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, accesarea datelor personale din baza de date a utilizatorului și adresarea întrebărilor într-un mod mai simplu, fără a mai fi necesară introducerea constantă a datelor personale sau a altor informații deja menționate. 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 își pot găsi răspunsuri în documentele 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

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, Academia de Studii Economice din București a introdus „Saro”, un asistentul virtual bazat pe inteligență artificială care oferă studenților posibilitatea de a obține informații academice și administrative [1]. 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 educația națională, chiar dacă utilizarea asistenților virtuali în România este încă la început. Aceste proiecte existente au scopul îmbunătățirea accesului la informații și să minimizeze timpului pierdut de studenți în procesele administrative.

1.4. Situația actuală pe plan internațional

La nivel internațional, implementarea asistenților virtuali în mediul educațional este mult mai avansată și cu un impact semnificativ asupra modului în care se desfășoară procesul de învățământ.

Ca exemplu, se poate aminti Universitatea din Toronto – Rotman School of Management, unde profesorii Joshua Gans și Kevin Bryan au dezvoltat „All Day TA”, un asistent virtual bazat pe inteligență artificială, lansat în 2025. Acesta oferă studenților suport personalizat 24/7, bazat exclusiv pe materiale aprobate de profesori, facilitând accesul imediat

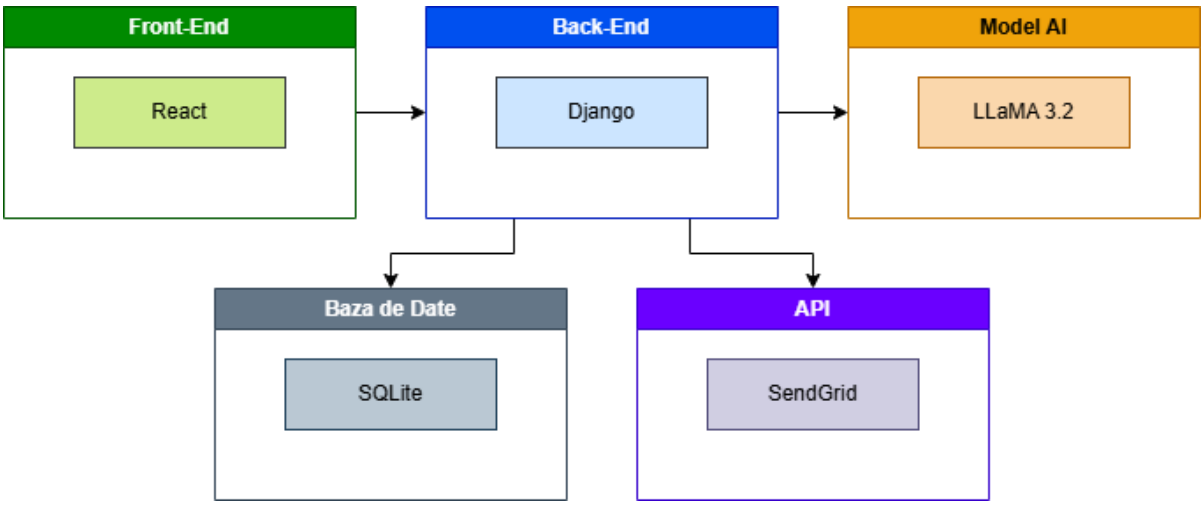
la răspunsuri corecte și relevante. Inițial implementat într-un curs pilot cu 300 de studenți, „All Day TA” a gestionat peste 12.000 de întrebări pe parcursul unui semestru, iar în prezent este folosit în aproximativ 100 de universități și școli de afaceri din întreaga lume [2]. Platforma are un cost redus de aproximativ 2 dolari per student și permite profesorilor să-și încarce propriile materiale de curs, oferind astfel un răspuns adaptat și precis.

Această inițiativă ilustrează direcția integrării inteligenței artificiale în educația internațională, unde asistenții virtuali, sistemele automate de feedback și instrumentele AI devin tot mai frecvent utilizați. Aceste proiecte existente au ca scop nu doar facilitarea procesului educațional, și creșterea calității învățării eficientizând timpului petrecut de studenți în procesele academice.

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 acestor tehnologii a fost făcută astfel încât să permită integrarea eficientă a unui model de inteligență artificială, dar și o experiență plăcută și intuitivă pentru utilizator.

În Figura 2.1 este prezentată diagrama arhitecturii sistemului în cadrul aplicației, care descrie modul în care sunt organizate componentele și fluxurile logice ale sistemului.



Figură 2.1 – Diagrama arhitecturii sistemului

2.1. Back-End

Pentru partea de back-end am ales limbajul de programare Python, împreuna cu framework-ul Django având în vedere avantajele oferite, dintre care se pot aminti doar câteva:

- Python este un limbaj flexibil, cu o documentația bine detaliată [3], foarte popular în dezvoltarea aplicațiilor web [4] și în domeniul inteligenței artificiale, care îl face o alegere potrivită pentru un sistem care integrează un model AI [5];
- Django, este un framework scris în limbajul de programare Python, conceput pentru dezvoltarea rapidă a aplicațiilor web [6], care implică gestionarea utilizatorilor, accesul la bază de date și procesare de date sensibile [7], conține o documentația bine detaliată [8] și suportul pentru API-uri REST prin intermediul Django REST Framework [9], permite astfel o comunicare eficientă între interfață și server.

Pentru partea de generare a răspunsurilor, am ales modelul LLaMA 3.2 (Large Language Model Meta AI 3.2), un model LLM (Large Language Model) [10] dezvoltat de Meta AI, pentru următoarele motive:

- Este un model gratuit și open-source [11];
- Rulează local, ai nevoie doar de un calculator performant [12];
- Este ușor de integrat și de folosit în aplicații [13].

2.1.1. Django

Pentru partea de back-end, am folosit Django, un framework care îndeplinește mai multe roluri esențiale:

- Expune un API REST prin care interfața din browser comunică cu serverul;
- Gestionează autentificarea utilizatorilor și drepturile de acces;
- Trimite emailuri pentru crearea conturilor sau resetarea parolei folosind un API de la SendGrid [14];
- 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 printr-un sistem de tip RAG (Retrieval-Augmented Generation) [15]. Procesul funcționează astfel:

- Când utilizatorul trimite o instrucțiune, aceasta este salvată și analizată;
- Se extrag cuvintele;
- Aceste cuvinte sunt folosite pentru a căuta documente relevante care conțin informațiile necesare;
- Documentele găsite sunt ordonate în funcție de numărul de cuvinte aflate în denumirea lor, dacă conțin mai multe cuvinte asemănătoare sunt poziționate primele;
- Documentele sunt parcurse în ordine, trimițând conținutul lor pe rând drept context și întrebarea către LLaMA 3.2 pentru a genera un răspuns;
- Răspunsul generat este validat astfel:
    - Dacă răspunsul generat este unul corespunzător și complet, acesta este returnat utilizatorului și procesul se oprește;

- Dacă răspunsul generat este unul greșit sau incomplet fișierul este ignorat și sistemul trece la următorul document;
- Dacă niciun fișier nu oferă informații suficiente, utilizatorul primește un mesaj care menționează că nu s-au găsit informații relevante pentru răspuns.

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

Pentru partea de front-end am ales biblioteca React împreună cu Vite pentru următoarele motive:

- React este o biblioteca JavaScript, cu o documentația bine detaliată, foarte populară în dezvoltarea aplicațiilor web, ea este bazată pe componente, ceea ce permite o organizare clară a codului și reutilizarea ușoara a lor fiind potrivită pentru aplicații web interactive [16];
- Vite este un instrument de build și servește la dezvoltarea aplicații web, este rapidă și simplă de utilizat, oferă o experiența ușoara în dezvoltare deoarece ea actualizează instant componentele [17].

2.2.1. React + Vite

Pentru partea de front-end, aplicația este dezvoltată utilizând biblioteca React, împreună cu Vite care are mai multe roluri esențiale;

