UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: [……]
SPECIALIZAREA: [……]

[Titlul lucrării]

LUCRARE DE DIPLOMĂ

Coordonator științific
[Titlul academic Prenume Nume]

Absolvent
[Prenume Nume]

Iași, 2022

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII LUCRĂRII DE DIPLOMĂ

Subsemnatul(a) _____,
legitimat(ă) cu _____ seria _____ nr. _____, CNP _____
autorul lucrării _____
_____
_____
elaborată în vederea susţinerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică şi Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iaşi, sesiunea _____ a anului universitar _____, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data Semnătura

_____

Cuprins

Rezumat

Un loc de muncă potrivit joacă un rol foarte esential în viața fiecărei persoane. O meserie si un mediu de lucru adecvat ne poate influenta in mod direct fericirea, indentitatea, statutul social, relatiile, programul de viata, ritmul si nu numai,

Din nefericire, efectul pandemiei Covid-19 a fost devastator si a procovat consecinte grave in ceea ce priveste domeniul angajarilor. Potrivit CMIE, aproximativ 27 de milioane de oameni si-au pierdut locul de munca in timpul desfasurarii acestui focar, somajul in masa facandu-si simtita prezenta in zona tinerilor cu varste cuprinse intre 15 si 29 de ani.

Pentru a contribui la sprijinirea efortului post-pandemic de reintregrare a oamenilor pe piata muncii, companiile publica constant noi oferte si oportunitati de angajare pe portalurile de job-uri, devenite principale medii si canale de comunicare dintre angajatori si angajati, imbunatatind si accelerand procesul de recrutare.

Platforme precum *https://www.linkedin.com/*, *https://www.ejobs.ro/*, *https://www.hipo.ro/*, au oferit serviciile lor romanilor de ani buni si au economisit timp si bani atat pentru solicitantii locurilor de munca cat si pentru organizatiile care doresc sa angajeze oameni. Cu toate acestea, tehnicile traditionale de gasire a informatiilor legate de slujbe incep cu timpul sa nu mai fie adecvate pentru utilizatorii acestor aplicatii. Numarul de rezultate returnate de platformele enuntate mai sus poate fi imens, astfel incat potentialii clienti sunt nevoiti sa petreaca o cantitate semnificativa de timp citind si analizand optiunile care le sunt prezentate. O abordare ce eficientizeaza solutionarea acestui impediment este crearea unor sisteme de recomandare.

În această lucrare, s-a depus un efort pentru a propune un sistem personalizat de potrivire a locurilor de muncă cu CV-urile potentialilor utilizatori, ce urmareste simplificarea procesului de gasire a unui loc de munca la nivel national.

Capitolul I

Capitolul II

Capitolul III

Capitolul IV

Capitolul V

Introducere

Descrierea problemei

In prezent principalele platforme care oferteaza locuri de munca in Romania sunt**:** ***https://www.linkedin.com/***, ***https://www.ejobs.ro/***, ***https://www.hipo.ro/***, ***https://www.undelucram.ro/***, ***https://www.romjob.ro/*** s.a.m.d. Scopul principal al acestora este sa faciliteze procesul de cautare a unei slujbe si sa diminueze timpul de recrutare. Însă, majoritatea acestor aplicații permit utilizatorilor doar sa introduca cuvinte cheie pentru cauta și a filtra locurile de munca, făcând căutarea unei oferte sa ajungă o munca anosta, repetitiva și obositoare(de ex pentru cuvântul cheie „**C++**” introdus în bara de căutare al site-ului ***eJobs*** am obținut 1266 de oferte). Numărul rezultat în urma cautarii este mare, dar nu este bine clasificat, iar ca și consecinta, clientul aplicației trebuie sa analizeze descrieriile fiecarui loc de munca, fapt ce nu poate fi pus în practica într-un timp util, așadar calitatea serviciului de gasire a slujbelor este scazuta și imprecisa. Majoritatea platformelor de acest gen folosesc indecsi inversi pentru a extrage înformatiile și pentru optimizarea cautarilor. Motoarele moderne de căutare folosesc algoritmi de clasare pentru sortarea rezultatelor precum Page Rank(dezvoltat de Google), dar aceștia nu ar funcționa într-un sistem cum este cel de căutare al locurilor de munca, deoarece criteriile de clasificare ar unei potențiale oferte ar trebui să fie foarte personalizate. Spre exemplu un post de munca vacant poate reprezenta o optiune excelentă pentru un utilizator, dar să fie nepotrivitia pentru alt client, deoarece aceștia au făcut facultati cu profiluri neasemanatoare, au o alte pregătire și educatie profesională, provin din medii și orase diferite etc. Așadar, s-a incercat și s-a cercetat o posibila soluție de potrivire intre o oferta de munca și un utilizator.

Pentru a realiza o căutare personalizata pentru fiecare client în parte, s-au cautat informații relevante și elocvente despre utilizator, cu ajutorul cărora am putea realiza un proces de atribuire a posturilor vacanțe. Deoarece CV-urile contin în fond cele mai importante date despre trecutul profesional al unui om, am considerat continutul acestora ca fiind o variabila potrivitra în cadrul procesului de clasificare. În aceasta lucrare, am creat un sistem web care utilizeaza datele personale ale utilizatorilor și descrierile locurilor de munca pentru a calcularea similitudinii și compatibitatii dintre om – oferta. S-a urmărit transferarea sarcinii de căutare a posturilor vacanțe prin efortul manual al clientilor(utilizand cuvinte cheie sau filtre vizuale) la generarea unor rezultate recomadate de către motorul de clasificare. În urma acestui proces, fiecare job primește un scor, iar în funcție de cât de mare este acesta, se decide cât de potrivit este un loc de munca pentru un utilizator și acorda o prioritate anumitor posturi.

Scopul final al aplicației este automatizarea și simplificarea gasirii unor rezultate relevante, prin adunarea și clasificarea locurilor de munca, pe baza CV-ului unui utilizator.

Abordari anterioare

Zheng et al. [44] și AlOtaibi et al. [3] au sintetizat categoriile de platforme de recrutare online existente și au enumerat avantajele și dezavantajele tehnicilor de recomandare folosite:

1. Recomandarea bazată pe conținut (CBR). Principiul unei recomandări bazate pe conținut este acela de a sugera elemente care au informații de conținut similare cu cele ale utilizatorilor corespunzători, precum Prospect [43].

2. Recomandarea prin filtrare colaborativă (CFR). Recomandarea prin filtrare colaborativă găsește utilizatorii similari care au aceleași gusturi cu utilizatorul țintă și recomandă articole pe baza a ceea ce au făcut utilizatorii similari, cum ar fi CASPER [35].

3. Recomandarea bazată pe cunoaștere (KBR). În recomandarea bazată pe cunoaștere, regulile și modelele obținute din cunoștințele funcționale ale modului în care un anumit articol îndeplinește cerințele unui anumit utilizator, sunt utilizate pentru a recomanda articole, cum ar fi Proactive [24].

4. Sistemele de recomandare hibride combină două sau mai multe tehnici de recomandare pentru a obține performanțe mai bune și pentru a depăși dezavantajele oricărei tehnici individuale.

Totuși, există câteva probleme în sistemele anterioare de recomandare a locurilor de muncă:

1. Cele mai multe sisteme pot procesa doar datele structurate ale CV-urilor și a descrierilor locurilor de muncă, dar în realitate ambele sunt în formate nestructurate, cum ar fi fișiere text sau pagini web HTML.

2. Sistemele care au module de extragere a informațiilor sunt concepute ca și recrutorii să selecteze candidații, nu pentru persoanele aflate în căutarea unui loc de muncă care vor sa găsească cea mai potrivita oferta.

3. În cadrul sistemelor enuntate, campurile după care se realizeaza potrivirea dintre CV și descrierile locurilor de munca sunt de o granulație grosieră. Pentru a îmbunătăți calitatea recomandării, trebuie să îmbunătățim granularitatea informațiilor care se extrag.

Capitolul 1. Fundamentarea teoretică.

1.1. Abordarea teoretica prezenta în lucrare

Scopul și problema de bază care intervine este potrivirea și asignarea adecvată a unei oferte de munca cu CV-ul unui utilizator, fapt ce induce necesitatea gasirii și calcularii unui factor de similitudine dintre cele doua variabile. Dacă gândim problema sub forma unui sistem de extragere de informatii(eng: Information Retrieval System), acesta va fi alcătuit din:

1. D – Reprezentarea documentelor, în cazul de fata mulțimea va fi alcatuita din locurile de munca
2. Q – Reprezentarea interogarii, în cazul nostru o putem vedea ca și CV-ului utilizatorului
3. F – un sistem de potrivire a unei relații intre D și Q, acesta fiind reprezentat sistemul care compara toate ofertele de munca cu fiecare utilizator
4. R(Q, Di) – O funcție de clasare care determina asemanarea dintre interogare și document pentru a găsi cel mai buna potrivire; în cazul de fata functia care genereaza scorul.

Jobify indexeaza multiple platforme pentru a găsi locuri de munca cu informații relevante, le analizeaza și extrage descrierile aferente și stocheaza în baza de date. Atunci când un nou utilizator apeleaza la sistemul de recomandare, se compara similtudinea dintre cele doua modele existente(descrierea ofertei și CV-ul) și ulterior se sorteaza toate posturile vacanțe existente pe baza acestuia.

Pentru a oferi o descriere formala problemei, vom folosi ***cv*** pentru a desemna modelul rezultat în urma procesarii CV-ului utilizatorului(care ar trebui sa aibă caracteristici precum nivelul actual dar și trecutul sau academic, calificarea și specializarea, compotentele și abilitățile sale Soft & Hard, diferite certificari obtinute în trecut s.a.m.d). Simbolul ***J*** reprezintă o multimea alcatuita din modelele de locuri de munca(rezultate în urma algoritmului de filtrare și curatare), iar ***j*** este un element din aceasta. Functia de similaritate *sim(cv, j)* confera rezultatul de similaritate dintre cele doua instante. În urma aplicarii acestui proces, vom obtine o lista cu ofertele de munca clasificate și sortate pe baza scorului obținut. Ecuația de calculare a similitudinii se regaseste mai jos:

$$\mathrm{sim}(cv, j) = \sum_{i=1}^{n} \mathrm{simfun}_i(cv_i, j_i) \times w_i$$

Unde valoarea lui sim(cv, j) este suma rezultatelor de similaritate ale diferitelor câmpuri de comparație înmulțit cu ponderile lor corespunzătoare. Diferite domenii precum specializarea, competențele, orașul de provenienta sau indecsi analitici de comparație pot avea funcții diferite pentru a calcula valorile de similaritate.

1.2. Provocari

În cadrul etapei de proiectare a sistemului, au apărut diferite complicatii. Una dintrele este reprezentata de dificultatea extragerilor de informații referitoare la oferte de munca, la ora actuala neexistand API-uri pentru posturile vacanțe din Romania. Pentru a solutiona acest aspect, s-a recurs la o metoda de indexare a website-urilor de pe piața romana, extragerea datelor și centralizarea acestora într-o baza de date comuna, aceasta fiind o etapa necesara pentru pregatirea modelarii algoritmului.

Pentru posibilitatea calcularii unui scor de similitudine dintre un job și un CV al unui utilizator, sistemele de recomandare necesita în general date sub forma structurata. Pentru a depasi aceasta incovenienta, motoarele existe pe piața solicita persoanelor aflate în căutare de locuri de munca și recrutorilor sa completeze formulare, câmp cu câmp(ex:educatie academica anterioara, experiența profesională, certificari, oraș de provenienta, interese, așteptări, soft

skills/hard skills, astepatari de la un posibil viitor angajat) . Totuși, majoritatea utilizatorilor sunt reticenti în a finaliza acest proces lung și anevoios, aceştia preferand sa își incarce în mod direct CV-urile în diferite formate(pdf, doc, txt etc.) fără a mai completă informații suplimentare. Așadar, dificultatea intervine la extragerea unor date structurate din surse de informații nestructurate, care pot fi folosite mai departe în procesul de clasare. Într-o prima faza de proiectare s-a încercat găsirea și aflarea unei liste ample de expresii, termeni tehnici și profesionali dar și informali(se așteaptă regasirea pasiunilor și a descrierilor personale într-un CV), fraze, cuvinte cheie cu conotatie relevanta pentru slujbele salvate, a carei intersectie cu descrierile locurilor de munca sa confere un rezultat suficient de precis pentru modelare.

A treia provocare este reprezentata de găsirea unei formule potrivite de calculare a similitudinii dintre modelele care ar trebui obtinute din etapa anterioara. S-a constatat ca simpla egalizare a cuvintelor cheie, sau termenii comuni nu sunt o metrica de comparație suficient de buna dintre cele doua modele, rezultatele fiind imprecise(un cv bogat în exprimare și cuvinte, cu o experiența generala și nespecializata pe diferite arii profesionale va fi mereu un rezultat bun pentru o descriere succinta, generica, în care nu se specifica cuvinte cheie și nici un scop sau o cerinta bine definita a angajarii pentru o munca posibil necalificata și posibil chiar un domeniu de activitate diferit). Un exemplu se regaseste în tabelul de mai jos:

Tabelul 1.1: Portiuni din descrierea unei slujbe si a unui CV

Porțiunea din CV	**Porțiunea din descrierea slujbei**
Cele mai mari calități ale mele sunt seriozitatea și perseverența. În cariera mea de operator producție, am ales întotdeauna să îmi asum responsabilitatea și să respect cerințele venite de la superiorii mei. Sunt o persoană care se adaptează foarte repede în colective noi, îmi place munca în echipă și întotdeauna dau tot ce pot pentru a îndeplini obiectivele companiei.	Compania de transport angajeaza Sofer profesionist C E pentru transport international (Spania-Franta-Belgia-Olanda-Italia), cu experienta in transport cu camioane frigorifice si box. Obiectivul nostru este o persoana serioasa, ambitioasa, perseverenta, adaptabila la condiții și medii noi de munca. Contractele de angajare pot fi realizate și pe un contract de tip PFA, persoana în cauza neavand superiori(este propriul angajat).

1.3 Tehnologii utilizate

Lucrarea de licență a fost dezvoltată cu ajutorul unei suite de tehnologii contemporane, iar alegerea lor a fost făcută cu intenția unei simbioze intre limbajele folosite, dar și pentru optimizarea performantelor sistemului și facilizarea scrierii codului. În cadrul proiectului meu se disting doua mari componente, fiecare dintre ele având și operand cu functionalitati diferite.

1.3.1 Node.JS

Partea de backend a fost realizata cu ajutorul utilitarului open-source ***Node.js.*** Denumit în mod obișnuit Node, este un motor de rulare JavaScript care permite rularea codului Js în afara unui browser web. Acesta se bazează pe motorul JavaScript Google V8, același folosit pentru procesarea codului de tip js în Chrome. Cu toate avantajele sale, Node.js joaca acum un rol critic în stiva tehnologica a multor companii de profil inalt, care depind de beneficiile sale unice. Node.js permite crearea de servere web și instrumente de rețea printr-o colecție de module care gestioneaza diverse functionalitati de baza. Sunt prevazute module precum Sistemul de fisire I/O, de rețea(DNS, HTTP, TCP, TLS, SSL sau UDP), binar, criptografice ș.a.md. Node.js aduce programarea bazata pe evenimente la serverele web, permitand dezvoltarea de aplicații scrie în JavaScript foarte rapide. Dezvoltatorii pot crea servere scalabile prin utilizarea unui model simplificat de programare bazata pe evenimente care folosește apeluri de sistem pentru a semnaliza finalizarea unei sarcini. Exista mii de bibloteci open-source pentru Node.js, majoritatea gazduite pe site-ul web a lui npm. Npm este managerul de pachete preinstalat pentru platforma server Node.js, ce usureaza achiziționarea de module și pachete noi și relevante pentru programatori.

Node.js este similar ca design și este influențat de sisteme precum Ruby's Event Machine și Python's Twisted. Node.js duce modelul evenimentului puțin mai departe. Prezinta o bucla de evenimente ca o construcție de rulare și nu ca o biblioteca. În alte sisteme, exista întotdeauna un apel de blocare pentrun a porni bucla de evenimente. De obicei, comportamentul este definit prin apeluri inverse la începutul unui script, iar la sfârșit un server este pornit printr-un apel de blocare precum *EventMachine:: run()*. În Node.js, nu este exista un astfel de apel de start-the event-loop. Node.js intra pur și simplu în bucla de evenimente după executarea scriptului de intrare. Node.js iese din bucla de evenimente când nu mai sunt apeluri inverse de efectuat. Acest comportament este ca JavaScript de browser – bucla de evenimente este ascunsă utilizatorului.

HTTP este un cetatean de prima clasa în Node.js, conceput având în vedere fluxul și latenta scazuta. Acest lucru face ca Node.js să fie bine potrivit pentru fundatia unuei biblioteci web sau a unui cadru.

Node.js fiind proiectat fără fire de execuție, nu înseamnă ca nu poate folosi mai multe nuclee de execuție.m Procesele de calcul secundare pot fi generate folosind *child_process.fork()*.

În sfârșit, utilizatorii node nu sunt îngrijorați de blocarea procesului, deoarece nu exista posibilitatea de a se bloca, în caz ca nu se dorește în mod explicit. Aproape nicio funcție din Node.js nu realizeaza direct I/O, așa ca procesul nu se blochează niciodată, cu excepția cazului în care I/O este efectuat folosind metode sincrone ale bibliotecii standard Node.js. Deoarece nimic nu se blochează, sistemele scalabile sunt foarte rezonabil de dezvoltat în Node.js.

1.3.2 Express.js

Express este un cadru de aplicații web pentru Node.js. Ofera diverse caracteristici care fac dezvoltarea aplicatiilor web rapidă și ușoară, ceea ce altfel, durează mai mult timp folosind doar Node.js.

Express.js se bazează pe modulul middleware Node.js numit connect care, la rândul sau, utilizeaza modulul http. Deci, orice middleware care se bazează pe conectare va funcționa și cu Express.js.

