CIURTIN ANDREI-LAURENTIU

432A

**Documentatie a proiectului la Programarea Interfetelor**

**pentru Baze de Date**

**- Tehnologia Python + Django–**

**Cerință**

Creați două aplicații care să conțină o bază de date creată în sistemul de gestiune al

bazelor de date MySQL și două interfețe pentru aceasta (baza de date este comună). La

crearea interfețelor se vor folosi două tehnologii la alegere (JSP, Hibernate, JPA, .NET, Python+Django, Python+Flask etc.)

Baza de date a fost creata in tehnologia: **PostgreSQL**

“**PostgreSQL** este un [sistem de baze de date relationale](https://ro.wikipedia.org/wiki/SGBD" \o "SGBD). Este disponibil gratuit sub o licentă [open source](https://ro.wikipedia.org/wiki/Open_source) de tip BSD. PostgreSQL nu este controlat de nici o companie, își bazează dezvoltarea pe o comunitate răspândită la nivel global, precum și câteva companii dezvoltatoare.” [1]

Am realizat baza de date in pgAdmin 4. Ea contine 3 tabele: **avocat**, **client** si **contract**. Asocierea dintre primele 2 tabele este de tipul M:N, iar tabelul **contract** este tabelul de legatura dintre acestea.

Tabela **avocat** contine urmatoarele coloane:

* id\_avocat, de tip BIGINT , cheie primara
* nume, de tip CHARACTER VARYING (64)
* prenume, de tip CHARACTER VARYING (64)
* emai , de tip CHARACTER VARYING (64)
* telefon, de tip CHARACTER VARYING (64)
* cnp, de tip CHARACTER VARYING (64)
* specializare, de tip CHARACTER VARYING (64)
* speta, de tip CHARACTER VARYING (64)

Tabela **client** contine urmatoarele coloane:

* id\_avocat, de tip BIGINT , cheie primara
* nume, de tip CHARACTER VARYING (64)
* prenume, de tip CHARACTER VARYING (64)
* emai , de tip CHARACTER VARYING (64)
* telefon, de tip CHARACTER VARYING (64)
* cnp, de tip CHARACTER VARYING (64)
* adresa, de tip CHARACTER VARYING (64)

Tabela intermediara **contract** contine urmatoarele coloane :

* id\_contract, de tip BIGINT, cheie primara
* id\_avocat\_ctr, de tip BIGINT
* id\_client\_ctr, de tip BIGINT
* valoare, de tip NUMERIC (numar cu semn si cu precizie de 2 zecimale)
* data\_start, de tip DATE
* data\_sfarsit, de tip DATE
* judecatorie, de tip CHARACTER VARYING (64)

Datorita asocierii M:N, id\_avocat\_ctr si id\_client\_ctr sunt chei straine (FK) pentru tabela **contract**.

Toate cheile primare au proprietatile: Not Null (NN) si Auto-Incremet (AI)

“O cheie primară este un câmp sau un set de câmpuri cu valori unice într-un tabel. Valorile cheii se pot utiliza pentru a face referire la înregistrări întregi, pentru că fiecare înregistrare are o valoare diferită pentru cheie. Fiecare tabel poate avea doar o cheie primară.” [2]

Diagrama asociata tabelelor este reprezentata in figura de mai jos:

