CIURTIN ANDREI-LAURENTIU

432A

**Documentatie a proiectului la Programarea Interfetelor**

**pentru Baze de Date**

**- Tehnologia Node.js–**

**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:

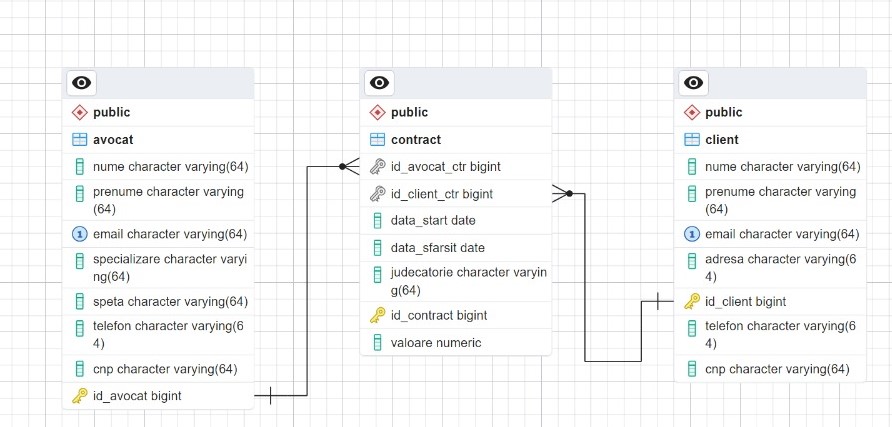


Fig. 1: diagrama asociata tabelelor

**Tehnologii utilizate**

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

**“Node.js** este un [mediu de execuție](https://ro.wikipedia.org/w/index.php?title=Sistem_de_execu%C8%9Bie&action=edit&redlink=1" \o "Sistem de execuție — pagină inexistentă) [JavaScript](https://ro.wikipedia.org/wiki/JavaScript) [de fundal](https://ro.wikipedia.org/wiki/Front-end_%C8%99i_back-end) [multiplatformă](https://ro.wikipedia.org/wiki/Multiplatform%C4%83" \o "Multiplatformă) [cu sursă deschisă](https://ro.wikipedia.org/wiki/Software_cu_surs%C4%83_deschis%C4%83), care rulează pe [motorul V8](https://ro.wikipedia.org/w/index.php?title=V8_(motor_JavaScript)&action=edit&redlink=1" \o "V8 (motor JavaScript) — pagină inexistentă)și execută cod JavaScript în afara unui [navigator web](https://ro.wikipedia.org/wiki/Navigator_web).” [3]

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

Aplicatia nu foloseste un ORM (Object Relational Mapper), deci nu se poate realiza o diagrama UML in absenta claselor si modelelor.

Pagina de baza a interfetei (home.ejs) facuta in embedded javascript serveste drept introducere in interfata noastra. Ea contine titlul firmei de avocatura si un buton “View Tables” care ne redirectioneaza spre fereastra in care alegem ce tabele vrem sa vizualizam (view.ejs)