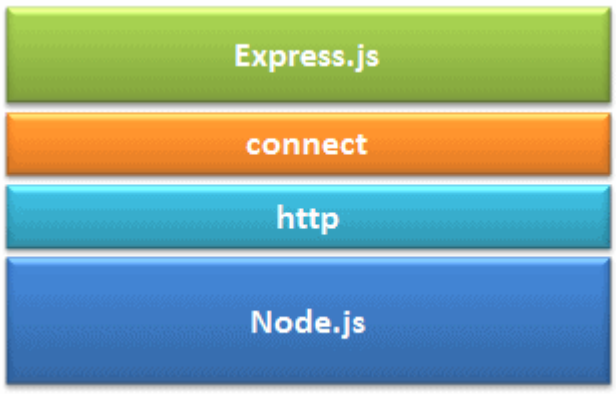


Figura 1.1 Express.js [1]

Avantajele Express.js sunt: usurarea dezvoltarii aplicației web Node.js, usurarea configurarii și personalizarea, permiterea definirii rutelor aplicației pe baza metodelor HTTP și a adreselor URL, includerea diverselor module middleware care se pot utiliza pentru a efectua sarcini suplimentare la cerere și răspuns, usurarea integrarii cu diferite motoare de sabloane precum Jade, Vash, permiterea definirii unui middleware de gestionare a erorilor, permiterea crearii unui REST API, facilizarea și usurarea conectarii la o baza de date precum MongoDB, Redis, MySQL etc.

1.3.3 Axios

Efectuarea de solicitari HTTP pentru preluarea sau salvarea datelor este una dintre cele mai frecvente sarcini pe care trebuie sa le facă o aplicație JavaScript la nivelul clientului. Bibliotecile terte – în special jQuery – au fost mult timp o modalitate populara de a interactiona cu API-urile de browser mai detaliate și de a elimina orice diferente intre browsere.

Axios este un client HTTP bazat pe promisiuni node.js pentru browser. Este izomorf (= *poate rula in browser și nodejs cu aceeași baza de cod*). Pe partea de server folosește modulul nativ de http pentru node.js, în timp ce pe client (browser) folosește XMLHttpRequests.

Caracteristici: XMLHttpRequests direct din browser, solictari de tip http de la node.js (*server-side*), accepta api-ul *Promise,* intercepteaza cererea și răspunsul, transforma datele de solicitare și răspuns, anulează cererile, conversie automata de date în formatul *JSON,* suport pe partea de client pentru protejarea atacurilor de tip XSRF.

1.3.4 Cheerio

Cheerio implementează un subset de jQuerry de baza. Acesta elimina toate incosecventele DOM și crutch browser din bilbioteca jQuerry, dezvăluind un API ușor de manevrat și folosit.

Cheerio analizează marcajul HTML și ofera un API pentru parcurgerea și manipularea structurii de date rezultate. Acesta nu interpreteaza rezultatul așa cum o face un browser web. Mai exact, nu produce o randare vizuala, nu aplica CSS, nu încarca resurse externe și nici nu executa cod JavaScript, care este obișnuit pentru o aplicație de tip SPA (a*plicatie cu o singura pagina*). Acest lucru face Cheerio mult, mult mai rapid decât alte soluții.

1.3.5 Mongoose

Mongoose este o biblioteca ODM (Object Data Modeling) bazata pe Node.js pentru MongoDB. Este asemănător cu un Object Relational Mapper (ORM), cum ar fi SQLAlchemy pentru bazele de date SQL traditionale. Problema pe care Mongoose își propune sa o rezolve este sa le permita dezvoltatorilor sa impuna o anumita schema la nivelul aplicației. Pe lângă aplicarea unei scheme, Mongoose ofera, de asemenea, o varietate de carlige, validare a modelului și alte caracteristici menite sa faciliteze lucrul cu MongoDB.

Validarea schemei MongoDB face posibila aplicarea cu ușurința a unei scheme peste baza de date mongoDB, mentinand în același timp un grad ridicat de flexibilitate.

1.3.6. ReactJS

React (cunoscut ca și React.js sau React.js) este o biblioteca JavaScript gratuita și open-source pentru construirea de interfete cu utilizatorul, bazate pe componente de UI. Este intretinut în prezent de Meta și de o comunitate de dezvoltatori individuali și companii.

React poate fi folosit ca baza în dezvoltarea de aplicații cu o singura pagina, mobile sau randate pe server cu cadre precum Next.js. Cu toate acestea, React este preocupat doar de gestionarea starilor și de redarea lor către DOM, astfel încât crearea de aplicații React necesita de obicei utilizarea de biblioteci suplimentare pentru rutare, precum și anumite functionalitati la nivelul clientului.

1.3.7. Componente

Codul React este format din entitati numite componente. Aceste componente sunt reutilizabile și trebuie formate în folderul src alplicatiei urmând conventia CamelCase de denumire. Componentele pot fi randate la un anumit nivel din DOM folosind biblioteca React DOM.

O alta caracteristica notabila este utilizarea unui model de obiect de document virtual sau DOM virtual. React creeaza un cache pentru structura de date în memorie, calculează diferențele rezultate și apoi actualizeaza eficient DOM-ul afisat în browser. Acest proces se numește reconciliere. Acest lucru permite programatorului sa scrie cod ca și cum întreaga pagina este randata la fiecare modificare, în timp ce biblioteca React randeaza doar subcomponentele care se schimba efectiv. Aceasta randare selectiva ofera o creștere majora a performantei. Se economiseste timp, deoarece nu se mai recalculeaza stilul CSS, aspectul și randarea pentru întreaga pagina.

1.3.8. Metodele ciclului de viata

Metodele ciclului de viața pentru componentele bazate pe clasa folosesc o forma de agatare care permite executarea codului la punctele de referinta pe durata de viața a unei componente.

ShouldComponentUpdate permite dezvoltatorului să prevină redarea inutilă a unei componente, returnând false dacă nu este necesară o randare.

componentDidMount apelată odată ce componenta s-a „montat” (componenta a fost creată în interfața cu utilizatorul, adesea prin asocierea cu un nod DOM. Acesta este folosit în mod obișnuit pentru a declanșa încărcarea datelor de la o sursă la distanță printr-un API.

ComponentWillUnmount este apelata imediat înainte ca și componenta să fie daramata sau demontata. Aceasta este folosită în mod obișnuit pentru a sterge dependetele care necesita resurse de componenta care nu vor fi eliminate pur și simplu odată cu demontarea componentei(de exemplu, eliminarea oricaror setInterval() instante care sunt legate de componenta sau un eventListener setat pe document)

render este cea mai importanța metoda a ciclului de viața și singura necesara în orice componenta. Este de obicei apelata de fiecare data când starea componentei este actualizata, ceea ce ar trebui să se reflecte în interfata cu utilizatorul.

1.3.9. JSX

JSX, sau JavaScript Syntax Extension, este o extensie a sintaxei limbajului JavaScript. Similar ca aspect cu HTML, JSX oferă o modalitate de a structura randarea componentelor folosind sintaxa familiară multor dezvoltatori. Componentele React sunt scrise de obicei folosind JSX, deși nu trebuie să fie (componentele pot fi scrise și în JavaScript pur). JSX este similar cu o altă sintaxă de extensie creată de Facebook pentru PHP numită XHP.

1.3.10. Carlige

Cârligele sunt funcții care permit dezvoltatorilor să se „prindă” în funcțiile de stare React și de ciclul de viață din componentele funcției. Hook-urile nu funcționează în cadrul claselor - vă permit să utilizați React fără clase.
React oferă câteva cârlige încorporate precum: useState, useContext, useReducer. Altele sunt documentate în Hooks API Reference. Cele mai frecvent utilizate, sunt pentru controlul stării și respectiv a efectelor secundare: useMemo, useEffect, useState.

Există reguli pentru cârlige care descriu modelul de cod caracteristic pe care se bazează cârligele. Este modalitatea modernă de a gestiona starea cu React.

1. Hook-urile ar trebui să fie apelate numai la nivelul superior (nu în interiorul buclelor sau declarațiilor if).
2. Cârligele ar trebui să fie apelate numai din componentele funcției React și din hook-urile personalizate, nu din funcții normale sau componente de clasă.

Deși aceste reguli nu pot fi aplicate în timpul execuției, instrumentele de analiză a codului, cum ar fi linters, pot fi configurate pentru a detecta multe greșeli în timpul dezvoltării. Regulile se aplică atât utilizării cârligelor, cât și implementării de cârlige personalizate, care pot numi și alte cârlige.

1.3.11 Python

Python a devenit unul dintre cele mai populare limbaje de programare din lume în ultimii ani. Este folosit în orice, de la învățarea automată la construirea de site-uri web și testarea software-ului. Poate fi folosit atât de dezvoltatori, cât și de non-dezvoltatori.

Cu ajutorul lui python s-a creat de la algoritmul de recomandare Netflix până la software-ul care controlează mașinile cu conducere autonomă. Python este un limbaj de uz general, ceea ce înseamnă că este conceput pentru a fi utilizat într-o gamă largă de aplicații, inclusiv știința datelor, dezvoltarea software și web, automatizare și în general realizarea de tools.

În general python este folosit pentru a construi site-uri web și software, pentru a automatiza sarcini și pentru a efectua analize de date. Python este un limbaj de uz general, ceea ce înseamnă că poate fi folosit pentru a crea o varietate de programe diferite și nu este specializat pentru probleme specifice. Această versatilitate, împreună cu ușurința pentru începători, l-au făcut unul dintre cele mai utilizate limbaje de programare astăzi. Un sondaj realizat de firma de analiză a industriei RedMonk a constatat că acesta a fost al doilea cel mai popular limbaj de programare printre dezvoltatori în 2021.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul *object-oriented*, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu „gunoier” (*garbage collector*). Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.
Implementarea de referință a Python este scrisă în *C* și poartă deci numele de *CPython*. Această implementare este software liber și este administrată de fundația *Python Software Foundation*.

1.3.12 Flask

Flask este un micro-framework web scris în Python. Este clasificat ca un micro-framework, deoarece nu necesită anumite instrumente sau biblioteci. Nu are un strat de abstractizare a bazei de date, validare de formulare sau orice alte componente în care bibliotecile terțe preexistente oferă funcții comune. Cu toate acestea, Flask acceptă extensii care pot adăuga caracteristici ale aplicației ca și cum ar fi implementate în Flask însuși. Există extensii pentru cartografii obiect-relațional, validarea formularelor, gestionarea încărcării, diverse tehnologii de autentificare deschisă și câteva instrumente comune legate de cadru.

Unele caracteristici care fac din Flask un cadru ideal pentru dezvoltarea de aplicații web sunt: oferă un server de dezvoltare și un depanator, folosește șabloane Jinja2, fste compatibil cu WSGI 1.0, oferă suport integrat pentru testarea unitară, sunt disponibile multe extensii pentru Flask, care pot fi folosite pentru a-și îmbunătăți funcționalitățile.

1.3.13 PDFMiner

PDFMiner este un instrument pentru extragerea de informații din documente PDF. Spre deosebire de alte instrumente legate de PDF, se concentrează în întregime pe obținerea și analiza datelor text. PDFMiner vă permite să obțineți locația exactă a textului într-o pagină, precum și alte informații, cum ar fi fonturi sau linii. Include un convertor PDF care poate transforma fișierele PDF în alte formate de text (cum ar fi HTML). Are un parser PDF extensibil care poate fi folosit în alte scopuri decât analiza textului.

1.3.14 Googletrans

Googletrans este o bibliotecă python gratuită și nelimitată care a implementat Google Translate API. Aceasta utilizează API-ul Google Translate Ajax pentru a efectua apeluri către metode precum detectarea și traducerea.

Caracteristici:

- rapid și fiabil - folosește aceleași servere pe care le folosește translate.google.com
- Detectare automată a limbii
- Traduceri în bloc
- Adresa URL personalizată a serviciului
- Suport HTTP/2

1.3.15 NLTK

Setul de instrumente pentru limbajul natural (NLTK) este o platformă utilizată pentru construirea de programe Python care funcționează cu date despre limbajul uman pentru aplicarea în procesarea statistică a limbajului natural (NLP).

Conține biblioteci de procesare a textului pentru tokenizare, parsare, clasificare, stemming, etichetare și raționament semantic. Include, de asemenea, demonstrații grafice și seturi de date eșantion.

Natural Language Toolkit este o bibliotecă open source pentru limbajul de programare Python, scrisă inițial de Steven Bird, Edward Loper și Ewan Klein pentru utilizare în dezvoltare și educație.

Vine cu un ghid practic care introduce subiecte în lingvistica computațională, precum și elementele fundamentale de programare pentru Python, ceea ce îl face potrivit pentru lingviștii care nu au cunoștințe profunde în programare, ingineri și cercetători care trebuie să se aprofundeze în lingvistica computațională, studenți și educatori.

NLTK include mai mult de 50 de corpus și surse lexicale, cum ar fi Penn Treebank Corpus, Open Multilingual Wordnet, Problem Report Corpus și Lin's Dependency Thesaurus.

1.3.16 Crontab

Cron este numit după cuvântul grecesc „Chronos” care este folosit pentru timp. Este un proces de sistem care va efectua automat sarcini conform programului specific. Este un set de comenzi care sunt utilizate pentru rularea sarcinilor obișnuite de programare. Crontab înseamnă „cron table”.

Crontab este, de asemenea, numele programului, care este folosit pentru a edita acel program. Este condus de un fișier crontab, un fișier de configurare care indică comenzile shell care trebuie rulate periodic pentru programul specific.

Daeomon-ul_cron este un proces de lungă durată care execută comenzi la anumite date și ore. Puteți folosi acest lucru pentru a programa activități, fie ca evenimente unice, fie ca sarcini recurente.

1.3.17 MongoDB

MongoD este o bază de date NoSQL orientată spre documente, utilizată pentru stocarea datelor cu volum mare. În loc să folosească tabele și rânduri ca în bazele de date relaționale tradiționale, MongoDB folosește colecții și documente. Documentele constau din perechi cheie-valoare care sunt unitatea de bază de date în MongoDB. Colecțiile conțin seturi de documente și funcții, care este echivalentul tabelelor de baze de date relaționale. MongoDB este o bază de date care a apărut la mijlocul anilor 2000.

MongoDB nu este un sistem de management al bazelor de date relaționale (RDBMS). Se numește bază de date „NoSQL”. Este opus bazelor de date bazate pe SQL, unde nu normalizează datele în scheme și tabele în care fiecare tabel are o structură fixă. În schimb, stochează datele în colecții ca documente bazate pe JSON și nu impune scheme. Nu are tabele, rânduri și coloane ca alte baze de date SQL (RDBMS).

În baza de date RDBMS, un tabel poate avea mai multe rânduri și coloane. În mod similar, în MongoDB, o colecție poate avea mai multe documente care sunt echivalente cu rândurile. Fiecare document are mai multe „câmpuri” care sunt echivalente cu coloanele. Documentele dintr-o singură colecție pot avea câmpuri diferite.
Următorul este un exemplu de document bazat pe JSON.

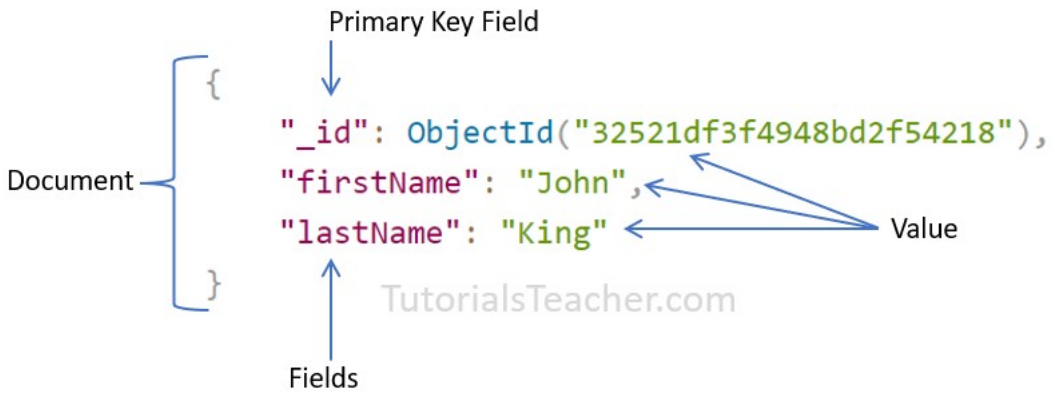


Figure 1.2 Document MongoDB

Caracteristici:

Fiecare bază de date conține colecții care la rândul lor conțin documente. Fiecare document poate fi diferit cu un număr diferit de câmpuri. Dimensiunea și conținutul fiecărui document pot fi diferite unele de altele.

Structura documentului este mai în concordanta cu modul în care dezvoltatorii își construiesc clasele și obiectele în limbajele de programare folosite. Dezvoltatorii spun adesea ca, clasele lor au mai degrabă o structura clara cu perechi cheie-valoare, decât rânduri și coloane.

Rândurile(sau documentele așa cum sunt numite în MongoDB), nu trebuie sa aibă o schema definita în prealabil. În schimb, campurile pot fi create din mers.

Modelul de date disponibil în MondoDB va permite sa reprezentați mai ușor relații ierarhice, sa stocati matrice și alte structuri mai complexe.

Scalabilitate – Mediile MongoDB sunt foarte scalabile. Companiile din întreaga lume au definit clustere, unele dintre ele ruland peste 100 de noduri cu aproximativ milioane de documente în baza de date.

Capitolul 2. Proiectarea aplicației

Arhitectura a sistemului se bazează trei componente principale: un frontend server de tip React.JS care servește cererile clientilor venite din UI, folosește elemente reutilizabile și comunica prin request-uri de tip HTTP cu serverul Node.JS, care la rândul sau, are un schimb continuu de informatii cu procesul de recomandare, scris în integralitate în Python. La nivel de interfatare și pentru facilitarea interactiunii serviciilor web RESTful regasim API's (Application Programming Interfaces) de tip REST atât pe instanța de tip Node.JS cât și pe cea de Python. Diagrama de blocuri, top-level a sistemului urmărește o vedere de asamblu asupra componentelor și cu ajutorul ei putem observa intrările și ieșirile sistemului. Aceasta este prezentata în figura de mai jos.