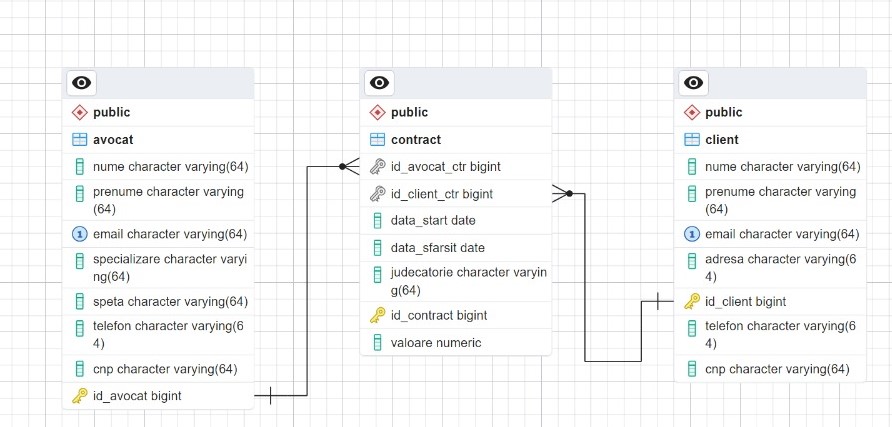


Fig. 1: diagrama asociata tabelelor

**Tehnologii utilizate**

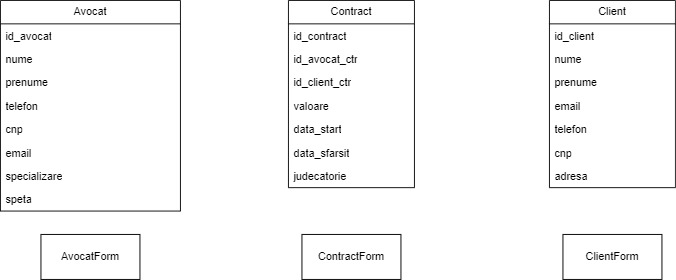
Pentru prima interfata am folosit NodeJS si pentru a doua interfata am folosit Python + Django.

“**Django** (/[ˈdʒæŋɡoʊ](https://ro.wikipedia.org/wiki/Alfabetul_Fonetic_Interna%C8%9Bional)/ ***[jang](https://ro.wikipedia.org/wiki/Wikipedia:Pronun%C8%9Bie" \o "Wikipedia:Pronunție)****[-goh](https://ro.wikipedia.org/wiki/Wikipedia:Pronun%C8%9Bie" \o "Wikipedia:Pronunție)*) este un [soft cadru pentru dezvoltarea aplicațiilor web](https://ro.wikipedia.org/w/index.php?title=Soft_cadru_pentru_dezvoltarea_aplica%C8%9Biilor_web&action=edit&redlink=1) ([en.](https://ro.wikipedia.org/wiki/Limba_englez%C4%83" \o "Limba engleză)*web application framework*) gratuit și cu [sursă deschisă](https://ro.wikipedia.org/wiki/Surs%C4%83_deschis%C4%83" \o "Sursă deschisă), scris în [Python](https://ro.wikipedia.org/wiki/Python), care urmează modelul arhitectural [Model-View-Controller](https://ro.wikipedia.org/w/index.php?title=Model-View-Controller&action=edit&redlink=1).

Scopul principal al acestui soft cadru pentru dezvoltarea aplicațiilor web este de a facilita crearea de website-uri complexe, fundate pe [baze de date](https://ro.wikipedia.org/wiki/Baze_de_date" \o "Baze de date). Django pune accent pe reutilizarea codului, pe modularitate, dezvoltare rapidă a site-urilor web, ghidându-se după principiul "nu te repeta" ([en.](https://ro.wikipedia.org/wiki/Limba_englez%C4%83" \o "Limba engleză)*Don't repeat yourself - DRY*). Django este codat de la un capăt la altul în [Python](https://ro.wikipedia.org/wiki/Python), chiar și fișierele de configurare și modelele de date sunt implementate în acest limbaj de programare. Django oferă și un panou administrativ, care, deși vine preinstalat, este opțional, prin intermediul acestuia se pot crea, citi, actualiza și șterge cu ușurință informații din baza de date. Acest panou de adminstrare este generat dinamic prin introspecție (prin analizarea tabelelor din baza de date) și poate fi ușor configurat prin modelele administrative de date.” [3]

Aceasta tehnologie permite crearea de interfete pentru pagini web intr-un mod intuitiv si accesibil incepatorilor.

