Universitatea Națională de Știință și Tehnologie POLITEHNICA Bucureşti
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Serviciu online pentru conversia documentelor Google Doc

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de
Inginer în domeniul *Electronică şi Telecomunicaţii*
programul de studii de licență *Reţele şi Software de Telecomunicaţii (ETTI – RST)*

Conducător(i) științific(i)
Ş.L.Dr.Ing. Radu LUPU

Absolvent
Delcea Andrei-Iacob

Anul 2025

Universitatea Națională de Știință și Tehnologie POLITEHNICA Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Program de studiu **RST**

TEMA PROIECTULUI DE DIPLOMĂ

a studentului **DELCEA R. Andrei Iacob, 444D**

1. Titlul temei: Serviciu online pentru conversia documentelor Google Doc

2. Descrierea temei:
Se va proiecta și se va implementa o aplicație web pentru conversia documentelor online din formatul Google Doc în formatul Pdf cu structura programabilă. Se va trata ca studiu de caz: orarul facultății. Cerințe generale de proiectare și implementare (caracteristici și funcții): portabilitatea interfeței utilizator, login bazat pe o bibliotecă de autentificare a utilizatorilor (la alegere), o tehnologie consacrată pentru baza de date. Documentele Pdf vor avea o structură definită de autor folosind sabloane Json și vor fi completate cu informații extrase dintr-un document online specificat de utilizator care este în format Google Doc. Pentru validarea soluției se va proiecta și se va implementa un prototip al sistemului peste o platformă reală de operare din familia Linux (la alegere). Se vor defini câteva scenarii de utilizare comune pentru testarea funcțională a sistemului. Rezultatele testelor vor fi prelucrate sub formă de tabele/grafice și apoi vor fi analizate. Opțional, se va evalua factorul QoE (ex. metoda MOS) gradul de specializare atins de către aplicație (ex. de metrici: complexitatea în utilizarea GUI de către utilizator, completitudine, fiabilitate). Activitatea presupune următoarele: cunoașterea noțiunilor de bază privind proiectarea și implementarea aplicațiilor web (arhitecturi software, baze de date, limbajele HTML, CSS, javascript, Python, Json, etc.); abilități de utilizare a serviciului Colab oferit de Google; cunoașterea limbii engleze la nivel mediu și disponibilitate pentru documentare și studiu individual folosind platforma de căutare Google; dorința de autoperfecționare.

3. Contribuția originală:
Va proiecta și se va implementa o aplicație web pentru conversia documentelor online din formatul Google Doc în formatul Pdf. Apoi, va efectua testarea funcțională a aplicației.

4. Resurse și bibliografie:
Note de curs PC, APC, POO, SDA, SO, RS, BD, SwTC

5. Data înregistrării temei: 2024-12-04 14:50:32

Conducător(i) lucrare,
Ş.L.Dr.Ing. Radu LUPU

Student,
DELCEA R. Andrei Iacob

Director departament,
Conf. dr. ing. Șerban OBREJA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod validare: **8cf70fb6d5**

Declarație de onestitate academică

Prin prezenta, declar că lucrarea cu titlul „Serviciu online pentru conversia documentelor Google Doc”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității Naționale de Știință și Tehnologie POLITEHNICA București, ca cerință parțială pentru obținerea titlului de Inginer în domeniul Electronică și Telecomunicații, programul de studii Rețele și Software de Telecomunicații (RST), este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

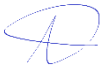
Declar că toate sursele utilizate, inclusiv cele de pe internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 11.06.2025

Absolvent,
Delcea Andrei-Iacob

Semnătura



_____

CUPRINS

LISTĂ DE ACRONIME

API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
BD	Baz de date
CSS	Cascading Style Sheets
GUI	Graphical User Interface
UI	User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JS	JavaScript
MOS	Mean Opinion Score
SQL	Structured Query Language
MySQL	My Structured Query Language
NLP	Natural Language Processing
PDF	Portable Document Format
QoE	Quality of Experience
AI	Artificial Inteligence
IA	Inteligență Artificială
REST	Representational State Transfer
RDBMS	Relational Database Management System
Redis	Remote Dictionary Server
LLM	Large Language Model
Llama	Large Language Model Meta AI
GGUF	Georgi Gerganov Unified Format
NER	Named Entity Recognition
REGEX	Regular Expressions
RESP	Redis Serialization Protocol
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart

LISTĂ DE FIGURI

1. INTRODUCERE

1.1. Scopul

Scopul principal al acestei lucrări este dezvoltarea unei aplicații capabile să extragă automat informații relevante din orarul universitar stocat în Google Docs, utilizând tehnologii moderne precum inteligența artificială, FastAPI și o bază de date rapidă, Redis. Lucrarea urmărește să propună o soluție eficientă de interpretare și structurare automată a datelor nestructurate, facilitând accesul organizat și rapid la informațiile din orar.

Prin utilizarea unui model de procesare a limbajului natural (NLP), proiectul își propune să identifice și să extragă automat elemente precum denumiri de materii, profesori, ore de curs, săli și grupe, transformând astfel conținutul unui document de tip tabelar într-o formă structurată, accesibilă prin intermediul unei interfețe programabile (API). Această funcționalitate este susținută de o arhitectură backend performantă, bazată pe FastAPI, și de o bază de date care permite stocarea, căutarea și manipularea eficientă a datelor extrase.

Lucrarea urmărește nu doar implementarea tehnică a acestei aplicații, ci și analizarea provocărilor întâmpinate în procesul de extragere automată a datelor din surse informale, precum și evidențierea avantajelor pe care automatizarea le poate aduce în mediul academic.

1.2. Motivația

Alegerea acestei teme a fost motivată de nevoia reală, întâlnită frecvent în rândul studenților, de a avea acces rapid, clar și centralizat la informațiile din orarul academic. În mod tradițional, orarele sunt distribuite în format Google Docs sau Excel, sub formă de tabele care necesită parcurgere manuală, fiind greu de procesat automat și susceptibile la erori de interpretare sau actualizare.

Această situație a generat ideea construirii unui sistem inteligent care să automatizeze extragerea și structurarea informațiilor din astfel de documente, oferind o bază de date coerentă, interogabilă și ușor de integrat în alte aplicații sau platforme web. În plus, utilizarea inteligenței artificiale în acest context oferă o oportunitate excelentă de aplicare practică a unor concepte teoretice învățate pe parcursul celor două cursuri de inteligență artificială întâlnite în facultate: ISIA și SIA.

Totodată, dezvoltarea unei aplicații cu tehnologii moderne precum FastAPI, împreună cu o bază de date și o componentă AI de procesare a datelor, reflectă tendințele actuale în industria software și oferă o experiență relevantă în construirea de soluții eficiente și scalabile. Tema aleasă îmbină atât aspecte de inginerie software, cât și provocări din zona inteligenței artificiale, ceea ce o face o alegere valoroasă atât din punct de vedere tehnic, cât și educațional.

2. TEHNOLOGII FOLOSITE

2.1. Limbaj Back-end – Python

Python

În ultimii ani, Python a devenit unul dintre cele mai populare limbaje de programare, fiind apreciat pentru simplitatea și versatilitatea sa [1], [2]. Creat de Guido van Rossum și lansat oficial în anii '90, Python a fost conceput pentru a pune accent pe lizibilitatea codului, oferind o sintaxă clară și ușor de învățat chiar și pentru cei aflați la început de drum în programare [1].

Unul dintre punctele forte ale acestui limbaj este adaptabilitatea sa într-o gamă largă de domenii, de la dezvoltarea de aplicații web și automatizarea sarcinilor repetitive, până la inteligență artificială, analiză de date și știința calculatoarelor [2]. Comunitatea activă și numărul mare de biblioteci disponibile transformă Python într-o unealtă puternică pentru dezvoltatori și cercetători deopotrivă.

În contextul actual al evoluției tehnologice, Python reprezintă un instrument esențial în rezolvarea problemelor complexe într-un mod eficient și intuitiv. Astfel, am ales Python pentru această lucrare pentru modul în care facilitează creerea de programe complexe, având anumite biblioteci de interes pentru acest proiect deja definite. Acesta excelează pe partea de inteligență artificială și va fii absolut necesar pe partea de procesare a informației.

Biblioteci de interes

- Beautiful Soup: Bibliotecă folosită pentru web scraping. Permite accesul facil la structura documentelor HTML/XML și extrage elemente după taguri, clase sau atribute [3].
- FastAPI: Framework pentru dezvoltarea de API-uri REST, bazat pe standardul OpenAPI si Pydantic pentru validarea datelor. Este foarte rapid, ușor de folosit și acceptă asynchronous programming, ceea ce îl face ideal pentru aplicații web moderne [4].
- Llama: Familie de modele lingivstice mari pentru inteligență artificială de tip open-source oferite de către Meta. Antrenate pe colecții de date extrem de mari, pot duce la bun sfârșit un mare număr de sarcini [5].
- Uvicorn: Uvicorn este un server ultra-rapid, ideal pentru a rula aplicații web asincrone construite cu FastAPI sau Starlette. Este compatibil cu aplicații care folosesc async/await, și permite rularea lor eficientă [6].

2.2. Limbaj Front-end – Comparatie ReactJS vs VueJS

ReactJS

ReactJS este o bibliotecă JavaScript open-source, utilizată pentru construirea interfețelor de utilizator dinamice și interactive. A fost dezvoltată de Facebook și lansată în 2013, având ca scop principal simplificarea procesului de creare a aplicațiilor web moderne, care necesită un flux de date bine organizat și o actualizare eficientă a componentelor vizuale [7].

Unul dintre conceptele cheie ale React este reprezentarea interfeței prin componente reutilizabile. Fiecare componentă reflectă o parte a paginii web și gestionează propria logică și stare, ceea ce face dezvoltarea mai modulară și mai ușor de întreținut. Totodată, React utilizează un mecanism numit Virtual DOM pentru a optimiza actualizarea paginii, reducând astfel costurile de re-randare și crescând performanța aplicației [8].

Datorită flexibilității și popularității sale, React este adesea folosit în combinație cu alte tehnologii moderne, precum API-uri REST, baze de date și framework-uri backend, pentru a construi aplicații full-stack robuste. Prin această abordare componentizată și reactivă, ReactJS se impune ca o soluție eficientă pentru dezvoltarea interfețelor web într-un mod scalabil și intuitiv [9].

VueJS

VueJS este un framework progresiv pentru construirea interfețelor web, dezvoltat de Evan You și lansat pentru prima dată în 2014 [10]. Deși este relativ tânăr în comparație cu alte tehnologii front-end, Vue a câștigat rapid popularitate datorită combinației dintre simplitate, flexibilitate și performanță [11].

Unul dintre avantajele definitorii ale VueJS este curba de învățare prietenoasă. Sintaxa sa este intuitivă, fiind ușor de înțeles pentru dezvoltatori care vin din zona HTML, CSS și JavaScript clasic. Vue permite crearea de componente reactive, care pot fi combinate pentru a forma aplicații complexe, fără a pierde din claritatea codului [11].

Arhitectura sa modulară oferă posibilitatea de a utiliza Vue ca o simplă bibliotecă pentru interfețe sau ca un framework complet, în funcție de nevoile proiectului. De asemenea, VueJS dispune de un ecosistem bine dezvoltat, incluzând instrumente precum Vue Router (pentru navigație) și Vuex (pentru gestionarea stării aplicației) [12].

Prin echilibrul dintre putere și ușurință de utilizare, VueJS oferă o soluție robustă pentru dezvoltarea aplicațiilor moderne, fără a impune constrângeri rigide asupra modului în care trebuie structurat proiectul [11].

Comparatie ReactJS vs VueJS

Deși VueJS pare a fi o soluție mai potrivită pentru proiectul acesta datorită simplității sale, și având în vedere dimensiule mici ale acestei lucrări, ReactJS încă este în acest moment principala tendință a lumii I.T. pentru front-end. Din acest motiv și datorită provocării suplimentare pe care le implică acest sistem diferit de abordarea tradițională, acest proiect va folosi ReactJS.

2.3. Interfață de programare aplicație – FastAPI

FastAPI

FastAPI este un framework modern pentru dezvoltarea de aplicații web și API-uri, construit pe standardul ASGI (Asynchronous Server Gateway Interface). Lansat în 2018 de Sebastián Ramírez, FastAPI a devenit rapid una dintre cele mai apreciate soluții pentru crearea de aplicații REST datorită vitezei ridicate, simplității în utilizare și suportului excelent pentru validarea automată a datelor [4].

Una dintre caracteristicile definitorii ale FastAPI este suportul nativ pentru programarea asincronă, ceea ce îl face potrivit pentru aplicații care gestionează multe cereri simultane, fără a compromite performanța. În plus, FastAPI utilizează tipurile de date din Python (cu ajutorul bibliotecii Pydantic) pentru a valida automat datele de intrare și a genera documentație interactivă, fără efort suplimentar din partea dezvoltatorului [4].

Prin integrarea facilă cu baze de date, sisteme de autentificare și tehnologii moderne precum machine learning sau microservicii, FastAPI oferă o bază solidă pentru dezvoltarea de aplicații scalabile și robuste. Este folosit atât în proiecte personale, cât și în aplicații de producție în companii mari, datorită eficienței și clarității sale [13].

Astfel, acest framework este foarte potrivit acestui tip de proiect datorită simplității sale, al caracteristicilor sale avansate care includ parsarea automata a fișierelor JSON, constrângeri implementate direct ori de către FastAPI, ori de către modelelor specifice folosite alături de acesta, oferite de Pydantic, și cel mai important viteza acestui sistem API, care face funcționarea aplicației mult mai plăcută pentru utilizator.

2.4. Bază de date – comparație MySQL vs Redis

MySQL

MySQL este un sistem de gestionare a bazelor de date relaționale (RDBMS), dezvoltat inițial de compania suedeză MySQL AB și, ulterior, achiziționat de Oracle Corporation. Este unul dintre cele mai utilizate sisteme de baze de date la nivel global, fiind open-source, stabil și bine documentat [14]. Datorită fiabilității și performanței sale, MySQL este o alegere comună pentru aplicații web, sisteme de management al conținutului, aplicații financiare sau platforme de comerț electronic.

Funcționarea MySQL se bazează pe modelul relațional, în care datele sunt organizate în tabele ce pot fi interconectate prin chei primare și externe. Limbajul standard de interogare utilizat este SQL (Structured Query Language), care permite operațiuni complexe de interogare, inserare, modificare și ștergere a datelor. MySQL oferă și mecanisme pentru protejarea integrității datelor, tranzacții și controale de acces [14], facilitând dezvoltarea unor aplicații robuste și sigure.

Un alt avantaj al MySQL constă în portabilitatea sa, fiind compatibil cu majoritatea limbajelor de programare (inclusiv Python, PHP, Java) și ușor de integrat cu framework-uri moderne precum FastAPI. Acesta permite scalarea aplicațiilor prin replicare, partajare și optimizare a interogărilor [15], fiind potrivit atât pentru aplicații mici, cât și pentru sisteme complexe distribuite.

Redis

Redis (acronim pentru REmote DIctionary Server) este un sistem de stocare a datelor în memorie, de tip key-value, dezvoltat iniți al de Salvatore Sanfilippo. Este recunoscut pentru viteza sa excepțională, fiind utilizat în principal ca soluție de cache, sistem de mesagerie, stocare temporară sau chiar bază de date primară în aplicații care necesită latență minimă [16].

Spre deosebire de sistemele relaționale tradiționale, Redis nu utilizează tabele și interogări SQL, ci se bazează pe un model simplu de date sub formă de chei și valori, unde valorile pot fi de mai multe tipuri: stringuri, liste, seturi, hash-uri sau structuri de tip sorted set [16]. Această abordare îl face extrem de rapid în accesarea datelor, mai ales atunci când este folosit pentru a păstra rezultate frecvent accesate sau sesiuni temporare.

Redis este un instrument esențial în arhitecturile moderne distribuite, fiind utilizat în combinație cu baze de date clasice pentru a reduce timpul de răspuns al aplicațiilor. De asemenea, oferă suport pentru replicare, persistență a datelor pe disc și publicare/subscriere (pub/sub), ceea ce îl face extrem de versatil în aplicații web, sisteme de notificare, aplicații în timp real sau machine learning [17].

Comparatie MySQL vs Redis

MySQL rămâne un sistem de bază în domeniul bazelor de date relaționale, cu un suport foarte bun pentru sisteme complexe, oferind consistență, integritate și persistență, pe când Redis este un sistem care favorizează cerințe mai simple, dar devine o necesitate atunci când este căutat un sistem rapid, cu latență minimă [14][16].

2.5. Inteligență artificială – Llama, Gemma, Deepseek, ChatGPT

Introducere în inteligența artificială

Inteligența artificială (IA) este un domeniu din informatică ce se ocupă cu dezvoltarea de sisteme capabile să reproducă anumite aspecte ale gândirii umane. Practic, IA permite calculatoarelor să „învețe" din date, să identifice tipare, să ia decizii și să îndeplinească sarcini care, până nu demult, puteau fi realizate doar de oameni [18]. Astăzi, întâlnim IA în tot mai multe aplicații – de la asistenți vocali și filtre anti-spam până la sisteme de recomandare sau traducere automată.

Tehnologiile din spatele inteligenței artificiale s-au diversificat în ultimii ani. Învățarea automată (machine learning), învățarea profundă (deep learning) și procesarea limbajului natural (NLP) sunt doar câteva dintre direcțiile principale. Un salt important a fost realizat prin apariția modelelor de limbaj de mari dimensiuni (Large Language Models – LLM), capabile să înțeleagă și să genereze text într-un mod foarte apropiat de cel uman. Aceste modele sunt antrenate pe cantități uriașe de informații și pot fi folosite într-o varietate de contexte – de la scriere automată de texte până la analiză de documente și cod sursă.

„Inteligența artificială nu va înlocui oamenii, dar oamenii care o folosesc o vor face." [29]
Acest citat al lui K. S. Rajasekaran afirmă că deși tehnologia bazată pe inteligență artificială nu are ca scop direct înlocuirea oamenilor, cei care învață să o folosească eficient vor avea un avantaj major. Practic, nu AI-ul în sine elimină locuri de muncă sau roluri, ci diferența va fi făcută de capacitatea oamenilor de a se adapta și de a colabora cu aceste tehnologii. Astfel, profesioniștii care integrează AI în activitatea lor pot deveni mai eficienți și mai competitivi decât cei care rămân la metode tradiționale.

„Inteligența artificială nu mai este doar un set de algoritmi. Este o schimbare profundă în modul în care abordăm rezolvarea problemelor și în felul în care oamenii interacționează cu tehnologia. Cu fiecare iterație, AI devine mai puțin un simplu instrument și mai mult un partener – un sistem care completează gândirea umană, ne anticipează nevoile și se adaptează complexităților lumii reale." [30]

În această secțiune a lucrării, voi analiza patru dintre cele mai cunoscute modele de IA dezvoltate în ultimii ani, pe care voi încerca să le adaptez proiectului meu: Llama (dezvoltat de Meta), Gemma (proiect Google), DeepSeek și ChatGPT (OpenAI).

Motivație folosire inteligență artificială

Motivul principal pentru care am ales să integrez inteligența artificială în acest proiect este dificultatea prelucrării automate a orarului facultății. Orarul este publicat în Google Docs sub forma unui tabel HTML, dar structura sa nu este una standardizată, ceea ce îngreunează interpretarea automată prin metode clasice de programare. De exemplu, identificarea corectă a legăturii dintre grupele de studenți și materiile aferente este adesea imprecisă atunci când se folosesc reguli fixe sau expresii regulate.

În acest context, folosirea unui model de inteligență artificială oferă o alternativă viabilă. Modelele de limbaj pot analiza conținutul HTML cu o mai bună „înțelegere" a contextului și pot extrage relațiile între entități (precum ore, grupe, săli și materii) într-un mod flexibil și adaptabil, chiar și atunci când datele sunt structurate neuniform.

Este adevărat că utilizarea AI presupune un cost în ceea ce privește timpul de procesare — algoritmii de

inteligență artificială au nevoie de mai multe resurse față de metodele clasice. Totuși, acest compromis este justificat prin creșterea fiabilității extragerii datelor și prin posibilitatea de a face față mai ușor modificărilor viitoare în formatul orarului. Astfel, AI devine un instrument esențial pentru a transforma un conținut dezorganizat într-un set de date coerent și ușor de folosit în aplicația finală.

Llama

Llama (Large Language Model Meta AI) este o familie de modele de limbaj dezvoltate de Meta AI, concepută pentru a oferi un echilibru între performanță și eficiență computațională. Aceste modele sunt open-source și au devenit populare în mediul academic și printre dezvoltatori datorită accesibilității și ușurinței de utilizare locală, fără a depinde de servicii cloud [5].

Pentru o primă evaluare am utilizat modelul Llama 3.2, cu o dimensiune de 3 miliarde de parametri. Acest model este compact și rulează local cu resurse hardware modeste, ceea ce îl face potrivit pentru testare și dezvoltare rapidă. Timpul de răspuns este redus, ceea ce permite rularea aplicației în mod eficient din punct de vedere al vitezei.

Cu toate acestea, în ceea ce privește acuratețea extragerii datelor din orarul facultății – o sarcină ce presupune interpretarea corectă a structurii HTML neuniforme – performanța modelului a fost modestă. În numeroase cazuri, modelul a generat rezultate incomplete sau incorecte, având o rată crescută de erori în identificarea corectă a legăturilor dintre grupe, materii și săli.

Gemma

Gemma este o serie de modele de limbaj dezvoltate de Google DeepMind, menite să ofere o alternativă open-source performantă la modelele comerciale precum ChatGPT. Modelele din această familie sunt compatibile cu formatul GGUF și pot fi rulate local prin infrastructura llama.cpp, ceea ce le face accesibile și pentru dezvoltatori individuali [19].

„Compania susține că este „cel mai bun model din lume care rulează pe un singur accelerator”, depășind concurența reprezentată de Facebook Llama, DeepSeek și OpenAI în ceea ce privește performanța pe un sistem cu un singur GPU, precum și având capabilități optimizate pentru rularea pe plăcile Nvidia și pe hardware dedicat pentru inteligență artificială. Encoder-ul vizual al Gemma 3 a fost de asemenea îmbunătățit, cu suport pentru imagini de înaltă rezoluție și pentru imagini non-patrate, iar noul clasificator de siguranță pentru imagini ShieldGemma 2 este disponibil pentru filtrarea atât a imaginilor de intrare, cât și a celor de ieșire, pentru conținut clasificabil ca fiind sexual explicit, periculos sau violent. Următoarea evaluare, am testat modelul Gemma 3, versiunea cu 27 de miliarde de parametri. Spre deosebire de LLaMA, acesta a oferit o precizie excelentă în identificarea detaliilor relevante din structura HTML a orarului – de la grupe și materii, până la ore și săli.” [34]

Cu toate acestea, din cauza dimensiunii mari a modelului și a resurselor hardware limitate disponibile pe sistemul meu, viteza de procesare a fost redusă. Acest lucru a dus la timpi de răspuns extrem de mari, impractici, ceea ce reprezintă compromisul care trebuie făcut pentru precizie maximă. În schimb, modelele Gemma cu mai putini parametrii s-au dovedit a fii mai rapide, dar imprecise.

