**UNIVERSITATEA TEHNICA “GHEORGHE ASACHI” IASI**

**FACULTATEA DE AUTOMATICA SI CALCULATOARE**

DISCIPLINA : BAZE DE DATE

GESTIONAREA ACTIVITATII INTR-UN CABINET VETERINAR

**COORDONATOR: Sorin Avram**

**STUDENT:Dobos Ioana-Victoria**

**Bruma Elena-Petruta**

**GRUPA: 1308B**

**Iasi, 2020**

**Titlu proiect : Tinerea in evidenta a activitatii dintr-un cabinet veterinar**

Proiectul isi propune sa stocheze intr-o baza de date clientii, medicii, pacientii (animalele), programarile aferente unui cabinet veterinar pentru o mai buna evidenta a acestora.

In cadrul aplicatiei dezvoltate se permite adaugarea/stergerea/modificarea de clienti si de programari.

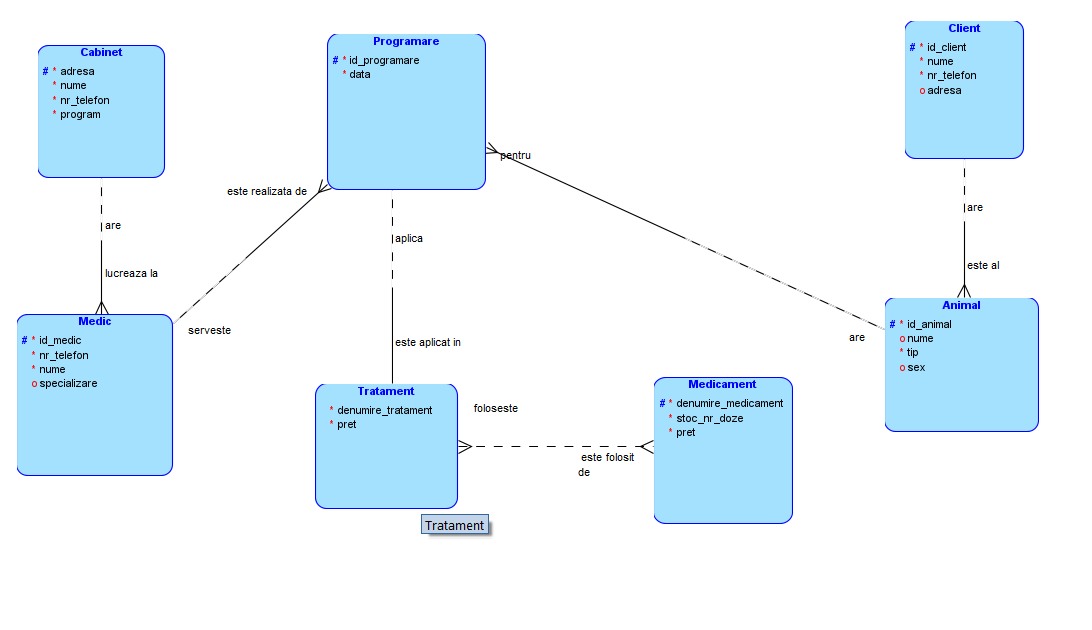
**Descrierea cerintelor si modul de organizare a proiectului**

Un cabinet veterinar, la fel ca orice afacere, presupune pe langa altele un volum de informatii care se prefera a fi structurat intuitiv astfel incat regasirea acestora sa fie cat mai usoara si la indemana. Este important pentru un cabinet bine organizat sa se tina evidenta clientilor, a animalelor acestora, a personalului, a programarilor din cadrul sau si a stocului de medicamente folosite, prin urmare aplicatia implementata isi propune sa rezolve acest aspect.

Informatiile de care avem nevoie :

1. **Cabinet :** ne intereseaza sa stim adresa cabinetului, numele sau, numarul de telefon pentru urgente sau programari si programul.
2. **Medic** : entitatea memoreaza medicii inregistrati, iar distinctia se face prin intermediul unui id; ne intereaza de asemenea numele, numar de telefon si eventual specializarea sa si neaparat adresa cabinetului la care lucreaza.
3. **Client** : clientii sunt inregistrati in sistem in mod unic cu ajutorul unui id; se memoreaza numele si numarul de telefon, eventual adresa la care sta ( pentru deplasari de urgenta la domiciliu )
4. **Animal** : animalele sunt inregistrate in mod unic cu ajutorul unui id, este optional sa stim numele sau sexul, dar trebuie sa stim neaparat tipul (caine, pisica, hamster, elefant, girafa etc); animalele sunt asignate unui singur client.
5. **Medicament** : entitatea memoreaza medicamentele prin intermediul denumirii si, de asemenea se doreste a se sti stocul respectivului medicament si pretul (practic am gandit acest proiect sa serveasca doar un cabinet veterinar, prin urmare aceasta entitate se judeca din acest punct de vedere si nu pentru multiple cabinete unde logica aplicata aici nu ar mai fi valida).
6. **Programare** : entitatea ce stocheaza intalnirile programate. Aici se retin toate intalnirile pe care le-au avut vreodata animalele cu medicii, medicul respectiv si data.
7. **Tratament** : entitate ce retine tratamentul aplicat unui animal daca acesta exista (spre exemplu o programare poate insemna o simpla constatare care nu presupune neaparat un tratament), el poate fi format sau nu din mai multe medicamente; se retine si pretul acestuia.

