# Introducere

Prezenta lucrare propune **dezvoltarea unei aplicații menite să faciliteze sintetizarea conținutului contractelor notariale**, bazându-se pe **conceptul de procesare a limbajului natural** (NLP) pentru automatizarea procesului. Aplicația se adresează în principal notariatelor, permițând stocarea documentelor contractuale în format digital.

Ideea din spatele unei astfel de aplicații a pornit din dorința de a cerceta **lingvistica computațională în contextul limbii române**, iar alegerea domeniului juridic reflectă unul dintre obiectivele pe care le-am urmărit în elaborarea acestei lucrări, și anume dezvoltarea unui sistem informatic care să aducă **plusvaloare** într-o arie de aplicabilitate. În acest sens, am identificat nevoia de **digitalizare a activităților specifice domeniului juridic** și am încercat să fac un pas spre ceea ce ar facilita această transformare în România.

În fond, contractele sunt tot mai prezente și în viața cotidiană, deși importanța acestora este adesea trecută cu vederea. Aproape toate activitățile din viața de zi cu zi sunt reglementate de reguli ale legii contractuale, de la cumpărături online uzuale, până la tranzacții mobiliare sau imobiliare. În ciuda acestui fapt, abordarea unui contract nu este întotdeauna o sarcină ușoară.

O sintetizare a documentelor contractuale poate fi considerată necesară pentru economisirea timpului și creșterea productivității interne în ceea ce privește un notariat sau o firmă de avocatură. Util în cazul unor documente voluminoase și complexe, abstractizarea unui contract reprezintă o codificare legală prin care se realizează extragerea unor informații importante, precum data la care este încheiat contractul, părțile contractante și alte clauze majore (obiectul contractului, prețul de vânzare etc.).

Așadar, sistemul informatic implementat permite păstrarea documentelor întocmite în cadrul unui birou notarial prin intermediul unei platforme care facilitează accesarea oricărui contract încărcat de utilizator, oferind totodată o scurtă sinteză a conținutului. Deși păstrarea documentelor în format fizic rămâne necesară în contextul legal, aplicația propusă optimizează activitățile interne și poate fi folosită ca o bază de date pentru contractele încheiate de-a lungul timpului. Astfel, accesarea oricărui act se află la un *click* distanță de orice angajat al unui birou notarial. Pe lângă accesul facil la documente, ceea ce se urmărește în construirea aplicației este **eficientizarea timpului și a resurselor** utilizate prin **automatizarea unor procese repetitive**.

# Abordări preexistente

## 2.1. Abordarea domeniului ales

Așa cum am menționat anterior, scopul principal al aplicației pe care se bazează această lucrare este acela de a sintetiza conținutul unor contracte, extrăgând informațiile esențiale precum data la care este încheiat contractul, tipul contractului (vânzare-cumpărare, schimb mobiliar, schimb imobiliar sau donație), numele contractanților, obiectul contractului și, după caz, prețul contractului (prețul de vânzare).

Aplicația propusă are la bază o **gramatică fără context** care descrie limba română, în alte cuvinte un set de reguli recursive utilizate pentru a construi un model ce poate fi aplicat asupra unor documente specifice. Astfel, provocarea principală în abordarea acestui domeniu a constituit-o chiar **definirea gramaticii** care să poată fi aplicată în contextul mai multor tipuri de contracte. Acest lucru a implicat o cercetare privind algoritmii care permit construirea unei astfel de structuri, dar și regulile după care aceștia funcționează. În cele din urmă, am ales să implementez modelul cu ajutorului **algoritmului Earley**, care este **cel mai eficient algoritm** general destinat gramaticilor independente de context.

În ceea ce privește aplicabilitatea în domeniul juridic, mai exact în **procesarea documentelor contractuale**, consider că o astfel de aplicație prezintă o reală utilitate pentru orice notariat sau firmă de avocatură, având în vedere volumul mare de documente procesate în cadrul acestora. În fond, inteligența artificială și-a îndreptat atenția încă de la început asupra proceselor de business uzuale, iar gestionarea contractelor a devenit necesară în tot mai multe câmpuri de activitate. În ciuda faptului că aplicația propusă se adresează birourilor notariale, ideea poate fi implementată și pentru clienți care să își stocheze diferite contracte într-un singur loc, fie că este vorba de persoane fizice sau companii. Pentru început, platforma este dedicată lucrătorilor din domeniul juridic, respectiv notarilor, ajutând astfel la **digitalizarea proceselor** și, cel mai important, la **eficientizarea timpului alocat sarcinilor de lucru**. De multe ori, este nevoie de anumite informații din contracte încheiate în trecut, iar căutarea lor înseamnă parcurgerea documentului și analizarea numeroaselor pagini, proces ce poate fi unul îndelungat.

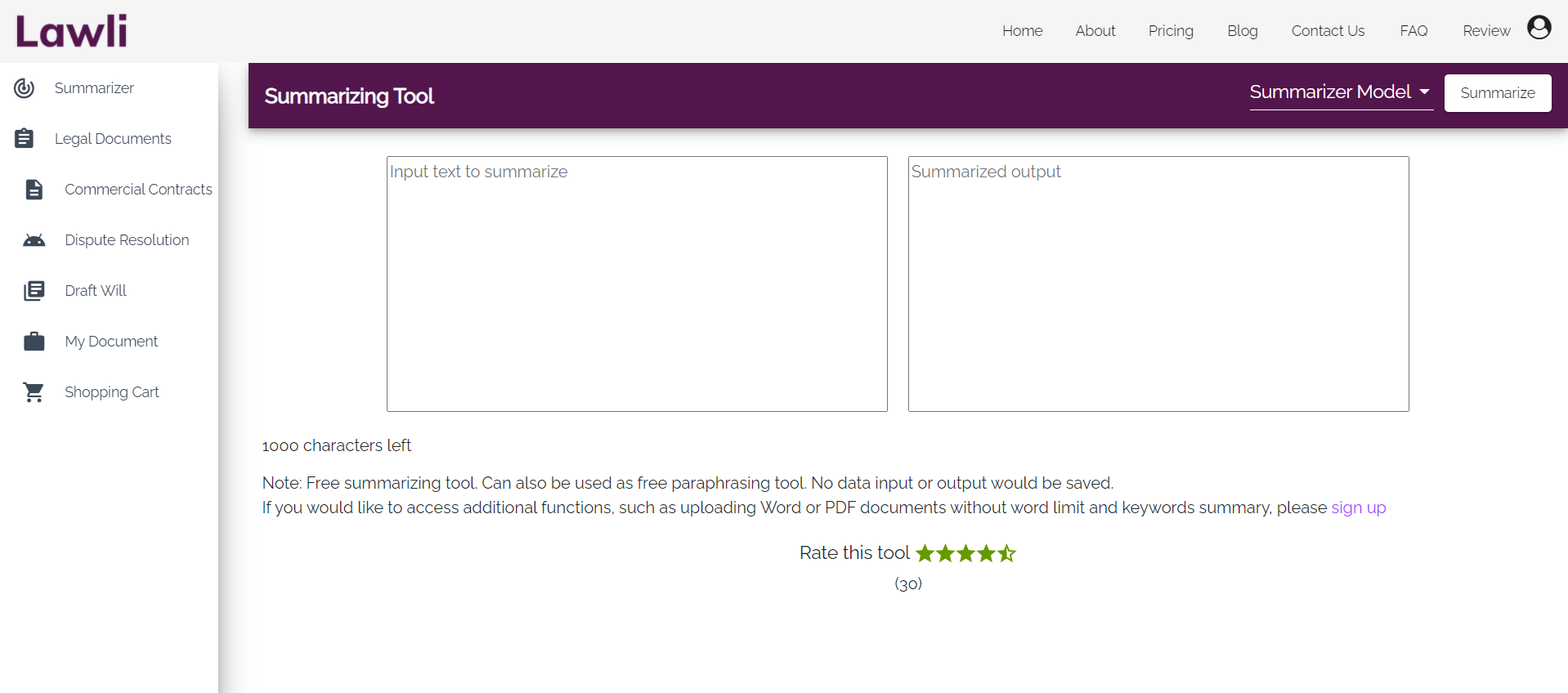
Așadar, aplicarea conceptelor de NLP asupra documentelor contractuale, așa cum voi detalia în următoarele capitole, urmărește o automatizare a sarcinilor uzuale, astfel optimizând procese costisitoare din punct de vedere al timpului.

## 2.2. Abordări similare

Mediul online pune la dispoziția utilizatorilor mai multe soluții pentru sintetizare documentelor, majoritatea fiind aplicabile asupra textelor redactate în limba engleză. Înainte de a analiza însă instrumentele existente pe piață, este important de menționat faptul că sintetizarea unui text poate fi abordată din 2 perspective. **Metoda extractivă** constă în identificarea unor secțiuni relevante ale textului (cuvinte cheie, fraze, propoziții) și generarea lor textuală prin producerea unor propoziții bazate pe textul inițial. **Metoda abstractivă** reproduce informațiile găsite într-un mod nou, generând astfel un nou text, mai succint, cuprinzând doar ideile principale ale textului original. În ciuda faptului că folosesc metode diferite de rezumare, fiecare dintre aplicațiile menționate în continuare urmăresc o analiză a textului pe baza Procesării Limbajului Natural, oferind soluții rapide pentru sintetizarea unor texte contractuale voluminoase.

**1. Lawli** este descris ca fiind unul dintre cei mai performanți asistenți juridici online, adresându-se în special firmelor de avocatură. Ceea ce îl diferențiază de celelalte servicii existente este modul în care folosește algoritmi de Deep Learning pentru a obține rezultate de ultimă generație.

Printre funcționalitățile principale ale aplicației se numără generarea unei game largi de documente, de la diferite contracte (comerciale, de angajare, de muncă) și acorduri de confidențialitate, la termeni și condiții pentru website-uri. Instrumentul de sintetizare, numit sugestiv **Lawli Summarizer,** folosește metoda abstractivă pentru a realiza rezumatul textului și oferă utilizatorilor 2 versiuni. Prima dintre ele poate fi folosită fără a avea un cont în aplicație, permițând introducerea directă a unui text de maximum 1000 de caractere, pentru care este furnizată automat o sinteză. Interfața acestei versiuni poate fi observată în figura de mai jos.

**Figura 2.2.1. Interfața Lawli Summarizer**

Cea de-a doua variantă a tool-ului de sintetizare se adresează utilizatorilor înregistrați și poate genera un rezumat al documentului încărcat în câteva secunde, indiferent de dimensiunea acestuia. Utilizatorii pot alege nivelul de concizie și pot adăuga cuvinte cheie care să fie căutate în documentul încărcat. Instrumentul acceptă documente Word (inclusiv .rtf, .doc, .docx), precum și în format .pdf. După încărcarea documentului, utilizatorul poate vizualiza rezumatul direct în browser, iar ulterior poate fi editat și exportat sub forma unui raport. Această versiune a serviciului este disponibilă pentru prețuri ce variază între 0.22$ și 1$ per pagină.

****

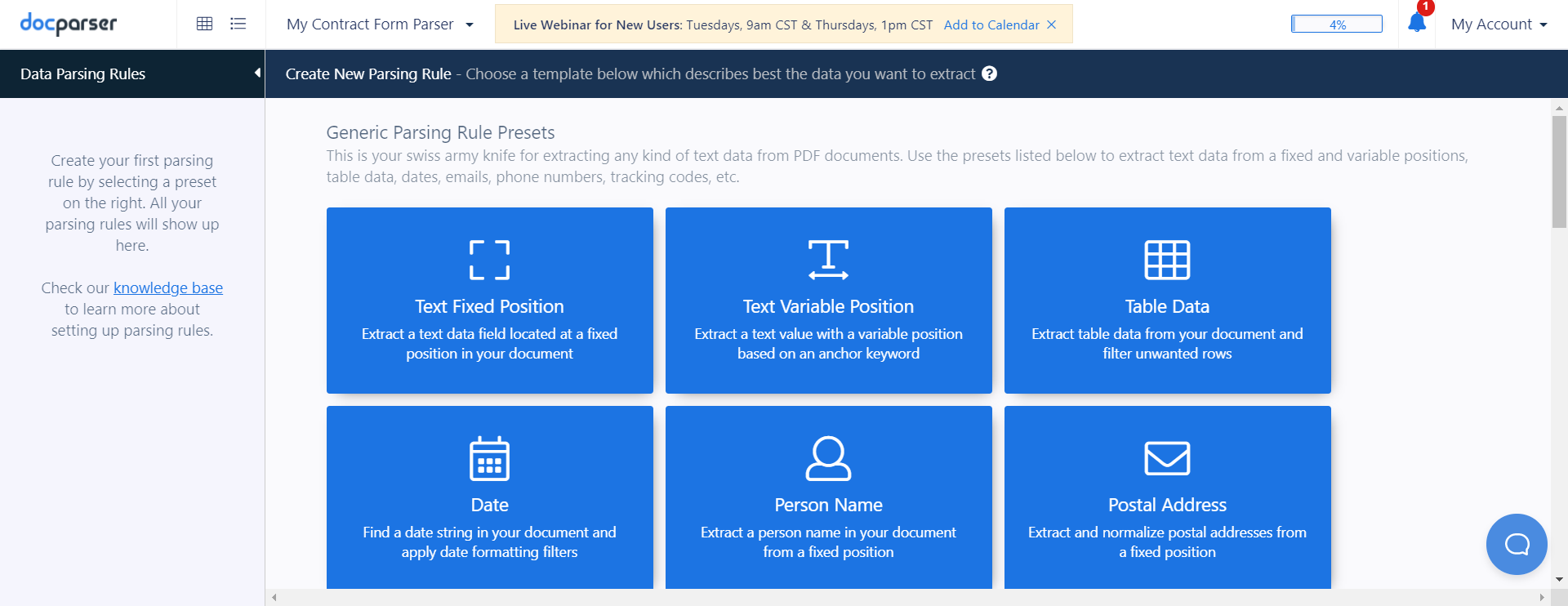
**Figura 2.2.2. Rezultat al sintetizării unui document prin Lawli**

Acest tool pus la dispoziție de platforma Lawli vine și cu o funcționalitate de stocare a documentelor încărcate, într-o librărie personală. Dacă utilizatorul nu optează pentru salvarea fișierelor în acest fel, sistemul realizează o ștergere automată tuturor documentelor la fiecare 2 luni.

**2. Docparser** este un software de procesare a documentelor, care permite utilizatorilor să extragă date din fișiere PDF, Word sau din imagini. Utilizând un motor OCR, aplicația suportă și introducerea unor documente scanate. Această soluție include o serie de opțiuni de integrare pentru a ajuta clienții să importe documente automat de la furnizorul lor de stocare în cloud (DropBox, Google Drive, OneDrive) și, ulterior, să exporte informațiile extrase în baza lor de date, foi de calcul sau sisteme ERP.

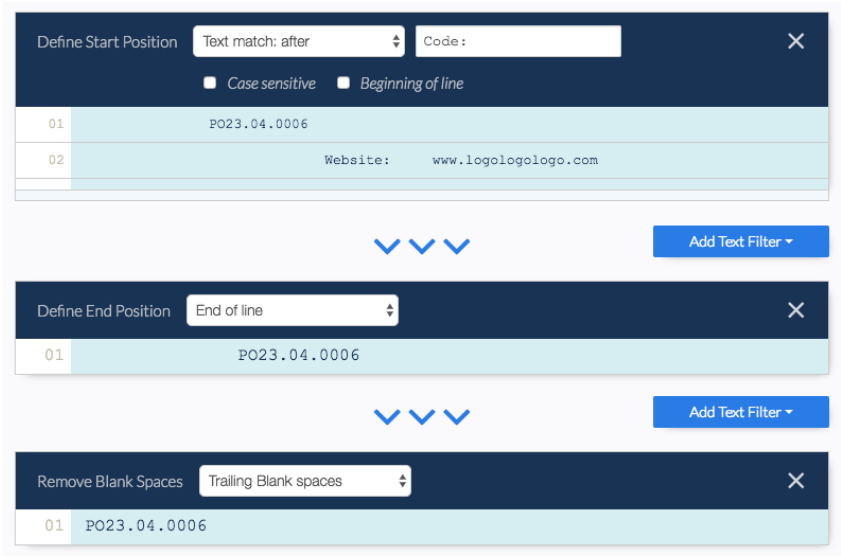
Spre deosebire de aplicația prezentată anterior, unde sintetizarea textului are loc exclusiv în *backend*, Docparser atribuie utilizatorului sarcina de a defini setul de reguli pentru parsarea documentului, astfel acesta având un mai mare control asupra datelor obținute.

