



Universitatea Ștefan cel Mare din Suceava

Facultatea de Inginerie Electrică și Știința Calculatoarelor

Domeniul: Calculatoare și Tehnologia Informației

Program de studii: Calculatoare

Alex - Asistent Virtual

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/internațional.....	6
2. Tehnologii utilizate	7
2.1. Back-End	7
2.1.1. Django	7
2.1.2. Integrarea cu LLaMA 3.2 și RAG	7
2.2. Front-End	8
2.2.1. React + Vite.....	8
3. Considerații de proiectare	9
3.1. Proiectarea bazei de date	9
3.1.1. Tabele și relațiile dintre tabele	9
3.2. Proiectare Back-End	11
3.2.1. Funcționalități administrative.....	11
3.2.2. Funcționalități de securitate.....	11
3.2.3. Procesare întrebărilor și comenzilor.....	11
3.3. Proiectare Front-End	12
3.3.1. Paginile si componentele aplicatiei	13
4. Considerente de implementare	14
4.1. Implementarea Back-End	14
4.1.1. Organizarea fișierelor și directoarelor Back-End	14
4.1.2. Funcționalități și logica de procesare	16
4.1.3. Funcționalități administrative.....	17
4.1.4. Gestionarea conturilor de utilizator	18
4.2. Implementarea Front-End	18
4.2.1. Organizarea fișierelor și directoarelor Front-End	19
4.2.2. Descrierea funcțională a componentelor	20
4.2.3. Descrierea funcțională a paginilor	21
5. Manual de utilizare	23
5.1. Cerințe de sistem.....	23
5.2. Instalare.....	23
5.2.1. Configurare Back-End.....	23

5.2.2. Configurare Front-End.....	24
5.3. Utilizare	24
6. Rezultate experimentale	26
6.1. Testarea răspunsurilor la întrebări.....	26
6.2. Testarea comenzilor	26
7. Concluzii și direcții viitoare de dezvoltare	27
7.1. Stadiul actual al proiectului.....	27
7.2. Direcții viitoare de dezvoltare.....	27
Bibliografie	28
Anexe 1 – Clasa ConversationChat (endpoint principal).....	29
Anexa 2 – Clasa RAG (Retrieval-Augmented Generation).....	32

Listă de Figuri

Figură 3.1 – Schema bazei de date.....	9
Figură 3.2 – Schema de funcționare back-end.....	12
Figură 3.3 – Schema de funcționare front-end.....	13
Figură 4.1 – Structura back-end.....	14
Figură 4.2 – Structura front-end.....	19

1. Introducere

Acest proiect reprezintă o nouă funcționalitate integrată în site-ul oficial al Facultății de Inginerie Electrică și Știința Calculatoarelor (FIESC), sub forma unui asistent virtual inteligent denumit Alex. Proiectul are obiectivul principal de a automatiza procesul de informare a studenților, reducând astfel interacțiunea directă cu secretariatul și timpul pierdut în căutarea informației pe site.

Alex este conceput pentru a răspunde întrebărilor frecvente ale studenților, pentru a genera documente oficiale (precum adeverințe), și pentru a oferi sprijin în completarea anumitor formulare sau documente disponibile pe site-ul facultății.

Interacțiunea cu Alex are loc printr-o interfață de tip chat, integrată direct în site-ul facultății. După autentificare, utilizatorul beneficiază de funcționalități suplimentare, precum posibilitatea de a genera adeverințe, păstrarea istoricului conversațiilor și pusul de întrebări mai ușor, deoarece nu mai este necesară reintroducerea constantă a datelor personale, iar răspunsurile sunt mai relevante datorită contextului conversațiilor anterioare.

Acest proiect demonstrează cum 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 se confruntă frecvent cu dificultăți în obținerea de documente necesare sau a informațiilor precum cele de bursă, taxe, sau grupa în care face parte, 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 depune sau solicita o simplă adeverință. În plus, multe dintre întrebările adresate secretariatului au deja răspunsuri disponibile pe site-ul facultății, însă acestea pot fi greu de găsit din cauza volumului mare de informații. Prin urmare, era necesară o soluție care să fie mai eficientă în accesul la informații și să reducă interacțiunile fizice inutile.

1.2. Justificarea temei

Studenții se confruntă adesea cu dificultăți atunci când trebuie să obțină informații sau documente de la secretariat, mai ales din cauza programului scurt de lucru și a cozilor lungi. În plus, multe dintre întrebările lor au deja răspunsuri pe site-ul facultății, dar sunt greu de găsit.

