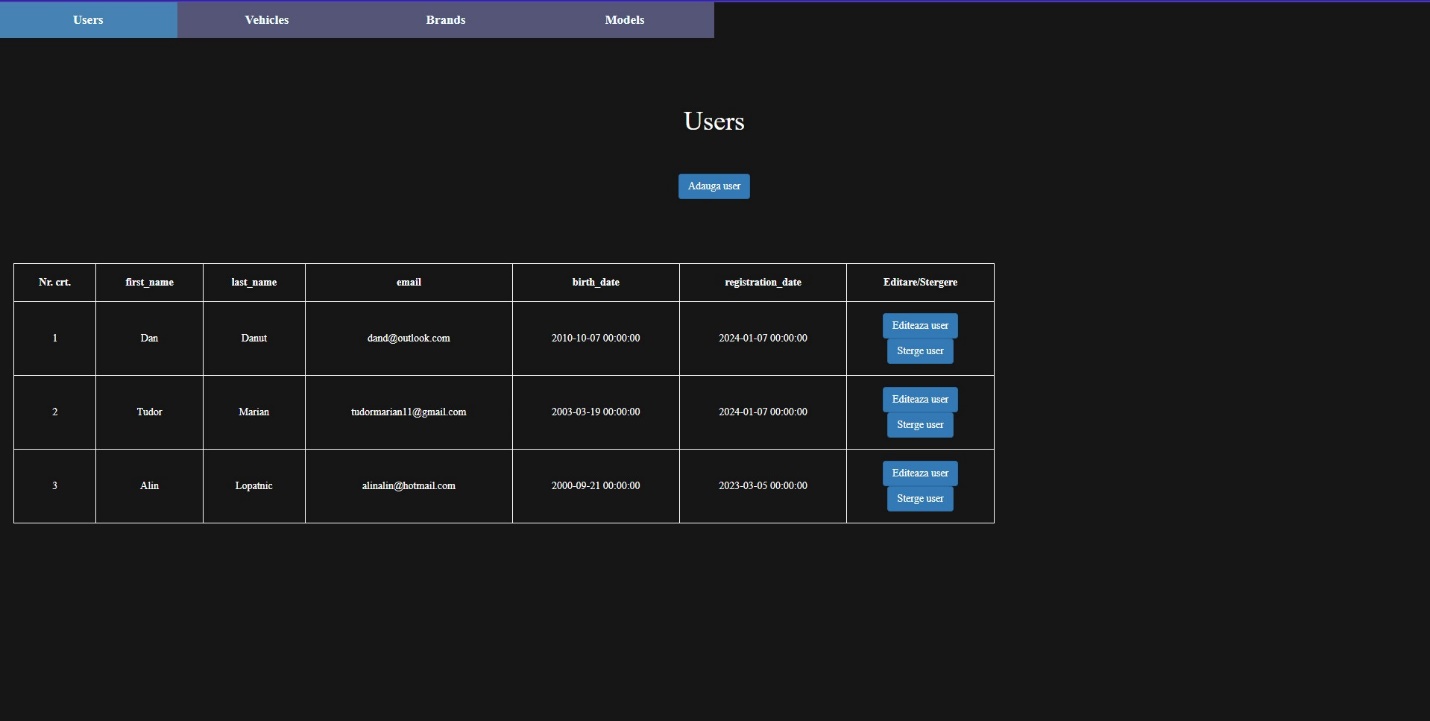
A treia formă normală se verifică prin dependența unui atribut UID de un alt element non UID.