Practic, după încărcarea unui document PDF, primul pas pe care clientul trebuie să îl facă este alegerea unui filtru existent pentru a crea reguli pentru extragerea informațiilor din textul documentului.

****

**Figura 2.2.3. Filtre pentru parsare în DocParser**

În funcție de cazul de utilizare, extragerea datelor se poate face pe bază de poziție, alegând opțiunea *Text Fixed Position* (de exemplu, în cazul unui număr de referință din antetul unui document), sau, pentru poziții variabile (opțiunea *Text Variable Position*), pot fi folosite diferite abordări, cea mai indicată fiind extragerea textului pe baza unor termeni ancora, așa cum este ilustrat în figura de mai jos.

****

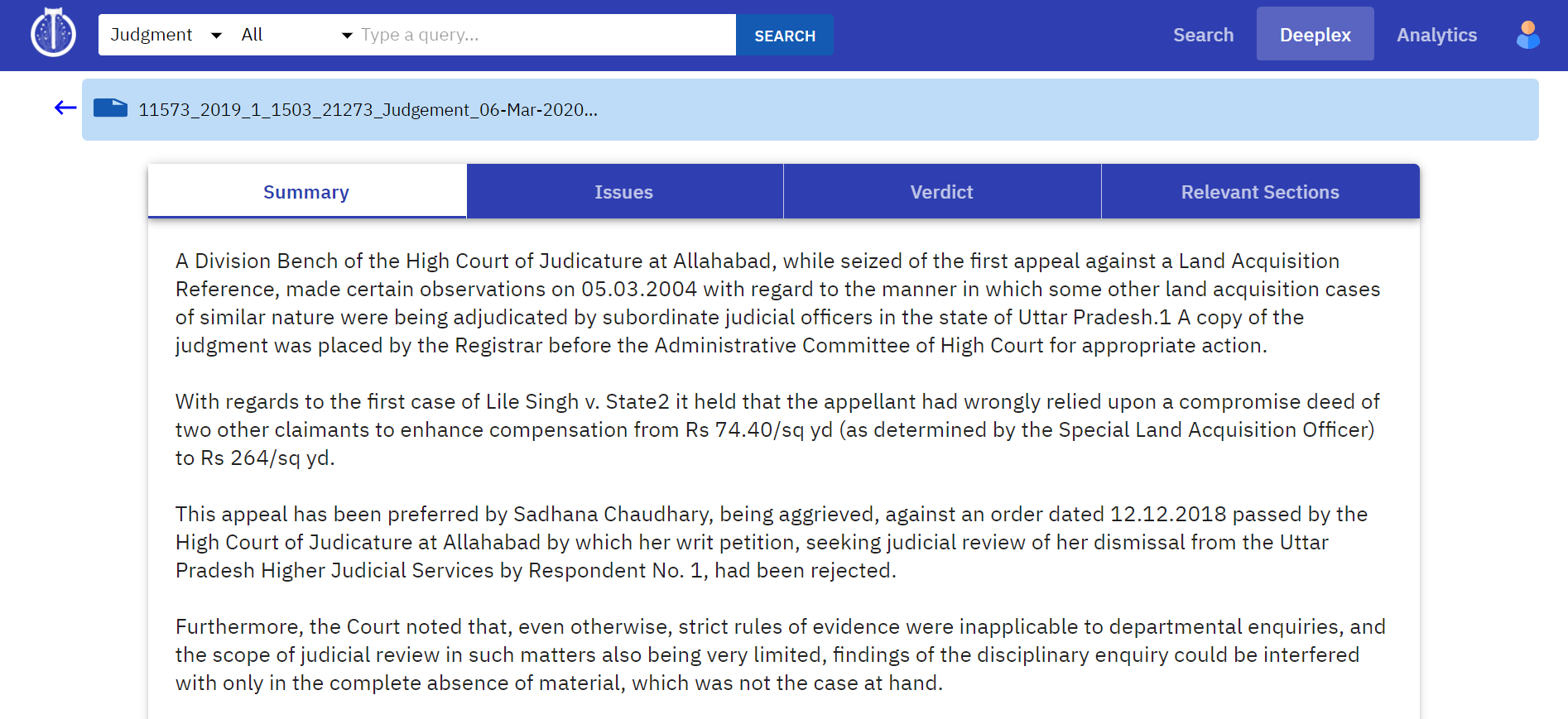
**Figura 2.2.4. Definirea unei reguli de parsare în Docparser**

*Sursă: platforma docparser.com*

Odată creată și salvată, regula de parsare poate fi aplicată asupra unui set de documente ce respectă aceeași structură. Rezultatele pot fi exportate sub diferite formate, de la Excel, CSV sau JSON, la Google Spreadsheets sau Salesforce.

Așadar, Docparser oferă o automatizare a fluxului de lucru ce implică extragere a datelor din documente, punând accentul pe personalizarea rezultatelor - este printre puținele aplicații web de acest tip care pun definirea regulilor de parsare în mâinile utilizatorilor, prin intermediul unei interfețe intuitive, oferind astfel flexibilitate în ceea ce înseamnă furnizarea anumitor informații din texte voluminoase.

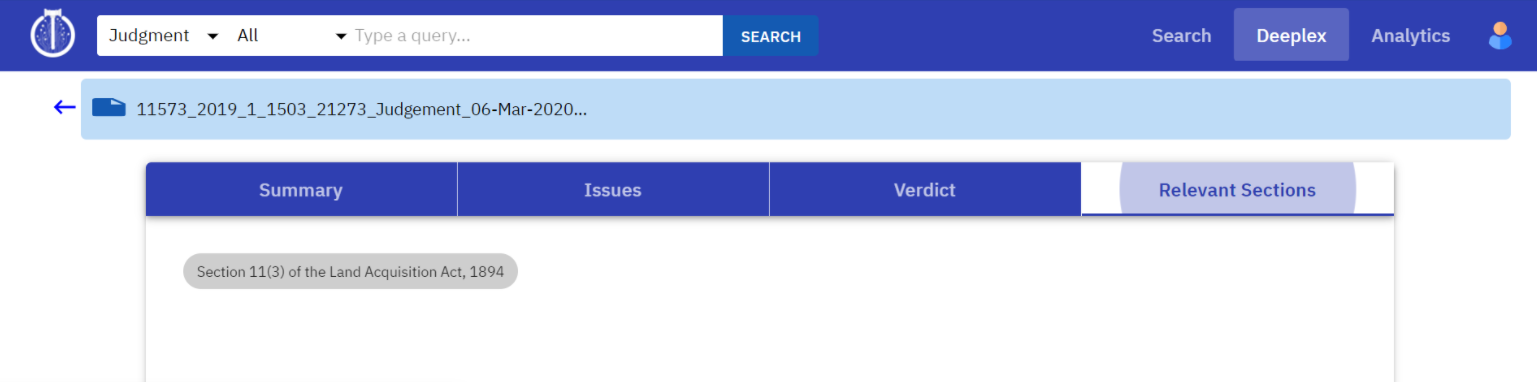
**3. LegalMind Summarizer** reprezintă o soluție care urmărește, de asemenea, creșterea productivității la locul de muncă prin implementarea Inteligenței Artificiale în sectorul juridic. Aplicația funcționează pe baza unui algoritm NLP care procesează fluxul de cazuri și clauze prezente în documentul încărcat, și afișează cele mai relevante aspecte din text. Asemenea platformei Lawli, LegalMind sintetizează textul printr-o metoda abstractivă, parafrazând conținutul pentru a ajunge la un rezultat cât mai concis, fără a pierde date esențiale.

****

**Figura 2.2.5. Sinteză obținută cu LegalMind Summarizer**

*Sursă: platforma LegalMind*

Totodată, în tab-ul *Relevant Sections* din cadrul platformei sunt prezentate articolele de lege citate în cadrul documentului încărcat de utilizator.

**Figura 2.2.6. Extragerea articolelor de lege prin LegalMind Summarizer**

*Sursă: platforma LegalMind*

LegalMind Summarizer reprezintă, astfel, un software util în procesele din domeniul juridic, rezumând deciziile judecătorești pentru o mai bună înțelegere a verdictului și, totodată, extrăgând date specifice din acestea, cum ar fi articolele de lege citate în cadrul textului.

## 2.3. Funcționalități ale aplicației în domeniul ales

Principala funcționalitate a aplicației propuse spre dezvoltare în această lucrare o reprezintă **sintetizarea conținutului unor documente contractuale**, furnizate de utilizator drept **fișiere de tip .docx**, prin analiza clauzelor principale și extragerea unor informații relevante precum data de încheiere a contractului, părțile contractante, obiectul contractului sau prețul de vânzare. Astfel, obiectivul general pe care această soluție îl urmărește este o **automatizare a sarcinilor de lucru din cadrul companiilor din domeniul juridic**, prin integrarea Prelucrării Limbajului Natural în procese uzuale, însă repetitive și solicitante din punct de vedere al timpului.

La o scurtă analiză comparativă, se observă că funcționalitatea de bază coincide cu cea a soluțiilor prezentate anterior, însă abordarea pe care o propun diferă prin mai multe aspecte. În primul rând, această aplicație este destinată firmelor de avocatură și birourilor notariale din țara noastră, **procesând texte redactate în limba română,** în timp ce soluțiile existente pe piață acceptă exclusiv documente în limba engleză, lucru de înțeles din moment ce este vorba de o limbă de circulație internațională.

Totodată, o altă deosebire o reprezintă modul de sintetizare folosit. Dacă aplicațiile enumerate folosesc o metodă abstractivă de sintetizare a conținutului, rezultatul fiind o parafrazare a textului inițial, mecanismul propus de mine va extrage datele esențiale și le va afișa **sub forma unei prezentări schematice**. Practic, rezultatul nu va lua forma rezumat clasic, ci va fi constituit dintr-o enumerare a aspectelor cheie găsite în document (numele contractanților, data de încheiere a contractului etc.).

În fond, aplicația pe care urmează să o dezvolt constituie un sistem informatic care în ciuda faptului că este mai primitiv decât soluțiile deja existente, este bazat pe o **gramatică EBNF** (*Extended Backus-Naur form*) construită în jurul limbii române, astfel contribuind la un început al automatizării proceselor ce țin de domeniul juridic din România. Sistemul ia forma unei aplicații Web ce oferă utilizatorilor o interfață *user-friendly,* facilitând astfel ceea ce se vrea a fi o digitalizare a domeniului juridic din țara noastră.

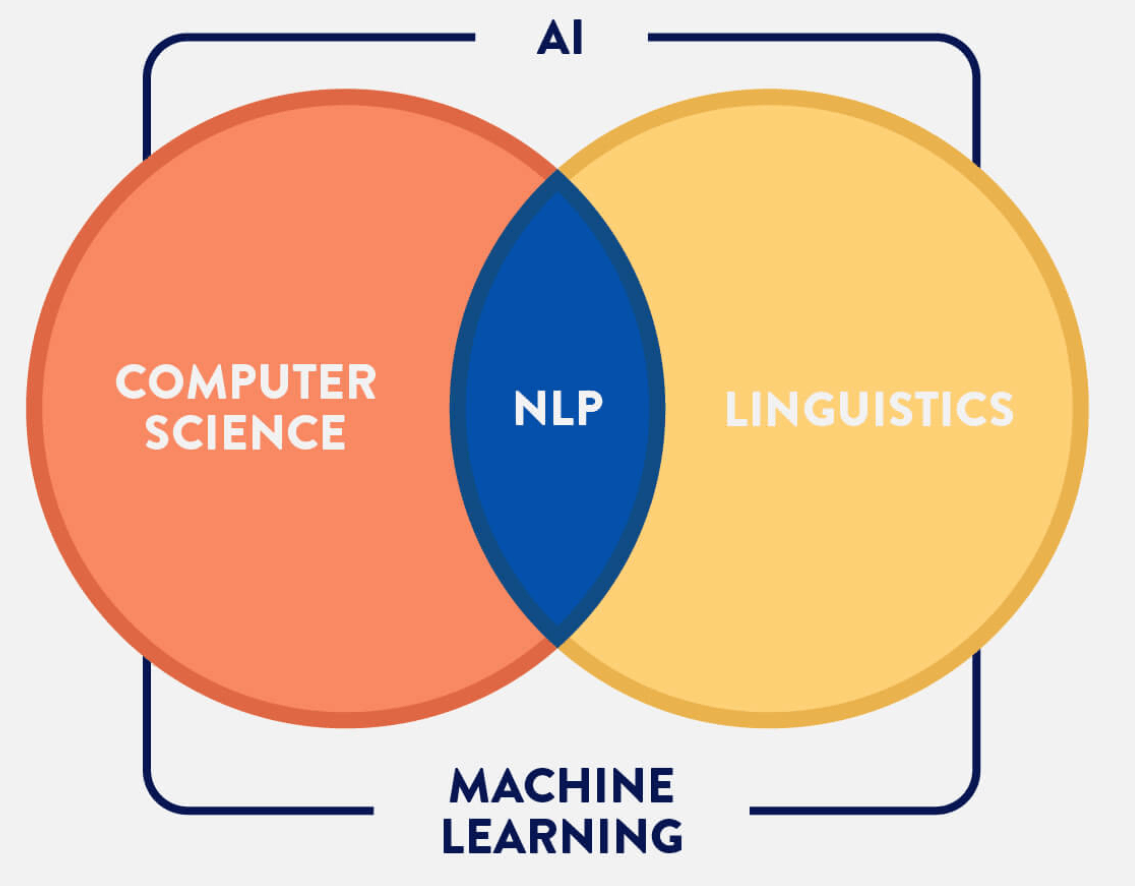
# Modelul conceptual

## 3.1. Prelucrarea limbajului natural

Așa cum am menționat anterior, modelul conceptual de la care a pornit aplicația are la bază o gramatică EBNF, însă pentru a putea defini o astfel de structură, este nevoie să înțelegem **conceptul general** care stă în spatele acesteia, și anume acela de procesare a limbajului natural. În continuare, voi aduce în lumină de unde a pornit dezvoltarea acestui domeniu, dar și impactul său asupra tehnologiei din ziua de astăzi.

În ultimii 5 ani, numărul utilizatorilor serviciilor de Internet a crescut cu 83%. Având în vedere că fiecare individ generează constant date prin diferite activități zilnice - de la o simplă căutare pe Google, până la folosirea aplicațiilor de mesagerie sau social media, este evident faptul că este produs un volum mare de text în fiecare zi. Studiile arată că în fiecare minut motorul de căutare Google înregistrează 3.5 miliarde de interogări, iar în ceea ce privește social media, numai platforma Facebook numără 4.3 miliarde de mesaje zilnic [1]. În alte cuvinte, în ziua de astăzi este necesar ca datele de tip text sau chiar audio să fie interpretate de calculatoare, iar aici intervine NLP.

Subdomeniu al inteligenței artificiale și lingvisticii, prelucrarea limbajului natural (*Natural Language Processing*, abreviat NLP) reprezintă **o abordare computerizată a analizei textului**, care se bazează interacțiunea dintre calculator și limbajul uman pentru procesarea datelor. Dacă limbajul natural reprezintă generic orice limbă care este vorbită sau scrisă de om, în scopul comunicării, pentru computere limbajul natural constituie un sistem compus dintr-un set de reguli și simboluri [2].