Alex a fost creat pentru a rezolva aceste probleme. Acesta ajută studenții să găsească rapid informațiile de care au nevoie, cum ar fi detalii despre taxe, burse sau grupa din care fac

parte. De asemenea, studenții pot genera documente oficiale, cum ar fi adeverințe, fără a fi nevoie să aștepte la cozi sau să interacționeze fizic cu secretariatul.

Astfel, proiectul își propune să economisească timp și să simplifice procesul de obținere a informațiilor și documentelor, oferind o soluție rapidă și accesibilă pentru studenți. Totodată, contribuie la digitalizarea activităților administrative, reducând interacțiunile fizice și îmbunătățind eficiența întregului sistem.

1.3. Situația actuală pe plan național/internațional

În România, implementarea asistenților virtuali în mediul educațional este încă la început, dar câteva instituții au început să adopte astfel de soluții pentru a sprijini studenții și a îmbunătăți accesul la informații și servicii administrative.

Un exemplu relevant este Academia de Studii Economice (ASE) din București, care a lansat asistentul virtual „Saro”, un sistem bazat pe inteligență artificială ce ajută studenții cu informații academice și administrative. Acest proiect a fost realizat cu ajutorul unui grant de 700.000 de dolari din partea Google.org, având scopul de a sprijini digitalizarea activităților administrative ale universității.

Această inițiativă demonstrează tendința de integrare a inteligenței artificiale în educație, chiar dacă utilizarea asistenților virtuali în România este încă la început. Proiectele existente au ca scop îmbunătățirea accesului la informații și reducerea timpului pierdut de studenți în procesele administrative.

2. Tehnologii utilizate

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

2.1. Back-End

2.1.1. Django

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

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

2.1.2. Integrarea cu LLaMA 3.2 și RAG

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

- a. Când utilizatorul trimite o întrebare, aceasta este salvată și analizată.
- b. Se extrag cuvintele-cheie (de exemplu „criterii”, „bursă” etc.), ignorând cuvintele comune.
- c. Aceste cuvinte-cheie sunt folosite pentru a căuta documente relevante care conțin informațiile necesare.
- d. Conținutul documentului găsit este extras și folosit drept context.
- e. Întrebarea și contextul sunt trimise către LLaMA 3.2, care generează un răspuns adaptat.
- f. Răspunsul este apoi validat automat și trimis către utilizator.

Dacă întrebarea este o comandă, cum ar fi „Generează o adeverință”, sistemul:

- verifică dacă utilizatorul este logat,

- încarcă șablonul PDF,
- completează automat datele personale,
- generează fișierul și oferă un link de descărcare.

2.2. Front-End

2.2.1. React + Vite

Interfața aplicației este dezvoltată în React, folosind Vite pentru o încărcare rapidă și un flux de lucru modern. Utilizatorul interacționează astfel:

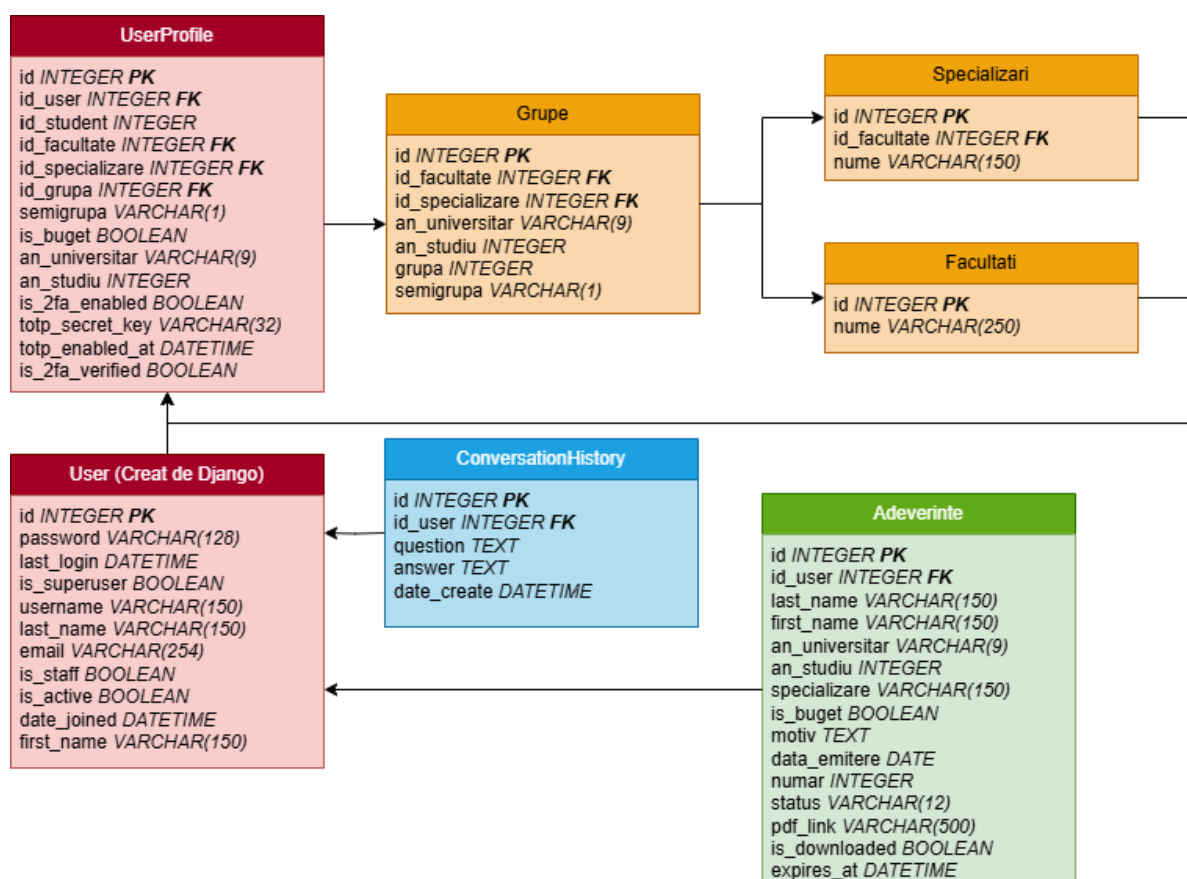
- La accesarea site-ului, apare o versiune a paginii oficiale a FIESC.
- În colțul din dreapta jos există un buton „Alex – Asistent Virtual”.
- La apăsarea butonului, se deschide o fereastră de tip chat, unde utilizatorul poate pune întrebări.
- Dacă utilizatorul se autentifică, are acces la funcții suplimentare:
 - Poate genera documente (de exemplu adeverințe),
 - Are acces la istoricul conversațiilor,
 - Întrebările sunt mai ușor de pus, pentru ca Alex reține contextul anterior.

3. Considerații de proiectare

3.1. Proiectarea bazei de date

Baza de date este implementată în SQLite, un sistem de gestiune a bazelor de date relaționale, integrat cu Django. Aceasta conține mai multe tabele care stochează informațiile necesare pentru funcționarea aplicației. Structura este construită astfel încât să permită extinderea ușoară a sistemului și menținerea unei organizări clare a datelor.

În Figura 3.1 este prezentat schema bazei de date utilizate în cadrul aplicației, care descrie structura logică 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 întrebărilor și răspunsurilor oferite de 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 și o serie de endpointuri pentru activități administrative, accesibile doar de către conturile cu rol de administrator. Pentru început ei au posibilitatea de a adăuga utilizatori noi în baza de date folosind un fișier excel. Aceștia mai au posibilitatea de actualiza la apăsarea unui buton, toate documentele esențiale utilizate de modelul LLaMA 3.2 în cadrul mecanismului RAG, pentru a putea furniza 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” care îți afișează toate comenzile valabile, și „Generează o adeverință cu motivul [motiv]” care pe baza informațiilor aflate în baza de date îți va genera o adeverință pe care o poți descărca și folosi imediat.
- Salvarea istoricului conversației pentru a face conversația cu LLaMA 3.2 mai ușoară pentru că acesta are acces la ceea ce sa vorbit anterior și nu mai trebuie ca utilizatorul să repete anumite lucruri la fiecare întrebare.
- Pentru utilizatorii care își doresc să își creeze un cont trebuie sa folosească un email de student care se află în baza de date.