Diagrama UML pentru interfata Django:

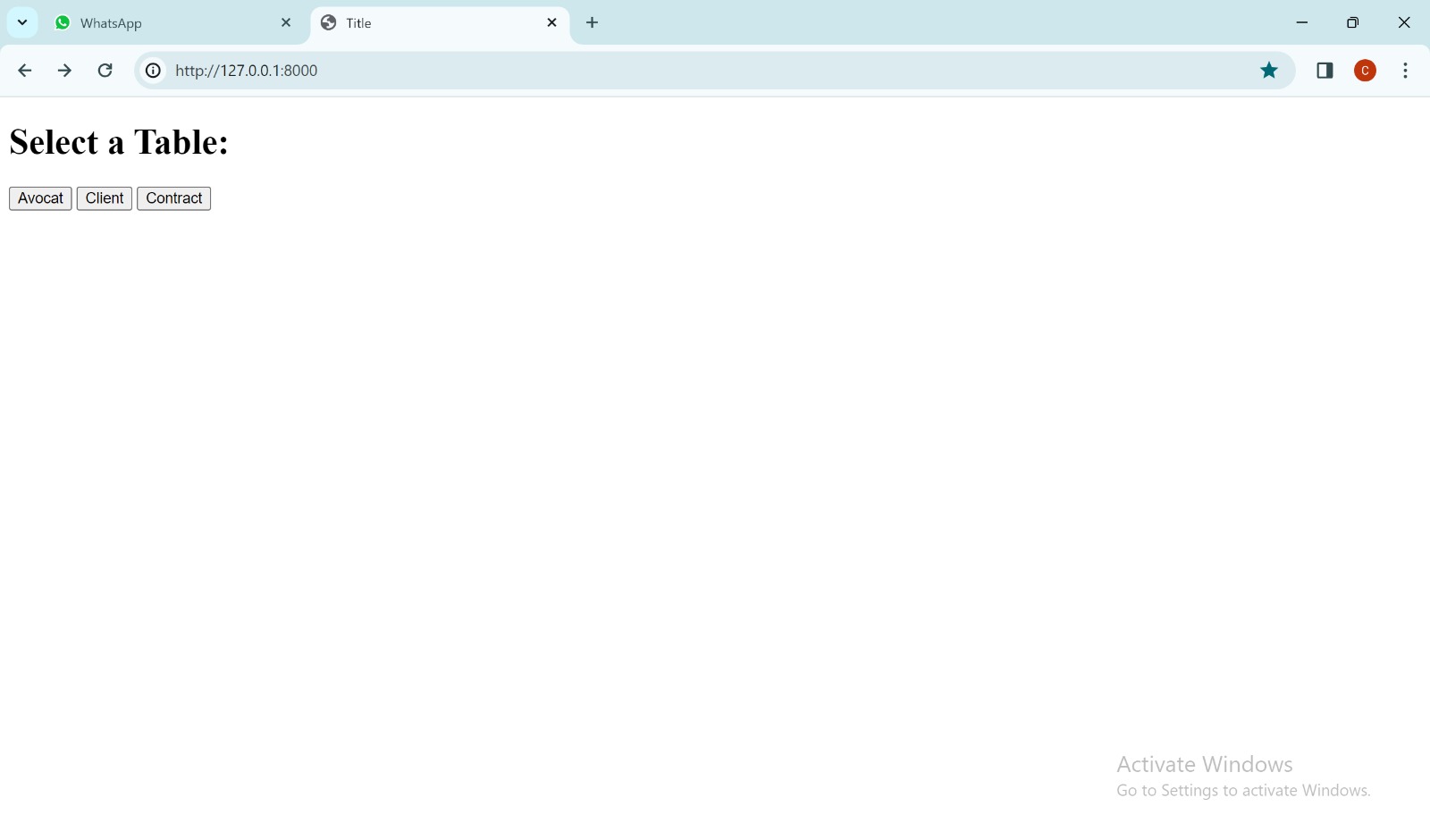


Se observa ca diagrama contine exclusiv clase, fara metode. Clasele poarta nume de modele (localizate in “models.py”), si nu contin metode. Acestea contin doar campurile tabelei, si au rolul de a abstractiza partea de SQL a lucrului cu bazele de date. Prin folosirea modelelor, nu mai este necesara scrierea manuala de cod SQL, deoarece Django o face in locul utilizatorului. Totusi, Django incearca sa creeze singur tabele, si nume de chei primare etc.., asa ca utilizatorul trebuie sa specifice daca se doreste conectarea unei tabele deja existente in baza de date, cat si ce nume au coloanele.