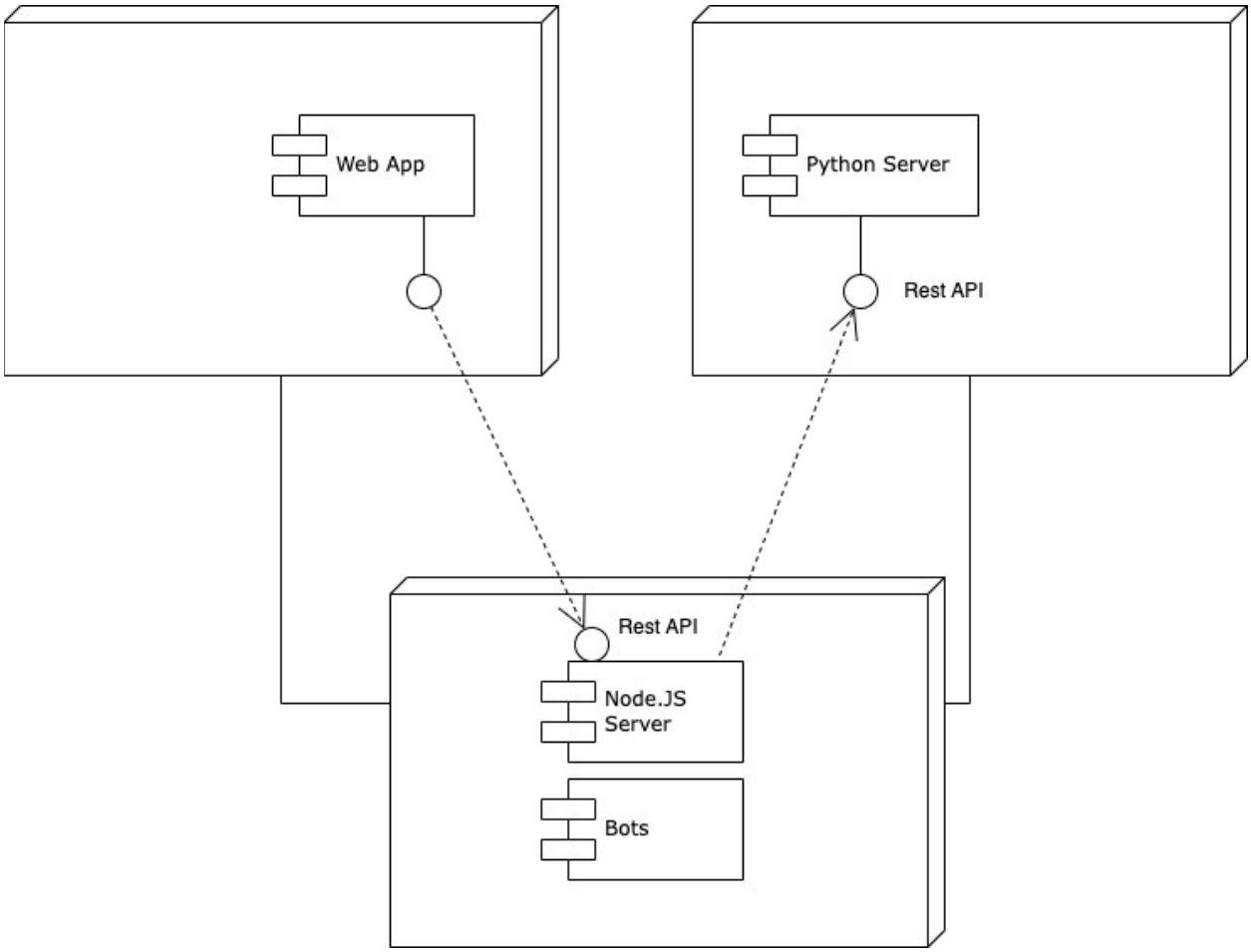


Figure 2.1 Diagrama de blocuri

Aplicația este una de tip client-server, unde exista un browser(ca și client) și doua servere web care comunica intre ele. Canalul de comunicare este bazat pe schimbul de informații cu ajutorul protocolului HTTP, iar stocarea datelor se face local, pe un server de tip MongoDB.

2.1 Arhitectura pe trei niveluri

Arhitectura cu trei niveluri este o arhitectură de aplicații software bine stabilită care organizează aplicațiile în trei niveluri de calcul logice și fizice: nivelul de prezentare sau interfața cu utilizatorul; nivelul aplicației, unde sunt prelucrate datele; și nivelul de date, unde datele asociate cu aplicația sunt stocate și gestionate.

Avantajul principal al arhitecturii cu trei niveluri este că, deoarece fiecare nivel rulează pe propria infrastructură, fiecare nivel poate fi dezvoltat simultan de o echipă de dezvoltare separată și poate fi actualizat sau scalat după cum este necesar, fără a afecta celelalte niveluri.

În cazul aplicației noastre se disting urmatoarele niveluri de dezvoltare:

- Nivelul de prezentare este reprezentat de interfata cu utilizatorul prin care acesta poate interactiona cu functionalitatile sistemului. În cazul de fata, acesta a fost scris în JavaScript, CSS, HTML și folosind ReactJS ca și framework, utilizand librarii ca Materialise și Boostrap pentru realizarea componentelor vizuale, prin care utilizatorul poate comunica cu logica de back-end modulelor.
- Nivelul de de Business este realizat în NodeJS și ExpressJS pe de o parte, iar pe de alta parte motorul de recomandare(eng: engine) a fost scris în Python. Aceste doua servere reprezintă practic puntea de comunicare dintre nivelul de client și cel de baza de date(de stocare a informațiilor). Regurile de afaceri și logica prin care se pot adauga, sterge, modifica resurse este prezenta la nivelul de controlere și servicii. În cazul aplicației noastre avem controlere atât pe partea de Node.JS: engine, job, users cât și pe partea de pyhon, unde avem mai multe functionalitati de business existente printre care se număra functia de recomandare, functia de asignare și programare unui task pentru un user nou înregistrat, functia prin care i se atribuie fiecarui utilizator o colecție noua de scoruri procesate pentru locuri de munca, fiecare logare.
- Nivelul de date numit și nivel de baza de date sau nivel de acces la date, este locul în care sunt stocate și gestionate informațiile procesate de aplicație și în cazul aplicației noastre acesta este reprezentat de un server de baze de date NoSQL(eng: Not only SQL). Informațiile sunt stocate în documente, care la rândul lor fac parte din niște colectii, a caror reuniune alcatuiesc baza noastră de date, foarte importanța în cadrul aplicației noastre. O diagrama reprezentativa pentru un sistem de tip trei straturi(eng: 3-tier system) se găsește în poza de mai jos:

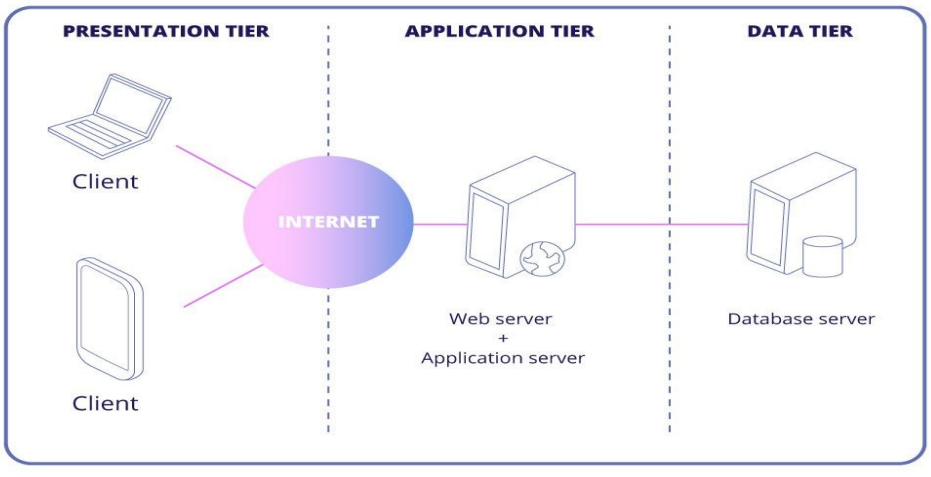


Figure 2.2. Arhitectura bazata pe trei straturi

2.2 Arhitectura de asamblu a sistemului

Arhitectura generala a sistemului și comunicarea dintre utilizator și componente este prezentata în Figura 2.3.

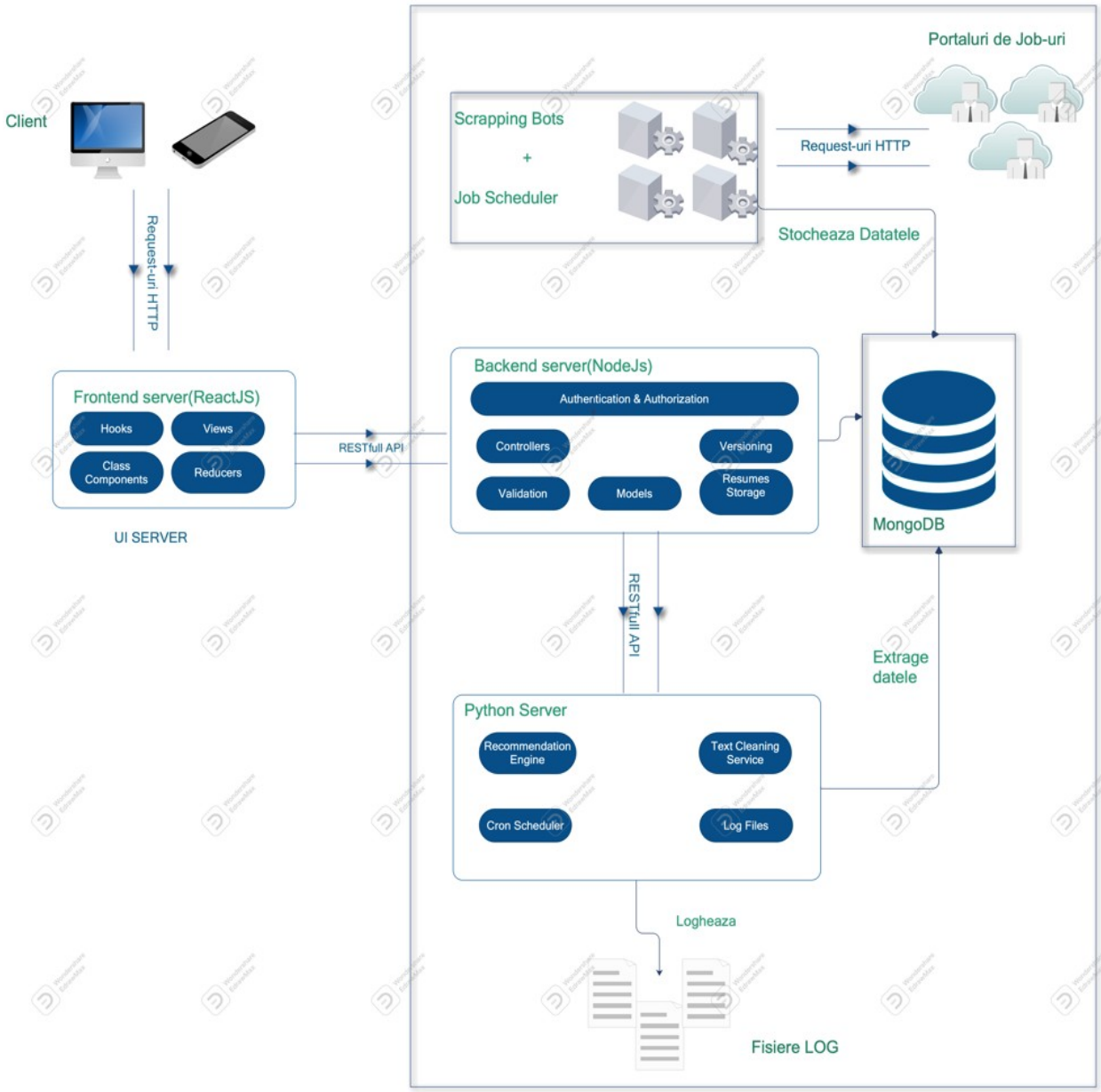


Figura 2.3. Arhitectura generala a sistemului

Comunicarea intre componentele din sistem se face prin cereri și răspunsuri pe baza protocolului HTTP. Arhitectura tinde spre una derivata din servicii(SOA), în care componentele secundare au rol de modularizare a programul și cresc scalabilitatea. De asemenea, prin ajutorul lor se creaza cod reutilizabil și independent de limbajul de programare. Așadar, s-au putut folosi mai multe tehnologii și limbaje de programare. În cadrul sistemului se disting mai multe subcomponente cu logica independenta, care interactioneaza și apeleaza la un schimb de informații constant pentru buna funcționare a aplicatiei.

2.2 Boti de scrapping web

Web scrapping-ul reprezintă o modalitate de a colecta date disponibile în mod public de pe WWW(World Wide Web). Scopul acestui proces, este sa automatizeze colectarea de date relevante, care ar lua foarte mult timp dacă ar fi executata pe o cale manuala.

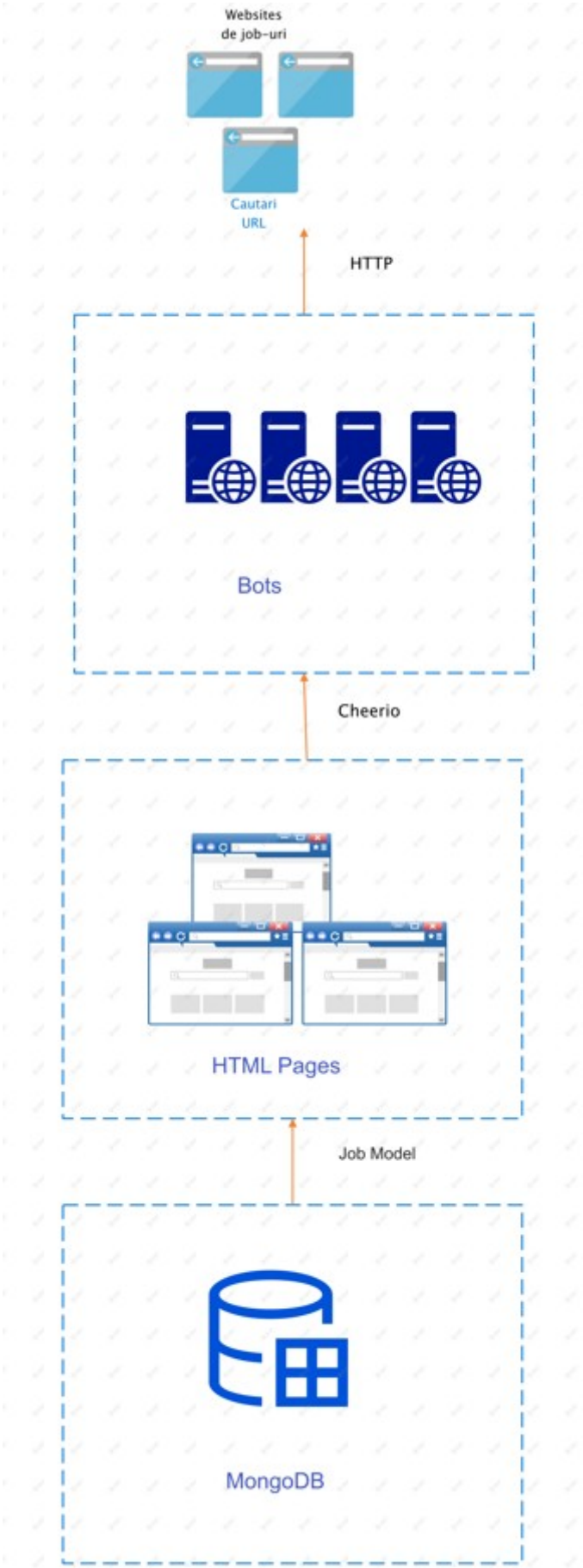


Figura 2.4 Extragerea informatiilor

Scraperele web automate funcționează într-un mod simplu de înțeles, fiecare proces de scrapping având mai multe etape. Instantele acestor procese se numesc roboti(sau boti), iar în cadrul aplicației noastre am atribuit câte un bot pentru fiecare website specializat pe oferte de munca. Întreg procesul de scrapping este reprezentat în Figura 2.4.

În prima instanța, se verifica dacă procesul de preluare de înformatii este legal, accesand fișierul *robots.txt* al fiecarui website luat în vedere.

Ca și prima etapa în procesul de functionare, robotii primesc ca și intrare unul sau mai multe *URL-uri* (eng: Uniform Resource Locator)*;* în cazul aplicației de fata vom extrage informațiile relevante de pe: https://www.ejobs.ro/, https://jobzz.ro/, https://www.romjob.ro/, https://www.hipo.ro/ având un numar total de patru boti. Ținta lor este sa extraga informațiile de interes din lista de locuri de munca, intentionand sa preia toate datele pentru fiecare slujba(eng: job) în parte.

În a doua etapa de scrapping, se va cauta pe pagina principala elemente care sa conțină numărul de pagini de pe website-uri, toate dispunand de paginare și se va extrage numărul total, ca mai apoi sa putem sa le parcurgem, punând procesul pe pauza de câteva secunde intre apeluri succesive pentru a evita blocarea ip-ului și primirea header-ului *Try-After* în raspuns. A fost abordata o varianta intenționat sincrona de parsare a paginilor, pentru a organiza datele cronologic și structurat în baza de date.

Ulterior, se calculează URL-ul fiecarei pagini, bazat pe cel al paginei principale plus o combinatie dintre un cuvânt specific și numărul curent. Mai apoi, se încarca întregul codul HTML-ul aferent fiecarei paginii pe care l-am folosit mai sus, pas în care este foarte importanța înțelegerea anatomiei paginei, fiecare website având o structura diferita. Utilizand unealta de inspectare din Google Chrome, vom cauta etichetele de interes pentru ofertele de munca, fiecare informație relevanta aflandu-se în interiorul acestora(ex: div, h1, h2, ul, img, table, tr etc.).

Mai apoi, cu ajutorul bibliotecei Cheerio(construita pe baza Jquerry) putem extrage datele din marcajele enuntate mai sus, făcând la nevoie cereri la url-uri noi penru a extrage informații aditionale despre job-uri: cum ar fi descrierea lor sau imaginea angajatorului. În majoritatea cazurilor, o resursa dintr-un cod HTML poate fi găsită pe baza id-ului, clasei sau a metatributelor ei. În cazul în care acestea nu sunt prezente, elementele au fost găsite pe baza manierei în care este alcatuita arhitectura Dom-ului. S-au identificat relații de rudenie intre elemente(ex: noduri părinte, noduri copii, frați sau surori) și pe baza precedentei în containerele parinte(ex: al doilea paragraf din al treilea div). De asemenea, în cazul tuturor botilor ofertele de munca erau organizate în liste (<ul> sau <ol>), fiind nevoie de o parcurgere suplimentara în cazul lor.

