Universitatea Stefan cel Mare din Suceava
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Domeniul: Calculatoare și Tehnologia Informației
Program de studii: Calculatoare

Eficientizarea activităților din cadrul secretariatului facultății prin implementarea unui asistent conversațional bazat pe modele lingvistice mari

Profesor îndrumător
prof. dr. ing. Turcu Cristina Elena

Student
Șandru Alexandru

Iulie 2025

Cuprins

Listă de Figuri

1. Introducere

Acest proiect propune o nouă funcționalitate ce poate fi integrată în site-ul oficial al Facultății de Inginerie Electrică și Știința Calculatoarelor (FIESC), sub forma unui asistent virtual inteligent denumit Alex. Scopul principal al proiectul este automatizarea procesului de informare a studenților, reducând astfel interacțiunea directă cu secretariatul și timpul pierdut în căutarea informațiilor pe site.

Alex este conceput pentru a răspunde întrebărilor frecvente ale studenților și pentru a genera adeverințe solicitate de aceștia.

Interacțiunea cu Alex are loc printr-o interfață de tip chat, ce poate fi integrată direct în site-ul facultății. După autentificare, utilizatorul beneficiază de funcționalități suplimentare, în funcție de drepturile de acces precum: generarea de adeverințe, păstrarea istoricului conversațiilor și adresarea întrebărilor într-un mod mai simplu, fără a mai fi necesară reintroducerea constantă a datelor personale. În plus, răspunsurile devin mai relevante datorită păstrării contextului conversațiilor anterioare.

Acest proiect demonstrează modul în care inteligența artificială poate fi integrată eficient într-o aplicație web educațională, pentru a îmbunătăți accesul la informații și experiența utilizatorului.

1.1. Descrierea problemei

Studenții facultății întâmpină adesea probleme cu lucruri precum documentele necesare în legătură cu bursele, taxele, grupurile și așa mai departe, din cauza programului limitat de lucru cu publicul al secretariatului. În multe cazuri, acest lucru implică statul la cozi lungi și pierderea unui timp considerabil doar pentru a solicita o simplă adeverință sau pentru a cere o informație.

De asemenea, multe dintre întrebările adresate secretariatului au deja răspunsuri disponibile pe site-ul facultății, însă acestea pot fi greu de găsit din cauza volumului mare de informații. Prin urmare, era necesar un sistem care să facă mai eficient accesul de informații și să reducă interacțiunile fizice inutile.

1.2. Justificarea temei

Studenții se confruntă adesea cu dificultăți atunci când trebuie să obțină informații sau adeverințe de la secretariat, mai ales din cauza programului scurt de lucru și a cozilor lungi. De

asemenea, multe dintre întrebările lor au deja răspunsuri pe site-ul facultății, deși acele răspunsuri sunt greu de găsit.

Alex a fost conceput pentru a aborda aceste probleme. Oferă studenților acces aproape instantaneu la informațiile specifice de care au nevoie, cum ar fi informații despre taxe, burse sau grupa din care fac parte. Studenții pot genera adeverințe, fără a fi nevoie să aștepte la cozi sau să interacționeze fizic cu secretariatul.

Prin urmare, proiectul își propune să economisească timp și să simplifice procesul de obținere a informațiilor și obținerea unei adeverințe, oferind o soluție rapidă și accesibilă pentru studenți. În același timp, ajută la digitalizarea activităților administrative, reducând interacțiunile fizice și îmbunătățind eficiența întregului sistem.

1.3. Situația actuală pe plan național/internațional

În România, implementarea asistenților virtuali în mediul educațional este încă la început, dar pe care unele instituții au implementat deja astfel de soluții pentru a sprijini studenții și a îmbunătăți accesul la informații și servicii administrative.

Ca exemplu există Academia de Studii Economice din București, care a introdus „Saro”, un asistentul virtual bazat pe inteligență artificială care oferă studenților posibilitatea de a obține informații academice și administrative. Proiectul a fost finalizat cu sprijinul unui grant de 700.000 de dolari de la Google.org, cu scopul de a sprijini digitalizarea activităților administrative ale universității.

Această inițiativă ilustrează direcția integrării inteligenței artificiale în educație, chiar dacă utilizarea asistenților virtuali în România este încă la început. Aceste proiectele existente au scopul îmbunătățirea accesului la informații și să minimizeze timpului pierdut de studenți în procesele administrative.

2. Tehnologii utilizate

Pentru realizarea proiectului *Alex – Asistent Virtual* au fost utilizate tehnologii moderne, atât pe partea de server (back-end), cât și pe partea de interfață (front-end). Alegerea acestora a fost făcută astfel încât să permită integrarea eficientă a unui model de inteligență artificială, dar și o experiență plăcută și intuitivă pentru utilizator.

2.1. Back-End

2.1.1. Django

Pentru partea de back-end, aplicația folosește Django, un framework web scris în limbajul de programare Python. Acesta are mai multe roluri esențiale:

- expune un API REST prin care interfața din browser comunică cu serverul,
- gestionează autentificarea utilizatorilor și drepturile de acces,
- salvează întrebările și răspunsurile în baza de date,
- procesează comenzile speciale, cum ar fi generarea de adeverințe în format PDF,
- accesează fișierele și extrage informațiile pentru procesul de generare a răspunsurilor.

2.1.2. Integrarea cu LLaMA 3.2 și RAG

Pentru generarea de răspunsuri inteligente, sistemul integrează modelul LLaMA 3.2 (Large Language Model Meta AI 3.2) printr-un sistem de tip RAG (Retrieval-Augmented Generation). Procesul funcționează astfel:

- Când utilizatorul trimite o întrebare, aceasta este salvată și analizată.
- Se extrag cuvintele.
- Aceste cuvinte sunt folosite pentru a căuta documente relevante care conțin informațiile necesare.
- Conținutul documentului găsit este extras și folosit drept context.
- Întrebarea și contextul sunt trimise către LLaMA 3.2, care generează un răspuns corespunzător întrebarii.
- Răspunsul este apoi validat automat și trimis către utilizator.

În cazul în care întrebarea este o comandă, cum ar fi „Generează o adeverință cu motivul [motiv]”, sistemul:

- Verifică dacă utilizatorul este autentificat.

- Încarcă șablonul HTML.
- Completează automat datele personale.
- Generează adeverința pe care îl convertește la PDF.
- Returnează un link de descărcare.

2.2. Front-End

2.2.1. React + Vite

Pentru partea de front-end, aplicația este dezvoltată utilizând React, împreună cu Vite pentru a încărcare rapidă și un flux de lucru modern.

Utilizatorul interacționează astfel:

- La accesarea site-ului, utilizatorul este întâmpinat de o versiune a paginii oficiale a FIESC.
- În colțul din dreapta jos al paginii se află un buton numit „Alex – Asistent Virtual”.
- La apăsarea acestuia, se deschide o fereastră de tip chat, unde utilizatorul poate pune întrebări.

Dacă utilizatorul se autentifică, are acces la funcții suplimentare:

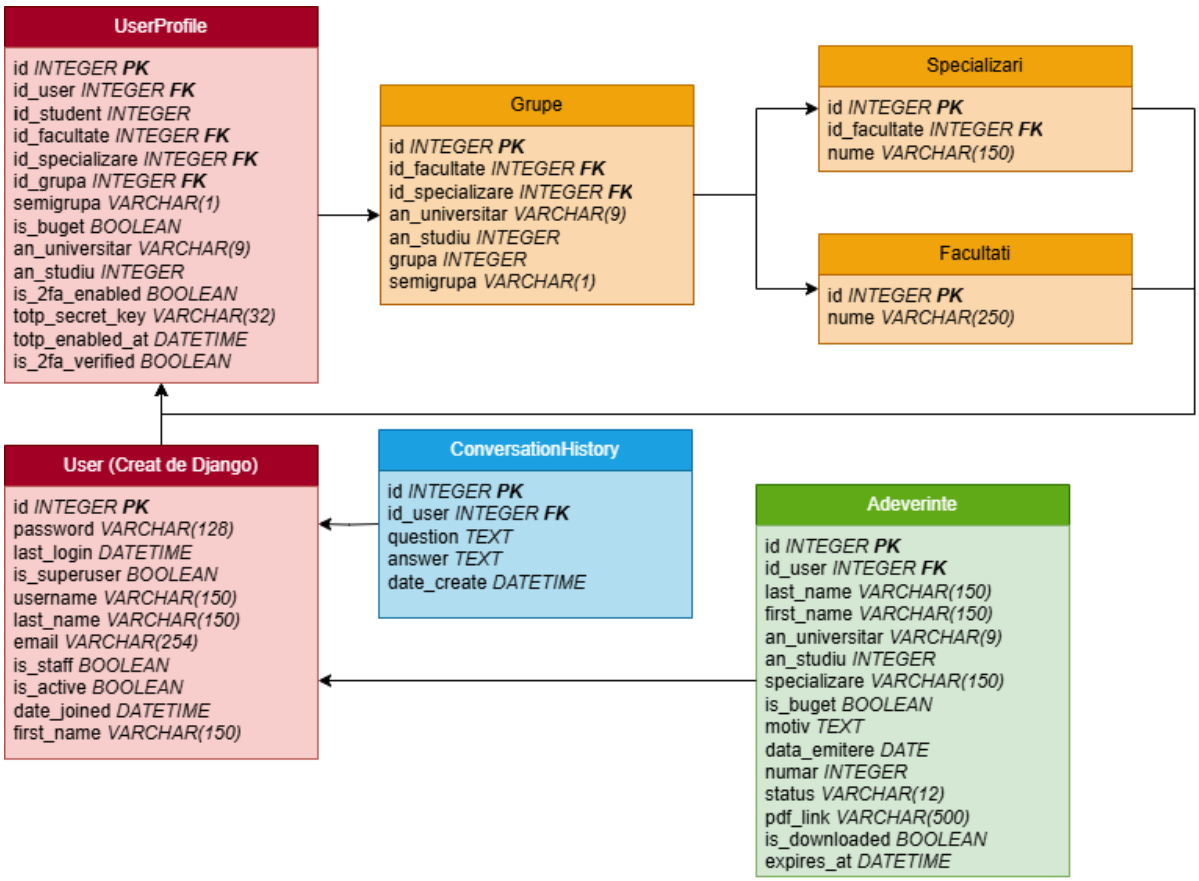
- Poate genera adeverințe.
- Are acces la istoricul conversațiilor.
- Răspunsurile devin mai relevante datorită păstrării contextului conversațiilor anterioare.

