

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

**Sistem de recomandare pentru rezervări online**

## **PROIECT DE DIPLOMĂ**

prezentat ca cerință parțială pentru obținerea titlului de Inginer  
în domeniul Calculatoare și Tehnologia Informației  
programul de studii de licență Ingineria Informației

**Coordonatori științifici:**

Ș.l. Dr. Ing. STOICA Elena Cristina

Ș.l. Dr. Ing. STOICA George Valentin

**Absolvent:**

DEAC Dragoș-Ștefan

**București**

2024



Universitatea "Politehnica" din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației  
Program de studiu **INF**

**Anexa 1**

**TEMA PROIECTULUI DE DIPLOMĂ**  
a studentului **DEAC V. Dragoș - Ștefan, 444A**

**1. Titlul temei:** Sistem de recomandare pentru rezervări online

**2. Descrierea temei și a contribuției personale a studentului (în afara părții de documentare):**

Proiectul își propune realizarea unui sistem de căutare și recomandare a locuințelor de închiriat/locurilor de cazare în funcție de preferințele utilizatorilor. Contribuția studentului va consta în studiul, propunerea și implementarea de soluții pentru un sistem de recomandare bazat pe algoritmi pe rating, cuvinte cheie, similitudine, geolocație. Implementarea practică se va realiza sub forma unei aplicații Web pentru rezervarea online a unei locuințe, având următoarele funcționalități și module:

- modul proprietari pentru gestionarea locuințelor de închiriat
- modul utilizatori pentru înrolare persoane, căutare și recomandare locuințe, închiriere, feedback
- modul de recomandare locuință în funcție de profilul utilizatorilor, de proprietățile locuinței și de ratingurile utilizatorilor
- modul hărți și geolocație pentru afișarea locuințelor disponibile să căutăți în funcție de zonă.

Pentru realizarea aplicației se va implementa o arhitectura multinivel, astfel:

- interfața utilizator
- componenta server pentru implementarea logicii de business
- persistența datelor va fi asigurată de sistemul de gestiune baze de date mongoDB

Se va măsura și analiza impactul volumului de date și a numărului de utilizatori simultani asupra performanței soluțiilor implementate folosind teste de stress.

**3. Discipline necesare pt. proiect:**

Tehnici de Programare în Internet, Inginerie Software, Programare Distribuită

**4. Data înregistrării temei:** 2023-12-08 08:10:43

**Conducător(i) lucrare,**  
Ș.L.Dr.Ing. Elena Cristina STOICA

**Student,**  
DEAC V. Dragoș - Ștefan

Ș.I. Dr. Ing. Stoica George Valentin

**Director departament,**  
Ș.L. dr. ing Bogdan FLOREA

**Decan,**  
Prof. dr. ing. Mihnea UDREA

Cod Validare: **de4bdb28a3**



### **Declarație de onestitate academică**

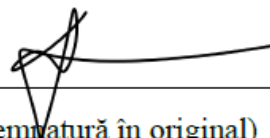
Prin prezenta declar că lucrarea cu titlul „Sistem de recomandare pentru rezervări online”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității „Politehnica” din București ca cerință parțială pentru obținerea titlului de Inginer în domeniul Electronică, Telecomunicații și Tehnologia Informației, programul de studii Ingineria Informației este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

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

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

București, 05.09.2024

Absolvent Dragoș-Ștefan DEAC

  
(semnătură în original)



## Cuprins

1. Introducere .....	15
1.1 Context și Motivație .....	15
1.2 Prezentarea Problemei .....	16
1.3 Obiective .....	16
1.4 Soluția Propusă .....	17
1.5 Rezultate .....	17
1.6 Structură și observații .....	18
2. Contextualizarea Proiectului .....	19
2.1 Baza Științifică a Turismului .....	19
2.2 Evaluarea Soluțiilor Existente în Turism .....	20
3. Tehnologii și Arhitecturi .....	23
3.1 Infrastructura Tehnică .....	23
3.2 Arhitectura Propusă .....	25
3.2.1 Stratul de interfață cu utilizatorul .....	26
3.2.2 Stratul de logică .....	26
3.2.3 Stratul de persistență a datelor .....	27
4. Proiectare și Modelare .....	29
4.1 Logică și Design al Componentelor .....	29
4.2 Modelarea Bazei de Date .....	30
5. Implementare și Dezvoltare .....	33
5.1 Dezvoltare Frontend .....	34
5.1.1 Pagina de autentificare .....	36
5.1.2 Pagina profilului de utilizator tip „Renter” .....	37
5.1.3 Pagina pentru recomandarea de hoteluri .....	38
5.1.4 Pagina profilului de utilizator tip „Landlord” .....	39
5.1.5 Pagina pentru listat unități de cazare .....	41
5.2 Dezvoltare Backend .....	42
5.2.1 Configurarea Mediului de Dezvoltare și a Bazelor de Date .....	42
5.2.2 Implementarea Algoritmilor de Recomandare .....	47
6. Testare .....	57
7. Concluzii .....	61