În final, se aplica o curatare a informațiilor extrase din slujbe, pentru ca acestea vor fi refolosite în viitor și organizate sub diferite forme de date(structurate sau semistructurate), astfel evitand erorile de parsare care ar putea sa apara la schimbul de informații dintre sisteme. Datele au fost organizate și salvate, pe baza unui model, definit la rândul de o schema în baza de date.

Pentru o mai buna acuratete, actualitate și precizie a datelor, a fost dezvoltat un serviciu care sa lanseze în execuție fiecare bot ca și un proces separat la un interval de timp(ex: 30 de minute, o oră, la fiecare 12 ore, în fiecare zi). Așadar, cele mai noi oferte de munca for fi găsite, iar utilizatorii vor putea accesa înformatii în timp real.

În cazul de fata, dificultatea procesului de extragere a informațiilor a constant în determinarea exactă a numarului de pagini de pe fiecare website, găsirea elementelor care aveau o imbricare complexa în nod-uri de tip container și evitarea blocarii de apeluri de către server. În repetate rânduri s-a primit codul de status: 429(prea multe cereri într-un interval de timp) culminand cu o eroare de conectare către server(*Error: connect ECONNREFUSED*), acest fapt crescând complexitatea procesului de scrapping. Pentru a evita aceasta eroare, a fost capturata într-un bloc de tip try-catch, raspunul a fost interpretat pentru a afla numărul de secunde de așteptare pentru a efectua o noua o cerere, urmând ca procesul sa doarmă pentru timpul cerut. Ulterior, s-au reincercat noi apeluri pentru a prelua datele de interes. Pentru toate instantele de roboti, s-au extras informații cu succes, fără a fi plasati într-o lista neagră.

2.3 Proiectarea aplicației frontend

Aplicația frontend a fost dezvoltata utilizand ReactJS deoarece acesta biblioteca ofera numeroase avantaje precum:

- complexitatea redusa fata de alte framework-uri(biblioteci) similare precum Angular sau Vue
- randarea interfetei utilizator (eng: User Interface) este foarte rapidă datorită VDOM-ului care vine odată cu aceasta librărie.
- elementele alte interfetei reutilizabile, care pot fi folosite oriunde în proiect.
- aplicația frontend are o arhitectura de tip SPA(eng: Single Page Application), diferita fata de modul în care sunt construite aplicatiile traditionale cu mai multe pagini.

2.3.1 Aplicație cu o singura pagina(eng: SPA)

SPA este tipul de aplicație web care funcționează în cadrul unui browser. Marele avantaj este ca platforma nu necesită reîncărcarea paginii atunci când trebuie afișate date noi. Aceasta ofera o randare rapidă după ce aplicația este incarcata complet în browser și creeaza un software foarte receptiv pentru utilizatorul final. Acesta vine și cu dezavantaje, precum timpul îndelungat necesar pentru prima încărcare , rutarea slaba și suportul limitat al browserelor invechite. În cazul aplicației noastre exista patru astfel de pagini(*/, register, login, dashboard*), iar alternarea lor se realizeaza prin intermediul unui *React Router.*

O diagrama a arhitecturii unei aplicații de tip SPA se găsește în diagrama de mai jos:

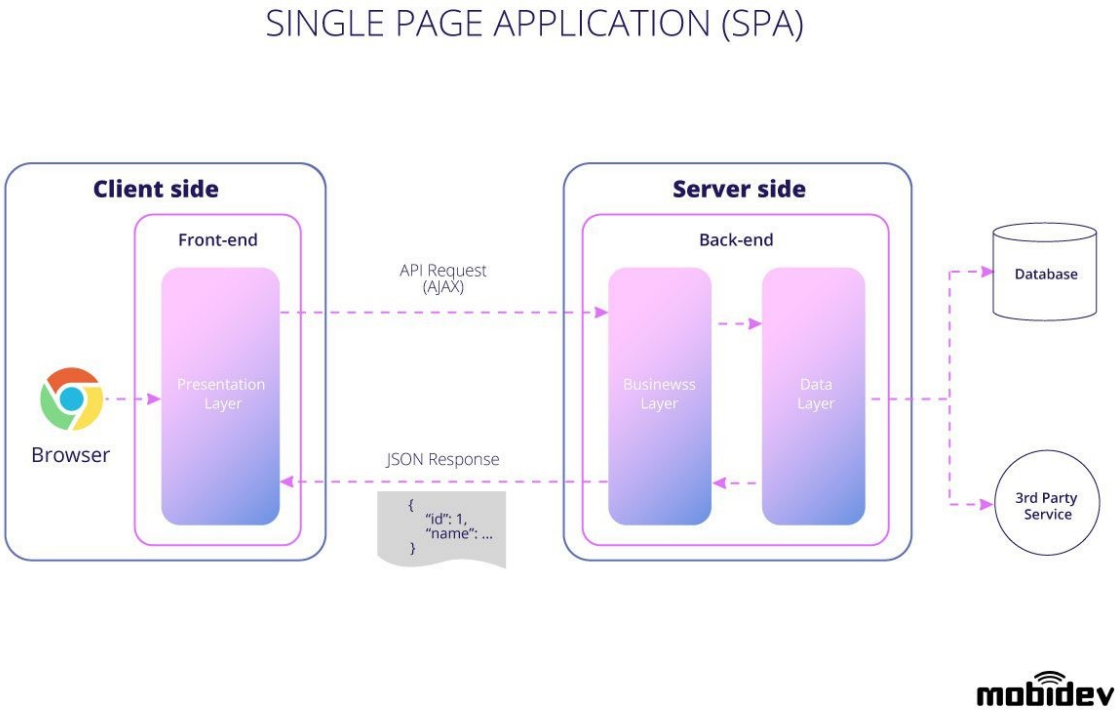


Figura 2.5. Aplicatie cu o singura pagina(SPA)

2.3.2. React Redux

În React, reductoarele oferă o modalitate de a actualiza starea unei aplicații folosind o acțiune, usurand complexitatea de proiectare a aplicației.

Un Reductor este o funcție pură care ia ca argumente starea unei aplicații și a acțiunii și returnează o stare nouă. De exemplu, un reductor de autentificare poate lua o stare inițială a unei aplicații sub forma unui obiect gol și o acțiune care îi spune că un utilizator s-a conectat și a returnat o nouă stare de aplicație cu un utilizator conectat.Funcțiile pure sunt funcții care nu au efecte secundare și vor returna aceleași rezultate dacă sunt transmise aceleași argumente.

În cadrul unei aplicații de tip React care folosește conceptul de recuctori, distingem mai multe componente:

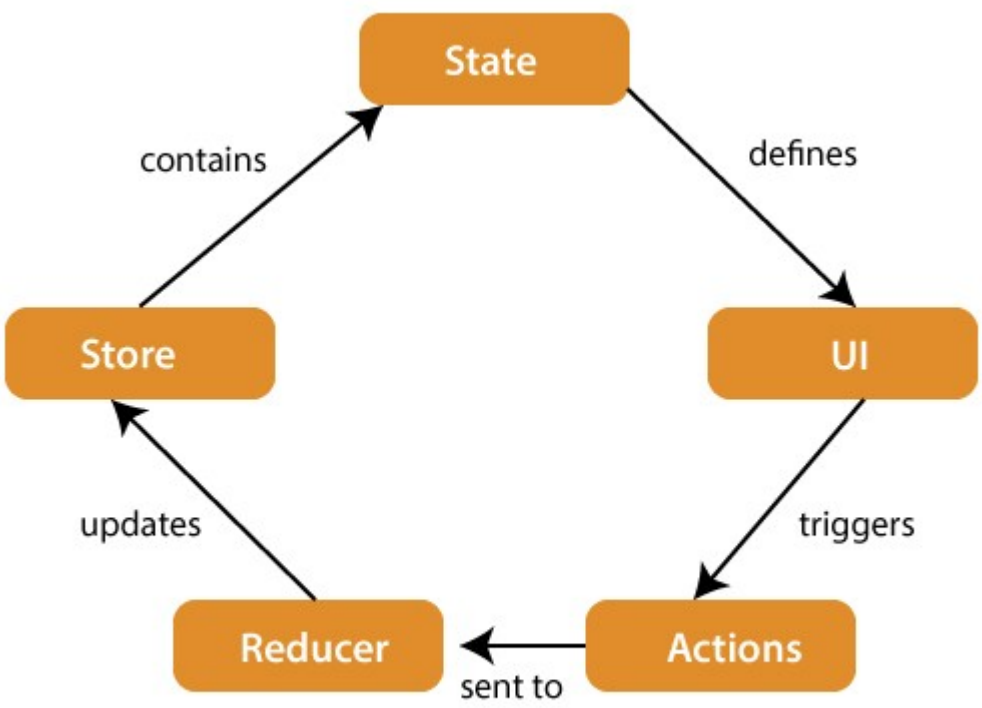


Figura 2.6. Maniera in care starile componentelor sunt gestionate

STARE(eng: state): reprezintă datele cu care lucrează o componenta - deține informatiile pe care le necesită și le foloseste o componentă și dictează comportamentul unei componente, omologul ei fiind o variabila care reține o valoare. Odată ce un state se schimbă, componenta se randeaza din nou. Dacă starea unei aplicații este gestionată de Redux, atunci reductorul este locul unde se desfășoară schimbarile de stare, astfel procesul de manageriere și de observare ale valorilor unei componente este mai ușor de urmărit..

DEPOZIT(eng: store): Un depozit este un loc în care se listează întreaga stare a aplicației web realizata în React. Gestionează starea aplicației și are o funcție de expediere (acțiune). Este ca o unitate centrala responsabila pentru toate părțile mobile din Redux și pentru gestionarea starilor lor.

ACȚIUNE: O acțiune este un obiect pur creat pentru a stoca informațiile despre evenimentul utilizatorului. Include informații precum tipul de acțiune, momentul apariției, locația apariției, coordonatele acesteia și starea pe care intenționează să o schimbe.

REDUCTOR(eng: Reducer): Reductorul citește și interpreteaza acțiunile trimise de componente și apoi actualizează depozitul în consecință. Este o funcție pură capabila să returneze o stare nouă, actualizata din starea precedenta.

Mecanismul prin care funcționează conceptul de reducator sau manager de stari este reprezentat în Figurile 2.6 și 2.7.

In cazul aplicației noastre componentele de Login și de Register vor fi legate la depozitul redux prin intermediul functiei *connect*. O eveniment realizat de un potențial client(de ex: un click pe un buton din UI sau trimiterea unui formular) vor declansa câte o acțiune (*registerUser, loginUser, setCurrentUser*) care va fi rederictionata către depozit. Ulterior, reductorii din aplicației vor specifica modul în care state-ul se va schimba(authReducer: SET_CURRENT_USER → isAuthenticated; USER_LOADING → loading: true; default: return current_state; errorReducer: GET_ERRORS: return payload; default: return current_state). De asemenea, procesul de aducere a ofertelor de munca din baza de date este implementata tot prin reductori, fiecare tip de actiune (MAKE_REQUEST, GET_DATA, ERROR, UPDATE_HAS_NEXT_PAGE) schimbând state-ul curent într-o maniera diferita. Atunci când se face o cerere către server, se dorește ca se afiseze un mesaj de așteptare(eng:loading), iar vectorul cu valori să fie gol; atunci când primim date de la server vom elimina mesajul de loading și vom popula container-ul cu oferte de munca; în eventualitatea în care apelul este încheiat cu o eroare (fie de la client, fie de pe backend) vom afisa un mesaj sugestiv pe ecran și nu vom avea slujbe de afisat în frontend; pentru componenta de paginare, vom verifica dacă mai exista elemente de adus din baza de date și vom actualiza valoarea variabilei: engine_hasNextPage.

Comunicarea componentelor din React se face într-o maniera unidirectionala de la părinte la copil(in cadrul ierarhiei), iar când un nod de la fundul arborelui dorește sa comunice cu predecesorii lui acesta va trebui să se folosească de obiectul props. În cazul de fata, reductorii usureaza acest proces, schimbul de date facandu-se bidirectional intre oricare doua componente, iar toate starile sunt manageriate și modificate în același loc(in depozitul de date).

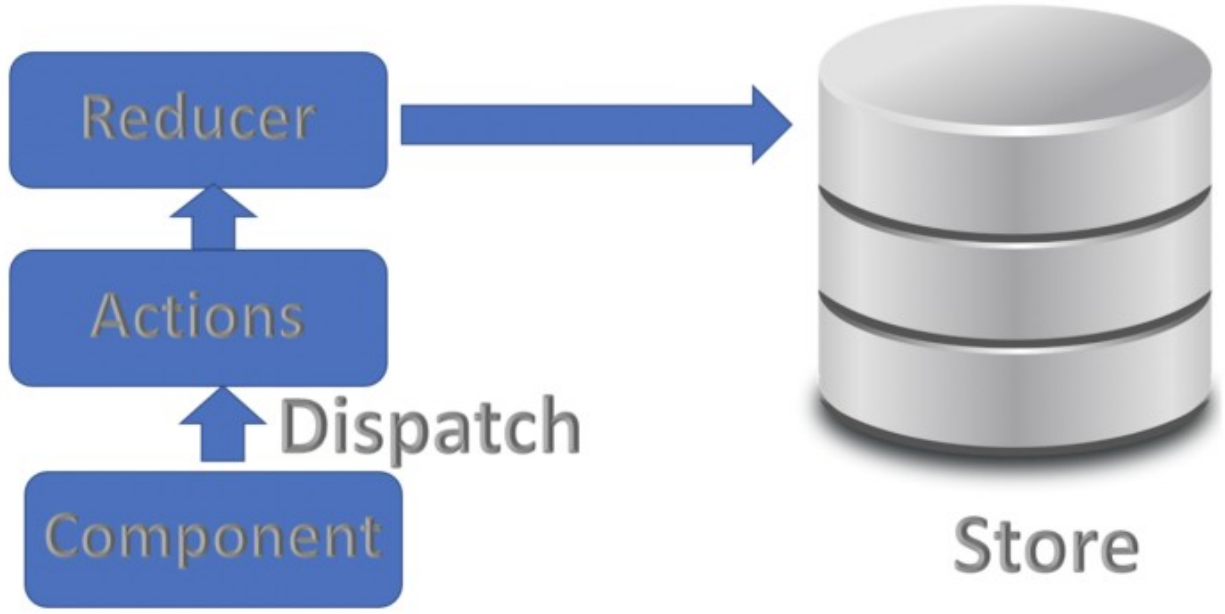


Figura 2.7. Modul de functionare a unui depozit

2.4 Baza de date

MongoDB este o baza de date open-source, conceputa atat pentru scalabilitate, cat si pentru rapiditatea cu care se poate dezvolta o aplicatie pe urma ei.

In principal, exista doua mari tipuri de sisteme de gestionare a bazelor de date: SGBD relational si NoSQL(Not Only SQL). In cadrul modelului relational informatiile sunt stocate intr-un format structurat, ce foloseste tabele si coloane. Pe de alta parte, MongoDB faciliteaza stocarea de date structurate sau nestructurate sub forma de documente folosind un format asemanator JSON, care este usor mapabil peste obiectele native in majoritatea limbajelor de programare moderne, fapt ceea ce il face o alegere naturala pe dezvoltatori, acestia nemaitrebuind sa se gandeasca la normalizarea datelor. Asadar, sistemul de stocare a datelor devine mai flexibil, varietatea datelor dintr-un document fiind mare. Motivatia alegerii unui sistem de tip NoSQL in aplicatia de fata a fost data de avantajele unui mediu de date care nu necesita o schema clar definita(eng: Schema-Less), neavand nevoie de relatii intre instantele sistemului. Mai mult, instanta curenta a unei baze de date poate fi salvata intru-un format semistructuruat(cum ar fi JSON sau CSV) acest lucru permintand organziarea datelor pentru mai multe cazuri de testare a aplicatiei(au fost salvate fisiere diferite pentru fiecare website pe care se executa procesul de scrapping).

In componenta bazei de date regasim trei colectii:

_id	ObjectId
name	String
email	String
password	String
resume	String
resumePath	String
jobDescription	String
last_login_date	String

Figura 2.9. users

_id	ObjectId
jobName	String
jobEmployer	String
jobLocation	String
jobDate	String
jobUrl	String
jobDescription	String
jobPageNumber	Integer
jobImageUrl	Integer

Figure 2.8. jobs

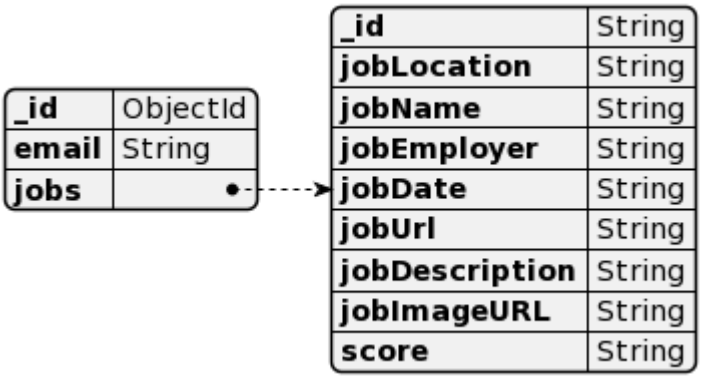


Figura 2.10. users_recommended_jobs

Tabela ***users*** conține informații legate de contul utilizatorilor aplicației. Aceștia sunt identificați pe baza unei adrese de email(am folosit acest câmp datoritata proprietatii de unicitate pe care confera acest câmp; nu pot exista doi utilizatori cu aceeași adresa de e-mail). În formularul de inregistrare, clientii aplicației vor mai fi nevoiți sa își aleaga un nume de utilizator

ce are caracter mai mult informativ în cadrul aplicației, o parola care va fi stocata în mod encriptat pentru a evita furtul de informație în cazul unui atac sau un acces ostil la baza de date a server-ului. Encriptia se realizeaza cu ajutorul modului Bcrypt care folosește un hash de tip Blowfish, cu o generare de salt suficient de mare încât algoritmul sa funcționeze lent, dar în limita rabdarii utilizatorului. De asemenea, clientul aplicatiei va mai avea la dispoziție un câmp în care sa își pună descrierea personala(*resume)*. Colectia va mai conține calea către curriculum vitae ul utilizatorului, iar fișierul va fi stocat local într-un folder, unic pentru fiecare client. Aceasta abordare a fost aleasa peste cea clasica(stocarea pdf-urilor și fisierelor text în baza de date, în tabela de utilizatori) deoarece timpul te interogare scade semnificativ(deoarece exista mai puține date transmise intre aplicație și baza de date), se elimina necesitatea unei eventuale preprocesari a fisierului pentru a putea fi stocat într-o colectie(ex: converitre în baza64 și reconvertire în formatul folosit înainte). Mai mult, stocarea pe hard disk este în general mai ieftină, echivalentul în RAM fiind semnificativ mai scump. O baza de date mai mare va folosi mai multă memorie RAM pentru a stoca indecsi și date frecvent interogate pentru a imbunatati performanța, ea fiind deja un consumator principal în cazul acestei memorii. De asemenea, documentele din MongoDB sunt limitate la 16MB în dimensiune, iar necesitatea stocarii unor fisiere mai mari aduce în prim plan folosirea bibliotecii *GridFS*, acest fapt crescând complexitatea aplicației. Ultimul câmp din colectia users reprezintă ultima data a logarii, variabila ce poate fi folosită în sistemul de executare de task-uri de recomandare pentru fiecare utilizator. Fiecarui client al aplicației ii este atribuit la inregistrara în platforma un proces daemon ce lucrează în background, a carui scop este sa lanseze în execuție functia de recomandare pe baza ultimei dați a logarii utilizatorului.