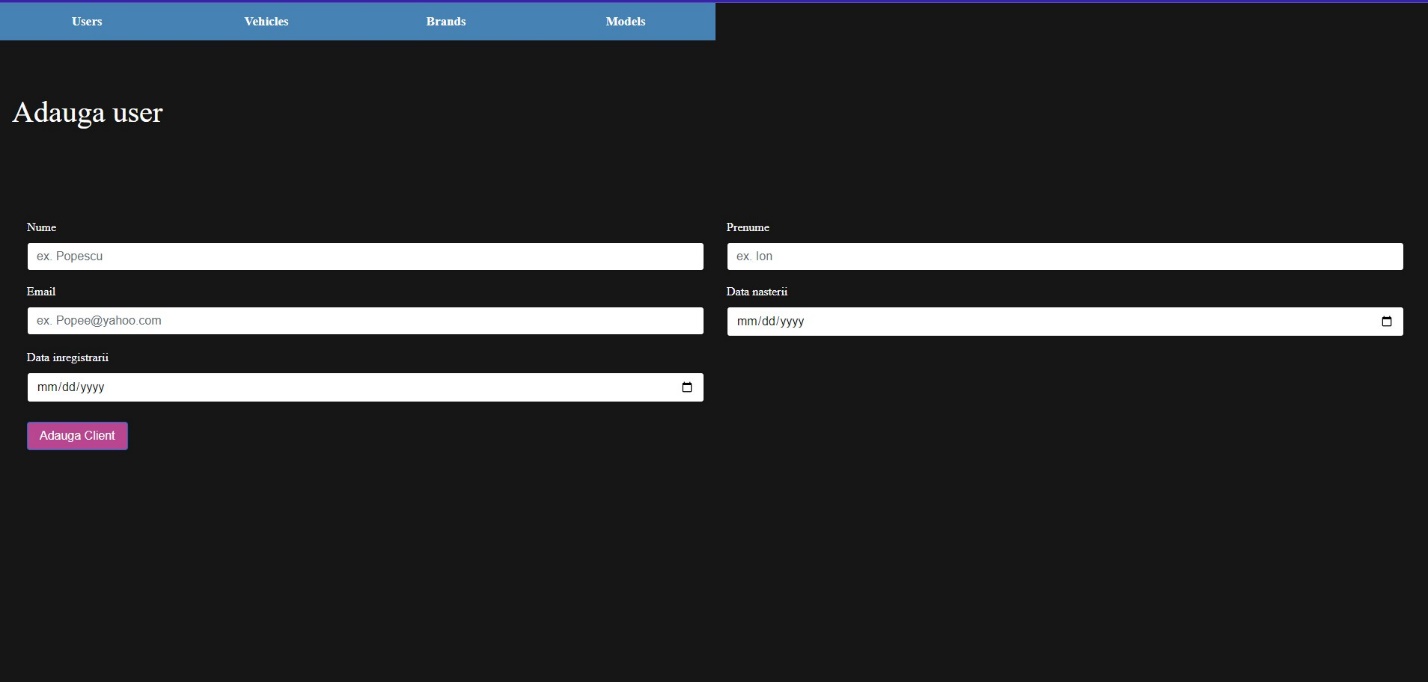
* Descrierea modalității de conectarea la baza de date din aplicație

