



Universitatea Ștefan 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 .....	4
1. Introducere .....	5
1.1. Descrierea problemei.....	5
1.2. Justificarea temei.....	5
1.3. Situația actuală pe plan național.....	6
1.4. Situația actuală pe plan internațional.....	6
2. Tehnologii utilizate .....	8
2.1. Back-End.....	8
2.1.1. Django .....	8
2.1.2. Integrarea cu LLaMA 3.2 și RAG .....	9
2.2. Front-End.....	9
2.2.1. React + Vite.....	10
3. Considerații de proiectare .....	11
3.1. Proiectarea bazei de date .....	11
3.1.1. Tabele și relațiile dintre tabele .....	11
3.2. Proiectare Back-End .....	13
3.2.1. Funcționalități administrative.....	13
3.2.2. Funcționalități de securitate.....	13
3.2.3. Procesare întrebărilor și comenzilor.....	13
3.3. Proiectare Front-End .....	15
3.3.1. Paginile si componentele aplicației .....	15
4. Considerente de implementare .....	17
4.1. Implementarea Back-End.....	17
4.1.1. Organizarea fișierelor și directoarelor Back-End .....	18
4.1.2. Funcționalități și logica de procesare .....	19
4.1.3. Funcționalități administrative.....	20
4.1.4. Gestionarea conturilor de utilizator .....	21
4.2. Implementarea Front-End.....	22
4.2.1. Organizarea fișierelor și directoarelor Front-End .....	23
4.2.2. Descrierea funcțională a componentelor.....	24
4.2.3. Descrierea funcțională a paginilor.....	25
5. Manual de utilizare .....	27
5.1. Cerințe de sistem.....	27
5.2. Instalare.....	27

5.2.1. Configurare Back-End.....	27
5.2.2. Configurare Front-End.....	28
5.3. Utilizare .....	29
6. Rezultate experimentale .....	31
6.1. Testarea răspunsurilor la întrebări.....	31
6.1.1. Evaluarea răspunsurilor în funcție de numărul de fișiere disponibile .....	31
6.2. Testarea comenzilor .....	33
7. Concluzii și direcții viitoare de dezvoltare .....	35
7.1. Stadiul actual al proiectului.....	35
7.2. Direcții viitoare de dezvoltare.....	35
Bibliografie .....	37
Anexe 1 – Clasa ConversationChat (endpoint principal).....	38
Anexa 2 – Clasa RAG (Retrieval-Augmented Generation).....	41

## Listă de Figuri

Figură 3.1 – Schema bazei de date.....	11
Figură 3.2 – Schema de funcționare back-end.....	14
Figură 3.3 – Schema de funcționare front-end.....	16
Figură 4.1 – Structura back-end.....	17
Figură 4.2 – Structura front-end.....	23

# 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 proiectului 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 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 există Academia de Studii Economice din București, care a introdus „Saro”, un asistent 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 timpul 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 există 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 proiectele 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 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

Pentru partea de back-end am ales limbajul de programare Python, împreună cu framework-ul Django pentru următoarele motive:

- 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 cu 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, care 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 printr-un sistem de tip RAG (Retrieval-Augmented Generation) [14]. 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;
- Conținutul documentului găsit este extras și folosit drept context;
- Întrebarea și contextul sunt trimise către LLaMA 3.2, care generează un răspuns corespunzător întrebării;
- 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

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

- React este o bibliotecă JavaScript, cu o documentația bine detaliată, foarte popular în dezvoltarea aplicațiilor web, ea este bazată pe componente, ceea ce permite o organizare clară a codului și reutilizarea ușoară a lor fiind potrivită pentru aplicații web interactive [15];
- Vite este un instrument de build și servește la dezvoltarea aplicației web, este rapidă și simplă de utilizat, oferă o experiență ușoară în dezvoltare deoarece ea actualizează instant componentele [16].