În continuare, tabela ***jobs*** conține ***i***nformații despre componenta principala în jurul în careia graviteaza functionalitatile aplicației**.** Ca și cheie primara s-a folosit un obiect de tip ObjectId din MongoDB care reprezintă o valoare unica formata dintr-un timestamp de patru octeti, reprezentând data crearii, o valoare aleatorie de cinci octeti(unica pentru mașina și proces) și un contor de trei octeti(initialziat cu o valoare aleatorie). În cele ce urmează s-au stocat informații referitoare la un loc de munca, precum titlul ofertei, numele angajatorului, data la care a fost postat job-ul, URL-ul unde se poate regăsi slujba pe website-ul pe care a fost publicata initial, descrierea acestuia care va fi folosită în etapa de recomandare și este afisata în frontend și în final, imaginea cu care se indentifica angajatorul.

Ultima colecție ***user_recommended_jobs***, este folosită exclusiv în cazul procesului de recomandare a job-urilor, individualizat pentru fiecare utilizator în parte. Tabela se populeaza în trei în momente diferite: la inregistrarea unui utilizator, procesul asignat acestuia este responsabil de recomandarea ofertelor de munca și popularea acestei tabela; la logarea unui nou client și la apasarea butonului *For You*. Câmpul email identifica în mod unic un utilizator(ca și în cazul tabeli users), iar jobs reprezinta un vector de subdocumente, rezultat în urma finalizarii procesului de recomandare. Pe lângă informațiile referitoare la o oferta de munca, se regaste un câmp care reprezintă scorul atribuit job-ului, rezultat în urma sistemului de asignare și potrivire, în funcție de cv-ul utilizatorului. Aceasta colecție simuleaza o memorie cache, accesarea datelor fiind făcută frecvent, iar acestea sunt precalculate recent. Așadar, se reduce timpul de astepare semnificativ pentru locurile de munca recomandate, utilizatorul având un rezultat instant(eng: feedback).

În timp ce SQL defineste o schema la crearea tabelului, în Mongoose aceasta reprezintă o structura de date a documentului(sau forma documentului) care trebuie salvat în baza de date, iar modele reprezintă obiecte care preiau o schema și creeaza o instanța a unui document, echivalent cu inregistrarile dintr-o baza de date relationala. Spre deosebire de o schema(care defineste structura documentului, valorile implicite și validarea campurilor), un model ofera o interfata spre baza de date, pentru crearea, interogarea, actualizarea, stergerea inregistrarilor etc. În cadrul aplicației de fata, s-au realizat trei scheme: UserSchema, RecommendedJobsSchema și JobSchema cu validarile aferente campurilor în funcție de necesitate, iar toate operatiile precum:

preluari, actualizari, stergari, instantieri ale documentelor din baza de date sunt efectuate prin aceasta interfata, de oricare dintre serviciile programului(inclusiv botii de scrapping).

2.4.1 Paginare

Paginarea este metoda de separare a conținutului digital în diferite pagini de pe un site web. Utilizatorii pot naviga între aceste pagini făcând clic pe linkuri, adesea sub formă de numere situate în partea de jos a paginii. Conținutul paginat este de obicei legat de o temă sau un scop comun.

O alta metoda de separare a fluxului de date venit de pe un server este, derularea infinită, practică întâlnită în experiența utilizatorului (UX), unde conținutul se încarcă continuu atunci când un clientul ajunge la partea de jos a paginii. Acest lucru creează experiența unui flux nesfârșit de informații pe o singură pagină, aparent fără sfârșit.

In aplicația de fata, ca metoda de subdivizare a informatiei, s-a recurs la paginare deoarece exista o cantitate mare de date, ce nu poate fi reprezentata în mod rezonabil într-o singura pagina. Așadar, experiența utilizatorului devine mai plăcută acesta nefiind copleșit de numărul mare de oferte de munca care pot fi afisate la un moment, iar navigarea devine mult mai ușoară. Mai mult, fluxul infinit de date este mai potrivit atunci când obiectul cautarii nu este bine definit(ex: în retelele sociale utilizatorii deruleaza cu un scop anume, ci vor să fie distrați și informați), fapt ce este în opozitie cu rolul aplicației de fata, unde clientii cauta oferte de munca.

În cadrul platformei, paginarea s-a realizat în doua moduri, iar aceasta este alcatuita dintr-o operație pe baza de date MongoDB și o procesare suplimentara pe frontend.

În cazul MongoDB, paginarea pentru colectia *jobs* eficiența s-a realizat în mai multi pasi, conectați intre ei:

1) In prima instanța s-a ales numele colectiei pe care se face paginarea(jobs), pe care se vor efectua mai multe operatii și interogari.

2) S-a aplicat functia de find, metoda folosită pentru a afisa documentele din colectia folosită în interogare, folosindu-se id-ul slujbei ca și index de cautare.

3) In continuare, s-a apelat skip cu scopul de a sari peste un numar de documente egal cu numărul paginii dorite inmultit cu numărul de elemente de pe o pagina(variabile a caror valori sunt transmise ca și parametrii în cererea venita de pe frontend când utilizatorul da click pe o noua pagina). Acest numar rezultat, reprezintă numărul necesar de elemente care trebuie ignorat pentru a ajunge la pagina ceruta.

4) Ulterior s-a aplicat metoda limit, folosită pentru a prelua un numar prestabilit de elemente din colecție, reprezentând numărul de instante pe care le-am afisat într-o pagina(acest numar poate fi văzut ca un offest de la începutul paginii curente).

5) În final s-a recurs la operatorul de proiectie, pentru a se extrage doar campurile de interes din urma rezultatelor interogarii.

Pentru colectia user_recommended_jobs s-au aplicat trei operatori suplimentari, aceasta având o structura diferita a datelor:

- S-a apelat operatorul de potrivire în interogare, pentru a găsi toți userii cu un anumit e-mail, iar ulterior s-au apelat functiile de agregare și de desfacere a datelor, deoarece în aceasta colecție ofertele de munca au fost stocate ca un vector de subdocumente. Așadar, a fost nevoie de o grupare suplimentara a elementelor.

Prin urmare, utilziand paginarea procesul de afisare, transfer și încărcare a datelor devine unul foarte rapid și eficient, deoarece informația este divizata în bucati(eng: chunks), utilizatorul având acces doar la o pagina de oferte de munca la un moment dat. Încărcarea tuturor ofertelor de munca pe o singura pagina(recomandate sau nerecomandate) ar fi fost un proces costisitor din punct de vederea al timpului, provocand o experiența neplacuta și nepotrivita pentru utilizatori, aceștia dorind o promptitudine afisarii datelor în acțiune de navigare pe website.

Capitolul 3. Implementarea aplicatiei

3.1. Arhitectura pe servicii si serverul Node.JS

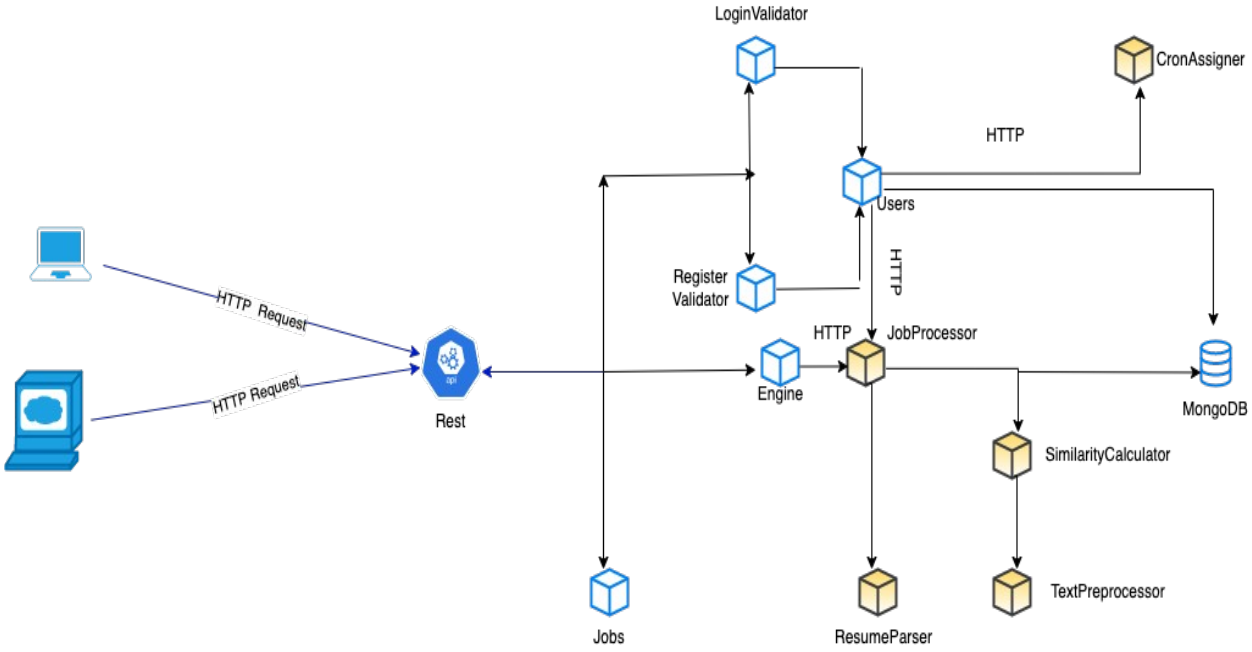


Figure 3.1. Diagrama de servicii

În ingineria software, arhitectura orientată pe servicii (SOA) este un stil arhitectural care acceptă orientarea către servicii. Prind definitie, un serviciu este o unitate discretă de funcționalitate care poate fi accesată de la distanță și care poate fi acționată și actualizată independent(ex: extragerea online a extrasului de card de credit). De asemenea, SOA este predestinat să fie independent de furnizori, produse și tehnologii.

În definitia SOA, un serviciu are patru proprietăți conform cărora acestea ar trebui proiectate: in mod logic, reprezintă o activitate comercială repetabilă cu un rezultat specificat; este de sine stătător; conceptual reprezintă o cutie neagră pentru clientii săi, insemnand ca acestia nu ar trebui sa știe cum este implementat ca sa îl poată folosi, poate fi compus din alte servicii.

Din punct de vedere al organziarii în ceea ce privește logica și partea computationala a aplicatiei, aceasta este alcatuita din doua servere ce ruleaza în mod independent, pe doua adrese diferite. Din punct de vedere functional, cele doua instante mentioante mai sus realizeaza și se ocupa de aspecte diferite ale platformei: serverul Node.Js interactioneaza direct cu serverul React.JS(aflat în partea de front-end) integrand servicii ca User.js, Jobs.js, Engine.js, validatori pe parte de Login cât și Register, dar și cu procesul de Python, în final acesta jucând și rol de middleware în aplicația noastră. Prin urmare, serverul de recomandare nu poate comunica în mod direct cu frontend-ul.

Pentru a obtine o modularizare eficiența în cadrul platformei, s-au împărțit functionalitatile implementate pe serverul Node.js în servicii diferite, fiecare dintre acesta având și implementand un scop și o logica diferita. Totodata, s-a urmărit ca interacțiunea să fie una slab cuplata, pentru a evita interdepenta dintre componente și o eventuala picare a intregului sistem în cazul în care un serviciu devine nefunctional.

O componenta de baza în functionalitatea sistemului este serviciul **Users.js**. Acesta este alcătuit din doua funcții: register și login.

Functia de inregistrare are ca scop crearea conturilor noilor utilizatori ai aplicației, proces înfăptuit prin mai mulți pași:

În prima instanța se acceseaza json-ul venit din UI cu datele inregistrarii noului client (nume, adresa de email, parola, CV) și se trece printr-un middleware de validare reprezentat de serviciul Validator. Cel din urma verifica ca fiecare câmp primit sa conțină informații și sa respecte o anumita structura: email-ul sa aibă structura aferenta și standardizata, parola să se incadreze intre un numar maxim și un numar minim de caractere, iar cele doua parole să fie egale, restul validarilor facandu-se în campurile formularului din frontend. Ulterior, se verifica existența unui director cu numele utilizatorului și în cazul în care acesta nu exista, creaza unul și încarca fișierul primit, salvand calea către ele în baza de date. Mai apoi se genereaza un hashing al parolei cu algoritmul Bcrypt, cu o valoare aleasa corespunzător și se salvează instanța utilizatorului în baza de date. Mai mult, se apeleaza serviciul care atribuie fiecarui nou utilizator un proces daemon de atribuire de oferte de munca, la o bucata de timp.

După finalizarea inregistrarii, un nou client va dori logarea și accesul către functionalitatile aplicației. Peste campurile formularului primit din frontend, se aplica o noua validare, pentru a ne asigura ca datele nu sunt invalide, vide sau prost formatate(ex: o adresa de e-mail care nu respecta standardul). În continuare, ne asiguram ca un utilizator a fost înregistrat pe baza adresei de email, iar în caz afirmativ intanintam în procesul de autentificare. Pentru a adauga un strat de securitate atunci când se acceseaza diferite rute ale aplicației, efectuat procesul de autentificare și autorizare pe baza tehnologiei JWT.

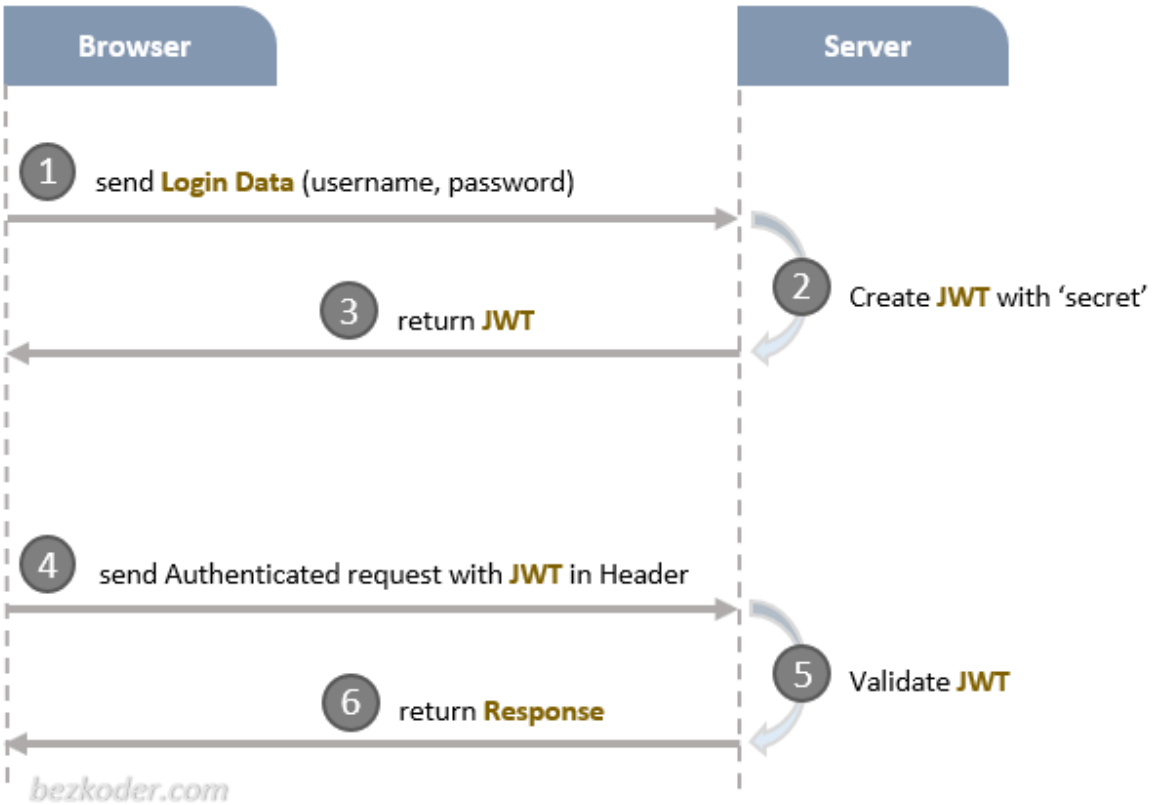


Figure 3.2. Flow-ul procesului de autentificare si autorizare al aplicatiei

JSON Web Token (JWT) este un standard care definește o modalitate compactă și autonomă de transmitere în siguranță a informațiilor între părți ca și obiect JSON. Dimensiunea compactă face ca tokenurile să fie ușor de transferat printr-un URL în corpul unui apel de tip POST sau în interiorul unui antet HTTP. Informațiile dintr-un JWT sunt semnate digital folosind o pereche de chei secrete sau publice/private. Avantajele acestei metode sunt scalarea rapidă(orice serviciu și endpoint din cadrul aplicației poate beneficia de aceasta metoda), verificarea este simpla și eficienta, aceasta este făcută la client și nu la server și este mai sigur decât autentificarea și autorizarea prin intermediul sesiunilor.