Deepseek

DeepSeek este o suită de modele de limbaj cu capabilități avansate în înțelegerea și generarea de cod, dezvoltată pentru a concura cu modele specializate precum Codex sau CodeLlama. Proiectul pune accent pe performanță în sarcini legate de programare, dar poate fi utilizat și pentru procesarea limbajului natural [20].

„Deși aceste modele sunt predispuse la erori și uneori inventează propriile fapte, ele pot îndeplini sarcini precum răspunsul la întrebări, redactarea de eseuri și generarea de cod informatic. La unele teste de rezolvare de probleme și raționament matematic, ele obțin scoruri mai bune decât un om obișnuit.

Modelul V3 a fost antrenat la un cost raportat de aproximativ 5,58 milioane de dolari americani. Acest lucru este considerabil mai ieftin decât GPT-4, de exemplu, care a costat peste 100 de milioane de dolari pentru a fi dezvoltat." [32]

„Our core technical positions are mostly filled by people who graduated this year or in the past one or two years", a spus Liang, directorul general al Deepseek într-un interview. [33]

DeepSeek also claims to have trained V3 using around 2,000 specialised computer chips, specifically H800 GPUs made by NVIDIA. This is again much fewer than other companies, which may have used up to 16,000 of the more powerful H100 chips.

Din partea Deepseek, am utilizat modelul deepseek-code-v2, versiunea cu 16 miliarde de parametri. Modelul s-a comportat mai bine decât Llama în ceea ce privește precizia, dar nu a atins nivelul de acuratețe oferit de Gemma. Totuși, avantajul său a fost un compromis echilibrat între viteză și performanță – fiind mai rapid decât Gemma, dar oferind rezultate semnificativ mai precise decât Llama, ceea ce îl face o alegere viabilă pentru partea de parsare a proiectului.

ChatGPT

„ChatGPT și inteligențele artificiale generative au luat lumea pe sus. Ele au permis contactul direct și masiv al omenirii cu o tehnologie care era, anterior, mascată. Deciziile bazate pe învățare automată și date vaste (Big Data) sunt, de mult timp, în viața noastră, de la AutoCorrect sau algoritmii de recomandare de pe Netflix și Amazon, la selecția CV-urilor pentru un job sau interpretarea imagisticii medicale. Totuși, inteligențele artificiale rămăseseră în culise, inaccesibile publicului larg. În ultimele luni, cu mic cu mare, de la copil la bunic, am experimentat, ne-am jucat și ne-am supărat pe inteligențele artificiale într-un fel trăit, emoțional. Aceasta a schimbat, inevitabil, și dezbaterea publică. Suntem preocupați de viitorul școlii și al locurilor de muncă." [35]

„Sam Altman, CEO-ul OpenAI, a postat că modelul este „nativ multimodal," ceea ce înseamnă că modelul poate genera conținut sau înțelege comenzi în voce, text sau imagini. Dezvoltatorii care vor să experimenteze cu GPT-4o vor avea acces la API, care este la jumătate din preț și de două ori mai rapid decât GPT-4 Turbo, a adăugat Altman pe platforma X." [36]

Pentru comparație, am testat și ChatGPT, serviciul oferit de OpenAI, accesat prin interfața web și utilizând unul dintre modelele lor de ultimă generație. Avantajul major al ChatGPT constă în acuratețea

ridicată și stabilitatea răspunsurilor, modelul reușind să înțeleagă foarte bine contextul și să extragă corect informațiile relevante din orarul HTML, chiar și atunci când structura acestuia era dezordonată [21].

Totuși, spre deosebire de celelalte modele testate local, ChatGPT necesită conexiune la internet și implică dependența de un serviciu extern, ceea ce îl face mai puțin potrivit pentru implementări offline sau complet autonome. De asemenea, folosirea sa gratuită este limitată, iar integrarea într-o aplicație proprie poate ridica probleme legate de costuri sau politică de utilizare.

Chiar dacă pare soluția principală în proiectul meu, ChatGPT servește drept standard de comparație, oferind o referință valoroasă în ceea ce privește calitatea rezultatelor obținute prin inteligență artificială.

Named Entity Recognition (NER)

Pentru a structura automat conținutul orarului universitar extras în format HTML, am explorat și aplicat tehnici de recunoaștere a entităților denumite (Named Entity Recognition – NER) [22]. Scopul acestor metode a fost identificarea și extragerea elementelor relevante, cum ar fi grupele de studenți, orele, materiile, profesorii și sălile de curs, dintr-un conținut textual nestrcturat.

Inițial, am testat soluții bazate pe biblioteci clasice din ecosistemul Python, precum spaCy și Flair, configurate și antrenate cu seturi de date ad-hoc pentru a recunoaște structura specifică orarelor universitare [23]. Aceste metode s-au dovedit utile pentru recunoașterea entităților simple, cum ar fi numele de săli sau orele, dar au avut dificultăți în a face corelarea între entități (de exemplu: identificarea legăturii corecte între o grupă și materia aferentă în același rând al tabelului).

În paralel, am analizat și tehnici NER mai recente, bazate pe transformers, folosind modele pre-antrenate ca bert-base-cased sau roberta-base, prin intermediul librăriei Hugging Face [24]. Acestea au oferit performanțe mai bune în identificarea contextului, dar au necesitat resurse hardware mai consistente și timp semnificativ de configurare și adaptare la domeniul specific al orarelor academice.

Totuși, din cauza structurii inconsistene a tabelului HTML extras din Google Docs (lipsa de etichete semantice, date amestecate în celule, lipsa unei structuri ierarhice clare), modelele NER clasice s-au dovedit insuficiente în rezolvarea completă a problemei. Acest lucru a motivat tranziția către utilizarea unor modele de limbaj mari (LLMs) capabile să „înțeleagă” contextul global și să ofere o extragere mai robustă, în ciuda structurii neuniforme a datelor.

Astfel, tehnicile NER au fost utile ca punct de plecare și testare inițială, dar în contextul acestui proiect au fost depășite în eficiență și precizie de modelele AI moderne, precum DeepSeek sau ChatGPT, care combină în mod implicit recunoașterea entităților cu înțelegerea semantică a întregului document.

2.6. Regular Expressions – REGEX

Introducere în REGEX

Expresiile regulate (REGEX – Regular Expressions) reprezintă un instrument esențial în procesarea textului, utilizat pentru a identifica, extrage sau înlocui șiruri de caractere care respectă un anumit tipar (pattern) [25][26]. Sunt frecvent folosite în limbaje de programare precum Python, JavaScript, Java sau C++, pentru sarcini precum validarea datelor (e.g., adrese de email), extragerea informațiilor din texte complexe sau căutări rapide în fișiere [27][28].

La bază, o expresie regulată este un șir de caractere speciale care descriu un șablon. De exemplu, expresia \d{2}-\d{2} poate fi utilizată pentru a identifica un format de tip orar (ex: „09-10”), unde \d reprezintă o cifră și {2} indică faptul că se caută exact două cifre consecutive. Alte simboluri comune includ . (oricaracter), * (de zero sau mai multe ori), +, [], () și |[28].

În cadrul acestui proiect, REGEX este una din opțiunile viabile pentru etapa de preprocesare a textului extras din orarul HTML, cu scopul de a identifica automat orele, sălile, seriile sau grupele de studenți, în special în cazurile în care informațiile erau amestecate în aceeași celulă sau lipsea o structură HTML clară. De exemplu, o expresie precum \d{3}[A-Z]{1,2} a fost utilă pentru a extrage coduri de grupe precum „441Aa” sau „421B”.

Deși expresiile regulate sunt foarte eficiente în detectarea unor tipare fixe sau parțial variabile, ele devin greu de gestionat în scenarii cu multă ambiguitate sau cu variații de formatare. În cazul orarului facultății, REGEX poate fi folosit cu ușurință pentru a extrage grupele de studenți, dar nu are opțiunea de a la corela la restul informației din orar. Grupele sunt puse pe coloane, iar aceste coloane nu sunt bine definite și separate în formatul HTML.

Astfel, cu toate că REGEX ar putea oferi o eficiență foarte ridicată consumând foarte puține resurse și timp de execuție, nu are capabilitatea, cel puțin nu singur, de a duce sarcina la bun sfârșit.

2.7. Web Scrapping – Beautiful Soup 4

Introducere in web scrapping

Web scraping-ul este procesul automatizat de extragere a datelor din pagini web. Spre deosebire de navigarea manuală, unde utilizatorul accesează și copiază datele vizual, web scraping-ul presupune utilizarea unor programe software care accesează conținutul paginilor în mod programatic, îl parsează și extrag informația dorită în formate utile (precum JSON, CSV sau baze de date) [39].

Acest proces este esențial în multe domenii precum agregarea datelor, analiza pieței, inteligența artificială, procesarea limbajului natural, dar și în aplicații educaționale sau administrative, cum este cazul aplicației mele de extras orare.

În general, un algoritm de web scraping urmează următorii pași:
1. Accesarea paginii web – printr-un client HTTP (de obicei biblioteca requests în Python).
2. Descărcarea codului sursă HTML – adică conținutul brut al paginii.
3. Parsarea conținutului HTML – folosind o bibliotecă precum Beautiful Soup, lxml sau parsel.
4. Navigarea și extragerea datelor relevante – pe baza structurii HTML a paginii (folosind tag-uri, clase, ID-uri sau structura tabelară).
5. Prelucrarea și salvarea datelor – în formate ușor de utilizat ulterior (fișiere, baze de date, etc) [37].

Un avantaj major al web scraping-ului este că permite automatizarea sarcinilor repetitive, dar și colectarea de date din surse unde nu există API-uri oficiale sau export direct [40].

Totuși, trebuie menționat că web scraping-ul ridică uneori probleme de etică și legalitate. Multe site-uri interzic explicit acest tip de acces automatizat prin termenii și condițiile de utilizare sau prin fișiere robots.txt. De aceea, este important ca aplicațiile de scraping să fie concepute astfel încât să nu afecteze negativ performanța serverului și să respecte politicile site-urilor accesate.

Beautiful Soup 4

Pentru prelucrarea orarelor, salvate în format HTML, am utilizat biblioteca Beautiful Soup 4, un instrument scris în Python care permite parcurgerea, căutarea și extragerea de informații din documente HTML și XML. Această bibliotecă este foarte folosită în proiecte de web scraping datorită simplității sale și a integrării fluide cu alte biblioteci precum requests [37].

Funcționarea Beautiful Soup este centrată pe ideea de parsing a unui fișier HTML într-o structură de tip arbore, unde fiecare element (tag, text, comentariu etc.) devine un nod navigabil. Cu ajutorul metodelor find, find_all, select, sau children, putem accesa foarte ușor informațiile relevante, fără a fi nevoie de o cunoaștere avansată a manipulării DOM-ului [38].

În cazul aplicației mele, Beautiful Soup a fost esențială pentru a extrage conținutul celulelor tabelelor HTML în care se regăsesc activitățile din orar. Fiindcă tabelele conțineau taguri rowspan și colspan, a fost necesară o parcurgere atentă a fiecărui tr și td, cu tratament special al pozițiilor ocupate în matricea finală. Avantajul major al Beautiful Soup a fost că mi-a permis să gestionez aceste structuri complexe într-un mod clar și intuitiv.

2.8. Password Hashing – Bcrypt generator

Introducere in password hashing

În orice aplicație care gestionează conturi de utilizatori, securitatea parolelor este esențială. Una dintre cele mai bune practici este să nu se salveze niciodată parolele în format clar (plaintext), ci să se aplice o tehnică de hashing. Prin hashing, parola este transformată într-un șir de caractere aparent aleatoriu, folosind o funcție matematică ireversibilă [42].

Spre deosebire de criptare, care poate fi inversată (cu cheia potrivită), hashingul este unidirecțional: o dată ce parola a fost transformată, nu mai poate fi reconstruită din hash.

În Python, hashingul unei parole poate fi realizat cu mai multe biblioteci, cum ar fi hashlib, bcrypt sau passlib. O funcție de hash comună, precum SHA-256, generează un șir de lungime fixă dintr-o parolă de orice dimensiune. Totuși, utilizarea directă a funcțiilor ca SHA-256 nu este suficientă pentru siguranța parolelor. Acestea se calculează rapid deci, pot fi supuse cu ușurință atacurilor de tip brute force sau rainbow table [41].

De aceea, este recomandată utilizarea algoritmilor de hashing special concepuți pentru parole, precum bcrypt, scrypt sau argon2, care sunt intenționat lenți și folosesc „salt”-uri. Un salt este o valoare aleatoare adăugată parolei înainte de hashing, pentru a preveni atacurile care folosesc pre-calculări (ex: rainbow tables) [41].

Într-o aplicație web în Python, procesul tipic de hashing al parolelor include: generarea unui salt, aplicarea funcției de hashing, stocarea hashului în baza de date, împreună cu saltul (dacă nu este încorporat automat).

La autentificare, parola introdusă este hashu-ită din nou și comparată cu valoarea stocată. Această comparație trebuie să fie făcută în mod sigur, pentru a preveni atacuri prin timing.

Bcrypt generator

După ce am fost învățat cât de important este să nu păstrăm parolele în clar la materii precum „Baze de date” sau „Bazele criptologiei”, și că hashingul este soluția standard pentru a le proteja, următorul pas a fost alegerea unui algoritm potrivit. În cadrul aplicației dezvoltate, am optat pentru biblioteca Bcrypt din Python, una dintre cele mai sigure și recomandate opțiuni pentru protejarea parolelor utilizatorilor.

Bcrypt are câteva avantaje esențiale care l-au făcut alegerea naturală:

- Adaugă automat un salt unic pentru fiecare parolă, prevenind astfel atacurile de tip rainbow table (în care se încearcă potrivirea unor hashuri pre-calculate).
- Include un factor de cost (număr de runde), ceea ce înseamnă că procesul de hashing este intenționat mai lent, tocmai pentru a face atacurile de forță brută mult mai dificil de realizat.
- Este foarte simplu de integrat cu Python, folosind funcții clare pentru generarea hashului și pentru verificarea parolei [43][44].

3. PROIECTARE

3.1. Arhitectura software

Întreaga aplicație va fii împărțită în următoarele componente: Front-End, Api, 2 Motoare de procesare, Bază de date. Toate componentele for fii controlate centralizat de către API.

Front-End

Aceasta este interfața utilizatorului, bazată pe ReactJS. Este o interfață modernă, cu o pagină de start în care utilizatorii trebuie inițial să creeze un cont. Aceste conturi au ca scop ținerea la distanță a utilizatorilor nedoriți, și a suprasolicitării aplicației cu intenții malițioase, dar au și scopul de a putea salva un anumit orar în urma extragerii acestuia, și a-l asocia utilizatorului la cerere.

După accesarea contului deja existent, sau adăugarea unui cont în baza de date, utilizatorul are de introdus adresa web a orarului pe care vrea sa îl extragă și de selectat între cele 2 motoare de procesare ale aplicației, Regex sau AI.

După un timp, procesarea va fii gata iar utilizatorul va putea alege daca vrea sa stocheze datele extrase din orar, sau doar să le vizualizeze.

Există și un panou de utilizator constant în partea de sus dreapta a paginii, unde utilizatorul poate alege daca vrea să se delogheze, sau să vada informații despre contul său, incluzând orarele salvate. La aceesarea paginii de informații de utilizator se afișează datele acestuia, urmate de o afișare secvențială a orarelor salvate, controlată de două butoane: „Înainte” și „Înapoi”. Lângă acestea se află și un buton de ștergere a orarului vizualizat.

API

Partea de API a aplicației folosește FastAPI pentru timp minim de răspuns. Acesta va lega între ele toate celelalte componente ale aplicației, și le va controla centralizat. Primul rol al acestuia este de a comunica cu interfața grafică și de a-i răspunde la interogări. Mereu când un utilizator apasă un buton, introduce date sau accesează o pagină, API-ul va furniza informațiile necesare frontend-ului pentru ca acesta să funcționeze. În lipsa deținerii acestor informații, le va cere de la baza de date sau de la unul dintre cele două motoare de procesare.

Acest API trimite cereri către motoarele de procesare pentru generarea de date structurate JSON și gestionează comenzile de lucru cu baza de date Redis pentru a accesa conturile utilizatorilor, a stocare orare in cadrul acestora sau pentru a le încărca în memoria cache a aplicației. În ajutorul său, câteva modele pydantic pentru constrângeri și interpretarea cu ușurință a datelor, și câteva toolkit-uri, cu functii predefinite.

În cadrul lui, am folosit endpoint-uri de Get pentru a obține orarele din baza de date sau din motoarele de procesare, sau pentru a obține date despre utilizatori. Endpoint-uri de Post am folosit pentru a transmite către backend ori orare, ori utilizatori. Dar și pentru a transmite anumiți parametrii de căutare, de exemplu un nume de utilizator după care trebuie întoarse orarele extrase de acel nume de utilizator.

Am folosit si endpoint-uri de Delete, unul dintre ele mai puțin important, pentru ștergerea de utilizatori. Altul se ocupă de ștergerea orarelor din cadrul unui utilizator, care poate fi destul de folositor având în vedere ca pot exista orare de care utilizatorul nu mai are nevoie. Dar cel mai important dintre ele, pentru comoditatea lucrării cu acest proiect, și dezvoltării sale, endpoint-ul de ștergere a memoriei Cache. Foarte folositor pe parcursul testării acestui proiect, deoarece orice modificare făcută la modul de procesare s-ar fi putut observa abia dupa 24h, după expirarea memoriei cache.

Primul motor de procesare

Primul motor de procesare funcționeaza în python pe bază de Regex. Scopul acestuia este de a întoarce un răspuns cât mai rapid. Acesta poate avea o precizie mai scăzută, sau îi pot lipsi o parte din datele așteptate, dar are eficiență maximă. Se face o singură parcurgere a textului HTML a orarului, linie cu linie și sunt extrase câmpurile de interes și stocate structurat în format JSON, cu ajutorul modelelor pydantic.

Codul Regex caută datele după anumite forme definite pentru fiecare câmp în parte. Verifică numărul de caractere al unui câmp, numărul de litere mari și caută anumite cuvinte si caractere cheie. Problemele principale întâlnite sunt:

- Stabilirea începutului și sfârșitului orarului, atât pe orizontală cât și pe orizontală
- Selectarea rândurilor de interes din tabel, si ignorarea celor ce nu conțin informații despre grupe sau materii
- Stabilirea în cadrul cărei coloane se află cursorul parsorului la un anumit moment dat. Numărul coloanei este esențial deoarece pe coloane, în capul tabelului, se află grupele. Iar stabilirea pe ce coloană ne aflăm la un anumit moment nu este ceva intuitiv, pe un rând se pot afla 50 de celule, dar și numai 2, sau niciuna. Trebuie luate în calcul dimensiunile celulelor, și spațiul ocupat deja de pe rândurile anterioare de către celulele care ocupă mai multe rânduri
- Aflarea numărului de materii dintr-o singură celulă. În cazul a doua parități, se pot afla doua materii într-o singură celulă, cu doi profesori diferiți și două săli diferite. De asemenea în cazul opționalelor pot fi și mai multe materii într-o singură celulă, nedelimitate practic prin nimic, față de parități care sunt delimitate printr-un număr de „/”-uri.
- Hotărârea când o anumită informație despre o materie, de exemplu profesorul sau sala, lipsește, și nu este doar omisă de către algoritm.

Al doilea motor de procesare

Al doilea motor de procesare ar fi trebuit inițial să folosească un model de inteligență artificială rulat local, DeepSeek Coder V2. Acesta ar fi trebuit să parcurgă tot textul tot linie cu linie, dar necesită un

timp de așteptare foarte mare, de aproximativ 15 minute, având 16 miliarde de parametrii. Nu am reușit să obțin din el nici o precizie suficient de satisfăcătoare pentru timpul de așteptare.

Din lipsa resurselor hardware pentru a antrena suficient un LLM, cât și pentru a rula modelul efectiv, am ajuns să folosesc API-ul oferit de OpenAI pe baza de token-uri. După crearea unui cont și achiziția a numărului de 2 milioane de token-uri (fiecare token reprezentând teoretic un cuvânt de conținut) am pus în funcțiune acest API, folosind cel mai nou model al lor, „gpt4-0".

Folosind biblioteca „Instructor" pentru a obține de fiecare dată un output structurat JSON, și folosind modele pydantic pe post de structură, validare și chiar pentru oferire de context modelului de AI, am reușit să obțin o performanță foarte ridicată, cu precizie foarte mare a rezultatelor obținute. A fost necesar un prompt foarte explicativ și detaliat pentru a-i conferi context modelului gpt-4o alături de împărțirea rândurilor de la intrare pe batch-uri pentru a nu pierde din importanța fiecărui rând individual.

Astfel, parcurgerea listei de rânduri de tabel obținute din web scrapper s-a făcut pe batch-uri de câte 10 rânduri pentru extragerea eficientă a datelor, dar și pentru a limita numărul de conținut trimis deodata modelului gpt-4o. Aceste outputuri ieșite în urma fiecărui batch sunt concatenate și aplatizate, în structura unei liste de orare, pentru care se genereaza ulterior id-uri de la 0 la numarul de rânduri-1.

Baza de date

Baza de date Redis functionează ca și bază de date pentru stocarea utilizatorilor și a orarelor acestora, dar și pe post de cache.

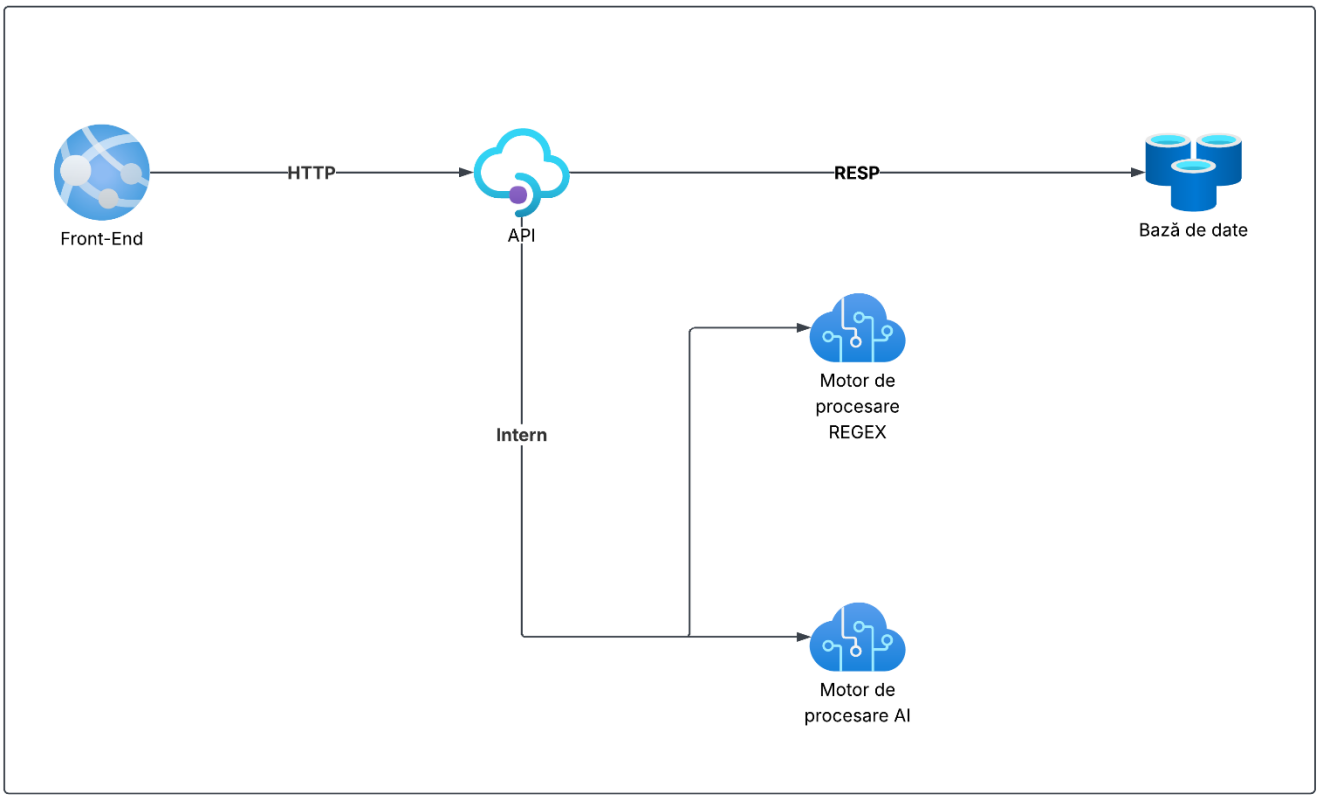
Stocarea utilizatorilor se face în urma completărilor în urma înregistrării în partea de interfață grafică și trimiterea datelor către API. Numele de utilizator, parola si adresă de mail sunt încărcate apoi în baza de date. Parola este transformată mai întâi în hash pentru a o securiza, folosind bcrypt.