### 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;

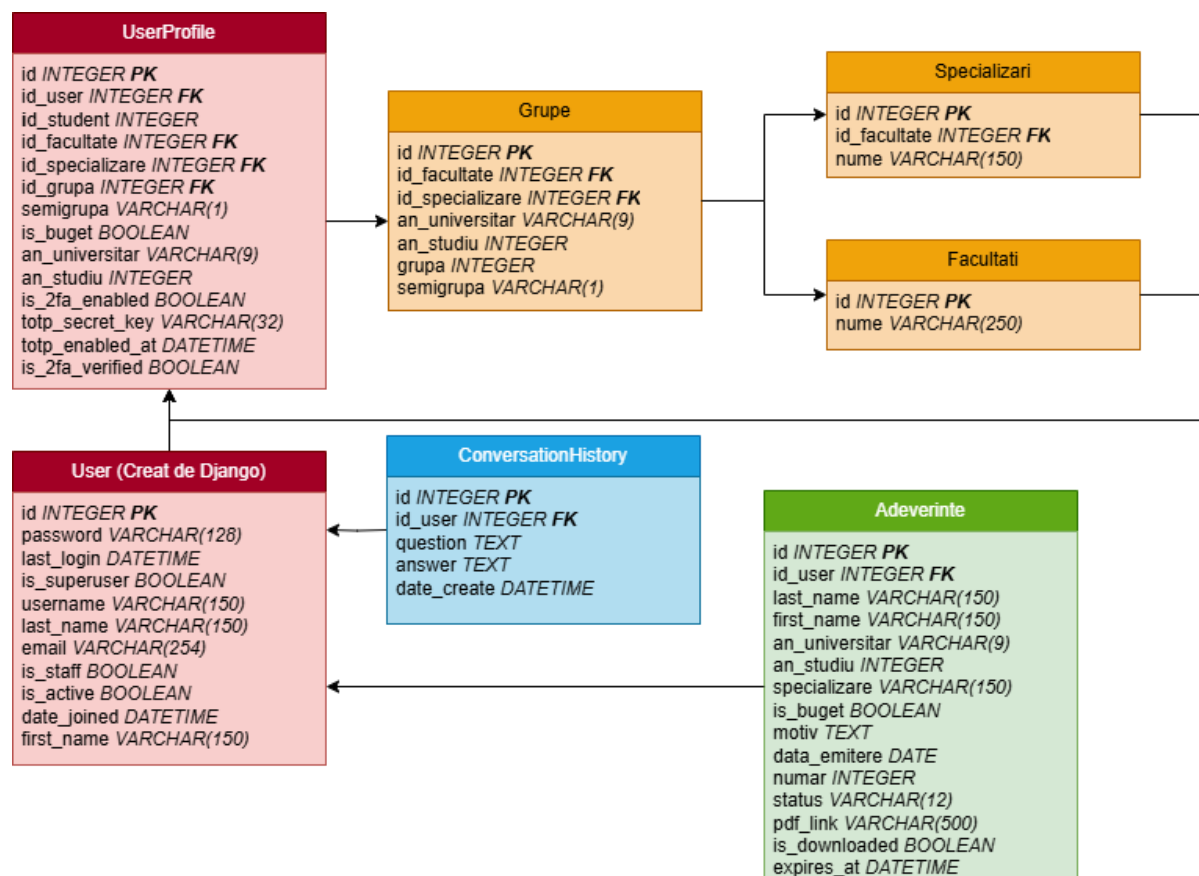
- Generează și afișează interfața grafică pe baza componentelor;
- Comunică cu serverul prin apeluri HTTP către API-ul RESET oferite de Django;
- Reacționează în timp real cu datele primite de la server, fără a fi nevoie de reîncărcarea paginii;
- Asigură o experiență ușoară și intuitivă pentru utilizator,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

În Figura 3.1 este prezentată schema bazei de date utilizate în cadrul aplicației, care descrie structura logică a tabelor 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. 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 adeverință cu motivul [motiv]” – informațiile aflate în baza de date corespunzătoare utilizatorului, sunt folosite pentru a genera o adeverință pe care o poți descărca și folosi imediat;
- Salvarea istoricului 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 să folosească un email de student aflat în baza de date.