Structura unui JWT consta dintr-un:

1) Antet: are doua părți, tipul token-ului și algoritmul pe care acesta îl folosește în semnare(ex: HS512) - { "alg": "HS256", "typ": "JWT" }

2) Corpul mesajului: acesta conține informațiile care vor fi transmise în frontend în cazul unei autentificari efectuate cu succes. În cazul aplicației curente acesta conține email și id-ul din baza de date a utilizatorului. - { "id": "1234567890", "name": "John Doe"}

3) Semnatura reprezentata care imbina antetul, corpul mesajului cu algoritmul folosit și o cheie secretă, stocata la server. - HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

În momentul în care utilizatorul reușește să se conecteze cu succes, acestuia ii va fi atribuit și returna un token web JSON, care va fi stocat pe browser în local storage. Așadar, ori de câte ori clientul va accesta o ruta sau o resursa protejata, agentul va plesa în antetul de autorizare al apelului jetonul personal. Continutul header-ului va avea forma: Authorization: Bearer <token>. Cel din urma reprezintă un mecanism de autorizare rapid și eficient, deoarece utilizatorul are acces pe rutele private(/dashboard) doar dacă acesta are stocat jwt-ul aferent.

Pe server, verificarea existenței și validitatii unui token se face prin intermediul middleware-ului Passport.js:

```
export default (passport) => {
  passport.use(
    new JwtStrategy(opts, (jwt_payload, done) => {
      User.findById(jwt_payload.id)
        .then((user) => {
          if (user) {
            return done(null, user);
          }
          return done(null, false);
        })
        .catch((err) => console.log(err));
    })
  );
};
```

Crearea acestuia este prezentata în blocul de mai jos:

```
const payload = {
        id: user.id,
        name: user.name
}

 jwt.sign(
```

```
          payload,
          keys.secretOrKey,
          {
            expiresIn: 31556926, // 1 year in seconds
          },
        (err, token) => {
          res.json({
            success: true,
            token: "Bearer " + token,
        });
    }
);
```

În final, se face un apel către endpoint-ul preprocess-jobs-for-users cu email și calea către CV-ul utilizatorului pentru a porni procesul de recomandare în avans, astfel reducandu-se timpul de așteptare a ofertelor clasate în frontend. În cazul unui cont nou, baza de date de slujbe recomandate o să fie goala, iar utilizatorul nu va primi feedback instant.

Engine.js este serviciul care are ca scop aducerea de job-uri recomandate în frontend. Acesta preia din request-ul de tip GET numărul paginii care trebuie afisate, dimensiunea paginii și email-ul utilizatorului după care se face paginarea și aplica un pipeline de funcții peste tabelea user_recommended_jobs pentru a extrage doar slujbele cerute. Funcționarea acestui proces este explicata în sectiunea 2.4.1.

```
await recommendedJobsModel
    .aggregate([
      { $match: { email: email } },
      { $unwind: "$jobs" },
      { $skip: size * (pageNo - 1) },
      { $limit: size },
      { $project: { jobs: 1, _id: 0 } }
    ])
```

Jobs.js are o functionalitate similara, doar ca paginarea se aplica pe toată lista de documente existența în colecție și nu pe un vector de subdocumente așa cum e în cazul engine.js.

```
  query.skip = size * (pageNo - 1);
  query.limit = size;
  await jobsModel
    .find({}, {}, query)
```

3.2. Implementarea unor functionalitati din frontend

În interacțiunea cu serverul Node.JS, procesul React.JS folosește doua carlige(eng.hooks) ce joaca rol de servicii.

UseFetchEngineJobs.js cât și useFetchJobs folosesc useEffect pentru a actualiza rezultatele de pe frontend. Aceasta funcție se apeleaza la fiecare schimbare a parametrilor, în cazul de fata când pagina se schimba. Modificarea unei pagini se traduce în aducerea de date noi de pe serverul din backend, iar acest aspect necesita o serie de apeluri.

Schimbarea brusca a obiectului params(de ex când introducem noi date în bara de căutare sau o apasare prea rapidă a butoanelor For You sau All Jobs) are ca și consecinta apelul excesiv a functiei useEffect, deci o serie de apeluri aleatoare și nedorite. Pentru a evita aceasta situație, de fiecare data când schimbam un parametru întâmplător, vom folosi un cancelToken din

axios pentru a salva cererea făcută într-un obiect și pentru o anula în final în zona de return a functiei. Functia de cancel nu garantează oprirea trimiterii unei cereri către servar, dar evita și ignora raspunul primit, executia codului continuand în blocul de catch(error). Acolo se verifica dacă eroarea a fost una de tip isCancel și se iese din funcție în caz afirmativ.

Cu excepția acestor situații nedorite de cereri(eng: request) către server, useEffect se apeleaza ori de câte modificam variabila page(schimbam pagina). În acel moment, se trimite un request initial în care plasam numărul și dimensiunea paginii ca și parametrii. În cazul unui răspuns cu un cod 200(simbolizand o cerere trimisa cu succes), parsam raspunul primit în format JSON și actualizam starea curenta pentru a putea afisa ofertele de munca(din loading true în loading false și populam vectorul de slujbe). În cazul unei erori, se afiseaza un mesaj sugestiv în frontend.

În continuare se executa un al doilea request, în cu scopul verificarii dacă serverul mai deține suficiente oferte de munca pentru a încarca o noua pagina. În cazul în care nu se primește de pe server mesajul *isEmpty* starea se modifica, iar componenta de paginare va fi randata în continuare. În caz contrar variabila hasNextPage va primi valoarea false, iar procesul de redirectare la următoarea pagina se va opri.

3.*3. Sistemul de recomandare și clasare a locurior de munca*

Sistemul folosește tehnica de extragere a informațiilor pentru a prelua pe de o parte descrierile locurilor de munca de pe diferite portaluri de job-uri, iar pe de alta parte pentru a extrage informațiile din datele primite, atât din ofertele de munca cât și din CV-urile utilizatorilor.

Procesul se va porni în unul dintre cazurile: utilizatorul apasa pe butonul For You, acesta se logeaza în cont sau în timpul executiei daemonului asignat acestuia la inregistrarea în aplicație. Acțiunea de în urma se va executa recurent, la un interval de timp prestabilit.

Modul de funcționare a motorului are loc în mai multe etape, sub forma unei conducte(eng: pipeline) începând cu adunarea informațiilor de către roboti.

În urma procesului de scrapping, fiecare bot salvează rezultatele(datele esentiale despre un loc de munca: titlu, descriere, angajator, data postarii, locatie, url etc) în baza de date locala. Aceste informații va servi ca intrare(ex: input) în sistemul de recomandare și clasare a slujbelor.

În continuare, când una dintre situatiile de mai sus are loc, sistemul va folosi ca paremetrii de intrare doua input-uri diferite. Pe de alta parte va extrage printr-o interogare toată lista de job-uri din baza de date, iar pe de alta parte va citi într-o variabila CV-ul oferit de utilizator la inregistrarea acestuia în platforma.

Din cauza multiplelor formate în care resume-ul clientului poate fi stocat, dar cât și dificultatea cu care acesta ar putea fi manipulat în urmatoarele operatii, se executa o preprocesare și o parsare a acestuia, pentru al aduce la un format general. Mai mult, unele tipuri de documente care contin elemente bogate(eng: rich text), imagini, o structura mai complicata, pot fi dificil de manevrat. Așadar, se detecteaza mai întâi extensia în care este primit fișierul, iar în funcție de situație se folosesc diferite biblioteci din python, potrivit formatului. Spre exemplu, vom folosi PDF Miner pentru parsarea unui document de tip PDF, și api-ul OS din python pentru a citi datele dintr-un fisier text simplu.

Rezultatul anterior va fi folosit ca parametru în cadrul serviciului JobProcessor. Acesta va prelua ca și parametrii din apelul efectuat în serviciul din Node.JS(nu exista un contact direct dintre frontend și motor, cel din urma serverste drept middleware în comunicare și functionalitate) și găsește limba în care a fost scris CV-ul. Acest proces este efectuat datorită nevoii de generalizare și de flexibilitate a sistemului. Unii utilizatori au scris resume-ul în romana, iar alții în engleza, aceștia netrebuind constransi la alegerea unui format standardizat, nici al formei structurii datelor, nici al limbii. După găsirea limbii de circulatie, se efectueaza o traducere a acestuia în funcție de caz: dacă este scris în romana, se genereaza o copie în limba engleza și invers. Ulterior, pentru fiecare loc de munca găsit în efectuarea interogarii asupra

bazei de date, se apeleaza serviciul de recomandare, în urma caruia se obtine pe lângă alte informații de interes, un scor de comparare. Pe baza acestei valori, job-urile sunt sortate în ordine cronologica(ofertele cu scor cât mai mare vor fi prioritare), iar tabela cu slujbe recomandate aferenta utilizatorului este actualizata. Aceasta stocheaza înformatiile în urma ultimei procesari, jucând un rol de memorie cache în cadrul sistemului nostru.

O diagrama functionala de asamblu a sistemului, în care sunt descrisi pasii care sunt urmați în timpul procesului de recomandare se afla în figura 3.3.

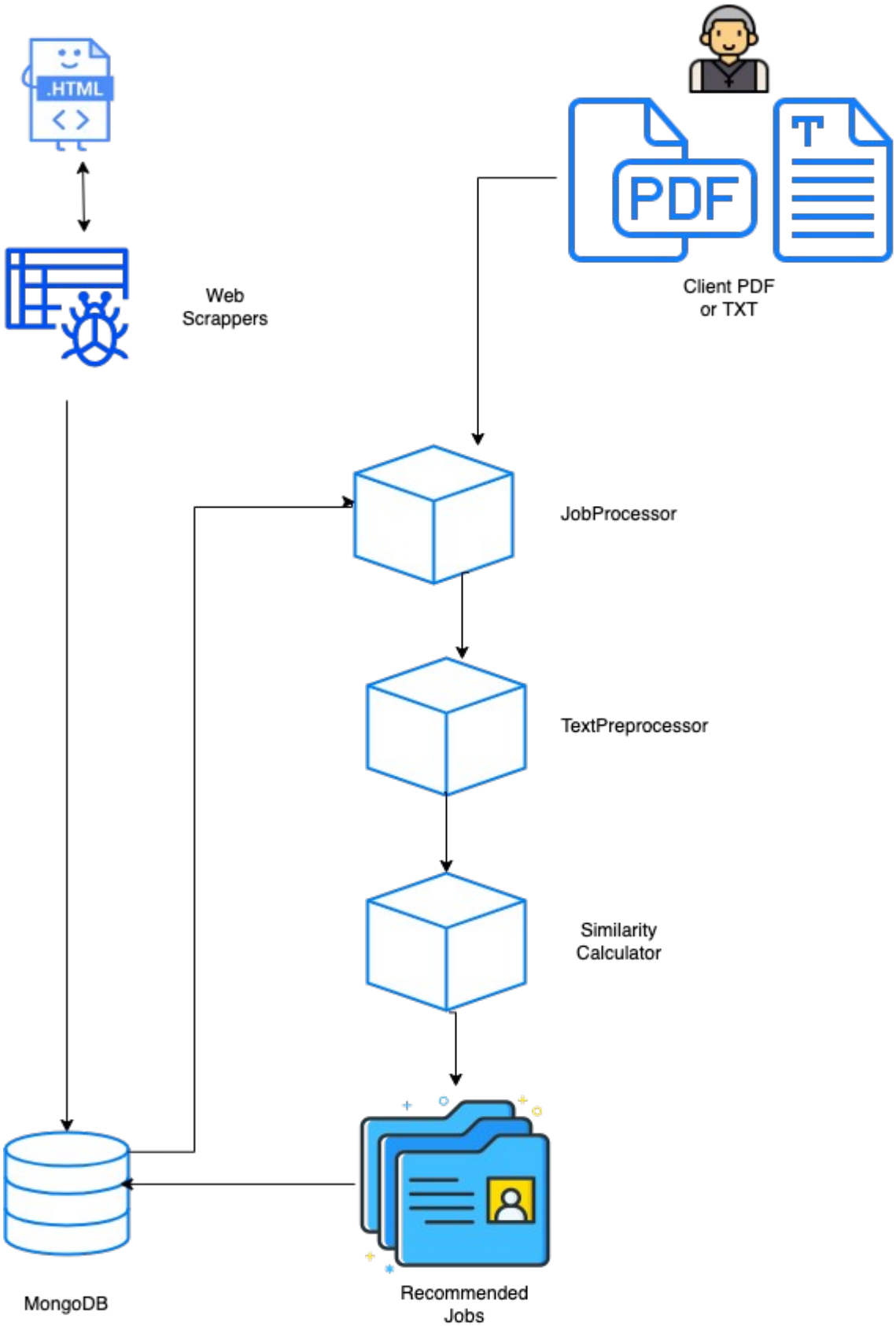


Figure 3.3: Diagrama functionala a procesului de recomandare

În cadrul procesului de funcționare a serviciului de recomandare exista doua etape diferite: preprocesarea și modelarea input-ului și găsirea unor scoruri de similaritate, ce servesc drept parametrii pentru functia de sortare și clasare.

După extragerea cv-ului și a descrierilor locurilor de munca într-un format procesabil, trebuie decisa limba de circulatie după care se efectueaza preprocesarea. Așadar, se detecteaza limba în care sunt scrise informațiile locului de munca și se decide dacă se folosește resume-ul în engleza sau în romana a utilizatorului, în cadrul procesarii. Dacă oferta este scrisa în limba engleza, se folosește CV-ul aferent și invers pentru o mai buna compatibilitate și similaritate în cadrul metodelor de analiza textuala.

Următoarea etapa deserveste preprocesarii și modelarii datelor. Aici au apărut mai multe dificultati în privința generalizarii procesului. S-a dorit o acoperire cât mai mare a industriei job-urilor și o funcționare cu orice tip de format de CV. Majoritatea oamenilor scriu informațiile din resume într-un mod foarte particularizat și cu tipare diferite. Unii oameni folosesc explicații și fraze bogate în semnificatii pentru se descrie pe ei sau experientele profesionale, iar alții prefer sa rezume cât de mult posibil acest proces, folosind doar câteva cuvinte cheie.

În vederea construirii unui model utilizabil într-o operație de comparare, au fost aplicate o serie de operație asupra textului. Ideea este transformarea acestuia într-o lista de cuvinte relevante(eng: tokens) și curatarea lui.

- În prima instanța se renunța la majuscula în cazul oricarui cuvânt pentru a nu se diferentia la nivelul comparatiei. Spre exemplu, cuvintele Ambitios și ambitios se reduc la aceeași semnificatie: ambitios.

- se înlocuiesc apostroafele scrise într-o forma diferita: ’ cu '.

- caracterele de tip newline (/n) se înlocuiesc cu un spațiu gol.

- se aplica operația de concatenare: cuvinte ca "P R O G A M A T O R" se inlocuieste cu "PROGRAMATOR".

- toate link-urile si hyperlink-urile se sterg din document.

- se reduc hashtags-urile, metiuni, punctuatii.

- prescurtarile sau alias-urile pentru specializarile academice se aduc intr-o forma standarizata, care va fi comuna si in oferta de munca si in CV(btech – bachelor of technology)

- am convertit abiltiati tehnice de programare care se scriu cu un alias aparte in forma lor lunga(C++ la cplusplus)

- toate caracterele non alfa-numerice si unicode s-au convertit la spatii(un emoji sau un caracter special nu este relevant in cadrul sistemului nostru)

- folosing libraria NLTK am extras o serie de cuvinte de legatura din fiecare limba (romana sau engleza) si le-am scos din input-uri. Cuvintele de legatura nu o au o semnificatie importanta in procesul de recomandare. Spre exemplu, atat intr-o oferta cat si intr-un resume se poate regasi cuvantul asadar, dar acesta nu confera nicio informatie de interes.

- se reduc toate spatiile albe

- In continuare, are loc procesul de lematizare a cuvintelor. Lematizarea este o tehnică de normalizare a textului utilizată în procesarea limbajului natural (NLP). Lematizarea reprezintă, în lingvistică, procesul de grupare a formelor flexionare ale unui cuvânt, astfel încât acestea să poată fi analizate ca o singură entitate, identificată prin lema cuvântului (forma sa de dicționar). În lingvistica computațională, lematizarea este procesul algoritmic de determinare a lemei unui cuvânt pe baza sensului intenționat. Spre deosebire de găsirea rădăcinii, lematizarea depinde de identificarea corectă a partii de vorbire și a sensului unui cuvânt într-o propoziție, precum și în contextul mai larg din jurul propoziției, precum propozițiile vecine sau chiar întregul document. Ca urmare, crearea de algorimti eficienți pentru lematizare este o arie de cercetare curentă. In cadrul acestui proiect operatia a fost folosita pentru a reduce cuvintele la forma lor de baza, in functie de contextul in care se afla. Astfel, acuratetea sistemului creste, nefacand diferenta la timpul sau maniera in care au fost scrise cuvintele. Exemplu de lematizare:

Intrare: “jump”, “jumps”, “jumped”, “jumping” → Iesire: jump = jump, jumped = jump, jumps = jump, jumping = jump.