Fig. 2: pagina principala

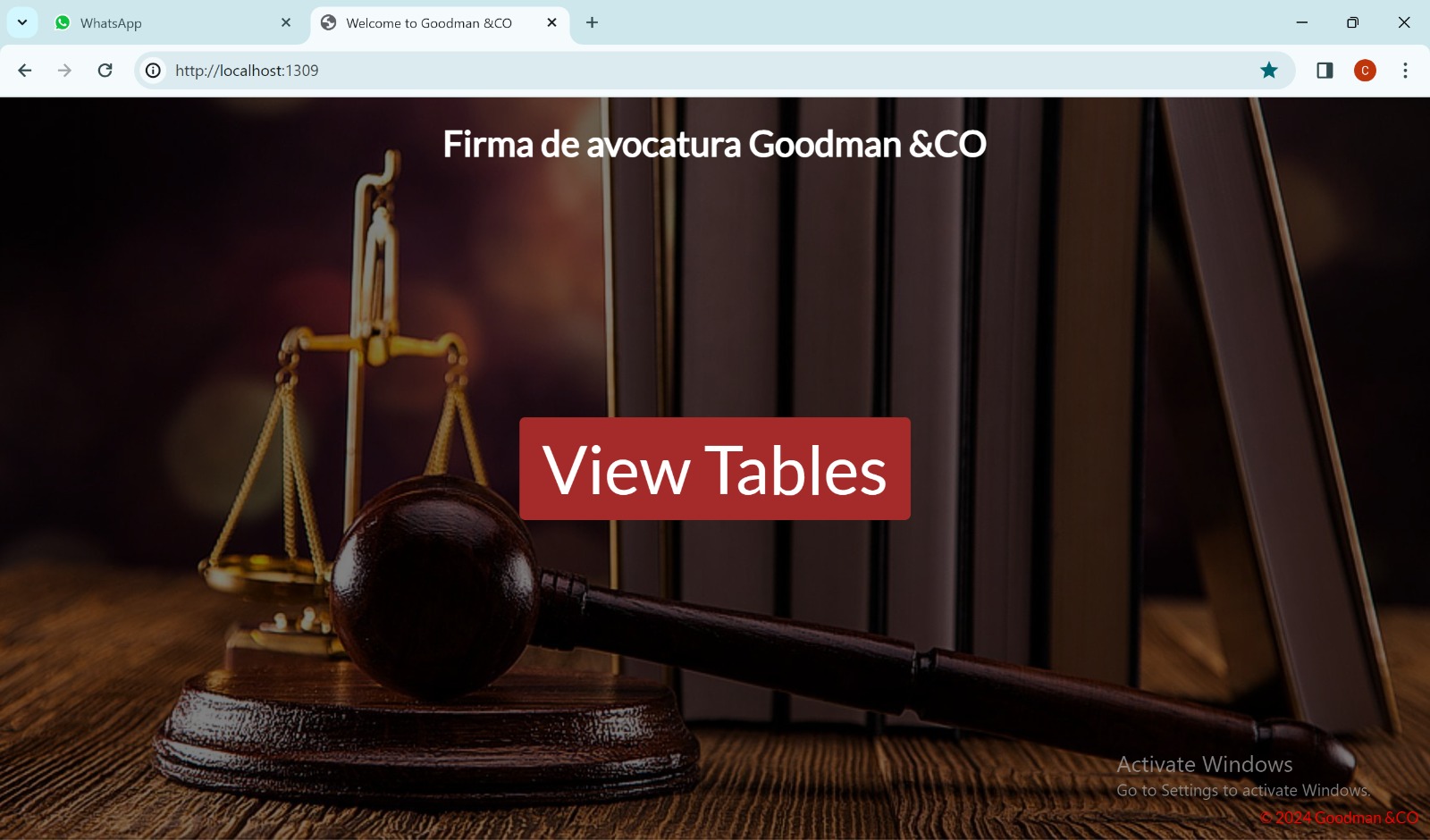
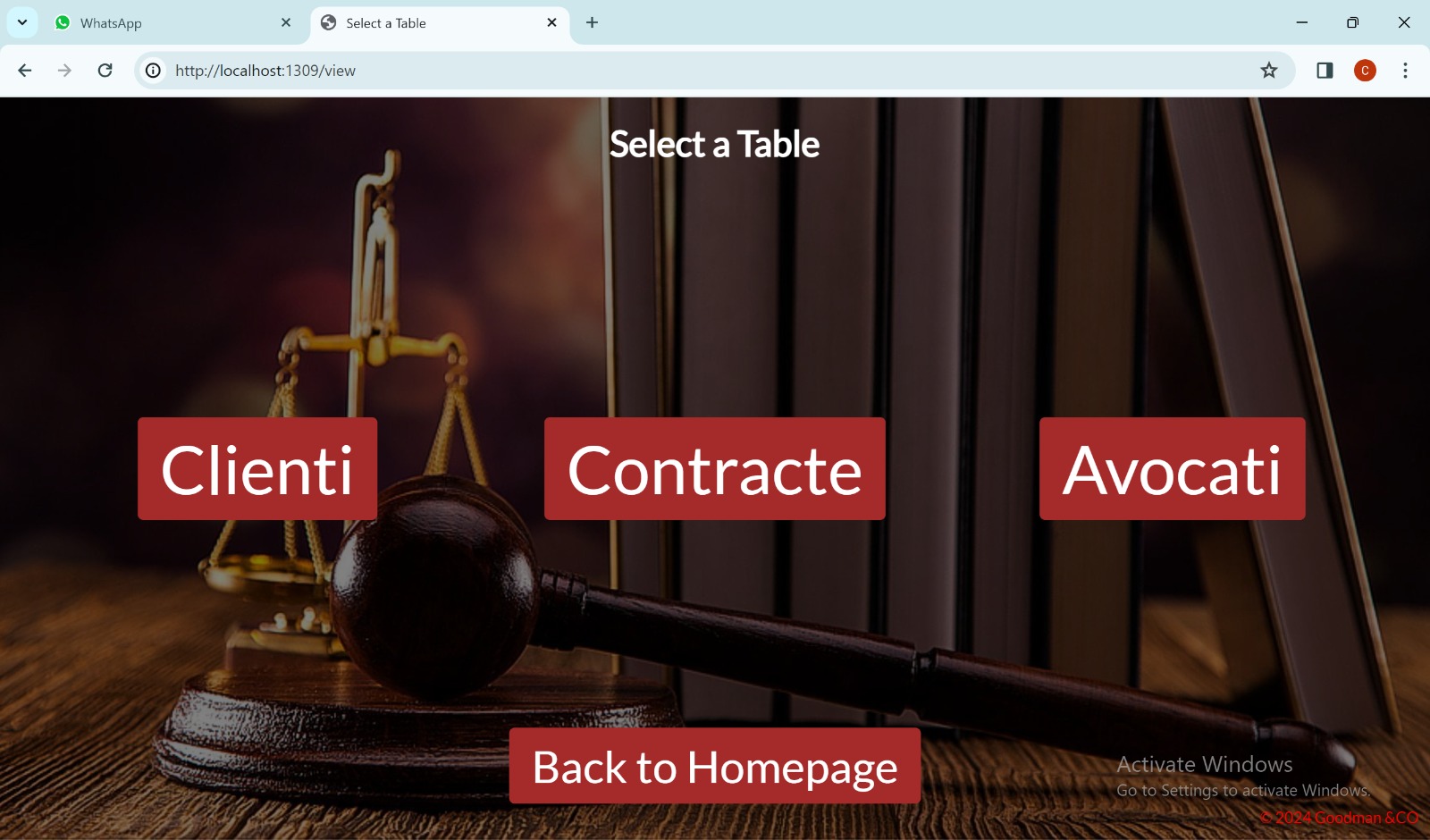