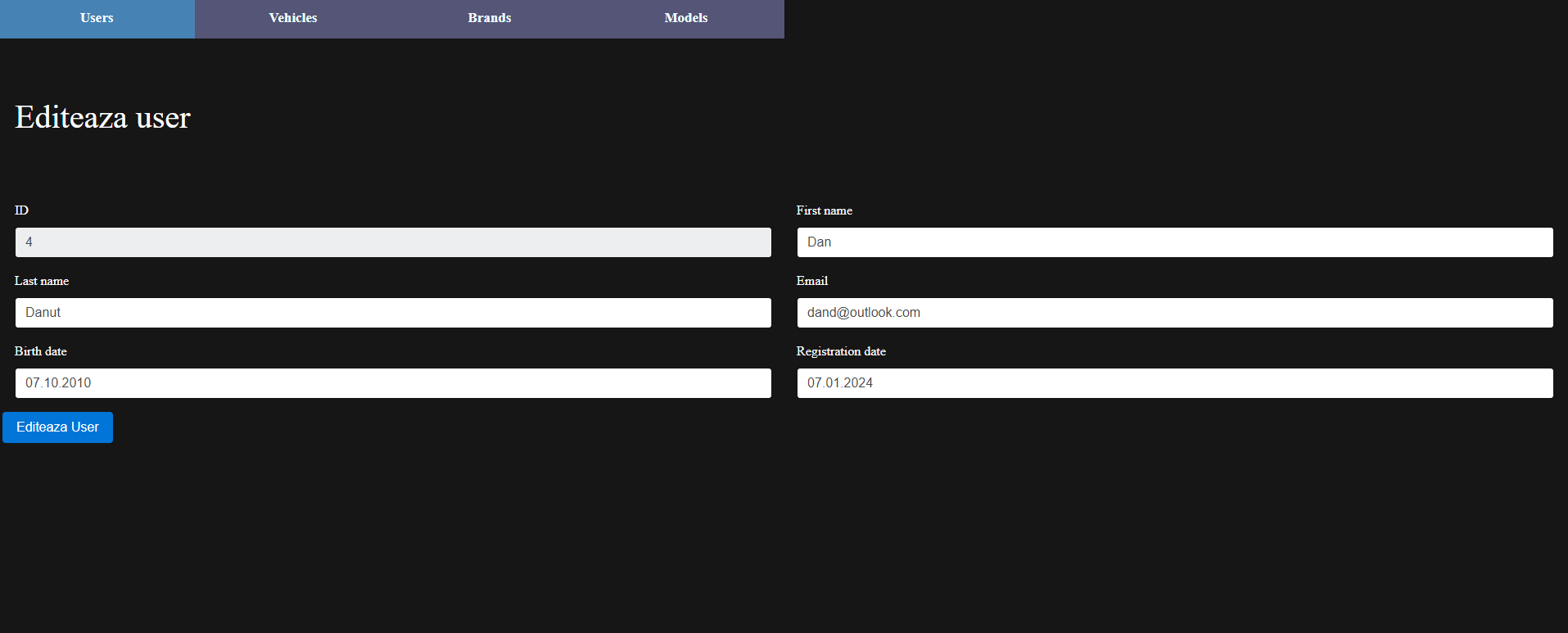
app = Flask(\_\_name\_\_)  
print('Connecting to Oracle...')  
username = "bd138"  
password = "bd138"  
dsn = "(DESCRIPTION = (ADDRESS\_LIST =(ADDRESS = (PROTOCOL = TCP)(HOST = bd-dc.cs.tuiasi.ro)(PORT = 1539)))(CONNECT\_DATA =(SERVICE\_NAME = orcl)))"  
con = cx\_Oracle.connect(username, password, dsn)  
print(f'Successfully connected to {username}! Oracle Database version: ', con.version)

Pentru a face conectarea aplicației la baza de date, am folosit cx\_Oracle.connect, utilizând contul de oracle din SQL Developer.