3. Considerații de proiectare

3.1. Proiectarea bazei de date

Baza de date este implementată utilizând SQLite, un sistem de gestiune a bazelor de date relaționale, integrat în framework-ul Django. Aceasta conține mai multe tabele care stochează informațiile necesare pentru funcționarea aplicației. Structura este construită astfel încât să permită extinderea ușoara a sistemului.

În Figura 3.1 este prezentată schema bazei de date utilizate în cadrul aplicației, care descrie structura logica a tabelelor și relațiile dintre acestea.



Figură 3.1 – Schema bazei de date

3.1.1. Tabele și relațiile dintre tabele

Baza de date este compusă din următoarele tabele:

1. Facultăți:

- Conține lista facultăților din cadrul instituției.
- Câmpuri: id (PK), nume.

2. Specializări:
- Reprezintă specializările disponibile în cadrul unei facultăți.
- Relație: FK către Facultăți.
- Câmpuri: id (PK), id_facultate (FK), nume.

3. Grupe:
- Conține informațiile despre grupele și semi-grupele studenților.
- Relații: FK către Facultăți și Specializări.
- Câmpuri: id (PK), id_facultate (FK), id_specializare (FK), an_universitar, an_studiu, grupa, semigrupa.

4. User (creat automat de Django):
- Conține datele de baza ale conturilor utilizatorilor.
- Câmpuri: id (PK), password, last_login, is_superuser, username, last_name, email, is_staff, is_active, date_joined, first_name.

5. UserProfile:
- Extinde informațiile despre utilizatorii înregistrați.
- Relații: FK către User, Facultăți, Specializări și Grupe.
- Câmpuri: id (PK), id_user (FK), id_student, id_facultate (FK), id_specializare (FK), id_grupa (FK), semigrupa, is_buget, an_universitar, an_studiu, is_2fa_enabled, totp_secret_key, totp_enabled_at, is_2fa_verified

6. Adeverințe:
- Păstrează datele despre adeverințele generate de utilizatori.
- Relație: FK către User.
- Câmpuri: id (PK), id_user (FK), last_name, first_name, an_universitar, an_studiu, specializare, is_buget, motiv, data_emitere, numar, status, pdf_link, is_downloaded, expires_at.

7. ConversationHistory:
- Stochează istoricul conversației cu asistentul virtual.
- Relație: FK către User.
- Câmpuri: id (PK), id_user (FK), question, answer, date_create.

3.2. Proiectare Back-End

Partea back-end este proiectat în limbajul de programare Python, folosind framework-ul Django, pentru dezvoltarea aplicațiilor web. Acesta gestionează logica aplicației, interacțiunile cu baza de date și comunicarea cu front-end.

3.2.1. Funcționalități administrative

Sistemul include o serie de endpointuri pentru activitățile administrative, accesibile doar de către conturile cu rol de administrator. Administratorii au posibilitatea de a adăuga utilizatori noi în baza de date folosind un fișier Excel. Aceștia mai au posibilitatea de a actualiza toate documentele PDF, utilizate de modelul LLaMA 3.2 în cadrul sistemului de RAG, pentru a putea genera răspunsuri corecte și relevante pe baza informațiilor.

3.2.2. Funcționalități de securitate

Utilizatorii au posibilitatea de a se autentifica în aplicație pentru a beneficia de funcționalități suplimentare cum ar fi:

- Posibilitatea de a folosi următoarele comenzi:
  - „comenzi” – afișează toate comenzile valabile;
  - „Generează o adeverința cu motivul [motiv]” – informațiile aflate în baza de date corespunzătoare utilizatorului, sunt folosite pentru a genera o adeverința pe care o poți descarcă și folosi imediat.
- Salvarea istoricul conversației pentru a face conversația cu LLaMA 3.2 mai ușoară, deoarece utilizatorii nu mai trebuie să repete informațiile care au fost oferite anterior prin anumite întrebări.
- Pentru utilizatorii care își doresc să își creeze un cont trebuie sa folosească un email de student aflat în baza de date.

3.2.3. Procesare întrebărilor și comenzilor

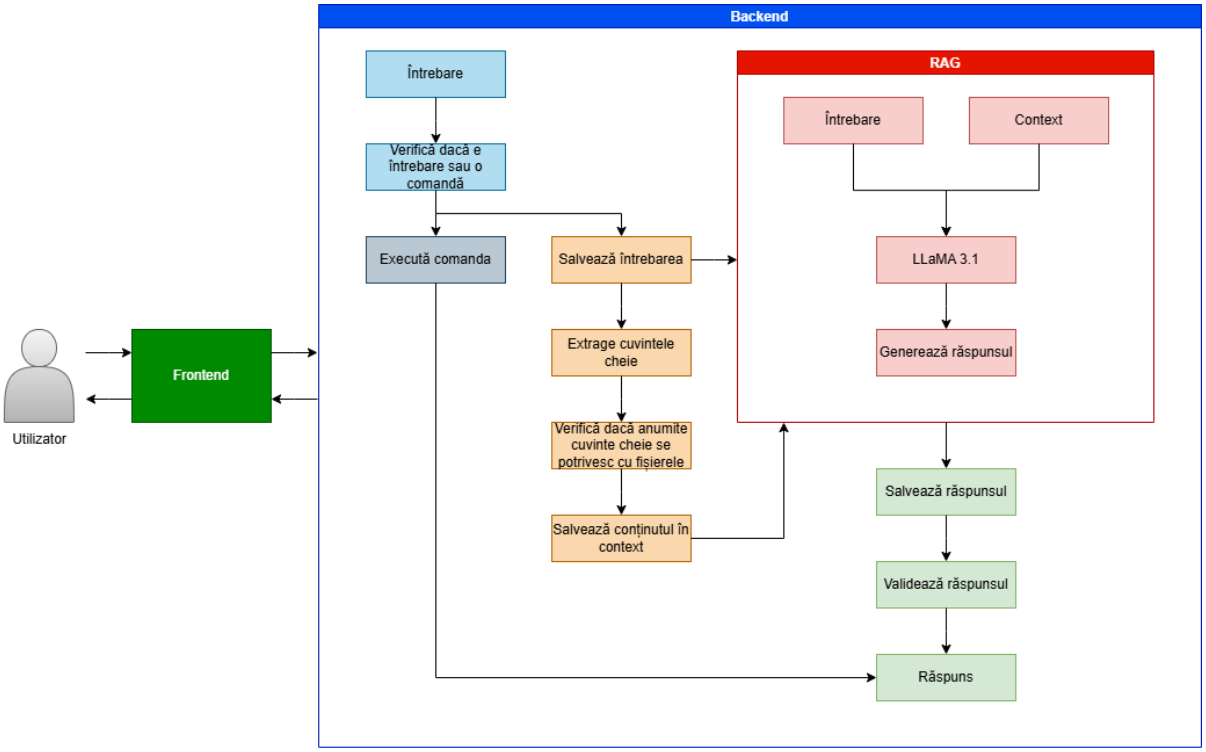
Când utilizatorul interacționează cu sistemul prin interfața de tip chat:

- Întrebarea este trimisă de la front-end către back-end.
- Back-end verifică dacă este întrebare sau comandă.
- Dacă este o întrebare:
  - Întrebarea se salvează;

- Cuvintele din întrebare, se verifică dacă se potrivesc cu un cuvânt aflat în numele unui document;
- Conținutul acestuia este salvat si trimis la modulul de RAG, pentru a genera un răspuns;
- Răspunsul este validat și returnat către utilizator.

- Dacă este o comandă:
  - se verifică dacă utilizatorul este autentificat;
  - după se verifică ce comandă a fost executată:
    - dacă e „comenzi” returnează toate comenzile valabile;
    - dacă e „Generează o adeverința cu motivul [motiv]” va returna un link către un fișier PDF care conține o adeverință completată automat pe baza informației corespunzătoare utilizatorului autentificat.