Partea de caching salvează orarul la o cheie combinată din url-ul orarului și motorul de procesare prin care a fost obținut. Această memorie cache are o valabilitate de 24h pentru a reduce timpul de procesare necesar la accesările multiple ale aceluiași orar. Mărirea acestui timp în care orarul rămâne în cache va reduce resursele folosite per total, și timpul de procesare pe care vor trebuie sa îl aștepte utilizatorii. Dar odată cu mărirea sa, apare problema actualizării orarului original. Orice actualizare nu va putea fii vizualizată înainte de expirarea memoriei cache. În cazul de față, am considerat că o zi este un timp potrivit.

Baza de date Redis este stocată prin intermediul Redis Stack Server pe WSL, subsistemul Linux din cadrul Windows. Utilizatorii, cât și Orarele, sunt stocate ca JSON sub formă de string, și li se atribuie o cheie specifică după care pot fi găsite cât mai rapid. Utilizatorii au pe post de cheie numele care este și identificatorul lor unic. Toate căutările de utilizatori se fac după nume, aproape instantaneu, cu o complexitate de doar O(1). Pentru orarele utilizatorilor a fost folosite sub formă de cheie combinația de nume utilizator, tip de procesare și URL. Atunci când un utilizator regenerează un orar deja generat de către el, se suprascrie pentru a nu ocupa spațiu inutil și a suprasolicita baza de date. Cât despre cheia orarelor în cache, este formată din tipul de procesare și URL, pentru ca să poată exista în permanență un răspuns diferit pentru fiecare tip de procesare.

3.2. Diagrame de proiectare

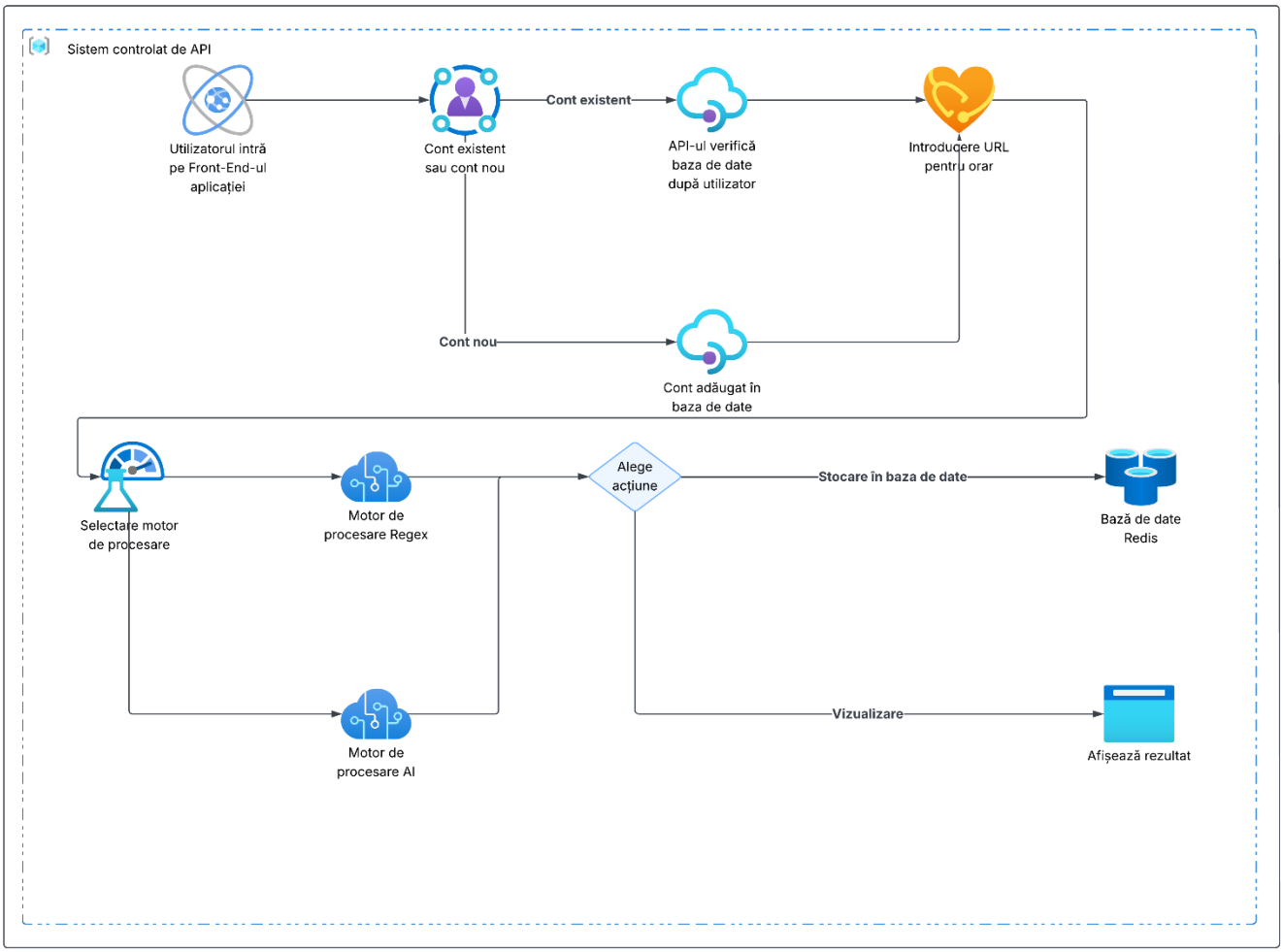
Diagrama de componente



Figură 3.2.1 - Diagrama de componente

În această diagramă se pot urmări cele 5 componente distincte ale aplicației, fiind conectate centralizat către API. Cele două motoare de procesare rulează pe aceeași mașină, și în același limbaj (python) cu API-ul, deci comunicarea lor este una internă. Componenta de Front-End comunică cu API-ul prin intermediul HTTP (Hypertext Transfer Protocol), iar baza de date Redis prin intermediul protocolului RESP (Redis Serialization Protocol).

Diagrama de interacțiune a utilizatorului și fluxul de procesare în aplicație



Figură 3.2.2 - Diagramă de activitate care include interacțiunile utilizatorului și fluxul de procesare al aplicației

Această diagramă de activitate ilustrează succesiunea de acțiuni pe care un utilizator le efectuează în cadrul aplicației, începând cu accesarea interfeței front-end, autentificarea sau înregistrarea, introducerea unui URL, selectarea motorului de procesare și, în final, alegerea între salvarea rezultatului în baza de date sau simpla vizualizare a acestuia. API-ul controlează fluxul și interacționează cu restul componentelor.

4. IMPLEMENTARE

Voi prezenta în acest capitol mai întâi elementele de bază ale aplicației, adică modele de date, tool-uri (adică fișiere cu funcții ajutătoare). Acestea vor fi folosite pe tot parcursul proiectului. Pe urmă voi discuta în ordine despre toate fișierele principale care se ocupă de backend, adică voi vorbi despre extragerea datelor, procesarea lor și apoi despre API. La final voi prezenta frontend-ul.

4.1. Modele de date

Pentru realizarea acestui proiect am folosit modele de date pydantic, care oferă constângeri și structură datelor cu care lucrez. Acestea sunt de ajutor la trimiterea datelor de la o componentă a aplicației la alta.

Cele mai importante modele de date sunt cele din fișierul ModeleActivitati.py, aici se află modele de date care se ocupă de activități și de orar. Prima dată voi discuta despre modelele de tip enumerație din acest fișier, care vor ajuta la construirea modelelor principale de date.

Modelul Zile conține zilele săptămânii.

```
class Zile(Enum):
    """Zilele saptamanii"""
    LUNI = "Luni"
    MARTI = "Marți"
    MIERCURI = "Miercuri"
    JOI = "Joi"
    VINERI = "Vineri"
    SAMBATA = "Sâmbătă"
    DUMINICA = "Duminică"
```

Modelul Categorie conține categoriile de materii.

```
class Categorie(Enum):
    """Categoria activitatii"""
    CURS = "Curs"
    SEMINAR = "Seminar"
    LABORATOR = "Laborator"
    PROIECT = "Proiect"
```

Modelul Paritate conține tipurile de paritate pe care apare materia, pară, impară, sau ambele.

```
class Paritate(Enum):
    """Categoria activitatii"""
    PAR = "Par"
    IMPAR = "Impar"
    AMBELE = "Ambele"
```

Toate aceste 3 modele sunt folosite pentru a valida câmpuri din modelul Activitate, care conține o instanță a unei materii în orar, alături de diverse date despre ea care au putut fi extrase: un id pentru identificare, nume, profesor, sală, zi, interval, durată, grupe, anul, categorie, paritate.

```
class Activitate(BaseModel):
```

```
    """Structura unei activitati"""
    id: int = Field
(description="Id-ul activitatii, incepe de la 0 si se incrementeaza")
    nume: str = Field
(description="Numele materiei. Exista materii cu nume mai lungi, dar si cu nume
scurte, ex: Programarea Calculatoarelor și Limbaje de Programare 2 (curs), SEP, ASC,
EIM, IM, G3D. De asemenea, materiile pot fi impartite si prin | sau || daca sunt mai
multe intr-o singura celula, caz in care acestea se desfasoara in paralel si sunt in
general optionale.")
    profesor: str = Field
(description="Profesorul care se ocupa de activitate, numele lui incepe cu prefixele
Prof., sau Profesor, sau Prof, prefixele nu vor fii pastrate la nume, profesorul
poate lipsi din celula, caz in care pui -")
    sala: str = Field
(description="Sala in care se desfasoara activitatea, de forma B303, sau A01 sau
Sala Orange, sau B208a. Apare ori in celula activitatii, ori in dreapta, ori deloc,
caz in care se completeaza cu -")
    zi: Zile = Field
(description="Ziua saptamanii in care se desfasoara activitatea, apare la
inceputulfiecarui rand de tabel")
    interval: str = Field
(description="Intervalul orar in care se desfasoara activitatea, se ia ora de start
de la inceputul randului de tabel si se aduna numarul de randuri ale celulei")
    durata: int = Field
(description="Durata activitatii, trebuie sa se potriveasca cu intervalul orar")
    grupe: List[str] = Field
(description="Grupele care participă la activitate, apar in primul rand de tabel
celulelor destinate. Sunt de forma 441Da, 423Bb.")
    anul: int = Field
(description="Anul de studiu, se extrage din numele uneia dintre grupele care
participa la activitate. A doua cifra este chiar anul de studiu. e.g. din 432Ab se
extrage anul egal cu 3")
    categorie: Categorie = Field
(description="Categoria activitatii (curs,seminar,laborator sau proiect, prescurtate
uneori (c),(s),(l),(p))")
    paritate: Paritate = Field
(description="Paritatea activitatii (par sau impa, sau ambele). Daca in continutul
celulei apare un /, atunci ce este la stanga de / este impar, iar ce este la dreapta
este par. Altfel in lipsa unui / ca si paritate avem Ambele")
```

Toate câmpurile au descrieri detaliate pentru a ajuta în procesarea modelului gpt-4o, care verifică inclusiv descrierile modelului pydantic.
Se pot observa tipurile de date ale field-urilor fiind ori string sau integer pentru tiprui de data mai simplu, sau Zile, Paritate și Categorie pentru câmpuri care au nevoie de această validare suplimentară.

Urmează modelul de utilizator, care stocheaza datele unui utilizator (nume, parola, email).

```
class Utilizator(BaseModel):
    nume: str = Field(description="Username-ul utilizatorului")
    parola: str = Field(description="Parola hashata a utilizatorului")
    email: EmailStr = Field(description="Adresa de email a utilizatorului")
```

Apoi modelele de cereri, care iau forma diverselor cereri pe care le trimite frontend-ul către API.

```
class CerereProcesare(BaseModel):

    url: str
```

```
    procesare: Procesare
    nume_utilizator: str


class CerereLogin(BaseModel):
    nume: str
    parola: str


class CerereRegister(BaseModel):
    nume: str
    parola: str
    email: EmailStr


class CerereStergereOrar(BaseModel):
    nume_utilizator: str
    procesare: Procesare
    url: str
```

Aceste modele de cereri preiau cu ușurință datele din partea frontend-ului, și extrag fix câmpurile de care au nevoie, fiind compatibile cu structura JSON a cererilor frontend-ului.

Nu în ultimul rând, modelele precompilate Regex. Acestea iau diverse tipare Regex pentru a identifica șiruri de caractere potrivite unui anumit tipar. De exemplu pentru grupe, se caută șiruri care încep cu 3 cifre, sunt urmate de o literă obligatoriu, și potențial încă o literă (pentru subgrupă). Pot avea în față și în spate între 0 și 2 spații libere.

```
import re


#Modele regex precompilate
model_grupe = re.compile("^[ ]{0,2}[0-9]{3}[A-Za-z][A-Za-z]{0,1}[ ]{0,2}$")


modele_zile = []
modele_zile.append(re.compile("^[ ]{0,2}[Ll][Uu][Nn][Ii][ ]{0,2}$"))
modele_zile.append(re.compile("^[ ]{0,2}[Mm][Aa][Rr][TtȚț][Ii][ ]{0,2}$"))
modele_zile.append(re.compile("^[ ]{0,2}[Mm][Ii][Ee][Rr][Cc][Uu][Rr][Ii][
]{0,2}$"))
modele_zile.append(re.compile("^[ ]{0,2}[Jj][Oo][Ii][ ]{0,2}$"))
```

```
modele_zile.append(re.compile("^[ ]{0,2}[Vv][Ii][Nn][Ee][Rr][Ii][ ]{0,2}$"))

modele_zile.append(re.compile("^[ ]{0,2}[Ss][AaÂâ][Mm][Bb][AaĂă][Tt][Ăă][
]{0,2}$"))

modele_zile.append(re.compile("^[ ]{0,2}[Dd][Uu][Mm][Ii][Nn][Ii][Cc][AaĂă][
]{0,2}$"))
```

Acesta este un model care caută cuvintele din header de forma „Sală”, „sala”, „SALA” sau derivate.

```
model_sala = re.compile("^[ ]{0,2}[Ss][Aa][Ll][AaĂă][ ]{0,2}$")
```

Iar aici sunt adăugate mai multe modele pentru săli efectiv. Primul model detectează săli de tipul „B02c”, „A304”, iar al doilea caută șiruri cu cuvântul „sală” în ele: „Sală Orange”.

```
modele_sali = []

modele_sali.append(re.compile("^[ ]{0,2}[BbAa][0-9]{1,4}[A-Za-z]{0,1}[ ]{0,2}$"))

modele_sali.append(re.compile("^[ ]{0,2}[Ss][Aa][Ll][AaĂă]"))


modele_sala_continut = []

modele_sala_continut.append(re.compile("([BbAa][0-9]{1,4}[A-Za-z]{0,1})"))

modele_sala_continut.append(re.compile(r"\b[Ss][Aa][Ll][AaĂă][ ]{0,2}\w+"))


model_activitate = re.compile("[A-Za-z]{2,}")


model_ore = re.compile("^[ ]{0,2}[0-9]{0,1}[0-9][-][0-9][0-9]{0,1}[ ]{0,2}$")


modele_categorie_curs = []

modele_categorie_curs.append(re.compile(r"\(+[ ]{0,2}[Cc][Uu][Rr][Ss][ ]{0,2}\)+"))

modele_categorie_curs.append(re.compile(r"\(+[ ]{0,2}[Ll][Ee][Cc][Tt][Uu][Rr][Ee][
]{0,2}\)+"))

modele_categorie_curs.append(re.compile(r"\(+[ ]{0,2}[Cc][ ]{0,2}\)+"))

modele_categorie_seminar = []

modele_categorie_seminar.append(re.compile(r"\(+[
]{0,2}[Ss][Ee][Mm][Ii][Nn][Aa][Rr][ ]{0,2}\)+"))

modele_categorie_seminar.append(re.compile(r"\(+[ ]{0,2}[Ss][ ]{0,2}\)+"))

modele_categorie_laborator = []
```

```
modele_categorie_laborator.append(re.compile(r"\(+[
]{0,2}[Ll][Aa][Bb][Oo][Rr][Aa][Tt][Oo][Rr][Yy][ ]{0,2}\)+"))

modele_categorie_laborator.append(re.compile(r"\(+[
]{0,2}[Ll][Aa][Bb][Oo][Rr][Aa][Tt][Oo][Rr][ ]{0,2}\)+"))

modele_categorie_laborator.append(re.compile(r"\(+[ ]{0,2}[Ll][ ]{0,2}\)"))

modele_categorie_proiect = []

modele_categorie_proiect.append(re.compile(r"\(+[
]{0,2}[Pp][Rr][Oo][Jj][Ee][Cc][Tt][ ]{0,2}\)+"))

modele_categorie_proiect.append(re.compile(r"\(+[
]{0,2}[Pp][Rr][Oo][Ii][Ee][Cc][Tt][ ]{0,2}\)+"))

modele_categorie_proiect.append(re.compile(r"\(+[ ]{0,2}[Pp][ ]{0,2}\)+"))


model_optional = re.compile(r"(?=\s*[Oo]([Pp][Tt][.\s]*\d+|[1-9]\.*(?=\W|$)))")



model_profesor = re.compile(r"(Prof\.?|Profesor|Professor)\s?([A-Za-zĂăÂâÎîȘșȚț .\-
]+)", re.IGNORECASE)


modele_caractere = []

modele_caractere.append(re.compile(r"^[^a-zA-Z0-9ĂăÂâÎîȘșȚț]+"))

modele_caractere.append(re.compile(r"[^a-zA-Z0-9ĂăÂâÎîȘșȚț]+$"))
```

Sunt prezente modele precompilate Regex pentru caractere speciale la margini de șir, profesori, săli, opționale, tipuri de materie sau nume de materie.

4.2. Tools

Aceste toolkit-uri pe care le voi prezenta sunt colecții de funcții necesare pentru funcționarea aplicației, strânse la un loc pentru modularitate și lizibilitate. Avem 2 toolkit-uri, unul pentru lucrul cu orarul, altul pentru lucrul cu utilizatori.

Începem cu toolkit-ul pentru orar. Prima funcție, incarca_orar_in_redis, se ocupă de încărcarea unui orar în baza de date, sub formă de JSON serializat ca string. Este folosită pe post de cheie o combinație din numele utilizatorului, tipul de procesare și url-ul orarului.

```
async def incarca_orar_in_redis(r: redis.Redis, orar: Orar) -> None:
    await         r.set(f"orar:{orar.nume_utilizator}:{orar.procesare}:{orar.url}",
orar.model_dump_json())
```

Funcția următoare, obtine_orare_utilizator, întoarce toate orarele ale unui utilizator. Pentru asta se face o scanare standard Redis, după un model care conține numele utilizatorului, o scanare care folosește un cursor și întoarce treptat cheile pe care le găsește în fiecare scanare, și ajustează cursorul. Se extrag orarele după cheile găsite și sunt întoarse sub formă de listă de orare.

```
async  def  obtine_orare_utilizator(r:  redis.Redis,  nume_utilizator:  str)  ->
list[Orar]:
    model = f"orar:{nume_utilizator}:*"
    cursor = 0
    orare = []

    while True:
        cursor, chei = await r.scan(cursor=cursor, match=model, count=100)
        if not chei:
            break
        for cheie in chei:
            data = await r.get(cheie)
            if data:
                try:
                    orar = Orar.model_validate_json(data)
                    orare.append(orar)
                except Exception as e:
                    print(f"Eroare la parsarea orarului de la cheia {cheie}: {e}")
        if cursor == 0:
            break

    return orare
```

Această funcție, incarca_orar_in_cache primește un orar, de obicei provenit din frontend, și îl încarcă în Redis cu un timp de expirare de 86400 de secunde, adică 24 de ore. Orarele din cache au ca și cheie doar tipul de procesare și url-ul, numele de utilizator nefiind relevant în cazul acesta.

```
async def incarca_orar_in_cache(r: redis.Redis, orar: Orar) -> None:
    await  r.set(f"orarCache:{orar.procesare}:{orar.url}",  orar.model_dump_json(),
ex=86400)
```

Funcția descarca_orar_din_cache caută să vadă dacă există un orar cu tipul de procesare si url-ul specifice. Dacă îl găsește îl întoarce, dacă nu întoarce None.

```
async def descarca_orar_din_cache(r: redis.Redis, procesare: Procesare, url: str) -
> Orar:
    data = await r.get(f"orarCache:{procesare}:{url}")
    if data is None:
        return None
    return Orar.model_validate_json(data)
```

Funcția verificare_orar_cache verifică dacă un orar se află deja în memoria cache.

```
async def verificare_orar_cache(r: redis.Redis, procesare: Procesare, url: str) ->
bool:
    data = await r.get(f"orarCache:{procesare}:{url}")
    return data is not None
```

Urmeaza acum toolkit-ul pentru utilizator.
Importurile sale sunt următoarele:

```
import bcrypt
from modele.ModeleUtilizator import *
from typing import Union
import redis.asyncio as redis
```

Funcție de creat utilizator. Se preiau numele, emailul și sunt adăugate într-un obiect Utilizator. Este adăugată și parola, dar asta abia după ce trece printr-o funcție hash pentru securitate, Bcrypt.

```
def creeaza_utilizator(nume: str, parola_clara: str, email: str) -> Utilizator:
    salt = bcrypt.gensalt()
    parola_hash = bcrypt.hashpw(parola_clara.encode('utf-8'), salt)
    return Utilizator(nume=nume, parola=parola_hash.decode('utf-8'), email=email)
```

Funcția de verificat parola. Se preia parola introdusă, este transformată în biți și verificată cu o funcție specială care o transformă în hash și o compară cu parola hash-uită.

```
def verifica_parola(parola_introdusa: str, item: Union[Utilizator, str]) -> bool:
    if isinstance(item, Utilizator):
        parola_hash = item.parola
    else:
        parola_hash = item

    return                          bcrypt.checkpw(parola_introdusa.encode('utf-8'),
parola_hash.encode('utf-8'))
```