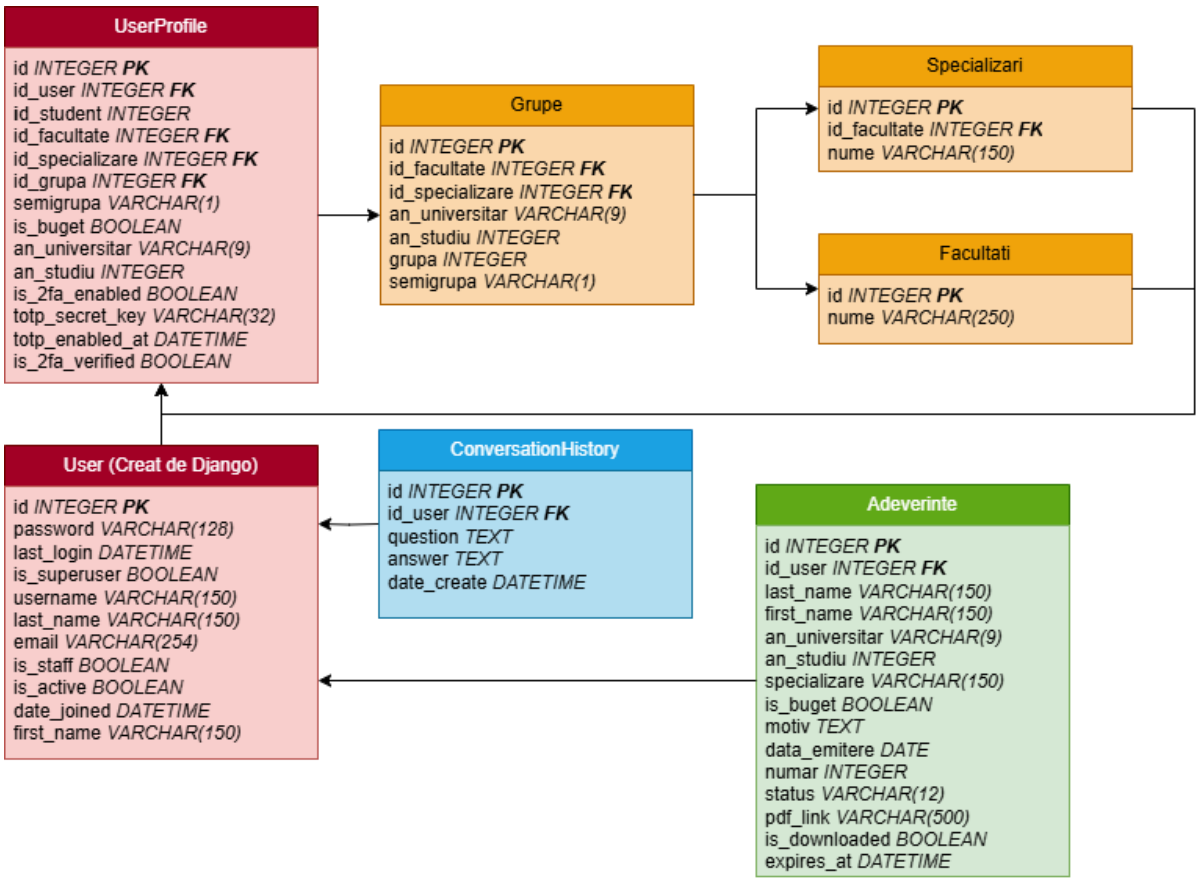
- Generează și afișează interfața grafică pe baza componentelor;
- Comunică cu serverul prin apeluri HTTP către API-ul RESET oferite de Django;
- Reacționează în timp real cu datele primite de la server, fără a fi nevoie de reîncărcarea paginii;
- Asigură o experiență ușoară și intuitivă pentru utilizator, prin tranziții rapide și timp redus de răspuns, datorită optimizărilor aduse de Vite.

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. Astfel, administratorii au posibilitatea de a adăuga utilizatori noi în baza de date folosind un fișier Excel care structura lui trebuie să fie următoarea:

- Rândul 1, coloana A: universitatea;
- Rândul 2, coloana A:facultatea;
- Rândul 3, coloana A: domeniul;
- Rândul 4, coloana A: programul de studii;
- Rândul 5, coloana A: anul universitar;
- Rândul 6, coloana A: anul de studii;
- Rândul 7, coloana A: trebuie lăsată liberă;
- Rândul 8 capul de tabel, cu următoarele coloane:
    - Coloana A: Nr. crt.;
    - Coloana B: ID student;
    - Coloana C: Numele;
    - Coloana D: Prenumele;
    - Coloana E: Email;
    - Coloana F: Identificatorul de semigrupă (de exemplu 3142B, unde 3 reprezintă facultatea, 1 specializarea, 4 anul, 2 grupa, B semigrupa);
    - Coloana G: Forma de finanțare (cu taxă/fără taxă);
- Începând cu linia 9, sunt introduși toți utilizatorii, în ordinea coloanelor menționate. Câte un utilizator pe rând.

De asemenea, 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 și accesarea datelor din baza de date 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 sau să ofere date personale disponibile deja în baza de date;
- Utilizatorii care își doresc să își creeze un cont trebuie sa folosească un email de student valid aflat în baza de date, deoarece aceștia vor primi un email cu un link către pagina unde își pot crea un cont;
- Utilizatorii care își doresc să își schimbe parola contului din anumite motive au posibilitatea să o facă folosindu-și email-ul asociat contului, prin trimiterea unei cereri de resetare a parolei.

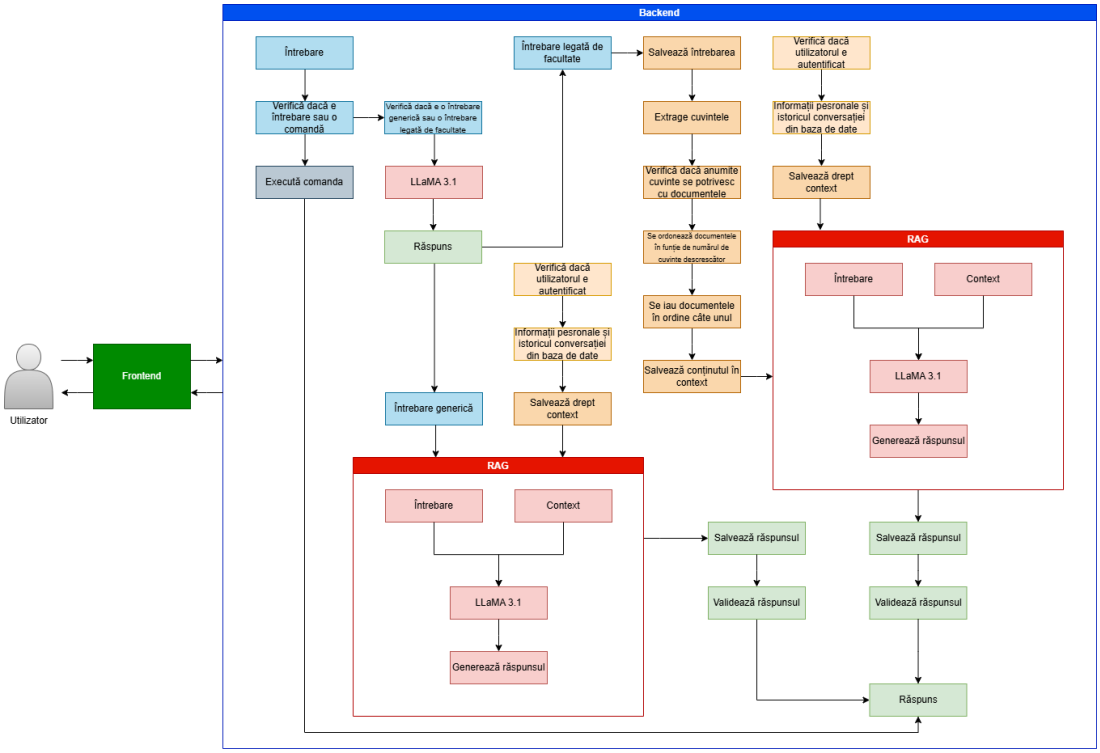
3.2.3. Procesare întrebărilor și comenzilor

Când utilizatorul interacționează cu sistemul prin interfața de tip chat se realizează o serie de operații ce sunt prezentate, pe scurt, în continuare:

- Instrucțiunea utilizatorului este trimisă de la front-end către back-end;
- Back-end verifică dacă este întrebare sau comandă;
- Dacă instrucțiunea este o întrebare:
  - Întrebarea se salvează;
  - Este trimisă la LLaMA pentru a verifica dacă e o întrebare generală sau o întrebare legată de facultate;
  - Dacă este o întrebare generală este transmisă direct la LLaMA care returnează un răspuns corespunzător întrebării;
  - Dacă este o întrebare legată de facultate aceasta este procesată astfel:
    - Întrebarea se salvează;
    - Cuvintele din întrebare, se verifică dacă se potrivesc cu un cuvânt aflat în numele unui document;

        - Documentele găsite sunt ordonate în funcție de numărul de cuvinte comune, dacă numărul de cuvinte comune este mai mare de cât documentul anteriorul sunt poziționate primele;
        - Documentele sunt parcurse în ordine, trimițând conținutul lor pe rând drept context și întrebarea către LLaMA pentru a genera un raspuns;
        - Răspunsul generat este validat astfel:
            - Dacă răspunsul generat este unul corespunzător și complet, acesta este returnat utilizatorului și procesul se oprește;
            - Dacă răspunsul generat este unul greșit sau incomplet fișierul este ignorat și sistemul trece la următorul document;
            - Dacă niciun fișier nu oferă informații suficiente, utilizatorul primește un mesaj care menționează că nu s-au găsit informații relevante pentru răspuns.
- Dacă instrucțiunea este o comandă:
    - Se verifică dacă utilizatorul este autentificat,
    - Apoi 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, împreună 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 aplicației