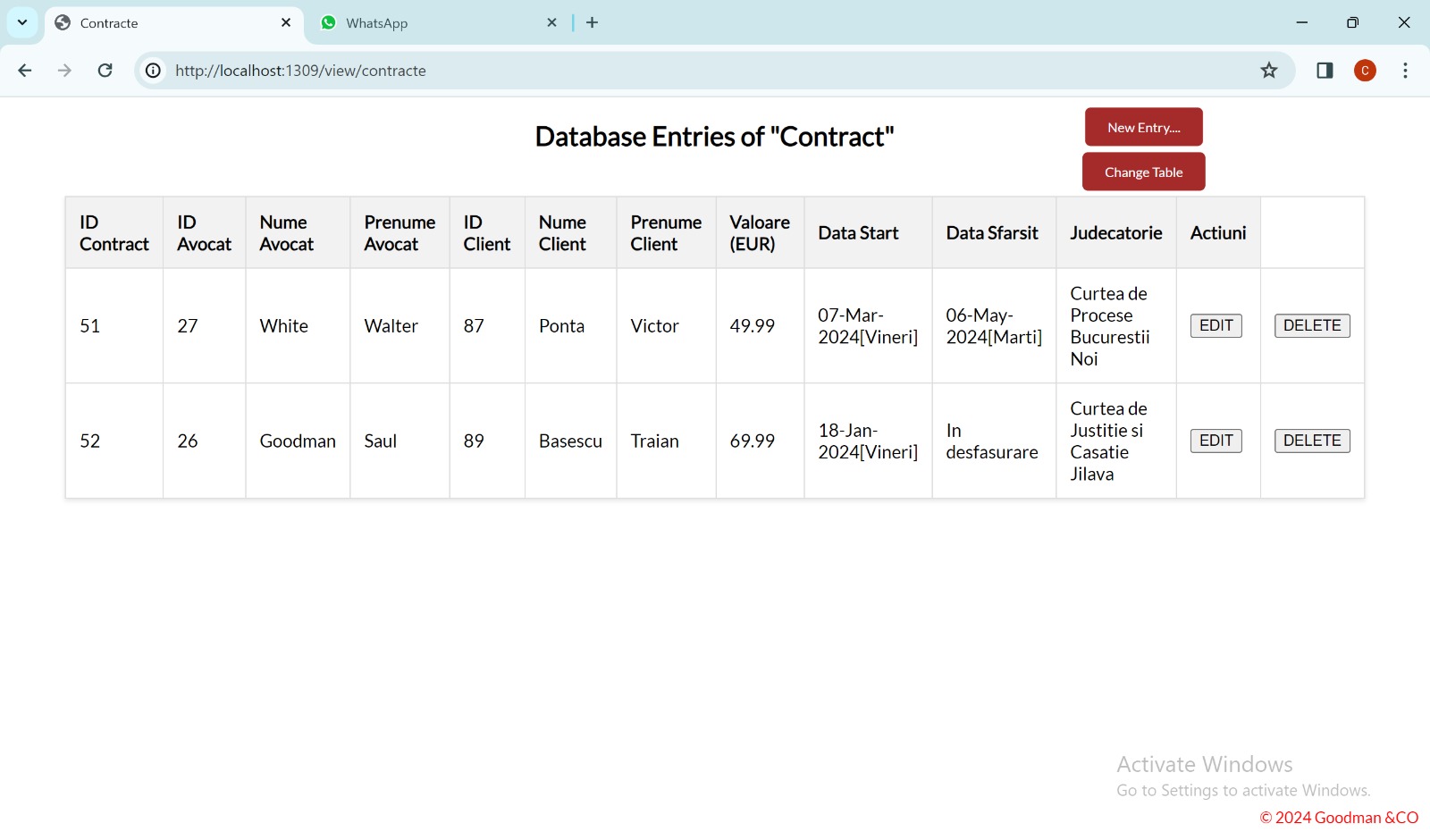
Fig. 3: pagina “view”