## Lista Figuri

1. Figura 2.1 Studiu de caz realizat personal ce indică tendința folosirii platformelor de rezervări .....	20
2. Figura 3. 1 Prezentarea unui model MVC [8] .....	26
3. Figura 4. 1 Schema logică a bazelor de date .....	31
4. Figura 5.1 Diagrama UML .....	33
5. Figura 5.1.1 Exemplu 1 HTML .....	34
6. Figura 5.1.2 Exemplu 2 HTML .....	35
7. Figura 5.1.3 Exemplu 3 HTML .....	35
8. Figura 5.1.4 Exemplu 4 HTML .....	35
8. Figura 5.1.5 Exemplu 5 HTML .....	35
9. Figura 5.1.6 Pagina Autentificare .....	36
10. Figura 5.1.7 Formularul de Înregistrare .....	37
11. Figura 5.1.8 Pagina Profilului de Utilizator .....	38
12. Figura 5.1.9 Pagina de Recomandări .....	39
13. Figura 5.1.10 Pagina Profilului de Proprietar .....	40
14. Figura 5.1.11 Formularul de Editare a Caracteristicilor Unui Hotel .....	40
15. Figura 5.1.12 Formular de Înregistrare a Unui Hotel .....	41
16. Figura 5.1.13 Formular de Înregistrare a Unui Hotel 2 .....	42
17. Figura 5.2.1 Pornire a Mediului Virtual (VENV – Virtual Environment) .....	42
18. Figura 5.2.2 Instalarea Modulului Django .....	43
19. Figura 5.2.3 Baza de Date Renters .....	44
20. Figura 5.2.4 Baza de Date Landlords .....	45
21. Figura 5.2.5 Baza de Date Hotels .....	46
22. Figura 5.2.6 Baza de Date Sejururi .....	47
23. Figura 5.2.7 Definirea Modelelor 1 .....	48
24. Figura 5.2.8 Definirea Modelelor 2 .....	48
25. Figura 5.2.9 Definirea Modelelor 3 .....	49
26. Figura 5.2.10 Definirea Modelelor 4 .....	49
27. Figura 5.2.11 Definirea Modelelor 5 .....	50
28. Figura 5.2.12 Importarea Bibliotecilor Necesare .....	50
29. Figura 5.2.13 Logica Filtrului Bazat pe Conținut .....	51
30. Figura 5.2.14 Logica Algoritmilor ce Decid În Funcție de Rating și Feedback .....	53
31. Figura 5.2.15 Definirea Unei Funcții ce Calculează Distanța Euclidiană .....	54

32. Figura 5.2.16 Calcularea Unui Centroid Geografic .....	54
33. Figura 5.2.17 Normalizarea .....	55
34. Figura 5.2.18 Crearea Unui Model Combinat 1 .....	56
35. Figura 5.2.18 Crearea Unui Model Combinat 2 .....	56
36. Figura 6.1 Descărcarea Uneltei Locust .....	57
37. Figura 6.2 Rezultatul Testelor de Stres .....	57
38. Figura 6.3 Reprezentarea Grafică a Testelor de Stres .....	56

## **Lista Tabele**

1. Tabel 5. 1 Câmpurile ce descriu un utilizator .....	42
2. Tabel 5.2 Câmpurile ce descriu un Renter .....	43
3. Tabel 5.3 Câmpurile ce descriu un Landlord .....	44
4. Tabel 5.4 Câmpurile ce descriu un hotel .....	45
5. Tabel 5.5 Câmpurile ce descriu un sejur .....	46



## **Sinopsis**

Un sistem de recomandare este un algoritm inteligent, de obicei asociat cu învățarea automată, care utilizează date pentru a sugera sau a recomanda consumatorilor produse adiționale. Tema recomandată (Sistem de recomandare pentru rezervări online) sugerează ideea impementării algoritmilor de recomandare în nișa rezervărilor online.