Clasele de Form se afla in fisierul “forms.py” si se conecteaza la modelele noastre pentru a putea genera form-uri.

Pagina de baza a interfetei (index.html) facuta in html serveste drept introducere in interfata noastra. Aceasta contine 3 butoane pentru selectarea tabelului pe care vrem sa-l vizualizam.

Fig. 2: pagina Index



Functionalitatea paginilor este similara, deci vom lua ca exemplu pagina de vizualizare a tabelului **contract**, cu toate functiile ei:

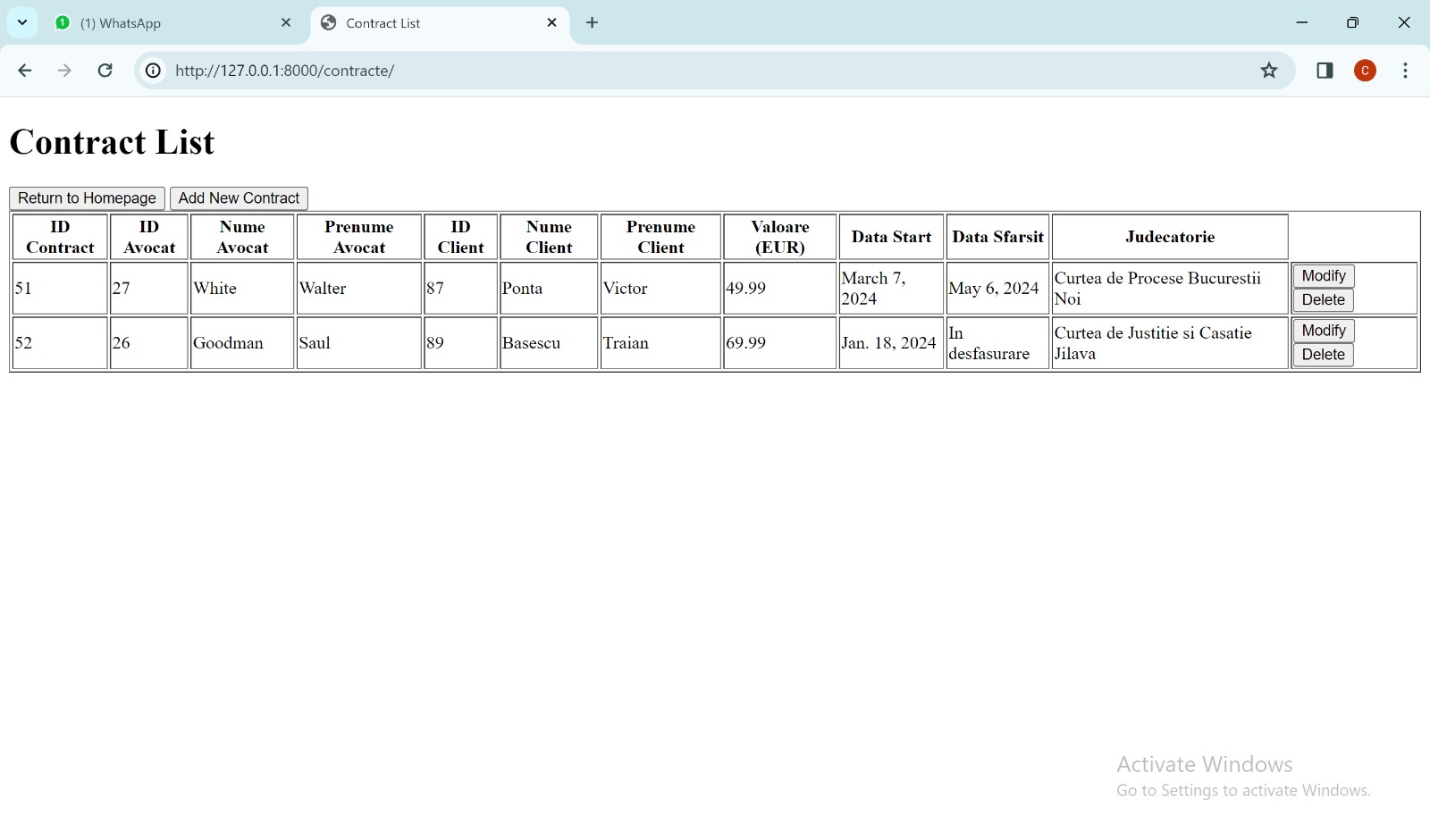


Fig.3: vizualizarea tabelei “contract”

Diferenta dintre pagina de vizualizare a tabelei “contract” si celelalte pagini de “view” este ca tabela “contract” contine si date din celelalte doua tabele, si anume: id, nume si prenume. Toate ferestrele de vizualizare contin butoane pentru adaugare, editare si stergere de date, dar si un buton pentru intoarcerea in fereastra “index” in caz ca utilizatorul doreste sa deschida pagina altui tabel. Functionalitatea fiecarei pagini in parte este definita atat in “views.py”, cat si in pagina html atribuita functiilor interfetei.

Codul urmator corespunde functiei de adaugare de inregistrari pentru tabela “contract”:

from .models import Contract

def contract\_list(request):

contracte = Contract.objects.select\_related('id\_avocat\_ctr', 'id\_client\_ctr').all().order\_by('id\_contract')

return render(request, 'main\_app/contract\_list.html', {'contracte': contracte})

Este important faptul ca toate modelele create de noi trebuie importate in “views.py”

Aceasta functie Python se foloseste de modelul Contract pentru a primi datele din aceasta tabela, ordonandu-le dupa cheia primara. Ultima linie de cod indica pagina web si url-ul pe care se aplica acest view. Paginile se afla in folderul dedicat “templates”, si sunt de tip html. Url-urile sunt in file-ul lor dedicat “urls.py”, si trebuie de asemenea importate in acesta si toate view-urile definite de noi.

from .views import contract\_list, add\_contract, modify\_contract, delete\_contract

path('contracte/', contract\_list, name='contract\_list'),

path('add\_contract/', add\_contract, name='add\_contract'),

path('modify\_contract/<int:contract\_id>/', modify\_contract, name='modify\_contract'),

path('delete\_contract/<int:contract\_id>/', delete\_contract, name='delete\_contract'),

Acest cod contine toate import-urile si caile definite pentru tabela “contract”. Similar se procedeaza si in cazul celorlalte tabele.