În Figura 3.2 este prezentată schema de funcționare a componentei principale back-end, care ilustrează fluxul de date de la utilizator către serverul aplicației.



Figură 3.2 – Schema de funcționare back-end

3.3. Proiectare Front-End

Partea front-end este realizată utilizând biblioteca JavaScript React, împreuna cu Vite pentru un timp de compilare redus. Aceasta este responsabilă de interacțiunea directă cu utilizatorul:

- Afișarea datelor preluate de la back-end;
- Transmiterea cererilor către server.

3.3.1. Paginile si componentele aplicatiei

Aplicația front-end este alcătuită din cinci pagini și două componente. Fiecare pagină contribuie la definirea structurii aplicației și a fluxul de utilizare, având un rol important în interacțiunea utilizatorului.

Componente:

- Frame – componenta care afișează site-ul oficial al facultății prin intermediul unui iframe.
- ChatWindow – punct de acces principal către funcționalitatea de chat cu asistentul virtual, rapid și ușor de accesat pentru întrebări scurte, aici se află punctul de plecare către LoginPage.

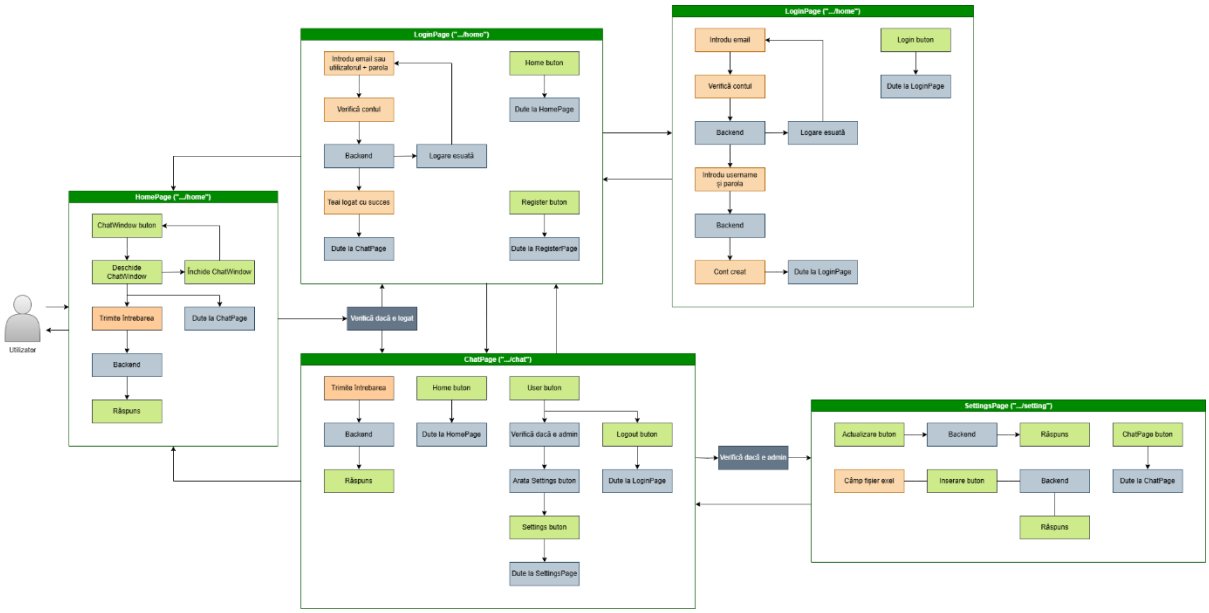
Pagini:

- HomePage – pagina principală care include componentele Frame și ChatWindow, aceasta este punctul de plecare pentru utilizatori, oferind acces rapid la chat și vizualizarea site-ului facultății.
- LoginPage – pagina de autentificare, unde utilizatorii își introduc datele pentru accesul în sistem, dacă autentificare reușește utilizatorul este redirecționat către ChatPage, dacă utilizatorul nu are cont își poate crea unul accesând butonul „Nu ai cont? Creaza unul!” unde va fi redirecționat către RegisterPage, in LoginPage se mai află un buton pentru a pleca înapoi spre pagina de pornire mai exact HomePage.
- RegisterPage – pagina de înregistrare, unde utilizatorii noi își pot crea un cont, cu condiția să aibă un email de student existent in baza de date, aici există un buton pentru a pleca înapoi spre pagina de autentificare mai exact LoginPage.
- ChatPage – pagina destinată utilizatorilor autentificați, oferind o interfață extinsă de chat care include istoricul conversațiilor, în această pagină exista un buton unde vei fi redirecționat către pagina de start mai exact HomePage, și un buton dropdown care conține butonul de deconectare unde vei fi redirecționat către LoginPage, care e

valabil doar pentru administratori mai au un buton de setări unde vor fi redirecționați către pagina de setări adică SettingsPage.

- SettingsPage – pagina accesibilă doar de administratori, ce permite actualizarea documentelor folosințe de asistentul virtual și importul utilizatorilor noi printr-un fișier Excel, aici mai există un buton unde vei fi redirecționat către ChatPage.

În Figura 3.3 este prezentată schema de funcționare a componentei front-end, care ilustrează fluxul interacțiunii dintre utilizator și interfața aplicației.



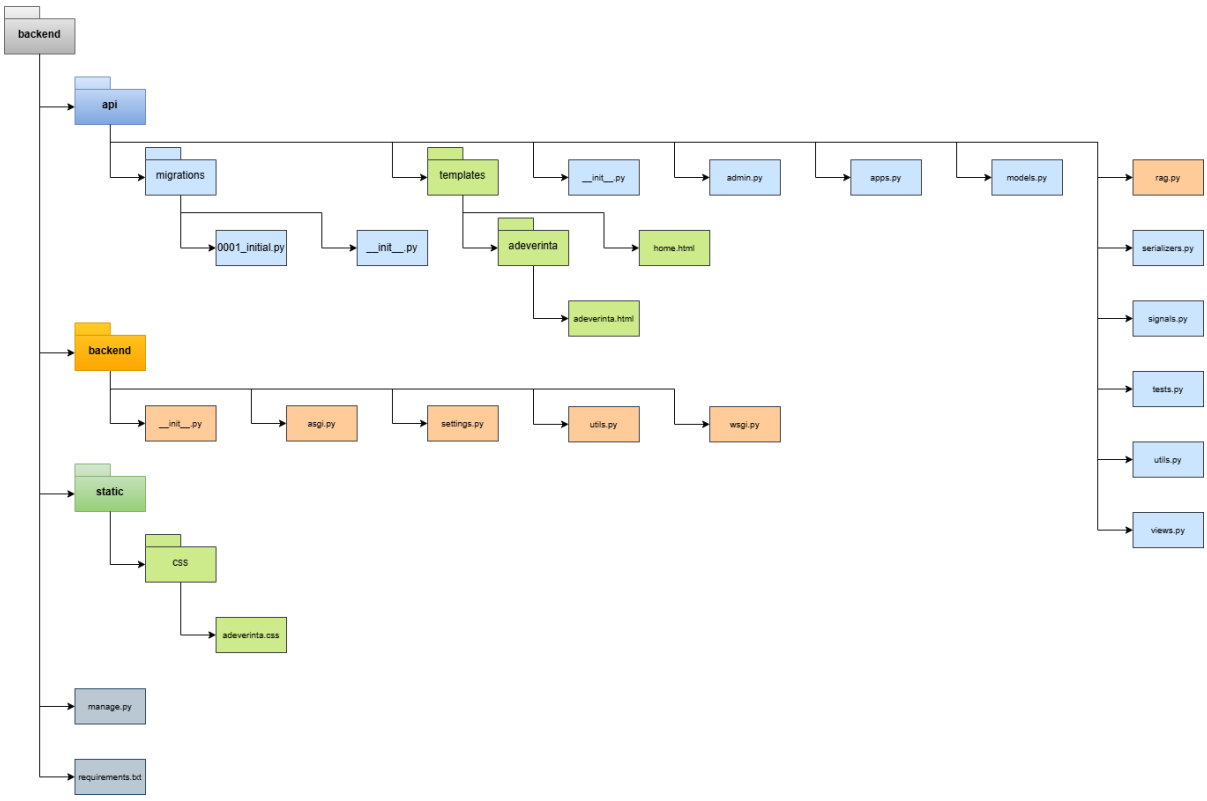
Figură 3.3 – Schema de funcționare front-end

4. Considerente de implementare

4.1. Implementarea Back-End

Componenta back-end a aplicației a fost realizata în limbajul de programare Python utilizând framework-ul Django, având rolul de a gestiona autentificarea, stocarea, și procesarea datelor, integrarea cu modelul LLaMA și logica de generare a răspunsurilor. În plus, el oferă funcționalități dedicate utilizatorilor autentificați, cum ar fi afișarea comenzilor disponibile si generarea automată a unei adeverințe în format PDF pe baza datelor din baza de date. Utilizatorii își pot crea conturi, iar istoricul conversației este păstrata pentru un dialog mai fluid cu asistentul virtual.

În Figura 4.1 este prezentat structura directorului back-end care include fișiere si directoarele generate automat de Django, precum si cele create pe parcurs pentru funcționalități personalizate.