O diagrama a intreg procesului de functionare se afla in Figura 3.4:

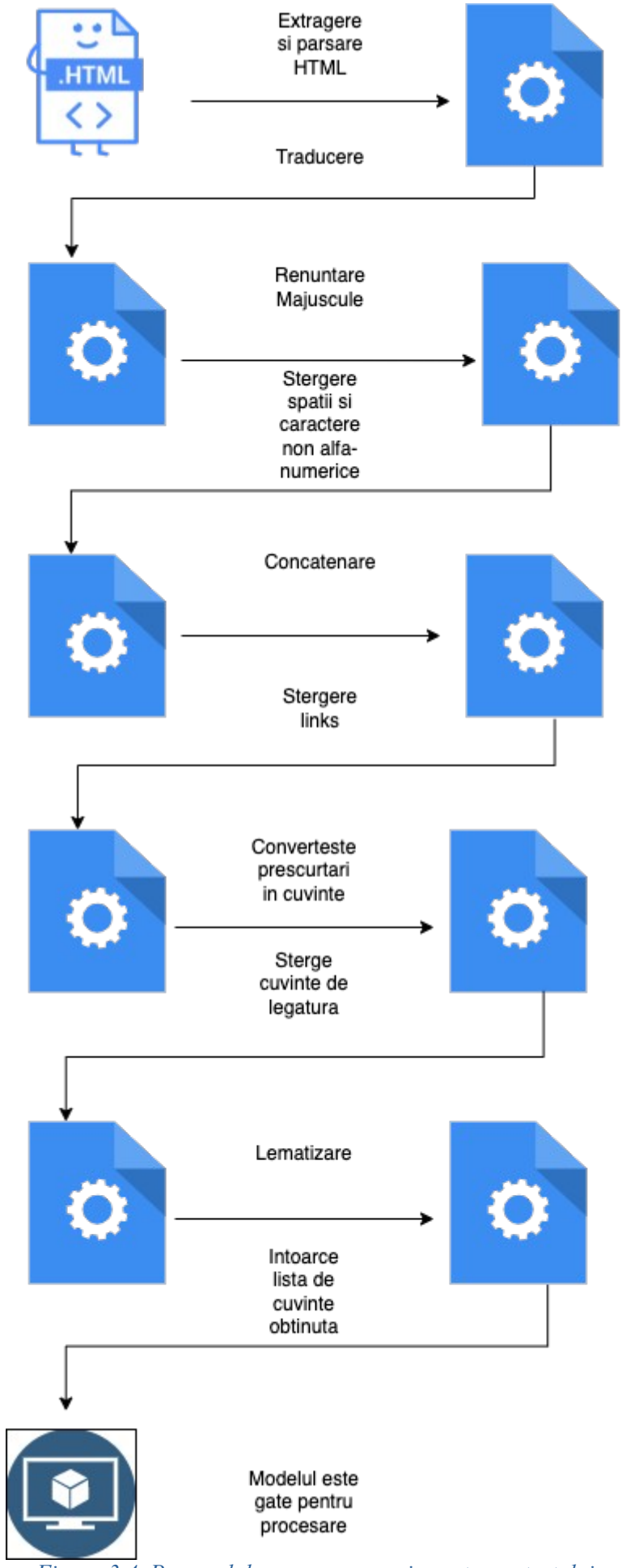


Figura 3.4. Procesul de preprocesare si curatare a textului

După etapa de preprocesare, se obțin modele cu informații de interes în procesul de gasire a similaritatii dintre un loc de munca și un resume. În vederea construirii formulei de comparație dintre cele doua entitati, s-a obținut o ecutatie formala a acesteia:

$$\text{sim}(cv, j) = \sum_{i=1}^{n} \text{simfun}_i(cv_i, j_i) \times w_i$$

Valoarea functiei sim(cv, j) este suma valorilor de similaritate ale diferitelor campuri de interes înmulțite cu ponderile lor corespunzatoare($simfun_i(cv_i, j_i)$ este corespondentul celui de al i-lea câmp). În cadrul sistemului nostru, în vederea compararii celor doua modele(resume și cel al ofertelor de munca) se iau în considerare patru variabile de interes: Similaritatea Cosinus, indexul Jaccard, numărul de cuvinte cheie comune dintre cele doua, locațiile comune și numărul comun de cuvinte dintre titlul job-ului și CV-ul angajatului. Valoarea rezultanta din sistemul nostru e suma produselor valorilor de similitudine ale tuturor campurilor de interes și ponderile aferente acestora. Fiecare dintre aceste funcții este introdusa în cele ce urmează:

Similaritatea Cosinus

O abordare comuna și ușor de impementat în comparatia documentelor pentru a determina similaritatea acestora pe baza de continut se bazează pe numarea numarului maxim de cuvinte comune. Cu toate acestea, aceasta soluție are un defect inerent. Pe măsura ce dimensiunea documentului crește, numărul de cuvinte comune tinde să fie din ce mai mare, chiar dacă textele abordeaza subiecte diferite. Putem avea suficiente cuvinte în comun, la corpusi care abordeaza teme complet diferite. Similaritatea cosinus intervine în solutionarea acestei rezolvari de tipul: număra cuvintele comune sau calculează distanța euclidiana.

Metrica mentionata în paragraful de mai sus este utilizata pentru a determina cât de asemănătoare sunt doua documente, indiferent de dimensiunea lor. Din punct de vedere matematic, similaritatea cosinus măsoară cosinusul unghiului dintre doi vectori proiectati intru-un spațiu multidimensional. În contextul de fata, cei doi vectori enuntati mai sus reprezintă matricelor de frecventa omoloage celor doua documente folosite în comparație: descrierea unui loc de munca și resume-ul utilizatorului. Avantajul acestei metode este ca atunci când un text este reprezentat într-un spațiu multidimensional(unde fiecare dimensiune corespunde unui cuvânt dintr-un document), similaritatea cosinus surprinde orientarea(unghiul) dintre cele doua documente, ci nu magnitudinea(metrica folosită în distanța euclidiana.). Spre exemplu, dacă doua documente sunt foarte indepartate în ceea ce privește dimensiunea(cuvantul facultate apare de 60 de ori într-un document și de 15 ori în altul), ele ar putea totuși avea un unghi mai mic când sunt reprezentate vectorial într-un spațiu ortogonal. Cu cât unghiul este mai mic, cu atât similaritatea este mai mare.

O reprezentare grafica a explicatiei de mai sus se găsește în Figura 3.5. Geometric vorbind, fiecare dintre vectorii desenati(Doc Dhoni_Small, Doc Sachin și Doc Dhoni) reprezintă plasarea într-un spațiu tridimensional a trei fragmente documente de câte trei cuvinte fiecare, unde scalarea pe fiecare dimensiune este reprezentata de numărul de aparitii a unui cuvânt în documentul respectiv. Din figura se observa, ca vectorul Doc Dhoni Small este mai apropriat de vectorul Doc Dhoni(unghiul dintre ele este mai mic), deci sunt mai similare. Desigur problema se poate generaliza ușor la o comparație dintre mai mulți vectori(mai multe documente) și mai mai multe cuvinte pentru fiecare(dimensiunea lor). Pentru ca nu putem reprezenta grafic un spațiu $n-dimensional$, se aplica în cadrul problemei de fata o generalizare a formulei de calculare a unghiului dintre doi vectori.

În aplicația de fata se folosesc urmatoarele formule generalizate:

Fie considerati doi vectori $V=(v_1, v_2, ..., v_n), W=(w_1, w_2, ..., w_n)$, atunci produsul lor scalar va fi egal cu: $v \cdot w=(v_1 \times w_1)+(v_2 \times w_2)+...+(v_n \times w_n)$, iar magnitudinea (modulul) unui

vector va fi egal cu: $\|V\|=\sqrt{v_1^2+v_2^2+...+v_n^2}$. În final, valoare cosinus va fi reprezentata de raportul dintre cele doua. Algoritmul este simetric, indiferent de cum se plaseaza vectorii aleși în formula.

Pentru a putea plasa un text în spațiul multi-dimensional, trebuie sa găsim transformare corespunzatoare a cuvinteor în numere. Cea mai simpla metoda, este generarea unei matrici de frecventa în care liniile reprezintă documentul curent, iar coloanele reprezintă cuvintele din fiecare text. La intersecția celor doua, se afla frecventa fiecaruia. Matricea rezultata, este considerată rara deoarece exista puține valori diferite de 0. Pentru realizarea operatiei de mai sus s-a folosit într-o abordare initiala CountVectorizer din sklearn.feature_extraction. Aceasta metoda, are totuși niste deficite: incapacitatea sa de a identifica cuvintele mai importante și cele mai puțin importante în analiza textului, considerarea cuvintelor cu o frecventa mai mare mai semnificative din punct de vedere statistic, incapacitatea de a indentifica vreo relație dintre cuvinte: precum similaritatea lingvistica. În ajutorul acestei probleme, am indentificat o metrica mai buna de vectorizare a textelor: TF-IDF. Modelarea statistica mentionata anterior, se bazează pe frecventa unui cuvânt dintr-un corpus, dar totodata ofera și o reprezentare numerica importantei unui cuvânt în cadrul analizei textului. Astfel, într-un corpus prioritatea este acordata cuvintelor de interes și frecventei de aparitie a unui termen. TFIDF se bazează pe o logica conform careia cuvintele prea abundente sau prea rare într-un text nu statistic importante în găsirea unui model din punct de vedere statistic. Formula pe care o folosește TFIDF în vectorizarea documentelor este prezentata mai jos:

$w_{i,j}=tf_{i,j}\times\log(\frac{N}{df_i})$, unde $w_{i,j}$ este ponderea unui termen i într-un document j, $tf_{i,j}$ este numărul de aparatii a lui i în j, N numărul total de documente dintr-un corpus și df_i numărul de documente care îl contin pe i.

Factorul logaritmic din formula penalizeaza matematic cuvintele care sunt prea rare sau prea frecvente, atribuindu-le scoruri scazute. In implementare, s-a folosit TfidVectorizer din aceeasi biblioteca. Mai pe matricea obtinuta prin vectorizare s-a aplicat formula pentru similaritatea cosinus si rezultatul s-a transformat in procente. Rezultatul statistic a fost bun pentru documentele bogate in fraze si cu termeni cheie bine folositi, totusi deficitul acestei metode este imposibilitatea gasirii unei similaritati lingvistice si semantice a textelor.

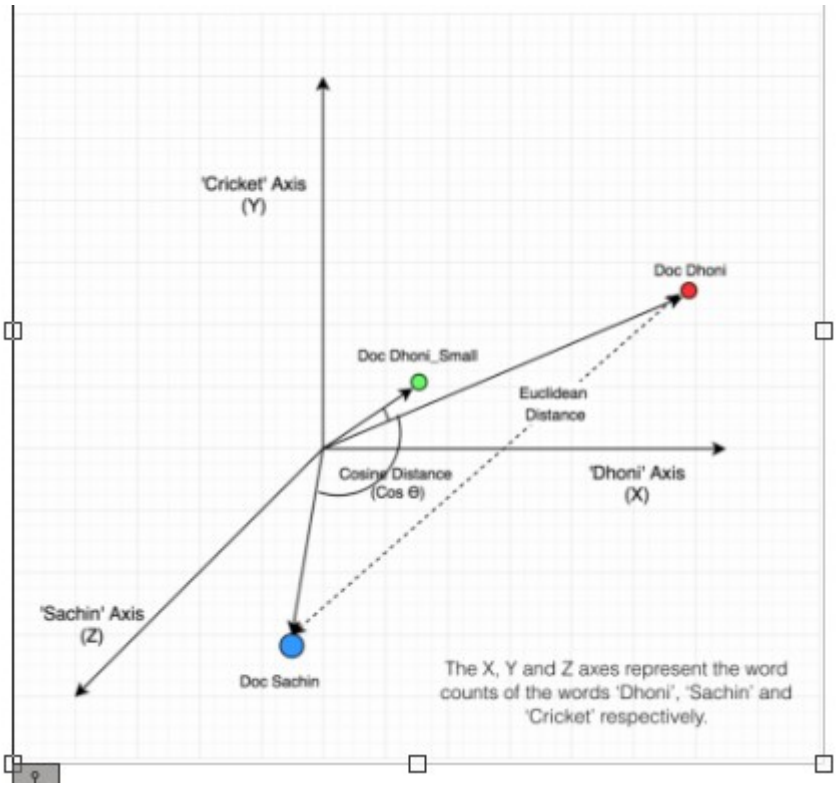


Figura 3.5. Plasarea intr-un spatiu 3-dimensional si unghiul dintre vectori

Indexul Jaccard

Similaritatea Jaccard mai este cunoscută și sub numele de indexul Jaccard sau Intersecție supra Uniune. Metrica de similaritate Jaccard este folosită în NLP pentru a determina cât de apropriate sunt doua documente de tip text din punct de vedere al continutului. Principiul de baza este calcularea numarului comun de cuvinte supra numărul total de cuvinte din corpus.

Reprezentarea matematica a indexului este:

$$J = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{(|A| + |B| - |A \cup B|)}$$, iar in cazul de fata aceasta poate fi scrisa ca:

$$J(doc_1, doc_2) = \frac{(doc_1 \cap doc_2)}{(doc_1 \cup doc_2)}$$, scorul rezultat apartinand intervalului 0, 1. Dacă doua documente sunt identice contextual rezultatul va fi 1, iar în caz contrar 0. Cele doua multimi vor fi reprezentate de listelele preprocesate de documente(atat în cazul resume-ului cât și în cazul CV-ului.)

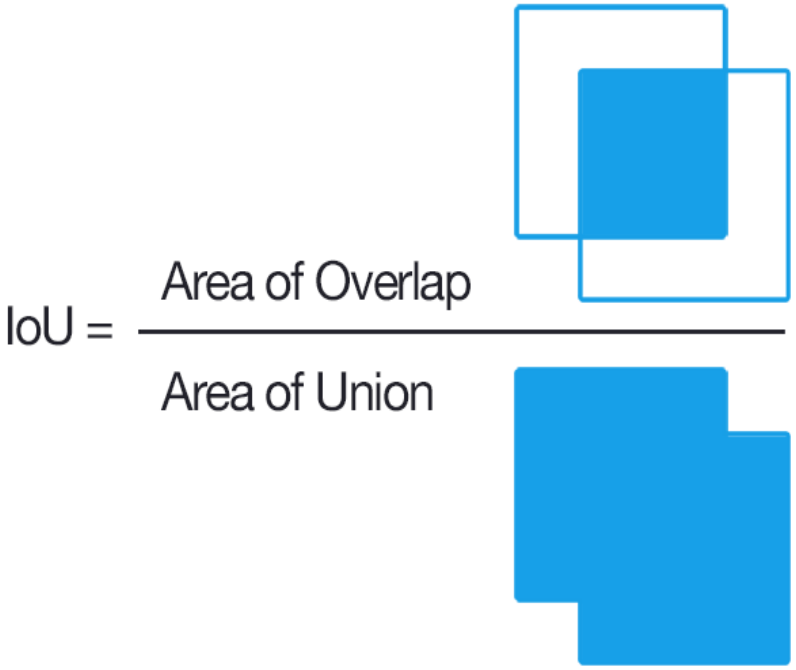


Figura 3.6: O reprezentare grafica a indexului Jaccard

Similitudinea Jaccard ia în considerare doar setul de cuvinte unice pentru fiecare document text. Acest lucru îl face să fie un bun candidat pentru evaluarea similitudinii documentelor atunci când repetiția nu reprezintă o problemă. Un exemplu a unei astfel de aplicații este compararea descrierilor de produse. De exemplu, dacă un termen precum "HD" sau "eficiență termică" este folosit de mai multe ori într-o descriere și doar o singură dată în alta, valorea distantei euclidiene sau a similaritatii cosinus ar scadea, acestea fiind construite pe baza spațiului vectorial și pe metrici de frecventa și aparatii a cuvintelor. Pe de alta parte, valorea indexului Jaccard rămâne neschimbata, acesta tinand cont doar de unicitatea cuvintelor. În cazul de fata, am folosit acest operator statistic pentru a pune în valoare și cazurile când descrierile locurilor de munca sau a resume-urile sunt foarte sumare și sunt expuse doar cuvinte cheie precum: c++, harnic, ambitios, java, nedorindu-se descalificarea și clasarea mai jos a acestor tipuri de documente. Așadar, nu vom tine cont de numărul de aparitii a unor cuvinte ci doar dacă se regasesc în ambele texte. Totodata, se dorește să se țină cont de descrierile cât mai detaliate ale locurilor de munca și CV-urilor, unde distanța cosinus are o performanța mai buna.

Procentul cuvintelor comune

Pentru a găsi o metrica de asemanare cât mai generala (luam în considere și cazurile în care descrierile locurilor de munca sau a candidatilor sunt foarte sumare), momente când tehnicile de vectorizare ne-ar da matrici foarte rare și n-ar performa foarte bine, rezultand într-un scor foarte mic. Pentru ca nu se dorește descalificarea situatiilor de genul în sistemul de nostru, s-a decis folosirea unei metrici care măsoară cât la suta dintre cuvintele dintr-un cv se afla în descrierea unui loc de munca. Totuși, acest procent va avea o pondere mica în formula de calcul, pentru ca scopul algoritmului nu este simpla intersectie dintre doua texte.

Cuvintele comune din titlu

În urma cercetarii și cautarilor asupra datelor, s-a constatat faptul ca exista multe oferte de munca cu o descriere foarte saraca, sumara(ex: Se cauta asistenta medicala cu experiența, răbdătoare, cu rezultate demonstrate.). În cazuri de acest gen, slujbele nu o să obțină un scop bun de clasare în pofida faptului ca utilizatorul a practicat și vrea sa practice fix aceasta meserie. Pentru a nu desconsidera aceste situații, s-a remarcat ca multe dintre aceste oferte au un titlu sugestiv, cu termeni cheie(ex: Caut asistenta șef cu experiența în Bihor/ Asistent medical generalist Iasi) și s-a calculat procentul dintre numărul de cuvinte comune dintre titlul descrierii și CV-ul utilizatorului. Fiind în mare parte cuvinte cheie, s-a observat ca clasarea are loc cu o precizie mai buna.

Locatii comune

Deoarece gama de arii profesionale pe platforma este foarte numeroasa și în cazul în care resume-ul unui utilizator este foarte sărac în informații și mai general(exista cazuri în care clientii aplicației sunt studenți fără experiența, iar intenția către o arie particulară de munca nu este bine specificata sau sugerata), scorurile pentru domenii specializate, care necesita experiența vor fi evident, mici. În genul asta de situații și deoarece s-a considerat ca un factor important în alegerea unui loc de munca este locația, s-a verificat dacă orașul/orasele prezente în oferta de munca se afla și în resume-ul utilizatorului. Astfel, putem atribui o valoare mai mare situatiilor: oraș în care s-au realizat studiile coincide cu orașul în care se afla locul de munca, orașul de provenienta este același cu al sediului companiei unde se efectueaza munca, dacă locația este remote aceasta poate fi luata în considerare, dacă este trecut și în cv-ul utilizatorului acest aspect etc.