Scopul acestui proiect este de a ajuta utilizatorii aplicației să își găsească unități de cazare potrivite nevoilor personale specifice, bazate pe comportamente și alegeri anterioare. Tot ca scop am avut și păstrarea funcționalității clasice a unei aplicații de recomandări, luând în considerare atât factorii de similitudine, rating, cuvinte cât și pe cei geologici pentru o recomandare cât mai calitativă.

Fiind o aplicație de recomandare pentru rezervari online, am plecat de la nevoia de a crea o interfață grafică simplă și intuitivă. Dorința a fost ca aplicația să poată fi folosită de orice tip de utilizator, indiferent de nivelul personal de pregătire în domeniul tehnic. Rezultatele sunt ușor de vizualizat și interpretat.



## 1. Introducere

Într-o epocă în care majoritatea părților din viețile noastre au ajuns să fie digitalizate (job, comerț, divertisment, educație, cercetare, marketing, etc), multe nișe au avut parte de o dezvoltare substanțială în ultimul timp, odată cu revoluționarea tehnologiilor AI (Inteligență Avansată). Importanța ecosistemului online este necontestabilă în cadrul tuturor nișelor de piață (o astfel de nișă este turismul). Nevoia dezvoltării acestuia a plecat de la o premiză simplă: nevoia de informație.

Informația, prin definiție, este data sau forma de dată de orice tip ce este filtrată și ne răspunde la o întrebare.

În prezent, majoritatea companiilor care vând un produs sau serviciu au incorporat algoritmi de recomandare, pentru ca experiența utilizatorului să fie cât mai plăcută. Acest lucru se rezuma la două mari aspecte: luarea unei decizii în cât mai puțin timp și alegerea unei oferte cât mai potrivite pentru nevoile și bugetul utilizatorului.

Dacă în urma cu 20 de ani, în clipa în care se dorea luarea unui concediu, se apela la recomandarea unei persoane sau la o agenție de turism, în prezent, oricine cu acces la internet poate să își planifice o vacanță cu un efort minim.

În urma acestei introduceri, doresc să subliniez impactul pe care îl au algoritmi de recomandare asupra simplificării procesului de căutare a serviciilor de rezervare online.

Scopul acestui proiect este de a integra anumiți algoritmi de recomandare în cadrul unei aplicații web pentru rezervări online. Partea de funcționalitate a fost implementată folosind framework-uri specifice Python. Aplicația este eficientă, iar rezultatele sunt ușor de analizat și interpretat prin interfața vizuală, creată folosind HTML și CSS. Datele utilizatorilor sunt stocate într-o bază de date folosind SQLite. Ca unități de cazare, am ales manual peste 100 de locații din diferite părți geografice ale României, cu diferite recenzii și facilități.

### 1.1 Context și Motivație

În ultimii ani, turismul a beneficiat substanțial în urma dezvoltării domeniilor tehnologiei informației și comunicațiilor (IT&C). Sisteme de recomandare (RS) au fost dezvoltate pentru a răspunde acestor preocupări, devenind instrumentele de sprijinire a deciziilor. În domeniul turismului, acestea sunt denumite „Sisteme de recomandare în domeniul turismului” (TRS). Turiștii și furnizorii de servicii turistice pot căuta, selecta, compara și lua decizii cu mare ușurință și eficient din punct de vedere al timpului și energiei investite.

Majoritatea sistemelor de recomandare turistică se concentrează pe estimări bazate pe preferințele și interesele utilizatorilor pentru alegerea destinațiilor, activităților și serviciilor turistice. Aceste TRS-uri oferă funcții de filtrare, sortare și potrivire între elemente, dar pentru a fi cu adevărat utile, trebuie să fie și "inteligente" în ceea ce privește aspectele tehnice. Conform [1], aceasta asigură scalabilitatea sistemului, transparența și acuratețea recomandărilor, precum și metode de validare. Totodată, sistemul

trebuie să fie ușor de utilizat și să permită feedback-ul și acceptarea utilizatorului. Proiectarea unui sistem TRS eficient trebuie să găsească un echilibru între aspectele practice și cele tehnice pentru a oferi utilizatorilor recomandări relevante și ușor de utilizat.