Figură 4.1 – Structura back-end

4.1.1. Organizarea fișierelor și directoarelor Back-End

Directorul „/api/” este directorul principal al aplicației. Conține atât fișiere generate automat cât și unele definite manual:

- views.py – include logica principală de procesare a cererilor de la frontend.
- models.py – definește modelele bazei de date.
- rag.py – conține logica pentru sistemul de RAG, care extrage informații din documente pentru a genera răspunsuri precise.
- serializers.py – sterilizează modelele pentru a fi trimise ca JSON în frontend.
- signals.py – folosit pentru a executa automat anumite evenimente.
- admin.py, apps.py, tests.py – fișiere Django standard pentru înregistrarea modelelor în panoul admin, configurarea aplicației si testare.
- urls.py – definește rutele aplicației back-end, legând adresele acesteia de funcțiile din views.py.

Directorul „/api/migrations" este generat automat de Django. Fișierul 0001_initial.py conține instrucțiuni pentru crearea tabelelor în baza de date, pe baza definițiilor din models.py.

Directorul „/api/templates" conține șabloane HTML pentru afișarea conținutului dinamic:

- home.html – pagina principală a aplicației.
- adeverinta/adeverinta.html – șablonul folosit pentru generarea adeverinței PDF, care este completat dinamic cu datele utilizatorului

Directorul „/backend/" este directorul central al proiectului Django, care conține fișierele de configurare globală:

- settings.py – aici sunt definite setările esențiale cum ar fi conexiunea la baza de date, aplicațiile instalate, căile către fișiere statice și media, integrare cu LLaMA etc.
- urls.py – rutele globale ale aplicației.
- asgi.py, wsgi.py – fisiere standard pentru rularea aplicației cu servere de producție.

Directorul „/static/css/" conține fișierele CSS folosite pentru stilul vizual a documentelor generate. Fișierul adeverinta.css definește stilul vizual pentru generarea unei adeverințe.

Directorul sursă „backend/" conține doua fișiere principale:

- manage.py – fișier utilizat pentru comenzi administrative, precum migrarea bazei de date sau rularea serverului local.
- requirements.txt – conține toate pachetele necesare rulării aplicației.

4.1.2. Funcționalități și logica de procesare

Componenta principală a aplicației este clasa „ConversationChat” care reprezintă un endpoint REST API responsabil pentru gestionarea interacțiunii dintre utilizatori si asistentul virtual. Acesta are rolul de a prelucra mesajele primite, de a oferi răspunsuri relevante folosind modelul LLaMA integrat cu mecanismul RAG, dar și de a genera o adeverința în format PDF în mod automat. (vezi Anexa 1 pentru codul complet al clasei)

1. Procesare întrebărilor primite

Endpointul primește de la frontend o întrebare. În funcție de conținut se declanșează una dintre următoarele acțiuni:

- Dacă mesajul este „comenzi”, se returnează o listă cu comenzile disponibile, utile pentru ghidarea utilizatorului.
- Dacă mesajul începe cu „genereaza o adeverinta cu motivul”, aplicația declanșează procesul de creare a unei adeverințe in format PDF pe baza profilului utilizatorului.
- În orice alt caz, întrebarea este trimisă către modelul LLaMA 3.2 prin intermediul clasei RAG, care extrage contextul relevant din fișierele PDF și generează un răspuns personalizat.

2. Generarea automată a adeverințelor

Aceasta este una dintre cele mai importante funcționalități ale aplicației. După ce este recunoscută o comanda de tipul „Genereaza o adeverinta cu motivul angajare” sunt parcurși pașii următori:

- Se extrage motivul solicitat de utilizator.
- Se încarcă profilul complet al acestuia din baza de date.
- Se creează o nouă adeverința după model, care conține toate datele necesare
- Conținutul adeverinței este încărcat într-un fișier HTML, în care sunt inserate datele utilizatorului.
- Fișierul HTML este convertit într-un fișier PDF folosind biblioteca WeasyPrint, apoi este salvata în fișier cu un nume unic.
- În final este generat un URL public către fișierul PDF, care este returnat ca răspuns către utilizator.

3. Generarea de răspunsuri personalizate cu RAG + LLaMA

Dacă mesajul primit nu se încadrează printre comenzi, acesta este procesat prin sistemul de RAG (vezi Anexa 2 pentru codul complet al clasei):

- Când utilizatorul trimite o întrebare, aceasta este salvată și analizată.

- Se extrag cuvintele-cheie, ignorând cuvintele comune.
- Aceste cuvinte-cheie sunt folosite pentru a căuta documente relevante care conțin informațiile necesare.
- Conținutul documentului găsit este extras și folosit drept context.
- Întrebarea și contextul sunt trimise către LLaMA 3.2, care generează un răspuns adaptat.
- Răspunsul este apoi validat automat și trimis către utilizator.

4. Alte funcționalități

Pe lângă funcționalitățile de baza, endpointul „ConversationChat” salvează fiecare întrebare și răspuns în istoricul conversației pentru utilizatorii autentificați.

4.1.3. Funcționalități administrative

Aplicația oferă două funcționalități esențiale rezervate administratorilor:

1. Descărcarea automată a documentelor oficiale (.pdf/.doc/.docx)

Această funcționalitate permite descărcarea automata a tuturor fișierelor disponibile de pe site-ul oficial al facultății, pentru apoi a fi folosite la procesarea întrebărilor studenților în mecanismul RAG.

Procesul este declanșat printr-un POST la un endpoint protejat, iar logica de baza este următoarea:

- Se accesează adresa URL specificată.
- Se analizează HTML-ul folosind BeautifulSoup pentru a extrage toate link-urile relevante.
- Se filtrează doar fișierele cu extensiile .pdf, .doc, sau .docx.
- Fișierele sunt descărcate local în directoarele pdfs și docs, în funcție de extensie.

2. Importul de utilizatori dintr-un fișier Excel

Această componentă permite încărcarea automată a datelor studenților pe baza unui fisier .xlsx. logica de baza este următoarea:

- Se primește un fișier Excel prin POST.
- Se extrage anul universitar și se parcurg rândurile începând cu poziția 9.
- Pentru fiecare student se extrag: ID, nume, prenume, email, codul de semigrupa, regimul de taxare.
- Se procesează codul semigrupei care include ID facultate, ID specializare, an de studiu, grupă, semigrupă.
- Se caută entitățile corespondente în baza de date.

- Se creează sau actualizează contul de User si UserProfile

Funcționalitatea asigură o introduce rapidă și eficientă a datelor.

4.1.4. Gestionarea conturilor de utilizator

Principalele funcționalități legate de gestionarea conturilor utilizatorilor în cadrul aplicației, respectiv autentificarea, verificarea existenței unui email și activarea unui cont predefinit.

1. Autentificarea utilizatorilor

Pentru autentificarea utilizatorilor, endpoint-ul LoginView primește în request două câmpuri: un identificator care poate să fie dintre adresa de email sau username-ul, și parola. Se încearcă mai întâi identificarea utilizatorului după email, iar dacă nu se găsește, după username. După identificare, parola este verificată prin funcția de autentificare Django. În cazul în care datele sunt corecte, se generează un token JWT care este returnat clientului împreună cu detalii despre utilizator.

2. Verificarea existenței unui email

Endpoint-ul CheckEmailExistsView este utilizat pentru a verifica dacă o adresă de email este deja înregistrată în sistem. Această funcționalitate este importantă pentru a preveni crearea de conturi duplicate sau pentru validarea datelor înainte de activare.

3. Activarea conturilor

Pentru utilizatorii creați anterior (de exemplu, prin importarea dintr-un fișier Excel), procesul de activare a contului se realizează prin endpoint-ul ActivateUserView. Acesta primește email-ul, username-ul dorit și parola printr-un request POST. Se verifică existența utilizatorului după email și dacă acesta nu are deja un username diferit setat. Dacă toate condițiile sunt îndeplinite, se actualizează username-ul și parola, iar contul este activat.