Schema logica :



**Tabelele din aceasta aplicatie sunt :**

* Cabinet
* Programare
* Medic
* Tratament
* Client
* Animal
* tratament\_medicament\_fk : asigura relatia de multi la multi intre tratament si medicament (un tratament nu inseamna mereu acelasi set de medicamente folosite, spre exemplu tratamentul “injectie pentru febra” poate presupune medicamente diferite de la caz la caz in functie de tipul animalului sau chiar varsta acestuia)
* Medicament

**Tehnologii folosite pentru back-end si front-end:**

* **back-end:** micro web framework-ul Flask oferit de python
* **front-end**: HTML5, CSS, Bootstrap

**Detalii despre relatiile care exista intre entitatile din cadrul proiectului :**

In realizarea proiectului se identifica relatii de tipul 1:n, 1:1 si n:m.

Intre tabela Cabinet si Medic exista o relatie 1:n cu sursa optionala in ideea in care un medic inregistrat lucreaza neaparat la un cabinet si numai unul, iar un cabinet poate avea sau nu un anumit medic in cadrul personalului propriu. (ar avea mai mult sens daca proiectul ar fi fost dezvoltat cu gandul de cabinete multiple, dar acest aspect este limitat la un cabinet unic, prin urmare am ales cea mai potrivita relatie chiar daca nu are - poate - sens in totalitate).

Intre tabela Medic si Programare este o relatie de 1:n cu sursa optionala, aceasta relatie permite ca un medic sa serveasca (sa aiba) SAU NU o programare/mai multe din cele existente, iar o programare sa fie asignata obligatoriu unui medic.

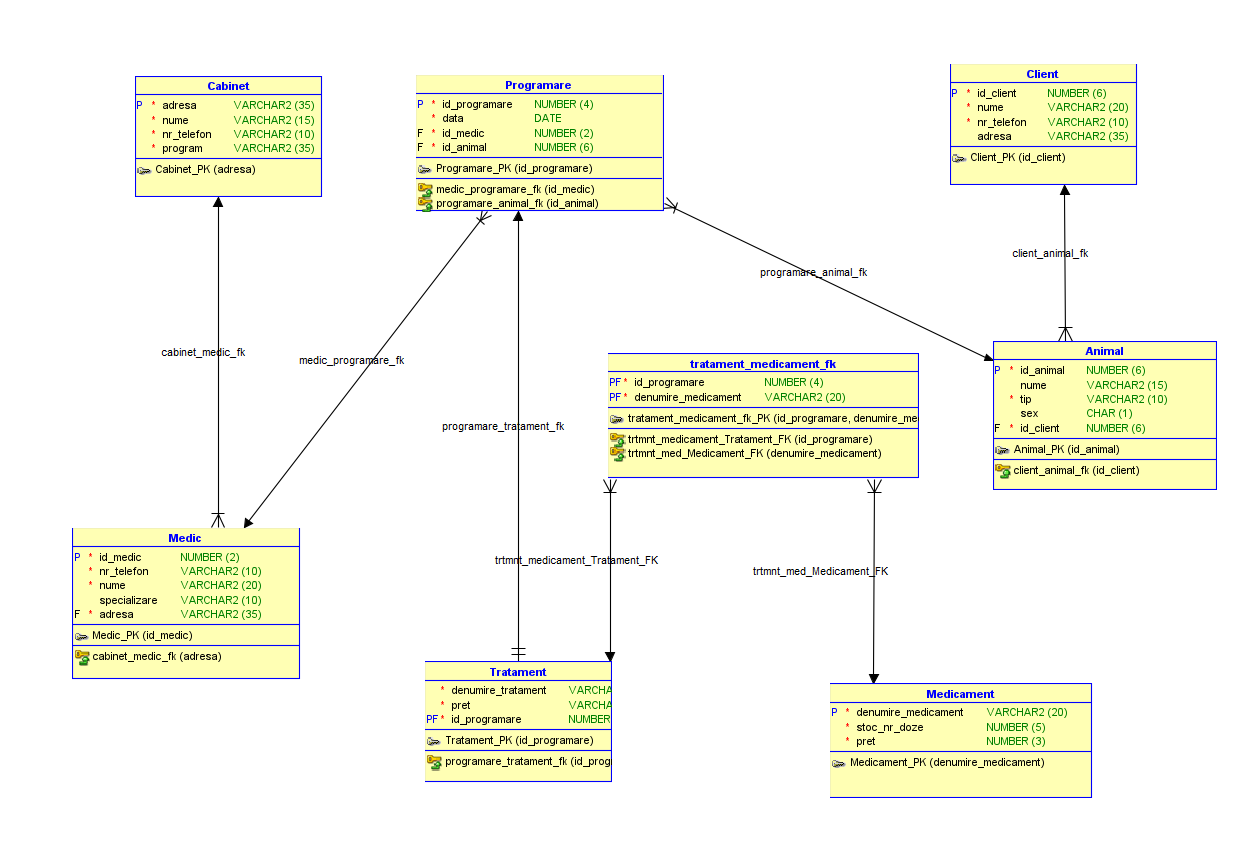
Intre tabela Programare si Tratament exista o relatie de 1:1 care reda situatia in care intr-o programare se efectueaza sau nu un tratament (sursa este optionala); daca exista un tratament, el este obligatoriu legat de o programare si numai una.