Conform [2], peste 70% din populația României a fost în minim o vacanță în anul 2023, dintre care peste 90% fiind în România. Un procentaj de doar 18% dintre aceștia au apelat la agenții de turism, restul luând decizia să își organizeze concediul singuri. Având acestea în vedere, am decis să creez o aplicație web prin care să se poată ajunge, într-un mod cât mai simplu, la o decizie privind rezervarea unei unități de cazare din România, pentru ca oricine să poată primi recomandări personalizate, fără nevoia intervenției unei agenții de turism.

Având acestea în vedere, am decis să creez o aplicație user-friendly ce are ca scop asistarea și ajutorarea în rezolvarea acestor probleme, astfel încât să se elimine, în mare parte, sentimentul de stres ce implică găsirea unei locații ideale.

## **1.2 Prezentarea Problemei**

În momentul actual, pe piață există un număr mare de astfel de platforme pentru recomandarea unităților de cazare. Principala problemă ce se dorește a fi rezolvată este eliminarea stresului ce face parte dintr-un concediu – organizarea acestuia în funcție de nevoi. Ca mai apoi să se ia decizia potrivită, trebuie făcută o filtrare între toate unitățile de cazare disponibile. Ce nu reușesc toate platformele existente să ofere sunt factorii de personalizare. Furnizarea de recomandări personalizate este esențială în a crea un serviciu calitativ.

De asemenea, interfața cu utilizatorul este unul dintre cele mai importante aspecte, deoarece nu toți oamenii au cunoștințe tehnice adecvate.

## **1.3 Obiective**

Utilizatorul tipic al aplicației este o persoană tipică, ce poate sau nu să aibă cunoștințe tehnice minime. Plecând de la acest lucru, am identificat o serie de obiective ce trebuie atinse pentru a face experiența una cât mai plăcută, încercând, totodată, să aducem anumitor afaceri turistice o prezență online.

Plecând de la aceasta, au fost identificate anumite aspecte ce trebuie luate în considerare.

În profilul de proprietar, înregistrarea unei unități de cazare trebuie să fie intuitivă. Totodată, datele necesare trebuie să fie minimale. Cantitatea de date ce trebuie oferite de către aplicație în scopul înregistrării trebuie să fie mică.



În profilul de client, alegerea unei unități de cazare trebuie să fie simplu de realizat. Recomandările trebuie să fie precise și să fie potrivite fiecărui utilizator în parte. Totodată, contează foarte mult viteza de încărcare a paginii. Conform [3], în 2006, după un studiu făcut de Amazon, fiecare zecime de secundă latență costă 1% din vânzări. Aceștia au tras o concluzie foarte importantă: Viteza de încărcare a paginii contează.

Așadar, deși aplicația dorește implementarea funcționalităților într-un mod cât mai eficient, trebuie să fie luat în considerare posibilitatea de extindere a funcționalităților în viitor. Având în vedere dezvoltarea rapidă a tehnologiei (îmbunătățirea și eficientizarea algoritmilor de recomandare), nu putem omite importanța de a avea opțiunea să aducem îmbunătățiri ale funcționalităților în mod constant.

#### **1.4. Soluția Propusă**

Având în vedere importanța necesității unei platforme ușor de folosit, destinată unui public ce nu a beneficiat de o pregătire tehnică și care dorește atât să își promoveze unitatea de cazare, cât și să elimine partea de stres și filtrare a cazărilor nepotrivite în momentul plecării în concediu, am dorit să creez o platformă ce permite acest lucru.

Aceasta permite creerea de conturi și salvarea datelor într-o manieră sigură, ca mai apoi să beneficieze de serviciile oferite – căutarea cazării sau listarea unei unități în scopul primirii persoanelor.

În scopul facilitării experienței utilizatorilor, vor exista module separate de proprietar și utilizator care vor putea fi accesate cu același cont. Recomandarea va fi făcută în funcție de mai multe variabile: Poziția geografică a unității de cazare, rating, similitudine și cuvinte cheie.

#### **1.5. Rezultate**

Am realizat un sistem de căutare și recomandare a locuințelor de închiriat/locurilor de cazare în funcție de preferințele utilizatorilor. Acesta oferă recomandări calitative în funcție de profilul utilizatorului.