In pagina de mai sus se afla 4 butoane: butonul de intoarcere in pagina principala, si 3 butoane pentru selectarea tabelului pe care vrem sa-l vizualizam.

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

Fig. 4: 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 “view” in caz ca utilizatorul doreste sa deschida pagina altui tabel. Aceasta pagina foloseste un “get” request, ce ne permite sa preluam date de la o sursa (in acest caz, baza de date) si sa le trimitem paginii respective. Codul asociat acestei pagini se afla in “server.js”:

app.get("/view/contracte", function (req, res) {

client.query(

`SELECT

c.id\_contract, c.id\_avocat\_ctr, a.nume AS nume\_avocat, a.prenume AS prenume\_avocat,

c.id\_client\_ctr, cl.nume AS nume\_client, cl.prenume AS prenume\_client,

c.valoare, c.data\_start, c.data\_sfarsit, c.judecatorie

FROM contract c

JOIN avocat a ON c.id\_avocat\_ctr = a.id\_avocat

JOIN client cl ON c.id\_client\_ctr = cl.id\_client

ORDER BY c.id\_contract`,

function (queryErr, queryRes) {

if (queryErr) {

res.render("pages/eroare", { eroare: String(queryErr) });

} else {

res.render("pages/contracte", {

pageTitle: 'Contracte',

entries: queryRes.rows,

nr\_rez: queryRes.rowCount

});

}

}

);

});

Acest cod se foloseste de query-uri SQL pentru a prelua datele necesare din tabelul “contract”. Functia JOIN ofera posibilitatea de a obtine date si din celelalte doua tabele. Toate datele sunt ordonate dupa ID.