Încarcă un utilizator în Redis, numele de utilizator este luat ca și cheie.

```
async def incarca_utilizator_in_redis(r: redis.Redis, utilizator: Utilizator) ->
None:
    await r.set(f"user:{utilizator.nume}", utilizator.model_dump_json())
```

Se descarcă un utilizator din Redis, fiind căutat după numele de utilizator.

```
async def descarca_utilizator_din_redis(r: redis.Redis, nume: str) -> Utilizator |
None:
    data = await r.get(f"user:{nume}")
    if data is None:
        return None
    return Utilizator.model_validate_json(data)
```

Verifică dacă mai există un utilizator cu același username.

```
async def verificare_utilizator_duplicat(r: redis.Redis, nume: str) -> bool:
    return await r.exists(f"user:{nume}") > 0
```

Extrage doar datele de utilizator cu excepția parolei, pentru securitate. Căutarea se face după numele de utilizator.

```
async def obtine_date_utilizator(r: redis.Redis, nume: str) -> bool:
    data = await r.get(f"user:{nume}")
    if data is None:
        return None
    utilizator = Utilizator.model_validate_json(data)
    return utilizator.nume, utilizator.email
```

4.3. Extragere date

Partea de extragere a datelor (webscraping) se face în limbaj Python folosind biblioteca BeautifulSoup, în fișierul WebScrapper.py din rădăcina proiectului.

```
from bs4 import BeautifulSoup, ResultSet, Tag
import requests
```

În acesta este o funcție care primește ca parametru un URL de tip string, și întoarce un ResultSet.

```
def webScrapper(url: str) -> ResultSet[Tag]:
```

Folosind biblioteca requests se face o cerere de tip get către pagina de google docs care conține orarul, tot conținutul html al paginii fiind stocat temporar într-o variabilă.

```
page_to_scrape = requests.get(url)
```

Asupra acestei variabile se foloseste parserul html al bibliotecii BeautifulSoup, prin care pagina html este împarțită după taguri, și structurată în cadrul programului.

```
soup = BeautifulSoup(page_to_scrape.text, "html.parser")
```

De aici, este cautat tagul „tbody” care conține toată informația propriu zisă a orarului.

```
body = soup.find('tbody')
```

Împărțind din nou conținutul după tagurile „tr” obținem toate rândurile individuale din tabel, primul rând reprezintă seriile studenților, al doilea grupele, iar restul sunt activitățiile efective din orar înlănțuite, fiecare rand reprezentând un anumit interval orar. Acestea sunt returnate sub formă de ResultSet pentru a fi procesate.

```
body_list = body.findAll('tr')
```

4.4. Procesare Regex

Acestea sunt importurile fișierului:

```
from bs4 import ResultSet, Tag
from WebScrapper import webScrapper
from modele.ModeleRegex import *
from modele.ModeleActivitati import *
```

Funcția principala, regexProcessing, primește un ResultSet, și întoarce în schimb o listă de activități extrase din el.

```
def regexProcessing(body_list: ResultSet[Tag], url: str, nume_utilizator: str):
    global index_rand_grupe
    global grupe_header
```

Prima data, se caută grupe în headerul tabelului. Pentru asta se iau la rând toate rândurile până se găsește unul cu mai multe potriviri. Astfel se detectează headerul. Odata ce a fost găsit headerul, doar se preiau toate grupele găsite, și sunt puse într-o listă alături de cuvântul „sala”, reprezentând o coloană cu săli. Dacă se adaugă „N/A” în listă, aceea va reprezenta o coloană goală.

```
#cautare grupe
    index_rand = 0

    for rand in body_list:
        grupe_header = []
        grupe_adaugate = 0
        for celula in rand:
            if verifica_model(celula.text, model_grupe):
                grupe_header.append(celula.text.strip())
                grupe_adaugate += 1
            #verificare coloana de sali
            elif verifica_model(celula.text, model_sala,):
                grupe_header.append("sala")
            else:
                grupe_header.append("N/A")
        if grupe_adaugate > 4:
            index_rand_grupe = index_rand
            break
        index_rand += 1
```

În cazul puțin probabil în care nu au fost găsite grupe, se completează lista cu „N/A”.

```
#verificare daca au fost gasite grupe
    if grupe_header == []:
        grupe_header = ["N/A"] * len(body_list[1])
```

Contorul acesta rezolvă cazul în care celulele acoperă mai multe rânduri. Deoarece pe rândurile următoare care sunt acoperite deja nu vom găsi celule în locurile deja acoperite, nu avem cum să știm pe ce coloană suntem cu adevărat.
Astfel, acest vector ține cont la parcurgere de ocupările pe mai multe rânduri ale celulelor.

```
#acest contor tine cont cate randuri mai este ocupata o coloana
    contor_coloana = [0] * len(grupe_header)
```

Se folosește o parcurgere specială care calculează constant coloana la care se află celula pe care o parcurge. Cu această parcurgere este apelată o funcție de căutare săli, care încearcă să mai găsească coloane care conțin săli, dar au fost omise în header.

```
#verificare suplimentara coloane de sali
    parcurgere_cu_index(body_list, cauta_sali)
```

Se inițializează o listă goala, și un dicționar mutabil, ale cărui modificări în funcții persistă în toată aplicația. Acest dicționar conține niște context de care se va trebui să se țină cont la trecerea de la o celulă la alta. Se apelează tot parcurgerea cu index, dar de data asta e folosită ca parametru altă funcție, și anume cauta_activitati, care caută activități într-o anumită celulă, extrage toate datele disponibile, și le adaugă într-o listă de activități.

```
#cautare activitate

    lista_activitati = []

    context = {"id_actual": -1,

               "contor_activitati": 0,

               "ora_start": "-",

               "zi": "-"}

    parcurgere_cu_index(body_list, cauta_activitati, lista_activitati, context)

    print("Lungime lista activitati: ", len(lista_activitati))
```

Se creează un obiect de tip orar folosind url-ul paginii extrase, numele de utilizator și lista extrasă.

```
orar    =    Orar(url=url,    nume_utilizator=nume_utilizator,    procesare="Regex",
activitati=lista_activitati)

    return orar
```

Funcția verifică model primește un model sau o listă de modele Regex, și verifică dacă există măcar o potrivire între textul de la intrare și unul din modele.

```
def verifica_model(text, *args):
    bool_list = []
    for intrare in args:
        if type(intrare) == list:
            for model in intrare:
                if model.search(text):
                    bool_list.append(True)
                else:
                    bool_list.append(False)
        else:
            if intrare.search(text):
                bool_list.append(True)
            else:
                bool_list.append(False)
    return any(bool_list)
```

Aceasta este funcția care caută săli, după modelul Regex specific, iar daca le găsește updatează lista de grupe și săli cu cuvântul cheie „sala”, deoarece daca găsește pe o coloană o sală individuală într-o celulă, atunci mai mult ca sigur coloana aceea este o coloană de săli.

```
def cauta_sali(celula, index, *args):

    global grupe_header

    if verifica_model(celula.text, modele_sali) and grupe_header[index] == "N/A":

        grupe_header[index] = "sala"

    return celula.text
```

Funcția afla_interval primește o oră de start, extrage numărul de rânduri ocupate de celulă (acestea reprezentând chiar durata activității în ore), și întoarce un intervalul orar ocupat de activitate, alături de durată.

```
def afla_interval(ora_start, celula):
    if ora_start:
        ora_start = int(ora_start)
        durata = int(celula.get('rowspan', '1'))
        interval = f"{ora_start:02}" + '-' + f"{ora_start + durata:02}"
        return interval, durata
    else:
        return '-', '-'
```

Funcția afla_grupe verifică grupele din cadrul activității, ținând cont de lungimea pe coloane a activității și de lista de grupe din header. Pe lângă grupe întoarce și anul, care este a doua cifră din numărul grupei.

```
def afla_grupe(celula, index):

    global grupe_header

    grupe = []

    for i in range(index,index+int(celula.get('colspan', '1'))):

        if verifica_model(grupe_header[i], model_grupe):

            grupe.append(grupe_header[i])

    if grupe != []:

        anul = grupe[0][1]

        return grupe, anul

    return ['-'], '-'
```

Funcția cauta_sala caută o sală strecurată în conținutul text al unei celule. Folosește o listă de modele Regex precompilate.

```
def cauta_sala(text):
    for model in modele_sala_continut:
        match = model.search(text)
        if match:
            sala = match.group(1)
            return sala
```

```
    return '-'
```

Similar cu funcția anterioară, aceasta caută în textul celulei un nume care începe cu „Profesor”, și extrage doar numele profesorului, fără prefix.

```
def cauta_profesor(text):

    match = model_profesor.search(text)

    if match:

        profesor = match.group(2)

        return profesor


    return '-'
```

Funcția imparte_pe_paritate caută caractere „/” în textul unei celule și împarte conținutul celulei în două. Prima jumătate este materia pe paritate impară, iar a doua jumătate pară. Dacă nu există acel caracter, atunci paritatea este „Ambele”

```
def imparte_pe_paritate(text):
    split = text.split('/')
    elemente = len(split)
    lista = []
    if elemente == 1:
        lista.append({"text": text, "paritate": "Ambele"})
    else:
        if verifica_model(split[0].strip(), model_activitate):
            lista.append({"text": split[0].strip(), "paritate": "Impar"})
        if verifica_model(split[elemente-1].strip(), model_activitate):
            lista.append({"text": split[elemente-1].strip(), "paritate": "Par"})

    return lista
```

Această funcție împarte textul celulei ori după caracterul „|”, ori după un model Regex care detectează opționale, deoarece în general acestea sunt cazurile când avem mai multe materii care se desfășoară în paralel, în aceeași celulă.

```
def imparte_mai_multe(text):

    split = text.split('|')

    elemente = len(split)

    lista = []

    if elemente == 1:

        split2 = re.split(model_optional, text.strip())

        for element2 in split2:

            if verifica_model(element2, model_activitate):
```

```
                lista.append(element2.strip())

    else:

        for element in split:

            if verifica_model(element, model_activitate):

                lista.append(element.strip())

    return lista
```

Această funcție este similară cu cele de mai sus, extrage categoria activității după textul celulei. Aceasta de obicei are forma de „(l)” sau „(curs)”.

```
def cauta_categorie(text):
    if verifica_model(text, modele_categorie_curs):
        return "Curs"
    elif verifica_model(text, modele_categorie_seminar):
        return "Seminar"
    elif verifica_model(text, modele_categorie_laborator):
        return "Laborator"
    elif verifica_model(text, modele_categorie_proiect):
        return "Proiect"
    else:
        return "Curs"
```

Funcția curata_dupa_model șterge orice șir care se potrivește modelelor oferite din textul primit.

```
def curata_dupa_model(text, *args) -> str:
    for intrare in args:
        if type(intrare) == list:
            for model in intrare:
                text_curatat = model.sub('', text).strip()
                text = text_curatat if text_curatat else text
        else:
            text_curatat = intrare.sub('', text).strip()
            text = text_curatat if text_curatat else text
    return text
```

curatare_nume folosește funcția de dinainte pentru a șterge din numele materiei rămășite din numele grupei, al profesorului, caractere speciale, categoria, sala.

```
def curatare_nume(text) -> str:
    textvechi=text
    #sterge numele profesorului, salii, tipul de activitate, dupa model
    text    =    curata_dupa_model(text,    model_profesor,    modele_sala_continut,
modele_categorie_proiect,
                              modele_categorie_seminar,  modele_categorie_laborator,
modele_categorie_curs, modele_caractere)
    return text
```

Una dintre funcțiile principale ale fișierului, cauta_activitati, primește o celulă și extrage orice dată posibilă folosindu-se de modele Regex, și toate funcțiile ajutătoare din restul fișierului. Verifică daca celula este o celulă de zi, daca da extrage ziua, daca nu verifică daca are ore în ea. Și același proces și pentru săli. Dacă nu s-a potrivit cu niciunul din aceste modele mai restrictive, se face și o verificare cu

modelul Regex pentru activități, care acceptă orice text conținând litere. Pe urmă se află intervalul orar, grupa, anul, se împarte celula pe parități daca este nevoie, și apoi se face încă o împărțire dacă sunt detectate mai multe materii pe celulă. La final se caută în numele materiei profesorul, sala, categoria, apoi se curăță numele materiei de elemente inutile și se adaugă în lista de activități.

```
def cauta_activitati(celula, index, *args):
    global grupe_header
    lista_activitati = args[0]
    context = args[1]
    contor_activitati = args[1]["contor_activitati"]
    id_actual = args[1]["id_actual"]
    zi = args[1]["zi"]
    ora_start = args[1]["ora_start"]
    text = celula.text.strip()
    continut = '-'
    profesor = '-'
    sala = '-'
    interval = '-'
    durata = '-'
    grupe = ['-']
    anul = '-'
    categorie = 'Curs'
    paritate = 'Ambele'
    if (verifica_model(text, modele_zile)):
        if verifica_model(text, modele_zile[0]):
            zi = "Luni"
        if verifica_model(text, modele_zile[1]):
            zi = "Marți"
        if verifica_model(text, modele_zile[2]):
            zi = "Miercuri"
        if verifica_model(text, modele_zile[3]):
            zi = "Joi"
        if verifica_model(text, modele_zile[4]):
            zi = "Vineri"
        if verifica_model(text, modele_zile[5]):
            zi = "Sâmbătă"
        if verifica_model(text, modele_zile[6]):
            zi = "Duminică"
    elif (verifica_model(text, model_ore)):
        ora_start = text.split('-')[0].strip()
    elif (verifica_model(text, modele_sali) and ora_start != '-'):
        sala = text
        for i in range(contor_activitati):
            if lista_activitati[-1-i].sala == '-':
                lista_activitati[-1-i].sala = sala
        contor_activitati = 0
    elif (not verifica_model(text, model_grupe)
          and verifica_model(text, model_activitate)
          and not verifica_model(grupe_header[index], model_sala)
          and ora_start != '-'):
        interval, durata = afla_interval(ora_start, celula)
        grupe, anul = afla_grupe(celula, index)
        if grupe != ['-']:
            continut = imparte_pe_paritate(text)
            print(text)
            for activitati in continut:
                paritate = activitati["paritate"]
                nNume = activitati["text"]
```

```
                continut2 = imparte_mai_multe(nNume)
                for activitate in continut2:
                    nume = activitate
                    contor_activitati += 1
                    id_actual = id_actual + 1
                    sala = cauta_sala(nume)
                    categorie = cauta_categorie(nume)
                    profesor = cauta_profesor(nume)
                    nume = curatare_nume(nume)
                    lista_activitati.append(Activitate(
                        id = id_actual,
                        nume = nume,
                        profesor = profesor,
                        sala = sala,
                        zi = zi,
                        interval = interval,
                        durata = durata,
                        grupe = grupe,
                        anul = anul,
                        categorie = categorie,
                        paritate = paritate,
                    ))
    if grupe_header[index] == "sala":
        contor_activitati = 0
    context["id_actual"] = id_actual
    context["contor_activitati"] = contor_activitati
    context["zi"] = zi
    context["ora_start"] = ora_start
    return lista_activitati
```

Ultima funcție din acest fișier, parcurgerea efectivă a celulelor în ordine, ținând tot timpul cont de celula în care ne aflăm. La fiecare trecere a unui rând scade din vectorul contor_activități un 1 pentru coloana la care se află. Daca coloana este liberă trece la procesatul celulei folosind o funcție primită ca parametru. Toate rezultatele acelei funcții sunt puse într-o listă și întoarse înapoi în funcția principală, regexProcessing.

```
def parcurgere_cu_index(body_list, functie, *args):
    global index_rand_grupe
    global grupe_header
    return_list = []
    final_rand = 0
    contor_coloana = [0] * len(grupe_header)
    try:
        for rand in body_list[index_rand_grupe + 1:]:
            if args:
                args[1]["ora_start"] = '-'
                args[1]["contor_activitati"] = 0
            index = 0
            for celula in rand:
                while contor_coloana[index] > 0:
                    contor_coloana[index] -= 1
                    if index == len(contor_coloana)-1:
                        final_rand = 1
                        break
                    index += 1
                if final_rand:
                    final_rand = 0
```

```
                break
            return_list.append(functie(celula, index, *args))
            colspan = int(celula.get('colspan', '1'))
            rowspan = int(celula.get('rowspan', '1'))
            contor_coloana[index:index + colspan] = [rowspan - 1] * colspan
            index += colspan
except Exception as e:
    print("Problema la parcurgerea cu index", e)

return return_list
```

4.5. Procesare AI

Acest fișier face aceeași procesare a tabelului ca la Regex, doar că folosind AI.
Acestea sunt importurile folositeȘ

```
from bs4 import ResultSet, Tag
from openai import AsyncOpenAI
import instructor
from dotenv import load_dotenv
from WebScrapper import webScrapper
from modele.ModeleActivitati import *
```

Funcția principală primește tot un result set, un url și un nume de utilizator.

```
async def aiProcessing(body_list: ResultSet[Tag], url: str, nume_utilizator: str):
```

Se încarcă mai întâi cheia API-ului OpenAI din fișierul „.env”.

```
load_dotenv()
```

Apoi definesc un batch_size, un rând de start, initializez o listă de răspunsuri și un client de tip instructor, folosind OpenAI.

```
start = 3
batch_size = 10
lista_raspunsuri = []
# creere client folosind instructor si OpenAI
client = instructor.from_openai(AsyncOpenAI())
```

Într-un while până la finalul parcurgerii listei de rânduri, se preiau câte 10 rânduri într-un batch, concatenează la un loc sub formă de conținut html, iar apoi e apelat requestul către api.

```
    while start < len(body_list):
        print("Suntem la randul: ",start)
        #obtinere continut html
        batch = [body_list[1]] + body_list[start : start + batch_size]
        #html_content = batch
        html_content = ''.join(str(row) for row in batch)

        #Prompt catre OpenAI, se foloseste o iesire structurata dupa model pydantic
        response = await client.chat.completions.create(
            model='gpt-4o',
            max_retries=10,
            messages=[
                { .......}
            ],
            response_model=List[Activitate]
        )
        lista_raspunsuri.extend(response)
        start += batch_size
```

Acesta este prompt-ul folosit în request:

```
{
```

```
                    "role": "user",
                    "content": f"""
                    Următorul conținut HTML reprezintă mai multe rânduri de orar
universitar:

                    {html_content}

                    Extrage TOATE activitățile individuale din acest HTML. Atenție:
                    - Unele celule pot conține MAI MULTE activități (ex: "CD(l) /
AP(s)"). Fiecare activitate trebuie tratată separat.
                    - Exista materii cu nume scurte, prescurtate, ex: SEP (l), ASC
(l), EIM (l), IM (l), G3D (l)
                    - Tipurile pot fi curs (c), laborator (l), seminar (s), proiect
(p) – dar pot apărea fără mențiune explicită.
                    - Dacă în text apare `/`, este foarte probabil să fie
activități separate cu paritate diferită. Extrage-le separat.
                    - Daca o activitate are numele lipsa, de ex: '-', atunci nu o
adauga in lista
                    - Completează toate câmpurile modelului Activitate:
                      - `nume`: numele materiei, fără simboluri auxiliare
                      - `profesor`: dacă există, fără prefixe (Prof., Profesor
etc.)
                      - `sala`: dacă nu e specificată, pune "-"
                      - `zi`, `interval`, `durata`, `grupe`, `anul`, `categorie`,
`paritate`

                    Returnează o LISTĂ JSON completă de obiecte de tip
`Activitate`, câte unul pentru fiecare activitate (inclusiv curs, laborator,
seminar si proiect).
                    Exemplu de output corect:
                    [Activitate(id=0, nume='SEP', profesor='-', sala='-',
zi=<Zile.LUNI: 'Luni'>, interval='09-11', durata=2, grupe=['441Ba', '441Bb'],
anul=4, categorie=<Categorie.LABORATOR: 'Laborator'>, paritate=<Paritate.IMPAR:
'Impar'>), Activitate(id=1, nume='ASC', profesor='-', sala='-', zi=<Zile.LUNI:
'Luni'>, interval='09-11', durata=2, grupe=['441Ba', '441Bb'], anul=4,
categorie=<Categorie.LABORATOR: 'Laborator'>, paritate=<Paritate.PAR: 'Par'>)]
                    """
                }
```

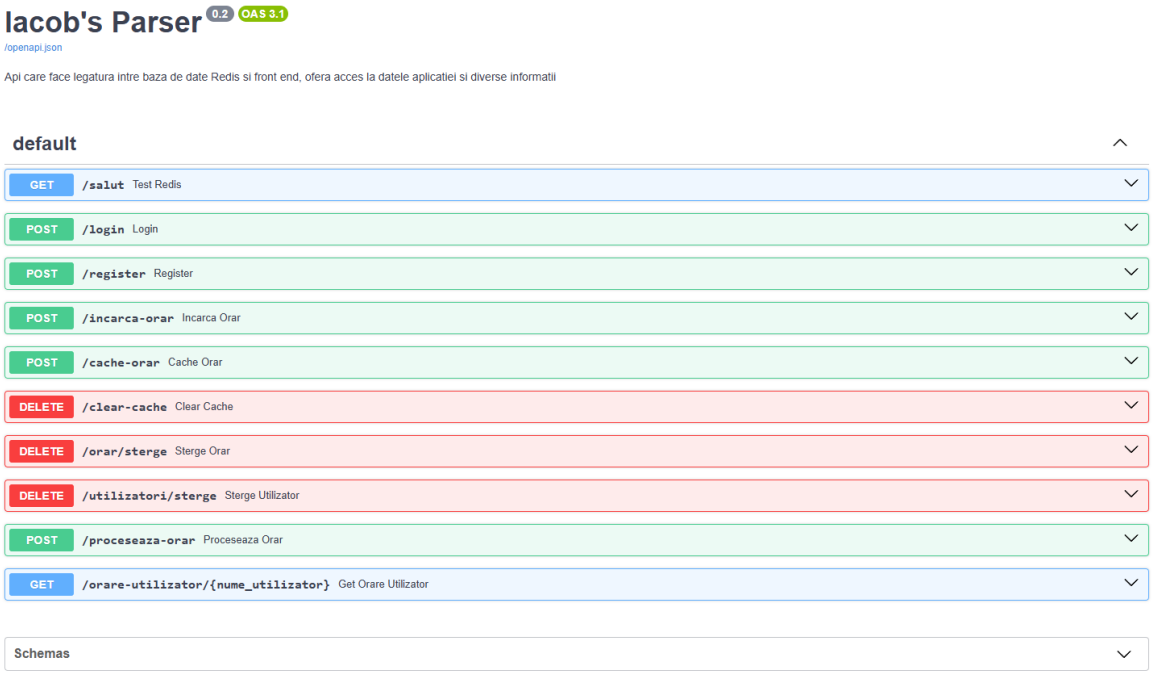
La final, lista de activități întoarsă de AI este pusă sub formă de orar și întoarsă catre API-ul aplicației.

```
    lista_raspunsuri.extend(response)

orar = Orar(url=url, nume_utilizator=nume_utilizator, procesare="AI",
activitati=lista_raspunsuri)
return orar
```

4.6. Interfață de programare aplicație

Partea de legătură dintre bază de date, backend și frontend a fost realizată prin intermediul unui API de tip FastAPI. Acesta definește un set de endpoint-uri asincrone și o structură a datelor de lucru prin care toate componentele proiectului pot comunica între ele. Deoarece endpointurile sunt asincrone, acestea nu se pot executa în paralel pe fire de execuție diferite, pentru performanță ridicată.