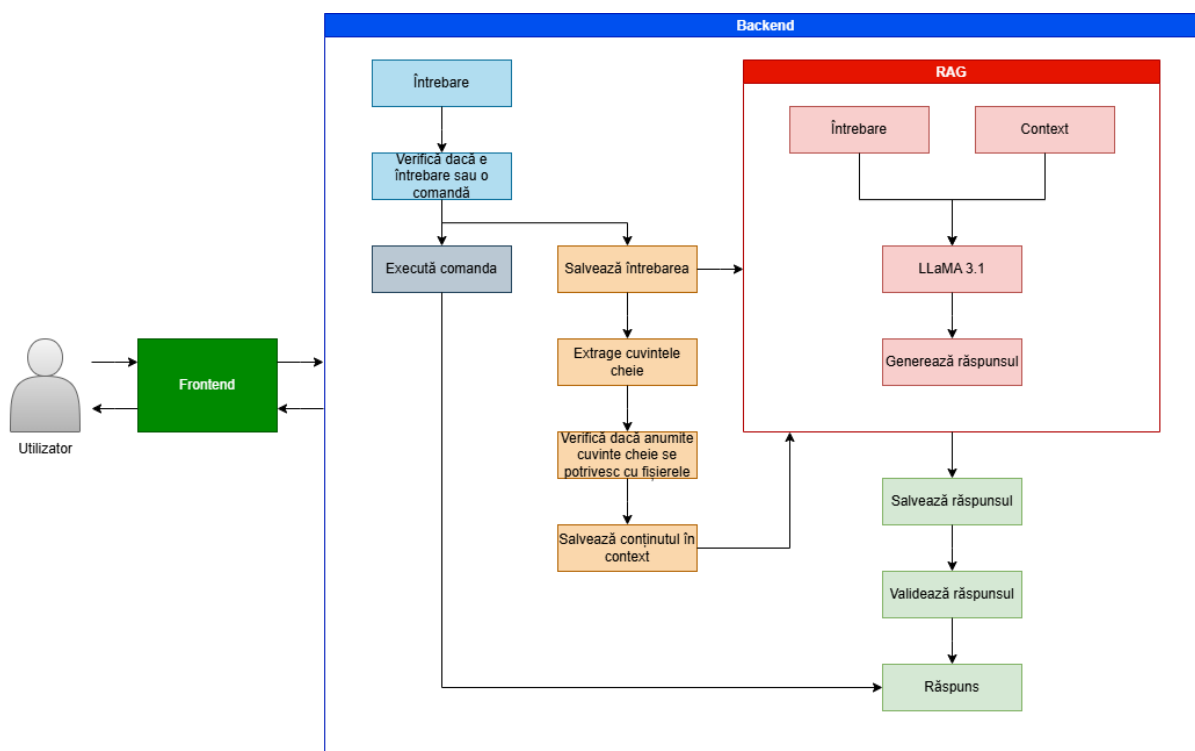
### **3.2.3. Procesare întrebărilor și comenzilor**

Când utilizatorul interacționează cu sistemul prin interfața de tip chat:

- Instrucțiune 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 și 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 adeverință 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 și componentele aplicației

Aplicația front-end este alcătuită din cinci pagini și două componente. Fiecare pagină contribuie la definirea structurii aplicației și a fluxului 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.