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

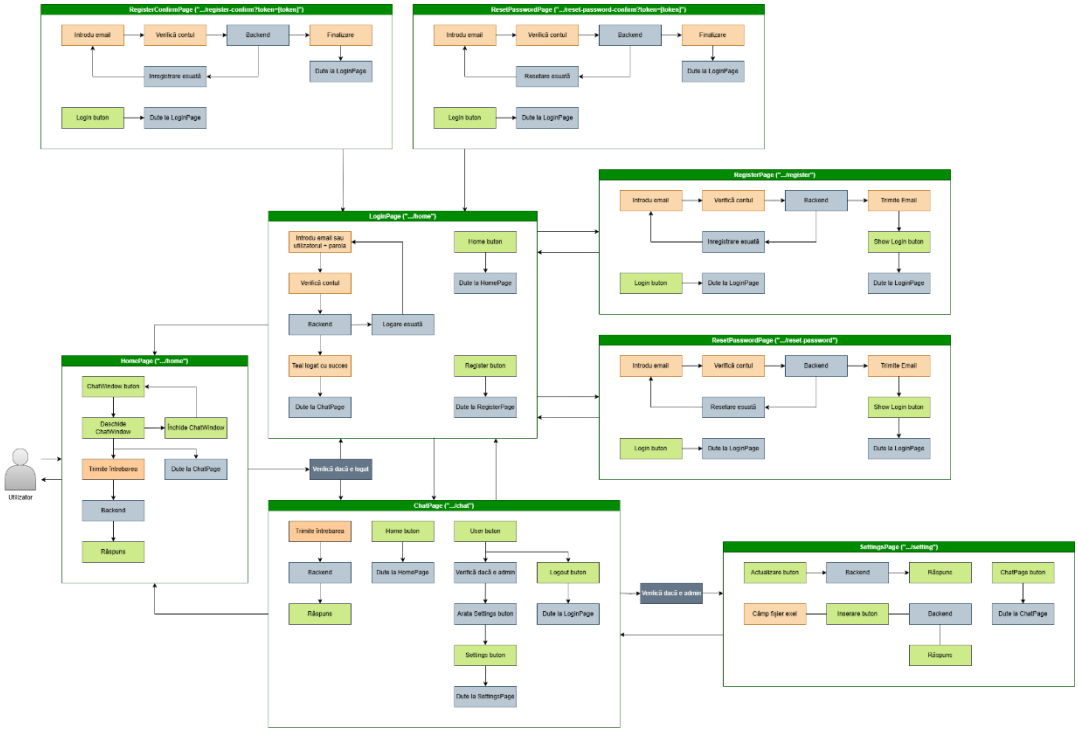
Componente:

- 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, în LoginPage se mai află un buton pentru a pleca înapoi spre pagina de pornire mai exact HomePage;
- RegisterPage – pagina de cerere pentru înregistrare, unde utilizatorii noi își pot crea un cont, cu condiția să aibă un email de student valid și existent în baza de date, aici există un buton pentru a pleca înapoi spre pagina de autentificare mai exact LoginPage;
- RegisterConfirmPage – pagina de finalizare a înregistrare, unde utilizatorii noi o pot accesa printr-un link primit printr-un mesaj pe email aici există un buton pentru a pleca înapoi spre pagina de autentificare mai exact LoginPage;
- ResetPasswordPage – pagina de cerere pentru resetarea parolei, unde utilizatorii își pot reseta parola din anumite motive, aici există un buton pentru a pleca înapoi spre pagina de autentificare mai exact LoginPage;
- ResetPasswordConfirmPage – pagina de finalizare pentru resetarea parolei, unde utilizatorii o pot accesa printr-un link primit printr-un mesaj pe email 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

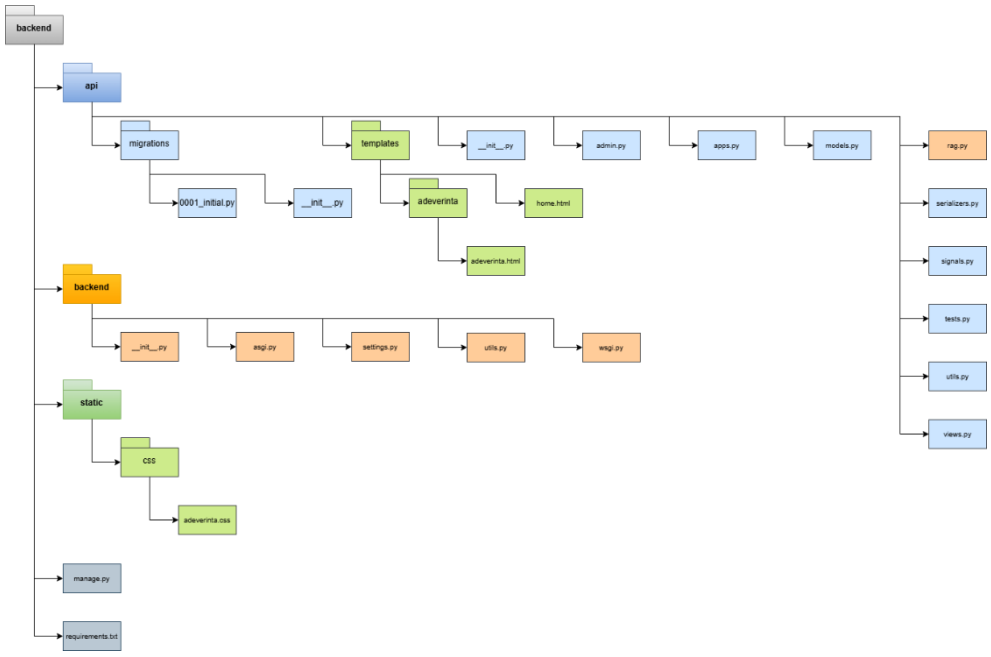
Componenta back-end a aplicației a fost realizată în limbajul de programare Python utilizând framework-ul Django, având rolul de a gestiona:

- Autentificarea, înregistrarea și schimbarea parolei utilizatorului;
- Stocarea și procesarea datelor în baza de date;
- Integrarea modelul LLaMA prin sistemul RAG;
- Funcționalitățile suplimentare disponibile utilizatorilor autentificați.

Funcționalitățile disponibile utilizatorilor autentificați sunt:

- Salvarea istoricului conversației;
- Afișarea comenzilor disponibile;
- Generarea automată a adeverințelor în format PDF pe baza datelor din baza de date corespunzătoare utilizatorului.

În Figura 4.1 este prezentată structura directorului back-end care conține fișiere si directoarele generate automat de Django, precum si cele adăugate manual 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 – conține logica principală de procesare a cererilor de la front-end;
- 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 – conține clasele care transformă datele din modelele Django în format JSON, pentru a fi transmite către front-end;
- signals.py – definește acțiunile care sunt executate automat de anumite evenimente;
- admin.py, apps.py, tests.py – fișiere standard Django pentru înregistrarea modelelor în panoul de administrare, configurarea aplicației si testare;
- urls.py – definește rutele aplicației, legând adresele acesteia de funcțiile din views.py.

Directorul „/api/migrations” este generat automat de Django.

- 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 adeverințelor în format PDF, care este completat automat cu datele utilizatorului.

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

- settings.py – fișierul de configurare principal, unde sunt definite aplicațiile instalate, conexiunea la baza de date, locațiile fișierelor statice, integrarea cu LLaMA și altele;
- urls.py – definește rutele globale ale aplicației;
- asgi.py, wsgi.py – fișiere standard pentru rularea aplicației.

Directorul „/static/css/” conține fișierele CSS folosite pentru stilul vizual a documentelor generate:

- 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 – lista cu toate pachetele Python necesare rulării aplicației, folosită pentru instalarea rapidă a dependențelor.

4.1.2. Funcționalități și logica de procesare

Componenta principală a aplicației este clasa „ConversationChat” (vezi Anexa 1 pentru codul complet al clasei) care definește 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 sistemul de RAG, dar și de a genera adeverinț în format PDF.

1. Procesare întrebărilor primite

Endpointul primește de la front-end un mesaj de la utilizator. În funcție de conținut se declanșează una dintre următoarele acțiuni:

- Dacă mesajul este „comenzi”, se returnează o listă cu toate comenzile disponibile, utile pentru ghidarea utilizatorului;
- Dacă mesajul începe cu „genereaza o adeverinta cu motivul”, aplicația declanșează procesul de creare a unei adeverințe în 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 funcționalități principale ale aplicației. Pentru a folosi aceasta funcționalitate utilizatorul trebuie să fie autentificat, după ce este recunoscută comanda de tipul „Genereaza o adeverinta cu motivul [motiv]” sunt parcurși pașii următori:

- Se extrage motivul solicitat de utilizator;
- Se încarcă profilul utilizatorului din baza de date;
- Se creează un obiect adeverința, care conține datele personale și motivul solicitat;
- Se generează un fișier HTML după șablonul predefinit „adeverinta.html”, în care sunt inserate datele corespunzătoare;

- Fișierul HTML este convertit într-un fișier PDF folosind biblioteca WeasyPrint, apoi este salvată local cu un nume unic;
- Se generează un link către fișierul PDF, care este returnat utilizatorului.

3. Generarea de răspunsuri personalizate folosind 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):

- Întrebarea se salvează;
- Este trimisă la LLaMA pentru a verifica dacă e o întrebare generală sau o întrebare legată de facultate;
- Dacă este o întrebare generală este transmisă direct la LLaMA care returnează un răspuns corespunzător întrebării;
- Dacă este o întrebare legată de facultate aceasta este procesată astfel:
    - Întrebarea se salvează;
    - Cuvintele din întrebare, se verifică dacă se potrivesc cu un cuvânt aflat în numele unui document;
    - Documentele găsite sunt ordonate în funcție de numărul de cuvinte comune, dacă numărul de cuvinte comune este mai mare de cât documentul anteriorul sunt poziționate primele;
    - Documentele sunt parcurse în ordine, trimițând conținutul lor pe rând drept context și întrebarea către LLaMA pentru a genera un raspuns;
    - Răspunsul generat este validat astfel:
        - Dacă răspunsul generat este unul corespunzător și complet, acesta este returnat utilizatorului și procesul se oprește;
        - Dacă răspunsul generat este unul greșit sau incomplet fișierul este ignorat și sistemul trece la următorul document;
        - Dacă niciun fișier nu oferă informații suficiente, utilizatorul primește un mesaj care menționează că nu s-au găsit informații relevante pentru răspuns.