3.2.3. Procesare întrebărilor și comenzilor

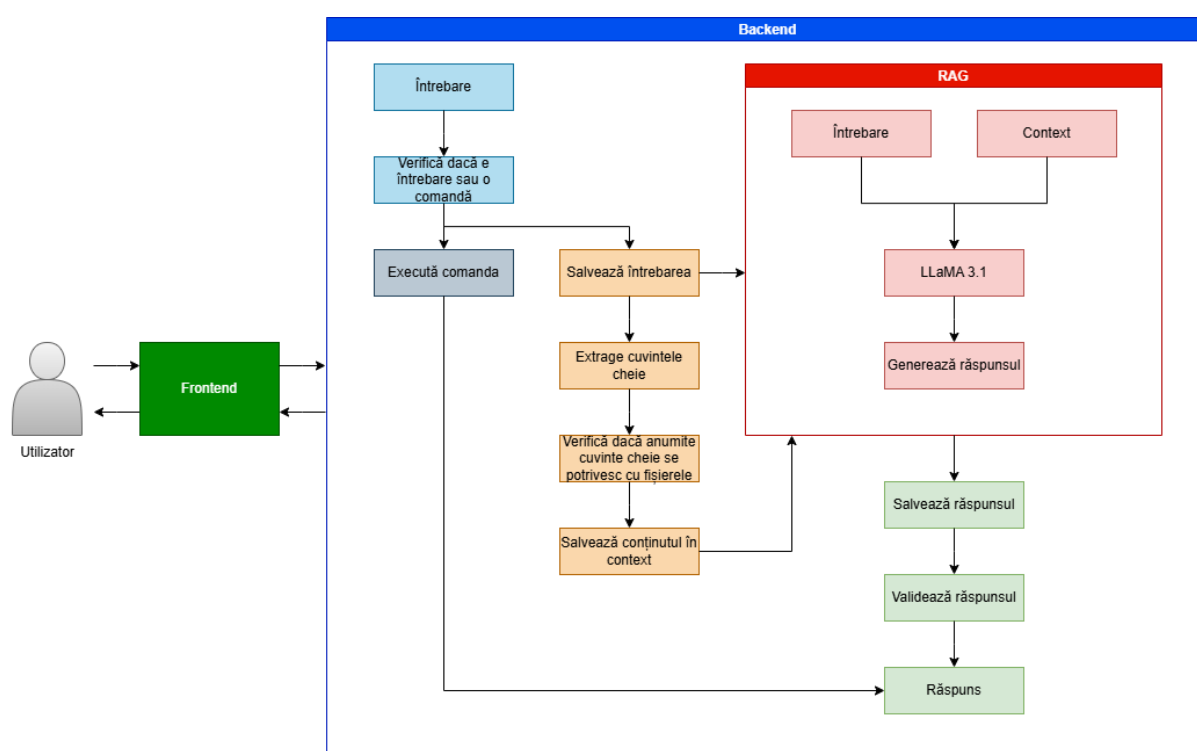
Când utilizatorul interacționează cu sistemul prin interfața de tip chat:

- Întrebarea este trimisă prin front-end către back-end.
- Back-end verifică dacă este întrebare sau comandă.
- Dacă este întrebare, se salvează, se extrag cuvintele cheie, acestea se verifică dacă se potrivesc cu un cuvânt aflat în numele unui document, și conținutul acestuia este

salvat și trimis la modulul de RAG, care generează un răspuns, acesta este validat și returnat către utilizator.

- Dacă este 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 cu o adeverință completată automat pe baza informației corespunzătoare utilizatorului autentificat.

În Figura 3.2 este prezentat 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. Acesta este responsabil de interacțiunea directă cu utilizatorul, afișarea datelor preluate de la back-end și transmiterea cererilor către server.

3.3.1. Paginile si componentele aplicatiei

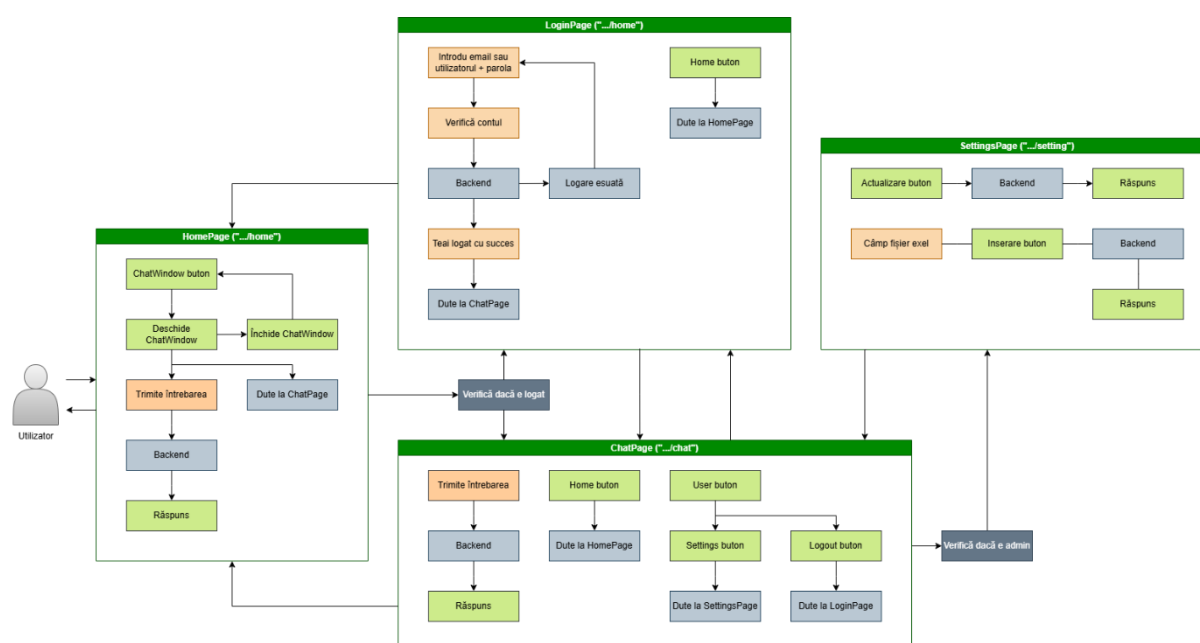
Aplicația front-end este alcătuită din patru pagini principale si o componentă importantă. Fiecare pagină contribuie la definirea structurii aplicației și a fluxul de utilizare, având un rol important în interacțiunea utilizatorului.