4.2. Implementarea Front-End

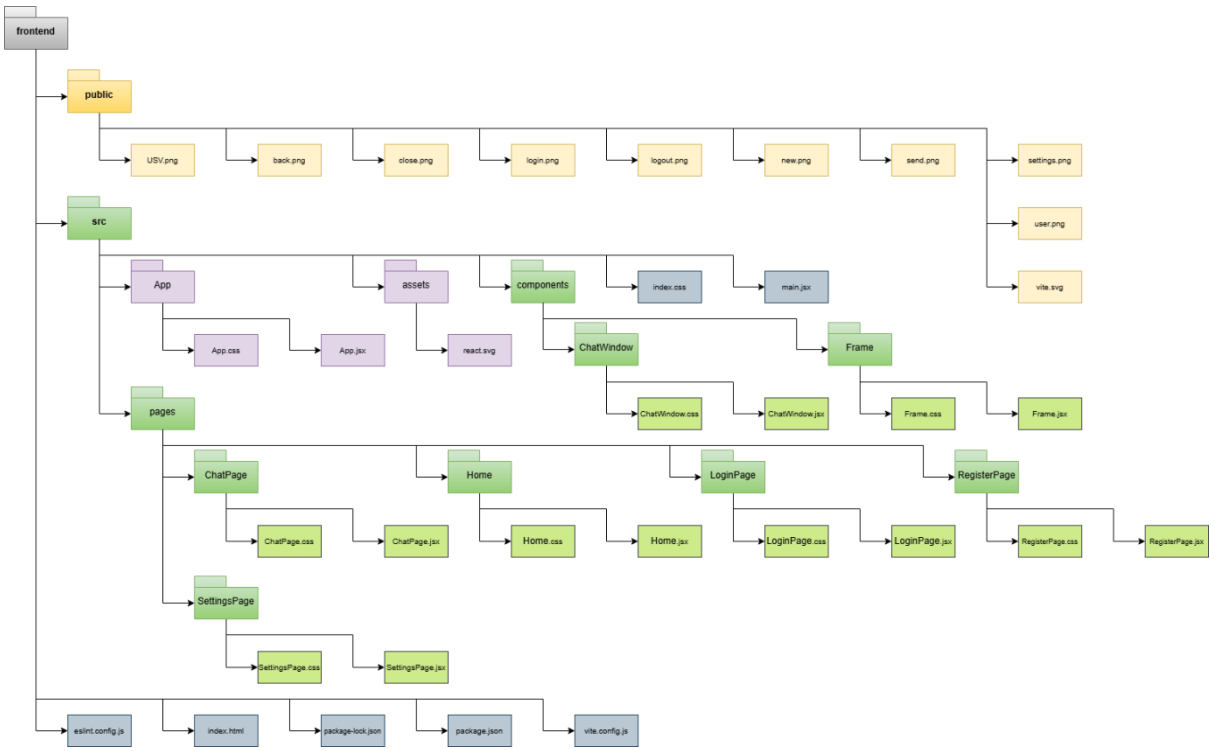
Componenta front-end a aplicației a fost dezvoltata utilizând biblioteca React, împreuna cu instrumentul de build Vite, pentru o experiența de dezvoltare mai rapidă și eficientă. Această componentă are rolul de a furniza o interfața interactiva și intuitivă prin care utilizatorii pot interacționa cu asistentul virtual și cu funcționalitățile aplicației.

Aplicația este organizată modular, fiecare funcționalitate majoră fiind implementată sub forma unei componente React separate.

Interfața comunica cu back-end prin intermediul unor requesturi HTTP utilizând biblioteca Axios, gestionând răspunsurile în mod asincron. S-a acordat o atenție deosebită experienței utilizatorului (UX), fiind implementate validări de formular, mesaje de eroare, feedback vizual și stări de încărcare. De asemenea, s-a utilizat localStorage pentru a păstra sesiunile utilizatorilor autentificați, iar istoricul conversațiilor este afișat într-un mod coerent și ușor de urmărit.

Stilizarea componentelor s-a realizat folosind CSS, asigurând un design responsive și o experiență frumoasă.

În Figura 4.2 este prezentat structura directorului front-end care include fișiere si directoarele generate automat de React + Vite, precum si cele create pe parcurs pentru funcționalități personalizate.



Figură 4.2 – Structura front-end

4.2.1. Organizarea fișierelor și directoarelor Front-End

Directorul „frontend/” este directorul principal al aplicației front-end și conține următoarele fișiere principale:

- eslint.config.js – fisierul de configurare pentru ergurile de linting, care asigură un stil coerent și fără erori in codul JavaScript/React.
- index.html – pagina HTML principală în care se încarcă aplicația React.

- package.json, package-lock.json – gestionează dependențele proiectului și scripturile de rulare și build.
- vite.config.js – configurarea bundler-ului și serverului de dezvoltare Vite, folosit pentru construirea și rularea aplicației.

Directorul „/public/” conține imagini utilizate ca iconițe pentru butoane.

Directorul „/src/” conține codul sursă al aplicației React și este structurat astfel:

- index.css – aspectul vizual globale pentru întreaga aplicație.
- main.jsx – punctul de intrare al aplicației React, care montează componenta principala în pagina HTML.

Directorul „/App/” conține componenta principala a aplicației:

- App.jsx – componenta React care gestionează structura generală și routing-ul aplicației.
- App.css – aspectul vizual al componentei App.

Directorul „/assets/” este director pentru resurse media.

Directorul „/components/” contine componentele React reutilizabile:

- /ChatWindow/ (ChatWindow.css + ChatWindow.jsx) – componenta pentru fereastra de chat
- /Frame/ (Frame.css + Frame.jsx) – componenta care oferă cadrul vizual general al aplicației

Directorul „/pages/” conține componentele care reprezintă paginile aplicației:

- /ChatPage/ (ChatPage.css + ChatPage.jsx) – pagina pentru funcționalitatea de chat.
- /Home/ (Home.css + Home.jsx) – pagina principală a aplicației.
- /LoginPage/ (LoginPage.css + LoginPage.jsx) – pagina de autentificare a utilizatorilor.
- /RegisterPage/ (RegisterPage.css + RegisterPage.jsx) – pagina pentru înregistrarea utilizatorilor noi
- /SettingsPage/ (SettingsPage.css + SettingsPage.jsx) – pagina pentru setările administratorului.

4.2.2. Descrierea funcțională a componentelor

Componenta Frame este o componentă simplă React care afișează un iframe ce încarcă pagina principală a site-ului FIESC. Această componentă are rolul de a încarca și afisa pagina

oficiala a Facultății de Inginerie Electrică și Știința Calculatoarelor într-un iframe, stfel încat utilizatorul să poată naviga pe site-ul oficial fără a părăsi aplicația/

Componenta ChatWindow reprezintă interfața principală și punctul de intrare pentru funcționalitatea de chat cu asistentul virtual „Alex”. La prima deschidere, componenta trimite automat un mesaj de bun venit către backend, solicitând generarea unui mesaj personalizat de introducere. Mesajul primit este afișat în fereastra de chat, iar utilizatorul poate începe dialogul. Utilizatorul poate introduce mesaje în câmpul de text și le poate trimite fie prin apăsarea butonului, fie prin apăsarea tastei enter. Pe durata generării răspunsului, în interfață este afișat un mesaj de tip „Se generează răspunsul...”. Fereastra de chat poate fi închisă și redeschisă, iar scroll-ul la ultimul mesaj este gestionat automat. Există un buton in dreapta sus pentru navigarea către o pagină de chat extinsă (sub forma unui +).

4.2.3. Descrierea funcțională a paginilor

Pagina Home reprezintă punctul principal de intrare în aplicație, ea este compusă din doua componente, Frame și ChatWindow

Pagina Login reprezintă punctul în care utilizatorul se poate autentifica. Pagina contine un formular de autentificare cu două câmpuri username sau email și parola. La apăsarea butonului „Conectare” sau apăsarea tastei enter, datele introduse unt validate local. Daca datele sunt valide, se face o cerere POST către endpointul de autentificare. Dacă autentificarea reușește, răspunsul conține un token JWT și informații despre utilizator care apoi sunt salvate in localStorage și utilizatorul este redirecționat către pagina de chat. În această pagina mai exista un buton de back aflat în stânga sus a paginii care redirecționează utilizatorul către pagina Home, dar și un buton numit „Nu ai cont? Creează unul!” unde daca utilizatorul îl apasă este redirecționat către pagina Register.

Pagina Register permite crearea unui cont nou pentru utilizatorii care nu dețin unul. Procesul este împărțit în doua etape pentru a verifica validarea emailului de student, și de ai acorda un username si parola. Utilizatorul își va introduce adresa de email de student unde la apăsarea butonului „Continuă” se trimite o cerere GET către endpontul CheckEmailExistsView pentru a verifica dacă emailul este valid și recunoscut în sistem, dacă emailul e valid, sunt afișate câmpurile pentru username și parola unde parola trebuie sa conțină minim 5 caractere, o litera mare, o cifră și un caracter special. Dacă validările trec este trimisă o cerere POST către endpointul ActivateUserView cu datele introduse, dacă înregistrarea este reușită utilizatorul este redirecționat către pagina de login, in caz contrar este afișat un mesaj de eroare. Aceasta pagină mai conține si un buton de back in partea din stânga sus a paginii.

Pagina Chat reprezintă componenta centrală a aplicației. Ea conține interfața de conversație dintre utilizator și asistentul virtual Alex. Aceasta este disponibilă exclusiv pentru utilizatorii autentificați și permite afișarea istoricului conversațiilor, trimiterea de mesaje și primirea de răspunsuri generate de modelul AI. La încărcarea paginii, aplicația verifică existența unui token în localStorage, dacă utilizatorul nu este logat, este redirecționat automat către pagina de Login, dacă există token valid, se încarcă informațiile utilizatorului și istoricul conversațiilor. La inițializarea, aplicației se trimite o cerere GET către endpointul ConversationHistoryFilteredView pentru a recupera conversațiile anterioare, mesajele sunt reordonate și afișate ca perechi între utilizator și bot. La fiecare mesaj nou, fereastra de chat derulează automat către ultimul mesaj. Aceasta pagină conține un buton de Home in partea din stânga sus, și un buton în dreapta sus a paginii care la apăsarea lui se deschide un buton dropdown unde un utilizator normal are doar un buton de deconectare, dar dacă utilizatorul este un administrator îi mai apare un buton de setări unde la apăsarea lui administratorul este redirecționat către pagina de Settings.