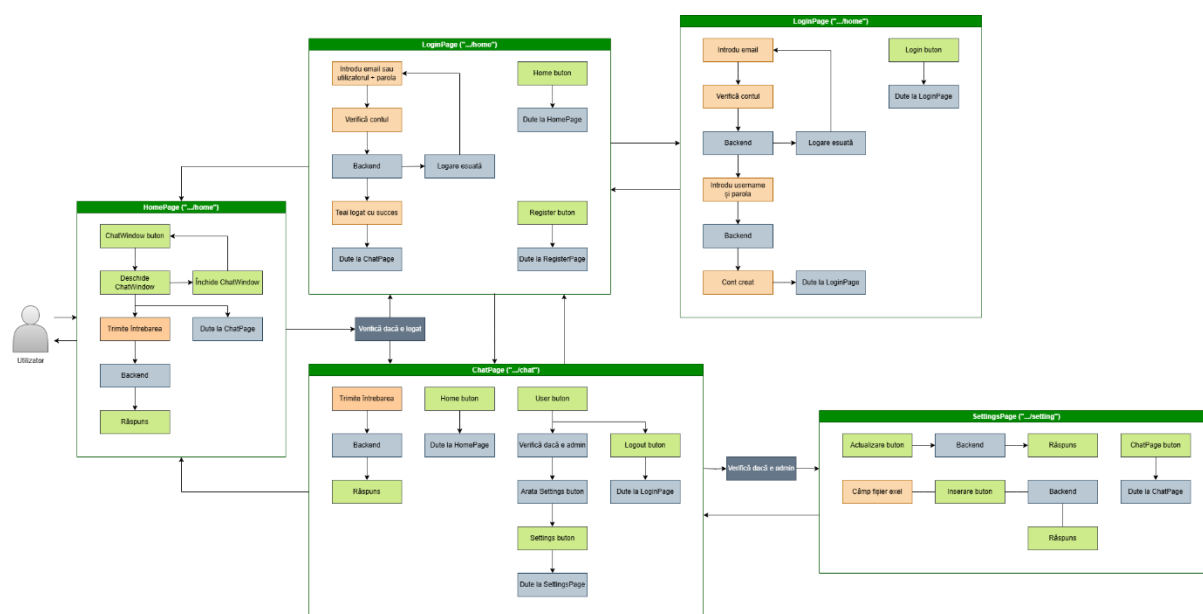
Pagini:

- 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? Crează 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 înregistrare, unde utilizatorii noi își pot crea un cont, cu condiția să aibă un email de student existent în 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 există 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 folosite 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

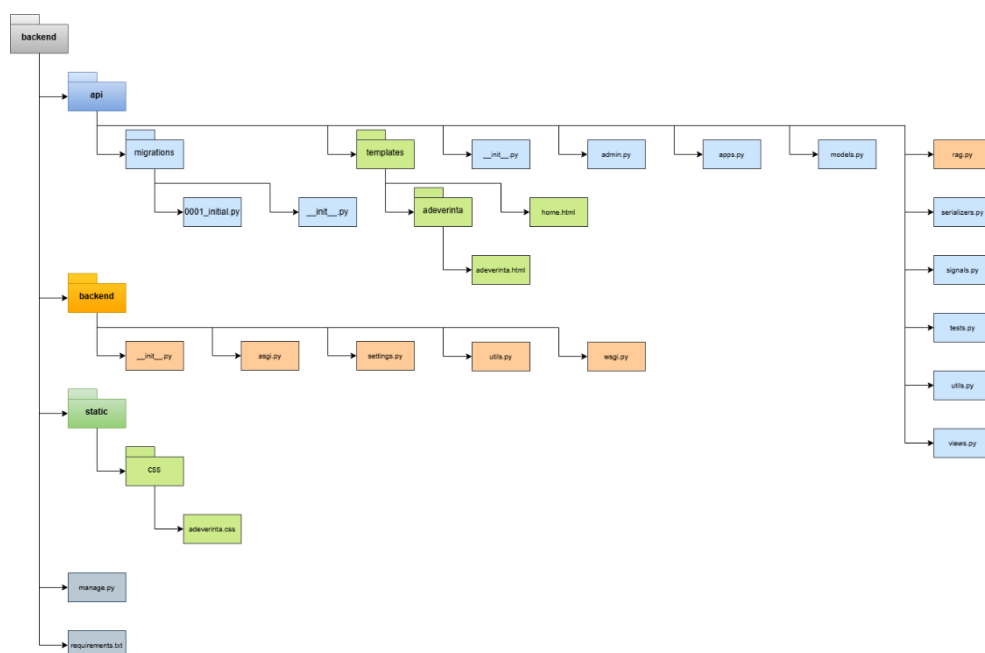
Componenta back-end a aplicației a fost realizată în limbajul de programare Python utilizând framework-ul Django, având rolul de a gestiona:

- Autentificarea și înregistrarea utilizatorilor;
- 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;
- Afășarea 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 și directoarele generate automat de Django, precum și cele adăugate manual pentru funcționalități personalizate.