Componenta „ChatWindow” funcționează ca punct de acces principal către funcționalitatea aplicației și este integrată în pagina principală, care reproduce vizual site-ul oficial al facultății prin intermediul unui „iframe”. Această componentă este un chat rapid si ușor de accesat pentru întrebări scurte si rapide adresate asistentului virtual.

Din această componentă utilizatorul poate să se autentifice redirecționându-l către „LoginPage”, din această pagină utilizatorul își poate crea un cont redirecționându-l către „RegisterPage”, dar pentru acest lucru el trebuie să fie în posesia unui cont de student care se află în baza de date. Dacă contul e valid e automat redirecționat către „ChatPage” acesta este un chat mai mare în care îi este afișat istoricul conversației.

Dacă utilizatorul este admin atunci acesta poate intra pe pagina de setări redirecționându-l către „SettingPage” în care va avea acces la un buton de update a documentelor esențiale asistentului virtual pentru a genera răspunsuri corecte, și poate insera noi utilizatori în baza de date folosind un fișier excel.

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



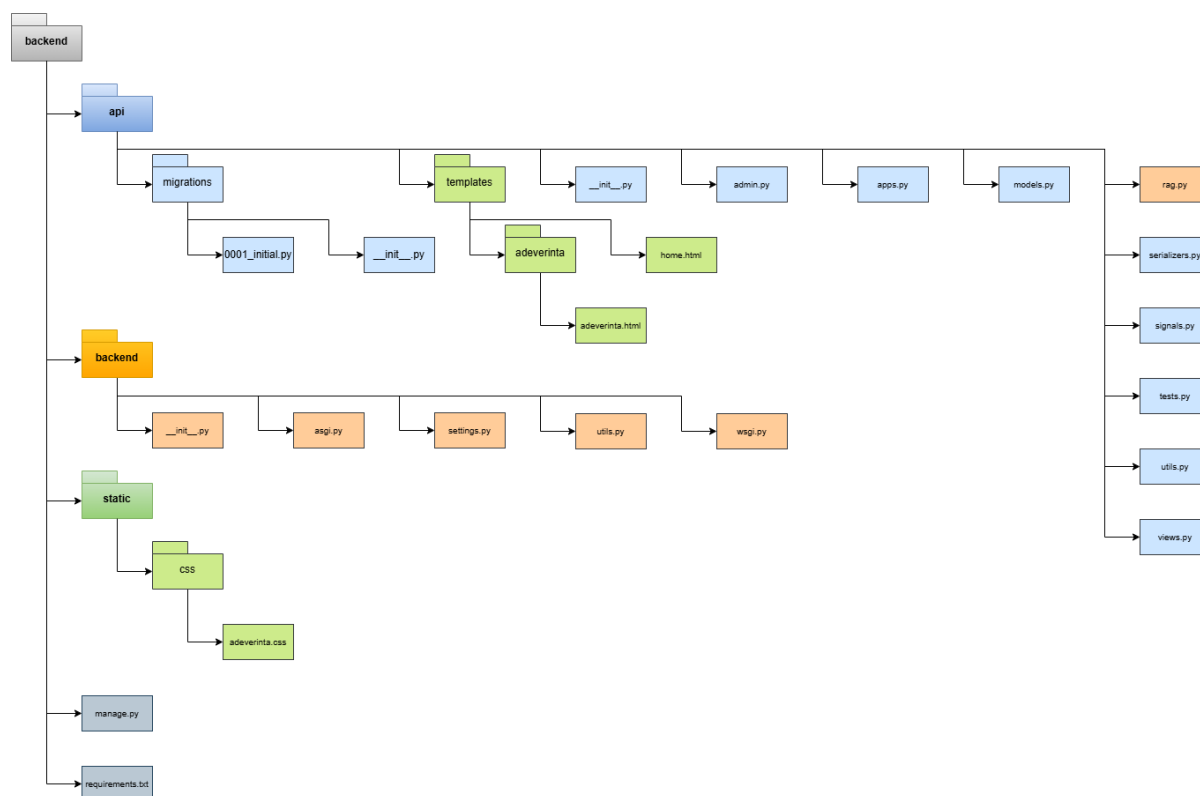
Figură 3.3 – Schema de funcționare front-end