Pagina Settings este o pagină destinată doar administratilor, această pagină oferă două funcționalități esențiale. Prima este importarea utilizatorilor printr-un fișier Excel, administratorul poate încărca un fișier Excel și este trimis către endpointul ImportUsersFromExcelView. Backendul procesează fișierul și adaugă automat utilizatorii în baza de date, și oferă feedback prin alerte de succes sau eroare în funcție de răspunsul primit de la server. A doua este actualizarea documentelor asistentului virtual, administratorul trimite o cerere către endpointul DownloadFilesView, această acțiune forțează backendul să încarce toate documentele esențiale folosite de asistentul virtual, ca răspunsurile generate de chatbot să fie actualizate, corecte și corespunzătoare întrebărilor utilizatorilor.

5. Manual de utilizare

5.1. Cerințe de sistem

Cerințe minime:

- Placă video: NVIDIA RTX 3050 Ti cu 4 GB VRAM
- Procesor: AMD Ryzen 7 6800H
- Memorie RAM: 16 GB DDR5
- Sistem de operare: Windows 10 sau mai recent

Cerințe recomandate:

- Placă video: NVIDIA RTX 4070 cu 8 GB VRAM
- Procesor: AMD Ryzen 7 7745HX
- Memorie RAM: 32 GB DDR5
- Sistem de operare: Windows 10 sau mai recent

5.2. Instalare

Clonează repository-ul folosind comanda:

git clone https://github.com/Xhadrines/Alex-VirtualAssistant.git

5.2.1. Configurare Back-End

Pașii pentru configurarea back-end:

1. Pentru a rula aplicația, trebuie să ai instalat:

- Python (v3.13.2 sau o versiune mai mare)
- pip (v24.3.1 sau o versiune mai mare)

2. Asigură-te că ai instalat gtk3-runtime de pe https://github.com/tschoonj/GTK-for-Windows-Runtime-Environment-Installer/releases
3. Asigură-te că te afli în directorul backend
4. Creează mediul virtual folosind comanda: *python -m venv .venv*
5. Activează mediul virtual folosind comanda: *.\.venv\Scripts\activate*
6. Instalează dependențele folosind comanda: *pip install -r requirements.txt*
7. Crează migrațiile folosind comanda: *python manage.py makemigrations*
8. Aplică migrațiile folosind comanda: *python manage.py migrate*
9. Instalează Ollama mergând pe site-ul oficial Ollama.
10. Instalează modelul Llama 3.2 folosind comanda: *ollama pull llama3.2*

11. Rulează proiectul folosind comanda: *python manage.py runserver*
12. Proiectul rulează pe: *http://127.0.0.1:2307/*

5.2.2. Configurare Front-End

Pașii pentru configurare front-end:

1. Pentru a rula aplicația, trebuie să ai instalat:
- Node.js (v22.13.1 sau o versiune mai mare)
- npm (v10.9.2 sau o versiune mai mare)
2. Asigură-te că te afli în directorul frontend
3. Creează un fișier .env și adaugă următoarea linie:

VITE_CHAT_API=http://<IP_BACKEND>:<PORT_BACKEND>

Note: Înlocuiește <IP_BACKEND> și <PORT_BACKEND> cu valorile corespunzătoare pe care le obții de la backend.

4. Instalează dependențele folosind comanda: *npm install*
5. Rulează proiectul folosind comanda: *npm run dev*
6. Proiectul rulează pe: *http://127.0.0.1:2002/*

5.3. Utilizare

Pentru a folosi aplicația trebuie sa deschizi pe un browser si să introdu-ți in bara de adrese URL-ul: „http://127.0.0.1:2002/”. Acolo va fi afișata o copie a site-ului oficial FIESC.

În partea din dreapta jos a paginii este un buton cu numele „Alex - Asistent Virtual” care va deschide o fereastra de chat simplă, unde poți pune întrebări pentru a cere informații de exemplu, „ce acte am nevoie pentru bursa socială ocazională?”, „ce burse sunt disponibile?”, „id-ul meu de student este [id_student] poți să îmi spui în ce grupă sunt?”, si multe altele.

Înainte de a adresa întrebări este necesară autentificarea cu un contul de administrator. Ca să te poți face acest lucru trebuie să apeși pe butonul din stânga sus al ferestrei de chat (are un simbol de +), după vei fi redirecționat către pagina de autentificare, acolo trebuie să te conectezi cu un cont de administrator (contul prestabilit de administrator este: nume „admin”, și parola „Admin#1”), după autentificare vei fi redirecționat către o pagina de chat. Acolo în partea din dreapta sus ai un buton (are un simbol de omuleț) pe care dacă îl apeși se deschide un meniu drop-down cu 2 opțiuni, una de deconectare pentru a ieși din cont, și una de setări. Accesează secțiunea setări unde vei fi redirecționat către pagina de setări, acolo vei avea un loc unde poți introduce un fișier exel pentru a introduce noi utilizatori, și un buton numit

„Actualizare” pe care trebuie apăsat înainte de a fi folosită aplicația pentru a descarcă toate documentele necesare cu informații de pe site-ul oficial FIESC unde va trebui să aștepte aproximativ 5 minute (sau mai puțin în funții de viteza de internet), după ce toate documentele au fost descărcate aplicația poate fi folosita de către toți utilizatorii.

Ca să primești un răspuns corect este necesar să formulezi întrebările clar și specific, de preferat de scris fără diacritice, pentru că întrebarea care este adresată sunt luate cuvintele cheie si ignorate cuvintele comune, pe baza acelor cuvinte cheie le compară cu numele documentelor pentru a extrage informațiile necesare ca să genereze un răspuns corespunzător.

Pentru a putea să iți faci un cont trebuie să te afli pe pagina de autentificare, acolo sub butonul de conectare ai un buton numit „Nu ai cont? Creează unul!” pe care dacă îl apeși vei fi redirecționat către pagina de creare a unui cont. Acolo pentru a putea crea unul trebuie să folosești un email de student care a fost introdus în baza de date de către administrator. Dacă acea adresă de email exista vei trece la următorul pas unde îți va cere să introduci un nume de utilizator si o parola (parola trebuie să fie de minim 5 caractere, să conțină o litera mare, un caracter special și un număr), o data ce contul a fost creat vei fi redirecționat către pagina de autentificare.

Avantajele unui cont de utilizator sunt următoarele:

1. Istoricul conversației este salvat asta înseamnă că face discuția cu „Alex” mai ușor pentru că el știe despre ce sa vorbit anterior și utilizatorul nu mai trebuie sa repete sau să spună informații personale pentru că acestea se află deja în baza de date.
2. Utilizatorul are acces la 2 comenzi:
   - „comenzi” – această comandă va returna toate comenzile pe care utilizatorul le poate folosi;
   - „generează o adeverință cu motivul [motiv]” – aceasta comandă va genera un fișier PDF care va conține o adeverință completată automat cu date din baza de date specifice utilizatorului care a folosit comanda pe baza unui șablon, și va returna un link de unde se poate descarcă fișierul.

6. Rezultate experimentale

Aici sunt prezentate rezultatele obținute în urma testării asistentului virtual bazat pe LLaMA 3.2 și RAG. Scopul testării este evaluarea capacității asistentului de a răspunde corect la întrebări pe baza documentelor încărcate, precum și verificarea funcționalității de generare a adeverințelor.

6.1. Testarea răspunsurilor la întrebări

6.1.1. Evaluarea răspunsurilor în funcție de numărul de fișiere disponibile

Aici analizată influența numărului de fișiere încărcate în sistem asupra acurateței răspunsurilor generate de asistentul virtual. Scopul este de a identifica pragul optim de documente care asigură un răspuns corect și relevant la o întrebare specifică. au fost păstrate inițial doar documentele care conțineau termeni comuni cu întrebarea, considerate relevante. Apoi, din acest set restrâns, s-a eliminat treptat câte un document, observându-se dacă și cum se degradează calitatea răspunsului.

Testul 1:

Număr fisiere: 5

Întrebare: „Ce documente am nevoie pentru bursa sociala ocazionala?”

Mesaj primit:

„Pentru a solicita Bursă Socială Ocazională, studenții trebuie să depună următoarele documente justificative: 1. Cererea studentului (Anexa 8) 2. Declarație pe propria răspundere privind veniturile nete (Anexa 8) 3. Acordul olograf al studentului privind prelucrarea datelor cu caracter personal (Anexa 8) 4. Documentele de identitate ale îngrijitorilor, dacă studenții sunt în stârpăciuni. 5. Proba de studiu, dacă studenții sunt în stârpăciuni. De asemenea, studenții care solicitează Bursă Specială "Prestigiu și Performanță" trebuie să prezinte documente suplimentare: * Certificatul de calificarea profesională sau universitar * Actele de absolvare ale cursurilor efectuate în străinătate * Documente care atestă prezența la concursuri, concursuri internaționale de sport și concursuri școlare, concursuri școlare etc. * Certificatul de absolvare al cursurilor efectuate în străinătate În funcție de caz, studenții pot trebui să prezinte alte documente sau să furnizeze informații suplimentare.”