**Fig. 3.1.1. Prelucrarea limbajului natural**

Deși ne-am putea gândi că procesarea limbajului natural a căpătat notorietate în ultimul deceniu, de fapt istoria NLP începe în anii ‘50, atunci când **Alan Turing** a publicat articolul “*Computing Machinery and Intelligence*” unde a prezentat testul care îi poartă numele. Practic, Turing susținea ideea că putem considera că un computer are inteligență artificială dacă poate “imita” un răspuns uman în anumite condiții. Câțiva ani mai târziu, urmărind același obiectiv, **Noam Chomsky** creează o **teorie** **a gramaticii generativ-transformaționale** prin care, privind gramatica ca pe un sistem de reguli ce genereaza anumite combinații de cuvinte, acesta reușește să aplice transformări și să producă noi propoziții pornind de la cele existente. În lucrarea “*Three models for the description of language*” (1956), Chomsky evidențiază diferite tipuri pentru formularea gramaticii unei limbi, bazându-se în special pe limba engleză. Primul model este constituit printr-un **proces Markov** cu un număr finit de pași, care produce simboluri de la stânga la dreapta. Cea de-a doua metodă, mai intuitivă, implică o listă de formule inițiale și o listă finită de substituții ale simbolurilor în anumite contexte, astfel formându-se **limbaje derivabile** (*derivable language*). Ultima categorie este reprezentată de **gramaticile transformaționale**, care permit substituiri și reguli specifice pentru rearanjarea unor șiruri finite de simboluri. Concluzia la care a ajuns Chomsky este aceea că un limbaj trebuie să aibă un **nucleu finit** ce respectă o anumită **structură**, dar care acceptă **transformări** cu scopul de a produce fraze mult mai complexe [3].

Pe lângă cercetările teoretice, au fost dezvoltate mai multe prototipuri care să demonstreze eficacitatea principiilor legate de NLP. Unul dintre primele programe bazate pe NLP, numit **ELIZA**, simula o conversație dintre un psiholog și un pacient în principal prin repetarea cuvintelor utilizatorului. A urmat **SHRDLU**, un robot care arăta diverse moduri în care puteau fi manipulate blocuri de diferite forme și culori, în funcție de indicațiile utilizatorului. În ciuda faptului că aceste prime sisteme prezentau destul de multe limitări, ele au demonstrat că procesarea limbajului natural este posibilă pentru computere [4].

În ultimele decenii, **domeniul lingvisticii computaționale a luat amploare** datorită creșterii cantității de text în format electronic asociată folosirii intensive a Internetului, dar nu numai. Printre funcționalitățile majore bazate pe NLP, care sunt întâlnite tot mai des în diverse aplicații, se numără analiza sentimentelor, traducerea automată, completarea cuvintelor (*autocomplete*), recunoașterea optică a caracterelor sau rezumarea automată a textului.

Avansând atât de mult în ultimele decenii, în acest moment scopul NLP a devenit ca, prin intermediul sistemelor bazate pe AI, să proceseze datele mai bine decât o pot face oamenii. Datorită faptului că interpretarea comportamentului uman încă reprezintă o provocare chiar și pentru Inteligența Artificială, perspectiva progresului în această direcție poate fi considerată una lentă. Este important însă de reținut că NLP reprezintă doar o formă avansată de AI, iar pe măsură ce tot mai multe companii la nivel global explorează acest domeniu, este creată o bază solidă de la care tehnologia poate dezvolta în continuare [5].

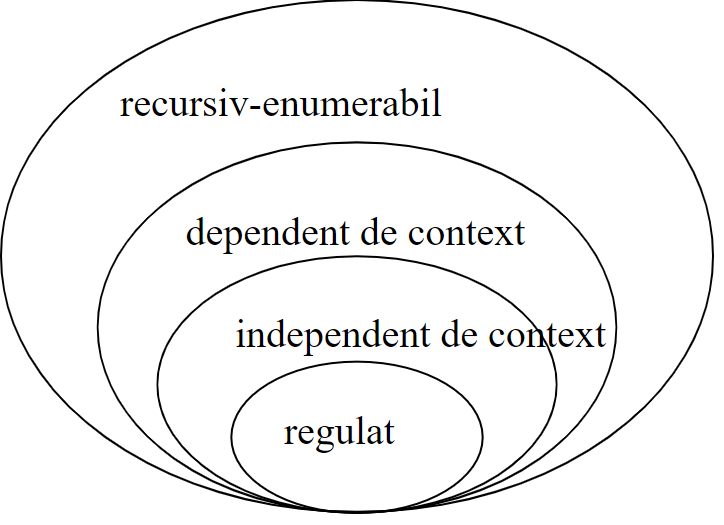
## 3.2. Detalii privind modelul conceptual

Așa cum am menționat anterior, prin realizarea acestei lucrări urmăresc dezvoltarea unei soluții web pentru extragerea de date esențiale din documente contractuale încărcate de utilizator. Algoritmul care stă la baza aplicației este construit în jurul unor noțiuni elementare de **lingvistică computațională,** un domeniu interdisciplinar, situat la intersecția informaticii cu lingvistica, ce se ocupă cu **modelarea statistică a limbajului natural**. Pentru a avea o perspectivă mai amplă asupra a ceea ce presupune această sferă, voi detalia conceptele specifice folosite în stadiul de dezvoltare a aplicației.

O primă idee ce trebuie înțeleasă este aceea a limbajului formal, utilizat ca bază a definirii unei gramatici. Acesta este prezent atât în matematică, cât și în informatică și lingvistică, și este constituit din cuvinte ale căror litere aparțin unui alfabet, cuvinte formate conform unui set specific de reguli. **Teoria limbajului formal** se ocupă, așadar, cu studiul tiparelor structurale interne ale unei limbi sau, mai pe scurt, cu regulile sintactice ale acesteia [6]. Deși nu face parte din definiție, noțiunea de **gramatică formală** este adesea asociată acestui concept, considerându-se că fiecare limbaj formal are în spate o asemenea gramatică ce îl descrie.

În teoria limbajului formal, o gramatică este definită ca **un set de reguli pentru formarea șirurilor** pornind de la alfabetul unui limbaj formal. Secvența ce constituie gramatica este formată din **simboluri non-terminale**, **simboluri terminale** și **un simbol de start**. Astfel, o regulă poate fi aplicată prin înlocuirea unei apariții a simbolurilor din partea stângă cu cele care apar în partea dreaptă, acest proces numindu-se **derivare**.

În lucrarea “*Syntactic Structures”*, lingvistul Noam Chomsky a clasificat tipurile de gramatici formale într-o serie de patru grupuri imbricate, cunoscute împreună sub numele de **ierarhia lui Chomsky** [7].

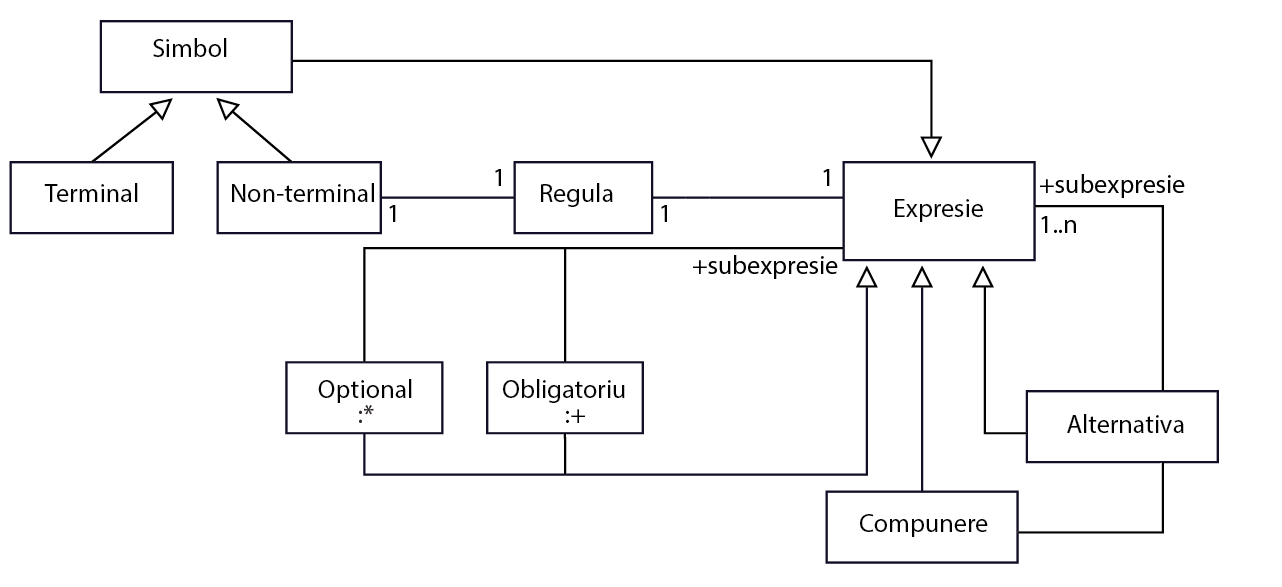


**Figura 3.2.1. Ierarhia lui Chomsky**

*Sursă: Wikipedia* [8]

* **Gramaticile de tip 0** sunt **gramatici nerestricționate**, numite și **recursiv-enumerabile,** și generează toate limbajele care pot fi recunoscute de o mașină Turing, incluzând practic toate gramaticile formale.
* **Gramaticile de tip 1** reprezintă gramaticile **dependente de context**, iar limbajele definite de această gramatică sunt doar cele care pot fi recunoscute de o mașină Turing limitată liniar.
* **Gramaticile de tip 2** nu depind de context (*context-free grammar*) și constituie **baza teoretică pentru structura expresiilor din majoritatea limbajelor de programare**, în ciuda faptului că sintaxa lor poate include și elemente dependente de context (datorită declarațiilor și a domeniului de aplicare).
* **Gramaticile de tip 3** sunt cele care **generează limbaje regulate**. Ele au un singur simbol non-terminal pe partea stângă, iar partea dreaptă constă într-un singur terminal, sau un terminal urmat de un non-terminal.

Pentru dezvoltarea aplicației propuse, este folosită o **gramatică independentă de context**, mai exact o gramatică cunoscută sub numele de gramatică **EBNF** (*extended* *Backus-Naur form*). Aceasta poate fi considerată un **metalimbaj**, din moment ce este folosită pentru a descrie o multitudine de limbaje, inclusiv **limbaje specifice** **unui domeniu** (*Domain Specific Languages*). Fiind vorba de o soluție dedicată domeniului juridic, este nevoie de o gramatică precisă, setul de reguli utilizat fiind adaptat ariei de activitate pentru o mai mare acuratețe a rezultatelor.



**Figura 3.2.2. Model conceptual al unei gramatici EBNF**

Obiectivele principale pe care o gramatică EBNF urmărește să le satisfacă sunt următoarele:

* Gramatica trebuie să fie **concisă**, astfel încât limbajul să fie definit succint și, implicit, să fie cât mai ușor de înțeles.
* Regulile definite trebuie **să** **nu prezinte ambiguități**.
* Gramatica trebuie să fie **formală**, pentru ca regulile să poată fi analizate si, ulterior, procesate de un computer.
* Notațiile folosite trebuie să fie **generale**, putând fi utilizate în mai multe contexte.
* Sintaxa trebuie să fie **liniară** (să poată fi exprimată sub forma unui singur șir de caractere).

În ceea ce privește limitările, este important de evidențiat faptul că EBNF solicită o secvență ordonată de simboluri, astfel că nu pot fi definite forme complexe de gramatici utilizându-se acest tip.

## 3.3. Componentele modelului

Sistemul este construit pornind de la un algoritm folosit în analiza lingvisticii de calcul, numit **Earley parser**. Acesta poate suporta orice gramatică independentă de context, iar avantajul îl reprezintă timpul scurt de analiză, care pentru o gramatică fără erori de ambiguitate, este de O(), unde n reprezintă lungimea șirului dat spre analiză, iar în cel mai rău caz performanța algoritmul este de O() [9].

Această abordare urmărește 4 aspecte importante, și anume **scopul curent**, **poziția de start**, **lista de componente rămase** de analizat pentru atingerea obiectivului**, punctul final** al obiectivului. Astfel, cele **3 componente principale** ale algoritmului sunt un *predictor*, un *scanner* și un *completer*. Pentru a înțelege cum funcționează acestea, trebuie menționat că Earley realizează **o analiză incrementală**, de la stânga la dreapta.

* **Predictor** are rolul de a anticipa următorii pași în analiza unei gramatici și de a genera noi reguli, folosindu-se de o abordare *top-down.*
* **Scanner** citește noile reguli.
* **Completer** recunoaște regula definită anterior și ajută la avansarea analizei.

Cu o idee generală asupra algoritmului Earley, putem să analizăm acum **structura unui parser**. Acesta are în spate 2 părți principale: un **lexer**, cunoscut și sub numele de tokenizer, și parserul propriu-zis. Acestea funcționează într-o ordine stabilită: mai întâi lexer-ul citește un input și produce *tokens*, pe care parserul le scanează, ulterior producând rezultatul analizei.

De exemplu, pentru analizarea unei expresii de adunare, lexerul citește textul și găsește concatenări de cifre, pe care le recunoaște drept numere. De asemenea, acesta găsește simbolul care constituie operația matematică efectuată și îi atribuie un token corespunzător. Parserul combină token-urile găsite și le grupează.

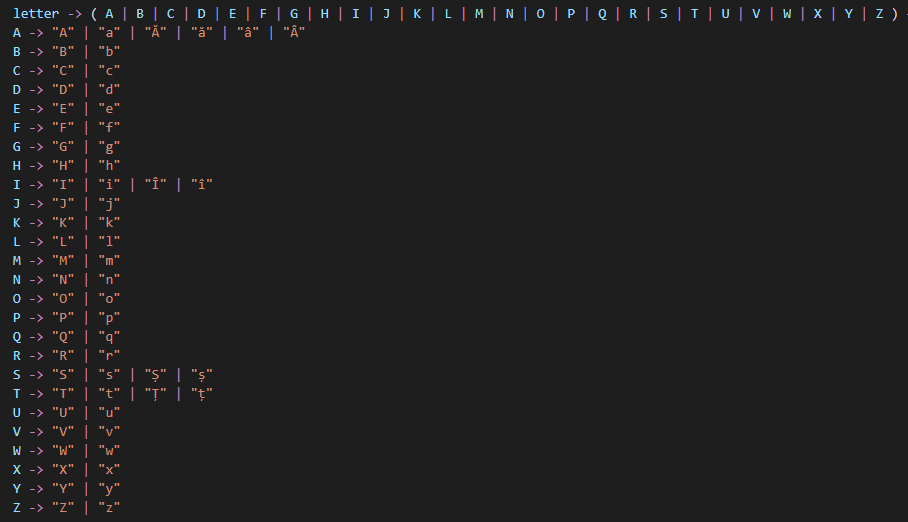


**Figura 3.3.1. Analiza unei expresii matematice**

Din figura anterioară observăm că, în final, rezultatul ia forma unui*Abstract Syntax Tree*, un **arbore de analiză** care reflectă structura expresiei primite ca *input*.

Îndreptându-ne atenția către modelul concret elaborat pentru dezvoltarea aplicației care stă la baza acestei lucrări, voi evidenția câteva aspecte esențiale de care am ținut cont în construirea acestei gramatici.

Așa cum am menționat anterior, noțiunile de bază ce intră în componența unui vocabular sunt simbolurile **terminale**, cele **non-terminale** și **regulile de producție.**

****

**Figura 3.3.2. Regula de producție pentru caracterele alfabetului latin**

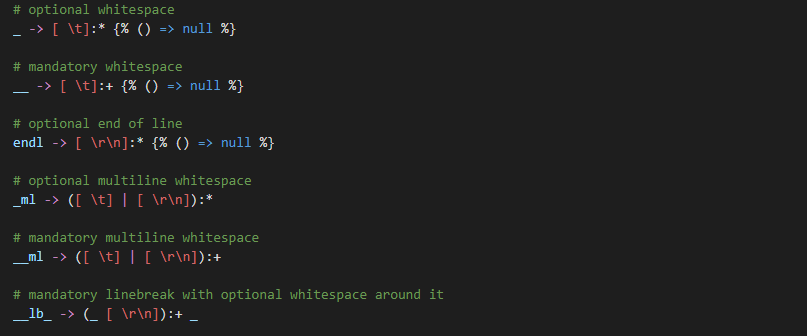
În figura de mai sus, fiecare literă așezată între ghilimele (de exemplu, "A") reprezintă un caracter terminal, iar literele definite în partea stângă constituie caracterele non-terminale. Regula de producție este întreaga definiție unui caracter non-terminal. Astfel, putem spune ca o literă poate avea oricare dintre valorile caracterelor de la A la Z, indiferent dacă este scrisă cu litere mici sau cu majuscule.