Figură 4.1 – Structura back-end

#### 4.1.1. Organizarea fișierelor și directoroarelor 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 transmise 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 și testare;
- urls.py – definește rutele aplicației, legând adresele acestora de funcțiile din views.py.

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

- 0001\_initial.py conține instrucțiuni pentru crearea tabelor 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 două 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 și 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 adeverință 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 utilizatorului este salvată;
- Se extrag cuvintele din întrebare;
- Cuvinte sunt comparate cu numele fișierelor PDF;
- Conținutul documentelor găsite este extras și folosit drept context;
- Întrebarea și contextul sunt trimise către LLaMA 3.2, care generează un răspuns pentru întrebare bazată pe informațiile din context;
- Răspunsul este validat 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 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 introducere 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 corespundente 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 legate de gestionarea conturilor utilizatorilor în cadrul aplicației, respectiv autentificarea, verificarea existenței unui email și activarea unui cont.

#### 1. Autentificarea utilizatorilor

Pentru autentificarea utilizatorilor, endpoint-ul LoginView primește în request 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, 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 lor dintr-un fișier Excel), procesul de activare a contului se realizează astfel prin endpoint-ul `ActivateUserView`. Acesta primește email-ul, username-ul și parola 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.

## 4.2. Implementarea Front-End

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

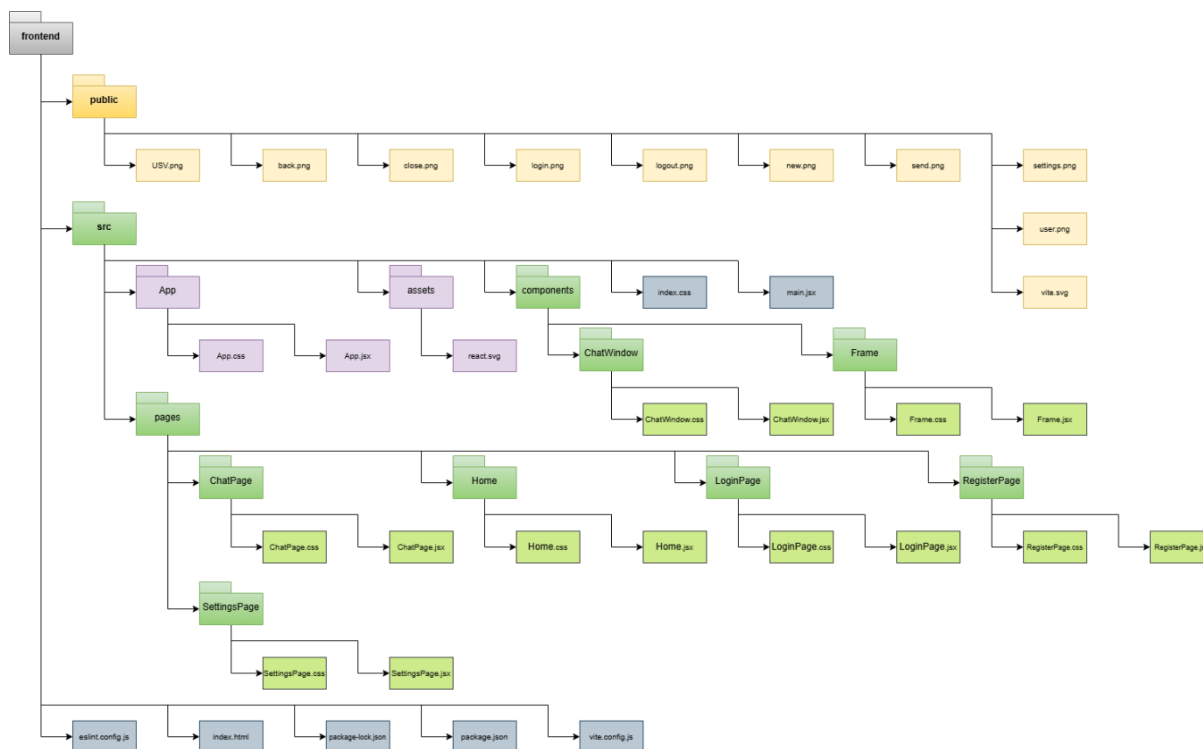
Această parte a aplicației are rolul de a furniza o interfață interactivă și intuitivă prin care utilizatorii pot interacționa cu asistentul virtual și cu funcționalitățile acesteia.