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 a fi folosite la procesarea întrebărilor în sistemul de RAG. Ea asigură actualizarea constantă a documentelor utilizate pentru generarea răspunsurilor.

Procesul este declanșat printr-un POST la un endpoint protejat, și parcurge următorii pași:

- Se accesează adresa URL a facultății;
- Se analizează HTML-ul folosind biblioteca BeautifulSoup;
- Sunt extrase toate link-urile și se filtrează doar fișierele cu extensiile .pdf, .doc și .docx;
- Fișierele sunt descărcate local în directoarele pdfs sau docs, în funcție de extensie.

2. Importul de utilizatori dintr-un fișier Excel

Această funcționalitate permite încărcarea automată a datelor în baza de date pe baza unui fișier Excel, pentru a asigura o introduce rapidă și eficientă a datelor, pașii parcurși sunt următorii:

- Se primește un fișierul Excel prin POST;
- Se extrage anul universitar și se parcurg rândurile începând cu poziția 9 (conform formatului standard);
- Pentru fiecare student se extrag:
    - ID,
    - Nume,
    - Prenume,
    - Email,
    - Codul de semi-grupă,
    - Regimul de taxare;
- Codul de semi-grupă este procesat pentru a extrage:
    - ID-ul facultății,
    - ID-ul specializării,
    - Anul de studiu,
    - Grupa,
    - Semi-grupa;

- Se caută entitățile corespondente în baza de date;
- Se creează sau actualizează conturile de utilizatori din tabela User și profilul acestora din tabela UserProfile.

4.1.4. Gestionarea conturilor de utilizator

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

1. Autentificarea utilizatorilor

Pentru autentificarea utilizatorilor, endpoint-ul LoginView primește în request cu două câmpuri:

- Un identificator care poate să fie adresa de email sau username-ul;
- Parola corespunzătoare contului.

Se încearcă mai întâi identificarea utilizatorului după email, în cazul în care nu este găsit, se încearcă 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 pentru a permite autentificarea, împreună cu detalii despre utilizator.

2. Verificarea existenței unui email

Endpoint-ul CheckEmailExistsView este utilizat pentru două funcționalități:

- Se verifică dacă o adresă de email este deja înregistrată în sistem. Această funcționalitate este importantă pentru a preveni crearea de conturi duplicate;
- Trimite un email către utilizator în următoarele cazuri:
    - Pentru a crea un cont de utilizator, aceasta ajută utilizatorii care nu dețin un cont de utilizator, să nu lase persoane să își creeze conturi și să acceseze datele din baza de date ale altei persoane;
    - Pentru resetarea parolei, aceasta ajuta utilizatorii care dețin un cont să își poată reseta parola din anumite motive.

3. Activarea conturilor

Pentru utilizatorii creați anterior de către un administrator prin importarea lor dintr-un fișier Excel, procesul de activare a contului se realizează astfel prin endpoint-ul ActivateUserView. Acesta primește adresa de email care a fost salvată în localStorage înainte

să fie trimis email către utilizatorul respectiv, token-ul JWT generat anterior pentru crearea contului, username-ul, și parola care au fost sunt introduse în link-ul primit pe email. Aceste date sunt trimite într-un request POST. Se verifică existența utilizatorului după email și dacă acesta nu are un username diferit setat. Dacă toate condițiile sunt îndeplinite, se actualizează username-ul și parola, iar contul devine activ.

4. Resetarea parolei

Pentru utilizatorii care dețin deja un cont procesul de resetare a parolei se realizează astfel prin endpoint-ul ResetPasswordConfirmView Acesta primește adresa de email care a fost salvată în localStorage înainte să fie trimis email către utilizatorul respectiv, token-ul JWT generat anterior pentru resetarea parolei, și noua parola care au fost introdusă în link-ul primit pe email. Aceste date sunt trimise într-un request POST. Se caută utilizatorul după adresa de email, după parola este actualizată și salvată în baza de date.

4.2. Implementarea Front-End

Componenta front-end a aplicației a fost dezvoltată utilizând biblioteca React, împreună cu Vite ca instrument de build, pentru o experiența de dezvoltare rapidă și eficientă.

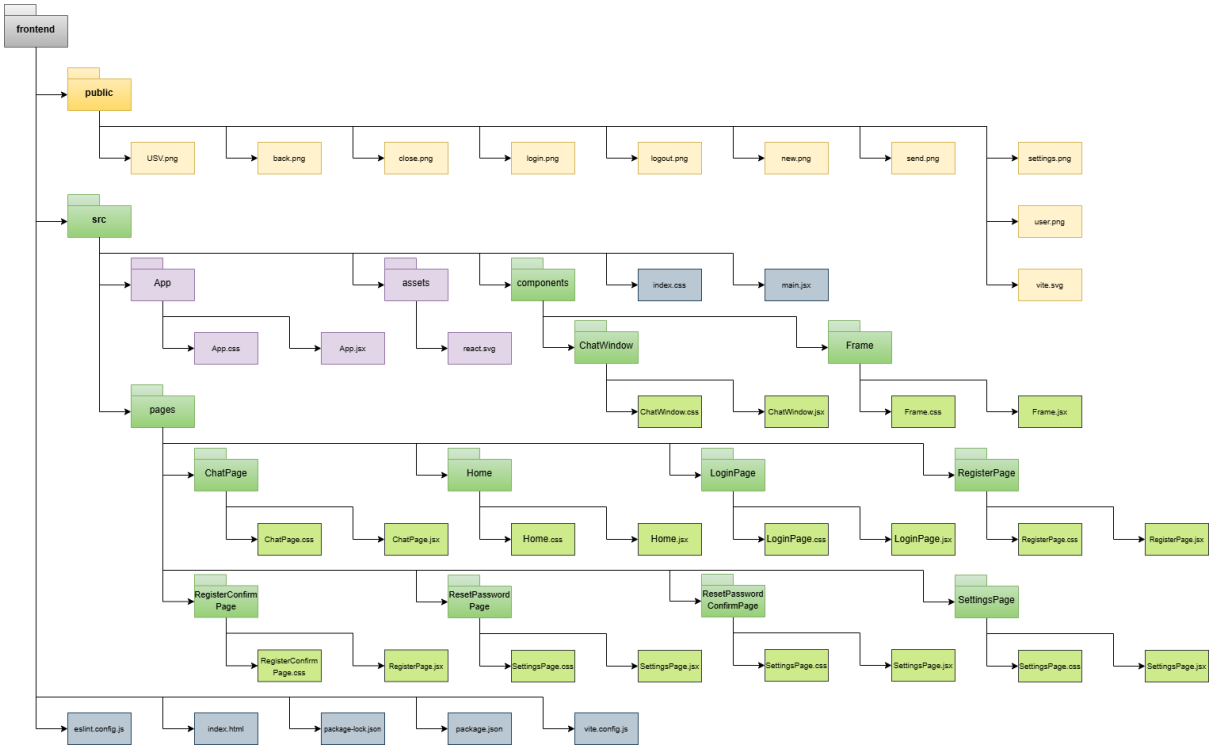
Această parte a aplicației are rolul de a furniza o interfața interactivă și intuitivă prin care utilizatorii pot interacționa cu asistentul virtual și cu funcționalitățile acesteia.

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

Interfața comunică 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.

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

În Figura 4.2 este prezentată structura directorului front-end care include fișiere și directoarele generate automat de React + Vite, precum si cele adăugate manual 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:

- eslint.config.js – fișierul de configurare pentru ESLint;
- index.html – fișierul HTML principal în care se încarcă aplicația React;
- package.json, package-lock.json – fișiere de configurare care definesc dependențele proiectului;
- vite.config.js – fișierul de configurare pentru instrumentul Vite, care construiește și rulează aplicația.

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

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

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

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

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

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

Directorul „/components/” conține componentele React reutilizabile:
- /ChatWindow/ (ChatWindow.css + ChatWindow.jsx) – componenta pentru fereastra de chat;
- /Frame/ (Frame.css + Frame.jsx) – componenta care oferă cadrul vizual 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;
- /RegisterConfirmPage/
  - (RegisterConfirmPage.css + RegisterConfirmPage.jsx) – pagina pentru înregistrarea utilizatorilor;
- /RegisterPage/
  - (RegisterPage.css + RegisterPage.jsx) – pagina pentru trimiterea cererii pe email cu formularul pentru înregistrarea utilizatorilor noi;
- /ResetPasswordConfirmPage/
  - (ResetPasswordConfirmPage.css + ResetPasswordConfirmPage.jsx) – pagina pentru resetarea parolei;
- /ResetPasswordPage/
  - (ResetPasswordPage.css + ResetPasswordPage.jsx) – pagina pentru trimiterea cererii pe email cu formularul pentru resetarea parolei;
- /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 încarcă și afișa pagina oficiala a Facultății de Inginerie Electrică și Știința Calculatoarelor într-un iframe, astfel încât 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 back-end, solicitând generarea unui mesaj personalizat de bun venit. 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 este dat la ultimul mesaj. Există un buton în dreapta sus a acestei componente pentru a naviga către o pagină de chat extinsă.