Daca apar erori, se va deschide pagina de eroare. In caz contrar, se va deschide pagina de vizualizare prin intermediul functiei res.render(…)

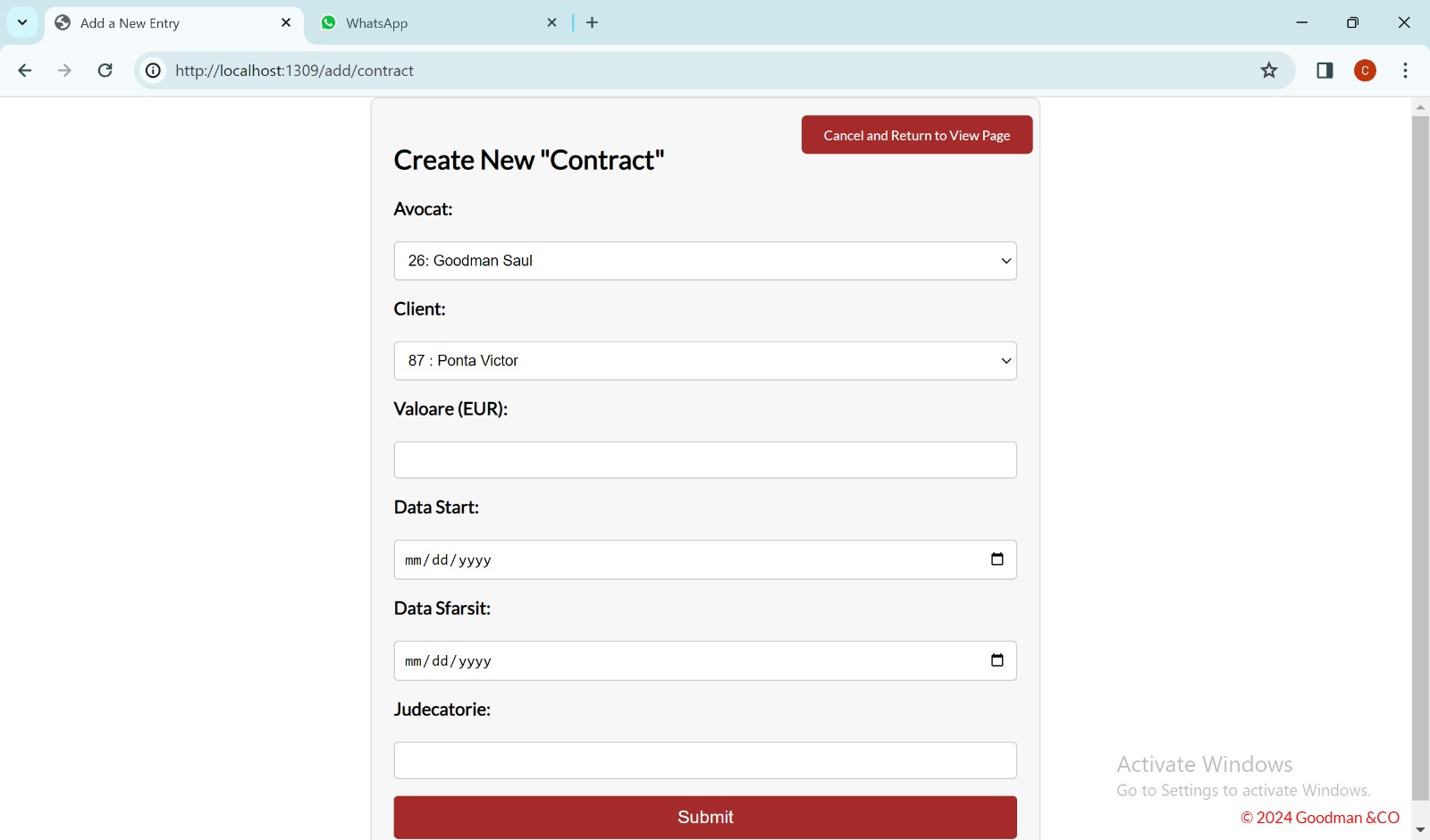


Fig. 5: adaugarea unui contract nou

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 (EUR)” 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: “Cancel”, care permite utilizatorului sa abandoneze operatia curenta si sa se intoarca in fereastra de “view”, si “Submit”, pentru a se crea noua inregistrare pe baza datelor introduse in form. Codul din server.js aferent acestei pagini se regaseste mai jos:

app.get('/add/contract', async (req, res) => {

try {

const avocatList = await client.query('SELECT id\_avocat, nume, prenume FROM avocat');

const clientList = await client.query('SELECT id\_client, nume, prenume FROM client');

if (avocatList.rows.length === 0 || clientList.rows.length === 0) {

res.render('pages/eroare', {

eroare: 'There must be at least one "client" and one "avocat" to create a contract.'