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 și 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 principală 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;
- /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 încarcă și afișa pagina oficială 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 doua 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 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 parolă. Utilizatorul își va introduce adresa de email de student unde la apăsarea butonului „Continuă” se trimite o cerere GET către endpointul 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 Login, în caz contrar este afișat un mesaj de eroare. Pagina Register 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 clona 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. Asigurați-vă 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 mediul virtual pentru proiect: *python -m venv .venv*

6. Activează mediul virtual: `.\.venv\Scripts\activate`
7. Instalează dependențele din fișierul `requirements.txt`: `pip install -r requirements.txt`
8. Creează migrațiile bazei de date: `python manage.py makemigrations`
9. Aplică migrațiile: `python manage.py migrate`
10. Instalează aplicația Ollama de pe site-ul oficial: <https://ollama.com/>
11. Descarcă modelul Llama 3.2 utilizat pentru a genera răspunsuri: `ollama pull llama3.2`
12. Rulează serverul Django: `python manage.py runserver`
13. Aplicația back-end va fi disponibilă pe: `http://127.0.0.1:2307/`

### 5.2.2. Configurare Front-End

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

1. Asigurați-vă 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 pe un browser și să introduci adresa URL „*http://127.0.0.1:2002/*”. Acolo va fi afișată o copie a site-ului oficial FIESC.

În partea dreaptă 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 am nevoie 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 pagină de chat. În partea dreaptă 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;
- Actualizarea documentelor folosite de asistentul virtual, prin apăsarea butonului „Actualizare”. Această operație trebuie executată înainte de a folosi aplicația pentru a descărca toate documentele necesare cu informații de pe site-ul oficial FIESC unde va trebui să aștepte aproximativ două minute (sau mai puțin în funcție de viteza de internet), după ce toate documentele au fost descărcate aplicația poate fi folosită 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 trece la următorul pas unde îți va cere să introduci un nume de utilizator și o parolă (parola trebuie să fie de minim 5 caractere, să conțină o literă mare, un caracter special și un număr), o dată ce contul a fost creat vei fi redirecționat către pagina de autentificare.

Avantajele unui cont de utilizator sunt următoarele:

1. Istoricul conversației este salvat și ajută utilizatorul să nu mai introducă date personale care au fost menționate anterior.
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 adevărilor.

### 6.1. Testarea răspunsurilor la întrebări

#### 6.1.1. Evaluarea răspunsurilor în funcție de numărul de fișiere disponibile

Vom analiza 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-au eliminat treptat câteva documente.

Întrebarea considerată pentru toate testele realizate: „Ce documente sunt necesare pentru bursa sociala ocazionala?”

Răspunsul așteptat la întrebare va fi:

- 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.

Testul 1:

Număr fișiere: 5

Mesaj primit: „Potrivit informativelor disponibile, pentru a solicita Bursă Socială Ocazională, aveți nevoie de următoarele documente: 1. **\*\*Căutare de bursă\*\***: Dețineți o scrisoarea de cerere de bursă socială ocazională, care să conțină informațiile despre dumneavoastră, despre proiectul sau activitatea pentru care doresc să obțineți bursă. 2. **\*\*Programele educaționale\*\***: Puteți solicita certificate de absolvire a programelor de studii (liceu, universitate, etc.) 3. **\*\*Cărți și documente\*\***: Puteți solicita cărți și documente necesare pentru proiectul sau activitatea pentru care doresc să obțineți bursă. 4. **\*\*Proiectul sau activitatea\*\***: Dețineți un plan de acțiune care descrie scopul, metodele de implementare, rezultatele și perioada de realizare a proiectului sau activității. 5. **\*\*Bilete de călătorie\*\***: Puteți solicita bilete de călătorie pentru vizite la evenimente culturale sau pentru participarea la concursuri științifice. 6. **\*\*Documente personale\*\***: Dețineți actele personale (passport, certificat de naștere, etc.), care să vă identifice și să confirme că sunteți cetățeanii României. 7. **\*\*Certificatele de absolvire\*\***: Puteți solicita certificatele de absolvire a programelor de studii, casei de comerț sau profesionale. Informația despre specificitatea documentelor care necesită pentru Bursa Socială Ocazională poate varia în funcție de organizație și de cerințe specifice ale bursiei.”

Concluzie: Răspunsul primit este gresit și nu conține lista exactă a documentelor necesare pentru bursa socială ocazională.

Testul 2:

Număr fișiere: 4

Mesaj primit: „În vederea acordării bursei sociale ocazionale, 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 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 De asemenea, documentele menționate anterior pot să conțină următoarele informații: \* Veniturile nete ale familiei \* Prezentarea evoluției și tabloul clinic al problemei medicale de care suferă studentul \* Informații despre



handicapul sau tulburările din spectrul autismului, dacă este cazul Documentele trebuie să fie complete, precise și autentice.”

Concluzie: Răspunsul este parțial corect și conține majoritatea documentelor necesare, dar include și informații suplimentare neprevăzute în răspunsul așteptat.

Testul 3:

Număr fișiere: 3

Mesaj primit: „Pentru a primi Bursa Socială Ocazională, urmează să depuneți 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. Afișat 04.10.2024 , ora 9.00”

Concluzie: Răspunsul este corect și complet, conținând toate documentele necesare pentru bursa socială ocazională, ceea ce indică că 3 fișiere sunt suficiente pentru un răspuns precis.

## 6.2. Testarea comenzilor

Aici este analizată funcționalitatea generării automate a adeverințelor în funcție dacă utilizatorul este autentificat sau nu. Scopul este de a verifica dacă asistentul generează corect o adeverință pentru utilizatorii autentificați și le transmite utilizatorilor care nu sunt autentificați „Doar persoanele autentificate pot genera o adeverinta!”.

Autentificat: NONE

Comandă: „Genereaza o adeverinta cu motivul angajare”

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

Concluzie: Răspunsul este corect. Sistemul refuză generarea adeverinței și afișează mesajul potrivit pentru utilizatorii neautentificați.

Autentificat: Șandru Alexandru

Comandă: „Genereaza o adeverinta cu motivul angajare”

Mesaj primit:

<b>UNIVERSITATEA "ȘTEFAN CEL MARE" DIN SUCEAVA</b>		
<b>FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR</b>		
		Nr. <u>2</u> /FIESC/ <u>20.06.2025</u>
<b>ADEVERINȚĂ</b>		
Studentul(a) <u>Sandru Alexandru</u> este înscris(ă) în anul universitar <u>2024/2025</u> în anul <u>4</u> de studii, program / domeniu de studii - licență:		
<input checked="" type="checkbox"/>	Calculatoare	
<input type="checkbox"/>	Automatică și Informatică Aplicată	
<input type="checkbox"/>	Inginerie Electronică, Telecomunicații și Tehnologii Informaționale / Rețele și Software de Telecomunicații	
<input type="checkbox"/>	Echipamente și Sisteme Medicale	
<input type="checkbox"/>	Sisteme Electrice	
<input type="checkbox"/>	Inginerie Energetică / Energetică și Tehnologii Informatică	
<input type="checkbox"/>	Echipamente și Sisteme de Comandă și Control pentru Autovehicule	
forma de învățământ IF, regim: <u>fara taxa</u>		
Adeverința se eliberează pentru a-i servi la angajare		
<b>DECAN,</b>	<b>SECRETAR ȘEF,</b>	<b>SECRETARIAT,</b>
Prof.univ.dr.ing. Laurențiu-Dan MILICI	Elena CURELARU	Laura DOSPINESCU Lucia POPESCU Otilia FRUNZĂ

Concluzie: Răspunsul este corect, a fost generată o adeverința conform datelor din baza de date, demonstrând funcționalitatea comenzii.