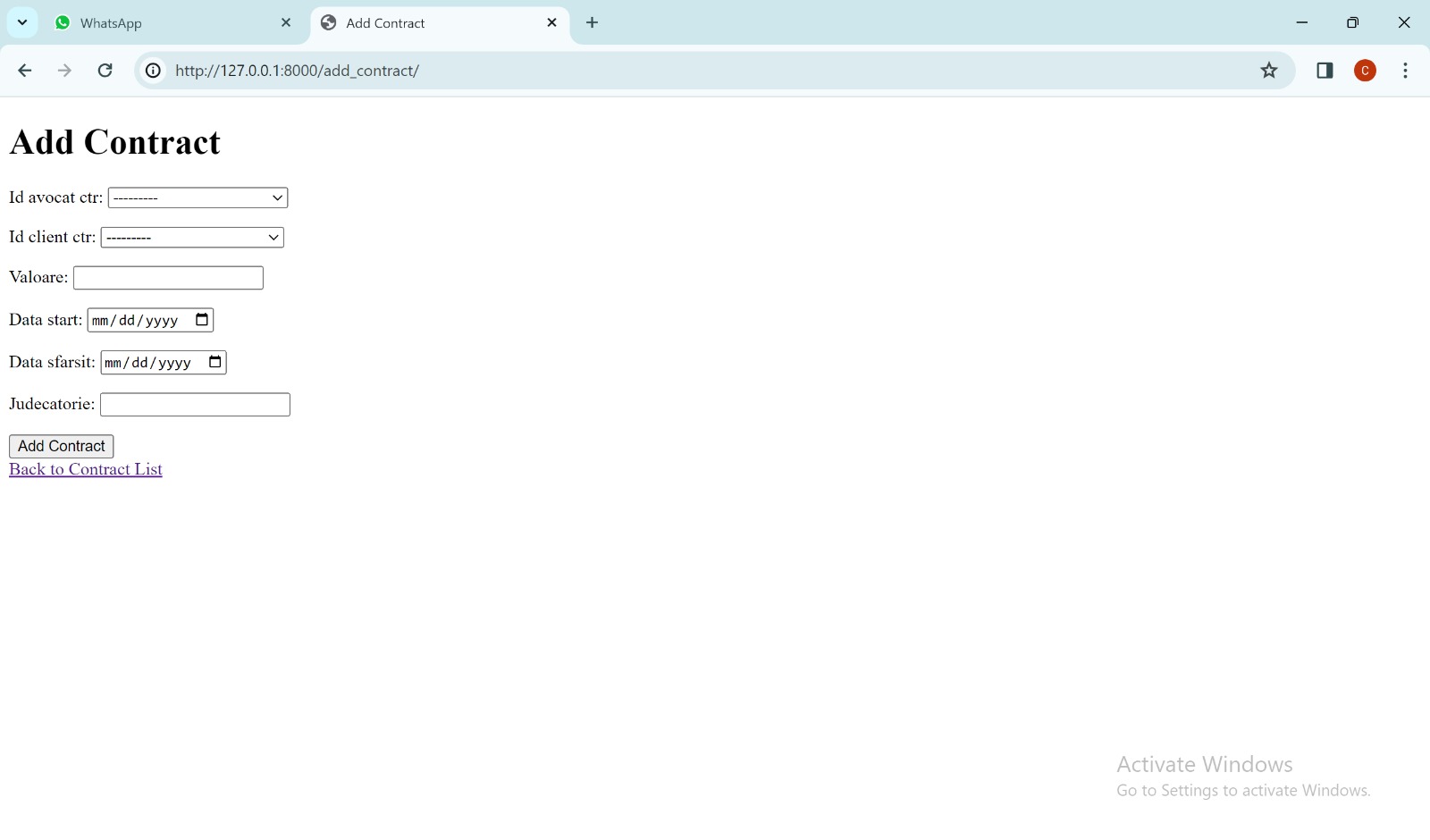


Fig.4: adaugarea unui contract nou

In figura de mai sus este prezentata fereastra de adaugare a unei noi inregistrari. Aceasta fereastra contine un form ce permite utilizatorului sa treaca date pentru un nou contract. Pagina de adaugare a acestei tabele este similara cu celelalte doua, exceptand faptul ca tipurile de date din aceasta tabela sunt ceva mai numeroase. Se poate observa ca datele ce corespund celorlalte tabele pot fi alese dintr-o lista, in functie de ID, nume si prenume, deoarece contractul leaga doua inregistrari de tip client si avocat. Campul “Valoare” este de tipul NUMERIC, si se pot alege numere cu semn si cu precizie de doua zecimale. Printr-un click pe campul pentru data de inceput si data de sfarsit se deschide un meniu de calendar, ce permite utilizatorului sa adauge aceste informatii cu usurinta.

De asemenea, se regasesc si doua butoane: “Back to Contract List”, care permite utilizatorului sa abandoneze operatia curenta si sa se intoarca in fereastra de “view”, si “Add Contract”, pentru a se crea noua inregistrare pe baza datelor introduse in form. Codul din “views.py” aferent acestei pagini se regaseste mai jos:

def add\_contract(request):

if request.method == 'POST':

form = ContractForm(request.POST)

if form.is\_valid():

form.save()

return redirect('contract\_list') # Redirect to contract list view after adding

else:

form = ContractForm()

return render(request, 'main\_app/new\_contract.html', {'form': form})

Este folosita o cerere de “post” care comunica serverului sa accepte datele tabelei si sa le stocheze la o anumita destinatie (in cazul nostru, in baza de date). Comanda SQL: “INSERT INTO…” este abstractizata de catre Django, dar se executa in fundal, facand posibila functia de adaugare a unui contract.