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, stocarea, și procesarea datelor, integrarea cu modelul LLaMA și logica de generare a răspunsurilor. În plus, el oferă funcționalități dedicate utilizatorilor autentificați, cum ar fi afișarea comenzilor disponibile și generarea automată a unei adeverințe în format PDF pe baza datelor din baza de date. Utilizatorii își pot crea conturi, iar istoricul conversației este păstrat pentru un dialog mai fluid cu asistentul virtual.

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



Figură 4.1 – Structura back-end

4.1.1. Organizarea fișierelor și directoarelor Back-End

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

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

Directorul „./api/migrations” este generat automat de Django. Fișierul 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 adeverinței PDF, care este completat dinamic cu datele utilizatorului

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

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

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

Directorul sursă „./backend/” conține două fișiere principale:

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

4.1.2. Funcționalități și logica de procesare

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

1. Procesare întrebărilor primite

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

- Dacă mesajul este „comenzi”, se returnează o listă cu comenzile disponibile, utile pentru ghidarea utilizatorului.
- Dacă mesajul începe cu „genereaza o adeverinta cu motivul”, aplicația declanșează procesul de creare a unei adeverințe 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 cele mai importante funcționalități ale aplicației. După ce este recunoscută o comandă de tipul „Genereaza o adeverinta cu motivul angajare” sunt parcurși pașii următori:

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

3. Generarea de răspunsuri personalizate cu RAG + LLaMA

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

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

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

4. Alte funcționalități

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

4.1.3. Funcționalități administrative

Aplicația oferă două funcționalități esențiale rezervate administratorilor:

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

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

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

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

2. Importul de utilizatori dintr-un fișier Excel

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

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

- Se creează sau actualizează contul de User si UserProfile

Funcționalitatea asigură o introducere rapidă și eficientă a datelor.

4.1.4. Gestionarea conturilor de utilizator

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

1. Autentificarea utilizatorilor

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

2. Verificarea existenței unui email

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

3. Activarea conturilor

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

4.2. Implementarea Front-End

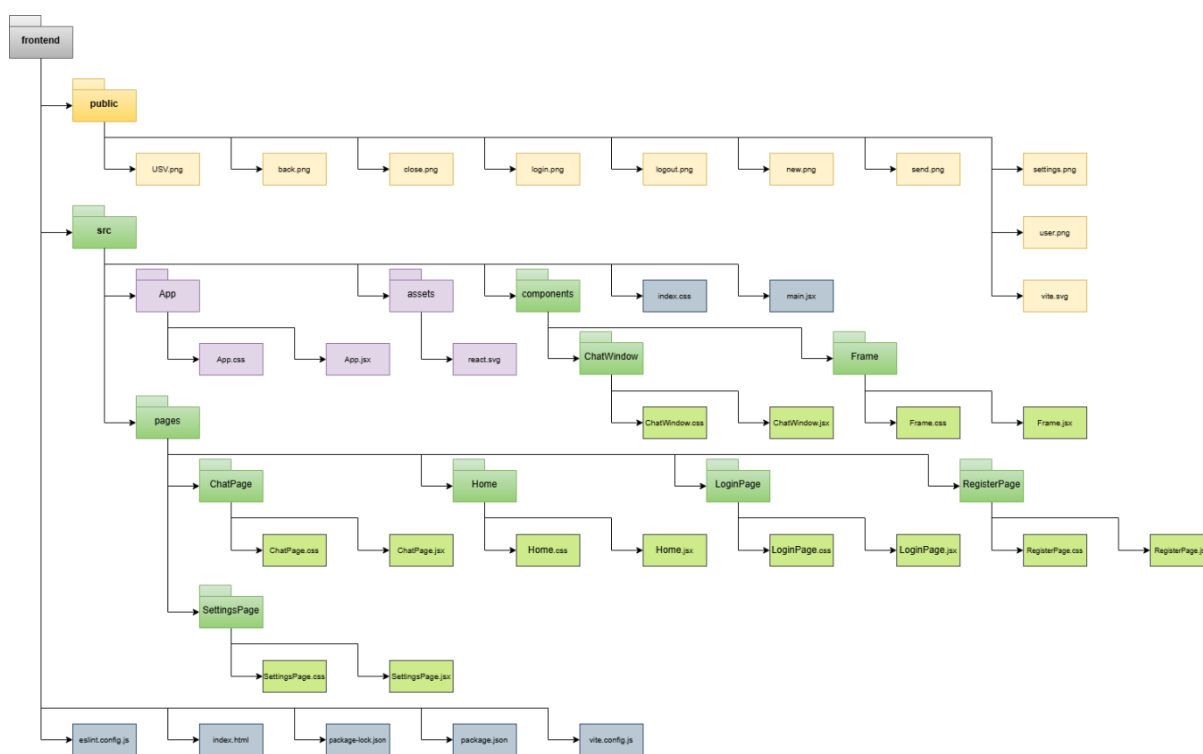
Componenta front-end a aplicației a fost dezvoltată utilizând biblioteca React, împreună cu instrumentul de build Vite, pentru o experiență de dezvoltare mai rapidă și eficientă. Această componentă are rolul de a furniza o interfață interactivă și intuitivă prin care utilizatorii pot interacționa cu asistentul virtual și cu funcționalitățile aplicației.