Alte convenții specifice sintaxei EBNF sunt următoarele:

* Simbolul "**|**" are rolul de a separa regulile alternative pentru un non-terminal.
* Repetiția de tipul 0..n este indicată cu ajutorul simbolului "**\***", iar repetiția de tipul 1..n prin simbolul "**+**".
* Caracterele explicite sunt marcate prin ghilimele.
* Gruparea caracterelor poate fi făcută în interiorul parantezelor.
* Un non-terminal urmat de mai multe simboluri terminale, fără a fi despărțite de alți modificatori, sugerează concatenarea acestora.
* Deși o gramatică nu acceptă regex drept caractere terminale, pot fi folosiți modificatori EBNF în acest scop, pentru definirea unei reguli ce ilustrează valorile pe care o cifră le poate lua spre exemplu.

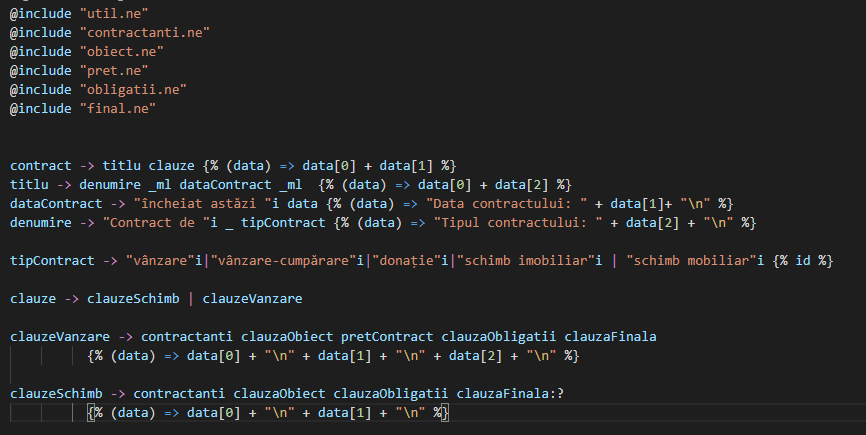
Gramatica trebuie să fie structurată pornind de sus în jos (*top-down*), ceea ce înseamnă că primele reguli au rolul de a descrie o schiță generală, iar spre final apar elemente tot mai detaliate, precum simboluri terminale pentru spațiul alb, cifre, litere etc. Astfel, de cele mai multe ori, regulile referite în partea de sus a unei gramatici sunt definite în partea de jos a acesteia.

Pentru o înțelegere cât mai clară a gramaticii definite, am ales să utilizez reguli de producție auxiliare pentru caractere frecvent întâlnite, precum spațiul alb, finalul unui rând sau începutul unui paragraf.



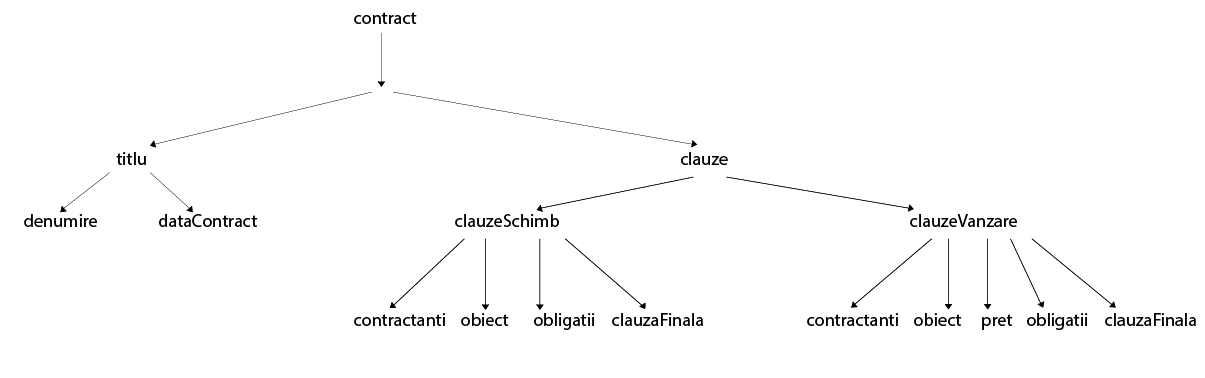
**Figura 3.3.3. Reguli de producție auxiliare**

Deși construirea unei gramatici are un caracter instinctiv, atunci când vine vorba de un nivel ridicat de complexitate, este important să fie respectate principii de *clean code* pentru a păstra totul cât mai organizat și ușor de înțeles. Tocmai din acest motiv, am structurat vocabularul în funcție de clauza pe care o descrie, în fișiere distincte. Astfel, importând aceste fișiere am reușit să definesc o gramatică ce poate identifica diferite tipuri de contracte.



**Figura 3.3.4. Gramatică ce descrie structura unui contract**

Așadar, pe baza gramaticii descrise în figura anterioară se poate distinge un arbore de sintaxă (*Abstract Syntax Tree*), care ilustrează nivelurile de detaliere din cadrul analizei realizate.



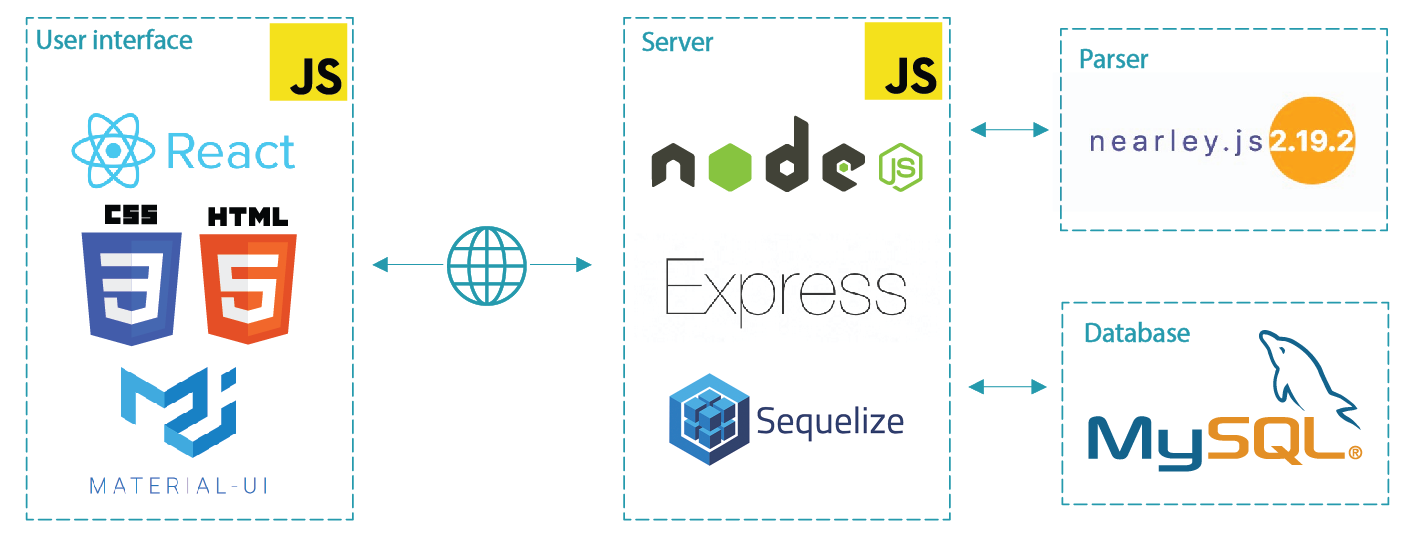
**Figura 3.3.5. Arborele de sintaxă al gramaticii definite**

Se observă ca sunt tratate diferite clauzele unui contract de schimb față de cele ale unui contract de vânzare-cumpărare, deoarece aceste documente au structuri distincte. La rândul lor, nodurile reprezentate de clauze nu pot fi considerate finale, din moment ce fiecare gramatica fiecărei clauze este definită separat.

În concluzie, componenta principală din spatele mecanismului de sintetizare a documentelor o reprezintă parserul bazat pe o gramatică EBNF, care reușește să analizeze texte ale contractelor și să furnizeze o abstractizare a acestora, prin extragerea aspectelor importante din cadrul fiecărei clauze.

# Tehnologii utilizate

Am ales ca soluția propusă în această lucrare să ia forma unei aplicații web, deoarece în acest fel oferă **accesibilitate** indiferent de *device*-ul folosit de utilizator. În dezvoltarea ei, am urmărit să folosesc tehnologii care protejeze datele utilizatorilor, să permită o mentenanță ușoară și, nu în ultimul rând, să aducă în fața clienților o interfață intuitivă.



**Figura 4.1.1. Tehnologiile utilizate în dezvoltarea aplicației**

## 4.1. JavaScript

Fiind unul dintre cele mai folosite limbaje de programare din lume la momentul actual, JavaScript a câștigat popularitate în ultimii ani mai ales în ceea ce privește dezvoltarea web. Caracterizat prin ușurință în interpretare și compilare rapidă, acesta înregistrează poate cea mai rapidă creștere în rândul limbajelor de programare, **66,7%** din programatori manifestând o preferință pentru utilizarea sa [10]. Printre motivele principale care au contribuit la această tendință în folosirea JavaScript se numără faptul că este un **limbaj multi-paradigmă**, ce implementează principiile programării orientate obiect (*Object Oriented Programming*), susținând în același timp filozofiile de programare procedurală.

Deoarece este un limbaj conceput pentru a rula în *browser* în principal, oferă un API standard ce permite procesarea de fișiere HTML și JSON. Este important de menționat că majoritatea browserelor moderne folosesc un compilator *just-in-time* (JIT), lucru care face din JavaScript un limbaj foarte rapid - mai rapid decât Java, și în unele cazuri la fel de rapid ca C/C++.

JavaScript poate fi folosit atât pentru partea de *backend*, cât și pentru cea de *frontend* a unei aplicații. Pentru că în urmă cu câțiva ani, compilatoarele JS erau disponibile doar în browser, JavaScript a fost numit un *Client-Side Scripting Language*.Acum însă, JavaScript suportă și funcționalitatea unui server, permițând accesul la resurse precum baza de date sau fișiere locale, fără intervenția directă a utilizatorului. Astfel, JavaScript oferă **o soluție completă** pentru dezvoltarea de aplicații web.

Din perspectiva clientului, JavaScript furnizează obiecte pe care le integrează apoi într-un *Document Object Model* (DOM), ce constituie practic interfața cu care utilizatorii iau contact. DOM structurează întreaga pagină web sub formă de noduri și obiecte, făcând întreaga reprezentare a documentului disponibilă prin intermediul unui singur API (*Application Programming Interfaces*) consistent. În acest fel pot fi create funcționalități complexe mult mai ușor, iar paginile web pot fi personalizate prin introducerea de animații, butoane, meniuri sau alte elemente specifice HTML [11].

Pe partea de server, JavaScript poate răspunde solicitărilor primite din partea browserului sub forma de *requests.* Astfel, pot fi realizate manipulări de fișiere sau interogări personalizate prin comunicarea cu baza de date, în urma cărora aspectul paginii web (DOM) poate suferi modificări în mod dinamic.

Așadar, aplicații complexe, cuprinzând atât *backend*, cât și *frontend*, pot fi dezvoltate exclusiv folosindu-se JavaScript. Putem spune, deci, că limbajul de programare se remarcă prin această versatilitate, dar și prin viteza de compilare și simplitatea sintaxei.

JavaScript este actualizat în fiecare an de către Asociația Europeană pentru Standardizarea Sistemelor de Informații și Comunicații (Ecma International), versiunea standardizată numindu-se ECMAScript.

## 4.2. Nearley.js

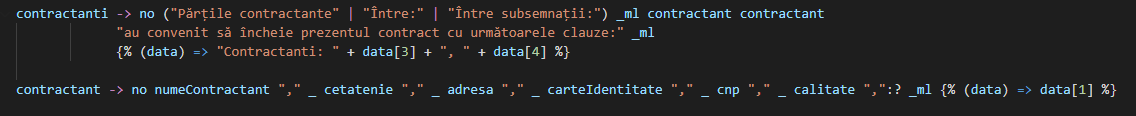
**Nearley** reprezintă un **toolkit Javascript** ce transformă șirurile de caractere în structuri de date semnificative (de exemplu, un obiect JSON), fiind primul instrument de acest gen care se bazează pe **algoritmul Earley**. Pe scurt, Nearley citește fiecare linie a unui document și tratează fiecare cuvânt și caracter ca ceea ce se numește un simbol terminal. Atunci când este găsită o potrivire cu unul dintre șirurile definite, poate fi folosită o metodă de post procesare pentru a prelucra datele identificate. Printre avantajele folosirii acestui instrument se numără faptul că poate analiza orice șir de caractere fără a intra într-o buclă infinită, iar în cazul existenței unor erori, Nearley afișează detaliat care este problema și unde se află mai exact în setul de reguli definite.

Nearley este alcătuit din **2 componente**, un *compiler* și un *parser,* amândouă fiind publicate sub forma unui pachet NPM unic, **compatibil atât cu Node.js, cât și cu majoritatea browserelor**. Aducând în lumină ușurința de utilizare, sunt necesare **3 etape** pentru ca instrumentul să realizeze o analiză a unui șir de caractere.

**1. Definirea gramaticii** reprezintă primul pas și constă în folosirea sintaxei **EBNF** pentru a descrie limbajul specific domeniului vizat.

În mod implicit, Nearley îmbracă tot ceea ce corespunde unei reguli definite într-o structură de tip *array*. Acest arbore de analiză (*parse tree*) este creat atunci când gramatica este identificată într-un text, însă de cele mai multe ori acest lucru nu e de ajuns. În general, prin folosirea unei gramatici urmărim nu doar să extragem, ci să și procesăm anumite date, iar în acest sens putem utiliza ceea ce Nearley numește *postprocessors* [12].

În esență, această post-procesare presupune existența unei funcții JavaScript care să returneze o versiune prelucrată a rezultatului. Funcțiile sunt cuprinse în interiorul unei structuri de tipul {%%} și sunt scrise fie după o regulă de producție, fie după fiecare alternativă a uneia. Pentru secvențele fără o încărcătură semantică, precum spațiul alb sau finalul unei linii, putem alege să fie ignorate folosind un *postprocessor* care să întoarcă valoarea *null*.



**Figura 4.2.1. Utilizarea funcțiilor de postprocesare**

În figura anterioară, sunt prezente două funcții de postprocesare. Prima este folosită pentru a formata rezultatul analizei clauzei ce cuprinde contractanții, furnizând practic formatul pe care urmăresc să îl ia rezultatul. Cea de-a doua funcție întoarce numele unui contractant, aflat pe poziția 1 în șirul ce constituie regula de producție a elementului non-terminal „contractant”.

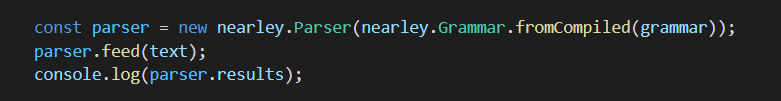
**2. Compilarea** reprezintă cea de-a doua etapă esențială în crearea unui obiect de tip *parser*. Deoarece Nearley nu este conceput pentru a fi folosit în *browser,* este nevoie ca gramatica definită să fie precompilată, generându-se astfel un fișier JavaScript, fișier care ulterior poate fi suportat de *browser*.

**3. Parsarea** constituie pasul final și concretizează rezultatele urmărite prin aplicarea regulilor de producție din cadrul gramaticii create. Pentru a putea utiliza un *parser* este necesar importul modulului *nearley*, dar și al gramaticii definite.



**Figura 4.2.2. Importul modulului nearley și al gramaticii**

Apoi, utilizăm gramatica pentru a crea un nou obiect *parser,* căruia putem să îi trimitem un șir de caractere pentru a fi analizat. În final, este interogată proprietatea *.results* a obiectului *parser,* pentru afișarea rezultatelor analizei textului.