Figură 4.6.1 – Pagina de documentație API

Acestea sunt importurile:

```
from contextlib import asynccontextmanager

from fastapi.middleware.cors import CORSMiddleware
from fastapi.requests import Request
import httpx
import uvicorn
from fastapi import FastAPI, HTTPException

from AiProcessing import aiProcessing
from RegexProcessing import regexProcessing
from WebScrapper import webScrapper
from modele.ModeleCereri import *
from tools.ToolkitUtilizator import *
from tools.ToolkitOrar import *
```

Mai întâi am contextul API-ului, care definește ce se întâmplă la pornirea API-ului și la oprirea acestuia. Aici deschid sau închid conexiunea Redis pentru a persista pe tot fișierul.

```
@asynccontextmanager
```

```
async def lifespan(app: FastAPI):
    #Conexiune Redis
    pool = redis.ConnectionPool(host="127.0.0.1", port=6379, db=0, password="user")
    app.state.redis = redis.Redis(connection_pool=pool)

    #Client Async
    app.state.http_client = httpx.AsyncClient()

    print("Startup complet")
    yield  #aici incepe rularea API-ului

    #Inchiderea API-ului
    app.state.redis.close()
    await app.state.http_client.aclose()
    print("Shutdown complet")
```

Aceasta este inițializarea API-ului, folosind contextul lifespan definit mai sus.

```
app = FastAPI(
    title="Iacob's Parser",
    description="Api care face legatura intre baza de date Redis si front end, ofera
acces la datele aplicatiei si diverse informatii",
    version="0.2",
    lifespan=lifespan
)
```

Urmează adăugarea unui middleware Cors, necesar pentru comunicarea cu ReactJS. Permite accesul de la frontend la api.

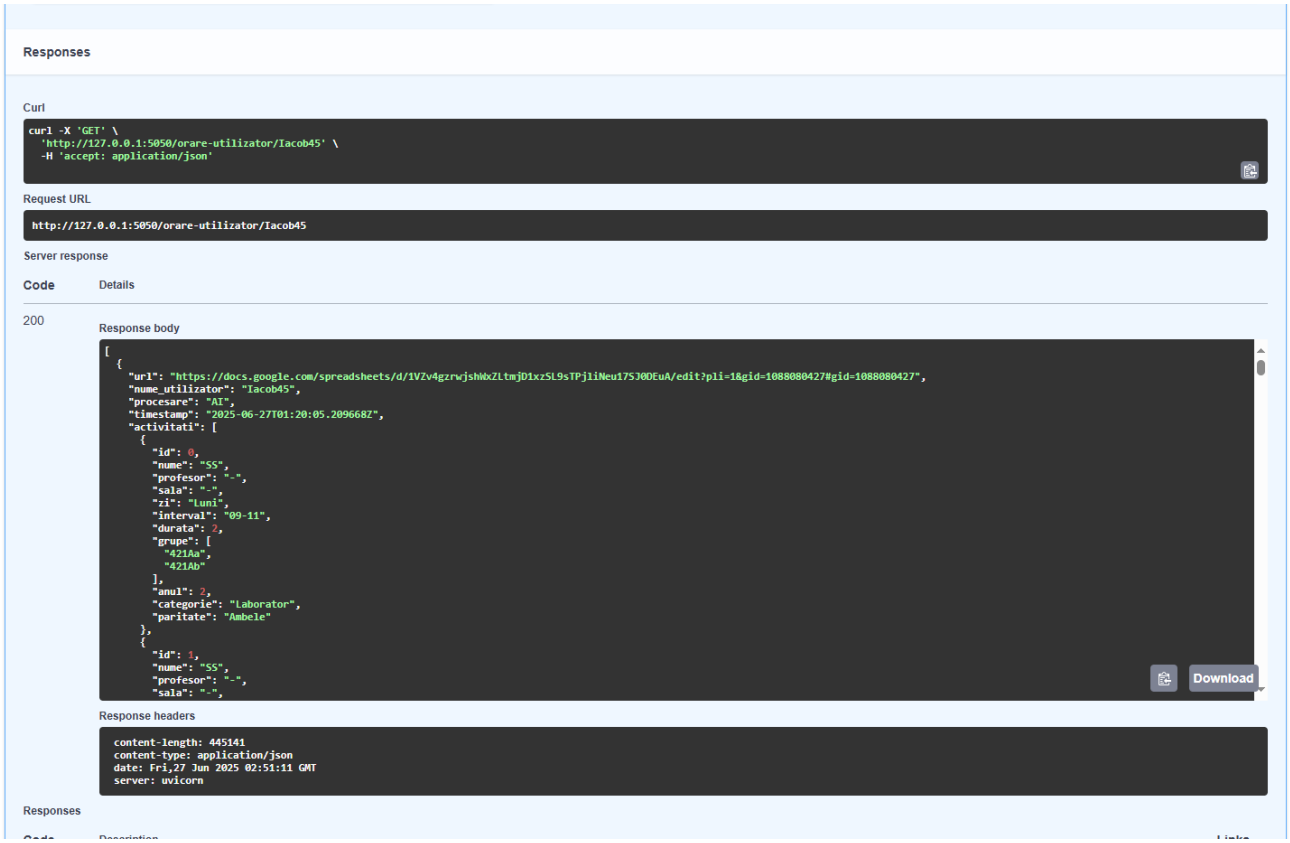
```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Un endpoint de test, face o încărcare în Redis și întoarce un răspuns.

```
@app.get("/salut")
async def test_redis(request: Request):
    r = request.app.state.redis
    r.set("test", "merge")
    return {"valoare": r.get("test")}
```

Endpoint care caută orare după un nume de utilizator și le întoarce sub formă de listă de orare.

```
@app.get("/orare-utilizator/{nume_utilizator}", response_model=list[Orar])
async def get_orare_utilizator(nume_utilizator: str, request: Request):
    r = request.app.state.redis
    orare = await obtine_orare_utilizator(r, nume_utilizator)
    return orare
```



Figură 4.6.2 – Endpoint de căutare orare după numele de utilizator

Endpoint care caută utilizatori după nume și întoarce numele si emailul lor.

```
@app.get("/utilizator/{nume_utilizator}")
async def get_date_utilizator(nume_utilizator: str, request: Request):
    r = request.app.state.redis
    nume, email = await obtine_date_utilizator(r, nume_utilizator)
    if not all([nume, email]):
        raise HTTPException(status_code=404, detail="Utilizator inexistent")
    return {
        "nume": nume,
        "email": email
    }
```

Endpoint de login, caută utilizatorul în baza de date după nume și apoi verifică daca parola introdusă este cea corectă. În cazul în care utilizatorul nu este găsit, sau parola este incorectă, se întoarce o eroare.

```
@app.post("/login")
async def login(cerere: CerereLogin, request: Request):
    print("login")
    r = request.app.state.redis
    utilizator = await descarca_utilizator_din_redis(r, cerere.nume)
    if not utilizator or not verifica_parola(cerere.parola, utilizator):
        raise HTTPException(status_code=401, detail="Date de autentificare
incorecte")
    return {"mesaj": "Autentificare reușită"}
```

Endpointul de register, primește datele utilizatorului, verifică daca mai există deja un utilizator cu același

nume și daca toate datele au fost completate, și daca totul este ok se creează un nou utilizator și se încarcă în baza de date.

```
@app.post("/register")
async def register(cerere: CerereRegister, request: Request):
    r = request.app.state.redis

    #verifica daca exista deja un utilizator cu acelasi nume
    if await verificare_utilizator_duplicat(r, cerere.nume):
        raise HTTPException(status_code=400, detail="Utilizatorul există deja")

    #creaza si salveaza nou utilizator
    if all([cerere.nume,cerere.parola,cerere.email]):
        print(cerere.nume)
        utilizator = creeaza_utilizator(cerere.nume, cerere.parola, cerere.email)
        await incarca_utilizator_in_redis(r, utilizator)
    else:
        raise HTTPException(status_code=400, detail="Nu au fost introduse date
pentru inregistrare")

    return {"mesaj": "Utilizator înregistrat cu succes"}
```

Endpoint pentru încărcarea unui orar în baza de date.

```
@app.post("/incarca-orar")
async def incarca_orar(orar: Orar, request: Request):
    r = request.app.state.redis
    await incarca_orar_in_redis(r, orar)
    return {"mesaj": "Orarul a fost încărcat cu succes în Redis"}
```

Endpoint pentru încărcarea unui orar în memoria cache.

```
@app.post("/cache-orar")
async def cache_orar(orar: Orar, request: Request):
    r = request.app.state.redis

    #verificam cache-ul actual
    exista = await verificare_orar_cache(r, orar.procesare, orar.url)
    if exista:
        return {"mesaj": "Orarul există deja în cache"}

    await incarca_orar_in_cache(r, orar)
    return {"mesaj": "Orarul a fost adăugat în cache"}
```

Endpointul care apelează procesarea propriu-zisă a aplicației. Verifică tipul de procesare cerut și apelează ori motorul de procesare Regex, ori cel AI. Orarul primit îl încarcă în cache și îl trimite la utilizator.

```
@app.post("/proceseaza-orar")
async def proceseaza_orar(cerere: CerereProcesare, request: Request) -> Orar:
    r = request.app.state.redis
    if await verificare_orar_cache(r, cerere.procesare, cerere.url):
        orar = await descarca_orar_din_cache(r, cerere.procesare, cerere.url)
        if orar is not None:
            return orar
    try:
        body_list = webScrapper(cerere.url)
        if cerere.procesare == Procesare.AI:
```

```
            orar = await aiProcessing(body_list, cerere.url, cerere.nume_utilizator)
        elif cerere.procesare == Procesare.REGEX:
            orar = regexProcessing(body_list, cerere.url, cerere.nume_utilizator)
        else:
            raise HTTPException(status_code=400, detail="Procesare invalidă")
        await incarca_orar_in_cache(r, orar)
        return orar
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Endpoint de golire a cache-ului pentru debugging sau diferite erori.

```
@app.delete("/clear-cache")
async def clear_cache(request: Request):
    r = request.app.state.redis
    #cauta toate cheile orarelor din cache
    chei = await r.keys("orarCache:*")
    if chei:
        await r.delete(*chei)
        return {"mesaj": f"S-au șters {len(chei)} intrări din cache."}
    return {"mesaj": "Cache-ul era deja gol."}
```

Aici se poate șterge un orar după numele de utilizator, tipul de procesare si email.

```
@app.delete("/orar/sterge")
async def sterge_orar(cerere: CerereStergereOrar, request: Request):
    r = request.app.state.redis
    cheie = f"orar:{cerere.nume_utilizator}:{cerere.procesare}:{cerere.url}"
    rezultat = await r.delete(cheie)
    if rezultat == 0:
        raise HTTPException(status_code=404, detail="Orar inexistent")
    return {"mesaj": "Orar șters cu succes"}
```

Aici se poate șterge un utilizator după username.

```
@app.delete("/utilizatori/sterge")
async def sterge_utilizator(
    nume: str,
    request: Request
):
    r = request.app.state.redis
    rezultat = await r.delete(f"user:{nume}")
    if rezultat == 0:
        raise HTTPException(status_code=404, detail="Utilizator inexistent")
    return {"mesaj": "Utilizator șters cu succes"}
```

Pornirea server-ului uvicorn pe portul 5050 (Pornește API-ul).

```
if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=5050)
```

4.7. Interfață grafică frontend

Aplicația frontend este făcută in ReactJS. Toate afișările web sunt apelate dinamic și au format JSX. Acestea sunt importurile fișierului de bază, App.js:

```
import React, { useState, useEffect } from "react";
import PaginaAutentificare from "./componente/PaginaAutentificare";
import PaginaProcesare from "./componente/PaginaProcesare";
import PaginaProfilUtilizator from "./componente/PaginaProfilUtilizator";
```

Mai întâi am creat aceste variabile care gestionează 2 state-uri permanente per browser. Numele utilizatorului conectat, și un state ajutător care verifică daca utilizatorul este pe profilul de utilizator sau nu.

```
const [utilizatorAutentificat, setUtilizatorAutentificat] = useState(null);
const [panouUtilizator, setPanouUtilizator] = useState(false);
```

La fiecare accesare a paginii se verifică dacă a fost stocat un nume de utilizator, dacă da atunci este acestei conexiuni pentru ca utilizatorul să nu trebuiască să se reconecteze la fiecare refresh de pagină. Se verifică și dacă utilizatorul este sau nu în profilul de utilizator, pentru a nu fi scos de acolo la refresh-ul paginii.

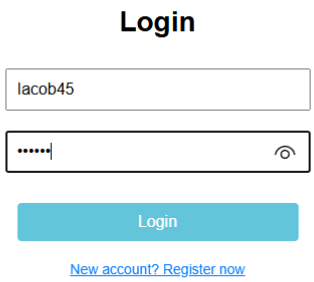
```
useEffect(() => {
    const utilizatorSalvat = localStorage.getItem("utilizator");
    const panou = localStorage.getItem("panou_utilizator") === "true";
    if (utilizatorSalvat) {
      setUtilizatorAutentificat(utilizatorSalvat);
    }
    setPanouUtilizator(panou);
  }, []);
```

Funcție de logout care resetează stările aplicației despre utilizator.

```
const logout = () => {
    localStorage.removeItem("utilizator");
    localStorage.removeItem("panou_utilizator");
    setUtilizatorAutentificat(null);
    setPanouUtilizator(false);
  };
```

Verifică daca utilizatorul e conectat, daca nu este conectat îi încarcă pagina de autentificare.

```
{utilizatorAutentificat ? (
 .... <PaginaProcesare numeUtilizator={utilizatorAutentificat} />
) : (
<PaginaAutentificare
          onLogin={(nume) => {
            setUtilizatorAutentificat(nume);
          }}
        />)
```



Figură 4.7.1 – Interfață grafică autentificare

Daca starea de panou utilizator este activată, utilizatorului îi este afișată pagina de profil.

```
!panouUtilizator ? (
 .... <PaginaProcesare numeUtilizator={utilizatorAutentificat} />
) : (
<PaginaProfilUtilizator numeUtilizator={utilizatorAutentificat} />)
```

Generarea unui panou de utilizator în dreapta sus a paginii atunci care îți oferă opțiunea de logout printr-un buton, sau opțiunea de a intra pe profilul de utilizator pe alt buton.

```
<div style={{ marginBottom: "10px", fontWeight: "bold" }}>
                Bine ai venit, {utilizatorAutentificat}!
              </div>
              <div
                style={{
                  display: "flex",
                  gap: "10px",
                  justifyContent: "flex-end",
                }}
              >
                <button
                  onClick={logout}
                  style={{
                    backgroundColor: "#fff",
                    border: "1px solid #333",
                    color: "#333",
                    padding: "6px 12px",
                    borderRadius: "5px",
                    cursor: "pointer",
                    fontSize: "14px",
                  }}
                >
                  Logout
                </button>
                <button
                  onClick={() => {
                    setPanouUtilizator(true);
```

```
                localStorage.setItem("panou_utilizator", true);
              }}
              style={{
                backgroundColor: "#fff",
                border: "1px solid #333",
                color: "#333",
                padding: "6px 12px",
                borderRadius: "5px",
                cursor: "pointer",
                fontSize: "14px",
              }}
            >
              Contul meu
            </button>
```

Acestea sunt variabilele paginii de autentificare. Referințele la inputuri și buton sunt pentru deplasarea cu enter pe UI-ul de autentificare.

```
const [esteInregistrare, setEsteInregistrare] = useState(false);
  const [nume, setNume] = useState("");
  const [parola, setParola] = useState("");
  const [email, setEmail] = useState("");
  const [eroare, setEroare] = useState("");

  const butonRef = useRef();
  const numeRef = useRef();
  const parolaRef = useRef();
  const emailRef = useRef();
```

Aceasta este funcția apelată când un input este în focus și se apasă o tastă. Se verifică dacă tasta este „Enter”, și dacă este atunci se trece la următorul input automat. Daca suntem la ultimul se apasă direct butonul de submit.

```
const handleEnter = (e, nextRef) => {
    if (e.key === "Enter") {
      e.preventDefault();
      if (nextRef === butonRef) {
        if (butonRef.current) {
          butonRef.current.click();
        }
      } else if (nextRef && nextRef.current) {
        nextRef.current.focus();
      }
    }
  };
```

Funcția de salvare utilizator. Îl stochează în browser și în starea aplicației.

```
const salveazaUtilizator = (numeUtilizator) => {
    localStorage.setItem("utilizator", numeUtilizator);
    onLogin(numeUtilizator);
  };
```

Funcția de trimitere autentificare, verifică dacă au fost introduse date în inputuri, și daca da, sunt trimise către API. Daca API-ul dă un răspuns pozitiv, atunci datele de utilizator sunt salvate și utilizatorul este

considerat autentificat.

```
const trimiteAutentificare = async () => {
    if (!nume || !parola) {
      alert("Nu au fost introduse date pentru login");
      return;
    }
    try {
      const response = await fetch("http://localhost:5050/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ nume, parola }),
      });
      const data = await response.json();
      if (response.ok) {
        salveazaUtilizator(nume);
      } else {
        alert(data.detail || "Eroare la autentificare");
      }
    } catch (e) {
      alert("Eroare la conectarea cu serverul");
    }
  };
```

Funcția de register este similară, verifică ca toate câmpurile să fie completate si trimite o cerere către API pentru înregistrare.

```
const trimiteInregistrare = async () => {
    if (!nume || !parola || !email) {
      alert("Nu au fost introduse date pentru inregistrare");
      return;
    }
    try {
      const response = await fetch("http://localhost:5050/register", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ nume, parola, email }),
      });
      const data = await response.json();
      if (response.ok) {
        salveazaUtilizator(nume);
      } else {
        alert(data.detail || "Eroare la înregistrare");
      }
    } catch (e) {
      alert("Eroare la conectarea cu serverul");
    }
  };
```

La scrierea într-un input, textul este actualizat pe măsură ce este scris, și se salvează o referință pentru funcția de mișcare cu enter între inputuri.

```
<input
        type="text"
        placeholder="Nume"
        value={nume}
        ref={numeRef}
```

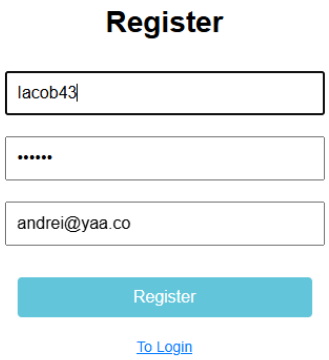
```
        onChange={(e) => setNume(e.target.value)}
        onKeyDown={(e) => handleEnter(e, parolaRef)}
        style={{
          margin: "10px",
          padding: "10px",
          width: "280px",
          fontSize: "16px",
        }}
      />
```

Câmpul de email apare doar dacă suntem pe modul de înregistrare.

```
{esteInregistrare && (
        <input
          type="email"
          placeholder="Email"
          value={email}
          ref={emailRef}
          onChange={(e) => setEmail(e.target.value)}
          onKeyDown={(e) => handleEnter(e, butonRef)}
          style={{
            margin: "10px",
            padding: "10px",
            width: "280px",
            fontSize: "16px",
          }}
        />
      )}
```

Acesta este câmpul pe care se poate apăsa pentru a te deplasa de la autentificare la înregistrare, și invers.

```
<div
        style={{
          fontSize: "14px",
          color: "#007BFF",
          cursor: "pointer",
          textDecoration: "underline",
        }}
        onClick={() => {
          setEsteInregistrare(!esteInregistrare);
        }}
      >
        {esteInregistrare ? "To Login" : "New account? Register now"}
      </div>
```

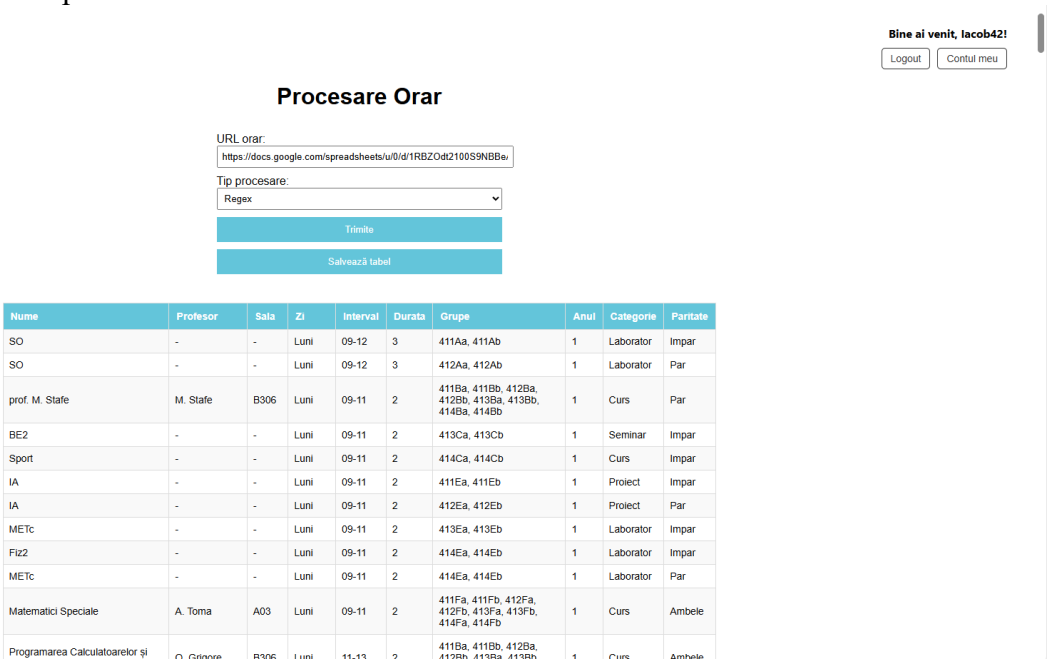


Figură 4.7.2 – Interfață grafică înregistrare

În funcție de asta este modificat și butonul de submit din „Login” în „Register”:

```
<button
  ref={butonRef}
  onClick={esteInregistrare ? trimiteInregistrare : trimiteAutentificare}
  style={{
    margin: "20px",
    padding: "10px 20px",
    width: "280px",
    fontSize: "16px",
    backgroundColor: "#63C5DA",
    color: "white",
    border: "none",
    borderRadius: "4px",
    cursor: "pointer",
  }}
>
  {esteInregistrare ? "Register" : "Login"}
</button>
```

Trecem acum la pagina de procesare.