});

} else {

res.render('pages/add\_contract', {

pageTitle: 'Add a New Entry',

avocatList: avocatList.rows,

clientList: clientList.rows

});

}

} catch (error) {

res.render("pages/eroare", { eroare: String(error) });

}

});

app.post('/create/contract', async (req, res) => {

const form = new formidable.IncomingForm({ multiples: false });

form.parse(req, async (err, fields, files) => {

if (err) {

//console.error(err);

res.render("pages/eroare", {eroare:String(err)});

return;

}

const {

id\_avocat\_ctr,

id\_client\_ctr,

valoare,

data\_start,

data\_sfarsit,

judecatorie

} = fields;

const query = `

INSERT INTO contract (id\_avocat\_ctr, id\_client\_ctr, valoare, data\_start, data\_sfarsit, judecatorie)

VALUES ($1, $2, $3, $4, $5, $6)

`;

values = [id\_avocat\_ctr, id\_client\_ctr, valoare, data\_start, data\_sfarsit, judecatorie];

values.forEach((element, index) => {

const element\_clean = String(element).replace(/[{}"]/g, '');

values[index] = element\_clean;

if(index == 4 && element\_clean == ''){

values[index] = null;

}

});

try {

await client.query(query, values);

res.redirect('/view/contracte');

} catch (error) {

console.error(error);

//res.status(500).send('Eroare.');

res.render("pages/eroare", {eroare:String(error)});

}

});

});

In afara unei cereri de “get”, este folosita si o cerere de “post” care comunica serverului sa accepte datele incluse in mesajul cererii si sa le stocheze la o anumita destinatie (in cazul nostru, in baza de date). Se vor respecta constrangerile setate in fisierul “add\_contract.ejs”, si daca nu apar erori, datele vor fi introduse prin intermediul unui query de “INSERT INTO” si se va realiza intoarcerea in pagina de vizualizare a tabelei respective.