Mesaj așteptat:

- cererea studentului (Anexa 8);
- declarație pe propria răspundere privind veniturile nete (Anexa 8);

- acordul olograf al studentului privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MÂNĂ;
- acordul olograf al fiecărui membru al familiei care realizează venituri privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MÂNĂ;
- declaraţia studentului privind autenticitatea documentelor depuse (Anexa 9);
- ancheta socială, în cazul în care nimeni din familia studentului, incluzându-l și pe acesta, nu realizează venituri sau cel puțin un părinte al studentului lucrează sau domiciliază în străinătate;
- copii de pe certificatul/certificatele de deces al/ale părintelui/părinților, de pe acte de stare civilă/decizia instanței de menținere a stării de arest/raportul de anchetă socială în cazul părinților dispăruți, dacă este cazul.

Concluzie: Răspunsul primit conține o parte din documentele esențiale cerute pentru bursa socială ocazională, precum cererea studentului și declarația pe propria răspundere (Anexa 8), dar prezintă o eroare importantă referitoare la anexele pentru acordurile olografe, menționând Anexa 8 în loc de Anexa 5 și lipsind complet acordurile membrilor familiei și declarația privind autenticitatea documentelor (Anexa 9). De asemenea, nu sunt menționate ancheta socială și documentele suplimentare legate de situații speciale. Astfel, răspunsul este parțial corect, dar incomplet și cu unele erori de referință.

Testul 2:

Număr fisiere: 3

Întrebare: „Ce documente am nevoie pentru bursa sociala ocazionala?”

Mesaj primit:

„Pentru a primi bursa socială ocazională, necesități următoarele documente justificative: 1. Cererea studentului (Anexa 8) 2. Declarație pe propria răspundere privind veniturile nete (Anexa 8) 3. Acordul olograf al studentului privind prelucrarea datelor cu caracter personal (Anexa 5) - SCRIS INTEGRAL DE MÂNĂ 4. Acordul olograf al fiecărui membru al familiei care realizează venituri privind prelucrarea datelor cu caracter personal (Anexa 5) - SCRIS INTEGRAL DE MÂNĂ 5. Declarația studentului privind autenticitatea documentelor depuse (Anexa 9) 6. Ancheta socială, în cazul în care nimeni din familia studentului, incluzându-l și pe acesta, nu realizează venituri sau cel puțin un părinte al studentului lucrează sau domiciliază în străinătate Documentele acestor anexe sunt importante pentru a demonstra calitatea cererii dumneavoastră și ca fiind eligibil pentru acordarea burselor sociale.”

Mesaj așteptat:

- cererea studentului (Anexa 8);
- declarație pe propria răspundere privind veniturile nete (Anexa 8);
- acordul olograf al studentului privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MÂNĂ;
- acordul olograf al fiecărui membru al familiei care realizează venituri privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MÂNĂ;
- declaraţia studentului privind autenticitatea documentelor depuse (Anexa 9);
- ancheta socială, în cazul în care nimeni din familia studentului, incluzându-l și pe acesta, nu realizează venituri sau cel puțin un părinte al studentului lucrează sau domiciliază în străinătate;
- copii de pe certificatul/certificatele de deces al/ale părintelui/părinților, de pe acte de stare civilă/decizia instanței de menținere a stării de arest/raportul de anchetă socială în cazul părinților dispăruți, dacă este cazul.

Concluzie: Răspunsul este aproape complet și corect, incluzând toate documentele principale solicitate (Anexele 5, 8, 9 și ancheta socială). Singura lipsă este menționarea explicită a copiilor documentelor pentru cazurile speciale (deces, stare civilă, etc.).

Testul 3:

Număr fisiere: 2

Întrebare: „Ce documente am nevoie pentru bursa sociala ocazionala?”

Mesaj primit:

„Pentru a urmări cererea de bursă socială ocazională, ai nevoie următoarele documente: 1. Cererea studentului (Anexa 8) 2. Declarație pe propria răspundere privind veniturile nete (Anexa 8) 3. Acordul olograf al studentului privind prelucrarea datelor cu caracter personal (Anexa 5) - SCRIS INTEGRAL DE MÂNĂ 4. Acordul olograf al fiecărui membru al familiei care realizează venituri privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MÂNĂ 5. Declarația studentului privind autenticitatea documentelor depuse (Anexa 9) 6. Ancheta socială, în cazul în care nimeni din familia studentului, incluzându-l și pe acesta, nu realizează venituri sau cel puțin un părinte al studentului lucrează sau domiciliază în străinătate 7. Copii de pe certificatul/certificatele de deces al/ale părintelui/părinților, de pe acte de stare civilă/decizia instanței de menținere a stării de arest/raportul de anchetă socială în cazul părinților dispăruți, dacă este cazul.”

Mesaj așteptat:

1. cererea studentului (Anexa 8);

2. declarație pe propria răspundere privind veniturile nete (Anexa 8);
3. acordul olograf al studentului privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MÂNĂ;
4. acordul olograf al fiecărui membru al familiei care realizează venituri privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MÂNĂ;
5. declaraţia studentului privind autenticitatea documentelor depuse (Anexa 9);
6. ancheta socială, în cazul în care nimeni din familia studentului, incluzându-l și pe acesta, nu realizează venituri sau cel puțin un părinte al studentului lucrează sau domiciliază în străinătate;
7. copii de pe certificatul/certificatele de deces al/ale părintelui/părinților, de pe acte de stare civilă/decizia instanței de menținere a stării de arest/raportul de anchetă socială în cazul părinților dispăruți, dacă este cazul.

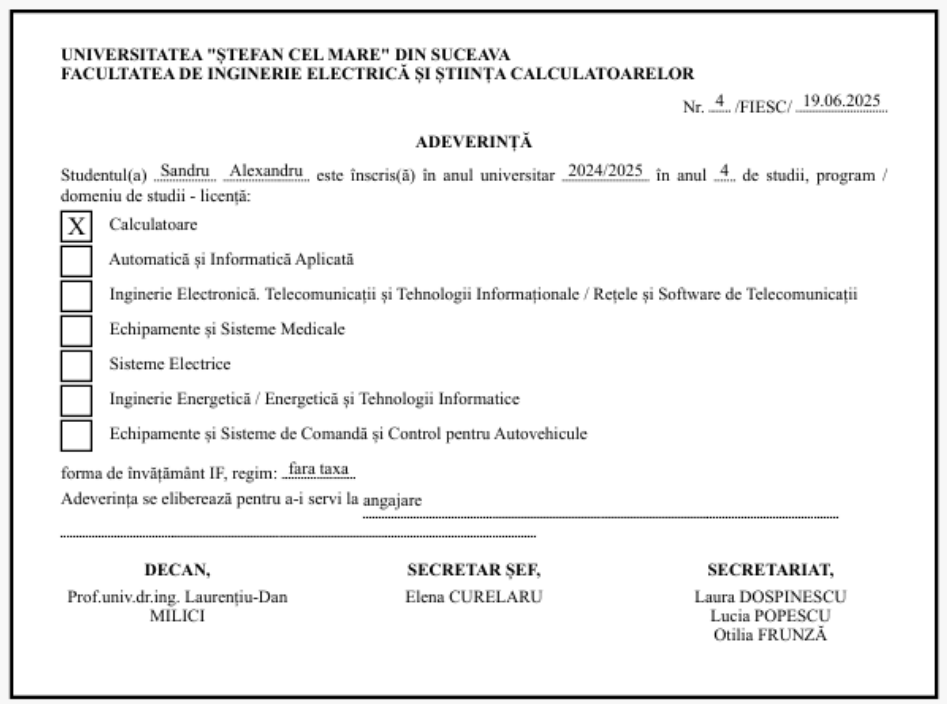
Concluzie: Răspunsul primit este complet și corespunde integral mesajului așteptat, incluzând toate documentele necesare pentru bursa socială ocazională, fără omisiuni sau erori. Acest răspuns este corect și complet.

6.2. Testarea comenzilor

Autentificat: Șandru Alexandru

Comandă: „Genereaza o adeverinta cu motivul angajare”

Mesaj primit:



UNIVERSITATEA "ŞTEFAN CEL MARE" DIN SUCEAVA
FACULTATEA DE INGINERIE ELECTRICĂ ŞI ŞTIINŢA CALCULATOARELOR

Nr. 4 /FIESC/ 19.06.2025

ADEVERINŢĂ

Studentul(a) Sandru Alexandru este înscris(ă) în anul universitar 2024/2025 în anul 4 de studii, program / domeniu de studii - licență:

- [X] Calculatoare
- [ ] Automatică și Informatică Aplicată
- [ ] Inginerie Electronică. Telecomunicații și Tehnologii Informaționale / Rețele și Software de Telecomunicații
- [ ] Echipamente și Sisteme Medicale
- [ ] Sisteme Electrice
- [ ] Inginerie Energetică / Energetică și Tehnologii Informatice
- [ ] Echipamente și Sisteme de Comandă și Control pentru Autovehicule

forma de învățământ IF, regim: fara taxa

Adeverința se eliberează pentru a-i servi la angajare

DECAN, Prof.univ.dr.ing. Laurențiu-Dan MILICI

SECRETAR ŞEF, Elena CURELARU

SECRETARIAT, Laura DOSPINESCU, Lucia POPESCU, Otilia FRUNZĂ

Concluzie: Testul a fost reușit cu succes, a fost generată o adeverința corect conform datelor din baza de date, demonstrând funcționalitatea adecvată a comenzii.