Figură 4.7.3 – Interfață grafică procesare orar

Variabilele de aici sunt:

```
  const [url, setUrl] = useState("");
  const [procesare, setProcesare] = useState("Regex");
  const [raspuns, setRaspuns] = useState(null);
  const [eroare, setEroare] = useState("");
```

Avem o funcție de trimitere cerere de procesare, aceasta preia din inputurile din UI un url, tipul de procesare, și din starea aplicației preia numele utilizatorului. Folosind aceste date face un request către endpointul api-ului de procesare orare. Odată primit răspunsul, este salvat și pe urmă afișat sub formă de tabel.

```
const trimiteCerereProcesare = async () => {
    const CerereProcesare = {
      url,
      procesare,
      nume_utilizator: numeUtilizator,
    };

    try {
      const response = await fetch("http://localhost:5050/proceseaza-orar", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(CerereProcesare),
      });

      if (!response.ok) {
        alert("Eroare la cererea de procesare");
      }

      const data = await response.json();
      setRaspuns(data);
```

```
    } catch (e) {
      alert("Eroare la cererea de procesare: " + e);
    }
  };
```

Funcția de salvare a unui orar. Apare un buton de salvare orar mereu când există un răspuns valid întors de la API. Apelează un endpoint de încărcare orar din API.

```
const salveazaOrar = async () => {
    if (!raspuns || !raspuns.activitati) {
      alert("Nu există un orar de salvat.");
      return;
    }

    try {
      const response = await fetch("http://localhost:5050/incarca-orar", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(raspuns),
      });

      const data = await response.json();
      alert(data.mesaj || "Orarul a fost salvat cu succes.");
    } catch (e) {
      alert("Eroare la salvarea orarului:", e);
    }
  };
```

Stilurile celulelor de tabel (tabel header, tabel division).

```
const thStyle = {
    padding: "10px",
    border: "1px solid #ddd",
    textAlign: "left",
  };

  const tdStyle = {
    padding: "8px",
    border: "1px solid #ddd",
  };
```

Aceasta este dropdown-ul care selectează tipul de procesare.

```
<label>
          Tip procesare:
          <select
            value={procesare}
            onChange={(e) => setProcesare(e.target.value)}
            style={{ width: "100%", padding: "6px" }}
          >
            <option value="Regex">Regex</option>
            <option value="AI">AI</option>
          </select>
        </label>
```

Iar acesta este tabelul în care sunt salvate datele din orar. Acest tabel apare doar daca există un orar cu activități în starea aplicației.

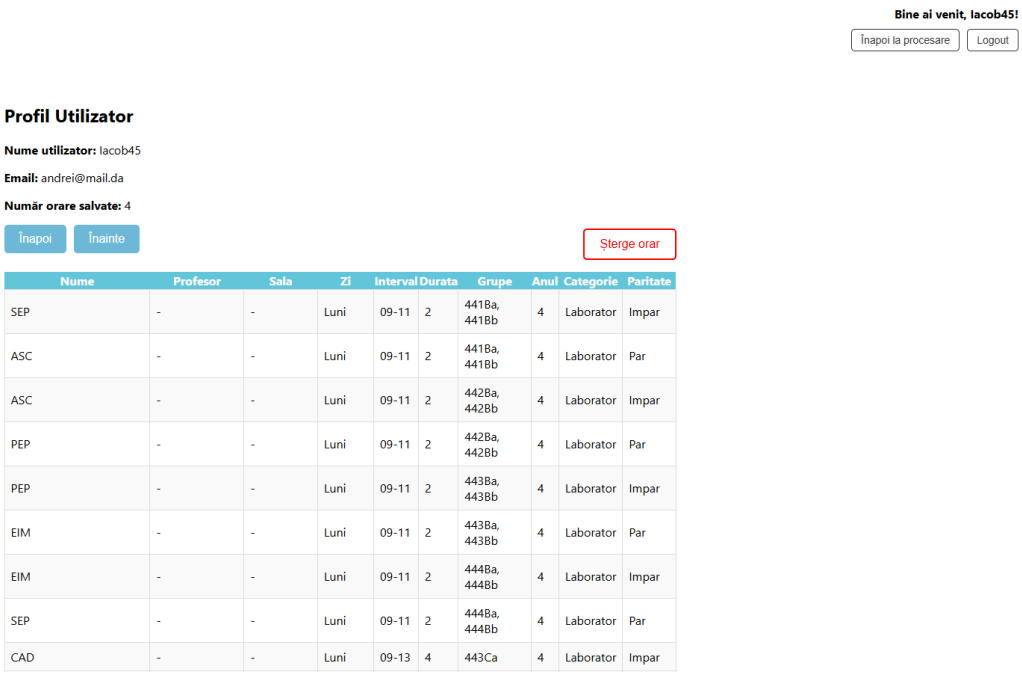
```
{raspuns && raspuns.activitati && (
        <div
          style={{
            marginTop: "40px",
            overflowX: "auto",
            width: "100%",
            maxWidth: "1000px",
          }}
        >
          <table
            style={{
              width: "100%",
              borderCollapse: "collapse",
              boxShadow: "0 0 10px rgba(0,0,0,0.1)",
              fontSize: "14px",
            }}
          >
            <thead style={{ backgroundColor: "#63C5DA", color: "white" }}>
              <tr>
                <th style={thStyle}>Nume</th>
                <th style={thStyle}>Profesor</th>
                <th style={thStyle}>Sala</th>
                <th style={thStyle}>Zi</th>
                <th style={thStyle}>Interval</th>
                <th style={thStyle}>Durata</th>
                <th style={thStyle}>Grupe</th>
                <th style={thStyle}>Anul</th>
                <th style={thStyle}>Categorie</th>
                <th style={thStyle}>Paritate</th>
              </tr>
            </thead>
            <tbody>
              {raspuns.activitati.map((act, index) => (
                <tr
                  key={index}
                  style={{
                    backgroundColor: index % 2 === 0 ? "#f9f9f9" : "#ffffff",
                  }}
                >
                  <td style={tdStyle}>{act.nume}</td>
                  <td style={tdStyle}>{act.profesor}</td>
                  <td style={tdStyle}>{act.sala}</td>
                  <td style={tdStyle}>{act.zi}</td>
                  <td style={tdStyle}>{act.interval}</td>
                  <td style={tdStyle}>{act.durata}</td>
                  <td style={tdStyle}>{act.grupe.join(", ")}</td>
                  <td style={tdStyle}>{act.anul}</td>
                  <td style={tdStyle}>{act.categorie}</td>
                  <td style={tdStyle}>{act.paritate}</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
```

La scrierea într-un input, textul este actualizat pe măsură ce este scris, și se salvează o referință pentru

funcția de mișcare cu enter între inputuri.

```
<input
        type="text"
        placeholder="Nume"
        value={nume}
        ref={numeRef}
        onChange={(e) => setNume(e.target.value)}
        onKeyDown={(e) => handleEnter(e, parolaRef)}
        style={{
          margin: "10px",
          padding: "10px",
          width: "280px",
          fontSize: "16px",
        }}
```

Trecem acum la pagina de profil a utilizatorului.



Figură 4.7.4 – Interfață grafică profil utilizator

Avem următoarele variabile:

```
const [orare, setOrare] = useState([]);
  const [indexCurent, setIndexCurent] = useState(0);
  const [email, setEmail] = useState("");
```

La accesarea paginii se trimit două request-uri către API prin care sunt căutate datele utilizatorului, și respectiv toate orarele salvate de acesta.

```
useEffect(() => {
    const fetchDate = async () => {
      try {
        const orareResp = await fetch(
          `http://localhost:5050/orare-utilizator/${numeUtilizator}`
        );
```

```
        const orareData = await orareResp.json();
        setOrare(orareData);

        const userResp = await fetch(
          `http://localhost:5050/utilizator/${numeUtilizator}`
        );
        const userData = await userResp.json();
        setEmail(userData.email);
      } catch (e) {
        alert("Eroare la încărcarea datelor utilizatorului:", e);
      }
    };

    fetchDate();
  }, [numeUtilizator]);
```

Această funcție trimite o cerere de a șterge un orar al utilizatorului.

```
const stergeOrar = async () => {
    const orarDeSters = orare[indexCurent];
    try {
      const response = await fetch("http://localhost:5050/orar/sterge", {
        method: "DELETE",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(orarDeSters),
      });
      if (!response.ok) {
        const e = await response.json();
        alert("Eroare la ștergerea orarului");
        return;
      }
      const orareNoi = [...orare];
      orareNoi.splice(indexCurent, 1);
      setOrare(orareNoi);
      setIndexCurent(Math.max(0, indexCurent - 1));
    } catch (e) {
      alert("Eroare la ștergerea orarului");
    }
  };
```

Această variabilă urmărește orarul care este în momentul de față afișat.

```
const orarCurent = orare[indexCurent];
```

Afișarea datelor utilizatorului.

```
<h2>Profil Utilizator</h2>
      <p>
        <strong>Nume utilizator:</strong> {numeUtilizator}
      </p>
      <p>
        <strong>Email:</strong> {email}
      </p>
      <p>
        <strong>Număr orare salvate:</strong> {orare.length}
      </p>
```

Două butoane de mers înainte și inapoi de la un orar la altul, butonul de înainte nu apare daca suntem la ultimul orar, și respectiv, cel de înapoi, dacă suntem la primul orar (index = 0).

```
{indexCurent > 0 && (
                <button
                  onClick={() => setIndexCurent(indexCurent - 1)}
                  style={{
                    padding: "10px 20px",
                    fontSize: "16px",
                    color: "white",
                    backgroundColor: "#6db8d6",
                    border: "none",
                    borderRadius: "5px",
                    cursor: "pointer",
                  }}
                >
                  Înapoi
                </button>
              )}
              {indexCurent < orare.length - 1 && (
                <button
                  onClick={() => setIndexCurent(indexCurent + 1)}
                  style={{
                    padding: "10px 20px",
                    fontSize: "16px",
                    color: "white",
                    backgroundColor: "#6db8d6",
                    border: "none",
                    borderRadius: "5px",
                    cursor: "pointer",
                  }}
                >
                  Înainte
                </button>
```

Butonul de ștergere al unui orar.

```
<button
              onClick={stergeOrar}
              style={{
                padding: "10px 20px",
                fontSize: "16px",
                backgroundColor: "#fff",
                border: "2px solid red",
                color: "red",
                borderRadius: "5px",
                cursor: "pointer",
                marginLeft: "auto",
              }}
            >
              Șterge orar
            </button>
```

Afișarea efectivă a tabelului care este urmărit în momentul acesta, prin variabila index.

```
<table style={{ width: "100%", borderCollapse: "collapse" }}>
```

```
  <thead style={{ backgroundColor: "#63C5DA", color: "white" }}>
    <tr>
      <th>Nume</th>
      <th>Profesor</th>
      <th>Sala</th>
      <th>Zi</th>
      <th>Interval</th>
      <th>Durata</th>
      <th>Grupe</th>
      <th>Anul</th>
      <th>Categorie</th>
      <th>Paritate</th>
    </tr>
  </thead>
  <tbody>
    {orarCurent.activitati.map((act, index) => (
      <tr
        key={index}
        style={{
          backgroundColor: index % 2 === 0 ? "#f9f9f9" : "#ffffff",
        }}
      >
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.nume}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.profesor}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.sala}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.zi}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.interval}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.durata}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.grupe.join(", ")}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.anul}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.categorie}
        </td>
        <td style={{ border: "1px solid #ddd", padding: "8px" }}>
          {act.paritate}
        </td>
      </tr>
    ))}
  </tbody>
</table>
```

5. CONCLUZII

În această lucrare am analizat, proiectat, și implementat tema „Serviciu online pentru conversia documentelor Google Doc”. Am început cu o parte teoretică în care am descris toate tehnologiile luate în considerare pentru realizarea acestei lucrări, inclusiv unele pe care nu am decis să le folosesc în final. Am analizat toate aceste tehnologii individual și le-am comparat între ele, alegând opțiunile care îmi ofereau cel mai ridicat nivel de performanță.

După ce am decis preliminar tehnologiile pe care urma să le folosesc, am început partea de proiectare, am stabilit structura pe care o are proiectul meu, centrată pe API, care are și rol de backend. Am creat niște diagrame care să descrie structura pe care o aveam în minte și acțiunile pe care le poate parcurge utilizatorul.

Nu in ultimul rând, am trecut la partea de implementare și testare a proiectului, pe parcursul căreia am mai tot făcut schimbări în urmă constatărilor practice asupra performanțelor și caracteristicilor tehnologiilor pe care le-am încercat. Am început construind motoarele de procesare, acestea fiind cele mai importante elemente ale proiectului făcând munca propriu-zisă. Pe urmă am început să lucrez la API, fiind centrul întregului proiect, și apoi în paralel cu API-ul am dezvoltat interfața frontend și baza de date.

În final, acest proiect a ajuns să folosească două motoare de procesare pentru a evidenția diferența programării clasice și a celei orientate pe inteligență artificială. Primul dintre ele folosește REGEX, obținând un rezultat rapid, cu un minim de resurse folosite. Celălalt motor de procesare folosește modelul AI de la OpenAI, gpt-4o, prin intermediul API-ului lor. Acest motor oferă performanțe mai ridicate și analize mai detalizate ale datelor, dar consumă mult mai multe resurse și necesită mult mai mult timp de procesare. Aceste două motoare sunt controlate de API pentru a oferi informații structurate în format JSON utilizatorului, și pentru a le stoca în baza de date REDIS. API-ul este de tip FastAPI pentru un timp minim de aducere a informațiilor, dar și datorită constrângerilor diferite pe care le poate aplica asupra datelor prin intermediul modelului pydantic. Baza de date REDIS este o bază de date foarte rapidă, care ruleaza în RAM, dar folosește și persistență, stocând continuu a datelor și în memoria nevolatilă. Aceasta stochează datele utilizatorilor, parolele lor fiind trecute printr-o funcție hash pentru securitate, Bcrypt, stochează orarele pe care utilizatorii doresc să le salveze în cadrul conturilor lor, și stochează și restul de orare prelucrate sub forma unei memorii cache care se golește zilnic. Interfața grafică pe care o accesează utilizatorii este făcută folosind ReactJS datorită caracteristicilor sale moderne.

„Una dintre cele mai mari provocări în domeniul inteligenței artificiale nu este doar replicarea inteligenței umane, ci și îmbunătățirea acesteia. Viitorul AI-ului nu este unul în care oamenii sunt înlocuiți cu mașini, ci unul în care acestea lucrează cu noi, ne extind capabilitățile și ne permite să ne concentrăm mai mult pe creativitate, etică și gândire strategică.” [31]

6. BIBLIOGRAFIE

[1] G. van Rossum, Limbajul de programare Python. Python Software Foundation, 2024. Disponibil la: https://www.python.org/

[2] M. Lutz, Învățarea limbajului Python. Programare orientată pe obiecte. Ediția a 5-a, O'Reilly Media, 2013.

[3] T. Grossman, Documentația Beautiful Soup. Crummy, 2023. Disponibil la: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[4] S. Montanari, Documentația oficială FastAPI, 2023. Disponibil la: https://fastapi.tiangolo.com/

[5] Meta AI, LLaMA – Modele lingvistice mari open-source, 2023. Disponibil la: https://ai.meta.com/llama/

[6] Echipa Uvicorn, Uvicorn – Server ASGI ultra-rapid pentru aplicații web asincrone, 2023. Disponibil la: https://www.uvicorn.org/

[7] Facebook Open Source, React – O bibliotecă JavaScript pentru interfețe de utilizator, 2024. Disponibil la: https://reactjs.org/

[8] A. Banks și E. Porcello, Learning React: Functional Web Development with React and Redux, O'Reilly Media, 2017.

[9] T. Holowaychuk, ReactJS vs. VueJS – Comparație tehnică și decizională, Dev.to, 2022. Disponibil la: https://dev.to/

[10] E. You, VueJS – The Progressive JavaScript Framework, 2024. Disponibil la: https://vuejs.org/

[11] A. Osmani, Designing with Vue, Smashing Magazine, 2021.

[12] Vue Team, Documentație oficială Vue Router și Vuex, 2023. Disponibil la: https://vuejs.org/guide/

[13] M. Tiago, Introducere în FastAPI: performanță și dezvoltare asincronă, RealPython, 2021. Disponibil la: https://realpython.com/fastapi-python-web-apis

[14] I. Petrescu, Bazele de date relaționale – MySQL în practică, Ed. Matrix Rom, București, 2021.

[15] R. Dobre, „Optimizarea performanței interogărilor SQL și replicarea în MySQL”, Revista Informatica Economică, vol. 25, nr. 3, pp. 57–63, 2021.

[16] M. Popescu, Introducere în Redis – Sisteme NoSQL de mare viteză, Ed. Universitară, 2022.

[17] C. Rusu, „Redis în arhitectura microserviciilor moderne”, Jurnalul de Tehnologii Web, vol. 18, nr. 2, pp. 12–17, 2023.

[18] S. Russell și P. Norvig, Inteligența artificială. O abordare modernă, Ed. Academică, București, 2020.

[19] Google calls Gemma 3 the most powerful AI model you can run on one GPU. Disponibil la: https://www.theverge.com/ai-artificial-intelligence/627968/google-gemma-3-open-ai-model

[20] DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence. Disponibil la: https://github.com/deepseek-ai/DeepSeek-Coder-V2

[21] What is ChatGPT?, TechTarget, publicat 4 martie 2025. Disponibil la: https://www.techtarget.com/whatis/definition/ChatGPT

[22] Named-entity recognition, Wikipedia. Disponibil la: https://en.wikipedia.org/wiki/Named-entity_recognition

[23] Facts & Figures – spaCy Usage, spaCy, 2025. Disponibil la: https://spacy.io/usage/facts-figures

[24] dslim/bert-base-NER model (fine-tuned BERT pentru NER), Hugging Face, 2023. Disponibil la: https://huggingface.co/dslim/bert-base-NER

[25] M. O. Rabin și D. Scott, Finite Automata and Their Decision Problems, IBM Journal of Research and Development, vol. 3, nr. 2, pp. 114-125, 1959.

[26] Expresii regulate (Regex), Wikipedia. Disponibil la: https://ro.wikipedia.org/wiki/Expresie_regulată

[27] M. G. N. Pătrașcu, Programare Python pentru începători, Editura Teora, București, 2020, cap. 8: „Expresii regulate”.

[28] M. Petre, Ghid practic pentru expresii regulate, Editura Polirom, Iași, 2018.

[29] K. S. Rajasekaran, *Artificial Intelligence and Machine Learning*, Delhi: PHI Learning, 2020.

[30] M. Tegmark, Life 3.0: Being Human in the Age of Artificial Intelligence, New York: Knopf, 2017.

[31] E. Brynjolfsson și A. McAfee, The Second Machine Age, Cambridge, MA: *MIT Press*, 2014.

[32] DeepSeek: How A Small Chinese AI Company Is Shaking Up US Tech Heavyweights, *NDTV World*, https://www.ndtv.com/world-news/deepseek-how-a-small-chinese-ai-company-is-shaking-up-us-tech-heavyweights-7584010

[33] Interview cu Liang, *36Kr*, 2023. Disponibil la: https://36kr.com/p/xxxxxx

[34] Google calls Gemma 3 the most powerful AI model you can run on one GPU, *The Verge*, 12 martie 2025. Disponibil la: https://www.theverge.com/ai-artificial-intelligence/627968/google-gemma-3-open-ai-model

[35] Inteligența artificială și liberul arbitru, *Contributors*, 2023. Disponibil la: https://www.contributors.ro/inteligenta-artificiala-si-liberul-arbitru/

[36] OpenAI releases GPT-4o, a faster model that's free for all ChatGPT users, *The Verge*, https://www.theverge.com/2024/5/13/24155493/openai-gpt-4o-launching-free-for-all-chatgpt-users

[37] L. Richardson, Beautiful Soup Documentation. Disponibil la: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[38] *Python Software Foundation*, "Beautiful Soup: We called him Tortoise because he taught us,". Disponibil la: https://pypi.org/project/beautifulsoup4/

[39]B. Martin, "Ultimate Guide to Web Scraping with Python Part 1: Requests and BeautifulSoup," *LearnDataSci*. Disponibil la: https://www.learndatasci.com/tutorials/ultimate-guide-web-scraping-w-python-requests-and-beautifulsoup/

[40]"Web Scraping Unveiled: Your Comprehensive Guide to Data Extraction," *DataScientest*, Nov. 6, 2023. Disponibil la: https://datascientest.com/en/web-scraping-unveiled-your-comprehensive-guide-to-data-extraction

[41]Auth0, "Password Storage: How to do it properly," *Auth0 Blog*, 2021. Disponibil la: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

[42]Wikipedia, "Cryptographic hash function," *Wikipedia*. Disponibil la: https://en.wikipedia.org/wiki/Cryptographic_hash_function

[43] OWASP, „Password Storage Cheat Sheet". Disponibil la: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[44] Python Software Foundation, „bcrypt – Password hashing for Python". Disponibil la: https://pypi.org/project/bcrypt/

ANEXĂ
COD SURSĂ

Fișiere SRC

WebScrapper.py

```
from bs4 import BeautifulSoup, ResultSet, Tag
import requests

def webScrapper(url: str) -> ResultSet[Tag]:
    print("\nStart WebScrapper")

    page_to_scrape = requests.get(url)
    soup = BeautifulSoup(page_to_scrape.text, "html.parser")

    body = soup.find('tbody')
    body_list = body.findAll('tr')

    return body_list
```