Intre entitatea Tratament si Medicament exista o relatie m:n ceea ce permite aplicarea in mod flexibil din punct de vedere a medicamentatiei in cadrul unui tratament (acelasi tratament poate fi realizat in alt mod datorita diversitatii actuale in materie de produse farmaceutice); aceasta relatie se transforma in modelul relational in tabela tratament\_medicament\_fk ce are ca intrari id\_programare si denumire\_medicament(normalizare 3FN), aspect ce introduce foarte multa flexibilitate structurii pentru regasirea informatiilor specifice fiecarei programari.

Intre tabela Animal – Programare este relatie 1:n cu sursa optionala (nu fiecare animal are o programare, un animal poate avea mai multe programari, iar o programare daca exista ea este asignata obligatoriu unui animal).

Intre tabela Client si Animal exista iarasi exista o relatie 1:n ce denota faptul ca un client poate avea mai multe animale, dar un animal este inregistrat doar cu un singur stapan.

Diagrama ER:



**Descrierea modalitatii de conectare la baza de date din aplicatie:**

Conexiunea la baza de date se realizeaza cu ajutorul functiei connect din cadrul modului cx\_Oracle

**Constrangerile folosite :**

1. Tip check :

- pentru a bloca cat mai mult inserarea de date eronate in BD, in randul numerelor de telefon am impus o verificare regex ca aceste sa respecte formatul 07[inca 8 numere];

- atributul sex din cadrul tabelei Animal poate lua doar valoarea f sau m (lista de valori )

- pretul unui tratament este verificat iarasi cu un regex, acesta contine doar cifre, nu poate incepe cu 0 si are maxim 5 cifre, minim 1.

- specializarea medicului contine doar litere (cu posibilitatea de spatiu intre cuvinte)

- atributele nume nu pot contine litere (exceptand cabinetul)

- lungimea inputului este verificata ori implicit prin regex, ori prin length ( atribut ) < numar\_maxim + 1 (spre exemplu la numele clientului este folosit acesta metoda ).

1. Tip NOT NULL

* memorarea de atribute mai degraba descriptive decat folositoare s-a evitat pentru a nu complica proiectul, prin urmare este mai efficient de numarat care sunt cele ce nu folosesc acest tip de constrangere : specializarea medicului (un medic poate fi specializat in pisici, dar poate trata si cazuri usoare ale cainilor, elefantilor, girafelor etc.), sexul si numele animalului (presupunem ca nu intereseaza neaparat) si adresa clientului ( el alege daca vrea sa isi impartaseasca adresa sau nu).

1. Tip PRIMARY KEY

* toate tabele prezinta un primary key care poate fi varchar (adresa cabinetului, se presupune ca aplicatia se extinde si poate referi mai multe cabinete pe viitor, dar nu trece de nivel local, prin urmare adresa trebuie sa fie unica; denumire\_medicament care e unica deoarece in caz de concentratii diferite ale aceluiasi medicament, se trece si concentratia in denumire) sau numeric cu autoincrementare (id\_medic, id\_animal, id\_client, id\_programare).