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

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

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

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



Figură 4.2 – Structura front-end

4.2.1. Organizarea fișierelor și directoarelor Front-End

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

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

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

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

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

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

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

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

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

Directorul „/components/” conține componentele React reutilizabile:

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

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

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

4.2.2. Descrierea funcțională a componentelor

Componenta Frame este o componentă simplă React care afișează un iframe ce încarcă pagina principală a site-ului FIESC. Această componentă are rolul de a încarca și 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 backend, solicitând generarea unui mesaj personalizat de introducere. Mesajul primit este afișat în fereastra de chat, iar utilizatorul poate începe dialogul. Utilizatorul poate introduce mesaje în câmpul de text și le poate trimite fie prin apăsarea butonului, fie prin apăsarea tastei enter. Pe durata generării răspunsului, în interfață este afișat un mesaj de tip „Se generează răspunsul...”. Fereastra de chat poate fi închisă și redeschisă, iar scroll-ul la ultimul mesaj este gestionat automat. Există un buton în dreapta sus pentru navigarea către o pagină de chat extinsă (sub forma unui +).

4.2.3. Descrierea funcțională a paginilor

Pagina Home reprezintă punctul principal de intrare în aplicație, ea este compusă din 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 apăsarea tastei enter, datele introduse sunt validate local. Dacă datele sunt valide, se face o cerere POST către endpointul de autentificare. Dacă autentificarea reușește, răspunsul conține un token JWT și informații despre utilizator care apoi sunt salvate în localStorage și utilizatorul este redirecționat către pagina de chat. În această pagina 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 dacă utilizatorul îl apasă este redirecționat către pagina Register.

Pagina Register permite crearea unui cont nou pentru utilizatorii care nu dețin unul. Procesul este împărțit în două etape pentru a verifica validarea emailului de student, și de a-i acorda un username și parola. 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 să conțină minim 5 caractere, o literă mare, o cifră și un caracter special. Dacă validările trec este trimisă o cerere POST către endpointul ActivateUserView cu datele introduse, dacă înregistrarea este reușită utilizatorul este redirecționat către pagina de login, în caz contrar este afișat un mesaj de eroare. Această pagină mai conține și un buton de back în partea din stânga sus a paginii.

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

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

5. Manual de utilizare

5.1. Cerințe de sistem

Cerințe minime:

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

Cerințe recomandate:

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

5.2. Instalare

Clonează repository-ul folosind comanda:

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

5.2.1. Configurare Back-End

Pașii pentru configurarea back-end:

1. Pentru a rula aplicația, trebuie să ai instalat:
 - Python (v3.13.2 sau o versiune mai mare)
 - pip (v24.3.1 sau o versiune mai mare)
2. Asigură-te că ai instalat gtk3-runtime de pe <https://github.com/tschoonj/GTK-for-Windows-Runtime-Environment-Installer/releases>
3. Asigură-te că te afli în directorul backend
4. Creează mediul virtual folosind comanda: *python -m venv .venv*
5. Activează mediul virtual folosind comanda: *.\.venv\Scripts\activate*
6. Instalează dependențele folosind comanda: *pip install -r requirements.txt*
7. Crează migrațiile folosind comanda: *python manage.py makemigrations*
8. Aplică migrațiile folosind comanda: *python manage.py migrate*
9. Instalează Ollama mergând pe site-ul oficial Ollama.
10. Instalează modelul Llama 3.2 folosind comanda: *ollama pull llama3.2*