* Aspect interfață pentru tabela Users

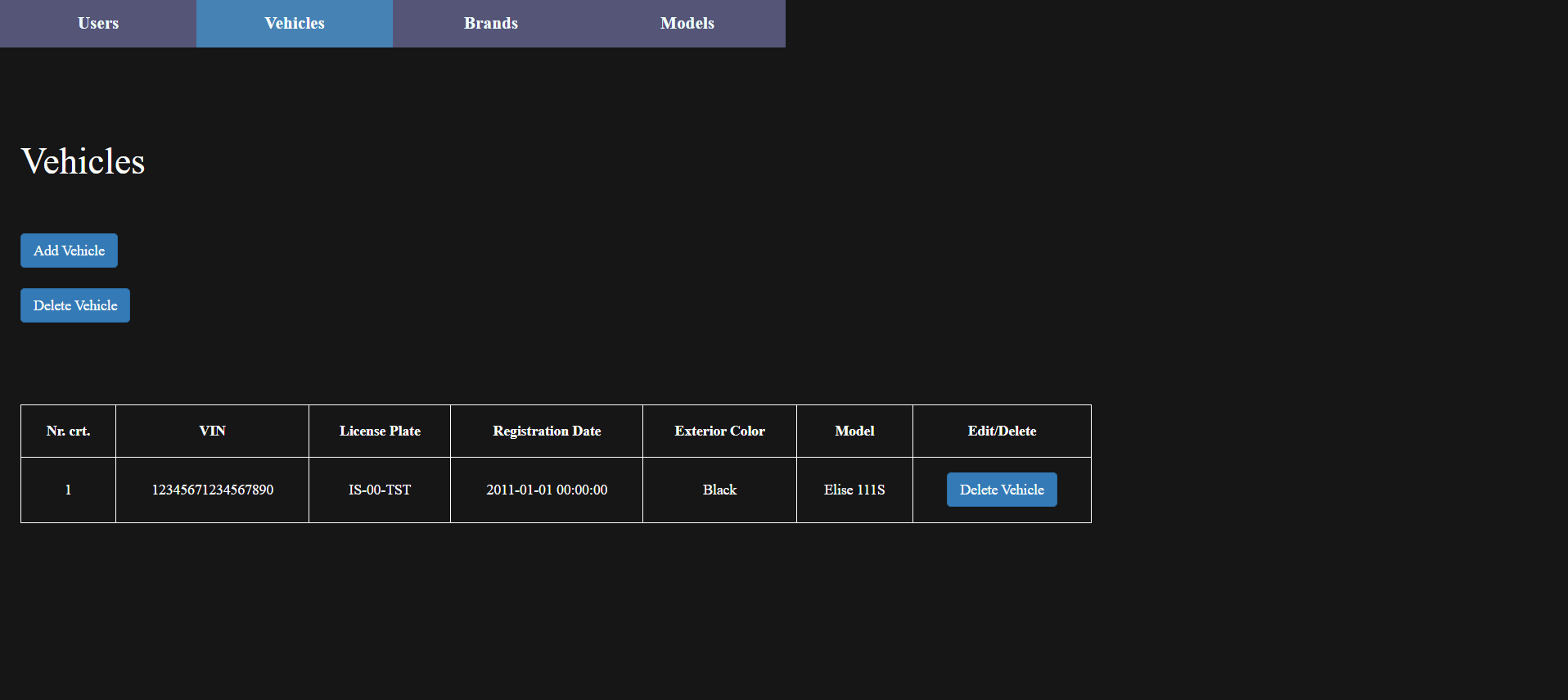


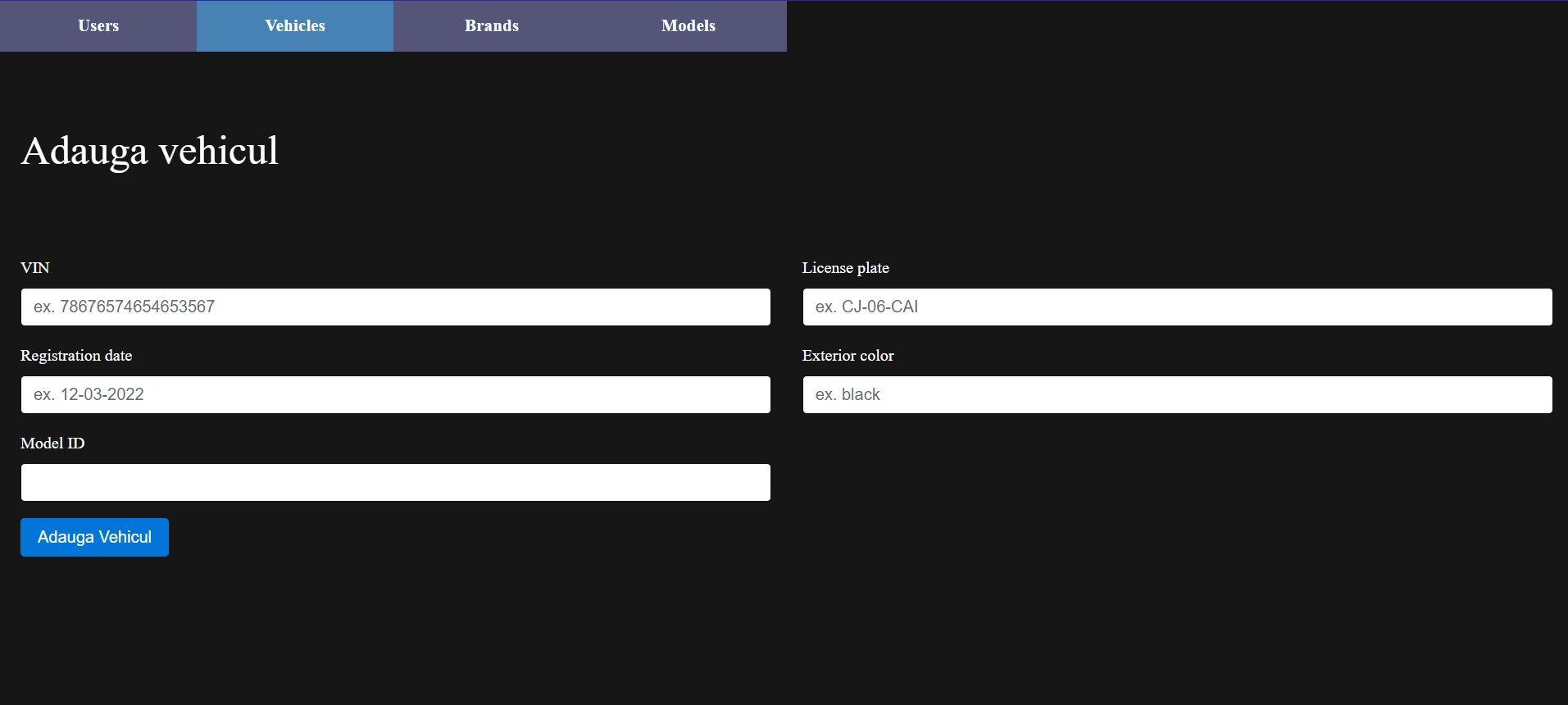




# users begin code  
  
@app.route('/')  
@app.route('/users')  
def Sel\_User():  
 users=[]  
 cur = con.cursor()  
 cur.execute('select \* from users')  
 for result in cur:  
 user = {'id': result[0], 'first\_name': result[1], 'last\_name': result[2], 'email': result[3],  
 'birth\_date': result[4], 'registration\_date': result[5]}  
 users.append(user)  
 cur.close()  
 return render\_template('users.html', users=users)  
  
  
@app.route('/addUser', methods=['GET', 'POST'])  
def add\_user():  
 error = None  
 if request.method == 'POST':  
 cur = con.cursor()  
 cur.execute('select max(id) from users')  
 for result in cur:  
 id = result[0]  
 cur.close()  
 if id is None:  
 id = 0  
 id += 1  
  
 cur = con.cursor()  
  
 values = []  
 values.append("'" + str(id) + "'")  
 values.append("'" + request.form['first\_name'] + "'")  
 values.append("'" + request.form['last\_name'] + "'")  
 values.append("'" + request.form['email'] + "'")  