1. Tip UNIQUE

* orice primary key existent e implicit unique; nu s-a constatat necesitatea unui atribut unique inafara de acestea momentan.

1. Tip FOREIGN KEY

* in urma trasarii relatiilor intre tabele au rezultat diverse chei straine ce sunt mandatory in tabelele copil, de asemenea cheia straina id\_programare din cadrul entitatii tratament este si PK al acesteia deoarece este convenabil de asociat denumirile medicamentelor folosite in cadrul unui tratament cu programarea care a aplicat tratamentul respectiv.

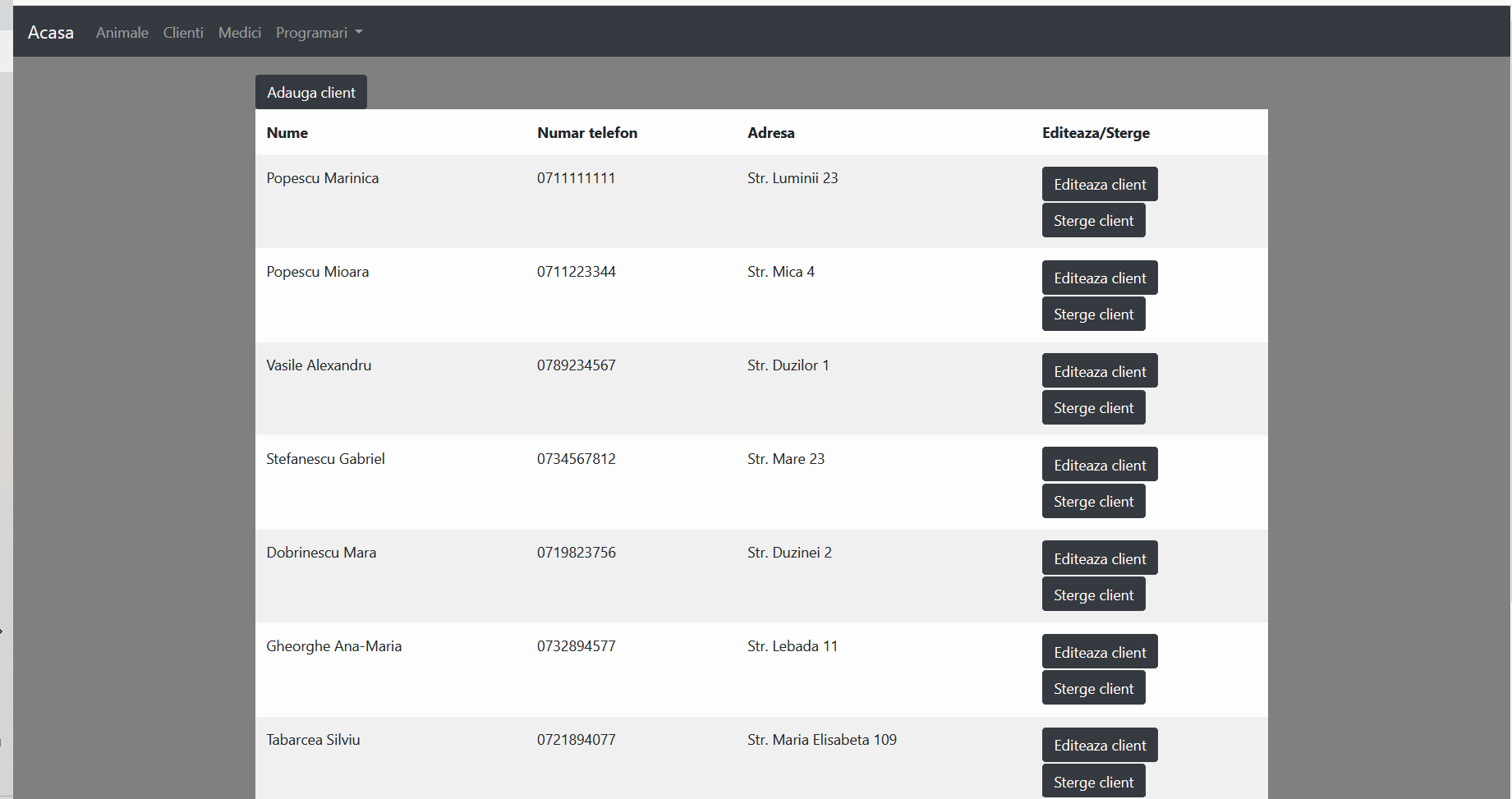
1. Tip TRIGGER

* se verifica ca data programarii introdusa sa nu fie mai mica decat data curenta.