RegexProcessing.py

```
from bs4 import ResultSet, Tag
from WebScrapper import webScrapper
from modele.ModeleRegex import *
from modele.ModeleActivitati import *

grupe_header = []
index_rand_grupe = -1

def verifica_model(text, *args):
    bool_list = []
    for intrare in args:
        if type(intrare) == list:
            for model in intrare:
                if model.search(text):
                    bool_list.append(True)
                else:
                    bool_list.append(False)
        else:
            if intrare.search(text):
                bool_list.append(True)
            else:
                bool_list.append(False)
    return any(bool_list)

def cauta_sali(celula, index, *args):
    global grupe_header
    if verifica_model(celula.text, modele_sali) and grupe_header[index] == "N/A":
        grupe_header[index] = "sala"
    return celula.text

def afla_interval(ora_start, celula):
```

```
    if ora_start:
        ora_start = int(ora_start)
        durata = int(celula.get('rowspan', '1'))
        interval = f"{ora_start:02}" + '-' + f"{ora_start + durata:02}"
        return interval, durata
    else:
        return '-', '-'

def afla_grupe(celula, index):
    global grupe_header
    grupe = []
    for i in range(index,index+int(celula.get('colspan', '1'))):
        if verifica_model(grupe_header[i], model_grupe):
            grupe.append(grupe_header[i])
    if grupe != []:
        anul = grupe[0][1]
        return grupe, anul
    return ['-'], '-'

def cauta_sala(text):
    for model in modele_sala_continut:
        match = model.search(text)
        if match:
            sala = match.group(1)
            return sala

    return '-'

def cauta_profesor(text):
    match = model_profesor.search(text)
    if match:
        profesor = match.group(2)
        return profesor

    return '-'

def imparte_pe_paritate(text):
    split = text.split('/')
    elemente = len(split)
    lista = []
    if elemente == 1:
        lista.append({"text": text, "paritate": "Ambele"})
    else:
        if verifica_model(split[0].strip(), model_activitate):
            lista.append({"text": split[0].strip(), "paritate": "Impar"})
        if verifica_model(split[elemente-1].strip(), model_activitate):
            lista.append({"text": split[elemente-1].strip(), "paritate": "Par"})

    return lista

def imparte_mai_multe(text):
    split = text.split('|')
    elemente = len(split)
    lista = []
    if elemente == 1:
        split2 = re.split(model_optional, text.strip())
        for element2 in split2:
            if verifica_model(element2, model_activitate):
                lista.append(element2.strip())
```

```
    else:
        for element in split:
            if verifica_model(element, model_activitate):
                lista.append(element.strip())
    return lista

def cauta_categorie(text):
    if verifica_model(text, modele_categorie_curs):
        return "Curs"
    elif verifica_model(text, modele_categorie_seminar):
        return "Seminar"
    elif verifica_model(text, modele_categorie_laborator):
        return "Laborator"
    elif verifica_model(text, modele_categorie_proiect):
        return "Proiect"
    else:
        return "Curs"

def curata_dupa_model(text, *args) -> str:
    for intrare in args:
        if type(intrare) == list:
            for model in intrare:
                text_curatat = model.sub('', text).strip()
                text = text_curatat if text_curatat else text
        else:
            text_curatat = intrare.sub('', text).strip()
            text = text_curatat if text_curatat else text
    return text

def curatare_nume(text) -> str:
    textvechi=text
    #sterge numele profesorului, salii, tipul de activitate, dupa model
    text    =    curata_dupa_model(text,    model_profesor,    modele_sala_continut,
modele_categorie_proiect,
                              modele_categorie_seminar,  modele_categorie_laborator,
modele_categorie_curs, modele_caractere)
    return text


def cauta_activitati(celula, index, *args):
    global grupe_header
    lista_activitati = args[0]
    context = args[1]
    contor_activitati = args[1]["contor_activitati"]
    id_actual = args[1]["id_actual"]
    zi = args[1]["zi"]
    ora_start = args[1]["ora_start"]
    text = celula.text.strip()
    continut = '-'
    profesor = '-'
    sala = '-'
    interval = '-'
    durata = '-'
    grupe = ['-']
    anul = '-'
    categorie = 'Curs'
    paritate = 'Ambele'
    if (verifica_model(text, modele_zile)):
        if verifica_model(text, modele_zile[0]):
```

```
            zi = "Luni"
        if verifica_model(text, modele_zile[1]):
            zi = "Marți"
        if verifica_model(text, modele_zile[2]):
            zi = "Miercuri"
        if verifica_model(text, modele_zile[3]):
            zi = "Joi"
        if verifica_model(text, modele_zile[4]):
            zi = "Vineri"
        if verifica_model(text, modele_zile[5]):
            zi = "Sâmbătă"
        if verifica_model(text, modele_zile[6]):
            zi = "Duminică"
    elif (verifica_model(text, model_ore)):
        ora_start = text.split('-')[0].strip()
    elif (verifica_model(text, modele_sali) and ora_start != '-'):
        sala = text
        for i in range(contor_activitati):
            if lista_activitati[-1-i].sala == '-':
                lista_activitati[-1-i].sala = sala
        contor_activitati = 0
    elif (not verifica_model(text, model_grupe)
          and verifica_model(text, model_activitate)
          and not verifica_model(grupe_header[index], model_sala)
          and ora_start != '-'):
        interval, durata = afla_interval(ora_start, celula)
        grupe, anul = afla_grupe(celula, index)
        if grupe != ['-']:
            continut = imparte_pe_paritate(text)
            print(text)
            for activitati in continut:
                paritate = activitati["paritate"]
                nNume = activitati["text"]
                continut2 = imparte_mai_multe(nNume)
                for activitate in continut2:
                    nume = activitate
                    contor_activitati += 1
                    id_actual = id_actual + 1
                    sala = cauta_sala(nume)
                    categorie = cauta_categorie(nume)
                    profesor = cauta_profesor(nume)
                    nume = curatare_nume(nume)
                    lista_activitati.append(Activitate(
                        id = id_actual,
                        nume = nume,
                        profesor = profesor,
                        sala = sala,
                        zi = zi,
                        interval = interval,
                        durata = durata,
                        grupe = grupe,
                        anul = anul,
                        categorie = categorie,
                        paritate = paritate,
                    ))
    if grupe_header[index] == "sala":
        contor_activitati = 0
    context["id_actual"] = id_actual
    context["contor_activitati"] = contor_activitati
```

```
    context["zi"] = zi
    context["ora_start"] = ora_start
    return lista_activitati

def parcurgere_cu_index(body_list, functie, *args):
    global index_rand_grupe
    global grupe_header
    return_list = []
    final_rand = 0
    contor_coloana = [0] * len(grupe_header)
    try:
        for rand in body_list[index_rand_grupe + 1:]:
            if args:
                args[1]["ora_start"] = '-'
                args[1]["contor_activitati"] = 0
            index = 0
            for celula in rand:
                while contor_coloana[index] > 0:
                    contor_coloana[index] -= 1
                    if index == len(contor_coloana)-1:
                        final_rand = 1
                        break
                    index += 1
                if final_rand:
                    final_rand = 0
                    break
                return_list.append(functie(celula, index, *args))
                colspan = int(celula.get('colspan', '1'))
                rowspan = int(celula.get('rowspan', '1'))
                contor_coloana[index:index + colspan] = [rowspan - 1] * colspan
                index += colspan
    except Exception as e:
        print("Problema la parcurgerea cu index", e)

    return return_list

def regexProcessing(body_list: ResultSet[Tag], url: str, nume_utilizator: str):
    global index_rand_grupe
    global grupe_header
    print("\nStart RegexProcessing")



    #cautare grupe
    index_rand = 0

    for rand in body_list:
        grupe_header = []
        grupe_adaugate = 0
        for celula in rand:
            if verifica_model(celula.text, model_grupe):
                grupe_header.append(celula.text.strip())
                grupe_adaugate += 1
            #verificare coloana de sali
            elif verifica_model(celula.text, model_sala,):
                grupe_header.append("sala")
            else:
                grupe_header.append("N/A")
        if grupe_adaugate > 4:
```

```
            index_rand_grupe = index_rand
            break
        index_rand += 1

    #verificare daca au fost gasite grupe
    if grupe_header == []:
        grupe_header = ["N/A"] * len(body_list[1])

    #acest contor tine cont cate randuri mai este ocupata o coloana
    contor_coloana = [0] * len(grupe_header)

    #verificare suplimentara coloane de sali
    parcurgere_cu_index(body_list, cauta_sali)

    print("Update grupe cu coloane sali:", grupe_header)


    #cautare activitate
    lista_activitati = []
    context = {"id_actual": -1,
               "contor_activitati": 0,
               "ora_start": "-",
               "zi": "-"}
    parcurgere_cu_index(body_list, cauta_activitati, lista_activitati, context)
    print("Lungime lista activitati: ", len(lista_activitati))


    orar   =   Orar(url=url,   nume_utilizator=nume_utilizator,   procesare="Regex",
activitati=lista_activitati)
    return orar
```

AiProcessing.py

```
from bs4 import ResultSet, Tag
from openai import AsyncOpenAI
import instructor
from dotenv import load_dotenv
from WebScrapper import webScrapper
from modele.ModeleActivitati import *

async def aiProcessing(body_list: ResultSet[Tag], url: str, nume_utilizator: str):
    print("\nStart AiProcessing")
    #incarcare cheie API
    load_dotenv()

    start = 3
    batch_size = 10
    lista_raspunsuri = []
    while start < len(body_list):
        print("Suntem la randul: ",start)
        #obtinere continut html
        batch = [body_list[1]] + body_list[start : start + batch_size]
        #html_content = batch
        html_content = ''.join(str(row) for row in batch)

        #creere client folosind instructor si OpenAI
        client = instructor.from_openai(AsyncOpenAI())
```

```
        #Prompt catre OpenAI, se foloseste o iesire structurata dupa model pydantic
        response = await client.chat.completions.create(
            model='gpt-4o',
            max_retries=10,
            messages=[
                {
                    "role": "user",
                    "content": f"""
                    Următorul conținut HTML reprezintă mai multe rânduri de orar
universitar:

                    {html_content}

                    Extrage TOATE activitățile individuale din acest HTML. Atenție:
                    - Unele celule pot conține MAI MULTE activități (ex: "CD(l) /
AP(s)"). Fiecare activitate trebuie tratată separat.
                    - Exista materii cu nume scurte, prescurtate, ex: SEP (l), ASC
(l), EIM (l), IM (l), G3D (l)
                    - Tipurile pot fi curs (c), laborator (l), seminar (s), proiect
(p) – dar pot apărea fără mențiune explicită.
                    - Dacă în text apare `/`, este foarte probabil să fie activități
separate cu paritate diferită. Extrage-le separat.
                    - Daca o activitate are numele lipsa, de ex: '-', atunci nu o
adauga in lista
                    - Completează toate câmpurile modelului Activitate:
                      - `nume`: numele materiei, fără simboluri auxiliare
                      - `profesor`: dacă există, fără prefixe (Prof., Profesor etc.)
                      - `sala`: dacă nu e specificată, pune "-"
                      - `zi`, `interval`, `durata`, `grupe`, `anul`, `categorie`,
`paritate`

                    Returnează o LISTĂ JSON completă de obiecte de tip `Activitate`,
câte unul pentru fiecare activitate (inclusiv curs, laborator, seminar si proiect).
                    Exemplu de output corect:
                    [Activitate(id=0,    nume='SEP',    profesor='-',    sala='-',
zi=<Zile.LUNI:  'Luni'>,  interval='09-11',  durata=2,  grupe=['441Ba',  '441Bb'],
anul=4,  categorie=<Categorie.LABORATOR:  'Laborator'>,  paritate=<Paritate.IMPAR:
'Impar'>),  Activitate(id=1,  nume='ASC',  profesor='-',  sala='-',  zi=<Zile.LUNI:
'Luni'>,   interval='09-11',   durata=2,   grupe=['441Ba',   '441Bb'],   anul=4,
categorie=<Categorie.LABORATOR: 'Laborator'>, paritate=<Paritate.PAR: 'Par'>)]
                    """
                }
            ],
            response_model=List[Activitate]
        )
        lista_raspunsuri.extend(response)
        start += batch_size

    #Afisare raspuns
    id_actual = 0
    for i in lista_raspunsuri:
        i.id=id_actual
        id_actual += 1
        print(i)
    print("Lungime lista activitati: ", len(lista_raspunsuri))
    orar   =   Orar(url=url,   nume_utilizator=nume_utilizator,   procesare="AI",
activitati=lista_raspunsuri)
    return orar
```

Api.py

```
from contextlib import asynccontextmanager

from fastapi.security import OAuth2PasswordBearer
from fastapi.middleware.cors import CORSMiddleware
from fastapi.requests import Request
import httpx
import redis.asyncio as redis
import uvicorn
from fastapi import FastAPI, HTTPException, Path, Query

from AiProcessing import aiProcessing
from RegexProcessing import regexProcessing
from WebScrapper import webScrapper
from modele.ModeleActivitati import *
from modele.ModeleCereri import *
from modele.ModeleUtilizator import *
from tools.ToolkitUtilizator import *
from tools.ToolkitOrar import *


#   pool   =   redis.ConnectionPool(host="127.0.0.1",   port=int(6379),   db=int(1),
password="user")
# r = redis.Redis(connection_pool=pool)
# asx = httpx.AsyncClient()


@asynccontextmanager
async def lifespan(app: FastAPI):
    #Conexiune Redis
    pool = redis.ConnectionPool(host="127.0.0.1", port=6379, db=0, password="user")
    app.state.redis = redis.Redis(connection_pool=pool)

    #Client Async
    app.state.http_client = httpx.AsyncClient()

    print("Startup complet")
    yield  #aici incepe rularea API-ului

    #Inchiderea API-ului
    app.state.redis.close()
    await app.state.http_client.aclose()
    print("Shutdown complet")


app = FastAPI(
    title="Iacob's Parser",
    description="Api care face legatura intre baza de date Redis si front end, ofera
acces la datele aplicatiei si diverse informatii",
    version="0.2",
    lifespan=lifespan
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
```

```
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/salut")
async def test_redis(request: Request):
    r = request.app.state.redis
    r.set("test", "merge")
    return {"valoare": r.get("test")}

@app.get("/orare-utilizator/{nume_utilizator}", response_model=list[Orar])
async def get_orare_utilizator(nume_utilizator: str, request: Request):
    r = request.app.state.redis
    orare = await obtine_orare_utilizator(r, nume_utilizator)
    return orare

@app.get("/utilizator/{nume_utilizator}")
async def get_date_utilizator(nume_utilizator: str, request: Request):
    r = request.app.state.redis
    nume, email = await obtine_date_utilizator(r, nume_utilizator)
    if not all([nume, email]):
        raise HTTPException(status_code=404, detail="Utilizator inexistent")
    return {
        "nume": nume,
        "email": email
    }

@app.post("/login")
async def login(cerere: CerereLogin, request: Request):
    print("login")
    r = request.app.state.redis
    utilizator = await descarca_utilizator_din_redis(r, cerere.nume)
    if not utilizator or not verifica_parola(cerere.parola, utilizator):
        raise   HTTPException(status_code=401,   detail="Date   de   autentificare
incorecte")
    return {"mesaj": "Autentificare reușită"}

@app.post("/register")
async def register(cerere: CerereRegister, request: Request):
    r = request.app.state.redis

    #verifica daca exista deja un utilizator cu acelasi nume
    if await verificare_utilizator_duplicat(r, cerere.nume):
        raise HTTPException(status_code=400, detail="Utilizatorul există deja")

    #creaza si salveaza nou utilizator
    if all([cerere.nume,cerere.parola,cerere.email]):
        print(cerere.nume)
        utilizator = creeaza_utilizator(cerere.nume, cerere.parola, cerere.email)
        await incarca_utilizator_in_redis(r, utilizator)
    else:
        raise  HTTPException(status_code=400,  detail="Nu  au  fost  introduse  date
pentru inregistrare")

    return {"mesaj": "Utilizator înregistrat cu succes"}


@app.post("/incarca-orar")
```

```
async def incarca_orar(orar: Orar, request: Request):
    r = request.app.state.redis
    await incarca_orar_in_redis(r, orar)
    return {"mesaj": "Orarul a fost încărcat cu succes în Redis"}


@app.post("/cache-orar")
async def cache_orar(orar: Orar, request: Request):
    r = request.app.state.redis

    #verificam cache-ul actual
    exista = await verificare_orar_cache(r, orar.procesare, orar.url)
    if exista:
        return {"mesaj": "Orarul există deja în cache"}

    await incarca_orar_in_cache(r, orar)
    return {"mesaj": "Orarul a fost adăugat în cache"}


@app.post("/proceseaza-orar")
async def proceseaza_orar(cerere: CerereProcesare, request: Request) -> Orar:
    r = request.app.state.redis
    if await verificare_orar_cache(r, cerere.procesare, cerere.url):
        orar = await descarca_orar_din_cache(r, cerere.procesare, cerere.url)
        if orar is not None:
            return orar
    try:
        body_list = webScrapper(cerere.url)
        if cerere.procesare == Procesare.AI:
            orar = await aiProcessing(body_list, cerere.url, cerere.nume_utilizator)
        elif cerere.procesare == Procesare.REGEX:
            orar = regexProcessing(body_list, cerere.url, cerere.nume_utilizator)
        else:
            raise HTTPException(status_code=400, detail="Procesare invalidă")
        await incarca_orar_in_cache(r, orar)
        return orar
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))



@app.delete("/clear-cache")
async def clear_cache(request: Request):
    r = request.app.state.redis
    #cauta toate cheile orarelor din cache
    chei = await r.keys("orarCache:*")
    if chei:
        await r.delete(*chei)
        return {"mesaj": f"S-au șters {len(chei)} intrări din cache."}
    return {"mesaj": "Cache-ul era deja gol."}

@app.delete("/orar/sterge")
async def sterge_orar(cerere: CerereStergereOrar, request: Request):
    r = request.app.state.redis
    cheie = f"orar:{cerere.nume_utilizator}:{cerere.procesare}:{cerere.url}"
    rezultat = await r.delete(cheie)
    if rezultat == 0:
        raise HTTPException(status_code=404, detail="Orar inexistent")
    return {"mesaj": "Orar șters cu succes"}
```

```
@app.delete("/utilizatori/sterge")
async def sterge_utilizator(
    nume: str,
    request: Request
):
    r = request.app.state.redis
    rezultat = await r.delete(f"user:{nume}")
    if rezultat == 0:
        raise HTTPException(status_code=404, detail="Utilizator inexistent")
    return {"mesaj": "Utilizator șters cu succes"}


if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=5050)
```

Modele de date

ModeleActivitati.py

```
from datetime import datetime, timezone
from typing import List

from pydantic import BaseModel, Field
from enum import Enum

class Zile(Enum):
    """Zilele saptamanii"""
    LUNI = "Luni"
    MARTI = "Marți"
    MIERCURI = "Miercuri"
    JOI = "Joi"
    VINERI = "Vineri"
    SAMBATA = "Sâmbătă"
    DUMINICA = "Duminică"

class Categorie(Enum):
    """Categoria activitatii"""
    CURS = "Curs"
    SEMINAR = "Seminar"
    LABORATOR = "Laborator"
    PROIECT = "Proiect"

class Paritate(Enum):
    """Categoria activitatii"""
    PAR = "Par"
    IMPAR = "Impar"
    AMBELE = "Ambele"

class Activitate(BaseModel):
    """Structura unei activitati"""
    id: int = Field(description="Id-ul activitatii, incepe de la 0 si se
incrementeaza")
    nume: str = Field(description="Numele materiei. Exista materii cu nume mai lungi,
dar si cu nume scurte,"
                                  " ex: Programarea Calculatoarelor și Limbaje de
```

```
Programare 2 (curs), SEP,"
                                    " ASC, EIM, IM, G3D. De asemenea, materiile pot fi
impartite"
                                    " si prin | sau || daca sunt mai multe intr-o
singura celula, caz in care acestea se"
                                    "  desfasoara  in  paralel  si  sunt  in  general
optionale.")
    profesor: str = Field(description="Profesorul care se ocupa de activitate, numele
lui incepe"
                                        " cu prefixele Prof., sau Profesor,"
                                        " sau Prof, prefixele nu vor fii pastrate la
nume,"
                                        " profesorul poate lipsi din celula, caz in
care pui -")
    sala: str = Field(description="Sala in care se desfasoara activitatea, de forma
B303, sau A01,"
                                    " sau Sala Orange, sau B208a. Apare ori in celula
activitatii, ori in dreapta"
                                    " celulei, ori deloc, caz in care se completeaza
cu -")
    zi: Zile = Field(description="Ziua saptamanii in care se desfasoara activitatea,
apare la inceputul"
                                              " fiecarui rand de tabel")
    interval:  str  =  Field(description="Intervalul  orar  in  care  se  desfasoara
activitatea, se ia ora de start de la"
                                        "  inceputul  randului  de  tabel  si  se  aduna
numarul de randuri ale celulei")
    durata: int = Field(description="Durata activitatii, trebuie sa se potriveasca
cu intervalul orar")
    grupe: List[str] = Field(description="Grupele care participă la activitate, apar
in primul rand de tabel,"
                                           "  deasupra  celulelor  destinate.  Sunt  de
forma 441Da, 423Bb.")
    anul: int = Field(description="Anul de studiu, se extrage din numele uneia dintre
grupele"
                                            " care participa la activitate. A doua
cifra este chiar anul de studiu."
                                            " e.g. din 432Ab se extrage anul egal
cu 3")
    categorie:     Categorie     =     Field(description="Categoria     activitatii
(curs,seminar,laborator sau proiect,"
                                                    "    prescurtate    uneori
(c),(s),(l),(p))")
    paritate: Paritate = Field(description="Paritatea activitatii (par sau impa, sau
ambele)."
                                                  " Daca in continutul celulei
apare un /, atunci ce este la stanga"
                                                  " de / este impar, iar ce este
la dreapta este par. Altfel,"
                                                  "  in  lipsa  unui  /  ca  si
paritate avem Ambele")

class Procesare(Enum):
    """Tipul de procesare"""
    AI = "AI"
    REGEX = "Regex"

class Orar(BaseModel):
    "Contine lista de activitati"
```

```
    url: str = Field(description="URL sursa al orarului")
    nume_utilizator:    str    =    Field(description="Identificatorul    unic    al
utilizatorului, numele")
    procesare: Procesare = Field(description="Motorul de procesare folosit")
    timestamp: datetime = Field(default_factory=lambda: datetime.now(timezone.utc),
description="Momentul când orarul a fost salvat")
    activitati:   List[Activitate]   =   Field(description="Lista   de   activitati
inlantuite")
```

ModeleCereri.py

```
from pydantic import BaseModel, EmailStr

from modele.ModeleActivitati import Procesare


class CerereProcesare(BaseModel):
    url: str
    procesare: Procesare
    nume_utilizator: str

class CerereLogin(BaseModel):
    nume: str
    parola: str

class CerereRegister(BaseModel):
    nume: str
    parola: str
    email: EmailStr

class CerereStergereOrar(BaseModel):
    nume_utilizator: str
    procesare: Procesare
    url: str
```

ModeleRegex.py

```
import re

#Modele regex precompilate
model_grupe = re.compile("^[ ]{0,2}[0-9]{3}[A-Za-z][A-Za-z]{0,1}[ ]{0,2}$")

modele_zile = []
modele_zile.append(re.compile("^[ ]{0,2}[Ll][Uu][Nn][Ii][ ]{0,2}$"))
modele_zile.append(re.compile("^[ ]{0,2}[Mm][Aa][Rr][TtȚț][Ii][ ]{0,2}$"))
modele_zile.append(re.compile("^[           ]{0,2}[Mm][Ii][Ee][Rr][Cc][Uu][Rr][Ii][
]{0,2}$"))
modele_zile.append(re.compile("^[ ]{0,2}[Jj][Oo][Ii][ ]{0,2}$"))
modele_zile.append(re.compile("^[ ]{0,2}[Vv][Ii][Nn][Ee][Rr][Ii][ ]{0,2}$"))
modele_zile.append(re.compile("^[           ]{0,2}[Ss][AaÂâ][Mm][Bb][AaĂă][Tt][Ăă][
]{0,2}$"))
modele_zile.append(re.compile("^[         ]{0,2}[Dd][Uu][Mm][Ii][Nn][Ii][Cc][AaĂă][
]{0,2}$"))

model_sala = re.compile("^[ ]{0,2}[Ss][Aa][Ll][AaĂă][ ]{0,2}$")
```

```
modele_sali = []
modele_sali.append(re.compile("^[ ]{0,2}[BbAa][0-9]{1,4}[A-Za-z]{0,1}[ ]{0,2}$"))
modele_sali.append(re.compile("^[ ]{0,2}[Ss][Aa][Ll][AaĂă]"))

modele_sala_continut = []
modele_sala_continut.append(re.compile("([BbAa][0-9]{1,4}[A-Za-z]{0,1})"))
modele_sala_continut.append(re.compile(r"\b[Ss][Aa][Ll][AaĂă][ ]{0,2}\w+"))

model_activitate = re.compile("[A-Za-z]{2,}")

model_ore = re.compile("^[ ]{0,2}[0-9]{0,1}[0-9][-][0-9][0-9]{0,1}[ ]{0,2}$")

modele_categorie_curs = []
modele_categorie_curs.append(re.compile(r"\(+[ ]{0,2}[Cc][Uu][Rr][Ss][ ]{0,2}\)+"))
modele_categorie_curs.append(re.compile(r"\(+[  ]{0,2}[Ll][Ee][Cc][Tt][Uu][Rr][Ee][
]{0,2}\)+"))
modele_categorie_curs.append(re.compile(r"\(+[ ]{0,2}[Cc][ ]{0,2}\)+"))
modele_categorie_seminar = []
modele_categorie_seminar.append(re.compile(r"\(+[
]{0,2}[Ss][Ee][Mm][Ii][Nn][Aa][Rr][ ]{0,2}\)+"))
modele_categorie_seminar.append(re.compile(r"\(+[ ]{0,2}[Ss][ ]{0,2}\)+"))
modele_categorie_laborator = []
modele_categorie_laborator.append(re.compile(r"\(+[
]{0,2}[Ll][Aa][Bb][Oo][Rr][Aa][Tt][Oo][Rr][Yy][ ]{0,2}\)+"))
modele_categorie_laborator.append(re.compile(r"\(+[
]{0,2}[Ll][Aa][Bb][Oo][Rr][Aa][Tt][Oo][Rr][ ]{0,2}\)+"))
modele_categorie_laborator.append(re.compile(r"\(+[ ]{0,2}[Ll][ ]{0,2}\)"))
modele_categorie_proiect = []
modele_categorie_proiect.append(re.compile(r"\(+[
]{0,2}[Pp][Rr][Oo][Jj][Ee][Cc][Tt][ ]{0,2}\)+"))
modele_categorie_proiect.append(re.compile(r"\(+[
]{0,2}[Pp][Rr][Oo][Ii][Ee][Cc][Tt][ ]{0,2}\)+"))
modele_categorie_proiect.append(re.compile(r"\(+[ ]{0,2}[Pp][ ]{0,2}\)+"))

model_optional = re.compile(r"(?=\s*[Oo]([Pp][Tt][.\s]*\d+|[1-9]\.*(?=\W|$)))")


model_profesor = re.compile(r"(Prof\.?|Profesor|Professor)\s?([A-Za-zĂăÂâÎîȘșȚț .\-
]+)", re.IGNORECASE)

modele_caractere = []
modele_caractere.append(re.compile(r"^[^a-zA-Z0-9ĂăÂâÎîȘșȚț]+"))
modele_caractere.append(re.compile(r"[^a-zA-Z0-9ĂăÂâÎîȘșȚț]+$"))
```