11. Rulează proiectul folosind comanda: *python manage.py runserver*

12. Proiectul rulează pe: *http://127.0.0.1:2307/*

5.2.2. Configurare Front-End

Pașii pentru configurare front-end:

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

- Node.js (v22.13.1 sau o versiune mai mare)
- npm (v10.9.2 sau o versiune mai mare)

2. Asigură-te că te afli în directorul frontend

3. Creează un fișier .env și adaugă următoarea linie:

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

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

4. Instalează dependențele folosind comanda: *npm install*

5. Rulează proiectul folosind comanda: *npm run dev*

6. Proiectul rulează pe: *http://127.0.0.1:2002/*

5.3. Utilizare

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

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

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

„Actualizare” pe care trebuie apăsat înainte de a fi folosită aplicația pentru a descărca toate documentele necesare cu informații de pe site-ul oficial FIESC unde va trebui să aștepte aproximativ 5 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.

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

Pentru a putea să 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. Acolo pentru a putea crea unul trebuie să folosești un email de student care a fost introdus în baza de date de către administrator. Dacă acea adresă de email există 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 asta înseamnă că face discuția cu „Alex” mai ușor pentru că el știe despre ce s-a vorbit anterior și utilizatorul nu mai trebuie să repete sau să spună informații personale pentru că acestea se află deja în baza de date.
2. Utilizatorul are acces la 2 comenzi:
 - „comenzi” – această comandă va returna toate comenzile pe care utilizatorul le poate folosi;
 - „generează o adeverință cu motivul [motiv]” – aceasta comandă va genera un fișier PDF care va conține o adeverință completată automat cu date din baza de date specifice utilizatorului care a folosit comanda pe baza unui șablon, și va returna un link de unde se poate descărca fișierul.

6. Rezultate experimentale

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

6.1. Testarea răspunsurilor la întrebări

Număr fișiere:

Întrebare:

Mesaj primit:

Mesaj așteptat:

Concluzie:

6.2. Testarea comenzilor

Autentificat:

Comandă:

Mesaj primit:

Concluzie:

7. Concluzii și direcții viitoare de dezvoltare

7.1. Stadiul actual al proiectului

Baza de date:

- Complet, posibil câteva retușuri.

Back-End:

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

Front-End:

- Complet, posibil câteva retușuri.

7.2. Direcții viitoare de dezvoltare

Îmbunătățirea sistemul de RAG:

- Introducerea unei etape avansate de filtrare a documentelor.
- Limitarea documentelor trimise la model, preia doar documentele care au 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.

Adăugarea unei funcționalități de feedback:

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

Extinderea interfeței administrative:

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

Schimbarea LLM-ului:

- Înlocuirea modelului local cu un LLM mai performant.

Autentificare mai sigura:

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

Testare automată:

- Introducerea testelor pentru backend.

Bibliografie

<https://ase.ro/>
<https://www.djangoproject.com/>
<https://www.drawio.com/>
<https://www.geeksforgeeks.org/large-language-model-llm/>
<https://www.geeksforgeeks.org/what-is-retrieval-augmented-generation-rag/>
<https://newsbucuresti.ro/ase-bucuresti-prima-universitate-din-romania-care-introduce-un-asistent-ai-pentru-studenti/>
<https://ollama.com/>
<https://react.dev/>
<https://sqlite.org/index.html>
https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2/

Anexe 1 – Clasa ConversationChat (endpoint principal)

Această anexă conține codul complet al clasei „ConversationChat”, care gestionează cererile POST. Clasa verifică mesajul primit 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
                )

            if (
                request.user and request.user.is_authenticated and
                question.lower().startswith("genereaza o adeverinta cu motivul")
            ):
                user = request.user
                trigger = "genereaza o adeverinta cu motivul"
                motiv = question[len(trigger):].strip()

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

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

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

```

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

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

context = {
    "numar": adeverinta.numar,
    "data": data_emitere.strftime("%d.%m.%Y"),
    "nume": adeverinta.last_name,
    "prenume": adeverinta.first_name,
    "anUniversitar": adeverinta.an_universitar,
    "anul": adeverinta.an_studiu,
    "specializare": adeverinta.specializare,
    "regim": adeverinta.tip_taxa,
    "motiv1": motiv1,
    "motiv2": motiv2,
}

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

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

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

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

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

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

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

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

```

```
        )

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

    answer = self.rag.get_response(question)

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

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

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

Anexa 2 – Clasa RAG (Retrieval-Augmented Generation)

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

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

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

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

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

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

        return matching_files

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

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

        return self.context

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

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



```
]

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

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

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