**Capturi:**

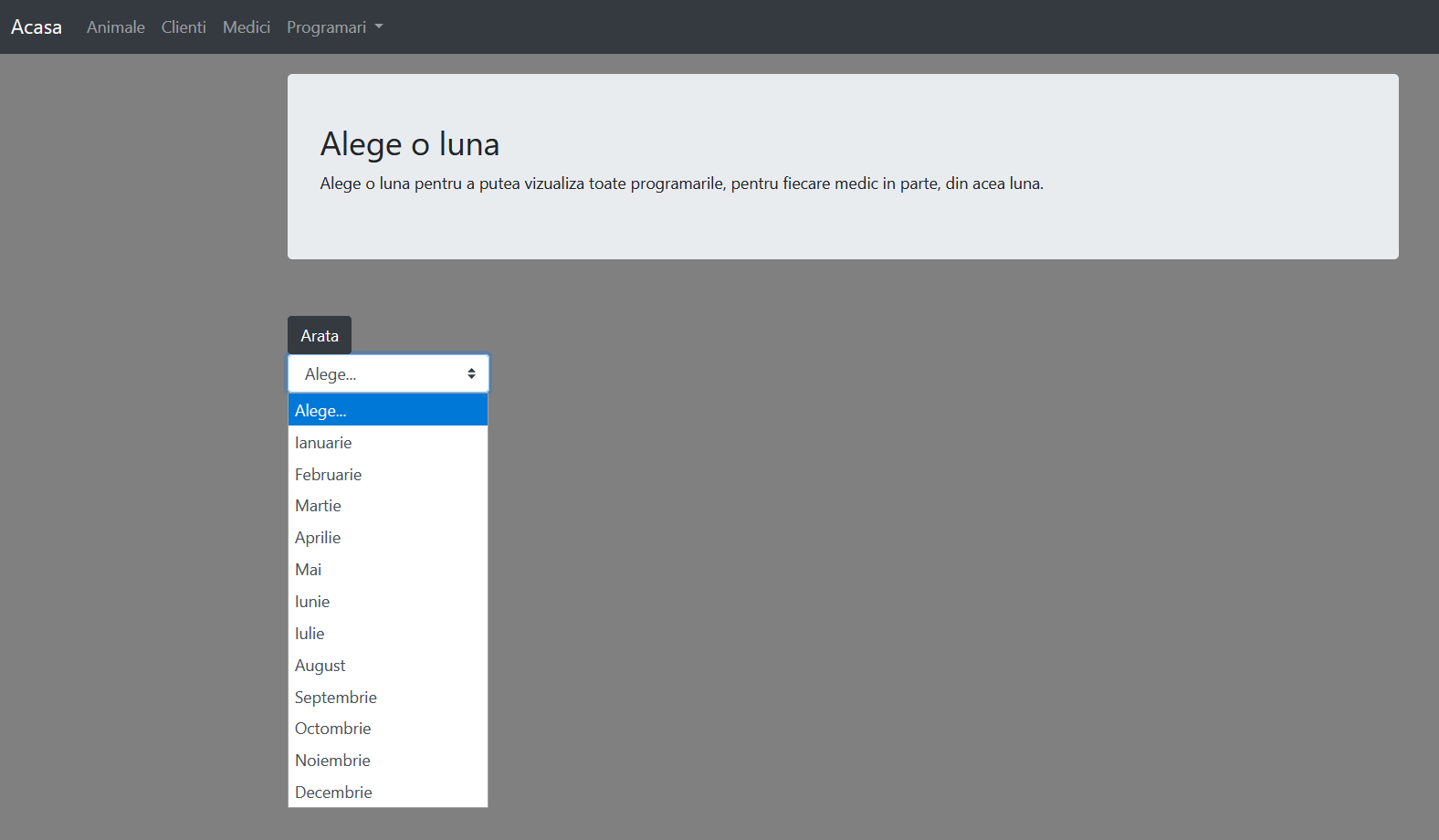
1)Exemplu de afisare

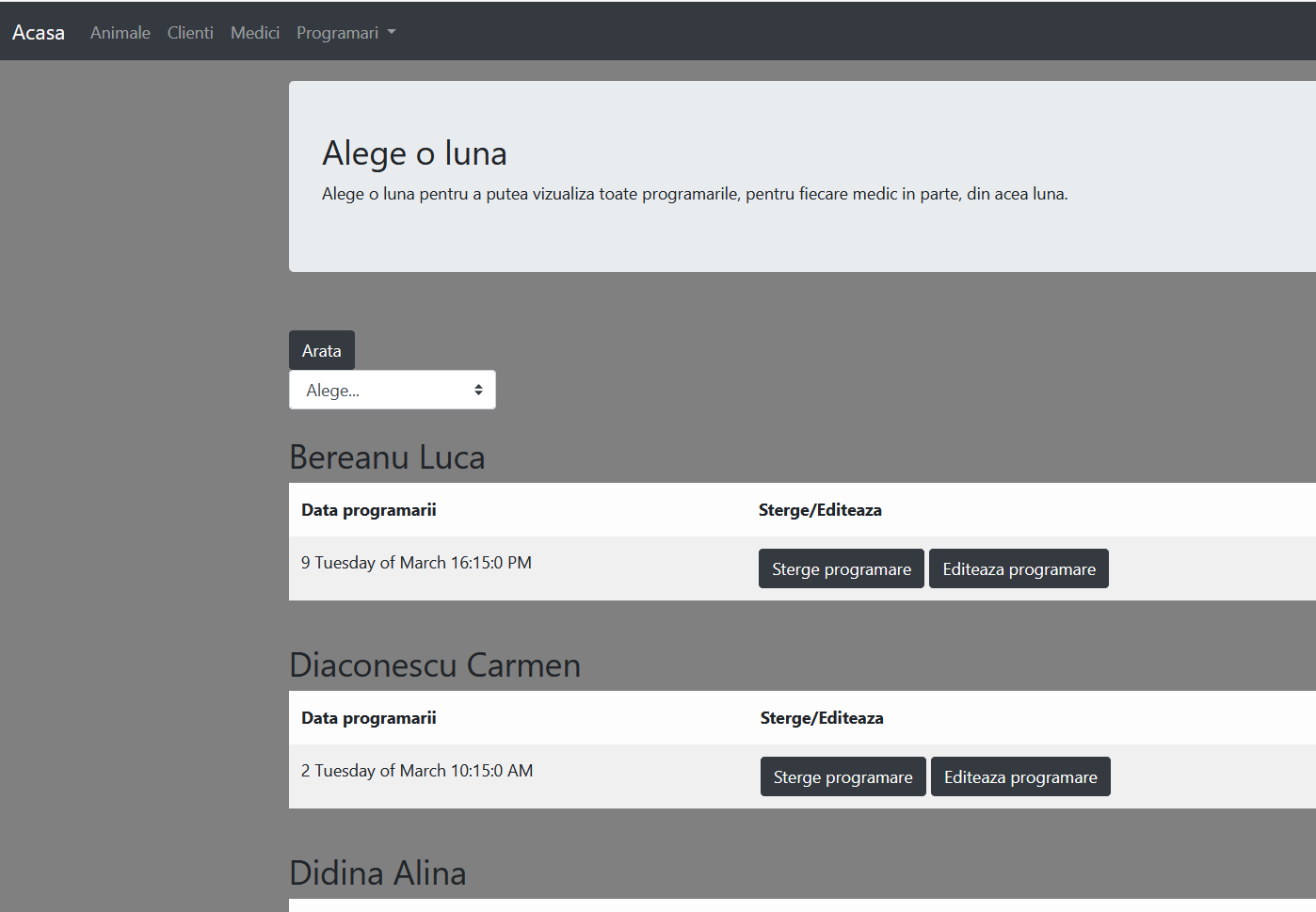
- afisare simpla: (de ex tabela client)



- afisare prelucrata: (de exemplu afisarea programarilor dintr-o luna selectata)