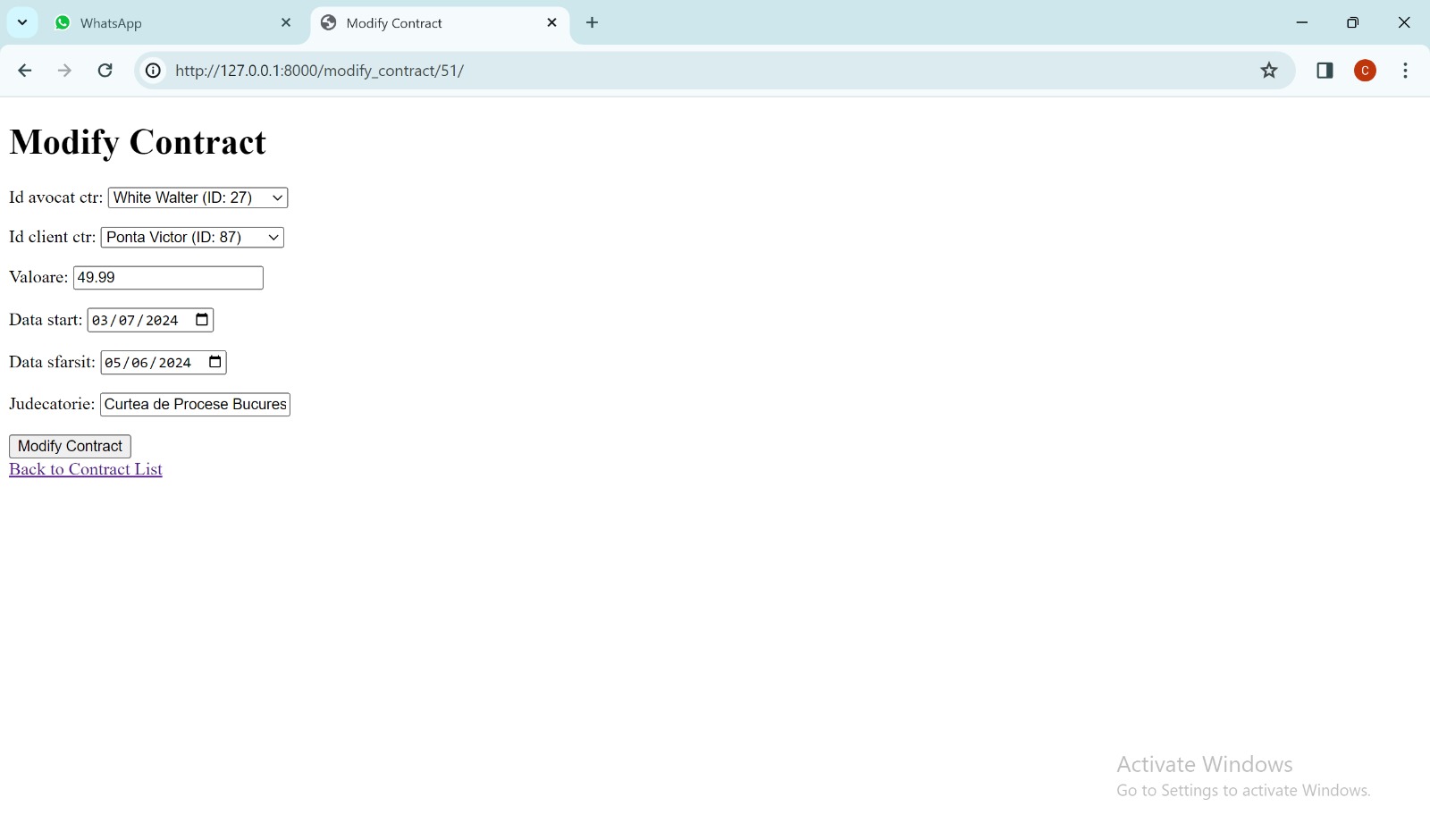


Fig.5: editarea unui contract deja existent

In figura de mai sus este prezentata pagina de editare a unei inregistrari din tabela “contract”. Functionalitatea este similara cu pagina de adaugare, cu exceptia faptului ca se utilizeaza functia get\_object\_or\_404(), care afiseaza datele in form. Fiecarei inregistrari din tabel ii este asociat un buton de “edit” (fig. 3), ceea ce face mai simplu procesul de selectare a inregistrarii pe care dorim sa o modificam.

View-ul asociat functiei de modificare a unei inregistrari se afla mai jos:

from django.shortcuts import get\_object\_or\_404

def modify\_contract(request, contract\_id):

contract = get\_object\_or\_404(Contract, id\_contract=contract\_id)

if request.method == 'POST':

form = ContractForm(request.POST, instance=contract)

if form.is\_valid():

form.save()

return redirect('contract\_list') # Redirect to contract list view after modification

else:

form = ContractForm(instance=contract)

return render(request, 'main\_app/modify\_contract.html', {'form': form})

O ultima functie a acestei interfete este stergerea datelor. Fiecarei inregistrari ii este asociat un buton de “delete” (fig. 3), care sterge instantaneu inregistrarea respectiva. Codul din spatele acestei functii este urmatorul:

def delete\_contract(request, contract\_id):

contract = Contract.objects.get(id\_contract=contract\_id)

contract.delete()

return redirect('contract\_list')

Functia de delete nu foloseste explicit o cerere de “post”. Este necesara doar functia “.delete()”. Apasarea butonului de “delete” aferent fiecarei inregistrari determina eliminarea acesteia din baza de date fara redirectionarea spre o fereastra noua. Astfel, stergerea de date este foarte simpla si accesibila, fara a mai fi nevoie de o pagina aditionala de confirmare.