Ponderile corespunzatoare rezultatelor functiilor de mai sus in formula de calcul a similaritatii dintre un job si un resume sunt:

$$Weights(X) = \begin{cases} 0.55 - HaveTheSameLocation,\ X = CosineSimilarity \\ 0.20,\ X = JaccardSimilarity \\ 0.15,\ X = WordsInTitle \\ 0.10\ If\ CosineSimilarity > 5\ else\ 0.02,\ X = Location \\ 1 - AllOfTheAbove,\ X = CommonWords \end{cases}$$

In cazul locatiei s-a luat o pondere care variaza in functie de distanta cosinus. In cazul unui cv incomplet, general, fara experiente anterioare de munca se doreste totusi sa se puna mai mult accent pe construirea similaritii pe baza contextului(Distanta cosinus) decat pe locatia in sine. Spre exemplu, in cazul unui resume sumar scris, general pentru un student din Cluj, nu se

doreste recomandarea tuturor locurilor de munca din Cluj indiferent de calificarea/expertiza ceruta. Celelalte ponderi sunt alese empiric, pe cale experimentala, punand accent pe similaritatea contextuala a textelor, prin atribuirea unei valori mai mari distantei cosinus cu vectorizare de tip TF-IDF. In continuare, prin atriburiea unei valori mari index-ului Jaccard consideram si ofertele care se bazeaza in special pe cuvinte cheie ci nu pe descrieri ample, urmat de numarul de termeni comuni dintre titlu si descriere. Mai apoi, urmeaza locatia si in final, cu cea mai putina relevanta, se afla procentul de cuvinte comune.

Prin varierea si diversificarea formulei de calcul s-a urmarit o abordare cat mai generala si mai putin particularizata pe anumite formate de descrieri/resume. Totodata, s-a incercat sa se tina cont si de semantica cuvintelor, nu doar o potrivire textuala a documentelor. Desigur, algoritmul functioneaza mai bine in situatiiile in care CV-urile si ofertele de munca sunt scrise in detaliu, cu accent pe termeni cheie si in aceeasi limba de circulatie, in aceste cazuri obtinandu-se un scor mult mai mare si mai precis de clasare.

3.3 Cronjoburi in Python

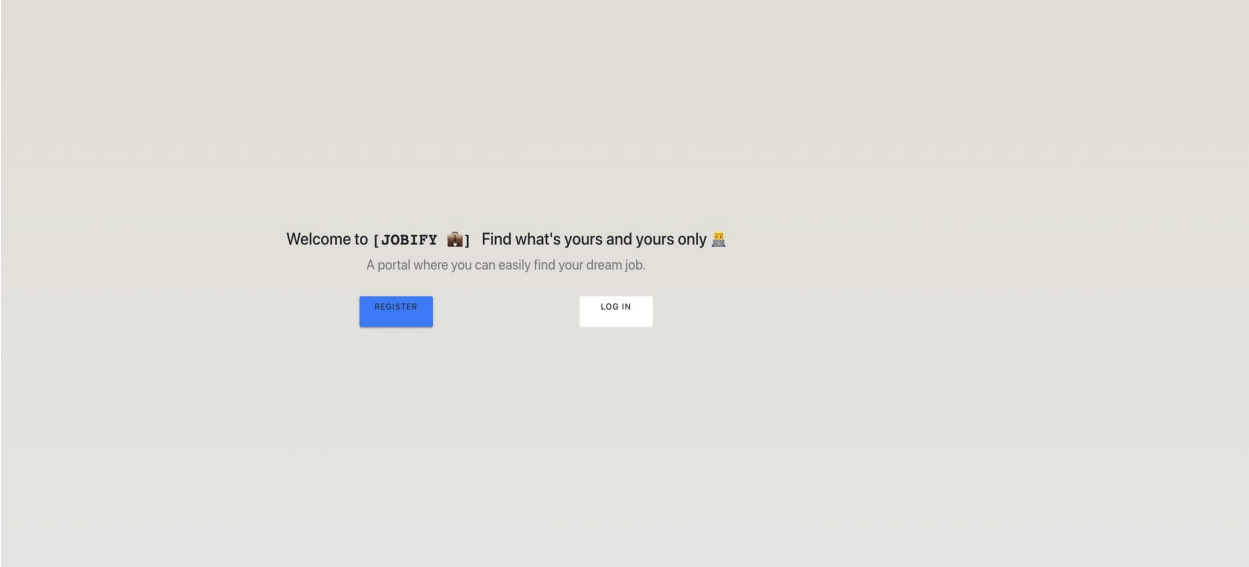
Un cron job este o comandă Linux utilizată pentru programarea sarcinilor care urmează să fie executate în viitor. În mod normal, aceasta este utilizată pentru a programa o operatie care se execută periodic - de exemplu intretinerea bazelor de date, curatarea de fisiere nedorite, trimiterea de email-uri la anumite ore etc. Procesele sunt definite intr-un crontab, fisier ce reprezinta locul in care sunt plasate comenzile ce urmeaza sa fie executate. Fiecare operatie din crontab incepe cu un sir de cinci simboluri semnificand minutul, ora, ziua, luna si ziua din saptamana in care se va executa procesul (* * * * *). In cazul aplicatiei noastre modulul cron se foloseste pentru a programa un daemon care se execute procesul de recomandare fiecarui utilizator nou inregistrat. Procesul se va executa periodic, la un interval de 30 de minute, pentru a pastra actualitatea si acuratetea rezultatelor. Codul prin care ii este atribuita o operatie de recomandare care se executa recurent fiecarui client nou este prezentat mai jos:

```
@app.route('/register-job-for-user', methods=['POST'])
def register_recommend_job_for_user():
    request_data = request.get_json()
    email = str(request_data["email"])
    pdf_path = str(request_data["resumePath"])

    python_path = "job-recommendation-engine/venv/bin/python"
    engine_path = "Recommendation-Engine/job-recommendation-engine" \
                  "/server-endpoint-engine.py"
    debug_path = "Recommendation-Engine/job-recommendation-engine" \
                 "/user-cronjobs-logs/cron_logs.txt"
    double_arrow = ">>"
        script_command = f"{python_path} {engine_path} {email} {pdf_path} {double_arrow}
{debug_path} 2>&1"

    with CronTab(user='silviuh1') as users_cron:
        new_user_job = users_cron.new(
            command=script_command,
            comment=email)

        new_user_job.minute.every(2)

    return f"[ {email} ]  JOB CREATED"
```

Capitolul 4. Interfata și functionalitati

În general, un portal de oferte de munca este o platforma web care permite utilizatorilor sa localizeze, sa vizualizeze și eventual sa aplice la slujbele dorite. Jobify reprezintă o platforma menita sa usureze procesul de căutare, documentare și aplicare pentru toți potentialii angajați, axata pe simplitate, eficiența, rapiditate menind sa ofere o interfata minimalista, dar intuitiva, astfel favorizand și utilizatorii mai puțin neexperimentati în folosirea tehnologiei sau navigarea site-urilor web. Partea vizuala a aplicației a fost proiectata cu scopul deschiderii și accesibilitatii acesteia de către un public cât mai larg: persoane tehnice și non-tehnice interesate de piața locurilor de munca și în căutare de o slujba noua. Interfata folosește o paleta redusa de culori: galben palid, verde de diferite nuante și tonuri, alb, accentul fiind pus pe minimalism și un bun contrast.

În deschiderea accesului la platforma, este afisata o pagina de start (Figura 4.1), simpla, intuitiva, ce conține butoane a carui scop este directionarea către pagina de inregistrare sau cea de logare.



Figua 4.1. Pagina principala a aplicatiei

Pentru a beneficia de functionalitatile oferite de aplicație, clientul are posibilitatea de a-si create un cont, altfel acesta nu poate fi directionat către pagina cu bord-ul locurilor de munca ofertate. În pagina de register(Figura 4.2) se regaseste un formular care trebuie completat cu atenție, campurile având validare atât pe frontend, cât și pe serverul de Node.js Utilizatorul trebuie să-și introduca un nume, o adresa de e-mail care servește drept identificator unic al utilizatorului, o parola care trebuie confirmată în câmpul următor, o descriere sumara și un CV care trebuie introdus cu ajutorul butonului Choose File. Cel din urma trebuie sa respecte un format: PDF sau txt.

Figura 4.2. Pagina de Register

1690

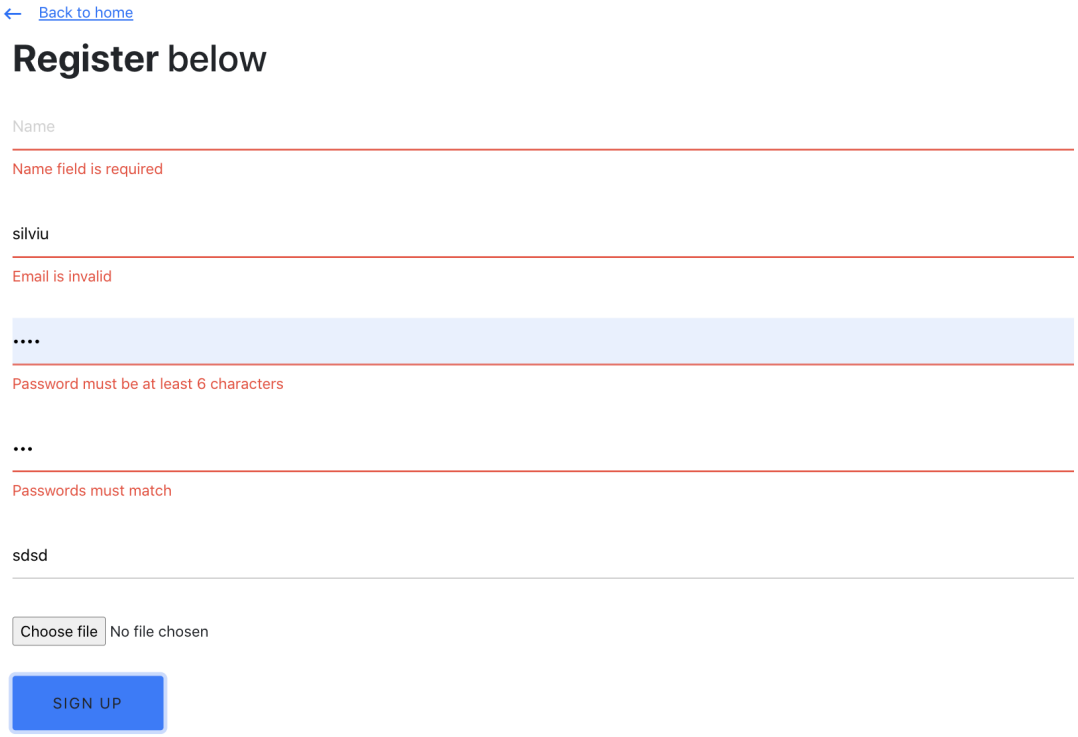


Figura 4.3. Validarile din pagina de inregistrare

După inregistrare, utilizatorul va fi redirectionat către pagina de login(Figura 4.4), unde se găsește un formular cu doua campuri: email și parola. După o logare cu credentialele introduse corect, utilizatorul va fi redirectionat către board-ul de job-uri, pagina care nu ar fi inaccesibila în urma unei autentificari eronate(prin mecanismul de autenficare realizat prin folosirea tokenului JWT, utilizatorul nu are acces la paginile private de pe website decât dacă este logat).

Figura 4.4. Pagina de Login

Pagina de vizualizare de oferte de munca este una simpla, a carei functionalitati sunt expuse în ecranul principal(Figura 4.5).

Cele doua butoane All Jobs și For reprezintă modul de vizualizare al job-urilor pentru utilizator. Astfel, dacă acesta dorește sa vadă toate slujbele, neclasate după vreun criteriu, acesta va apasa pe butonul din stânga și va derula în jos. Informațiile de interes sunt organizate în jobcards (Figura 4.7), aceste evidentiind numele slujbei, angajatorul, data la care a fost postat, locația, programul de munca(full/part time), o poza sugestiva, url-ul către oferta preluata de pe site-ul initial și doua butoane interactive. Prin apasarea butonului view, utilizatorul poate vizualiza și citi descrierea job-ului, iar prin butonul Apply, acesta va fi direct directionat către pagina ofertei pentru mai multe detalii și eventual pentru a aplica. Butoanele își schimba textul în mod dinamic(view – hide, apply - applied) petru a oferi o experiența cât mai intuitiva și feedback instant clientului.

Prin apasarea butonului For You, utilizatorul are posibilitatea de a vizualiza personalizat ofertele de munca care ii se potrivesc cel mai bine. Acestea sunt afisate crescător, în ordinea scorurilor de clasare. Toate ofertele de munca vor fi afisate paginat, clientul doar trebuind sa apase pe o pagina noua, accentul punandu-se pe rapiditate și evitand supraincarcarea vizuala. În centrul paginii se afla un searchbar interactiv(Figura 4.6) prin care utilizatorul poate cauta pe pagina curenta o oferta de munca în funcție de numele ofertei, angajator, data sau locatie. În colțul drept de sus al paginii, va fi afisat numele utilizatorului alături de un buton cu care acesta se poate deloga din aplicație. Dacă acesta închide prin alta modalitate platforma(prin butonul x sau ctrl+w pe google chrome) acesta va rămâne logat, procesul de autentificare nemaitrebuind să fie parcurs.

Figura 4.5. Pagina de vizualizare a ofertelor de munca

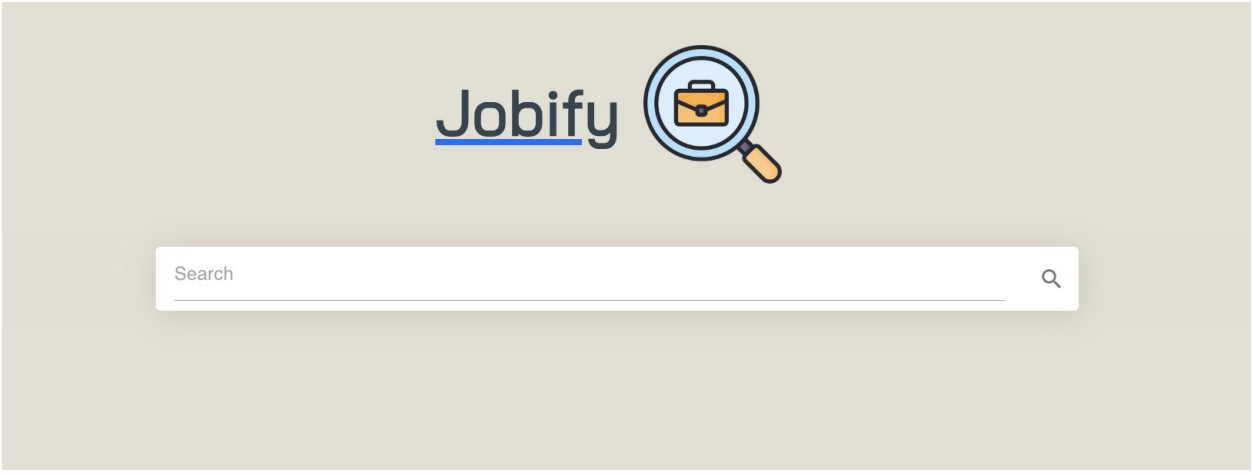


Figura 4.6. Bara de cautare

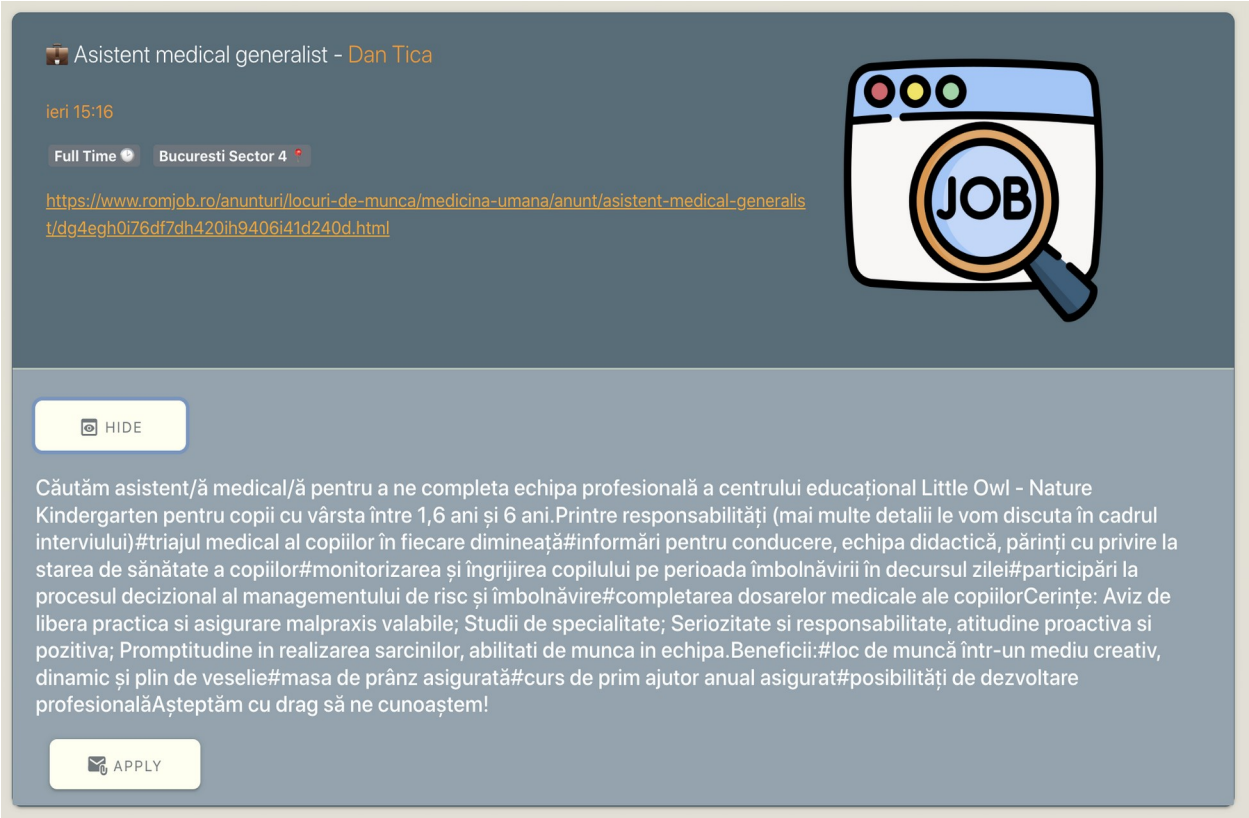


Figura 4.7. Afisarea unei oferte de munca