4.2.3. Descrierea funcțională a paginilor

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

Pagina Login reprezintă punctul în care utilizatorul se poate autentifica. Pagina conține un formular de autentificare cu două câmpuri username sau email și parola. La apăsarea butonului „Conectare” sau la apăsarea tastei enter, datele introduse sunt trimise cu o cerere POST către endpointul de autentificare. Dacă contul e valid, răspunsul conține un token JWT și informații despre utilizator care apoi sunt salvate în localStorage și utilizatorul este redirecționat către pagina de chat. În pagina Login mai există 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 utilizatorul este redirecționat către pagina Register.

Pagina Register permite trimiterea cererii pe email cu formularul pentru 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 a trimite cererea pe email-ul respectiv. 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, este trimis un link pe email-ul introdus cu un formular în care utilizatorul va trebui să îl completeze, dacă trimiterea email-ului a reușit, îi este afișat utilizatorul că email-ul a fost trimis către linkul introdus și îi apare butonul „Înapoi

la Login” care la apăsarea lui utilizatorul este redirecționat către pagina Login. Pagina Register mai conține un buton de back în partea stânga sus a paginii unde utilizatorul este redirecționat către pagina Login.

Pagina RegisterConfirm se poate accesa doar prin link-ul care conține un token JWT de valabilitate trimis pe email în care 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 Login, în caz contrar este afișat un mesaj de eroare. Pagina RegisterConfirm mai conține un buton de back în partea stânga sus a paginii unde utilizatorul este redirecționat către pagina Login.

Pagina ResetPassword permite trimitea cererii pe email cu formularul pentru resetarea parolei unui cont de utilizatori existent. Procesul este împărțit în doua etape pentru a verifica validarea emailului de student, și de a trimite cererea pe email-ul respectiv. 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, este trimis un link pe email-ul introdus cu un formular în care utilizatorul va trebui să îl completeze, dacă trimiterea email-ului a reușit, îi este afișat utilizatorul că email-ul a fost trimis către linkul introdus și îi apare butonul „Înapoi la Login” care la apăsarea lui utilizatorul este redirecționat către pagina Login. Pagina ResetPassword mai conține un buton de back în partea stânga sus a paginii unde utilizatorul este redirecționat către pagina Login.

Pagina ResetPasswordConfirm se poate accesa doar prin link-ul care conține un token JWT de valabilitate trimis pe email în care sunt afișate câmpurile pentru a introduce noua 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 ResetPasswordConfirmView cu datele introduse, dacă înregistrarea este reușită utilizatorul este redirecționat către pagina Login, în caz contrar este afișat un mesaj de eroare. Pagina ResetPasswordConfirm mai conține un buton de back în partea stânga sus a paginii unde utilizatorul este redirecționat către pagina Login.

Pagina Chat reprezintă componenta centrală a aplicației. Interfața acestei pagini este de tip chat unde utilizatorul poate ține o conversația cu Alex. Pagina este disponibilă doar pentru utilizatorii autentificați și permite afișarea istoricului conversațiilor. 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 conversației. 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. Pagina Chat conține un buton în partea stânga sus a paginii unde utilizatorul este redirecționat către pagina Home, și un buton în partea dreapta sus a paginii care la apăsarea lui se deschide un meniu dropdown unde un utilizator normal are doar un buton de deconectare, dar dacă utilizatorul este un administrator îi mai apare un buton numit „Setări” unde la apăsarea lui administratorul este redirecționat către pagina de Settings.

Pagina Settings este o pagină destinată doar administraților, 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, unde este procesat 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, care declanșează descărcarea documentele esențiale folosite de asistentul virtual, acest proces asigură generarea de răspunsuri să fie actualizată și corectă.

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 recentă

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 recentă

5.2. Instalare

Repository-ul se poate colna de pe GitHub folosind comanda:

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

5.2.1. Configurare Back-End

Pentru a configura partea back-end, se urmează următorii pași:

1. Asigură-te că ai instalate următoarele:
    - Python versiunea 3.13.2 sau mai recentă;
    - pip versiunea 24.3.1 sau mai recentă.

2. Instalează GTK 3 Runtime de la adresa:

https://github.com/tschoonj/GTK-for-Windows-Runtime-Environment-Installer/releases

3. Accesează directorul virtual-assistant din repository: *cd virtual-assistant*

4. Accesează subdirectorul backend din repository: *cd backend*

5. Creează un fișier .env în rădăcina proiectului și adaugă următoarele linii:
SENDGRID_API_KEY=[API_KEY_SENDGRID]
SENDGRID_FROM_EMAIL=[EMAIL_VALIT]
Notă: Înlocuiește [API_KEY_SENDGRID] cu o cheie API de pe SendGrid și [EMAIL_VALIT] cu un email valid (exemplu: user@example.com)

6. Creează un mediul virtual pentru proiect: *python -m venv .venv*

7. Activează mediul virtual: *.\.venv\Scripts\activate*

8. Instalează dependențele din fișierul *requirements.txt*: *pip install -r requirements.txt*

9. Creează migrațiile bazei de date: *python manage.py makemigrations*

10. Aplică migrațiile: *python manage.py migrate*

11. Instalează aplicația Ollama de pe site-ul oficial: *https://ollama.com/*

12. Descarcă modelul Llama 3.2 utilizat pentru a genera răspunsuri: *ollama pull llama3.2*

13. Rulează serverul Django: *python manage.py runserver*

14. Aplicația back-end va fi disponibila pe: *http://127.0.0.1:2307/*

5.2.2. Configurare Front-End

Pentru a configura partea front-end, se urmează următorii pași:

1. Asigură-te că ai instalate următoarele:
   - Node.js versiunea 22.13.1 sau mai recentă
   - npm versiunea 10.9.2 sau mai recentă

2. Accesează directorul virtual-assistant din repository: *cd virtual-assistant*

3. Accesează subdirectorul frontend din repository: *cd frontend*

4. Creează un fișier .env în rădăcina proiectului și adaugă următoarea linie:

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

Notă: Înlocuiește <IP_BACKEND> și <PORT_BACKEND> cu IP-ul și portul folosit de aplicația back-end (exemplu: 127.0.0.1:2307)

5. Instalează dependențele proiectului: *npm install*

6. Rulează aplicația: *npm run dev*

7. Aplicația front-end va fi disponibilă pe: *http://127.0.0.1:2002/*

5.3. Utilizare

Pentru a folosi aplicația trebuie să deschizi un browser și să introdu-ți adresa URL „http://127.0.0.1:2002/”. Acolo va fi afișată o copie a site-ului oficial FIESC.

În partea dreapta jos a paginii este un buton cu numele „Alex - Asistent Virtual” care va deschide o fereastră de chat simplă, unde utilizatorul poate pune întrebări pentru a cere informații de exemplu, „Ce documente sunt necesare pentru bursa socială ocazională?”.

Înainte de a folosi aplicația este necesară autentificarea cu contul de administrator. Ca să poți face acest lucru trebuie să apeși pe butonul din stânga sus al ferestrei de chat (un simbol de +), vei fi redirecționat către pagina de autentificare, acolo trebuie să te conectezi cu un cont de administrator:

- Nume utilizator: admin;
- Parolă: Admin#1.

După autentificare vei fi redirecționat către o pagina de chat. În partea dreapta sus ai un buton (un simbol de omuleț) la apăsarea lui se deschide un meniu dropdown cu următoarele opțiuni:

- Deconectare pentru a ieși din cont;
- Setări pentru a accesa setările aplicației (această secțiune poate fi accesată doar de administrator).

Accesând secțiunea setări vei fi redirecționat către pagina de setări, unde administratorul va avea acces la:

- Importarea utilizatorilor dintr-un fișier Excel care trebuie să conțină următorul format:
    - Rândul 1, coloana A: universitatea;
    - Rândul 2, coloana A:facultatea;
    - Rândul 3, coloana A: domeniul;
    - Rândul 4, coloana A: programul de studii;
    - Rândul 5, coloana A: anul universitar;
    - Rândul 6, coloana A: anul de studii;
    - Rândul 7, coloana A: trebuie lăsată liberă;
    - Rândul 8 capul de tabel, cu următoarele coloane:
        - Coloana A: Nr. crt.;
        - Coloana B: ID student;
        - Coloana C: Numele;
        - Coloana D: Prenumele;
        - Coloana E: Email;
        - Coloana F: Identificatorul de semigrupă (de exemplu 3142B, unde 3 reprezintă facultatea, 1 specializarea, 4 anul, 2 grupa, B semigrupa);
        - Coloana G: Forma de finanțare (cu taxă/fără taxă);
    - Începând cu linia 9, sunt introduși toți utilizatorii, în ordinea coloanelor menționate. Câte un utilizator pe rând.
- Actualizarea documentelor folosite de asistentul virtual, prin apăsarea butonului „Actualizare”. Această operație trebuie executată înainte de a folosi aplicația pentru a descarcă toate documentele necesare cu informații de pe site-ul oficial FIESC unde va trebui să aștepte aproximativ doua 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.

Pentru a primi un răspuns corect este necesar să formulezi întrebările clar și specific, de preferat fără diacritice, pentru că aplicația preia cuvintele din întrebarea care a fost adresată și pe baza acelor cuvinte le compară cu numele documentelor pentru a extrage informațiile necesare care ajută la generarea unui răspuns corespunzător.