**Concluzii**

In concluzie, Python + Django este o tehnologie destinate sa reduca workflow-ul si sa transforme creatia unei interfete intr-o activitate simpla si intuitiva. Lucrul cu bazele de date este simplificat prin abstractizarea script-urilor SQL, iar partea de server si de conectare a bazei de date este usor accesibila oricarui utilizator.

Folosind interfata creata, o firma de avocatura isi poate gestiona foarte usor afacerile, adaugand avocati, clienti, cat si contracte intre acestia. Nu este nevoie de expertiza in SQL pentru a opera aceasta interfata, data fiind simplitatea acesteia.

Nu numai tehnologia folosita pentru implementarea interfetei conteaza. Este foarte importanta si aplicatia in care este construita baza de date. PostgreSQL s-a dovedit a fi o alegere buna in realizarea acestui proiect, datorita interfetei prietenoase si a multitudinii de unelte pentru gestionarea tabelelor si a constrangerilor dintre entitatile acestora.

Bibliografie:

[1] <https://ro.wikipedia.org/wiki/PostgreSQL>

[2] <https://support.microsoft.com/ro-ro/topic/ad%C4%83ugarea-sau-modificarea-cheii-primare-a-unui-tabel-%C3%AEn-access-07b4a84b-0063-4d56-8b00-65f2975e4379#:~:text=O%20cheie%20primar%C4%83%20este%20un,avea%20doar%20o%20cheie%20primar%C4%83>.

[3]https://ro.wikipedia.org/wiki/Django\_(web\_framework)#:~:text=Django%20(%2F%CB%88d%CA%92%C3%A6%C5%8B,arhitectural%20Model%2DView%2DController.

Alte surse de inspiratie:

<https://www.youtube.com/> pentru numeroase tutorial de Python + Django si PostgreSQL

<https://stackoverflow.com/> pentru erorile de cod intampinate pe parcurs