 values.append("'" + datetime.strptime(str(request.form['birth\_date']), '%Y-%m-%d').strftime('%d-%b-%y') + "'")  
 values.append("'" + datetime.strptime(str(request.form['registration\_date']),'%Y-%m-%d').strftime('%d-%b-%y') + "'")  
  
 fields = ['id', 'first\_name', 'last\_name', 'email', 'birth-date', 'registration\_date']  
 query = f"INSERT INTO (SELECT id, first\_name, last\_name, email, birth\_date, registration\_date FROM users) VALUES ({id},{values[1]}, {values[2]}, {values[3]}, {values[4]}, {values[5]})"  
  
 cur.execute(query)  
 cur.execute('commit')  
 return redirect('/users')  
 else:  
 return render\_template('addUser.html')  
  
@app.route('/deleteUser', methods=['GET', 'POST'])  
def del\_user():  
 if request.method == 'POST':  
 aux = request.form['id']  
 print(aux)  
 cur = con.cursor()  
 cur.execute('delete from user\_vehicles where user\_id = ' + aux)  
 cur.execute('commit')  
 cur.close()  
 cur = con.cursor()  
 cur.execute('delete from users where id = ' + aux)  
 cur.execute('commit')  
  
 # cur.close()  
  
 return redirect('/users')  
 else:  
 return render\_template('deleteUser.html')  
  