****

**Figura 4.2.3. Interogarea rezultatelor analizei gramaticii**

Rezultatul analizei returnează o matrice, deoarece o gramatică poate analiza un șir în mai multe moduri diferite. Acest lucru se întâmplă în cazul gramaticilor ambigue, pentru care există mai multe derivări la stânga, mai multe derivări la dreapta sau mai mult de un arbore de analiză pentru șirul de caractere furnizat. Deși Nearley oferă toate analizele unei gramatici, este recomandat ca ambiguitățile să fie evitate.

Pentru a testa dacă o gramatică este ambiguă, se poate verifica valoarea *parser.results.length*: dacă aceasta nu este exact 1, atunci există mai multe rezultate datorate ambiguităților ce țin de gramatica definită. Pentru a corecta acest aspect, putem interoga pur și simplu *parser.results[0]*.

Așadar, am ales să folosesc Nearley în construirea și prelucrarea gramaticii pentru că oferă o analiză rapidă a expresiilor, remarcându-se totodată prin ușurința în utilizare.

## 4.3. Node.js

În ziua de astăzi, orice aplicație web de succes trebuie să permită și să gestioneze o creștere eficientă în timp (fie că vorbim de numărul de utilizatori sau de funcționalitățile adăugate ulterior), deci este esențial să fie proiectată încă de la început avându-se în vedere **scalabilitatea**. Când utilizăm orice platformă web, nu vrem să așteptăm ca o imagine să se încarce, sau un formular să fie procesat, pentru că din perspectiva unui utilizator, pe lângă funcționalitate, primul aspect pe care îl remarcăm cu privire la o aplicație este cât de rapid procesează datele introduse de noi, indiferent de volumul de utilizatori cu care serverul se confruntă la momentul respectiv. Aici intervine Node.js, care gestionează **comunicarea client-server** într-un mod **asincron**.

Node.js reprezintă un mediu de execuție *open-source*, de tip *cross-platform*, care permite programatorilor să folosească JavaScript pentru a produce **conținut dinamic** al paginilor web prin rularea de scripturi din server. Dezvoltarea de servere web rapide în JavaScript devine astfel posibilă prin integrarea programării bazate pe evenimente în serverele ce stau în spatele aplicațiilor.

Toate API-urile bibliotecii Node.js au un caracter asincron, ceea ce se reflectă în faptul că **nu pot exista blocaje**. Practic, un server nu așteaptă ca un API să returneze date, ci apelează următorul API, iar răspunsul de la API-ul anterior ajunge la server prin intermediul unui mecanism de notificare. În acest sens, un concept important ce trebuie înțeles este acela de *callback.* Echivalent asincron al unei funcții, un *callback* este apelat la finalizarea unei sarcini. Astfel de funcții sunt des folosite în Node, întrucât toate API-urile specifice sunt scrise în așa fel încât să le accepte [13].

Node.js funcționează pe baza unor librării reutilizabile, numite **module**. Acestea sunt gestionate de *Node Package Manager* (**NPM**), care constituie cel mai mare registru software din lume, dezvoltatorii putând să partajeze sau să folosească pachetele puse la dispoziție pe website-ul oficial [*www.npmjs.com*](http://www.npmjs.com)*.* Modulele necesare pot fi instalate local prin apelarea comenzii **npm install** în terminal, iar apoi pot fi importante în cadrul aplicației prin intermediul funcției *built-in* **require,** care citește și execută fișierul JavaScript corespunzător pachetului, iar apoi îl returnează sub forma unui obiect exportat [14].

Prin urmare, principalele motive care m-au făcut să aleg Node.js pentru implementarea aplicației sunt **performanța** în ceea ce privește **scalabilitatea** și **optimizarea randamentului**, dar și **viteza** de execuție, rulând pe motorul **Google V8**. De asemenea, accesul la o gamă vastă de **pachete NPM** disponibile a reprezentat un alt avantaj al Node.js, deoarece acestea oferă funcții simplificate care pot suporta un nivel ridicat de complexitate.

## 4.4. Express.js

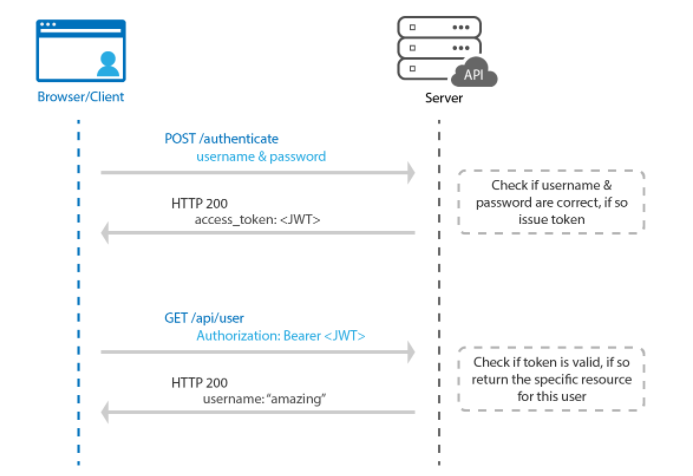
Express.js reprezintă cel mai popular *framework* pentru aplicațiile scrise cu Node.js, întrucât necesită o configurație minimă pentru a porni o aplicație sau un API. Definit prin flexibilitate și minimalism în scrierea codului, acest framework permite configurarea unor funcții *middleware* cu rolul de **a gestiona solicitările HTTP (Hypertext Transfer Protocol)** bazate pe operațiile de tip **CRUD (Create, Read, Update, Delete)** pe rute diferite, care determină modul în care aplicația răspunde la o cerere a clientului către un anumit *endpoint.*

Principalele metode HTTP sunt următoarele:

* **POST**, utilizată pentru crearea de noi resurse,
* **GET**, folosită pentru a citi sau a extrage resurse, returnând reprezentări sub formă de XML sau JSON,
* **PUT**, care are rolul de a actualiza resurse,
* **DELETE**, utilizată pentru ștergerea resurselor.

Pentru a simplifica integrarea anumitor funcționalități în aplicație, am folosit câteva funcții *middleware*. Acestea acționează ca un *layer* intermediar, ascuns, care permite comunicarea între client și server, și gestionarea corespunzătoare a datelor manipulate.

* **Multer** controlează resurse de tipul *multipart/form-data,* adăugând un obiect *body* și un obiect de tip *file* la *request*-ul primit. Acesta este necesar pentru **încărcarea fișierelor** în cadrul unei aplicații și oferă control total prin stocarea lor pe disc, prin folosirea motorului *DiskStorage*.
* **Body-parser** permite parsarea corpului unei solicitări primite.
* **Bcrypt** constituie o funcție de **criptare a parolelor**, care oferă diferite grade de **securitate**, în funcție de numărul de iterații ales de dezvoltator. Astfel, aplicația asigură utilizatorilor o siguranță împotriva atacurilor cibernetice.
* **JSON Web Token** (JWT) reprezintă atât un standard, cât și un *middleware,* care face posibilă transmiterea în siguranță a datelor în cadrul unei aplicații. Cel mai frecvent scenariu care implică JWT este **autentificarea**. În momentul în care utilizatorul se conectează într-o aplicație, ii este atribuit un *bearer token*, care îi permite să acceseze rute și resurse în sesiunea de lucru curentă. Tokenul este constituit dintr-un șir de caractere criptat, care este trimis către *header*-ul *Authorization* ori de câte ori se fac request-uri către resursele care se doresc a fi protejate [15].



**Figura 4.4.1. Autentificare prin JSON Web Token**

*Sursă: site developer.okta.com* [16]

## 4.5. MySQL

Având în vedere că aplicația propusă implică gestionarea unei colecții de fișiere încărcate de utilizator, pentru stocarea și manipularea datelor este nevoie ca o bază de date să fie proiectată. În acest sens, am ales să utilizez MySQL, fiind **unul dintre cele mai populare sisteme de gestiune a bazelor de date** la acest moment.

Pornind de la limbajul SQL (*Structured Query Language*), MySQL administrează **baze de date relaționale**, ceea ce înseamnă că resursele sunt organizate în mai multe tabele între care există relații ce ajută la structurarea datelor în ansamblu. Astfel, pe baza acestor relații, este generată **o schemă a bazei de date**.

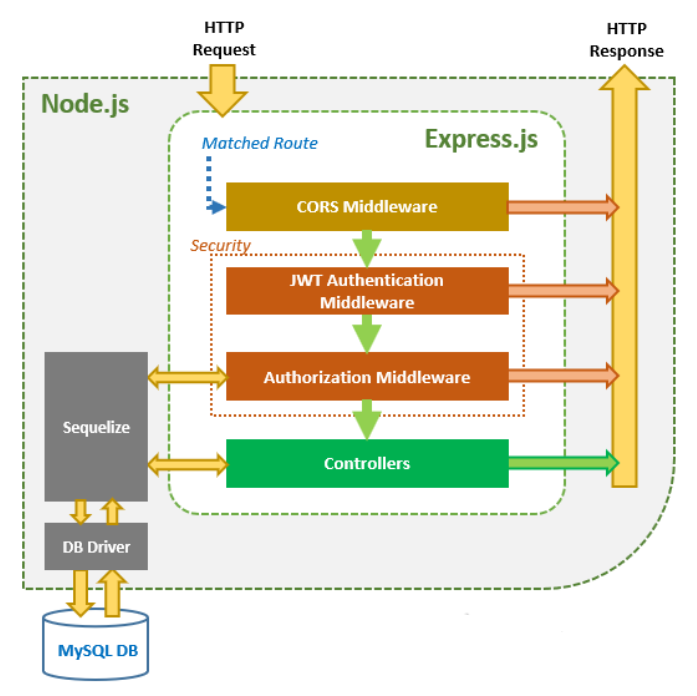
Printre avantajele folosirii acestui SGBD se numără **securitatea datelor** și **scalabilitatea,** aspecte pe care le-am urmărit încă din stadiul de proiectare a bazei de date. Stând la baza multor aplicații complexe, precum Facebook, Twitter sau Youtube, MySQL este capabil să gestioneze **volume uriașe de date (**limita implicită pe care MySQL o suportă este de 4 GB, însă poate fi mărită la 8 TB**)**, asigurând în același timp o **viteză optimă** [17].

## 4.6. Sequelize

Sequelize reprezintă o **bibliotecă ORM** (*Object-relational mapping*) care oferă acces la resursele stocate în baza de date MySQL. Practic, Sequelize constituie o punte între datele manipulate prin intermediul rutelor create în partea de backend a aplicației și baza de date efectivă. Această legătură se realizează prin definirea unor modele, care sunt echivalentul tabelelor reprezentate în codul sursă al aplicației. Fiind o abstractizare a unei entități din baza de date, Sequelize reține numele tabelei și denumirile coloanelor, alături de tipurile de date ale acestora.

Odată construite modelele, acestea pot fi **sincronizate cu baza de date existentă**, prin intermediul funcției *sync()*. În cazul în care nu există o bază de date, Sequelize oferă posibilitatea de a **automatiza crearea oricărei tabele** definite anterior ca model. Ulterior, dezvoltatorii au la dispoziție mai multe metode integrate pentru realizarea interogărilor. De exemplu, interogările de tipul SELECT sunt transpuse în cod prin funcții predefinite precum *findOne*, *findAll* sau *findByPk*.

Așadar, împreună cu Express, MySQL și Sequelize, Node.js face posibilă construirea unui **API RESTful** (*Representational State Transfer*) ce constituie arhitectura componentei de *backend* a aplicației, așa cum este ilustrat în figura de mai jos [18].



**Figura 4.6.1. RESTful API**

*Sursă: site bezkoder.com* [19]

## 4.7. React.js

Pentru partea de *frontend* a aplicației, am ales să utilizez React.js, o librărie open-source care are rolul de a construi **interfețe** și **componente UI** (*User Interface*). Fiind vorba de o **aplicație SPA***(Single Page Application),* React a reprezentat opțiunea potrivită pentru dezvoltarea componentei vizuale, deoarece permite modificarea dinamică a datelor, fără ca pagina să fie reîncărcată.

React prezintă o **sintaxă opțională**, derivată din JavaScript, numită **JSX,** care face construirea de noi elemente mult mai ușoară. De asemenea, JSX vine în ajutor și la capitolul **performanță** a aplicației, asigurând o redare rapidă a datelor. Acest lucru se datorează și implementării **Virtual DOM,** concept care presupune, după cum sugerează și numele, o reprezentare virtuală a DOM astfel încât modificările sunt prima dată introduse la nivelul arborelui virtual, iar ulterior, prin aplicarea unui minim de operații, DOM real este actualizat corespunzător, asigurând astfel un **timp minim consumat**.

Un alt concept cheie pe care se bazează React îl constituie **componentele**. Acestea sunt piese **reutilizabile**, dar **independente** care funcționează precum metodele JavaScript: acceptă intrări arbitrare, numite *props,* și returnează elemente ce fac parte din interfața aplicației. Componentele pot fi structurate sub formă de **clase** sau **funcții**. Deși din punct de vedere al performanței, nu există o diferență între cele două variante, în ceea ce privește implementarea propriu-zisă, lucrurile stau diferit.

* Componentele sub forma de clase sunt componente *stateful* pentru că implementează o stare pentru fiecare obiect. Cu o logică mai complexă în spate, acestea utilizează o metodă *render()* pentru a întoarce un rezultat sub forma unei structuri JSX, care este transpus ulterior sub forma unei interfețe.
* Componentele sub formă de funcții sunt cunoscute drept *stateless,* deoarece acceptă pur și simplu datele iar apoi le redau sub forma unei interfețe. Acceptă *props*, însă în final oferă o soluție fără a utiliza starea obiectului furnizat. Acestea nu folosesc metoda render(). Pentru că aceste componente sunt simple funcții JavaScript, ele nu acceptă *Lifecycle Methods* precum *componentDidMount()* sau *setState().* În schimb, pot fi folosite **React Hooks** în interiorul unor astfel de componente, pentru a imita comportamentul metodelor menționate anterior. De exemplu, pentru reținerea stării se poate utiliza *useState* [19].

Cu un impact semnificativ asupra **productivității** în ceea ce privește dezvoltarea componentei de *frontend* și **mentenanța** ulterioară a acesteia, React facilitează procesul de creare a componentelor recurente, aspect care m-a determinat să aleg această tehnologie pentru implementarea aplicației.

## 4.8. MaterialUI

Bazat pe Google Material Design, Material UI reprezintă un framework open-source pentru componente React, care urmărește să ofere o **experiență digitală completă** prin prisma graficii *frontend*. Punând la dispoziția utilizatorilor **peste 60.000 de componente**, Material UI își concentrează atenția asupra furnizării unor designuri moderne, simpliste, dar care atrag atenția prin modul în care componentele aruncă umbre și reflectă lumina, dar și prin folosirea animațiilor și a tranzițiilor.

Ceea ce m-a determinat să folosesc Material UI, pe lângă colecția vastă de componente și designurile plăcute, a fost **documentația detaliată** disponibilă pe site-ul oficial [material-ui.com](https://material-ui.com/). De asemenea, un aspect important al acestui framework este că elementele sunt consecvente în tonuri de culoare, însă pot fi personalizate de către dezvoltator, ceea ce oferă un caracter unitar și autentic întregii aplicații.