Pentru a putea să îți 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. Ca să îți poți 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 primi un mesaj pe email-ul trimis cu un link care, odată accesat, te va redirecționat către o pagină web, unde îți va cere să introduci un nume de utilizator o parolă (parola trebuie să fie de minim 5 caractere, să conțină o litera mare, un caracter special și un număr) și să confirmi parola, o data ce contul a fost creat vei fi redirecționat către pagina de autentificare.

În cazul în care ai uitat parola de la cont sau îți dorești să îți resetezi parola, pe pagina de autentificare, acolo sub butonul de ați face un cont ai un buton numit „Ai uitat parola? Resetează-o!”, pe care dacă îl apeși vei fi redirecționat către pagina de resetare a parolei. Dacă acea adresă de email exista vei primi un mesaj pe email-ul trimis cu un link care, odată accesat, te va redirecționat către o pagină web, unde îți va cere să introduci parola noua (parola trebuie să fie de minim 5 caractere, să conțină o litera mare, un caracter special și un număr) și să confirmi parola, o data ce parola a fost resetată vei fi redirecționat către pagina de autentificare.

Avantajele unui cont de utilizator sunt următoarele:

1. Istoricul conversației este salvat și asistentul virtual are acces la datele personale în care ajută utilizatorul să nu mai introducă date care au fost menționate anterior sau date personale aflate deja în baza de date.
2. Utilizatorul are acces la 2 comenzi:
   - „comenzi” – această comandă va returna toate comenzile valabile;
   - „generează o adeverință cu motivul [motiv]” – aceasta comandă va genera o adeverință în format PDF pe baza unui șablon, care e completat automat cu datele din baza de date specifice utilizatorului, și va returna un link de unde se poate descărca acel fișier.

6. Rezultate experimentale

În continuare 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 întrebările utilizatorului

6.1.1. Analiza acurateței răspunsurilor furnizate la întrebările privind aspectele administrative universitare

În cadrul testării sistemului au fost considerate mai multe scenarii. În continuare sunt prezentate doar câteva din testele realizate și rezultatele obținute.

Una din întrebările adresate sistemului în cadrul procesului de evaluare a răspunsurilor referitoare la aspectele universitare a fost: „Ce documente sunt necesare pentru bursa sociala ocazionala?”

Răspunsul corect la această la întrebare trebuie să conțină următoarea listă:

- 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;
- declarația 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.

Mesajul care a fost generat și primit de către asistentul virtual: „Pentru bursa sociala ocazionala, v-ati rugat sa depuneti urmatoarele documente justificative:

1. Cererea studentului (Anexa 8)

2. Declaratie pe propria raspundere privind veniturile nete (Anexa 8)
3. Acordul olograf al studentului privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MANA
4. Acordul olograf al fiecarui membru al familiei care realizeaza venituri privind prelucrarea datelor cu caracter personal (Anexa 5) – SCRIS INTEGRAL DE MANA
5. Declaratia studentului privind autenticitatea documentelor depuse (Anexa 9)
6. Ancheta sociala, in cazul in care nimeni din familia studentului, incluzandu-l si pe acesta, nu realizeaza venituri sau cel putin un parinte al studentului lucreaza sau domiciliaza in strainatate
7. Copii de pe certificatul/certificatele de deces al/ale parintelui/parintilor, de pe acte de stare civila/decizia instantei de mentinere a starii de arest/raportul de ancheta sociala in cazul parintilor disparuti, daca este cazul.”

Concluzie: Sistemul a identificat corect întrebarea fiind una legată de facultate. A reușit să găsească un document, pe care conținutul lui la ajutat să genereze un răspuns corespunzător și complet. Dintr-un set de 160 de fișiere PDF care conțineau informațiile legate de facultate, sistemul a reușit să selecteze documentele necesare și să extragă conținutul corespunzător pentru a genera răspunsul. În concluzie, testul a avut succes, se poate aprecia că sistemul funcționează corect pentru căutarea, selecția, și validarea informațiilor, dar, în același timp, se poate remarca faptul că LLaMA 3.2 nu generează răspunsuri corecte gramatical în limba română.

6.1.2. Evaluarea răspunsurilor generale

Întrebarea considerată pentru evaluare a răspunsurilor generale: „Care sunt datele mele personale?”

Răspunsul corect la această la întrebare trebuie să conțină următoarea listă:

- ID-ul de student: 66126
- Facultatea: Facultatea de Inginerie Electrică și Știința Calculatoarelor
- Specializare: Calculatoare
- Grupa: 2
- Semigrupa: B
- Tipul taxei: Fără Taxă
- Anul universitar: 2024 – 2025
- Anul de studiu: 4

Mesajul care a fost generat și primit de către asistentul virtual: „Datele personale ale dvs. sunt:

- id_student: 66126
- facultate: Facultatea de Inginerie Electrica si Stiinta Calculatoarelor
- specializare: Calculatoare
- grupa: 2
- semigrupa: B
- tip_taxa: fara taxa
- an_universitar: 2024/2025
- an_studiu: 4”

Concluzie: Sistemul a identificat corect întrebarea fiind una generală. A reușit să genereze un răspunsul corespunzător pe baza informațiilor personale ale utilizatorului. În concluzie testul a avut succes pentru că sistemul funcționează corect pentru extragerea datelor personale ale utilizatorului.

6.2. Testarea generării adeverinței

În cadrul acestei activități de testare a fost analizată funcționalitatea generării automate a adeverințelor în funcție dacă utilizatorul este autentificare sau nu. Scopul este de a verifica dacă asistentul generează corect o adeverința pentru utilizatorii autentificați și le transmite utilizatorilor care nu sunt autentificați „Doar persoanele autentificate pot genera o adeverința!”.

Autentificat: NONE

Comanda folosită: „Genereaza o adeverinta cu motivul angajare”

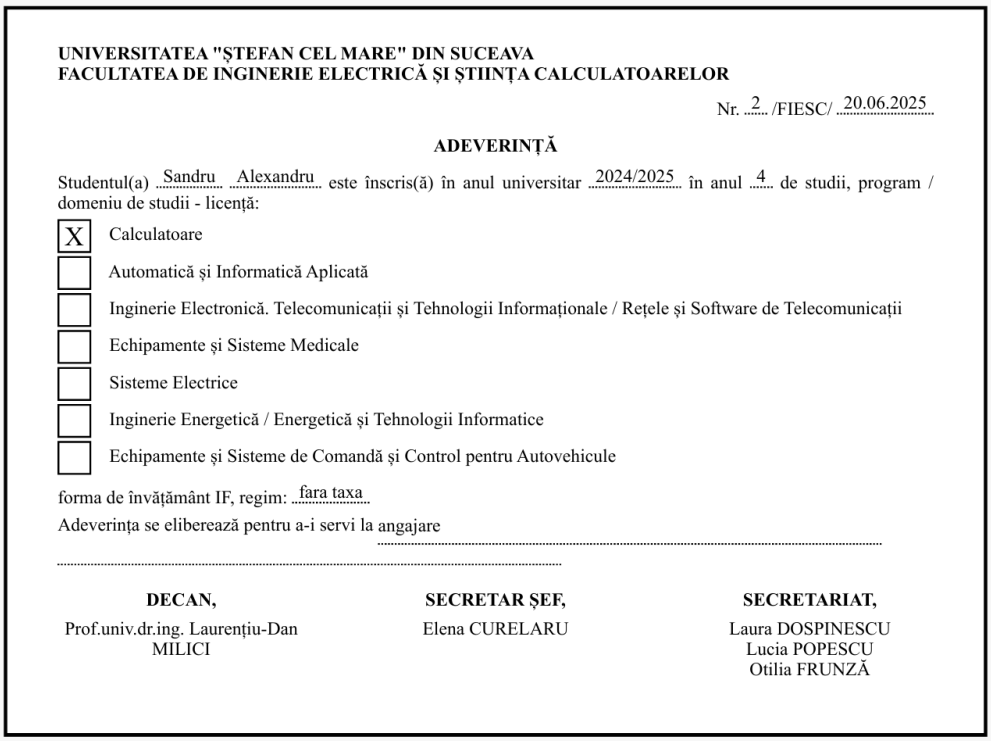
Mesaj primit: „Doar persoanele autentificate pot genera o adeverința!”

Concluzie: Sistemul refuză generarea adeverinței și afișează mesajul corespunzător pentru utilizatorii neautentificați. În concluzie, se poate aprecia că testul a avut succes.

Autentificat: Șandru Alexandru

Comanda folosită: „Genereaza o adeverinta cu motivul angajare”

Mesaj primit:



UNIVERSITATEA "ȘTEFAN CEL MARE" DIN SUCEAVA
FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR

Nr. 2 /FIESC/ 20.06.2025

ADEVERINȚĂ

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 ȘEF, Elena CURELARU

SECRETARIAT, Laura DOSPINESCU, Lucia POPESCU, Otilia FRUNZĂ

Concluzie: Sistemul a generat o adeverința conform datelor din baza de date a utilizatorului. În concluzie testul a avut succes, putându-se aprecia că sistemul funcționează corect.

7. Concluzii și direcții viitoare de dezvoltare

Proiectul denumit „Eficientizarea activităților din cadrul secretariatului facultății prin implementarea unui asistent conversațional bazat pe modele lingvistice mari” a urmărit crearea unei soluții inteligente pentru a simplifica accesul studenților la informații administrative și să automatizeze procesul de generare a adeverințelor. Sistemul a integrat cu succes aceste soluții, oferind răspunsuri rapide și relevante la întrebările utilizatorilor, bazate pe documentele descărcate de pe site-ul facultății, scurtând astfel timpul de așteptat și volumul de muncă al secretariatului. Implementarea asistentului virtual a demonstrat că tehnologiile bazate pe modele lingvistice mari pot fi aplicate eficient în contextul administrativ.