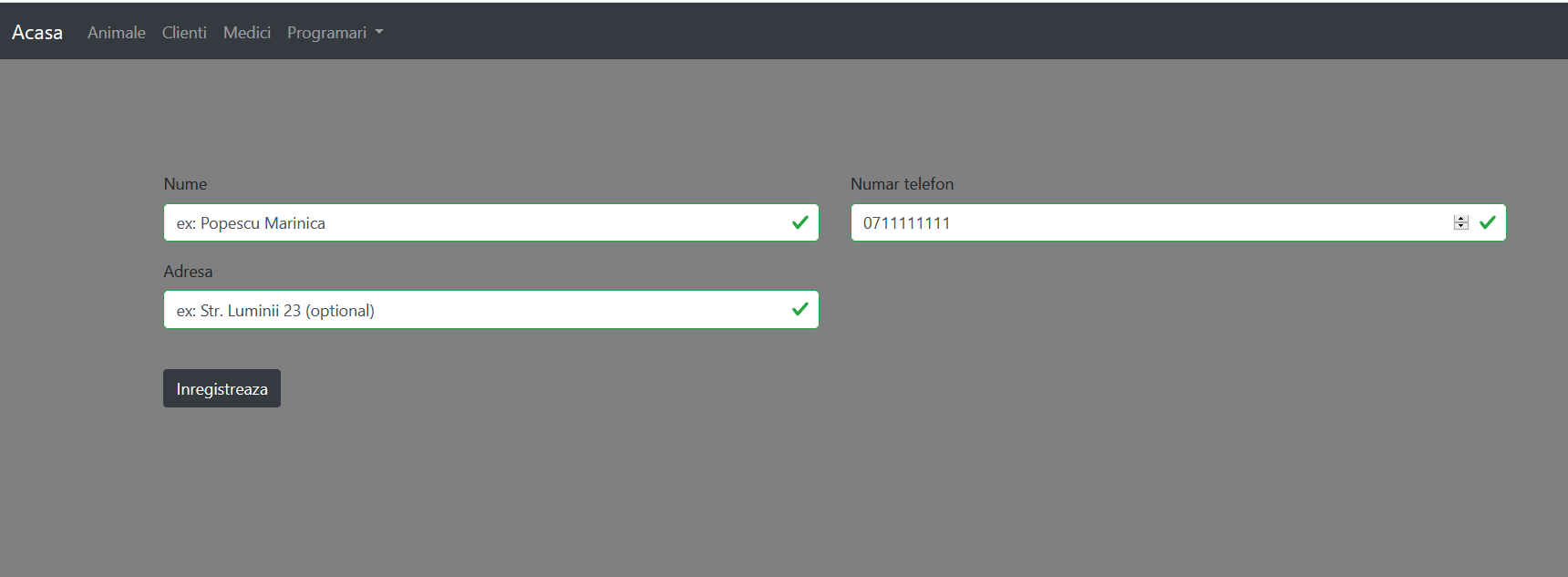
1)alegere luna



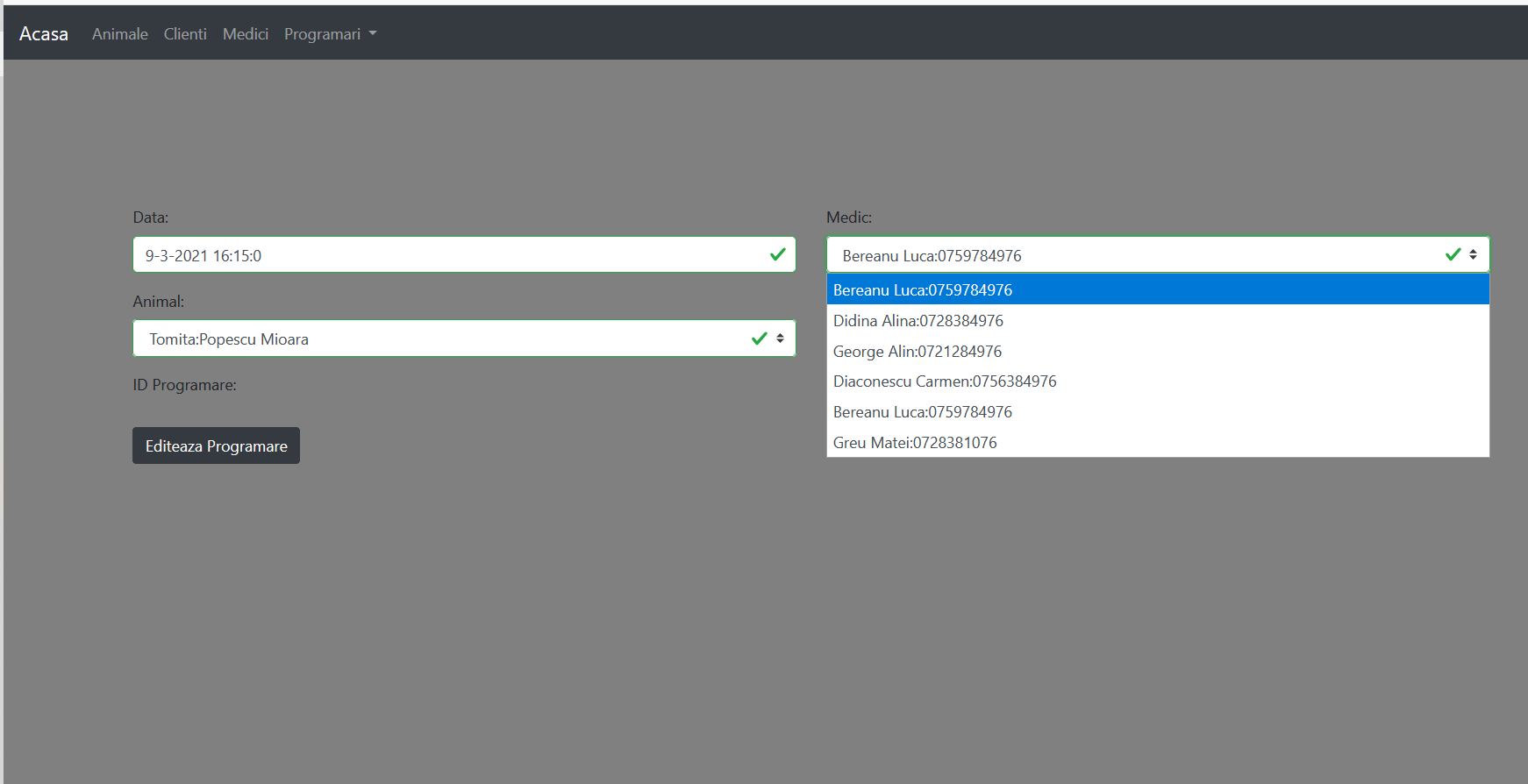
2)afisare

2)Exemplu de pagini pentru adaugare/modificare :

-adaugare client

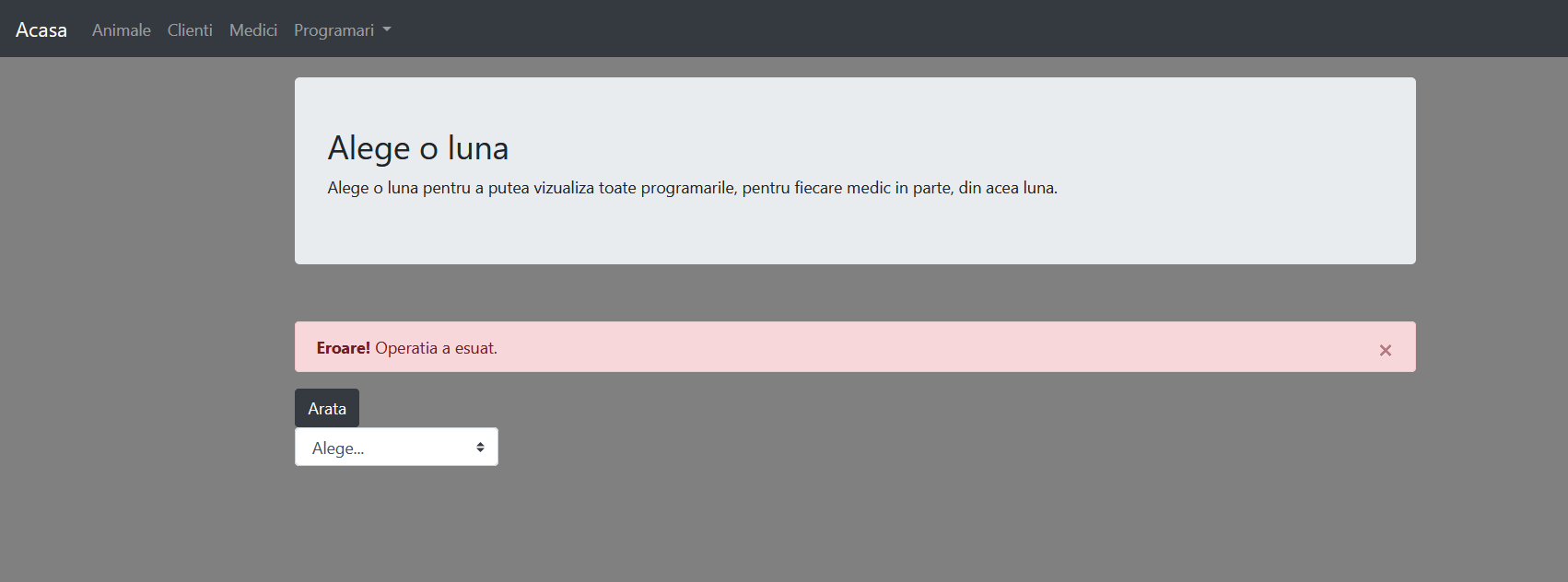


-modificare programare: ( a se observa faptul ca userul alege tuple pentru FK, nu valori de id)



3)Exemple de validare/invalidare a operatiei :

editare de programare invalida:



editare de date client valida:



***Nota:* mesajele de validare/invalidare apar dupa orice stergere/inserare/modifcare in tabela client sau programare.**