ModeleUtilizator.py

```
from pydantic import BaseModel, Field, EmailStr

class Utilizator(BaseModel):
    nume: str = Field(description="Username-ul utilizatorului")
    parola: str = Field(description="Parola hashata a utilizatorului")
    email: EmailStr = Field(description="Adresa de email a utilizatorului")
```

Tools

ToolkitOrar.py

```
from modele.ModeleActivitati import *
import redis.asyncio as redis


async def incarca_orar_in_redis(r: redis.Redis, orar: Orar) -> None:
    await   r.set(f"orar:{orar.nume_utilizator}:{orar.procesare}:{orar.url}",
orar.model_dump_json())

async def obtine_orare_utilizator(r: redis.Redis, nume_utilizator: str) ->
list[Orar]:
    model = f"orar:{nume_utilizator}:*"
    cursor = 0
    orare = []

    while True:
        cursor, chei = await r.scan(cursor=cursor, match=model, count=100)
        if not chei:
            break
        for cheie in chei:
            data = await r.get(cheie)
            if data:
                try:
                    orar = Orar.model_validate_json(data)
                    orare.append(orar)
                except Exception as e:
                    print(f"Eroare la parsarea orarului de la cheia {cheie}:
{e}")
        if cursor == 0:
            break

    return orare

async def incarca_orar_in_cache(r: redis.Redis, orar: Orar) -> None:
    await                    r.set(f"orarCache:{orar.procesare}:{orar.url}",
orar.model_dump_json(), ex=86400)

async def descarca_orar_din_cache(r: redis.Redis, procesare: Procesare, url:
str) -> Orar:
    data = await r.get(f"orarCache:{procesare}:{url}")
    if data is None:
        return None
    return Orar.model_validate_json(data)

async def verificare_orar_cache(r: redis.Redis, procesare: Procesare, url:
str) -> bool:
    data = await r.get(f"orarCache:{procesare}:{url}")
    return data is not None
```

ToolkitUtilizator.py

```
import bcrypt
```

```
from modele.ModeleUtilizator import *
from typing import Union
import redis.asyncio as redis

def creeaza_utilizator(nume: str, parola_clara: str, email: str) -> Utilizator:
    salt = bcrypt.gensalt()
    parola_hash = bcrypt.hashpw(parola_clara.encode('utf-8'), salt)
    return Utilizator(nume=nume, parola=parola_hash.decode('utf-8'), email=email)

def verifica_parola(parola_introdusa: str, item: Union[Utilizator, str]) -> bool:
    if isinstance(item, Utilizator):
        parola_hash = item.parola
    else:
        parola_hash = item

    return                          bcrypt.checkpw(parola_introdusa.encode('utf-8'),
parola_hash.encode('utf-8'))

async def incarca_utilizator_in_redis(r: redis.Redis, utilizator: Utilizator) ->
None:
    await r.set(f"user:{utilizator.nume}", utilizator.model_dump_json())

async def descarca_utilizator_din_redis(r: redis.Redis, nume: str) -> Utilizator |
None:
    data = await r.get(f"user:{nume}")
    if data is None:
        return None
    return Utilizator.model_validate_json(data)

async def verificare_utilizator_duplicat(r: redis.Redis, nume: str) -> bool:
    return await r.exists(f"user:{nume}") > 0

async def obtine_date_utilizator(r: redis.Redis, nume: str) -> bool:
    data = await r.get(f"user:{nume}")
    if data is None:
        return None
    utilizator = Utilizator.model_validate_json(data)
    return utilizator.nume, utilizator.email
```

Fișiere Frontend

App.js

```
import React, { useState, useEffect } from "react";
import PaginaAutentificare from "./componente/PaginaAutentificare";
import PaginaProcesare from "./componente/PaginaProcesare";
import PaginaProfilUtilizator from "./componente/PaginaProfilUtilizator";

function App() {
  const [utilizatorAutentificat, setUtilizatorAutentificat] = useState(null);
  const [panouUtilizator, setPanouUtilizator] = useState(false);

  useEffect(() => {
    const utilizatorSalvat = localStorage.getItem("utilizator");
    const panou = localStorage.getItem("panou_utilizator") === "true";
    if (utilizatorSalvat) {
```

```
    setUtilizatorAutentificat(utilizatorSalvat);
  }
  setPanouUtilizator(panou);
}, []);

const logout = () => {
  localStorage.removeItem("utilizator");
  localStorage.removeItem("panou_utilizator");
  setUtilizatorAutentificat(null);
  setPanouUtilizator(false);
};

return (
  <>
    {utilizatorAutentificat ? (
      !panouUtilizator ? (
        <div style={{ position: "relative", minHeight: "100vh" }}>
          <div
            style={{
              position: "absolute",
              top: "-60px",
              right: "40px",
              textAlign: "right",
            }}
          >
            <div style={{ marginBottom: "10px", fontWeight: "bold" }}>
              Bine ai venit, {utilizatorAutentificat}!
            </div>
            <div
              style={{
                display: "flex",
                gap: "10px",
                justifyContent: "flex-end",
              }}
            >
              <button
                onClick={logout}
                style={{
                  backgroundColor: "#fff",
                  border: "1px solid #333",
                  color: "#333",
                  padding: "6px 12px",
                  borderRadius: "5px",
                  cursor: "pointer",
                  fontSize: "14px",
                }}
              >
                Logout
              </button>
              <button
                onClick={() => {
                  setPanouUtilizator(true);
                  localStorage.setItem("panou_utilizator", true);
                }}
                style={{
                  backgroundColor: "#fff",
                  border: "1px solid #333",
                  color: "#333",
                  padding: "6px 12px",
```

```
            borderRadius: "5px",
            cursor: "pointer",
            fontSize: "14px",
          }}
        >
          Contul meu
        </button>
      </div>
    </div>

    <div style={{ marginTop: "100px" }}>
      <PaginaProcesare numeUtilizator={utilizatorAutentificat} />
    </div>
  </div>
) : (
  <div style={{ position: "relative", minHeight: "100vh" }}>
    <div
      style={{
        position: "absolute",
        top: "-60px",
        right: "40px",
        textAlign: "right",
      }}
    >
      <div style={{ marginBottom: "10px", fontWeight: "bold" }}>
        Bine ai venit, {utilizatorAutentificat}!
      </div>
      <div
        style={{
          display: "flex",
          gap: "10px",
          justifyContent: "flex-end",
        }}
      >
        <button
          onClick={() => {
            setPanouUtilizator(false);
            localStorage.setItem("panou_utilizator", false);
          }}
          style={{
            backgroundColor: "#fff",
            border: "1px solid #333",
            color: "#333",
            padding: "6px 12px",
            borderRadius: "5px",
            cursor: "pointer",
            fontSize: "14px",
          }}
        >
          Înapoi la procesare
        </button>
        <button
          onClick={logout}
          style={{
            backgroundColor: "#fff",
            border: "1px solid #333",
            color: "#333",
            padding: "6px 12px",
            borderRadius: "5px",
```

```
                    cursor: "pointer",
                    fontSize: "14px",
                  }}
                >
                  Logout
                </button>
              </div>
            </div>

            <div style={{ marginTop: "100px" }}>
              <PaginaProfilUtilizator numeUtilizator={utilizatorAutentificat} />
            </div>
          </div>
        )
      ) : (
        <PaginaAutentificare
          onLogin={(nume) => {
            setUtilizatorAutentificat(nume);
          }}
        />
      )}
    </>
  );
}
export default App;
```

PaginaAutentificare.js

```
import React, { useState, useRef } from "react";

function PaginaAutentificare({ onLogin }) {
  const [esteInregistrare, setEsteInregistrare] = useState(false);
  const [nume, setNume] = useState("");
  const [parola, setParola] = useState("");
  const [email, setEmail] = useState("");
  const [eroare, setEroare] = useState("");

  const butonRef = useRef();
  const numeRef = useRef();
  const parolaRef = useRef();
  const emailRef = useRef();

  const handleEnter = (e, nextRef) => {
    if (e.key === "Enter") {
      e.preventDefault();
      if (nextRef === butonRef) {
        if (butonRef.current) {
          butonRef.current.click();
        }
      } else if (nextRef && nextRef.current) {
        nextRef.current.focus();
      }
    }
  };

  const salveazaUtilizator = (numeUtilizator) => {
    localStorage.setItem("utilizator", numeUtilizator);
    onLogin(numeUtilizator);
```

```
};

const trimiteAutentificare = async () => {
  if (!nume || !parola) {
    alert("Nu au fost introduse date pentru login");
    return;
  }
  try {
    const response = await fetch("http://localhost:5050/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ nume, parola }),
    });
    const data = await response.json();
    if (response.ok) {
      salveazaUtilizator(nume);
    } else {
      alert(data.detail || "Eroare la autentificare");
    }
  } catch (e) {
    alert("Eroare la conectarea cu serverul");
  }
};

const trimiteInregistrare = async () => {
  if (!nume || !parola || !email) {
    alert("Nu au fost introduse date pentru inregistrare");
    return;
  }
  try {
    const response = await fetch("http://localhost:5050/register", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ nume, parola, email }),
    });
    const data = await response.json();
    if (response.ok) {
      salveazaUtilizator(nume);
    } else {
      alert(data.detail || "Eroare la înregistrare");
    }
  } catch (e) {
    alert("Eroare la conectarea cu serverul");
  }
};

return (
  <div
    style={{
      display: "flex",
      flexDirection: "column",
      alignItems: "center",
      justifyContent: "start",
      height: "100vh",
      paddingTop: "80px",
      fontFamily: "Arial",
      fontSize: "18px",
    }}
  >
```

```
<h2 style={{ fontSize: "28px", marginBottom: "20px" }}>
  {esteInregistrare ? "Register" : "Login"}
</h2>

<input
  type="text"
  placeholder="Nume"
  value={nume}
  ref={numeRef}
  onChange={(e) => setNume(e.target.value)}
  onKeyDown={(e) => handleEnter(e, parolaRef)}
  style={{
    margin: "10px",
    padding: "10px",
    width: "280px",
    fontSize: "16px",
  }}
/>

<input
  type="password"
  placeholder="Parolă"
  value={parola}
  ref={parolaRef}
  onChange={(e) => setParola(e.target.value)}
  onKeyDown={(e) =>
    handleEnter(e, esteInregistrare ? emailRef : butonRef)
  }
  style={{
    margin: "10px",
    padding: "10px",
    width: "280px",
    fontSize: "16px",
  }}
/>

{esteInregistrare && (
  <input
    type="email"
    placeholder="Email"
    value={email}
    ref={emailRef}
    onChange={(e) => setEmail(e.target.value)}
    onKeyDown={(e) => handleEnter(e, butonRef)}
    style={{
      margin: "10px",
      padding: "10px",
      width: "280px",
      fontSize: "16px",
    }}
  />
)}

<button
  ref={butonRef}
  onClick={esteInregistrare ? trimiteInregistrare : trimiteAutentificare}
  style={{
    margin: "20px",
    padding: "10px 20px",
```

```
          width: "280px",
          fontSize: "16px",
          backgroundColor: "#63C5DA",
          color: "white",
          border: "none",
          borderRadius: "4px",
          cursor: "pointer",
        }}
      >
        {esteInregistrare ? "Register" : "Login"}
      </button>

      <div
        style={{
          fontSize: "14px",
          color: "#007BFF",
          cursor: "pointer",
          textDecoration: "underline",
        }}
        onClick={() => {
          setEsteInregistrare(!esteInregistrare);
        }}
      >
        {esteInregistrare ? "To Login" : "New account? Register now"}
      </div>

      {eroare && (
        <p style={{ color: "red", marginTop: "15px", fontSize: "14px" }}>
          {eroare}
        </p>
      )}
    </div>
  );
}

export default PaginaAutentificare;
```

PaginaProcesare.js

```
import React, { useState } from "react";
import PaginaProfilUtilizator from "./PaginaProfilUtilizator";

function PaginaProcesare({ numeUtilizator }) {
  const [url, setUrl] = useState("");
  const [procesare, setProcesare] = useState("Regex");
  const [raspuns, setRaspuns] = useState(null);
  const [eroare, setEroare] = useState("");

  const trimiteCerereProcesare = async () => {
    const CerereProcesare = {
      url,
      procesare,
      nume_utilizator: numeUtilizator,
    };

    try {
      const response = await fetch("http://localhost:5050/proceseaza-orar", {
```

```
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(CerereProcesare),
    });

    if (!response.ok) {
      alert("Eroare la cererea de procesare");
    }

    const data = await response.json();
    setRaspuns(data);
  } catch (e) {
    alert("Eroare la cererea de procesare: " + e);
  }
};

const salveazaOrar = async () => {
  if (!raspuns || !raspuns.activitati) {
    alert("Nu există un orar de salvat.");
    return;
  }

  try {
    const response = await fetch("http://localhost:5050/incarca-orar", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(raspuns),
    });

    const data = await response.json();
    alert(data.mesaj || "Orarul a fost salvat cu succes.");
  } catch (e) {
    alert("Eroare la salvarea orarului:", e);
  }
};

const thStyle = {
  padding: "10px",
  border: "1px solid #ddd",
  textAlign: "left",
};

const tdStyle = {
  padding: "8px",
  border: "1px solid #ddd",
};

return (
  <div
    style={{
      display: "flex",
      flexDirection: "column",
      alignItems: "center",
      marginTop: "40px",
      fontFamily: "Arial, sans-serif",
      padding: "0 20px",
    }}
  >
    <div
```

```
  style={{
    width: "100%",
    maxWidth: "400px",
    display: "flex",
    flexDirection: "column",
    gap: "10px",
  }}
>
  <h1 style={{ textAlign: "center" }}>Procesare Orar</h1>

  <label>
    URL orar:
    <input
      value={url}
      onChange={(e) => setUrl(e.target.value)}
      style={{ width: "100%", padding: "6px" }}
    />
  </label>

  <label>
    Tip procesare:
    <select
      value={procesare}
      onChange={(e) => setProcesare(e.target.value)}
      style={{ width: "100%", padding: "6px" }}
    >
      <option value="Regex">Regex</option>
      <option value="AI">AI</option>
    </select>
  </label>

  <button
    onClick={trimiteCereretProcesare}
    style={{
      width: "100%",
      padding: "10px",
      backgroundColor: "#63C5DA",
      border: "none",
      color: "white",
      cursor: "pointer",
    }}
  >
    Trimite
  </button>
  <button
    onClick={salveazaOrar}
    style={{
      width: "100%",
      padding: "10px",
      backgroundColor: "#63C5DA",
      border: "none",
      color: "white",
      cursor: "pointer",
    }}
  >
    Salvează tabel
  </button>
  {eroare && <p style={{ color: "red", marginTop: "10px" }}>{eroare}</p>}
</div>
```

```
      {raspuns && raspuns.activitati && (
        <div
          style={{
            marginTop: "40px",
            overflowX: "auto",
            width: "100%",
            maxWidth: "1000px",
          }}
        >
          <table
            style={{
              width: "100%",
              borderCollapse: "collapse",
              boxShadow: "0 0 10px rgba(0,0,0,0.1)",
              fontSize: "14px",
            }}
          >
            <thead style={{ backgroundColor: "#63C5DA", color: "white" }}>
              <tr>
                <th style={thStyle}>Nume</th>
                <th style={thStyle}>Profesor</th>
                <th style={thStyle}>Sala</th>
                <th style={thStyle}>Zi</th>
                <th style={thStyle}>Interval</th>
                <th style={thStyle}>Durata</th>
                <th style={thStyle}>Grupe</th>
                <th style={thStyle}>Anul</th>
                <th style={thStyle}>Categorie</th>
                <th style={thStyle}>Paritate</th>
              </tr>
            </thead>
            <tbody>
              {raspuns.activitati.map((act, index) => (
                <tr
                  key={index}
                  style={{
                    backgroundColor: index % 2 === 0 ? "#f9f9f9" : "#ffffff",
                  }}
                >
                  <td style={tdStyle}>{act.nume}</td>
                  <td style={tdStyle}>{act.profesor}</td>
                  <td style={tdStyle}>{act.sala}</td>
                  <td style={tdStyle}>{act.zi}</td>
                  <td style={tdStyle}>{act.interval}</td>
                  <td style={tdStyle}>{act.durata}</td>
                  <td style={tdStyle}>{act.grupe.join(", ")}</td>
                  <td style={tdStyle}>{act.anul}</td>
                  <td style={tdStyle}>{act.categorie}</td>
                  <td style={tdStyle}>{act.paritate}</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
    </div>
  );
}
```

```
export default PaginaProcesare;
```

PaginaProfilUtilizator.js

```
import React, { useEffect, useState } from "react";

function PaginaProfilUtilizator({ numeUtilizator }) {
  const [orare, setOrare] = useState([]);
  const [indexCurent, setIndexCurent] = useState(0);
  const [email, setEmail] = useState("");

  useEffect(() => {
    const fetchDate = async () => {
      try {
        const orareResp = await fetch(
          `http://localhost:5050/orare-utilizator/${numeUtilizator}`
        );
        const orareData = await orareResp.json();
        setOrare(orareData);

        const userResp = await fetch(
          `http://localhost:5050/utilizator/${numeUtilizator}`
        );
        const userData = await userResp.json();
        setEmail(userData.email);
      } catch (e) {
        alert("Eroare la încărcarea datelor utilizatorului:", e);
      }
    };

    fetchDate();
  }, [numeUtilizator]);

  const stergeOrar = async () => {
    const orarDeSters = orare[indexCurent];
    try {
      const response = await fetch("http://localhost:5050/orar/sterge", {
        method: "DELETE",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(orarDeSters),
      });
      if (!response.ok) {
        const e = await response.json();
        alert("Eroare la ștergerea orarului");
        return;
      }
      const orareNoi = [...orare];
      orareNoi.splice(indexCurent, 1);
      setOrare(orareNoi);
      setIndexCurent(Math.max(0, indexCurent - 1));
    } catch (e) {
      alert("Eroare la ștergerea orarului");
    }
  };

  const orarCurent = orare[indexCurent];
```

```
return (
  <div style={{ maxWidth: "900px", margin: "0 auto", paddingTop: "50px" }}>
    <h2>Profil Utilizator</h2>
    <p>
      <strong>Nume utilizator:</strong> {numeUtilizator}
    </p>
    <p>
      <strong>Email:</strong> {email}
    </p>
    <p>
      <strong>Număr orare salvate:</strong> {orare.length}
    </p>

    {orarCurent && (
      <div>
        <div
          style={{
            display: "flex",
            justifyContent: "space-between",
            alignItems: "center",
            marginBottom: "10px",
          }}
        >
          <div
            style={{
              display: "flex",
              gap: "10px",
              alignItems: "center",
              marginBottom: "15px",
            }}
          >
            {indexCurent > 0 && (
              <button
                onClick={() => setIndexCurent(indexCurent - 1)}
                style={{
                  padding: "10px 20px",
                  fontSize: "16px",
                  color: "white",
                  backgroundColor: "#6db8d6",
                  border: "none",
                  borderRadius: "5px",
                  cursor: "pointer",
                }}
              >
                Înapoi
              </button>
            )}
            {indexCurent < orare.length - 1 && (
              <button
                onClick={() => setIndexCurent(indexCurent + 1)}
                style={{
                  padding: "10px 20px",
                  fontSize: "16px",
                  color: "white",
                  backgroundColor: "#6db8d6",
                  border: "none",
                  borderRadius: "5px",
                  cursor: "pointer",
                }}
```

```
        >
          Înainte
        </button>
      )}
    </div>
    <button
      onClick={stergeOrar}
      style={{
        padding: "10px 20px",
        fontSize: "16px",
        backgroundColor: "#fff",
        border: "2px solid red",
        color: "red",
        borderRadius: "5px",
        cursor: "pointer",
        marginLeft: "auto",
      }}
    >
      Șterge orar
    </button>
  </div>

  <table style={{ width: "100%", borderCollapse: "collapse" }}>
    <thead style={{ backgroundColor: "#63C5DA", color: "white" }}>
      <tr>
        <th>Nume</th>
        <th>Profesor</th>
        <th>Sala</th>
        <th>Zi</th>
        <th>Interval</th>
        <th>Durata</th>
        <th>Grupe</th>
        <th>Anul</th>
        <th>Categorie</th>
        <th>Paritate</th>
      </tr>
    </thead>
    <tbody>
      {orarCurent.activitati.map((act, index) => (
        <tr
          key={index}
          style={{
            backgroundColor: index % 2 === 0 ? "#f9f9f9" : "#ffffff",
          }}
        >
          <td style={{ border: "1px solid #ddd", padding: "8px" }}>
            {act.nume}
          </td>
          <td style={{ border: "1px solid #ddd", padding: "8px" }}>
            {act.profesor}
          </td>
          <td style={{ border: "1px solid #ddd", padding: "8px" }}>
            {act.sala}
          </td>
          <td style={{ border: "1px solid #ddd", padding: "8px" }}>
            {act.zi}
          </td>
          <td style={{ border: "1px solid #ddd", padding: "8px" }}>
            {act.interval}
```

```
                  </td>
                  <td style={{ border: "1px solid #ddd", padding: "8px" }}>
                    {act.durata}
                  </td>
                  <td style={{ border: "1px solid #ddd", padding: "8px" }}>
                    {act.grupe.join(", ")}
                  </td>
                  <td style={{ border: "1px solid #ddd", padding: "8px" }}>
                    {act.anul}
                  </td>
                  <td style={{ border: "1px solid #ddd", padding: "8px" }}>
                    {act.categorie}
                  </td>
                  <td style={{ border: "1px solid #ddd", padding: "8px" }}>
                    {act.paritate}
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
    </div>
  );
}

export default PaginaProfilUtilizator;
```