7.1. Concluzii principale

În urma testelor realizate, sistemul a dovedit o capacitate bună de a identifica și răspunde corect la întrebările utilizatorilor, bazându-se pe documentele descărcate de pe site-ul facultății. Funcționalitatea de generare a adeverințelor funcționează conform așteptărilor, fiind accesibilă doar utilizatorilor autentificați.

7.2. Stadiul actual al proiectului

Baza de date:

- Este completă și funcțională, conține toate informațiile necesare pentru folosirea aplicației;
- Sunt necesare retușuri minore la structura bazei de date.

Back-End:

- E complet și funcțională, conține toate endpointurile necesare pentru folosirea aplicației;
- Extinderea endpointului DownloadFilesView pentru a putea descarcă toate documentele aflate pe site, nu doar de pe secțiunea documente;
- De adăugat sau completat, mesajele de eroare și excepțiile care au rămas necompletate;
- De împărțiți codul mai bine pe fișiere, nu doar câteva;
- De făcut retușuri minore pe partea de text a mesajelor de eroare și a excepțiilor pentru o claritate mai buna a problemei.

Front-End:

- E complet și asigură o experiență de utilizate ușoară;
- Sunt necesare modificări minore pe partea de validare și afișarea validărilor;
- Sunt necesare retușuri minore pe partea textelor care sunt afișate.

7.3. Direcții viitoare de dezvoltare

Printre direcțiile viitoare de dezvoltare se numără îmbunătățirea sistemului 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 în numele lor cuvinte asemănătoare cu întrebarea;
- Limitarea conținutului extras din documente, preia doar conținutul care are nevoie nu întregul document;
- Ajutarea utilizatorului la completarea documentelor;
- Salvarea răspunsurilor și a întrebărilor utilizate frecvent, pentru a scădea timpul de așteptare a răspunsului.

De asemenea, ar fi utilă adăugarea unei funcționalități de tip 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 reprezintă o altă funcționalitate ce este avută în vedere:

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

Având în vedere dezvoltarea continuă a LLM-urilor, în continuare, se are în vedere considerarea altor modele de tip LLM, mai performante și realizarea mai multor teste relevante.

În vederea creșterii nivelului de securitate pentru conturile considerate în cadrul aplicației, reducând astfel riscul accesului neautorizat, se are în vedere implementarea autentificării în doi pași (2FA) pentru conturi de administrator.

Pentru asigurarea calității codului se vor realiza mai multe teste, inclusiv testare automată, având în vedere introducerea testelor pentru back-end.

Bibliografie

[1] „ASE București, prima universitate din România care introduce un asistent AI pentru studenți," 13 Aprilie 2025. [Interactiv]. Available: https://newsbucuresti.ro/ase-bucuresti-prima-universitate-din-romania-care-introduce-un-asistent-ai-pentru-studenti/. [Accesat 6 Mai 2025].

[2] „Business schools ease their resistance to AI," 16 Martie 2025. [Interactiv]. Available: https://www.ft.com/content/daa0f68d-774a-4e5e-902c-5d6e8bf687dc. [Accesat 16 Iunie 2025].

[3] „About Python," [Interactiv]. Available: https://www.python.org/about/.

[4] „Best Web Development Languages," 28 Aprilie 2025. [Interactiv]. Available: https://www.geeksforgeeks.org/blogs/best-web-development-languages/.

[5] „Top AI Programming Languages in 2025: A Comprehensive Guide," [Interactiv]. Available: https://digiscorp.com/top-ai-programming-languages-in-2025-a-comprehensive-guide/.

[6] „The Web framework for perfectionists with deadlines," [Interactiv]. Available: https://www.djangoproject.com/.

[7] „Why Django?," [Interactiv]. Available: https://www.djangoproject.com/start/overview/.

[8] „Django Documentation," [Interactiv]. Available: https://docs.djangoproject.com/en/5.2/.

[9] „Django REST framework," [Interactiv]. Available: https://www.django-rest-framework.org/.

[10] „What is a Large Language Model (LLM)," 22 Ianuarie 2025. [Interactiv]. Available: https://www.geeksforgeeks.org/large-language-model-llm/.

[11] „Llama 3.2: Revolutionizing edge AI and vision with open, customizable models," 25 Septembrie 2024. [Interactiv]. Available: https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

[12] „Ollama," [Interactiv]. Available: https://ollama.com/.

[13] Ollama, „ollama-python," [Interactiv]. Available: https://github.com/ollama/ollama-python.

[14] „SendGrid," [Interactiv]. Available: https://sendgrid.com/en-us.

[15] „What is Retrieval-Augmented Generation (RAG) ?," 10 Februarie 2025. [Interactiv]. Available: https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/.

[16] „Top 10 Front-End Frameworks in 2025," 13 Decembrie 2024. [Interactiv]. Available: https://www.geeksforgeeks.org/blogs/top-front-end-frameworks/.

[17] „Getting Started," [Interactiv]. Available: https://vite.dev/guide/.

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):
    # Endpoint public – nu este necesara autentificarea
    permission_classes = [AllowAny]

    # Instantierea clasei RAG
    rag = RAG()

    def post(self, request):
        try:
            # Extrage intrebarea trimisa de utilizator
            question = request.data.get("message")

            # Elimina diacriticele din intrebare
            question = remove_diacritics(question)

            # Verifica daca intrebarea este goala
            if not question:
                return Response(
                    {"error": "Mesajul nu poate fi gol."},
                    status=status.HTTP_400_BAD_REQUEST,
                )

            # Comanda pentru listarea comenzilor disponibile
            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"
                )

                # Salveaza comanda in istoric daca utilizatorul este autentificat
                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)

            # Generarea unei adeverinte pentru utilizatori autentificati
            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()

    # Verifica daca motivul a fost specificat
    if not motiv:
        return Response(
            {"answer": "Te rog specifica motivul pentru adeverinta."},
            status=status.HTTP_200_OK,
        )

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

    # Genereaza datele pentru noua adeverinta
    numar = Adevetinta.objects.count() + 1
    data_emitere = timezone.now().date()

    # Creeaza obiectul Adeverinta in baza de 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),
    )

    # Imparte motivul in doua linii
    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()

    # Creeaza contextul pentru template-ul HTML
```

```
        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,
        }

        # Genereaza continutul HTML al adeverintei
        html_string = render_to_string("adeverinta/adeverinta.html", context)

        # Genereaza fisierul PDF pe baza HTML-ului
        pdf_bytes = HTML(
            string=html_string, base_url=request.build_absolute_uri("/")
        ).write_pdf()

        # Creeaza un nume unic pentru fisierul PDF
        unique_id = str(uuid.uuid4())
        filename = f"adeverinta_{unique_id}.pdf"
        pdf_path = os.path.join(settings.MEDIA_ROOT, filename)

        # Creeaza folderul MEDIA daca nu exista
        if not os.path.exists(settings.MEDIA_ROOT):
            os.makedirs(settings.MEDIA_ROOT)

        # Scrie fisierul PDF pe disc
        with open(pdf_path, "wb") as f:
            f.write(pdf_bytes)

        # Genereaza URL-ul public pentru fisierul PDF
        pdf_url = request.build_absolute_uri(settings.MEDIA_URL + filename)

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

        # Salveaza in istoric intrebarea si raspunsul
        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)

    # Blocare generare adeverinta pentru utilizatori neautentificati
    if question.lower().startswith("genereaza o adeverinta cu motivul"):
        answer_text = f"Doar persoanele autentificate pot genera o adeverinta!"
        return Response({"answer": answer_text}, status=status.HTTP_200_OK)
```

```
        # Preluare datele personale pentru utilizatorul autentificat
        user_profile_data = None
        if request.user and request.user.is_authenticated:
            user_profile_data = get_user_profile_data(request.user)

        # Preluare istoric conversatie pentru utilizatorul autentificat
        conversation_history = None
        if request.user and request.user.is_authenticated:
            conversation_history = ConversationHistory.objects.filter(
                user=request.user
            ).order_by("id")[:10]

        # Apeleaza RAG pentru a genera raspunsul pe baza intrebarii si a contextului
        answer = self.rag.get_response(
            question, user_profile_data, conversation_history
        )

        # Salveaza conversatia in istoric daca utilizatorul este autentificat
        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
            )

        # Returneaza raspunsul final
        return Response({"answer": answer}, status=status.HTTP_200_OK)

    except Exception as e:
        # Tratare erori interne de server
        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"):
        """
        Initializeaza o instanta a clasei RAG, setand directorul in care sunt stocate
fisierele PDF.
        """
        self.pdf_directory = pdf_directory
        self.context = []

    def is_faculty_question(self, question):
        """
        Determina daca intrebarea utilizatorului este legata de facultate sau este una
generala.
        Trimite intrebarea catre un model LLM care raspunde cu 'facultate' sau 'generala'.

        Parametru:
            question (str): Intrebarea utilizatorului.