## 7. Concluzii și direcții viitoare de dezvoltare

### 7.1. Stadiul actual al proiectului

Baza de date:

- Completă, 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:

- Completă, posibil câteva retușuri.

### 7.2. 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 relevanță 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] „What is Retrieval-Augmented Generation (RAG) ?,” 10 Februarie 2025. [Interactiv]. Available: <https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/>.
- [15] „Top 10 Front-End Frameworks in 2025,” 13 Decembrie 2024. [Interactiv]. Available: <https://www.geeksforgeeks.org/blogs/top-front-end-frameworks/>.
- [16] „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 diferențindu-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)

            trigger = "genereaza o adeverinta cu motivul"
            if (
                request.user and request.user.is_authenticated and
                question.lower().startswith(trigger)
            ):
                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=request.user)
    except UserProfile.DoesNotExist:
        return Response(
            {"error": "Profilul utilizatorului nu a fost găsit."},
            status=status.HTTP_404_NOT_FOUND
        )

    adeverinta = Adeverinta.objects.create(
        user=request.user,
        last_name=request.user.last_name,
        first_name=request.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=Adeverinta.objects.count() + 1,
        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": timezone.now().date().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()

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

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

```

```

        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}"

        ConversationHistory.objects.create(
            user=request.user,
            question=question,
            answer=answer_text
        )

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

    if question.lower().startswith(trigger):
        return Response(
            {"answer": "Doar persoanele autentificate pot genera o adeverinta!"},
            status=status.HTTP_200_OK
        )

    conversation_history = None
    if request.user and request.user.is_authenticated:
        conversation_history = ConversationHistory.objects.filter(
            user=request.user
        ).order_by('id')[:10]

    answer = self.rag.get_response(question, conversation_history)

    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'):
        """
        Inițializează clasa RAG cu directorul pentru fișiere PDF.
        """
        self.pdf_directory = pdf_directory
        self.context = []

    def extract_words_from_filename(self, filename):
        """
        Extrage cuvintele din numele fișierului folosind separatorii specificați.
        """
        words = re.sub(r'[_]', ' ', filename)
        words = words.replace('.pdf', '').split()
        return words

    def read_pdf_content(self, pdf_path):
        """
        Citește conținutul unui fișier PDF.
        """
        try:
            reader = PdfReader(pdf_path)
            content = ''
            for page in reader.pages:
                content += page.extract_text()
            return content
        except Exception:
            return ''

    def find_matching_files(self, question):
        """
        Găsește fișierele care conțin cuvinte similare cu întrebarea.
        """
        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 and filename not in matching_files:
                        matching_files.append(filename)
```

```

    return matching_files

def get_context(self, question):
    """
    Generează contextul pentru întrebare din fișierele PDF.
    """
    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, conversation_history=None):
    """
    Obține răspunsul de la modelul LLM folosind contextul generat.
    """
    context = self.get_context(question)

    messages = [
        {
            "role": "system",
            "content": (
                "Te rog să răspunzi în limba română. Pe tine te cheamă „Alex” și ești  

                asistentul virtual al "  

                "Facultății de Inginerie Electrică și Știința Calculatoarelor. Nu trebuie  

                să îți spui numele de "  

                "fiecare dată, ci doar când ești întrebat. Dacă trebuie să reții un lucru,  

                vei răspunde cu un mesaj "  

                "afirmativ și vei menționa clar ceea ce trebuie să reții."
            )
        }
    ]

    if context:
        messages.append({
            "role": "system",
            "content": "Analizează următorul context și generează un răspuns scurt, direct  

            și la obiect pe baza întrebării: " + " ".join(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)
    return response['message']['content']
except Exception as e:
    return f"Eroare la generarea răspunsului: {str(e)}"
```