@app.route('/editUser', methods=['GET', 'POST'])  
def edit\_user():  
 user = 0  
 cur = con.cursor()  
  
 id = "'" + request.form['id']+"'"  
 cur.execute('select id from users where id =' + id)  
 for result in cur:  
 user = result[0]  
 cur.close()  
  
 first\_name = "'" + request.form['first\_name'] + "'"  
 last\_name = "'" + request.form['last\_name'] + "'"  
  
 email = "'" + request.form['email'] + "'"  
 birth\_date = "'" + request.form['birth\_date'] + "'"  
 registration\_date = "'" + request.form['registration\_date'] + "'"  
  
 cur = con.cursor()  
 query = "UPDATE users SET first\_name = %s, last\_name = %s, email = %s, birth\_date = %s, registration\_date = %s WHERE id = %s" % (first\_name, last\_name, email, birth\_date, registration\_date, user)  
 cur.execute(query)  
  
 return redirect('/users')  
  
@app.route('/getUser', methods=['POST'])  
def get\_user():  
 user = request.form['id']  
 cur = con.cursor()  
 cur.execute('SELECT \* FROM users WHERE id =' + user)  
  
 usrs = cur.fetchone()  
 id = usrs[0]  
 first\_name = usrs[1]  
 last\_name = usrs[2]  
 email = usrs[3]  
 birth\_date = datetime.strptime(str(usrs[4]), '%Y-%m-%d %H:%M:%S').strftime('%d.%m.%Y')  
 registration\_date = datetime.strptime(str(usrs[5]), '%Y-%m-%d %H:%M:%S').strftime('%d.%m.%Y')  
  
 cur.close()  
 return render\_template("editUser.html", id = id, first\_name = first\_name, last\_name = last\_name, email = email,  
 birth\_date = birth\_date, registration\_date = registration\_date)  
  
# users end code  
# -------------------------------------------#

* Aspect interfață pentru tabela Vehicles





vehicles begin code  
  
@app.route('/vehicles')  
def Select\_vehicle():  
 vehicles = []  
  
 cur = con.cursor()  
 cur.execute('SELECT \* FROM vehicles')  
 for result in cur:  
 vehicle = {}  
 vehicle['id'] = result[0]  
 vehicle['vin'] = result[1]  
 vehicle['license\_plate'] = result[2]  
 vehicle['registration\_date'] = result[3]  
 vehicle['exterior\_color'] = result[4]  
 vehicle['model\_id'] = result[5]  
  
 # Obține numele modelului pentru vehicul  
 cur.execute('SELECT Name FROM models WHERE ID = :model\_id', {'model\_id': result[5]})  
 model\_name = cur.fetchone()  
 vehicle['name'] = model\_name[0] if model\_name else None  
 vehicles.append(vehicle)  
 print(vehicles)  
 return render\_template('vehicles.html', vehicles=vehicles)  
  
@app.route("/addVehicle", methods=['GET', 'POST'])  
def Add\_vehicle():  
 error = None  
 if request.method == 'POST':  
 veh = 0  
 cur = con.cursor()  
 cur.execute('select max(id) from vehicles')  
 for result in cur:  
 veh = result[0]  
 cur.close()  
 veh += 1  
  
  
 cur = con.cursor()  
 values = []  
 values.append("'" + str(veh) + "'")  
 values.append("'" + request.form['vin'] + "'")  
 values.append("'" + request.form['license\_plate'] + "'")  
 values.append("'" + request.form['registration\_date'] + "'")  
 values.append("'" + request.form['exterior\_color'] + "'")  
 values.append("'" + request.form['model\_id'] + "'")  
 query1 = f"INSERT INTO vehicles(vin, license\_plate, registration\_date, exterior\_color, model\_id) VALUES ({values[1]},{values[2]},{values[3]}, {values[4]},{values[5]})"  
 print(query1)  
 cur.execute(query1)  
 cur.execute('commit')  
 return redirect('/vehicles')  
 else:  
 model = []  
 cur = con.cursor()  
 cur.execute('select id from models')  
 for result in cur:  
 model.append(result[0])  
 cur.close()  
 return render\_template('addVehicle.html')  
  
  
@app.route('/delVehicle', methods=['GET', 'POST'])  
def Delete\_Vehicle():  
 id = request.form['id']  
 cur = con.cursor()  
 cur.execute('DELETE FROM vehicles WHERE id=' + id)  
 cur.execute('COMMIT')  
 return redirect('/vehicles')