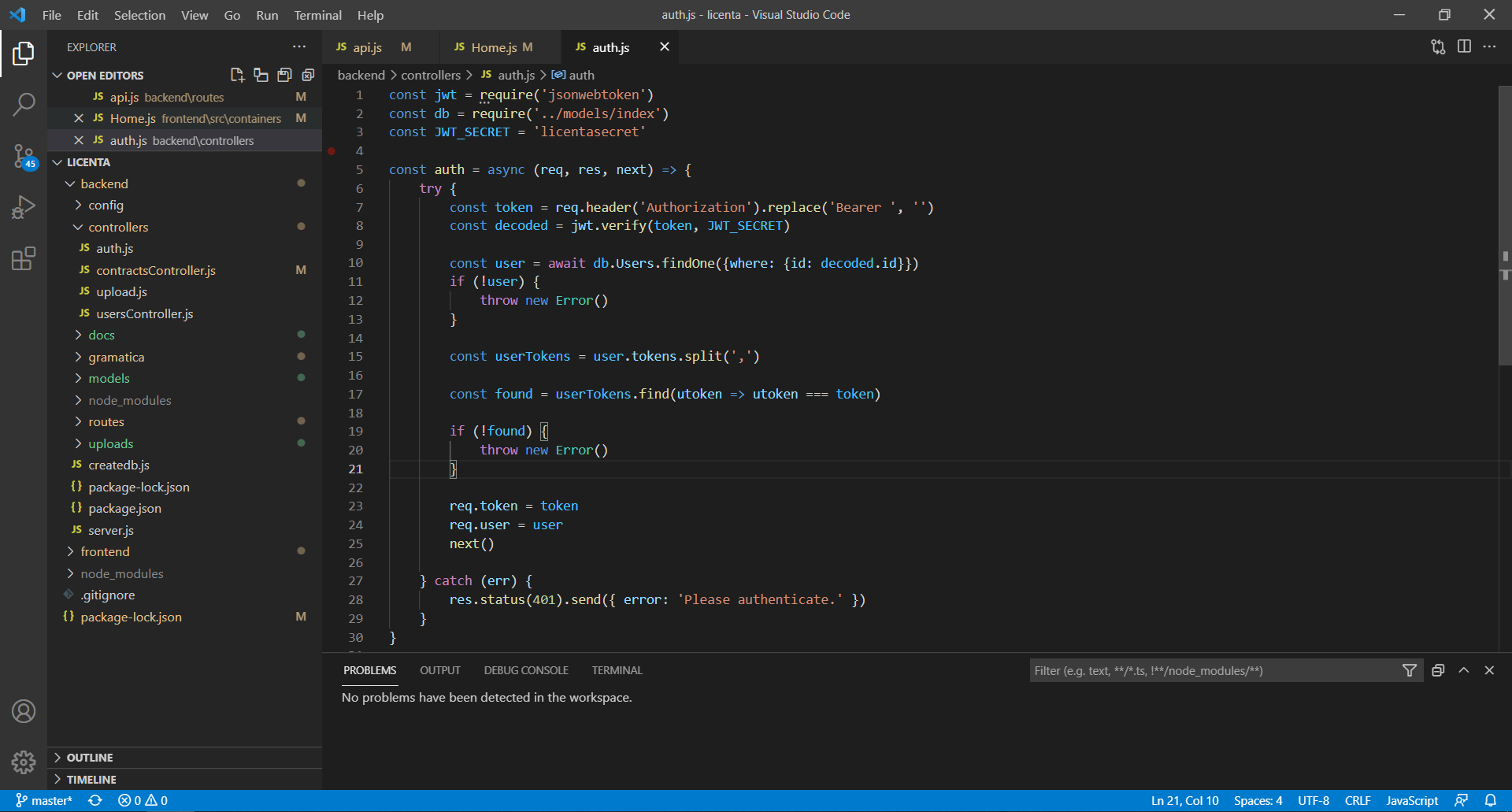
# Detalii de implementare

## 5.1. Medii de dezvoltare utilizate

**1. Visual Studio Code**

Ca mediu de dezvoltare principal, am ales să utilizez *Visual Studio Code*, fiind un editor de text cunoscut, cu care am avut tangențe și în cadrul unor materii din facultate. Creat de Microsoft, acesta este destinat în primul rând pentru JavaScript, TypeScript și Node.js, însă prezintă o colecție vastă de extensii pentru alte limbaje de programare, precum C#, C++, Java sau Python.

În contextul scrierii de cod în JavaScript, VS Code se remarcă prin câteva funcționalități avansate, care au rolul de a ghida dezvoltatorii atunci când vine vorba de greșeli de scriere. **IntelliSense** oferă completarea inteligentă a codului, prin ferestre *pop-up* care afișează sugestii chiar în timpul tastării, accelerând astfel procesul prin reducerea greșelilor de sintaxă. Totodată, VS Code pune la dispoziție un *debugger* interactiv, care permite inspectarea variabilelor și executarea de comenzi din consolă.



**Figura 5.1.1. Interfața Visual Studio Code**

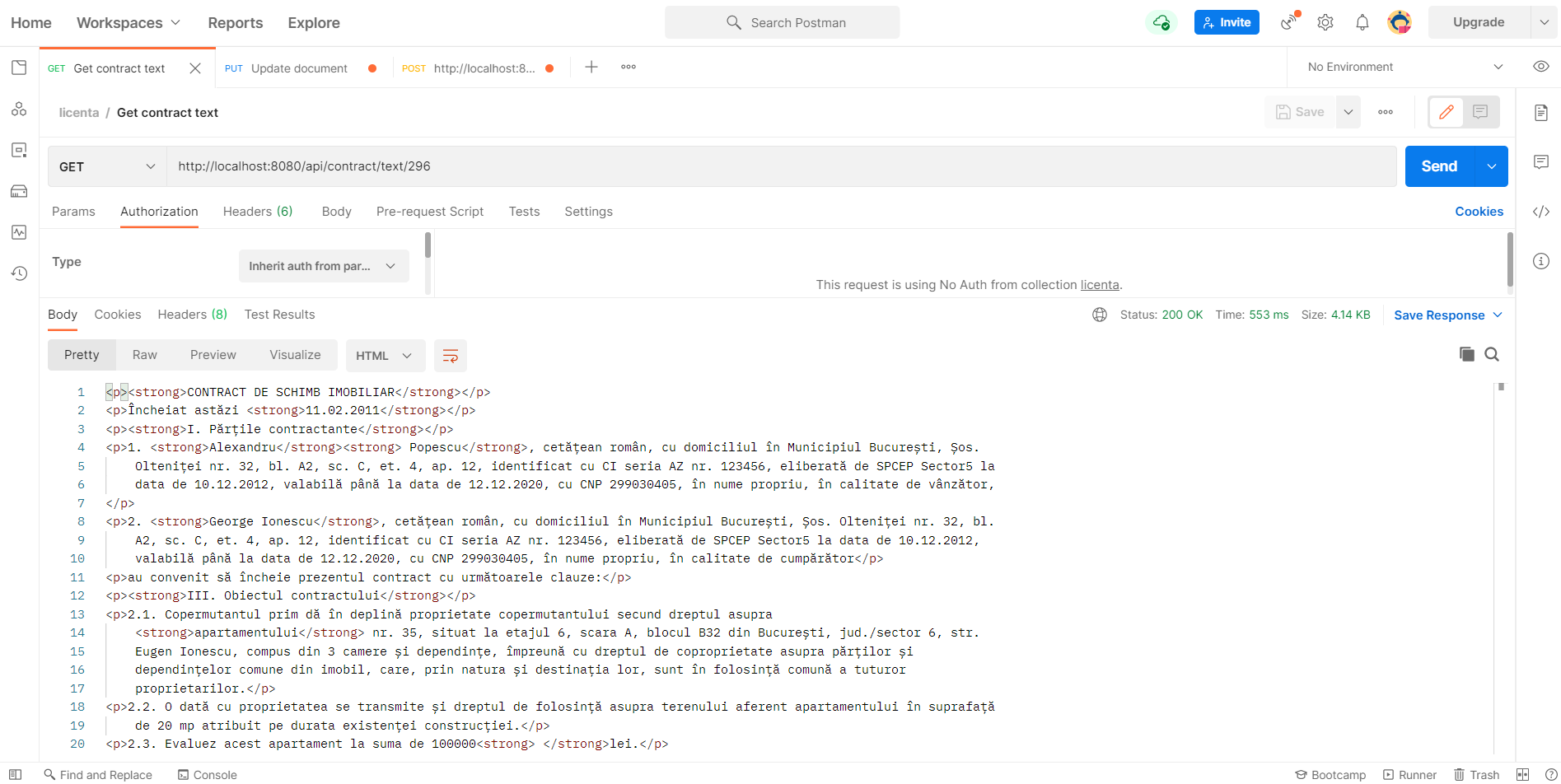
De asemenea, VS Code oferă o **sincronizare facilă cu platforma GitHub**, unde codul sursă poate fi stocat pe măsură ce aplicația este implementată, oferind un istoric al versiunilor acestuia. Cereri de preluare sau încărcare a codului într-un repository pot fi făcute direct din terminal, cu un minim de efort din partea dezvoltatorilor. Astfel, VS Code urmărește să facă fluxurile de lucru mai rapide, atunci când vine vorba de sarcini comune și lucru în echipă.

**2. Postman**

Pentru **testarea funcționalității API-urilor** create am ales să folosesc Postman, fiind cea mai populară platformă de acest tip la momentul actual. Scopul principal al acestui instrument este de a crea solicitări HTTP de diferite tipuri (GET, POST, PUT, DELETE), prin introducere unei rute de către utilizator, și de a afișa răspunsul de la server. Dacă în mod normal, pentru verificarea unui API, dezvoltatorul ar trebui să creeze o funcție în care să folosească ruta definită anterior în care să prelucreze modul în care este afișat răspunsul, folosind această platformă **timpul dedicat testării este redus considerabil**.

Postman vine **sub forma unei aplicații desktop** cu o interfață elegantă, dar care poate să nu fie cea mai intuitivă pentru un începător, datorită multitudinii de funcționalități pe care le pune la dispoziție. În ciuda acestui fapt, după ce te familiarizezi cu platforma, întreg procesul de testare devine rapid.

**Postman acceptă orice fel de solicitare**, având opțiuni pentru setarea de *Headers*, *Body* sau metode de autorizare (*Authorization*) pentru rutele care necesită autentificarea utilizatorului. Ulterior, aplicația permite salvarea oricărei solicitări testate, afișând **parametri de performanță** precum timpul de răspuns sau status codul răspunsului.

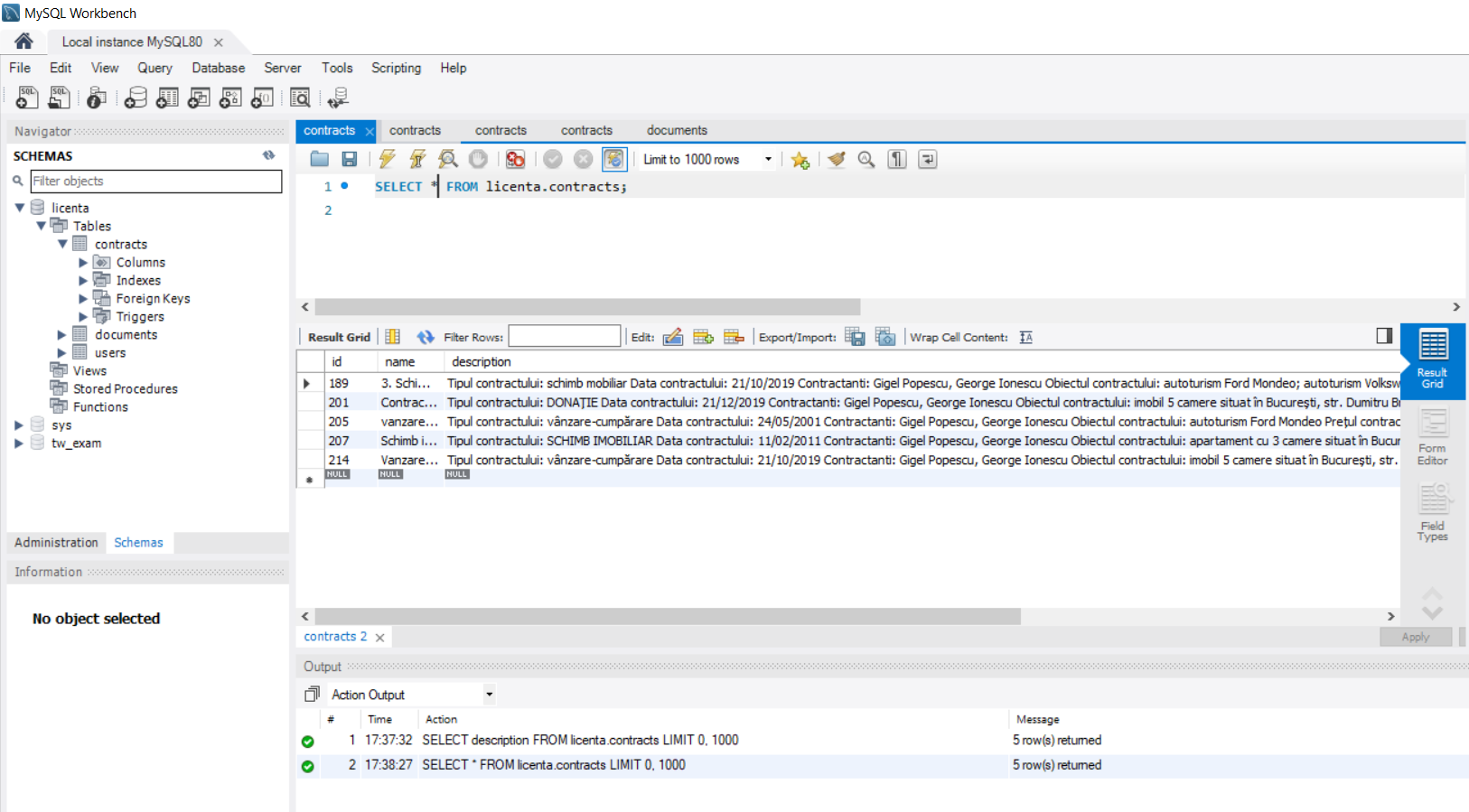


**Figura 5.1.2. Interfața Postman**

În figura de mai sus, poate fi observat răspunsul unui *request* de tip GET, care are rolul de a extrage textul unui document identificat prin *id* în baza de date.

**3. MySQL Workbench**

MySQL pune la dispoziția utilizatorilor un instrument de proiectare a bazelor de date, numit MySQL Workbench. Acesta facilitează lucrul cu bazele de date, integrând dezvoltarea și întreținerea tabelelor într-un singur mediu.



**Figura 5.1.3. Interfața MySQL Workbench**

Așadar, am ales să folosesc această platformă pentru a avea o vizibilitate mai bună asupra întregii baze de date, dar și pentru a face modificări rapide asupra ei.

## 5.2. Dezvoltarea aplicației

Aplicația propusă urmărește în primul rând extragerea de date relevante, care să constituie o sinteză a unor documente contractuale încărcate de utilizator. **Arhitectura** aplicației respectă **modelul specific SPA** (*Single Page Application*), care implică un flux asincron de date între client și server, asociat fiecărei acțiuni a utilizatorului în cadrul platformei. Astfel, principalele framework-uri utilizate sunt **Node.js** și **React.js**.

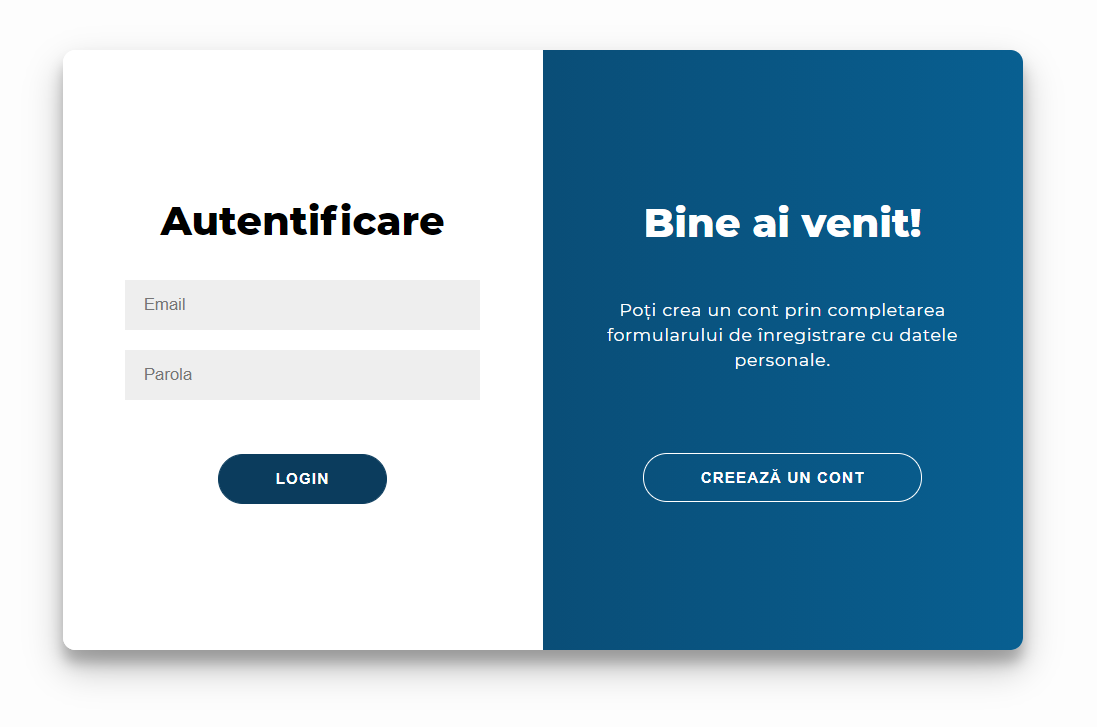
Primul pas în construirea componentei de *backend* a constat în inițializarea aplicației prin executarea comenzii *npm init* în terminal. Fiind vorba de un **serviciu web REST** (*Representational State Transfer*), am descărcat și inclus modulul *express* în lista de dependințe cu ajutorul comenzii *npm install express --save*. În acest moment, având dependințele instalate, am trecut la configurarea fișierului *server.js*: am definit o constantă *app*, iar apoi am specificat portul pe care să ruleze serverul (8080).