        Returneaza:
            str: 'facultate' sau 'generala'
        """
        messages = [
            {
                "role": "system",
                "content": (
                    "Tu trebuie sa decizi daca intrebarea este legata de Facultatea de
Inginerie Electrica si Stiinta Calculatoarelor "
                    "sau daca este o intrebare generala, care nu are legatura cu
facultatea.\n\n"
                    "Intrebarile legate de facultate includ, dar nu se limiteaza la: proceduri
administrative, documente necesare pentru burse, programari, examene, orar, structura
facultatii, resurse si servicii oferite de facultate.\n"
                    "Intrebarile generale pot fi orice intrebare care nu tine de facultate, de
exemplu: 'Cum il cheama pe creatorul Python?', 'Cat face 1+1?', 'Care este capitala Frantei?',
'Retine o anumita valoarea', 'Care sunt datele mele personale?' \n\n"
                    "Raspunde doar cu un singur cuvant:\n"
                    "- 'facultate' daca intrebarea este legata de facultate,\n"
                    "- 'generala' daca intrebarea este generala sau nu are legatura cu
facultatea."
                ),
            },
            {"role": "user", "content": question},
        ]
```

```
        try:
            response = ollama.chat(model="llama3.2", messages=messages)
            verdict = response["message"]["content"].strip().lower()

            return verdict
        except Exception as e:
            return f"Eroare la functia is_faculty_question: {str(e)}"

    def extract_words_from_filename(self, filename):
        """
        Extrage cuvintele din numele unui fisier PDF, inlocuind separatorii comuni cu spatii.

        Parametru:
            filename (str): Numele fisierului.

        Returneaza:
            list: Lista de cuvinte extrase din numele fisierului.
        """
        words = re.sub(r"[-_]", " ", filename)
        words = words.replace(".pdf", "").split()

        return words

    def read_pdf_content(self, pdf_path):
        """
        Citeste continutul text dintr-un fisier PDF.

        Parametru:
            pdf_path (str): Calea catre fisierul PDF.

        Returneaza:
            str: Continutul text al fisierului, concatenat pe toate paginile.
        """
        try:
            reader = PdfReader(pdf_path)
            content = ""

            for page in reader.pages:
                content += page.extract_text()

            return content
        except Exception as e:
            return f"Eroare la functia read_pdf_content: {str(e)}"

    def find_matching_files(self, question):
        """
        Gaseste fisiere PDF al caror nume contine cuvinte care apar si in intrebare.

        Parametru:
            question (str): Intrebarea utilizatorului.
```

```
        Returneaza:
            list: Nume de fisiere relevante, ordonate descrescator dupa numarul de potriviri.
        """
        matching_files = []
        question_words = re.sub(r"[^\w\s]", "", question).lower().split()
        files_with_scores = []

        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()
                )
                matched_words_count = sum(word in file_words for word in question_words)

                if matched_words_count > 0:
                    files_with_scores.append((filename, matched_words_count))

        files_with_scores.sort(key=lambda x: x[1], reverse=True)
        matching_files = [filename for filename, score in files_with_scores]

        return matching_files

    def get_context(self, filename):
        """
        Obtine continutul text dintr-un fisier PDF specificat.

        Parametru:
            filename (str): Numele fisierului PDF.

        Returneaza:
            str: Textul extras din fisier.
        """
        pdf_path = os.path.join(self.pdf_directory, filename)
        content = self.read_pdf_content(pdf_path)

        return content

    def get_response(self, question, user_profile_data=None, conversation_history=None):
        """
        Genereaza un raspuns pentru intrebarea utilizatorului, folosind documente relevante
sau doar modelul LLM.

        Parametri:
            question (str): Intrebarea utilizatorului.
            conversation_history (list): Istoricul conversatiei, daca exista.
```

```
        Returneaza:
            str: Raspunsul generat de model.
        """
        tip_intrebare = self.is_faculty_question(question)

        if tip_intrebare == "facultate":
            matching_files = self.find_matching_files(question)
            question_no_diacritics = remove_diacritics(question).lower()
            question_words = re.findall(r"\w+", question_no_diacritics)

            for filename in matching_files:
                context = self.get_context(filename)

                if not context:
                    continue

                messages = [
                    {
                        "role": "system",
                        "content": (
                            "Cand oferi un raspuns, asigura-te ca este prezentat intr-un mod
placut intregul mesaj: cu aliniere, italic, bold si alte elemente de formatare adecvate
contextului. Stilul raspunsului ar trebui sa fie prietenos si primitor, pentru a face
utilizatorul sa se simta confortabil."
                            "Te rog sa raspunzi in limba romana fara diacritice."
                            "Numele tau personal este „Alex” si esti asistentul virtual al
Facultatii de Inginerie Electrica si Stiinta Calculatoarelor. "
                            "Iti poti spune care e numele tau doar daca te intreba cineva sau
la mesaje de bun venit cand te prezinti. "
                            "Daca trebuie sa retii un lucru, vei raspunde cu un mesaj
afirmativ si vei mentiona clar ceea ce trebuie sa retii."
                        ),
                    },
                    {
                        "role": "system",
                        "content": (
                            "Cand oferi un raspuns, asigura-te ca este prezentat intr-un mod
placut intregul mesaj: cu aliniere, italic, bold si alte elemente de formatare adecvate
contextului. Stilul raspunsului ar trebui sa fie prietenos si primitor, pentru a face
utilizatorul sa se simta confortabil."
                            f"Datele personale ale utilizatorului sunt: {user_profile_data}\n"
                            "Foloseste **doar** informatiile din contextul pentru a raspunde.
Respecta urmatoarele reguli stricte:\n"
                            "- Daca intrebarea contine o categorie (ex. bursa sociala
ocazionala), raspunde **doar daca** acea categorie este mentionata **identic** sau ca titlu
clar in context.\n"
                            "- Daca contextul se refera la o subcategorie (ex. bursa sociala
ocazionala categoria bolnavi) dar intrebarea este generala, raspunde doar: NU.\n"
                            "- Nu raspunde daca informatia este partiala, lipseste, sau este
pentru alt caz decat cel intrebat.\n"
```

```
                            "- Nu interpreta. Nu deduce. Nu extinde. Nu completa.\n"
                            "- Daca nu exista o potrivire perfecta intre intrebare si context,
raspunde doar: NU.\n\n"
                            "Raspunde clar si complet cu toate informatiile gasite in limba
romana, fara diacritice. Nu mentiona sursa informatiilor."
                            f"Contextul este: {context}"
                        ),
                    },
                ]

                if conversation_history:
                    for item in conversation_history:
                        messages.append({"role": "user", "content": item.question})
                        messages.append({"role": "assistant", "content": item.answer})

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

                try:
                    response = ollama.chat(model="llama3.2", messages=messages)
                    raspuns_text = response["message"]["content"].strip()
                    raspuns_text = remove_diacritics(raspuns_text)

                    negative_indicators = [
                        "nu exista",
                        "nu au fost",
                        "nu sunt suficiente",
                        "nu consider",
                        "nu pot raspunde",
                        "nu se poate",
                        "nu am gasit",
                        "nu sunt informatii",
                        "nu am suficiente",
                        "nu raspund",
                        "nu pot oferi",
                        "nu este suficienta",
                        "nu este suficient",
                    ]

                    if raspuns_text.lower() == "nu" or any(
                        phrase in raspuns_text.lower() for phrase in negative_indicators
                    ):
                        continue

                    if any(word in raspuns_text.lower() for word in question_words):
                        return raspuns_text
                    else:
                        continue

                except Exception as e:
                    return f"Eroare la functia get_response: {str(e)}"
```

```
            return "Imi pare rau, dar nu am gasit un raspuns la intrebarea ta."

        else:
            # Pentru intrebari generale, se foloseste doar modelul LLM, fara context.
            messages = [
                {
                    "role": "system",
                    "content": (
                        "Cand oferi un raspuns, asigura-te ca este prezentat intr-un mod
placut intregul mesaj: cu aliniere, italic, bold si alte elemente de formatare adecvate
contextului. Stilul raspunsului ar trebui sa fie prietenos si primitor, pentru a face
utilizatorul sa se simta confortabil."
                        f"Datele personale ale utilizatorului sunt: {user_profile_data}\n"
                        "Te rog sa raspunzi in limba romana fara diacritice."
                        "Numele tau personal este „Alex” si esti asistentul virtual al
Facultatii de Inginerie Electrica si Stiinta Calculatoarelor. "
                        "Iti poti spune care e numele tau doar daca te intreba cineva sau la
mesaje de bun venit cand te prezinti. "
                        "Daca trebuie sa retii un lucru, vei raspunde cu un mesaj afirmativ si
vei mentiona clar ceea ce trebuie sa retii dar doar acel lucru."
                    ),
                }
            ]

            if conversation_history:
                for item in conversation_history:
                    messages.append({"role": "user", "content": item.question})
                    messages.append({"role": "assistant", "content": item.answer})

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

            try:
                response = ollama.chat(model="llama3.2", messages=messages)
                raspuns_text = response["message"]["content"].strip()
                raspuns_text = remove_diacritics(raspuns_text)

                return raspuns_text
            except Exception as e:
                return f"Eroare la functia get_response: {str(e)}"


def remove_diacritics(text):
    """
    Elimina diacriticele dintr-un text, folosind normalizarea Unicode.

    Parametru:
        text (str): Textul de procesat.

    Returneaza:
        str: Textul fara diacritice.
    """
```

```
nfkd_form = unicodedata.normalize("NFD", text)
only_ascii = "".join([c for c in nfkd_form if not unicodedata.combining(c)])

return only_ascii
```