Autentificat: NONE

Comandă: „Genereaza o adeverinta cu motivul angajare”

Mesaj primit: „Doar persoanele autentificate pot genera o adeverinta!”

Concluzie: Testul a fost reușit cu succes, a fost generată o adeverința corect conform datelor din baza de date, demonstrând funcționalitatea adecvată a comenzii.

7. Concluzii și direcții viitoare de dezvoltare

7.1. Stadiul actual al proiectului

Baza de date:

- Complet, posibil câteva retușuri.

Back-End:

- Trebuie optimizat sistemul de RAG. (dacă sunt puține fișiere generează un răspuns corect, dacă sunt prea multe fișiere generează un răspuns parțial corect)
- De făcut retușuri în cod.

Front-End:

- Complet, posibil câteva retușuri.

7.2. Direcții viitoare de dezvoltare

Îmbunătățirea sistemul de RAG:

- Introducerea unei etape avansate de filtrare a documentelor.
- Limitarea documentelor trimise la model, preia doar documentele care au relevantă întrebării nu toate care conțin in numele lor cuvinte asemănătoare cu întrebarea.
- Limitarea conținutului extras din documente, preia doar conținutul care are nevoie nu întregul document.

Adăugarea unei funcționalități de feedback:

- Utilizatorii pot marca răspunsurile drept „util” sau „neutile”, ca să fie colectate pentru a antrena sau ajusta modelul pe baza feedback-ului real.

Extinderea interfeței administrative:

- Posibilitatea de a șterge, dezactiva, și actualiza utilizatori direct din interfață, nu doar adăugarea/actualizarea datelor printr-un fișier Excel.
- Adăugarea unui panou de statistici pentru, utilizatori, întrebări frecvente, erori, etc.

Schimbarea LLM-ului:

- Înlocuirea modelului local cu un LLM mai performant.

Autentificare mai sigura:

- Implementare autentificării în doi pași (2FA) pentru conturi de administrator.

Testare automată:

- Introducerea testelor pentru backend.

Bibliografie

https://ase.ro/

https://www.djangoproject.com/

https://www.drawio.com/

https://www.geeksforgeeks.org/large-language-model-llm/

https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/

https://newsbucuresti.ro/ase-bucuresti-prima-universitate-din-romania-care-introduce-un-asistent-ai-pentru-studenti/

https://ollama.com/

https://react.dev/

https://sqlite.org/index.html

https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2/

Anexe 1 – Clasa ConversationChat (endpoint principal)

Această anexă conține codul complet al clasei „ConversationChat”, care gestionează cererile POST. Clasa verifică mesajul primit diferențiindu-l între întrebări și comenzi, integrează funcționalitatea de generare a adeverințelor pentru utilizatorii autentificați și interacționează cu clasa RAG pentru a oferi un răspuns relevant.

```
class ConversationChat(APIView):
    permission_classes = [AllowAny]
    rag = RAG()

    def post(self, request):
        try:
            question = request.data.get('message')
            if not question:
                return Response(
                    {"error": "Mesajul nu poate fi gol."},
                    status=status.HTTP_400_BAD_REQUEST
                )

            if question.strip().lower() == "comenzi":
                answer_text = (
                    "Comenzile disponibile sunt:\n"
                    "- Genereaza o adeverinta cu motivul [motiv]\n"
                    "  Exemplu: Genereaza o adeverinta cu motivul angajare\n"
                )

                if request.user and request.user.is_authenticated:
                    ConversationHistory.objects.create(
                        user=request.user,
                        question=question,
                        answer=answer_text
                    )

                return Response(
                    {"answer": answer_text},
                    status=status.HTTP_200_OK
                )

            if (
                request.user and request.user.is_authenticated and
                question.lower().startswith("genereaza o adeverinta cu motivul")
            ):
                user = request.user
```

```
trigger = "genereaza o adeverinta cu motivul"
motiv = question[len(trigger):].strip()

if not motiv:
    return Response(
        {"answer": "Te rog specifică motivul pentru adeverință."},
        status=status.HTTP_200_OK
    )

try:
    profile = UserProfile.objects.get(user=user)
except UserProfile.DoesNotExist:
    return Response(
        {"error": "Profilul utilizatorului nu a fost găsit."},
        status=status.HTTP_404_NOT_FOUND
    )

numar = Adevetinta.objects.count() + 1
data_emitere = timezone.now().date()

adeverinta = Adevetinta.objects.create(
    user=user,
    last_name=user.last_name,
    first_name=user.first_name,
    an_universitar=profile.an_universitar,
    an_studiu=profile.an_studiu,
    specializare=profile.specializare.nume,
    tip_taxa=profile.tip_taxa,
    motiv=motiv,
    numar=numar,
    expires_at=timezone.now() + relativedelta(years=1)
)

motiv = adeverinta.motiv or ""
max_len = 70
split_pos = motiv.rfind(" ", 0, max_len)
if split_pos == -1:
    split_pos = max_len
motiv1 = motiv[:split_pos]
motiv2 = motiv[split_pos:].strip()

context = {
    "numar": adeverinta.numar,
    "data": data_emitere.strftime("%d.%m.%Y"),
    "nume": adeverinta.last_name,
```

```
    "prenume": adeverinta.first_name,
    "anUniversitar": adeverinta.an_universitar,
    "anul": adeverinta.an_studiu,
    "specializare": adeverinta.specializare,
    "regim": adeverinta.tip_taxa,
    "motiv1": motiv1,
    "motiv2": motiv2,
}

html_string = render_to_string(
    'adeverinta/adeverinta.html',
    context
)

pdf_bytes = HTML(
    string=html_string,
    base_url=request.build_absolute_uri('/')
).write_pdf()

unique_id = str(uuid.uuid4())
filename = f"adeverinta_{unique_id}.pdf"
pdf_path = os.path.join(settings.MEDIA_ROOT, filename)

if not os.path.exists(settings.MEDIA_ROOT):
    os.makedirs(settings.MEDIA_ROOT)

with open(pdf_path, "wb") as f:
    f.write(pdf_bytes)

pdf_url = request.build_absolute_uri(
    settings.MEDIA_URL + filename
)

answer_text = (
    f"Adeverinta a fost creata si e valabila un an de zile: {pdf_url}"
)

if request.user and request.user.is_authenticated:
    ConversationHistory.objects.create(
        user=request.user,
        question=question,
        answer=answer_text
    )

return Response(
```

```
                {"answer": answer_text},
                status=status.HTTP_200_OK
            )

        answer = self.rag.get_response(question)

        if request.user and request.user.is_authenticated:
            logger.info(f"User {request.user.username} is authenticated.")
            ConversationHistory.objects.create(
                user=request.user,
                question=question,
                answer=answer
            )

        return Response(
            {"answer": answer},
            status=status.HTTP_200_OK
        )

    except Exception as e:
        return Response(
            {"error": str(e)},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )
```

Anexa 2 – Clasa RAG (Retrieval-Augmented Generation)

Această anexă conține codul complet al clasei RAG, responsabilă pentru căutarea fișierelor PDF relevante în funcție de întrebare, extragerea conținutului acestora și generarea unui context care este transmis către modelul LLaMA 3.2 pentru formularea unui răspuns.

```
class RAG:
    def __init__(self, pdf_directory='pdfs'):
        self.pdf_directory = pdf_directory
        self.context = []

    def extract_words_from_filename(self, filename):
        words = re.sub(r'[-_]', ' ', filename)
        words = words.replace('.pdf', '').split()
        return words

    def read_pdf_content(self, pdf_path):
        try:
            reader = PdfReader(pdf_path)
            content = ''
            for page in reader.pages:
                content += page.extract_text()
            return content
        except Exception as e:
            return ''

    def find_matching_files(self, question):
        matching_files = []
        question_words = re.sub(r'[^\w\s]', '', question).lower().split()

        for word in question_words:
            for filename in os.listdir(self.pdf_directory):
                if filename.endswith('.pdf'):
                    file_words = re.sub(r'[^\w\s]', '', '
'.join(self.extract_words_from_filename(filename))).lower().split()
                    if word in file_words:
                        if filename not in matching_files:
                            matching_files.append(filename)

        return matching_files

    def get_context(self, question):
        self.context = []
        matching_files = self.find_matching_files(question)
```

```
        for filename in matching_files:
            pdf_path = os.path.join(self.pdf_directory, filename)
            content = self.read_pdf_content(pdf_path)
            if content:
                self.context.append(content)

        return self.context

    def get_response(self, question):
        context = self.get_context(question)

        messages = [
            {
                "role": "system",
                "content": "Te rog sa raspunzi in limba romana, pe tine te cheama
'Alex' si esti asistentul virtual pentru 'Facultatea de Inginerie Electrica si
Stiinta Calculatoarelor'."
            }
        ]

        if context:
            messages.append(
                {
                    "role": "system",
                    "content": "Analizeaza urmatorul context si genereaza un
raspuns detaliat si relevant: " + " ".join(context)
                },
            )

        messages.append(
            {
                "role": "user",
                "content": question
            }
        )

        try:
            response = ollama.chat(model="llama3.2", messages=messages)
            return response['message']['content']
        except Exception as e:
            return f"Eroare la generarea raspunsului: {str(e)}"
```