Fig. 5: adaugarea unui contract nou

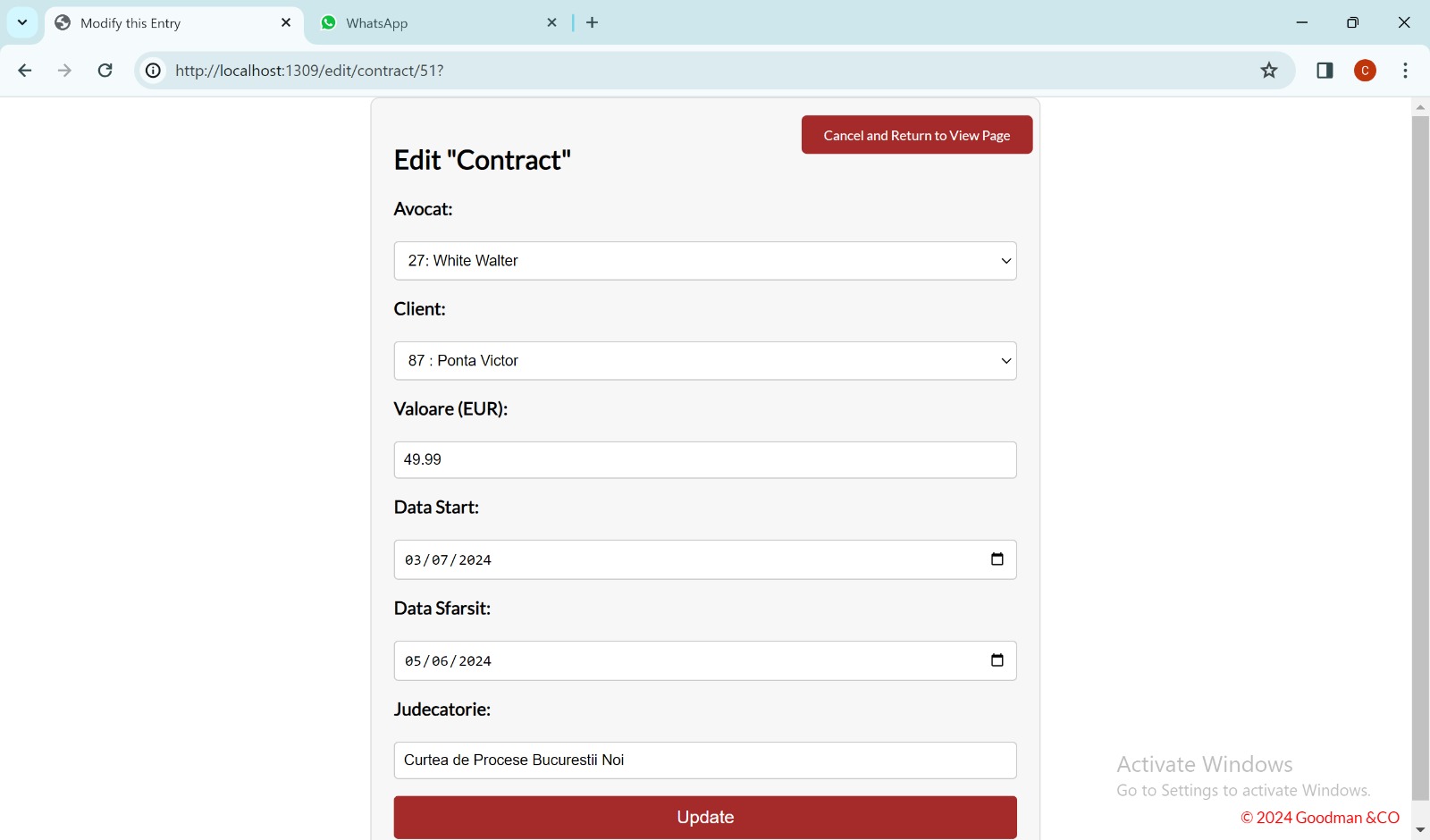


Fig. 6: editarea unui contract deja existent

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 in cererea de “get” este inclus un query pentru preluarea datelor si stocarea acestora in formular. De asemenea, este inclus un query de “UPDATE” in loc de “INSERT INTO”. Fiecarei inregistrari din tabel ii este asociat un buton de “edit” (fig. 4), ceea ce face mai simplu procesul de selectare a inregistrarii pe care dorim sa o modificam.

Codul aferent query-ului de UPDATE folosit:

UPDATE contract

SET id\_avocat\_ctr = $1,

id\_client\_ctr = $2,

valoare = $3,

data\_start = $4,

data\_sfarsit = $5,

judecatorie = $6

WHERE id\_contract = $7;

O ultima functie a acestei interfete este stergerea datelor. Fiecarei inregistrari ii este asociat un buton de “delete” (fig. 4), care sterge instantaneu inregistrarea respectiva. Codul din spatele acestei functii este urmatorul:

app.post("/delete/contract/:id", function(req, res){

const contractId = req.params.id;

client.query('DELETE FROM contract WHERE id\_contract = $1', [contractId], function(err, rez){

if(err){

//console.error(err);

res.render("pages/eroare", {eroare:String(err)});

return;

}

console.log('Delete successful');

res.redirect("/view/contracte");

});

});

Functia de delete foloseste doar o cerere de “post”. 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, Node.js este o tehnologie vasta, dar usoara de stapanit cu scopul de a creea numeroase servere si pagini web. Lucrul cu bazele de date este simplificat prin utilizarea functiilor de query, si codul pentru initializarea serverului si conectarea bazei de date este scurt si usor de inteles.

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://en.wikipedia.org/wiki/Node.js>

Alte surse de inspiratie:

<https://www.youtube.com/> pentru numeroase tutorial de Node.js si PostgreSQL

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