În continuare, pentru partea de baze de date, am instalat pachetele *mysql2* și *sequelize* prin comenzile *npm* specifice. După ce am definit câte un model pentru fiecare tabelă din baza de date (*Contracts* și *Users),* am instanțiat un obiect *sequelize* în fișierul *index.js* și am realizat conexiunea la baza de date folosind metoda *authenticate().* În final, am specificat modele folosite și legăturile dintre ele.

În directorul corespunzător componentei de *backend* se găsește și gramatica utilizată pentru parsarea documentelor. Pentru că era vorba de contracte, am ales să analizez fiecare clauză contractuală separat, iar la final am importat gramaticile în fișierul *grammar2.ne*, care descrie întreaga gramatică a unui document.

Toate elementele ce țin de interfața aplicației se găsesc în directorul *frontend*, creat cu ajutorul comenzii *npx create-react-app*, iar pentru inițializarea componentei și afișarea interfeței în *browser* se folosește instrucțiunea *npm start*.

Când utilizatorul accesează aplicația, acesta este redirecționat către un formular de *Login*. Pentru a putea avea acces la funcționalitățile aplicației, autentificarea este obligatorie. Așadar, dacă acesta nu are un cont, are opțiunea de a crea unul prin formularul de *Register*.



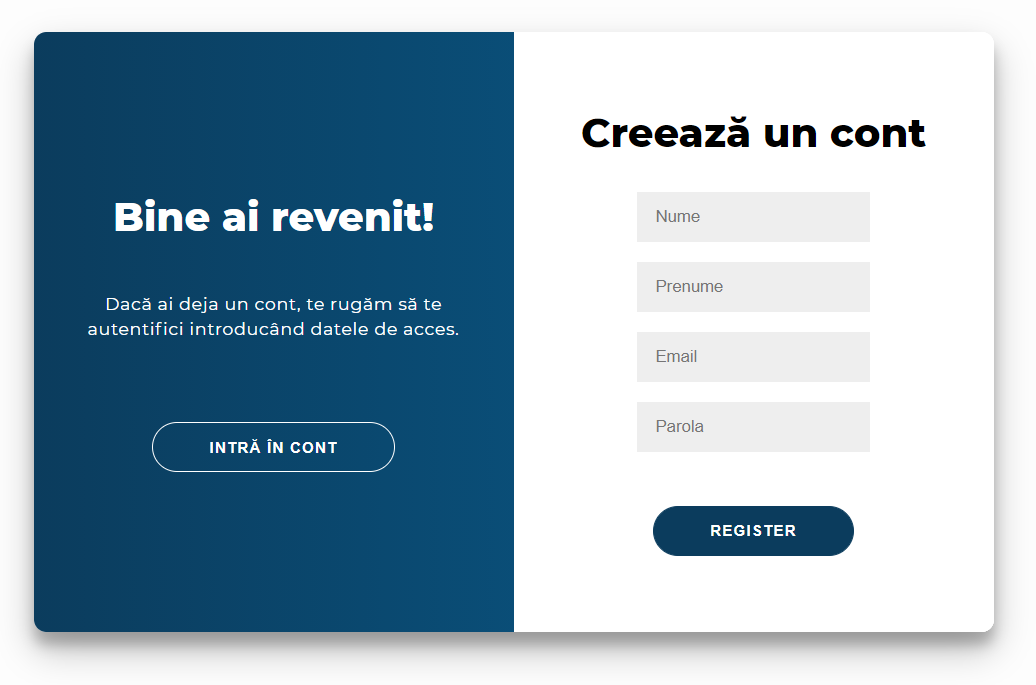
**Figura 5.2.1. Formular de autentificare**

În *backend*, am folosit *Bearer Authentication* pentru autorizarea accesului unui utilizator. User-ul este autentificat prin **JSON Web Tokens**, niște șiruri de caractere criptate automat de server ca răspuns al cererii de *login*. Întregul middleware care face posibilă autentificarea se găsește în fișierul *auth.js*.



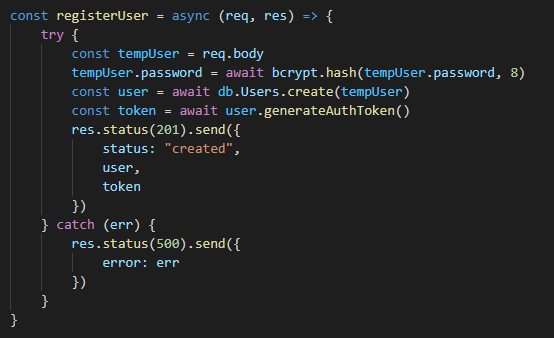
**Figura 5.2.2. JWT Bearer Authentication middleware**

Pentru formularul de *Register*, utilizatorul trebuie să introducă datele personale (nume, prenume, email și parola) pentru crearea unui nou cont. Pentru fiecare câmp al formularului există validări, iar utilizatorul este notificat în cazul în care datele introduse sunt invalide.



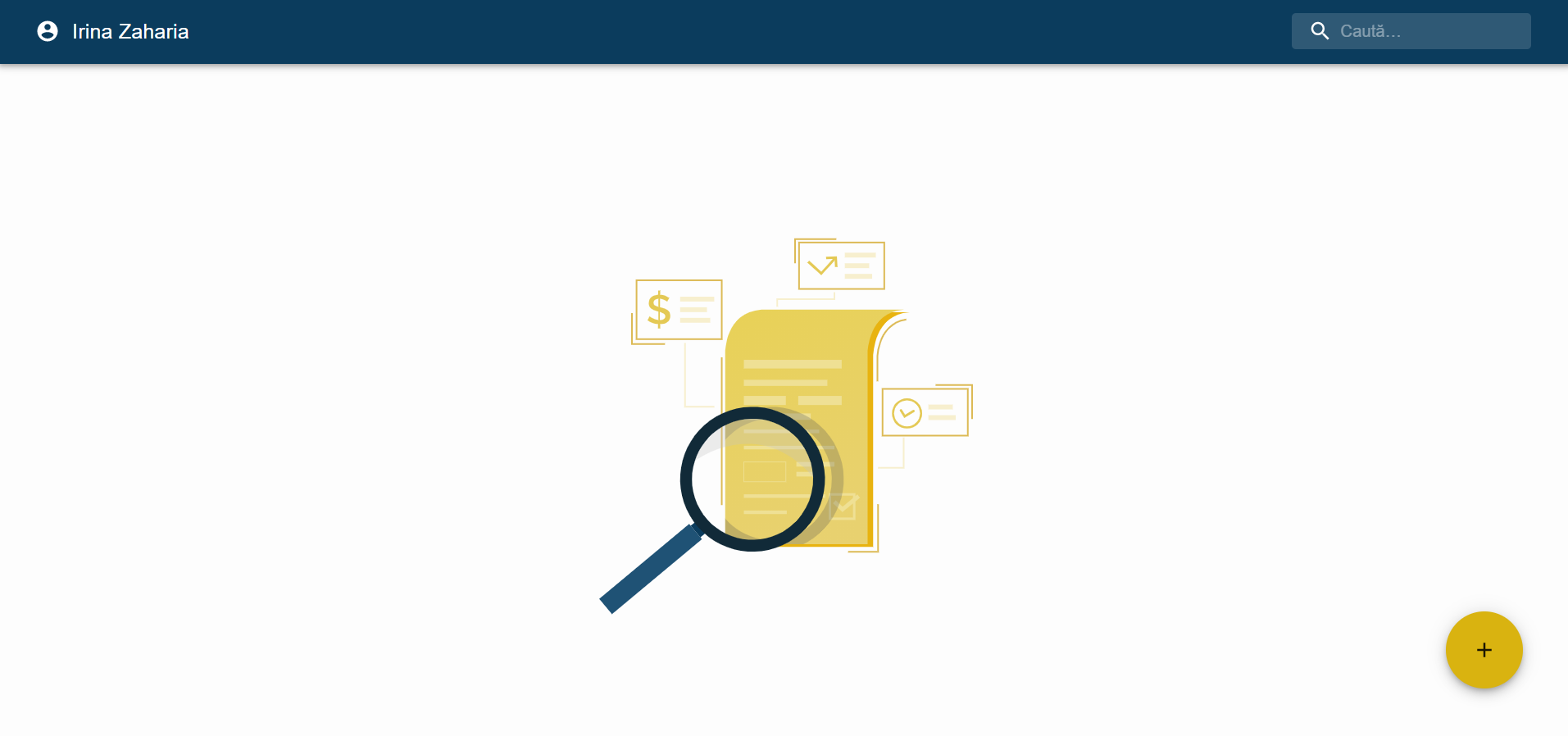
**Figura 5.2.3. Formular de înregistrare**

În momentul creării unui cont, un nou utilizator este adăugat în baza de date. Am ales să criptez parola, prin intermediul librăriei *bcrypt,* pentru a spori gradul de securitate al sistemului implementat.



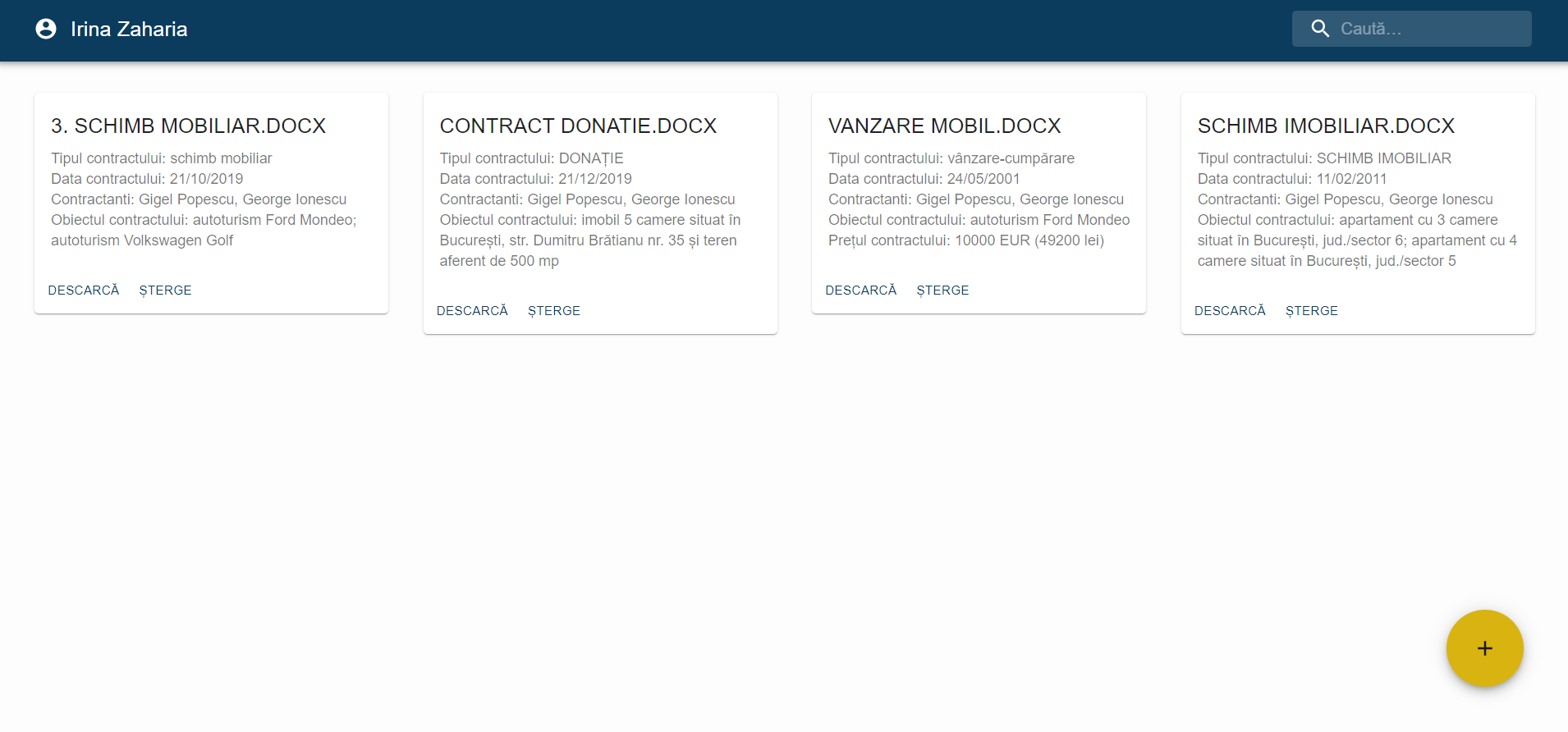
**Figura 5.2.4. Funcția de înregistrare a unui nou cont**

După ce utilizatorul se autentifică, acesta este redirecționat către pagina principală a aplicației. În cazul unui cont nou creat, pagina principală arată ca în figura de mai jos.



**Figura 5.2.5. Pagina principală a aplicației pentru un utilizator nou**

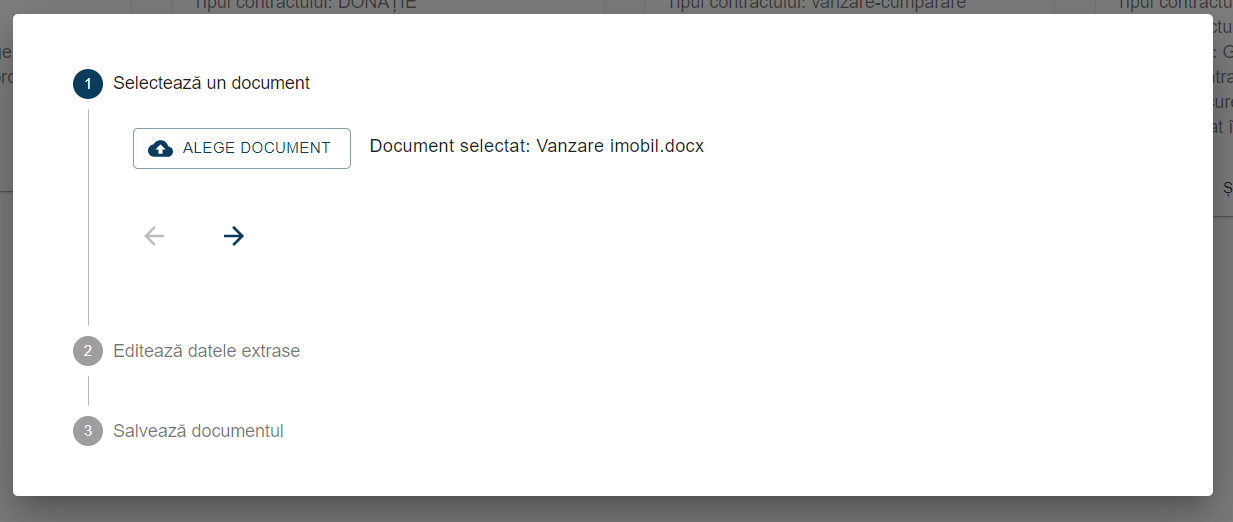
Pentru un utilizator care deja a folosit aplicația, pagina principală afișează întreaga colecție de documente încărcate anterior pe platformă.



**Figura 5.2.6. Pagina principală a aplicației**

Adăugarea unui nou document este posibilă prin apăsarea butonului galben marcat cu semnul “**+**”din dreapta paginii. Acesta reprezintă un *Floating Action Button*, așa că în cazul unui ecran mai mic, va apărea deasupra oricărui alt element din pagină, oferindu-i utilizatorului acces permanent la funcționalitatea de bază a aplicației. Odată apăsat, butonul declanșează deschiderea unui *Dialog* prin care utilizatorul poate parcurge pașii necesari adăugării unui contract în colecție (pași care sunt prezentați în interfață sub forma unui *Stepper*).

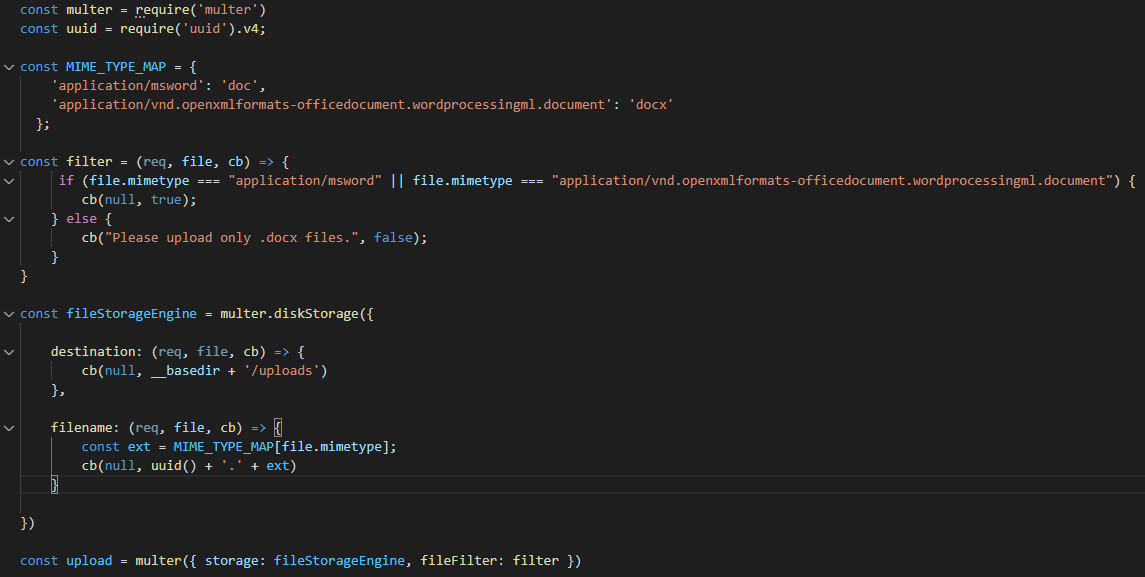
În primul rând, utilizatorul trebuie să aleagă un **document cu extensia .docx** spre sintetizare, iar acest pas este definitoriu. Astfel, butonul care permite navigarea către etapa următoare rămâne inactiv până alegerea unui contract. În momentul în care un document este selectat, numele acestuia apare pe ecran.



**Figura 5.2.7. Selectarea unui document în aplicație**

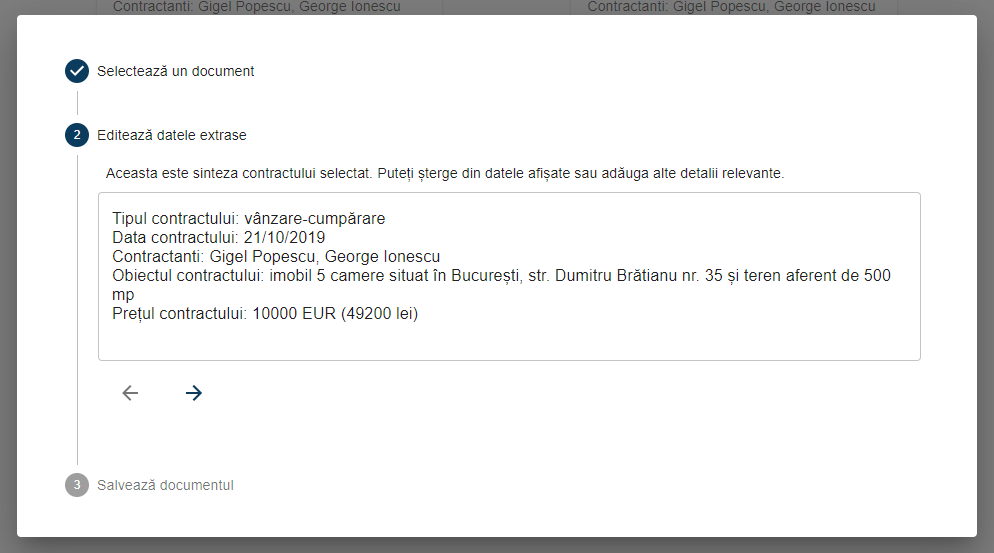
Mecanismul de încărcare de fișiere (care se găsește în *upload.js*) este posibil prin intermediul librăriei ***multer****,* care procesează date de tip ***multipart/form-data***. Pentru a configura modul în care cererea de upload este procesată, am folosit următoarele proprietăți ale bibliotecii:

* *diskStorage*, care oferă **control total asupra stocării fișierelor pe disc** prin cele două opțiuni: *destination* și *filename*. Astfel, am putut selecta directorul *uploads* ca destinație pentru fișierele încărcate, fiecare document primind un nume sub forma unui identificator universal unic (*uuid*).
* *fileFilter*, care controlează tipul de fișiere acceptate. În cazul de față este vorba despre fișiere Word, așa că am pus o condiție ca *mimetype*-ul să respecte formatul specific.

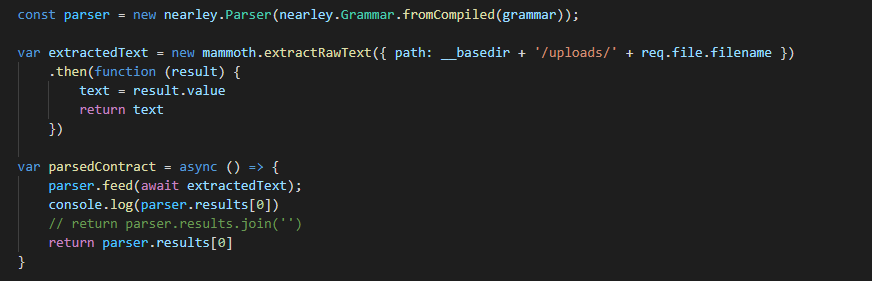


**Figura 5.2.8. Mecanism de upload al fișierelor**

Cel de-al doilea pas în procesul de adăugare a unui document în colecția din cadrul platformei îl reprezintă editarea sintezei unui contract. Utilizatorul poate fie să șteargă sau să adauge detalii în descrierea afișată automat, pe baza documentului selectat anterior.

**Figura 5.2.9. Editarea datelor extrase din document**

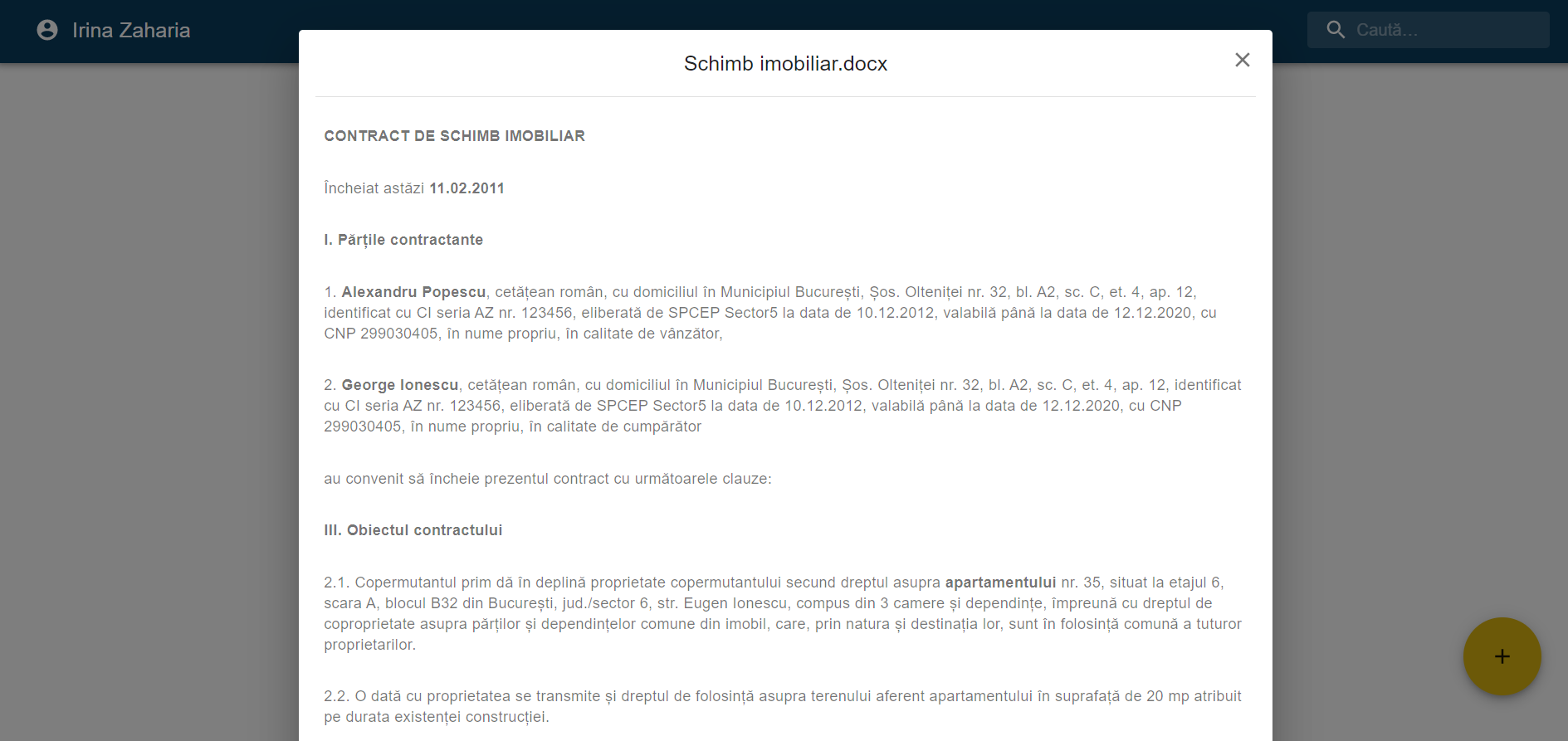
Prelucrarea textului are loc în *backend*, acolo unde am folosit biblioteca *mammoth* pentru a extrage întreg textul din cadrul unui document, iar asupra acestuia am aplicat algoritmul de procesare a gramaticii definit cu ajutorul *nearley.* Astfel, parser-ul returnează exact sinteza ce conține datele importante despre clauzele principale ale contractului încărcat.



**Figura 5.2.10. Prelucrarea textului unui document pe baza gramaticii definite**

Ultimul pas constă în salvarea contractului în colecția afișată în pagina principală. Este important de menționat că utilizatorul are acces doar la contractele încărcate de el, nu și la cele încărcate de alți utilizatori.

Revenind la pagina principală, pentru fiecare document încărcat este afișată sinteza acestuia, iar utilizatorul are posibilitatea de a vedea întreg textul unui contract prin simpla apăsare a componentei corespunzătoare.



**Figura 5.2.11. Accesarea conținutului unui document**

Pentru a extrage textul unui document sub forma unui fișier HTML, am utilizat funcția convertToHTML, pusă la dispoziție de biblioteca mammoth. Aceasta respectă formatarea inițială a fișierului și permite, totodată, modificări ulterioare asupra acesteia.



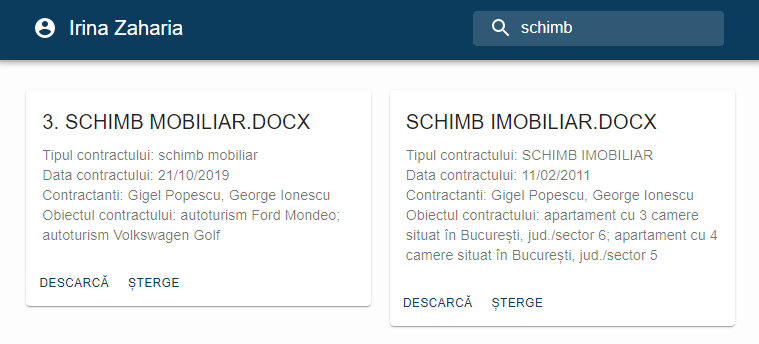
**Figura 5.2.12. Funcția pentru extragerea textului unui document**

De asemenea, componenta ce desemnează un contract pune la dispoziția utilizatorilor două butoane prin care documentul poate fi șters din colecție sau descărcat în *device*-ul personal.



**Figura 5.2.13. Componenta corespunzătoare unui document**

Pentru a facilita accesul la un anumit document din cadrul unei colecții mai numeroase, am adăugat un *search bar* în partea de sus a paginii. Acesta permite filtrarea colecției de documente în funcție de cuvintele cheie introduse de utilizator. Astfel, am urmărit să adaug un plus în ceea ce privește experiența utilizatorului,



**Figura 5.2.14. Filtrarea colecției de documente pe baza unui search bar**

În final, utilizatorul are opțiunea de a ieși din cont prin apăsarea *icon*-ului de lângă numele său.

Pe parcursul dezvoltării componentei de *frontend*, am avut în vedere conturarea unor **interfețe intuitive**, astfel încât experiența utilizatorului în cadrul platformei să fie una optimă.

# Concluzii

În final, această lucrare s-a concretizat prin dezvoltarea unei aplicații web funcționale, care permite sintetizarea conținutului unor documente contractuale – aspect care a constituit obiectivul principal încă de la început. În ciuda faptului că soluția propusă nu atinge performanțele aplicațiilor existente pe piață, aceasta se diferențiază de acestea prin faptul că face posibilă analiza unor **texte redactate în limba română**. Astfel, consider că întregul sistem poate reprezenta un punct de plecare pentru ceea ce înseamnă o digitalizare a proceselor ce aparțin domeniului juridic din România.

Pe viitor, **funcționalitățile aplicației pot fi extinse** în funcție de cerințele pieței. Spre exemplu, gramatica de la bază poate fi dezvoltată astfel încât să suporte mai multe tipuri de contracte, dar și alte documente precum acorduri între părți, decizii juridice sau alte acte care țin de această sferă. O altă idee care poate fi implementată ulterior este aceea de a crea o interacțiune între lucrătorii din domeniu și clienții lor. Practic, sistemul ar putea să permită avocaților sau notarilor să încarce documente, iar contractanții ar putea avea de asemenea acces pentru vizualizarea lor în aplicație. Nu în ultimul rând, un aspect ce poate fi luat în calcul pentru dezvoltarea viitoare a aplicației este modificarea accesului la documente, astfel încât transmiterea de documente între angajații unei firme să fie cât mai rapidă. De asemenea, poate fi creată și o colecție comună de fișiere, la care toți angajații să aibă acces. Prin funcționalitatea de bază, dar și prin aceste adăugări ulterioare, sunt de părere că aplicația poate avea un **impact major asupra productivității la locul de muncă,** prin automatizarea unor sarcini costisitoare din punct de vedere al timpului.

În ceea ce privește conceptul general din spatele soluției propuse, și anume acela de **procesare a limbajului natural,** a fost necesar un studiu aprofundat al aspectelor ce țin de **ligvistica computațională,** domeniu cu care nu eram familiarizată înainte de dezvoltarea acestui proiect. Citind articole atât despre începuturile NLP, cât și despre diferitele moduri în care este integrat în aplicații în ziua de astăzi, am ajuns la concluzia că acesta este în plină ascensiune, dezvoltându-se constant și fiind tot mai folosit, din moment ce nevoia de analiză de text este tot mai mare.

Așadar, utilizând principii caracteristice NLP, am reușit să ating obiectivul lucrării prin dezvoltarea unei aplicații web care să aducă un element de noutate în domeniul juridic din România, constând în funcționalitatea principală – abstractizarea textului unui document contractual prin extragerea informațiilor esențiale.