În scopul testării, am populat o bază de date cu un 130 de unități de cazare de pe teritoriul țării. Am creat un profil de utilizator și am adăugat la preferințe anumite unități de cazare, urmând să-mi fie făcut un profil. Recomandarea a fost bazată pe geolocație (Coordonate geografice tip latitudine și longitudine), cuvinte cheie (WiFi, Parcare, Piscina, Jacuzzi, etc), rating-ul facut de către alți utilizatori (Număr real între 1 și 5) și similitudinea între unitățile ce pot fi sugerate și cele care sunt deja adăugate la favorite.

Rezultatul a fost unul bun, recomandările au fost calitative. Acestea s-ar putea îmbunătăți odată cu creșterea numărului de unități de cazare disponibile.

## **1.6. Structură și observații**

Structura acestei lucrări este compusă din 9 capitole. Primul capitol se axează pe context și prezintă motivația, obiectivele propuse, soluțiile propuse și rezultatele obținute. Al doilea capitol se bazează pe un chestionar care descrie nevoile proiectului pentru a identifica principalele caracteristici ale produselor existente și problemele cu care se confruntă utilizatorii.

În capitolul al treilea este prezentat un studiu de caz ce face o analiză a soluțiilor disponibile pe piață ce au implementat soluții de recomandare în domeniul turismului.

Capitolul al patrulea analizează tehnologiile utilizate pentru a construi aplicația, inclusiv baza de date, front-end și back-end, ca apoi, să fie descrisă arhitectura aplicației, funcționalitatea logică și proiectarea bazei de date în capitolul 5.

Capitolul 6 oferă informații detaliate despre implementare, inclusiv caracteristicile componentelor utilizate, provocările întâmpinate în timpul dezvoltării și alte caracteristici implementate.

Capitolul 7 descrie procesul de testare rezultat și concluziile trase în urma acestuia, iar capitolul 8 sintetizează rezultatele obținute. Ultimul capitol este reprezentat de bibliografie.

## **2. Contextualizarea Proiectului**

Odată cu creșterea fenomenală a cererii de turism în lume în ultimele două decenii, există un interes tot mai mare pentru cercetarea în domeniul turismului. În urmă cu douăzeci de ani, existau doar câteva reviste academice care publicau cercetări legate de turism. În prezent, există peste 70 de reviste care deservește o comunitate de cercetare înfloritoare care acoperă peste 3000 de instituții de învățământ superior de pe cinci continente. Fiind unul dintre domeniile importante în cercetarea turismului, modelarea și prognoza cererii în turism a atras atenția atât a cadrelor universitare, cât și a practicienilor.

### **2.1 Baza Științifică a Turismului**

Turismul reprezintă “acțiunea și procesul de petrecere a timpului departe de casă în căutare de recreere, relaxare și plăcere, utilizând în același timp serviciile comerciale”. Așadar, turismul este un produs al aranjamentelor sociale moderne. Acesta reprezintă o noutate, fiind practicat, de oamenii de rând, începând cu secolul al 20-lea.

Conform unei analize cuprinzătoare realizate de Li, Song și Witt (2005), 420 de studii pe această temă au fost publicate în perioada 1960-2002. Majoritatea acestor studii se concentrează pe aplicarea diferitelor tehnici, atât calitative, cât și cantitative, pentru a modela și prognoza ideea de turism.

Analiza oferă o prezentare completă a tuturor metodelor utilizate în modelarea și prognoza cererii de turism, inclusiv modelele de serii temporale, abordarea econometrică, precum și unele metode statistice și non-statistice noi și emergente. Obiectivul principal este, prin urmare, de a investiga dacă există noi tendințe/probleme care au apărut recent în literatura de specialitate privind previziunile în domeniul turismului și de a sugera noi direcții de cercetare viitoare pe baza noilor tendințe și probleme identificate.

Un lucru pe care nu trebuie să îl ignorăm este impactul social al turismului.

Turismul a înregistrat o transformare semnificativă de când SARS-COV-2 a fost declarat pandemie de către Organizația Mondială a Sănătății în martie 2020. De atunci, coronavirusul s-a răspândit exponențial, ceea ce a dus la o serie de efecte negative asupra sănătății mentale a rezidenților la nivel global. Impactul continuă, de asemenea, să fie resimțit în mare măsură de mai mulți rezidenți al căror mijloc de trai depinde de turism. Factorii de decizie politică în domeniul turismului, practicienii și cercetătorii din întreaga lume sunt implicați în discuții privind acțiuni de consolidare pentru a salva această industrie extrem de vulnerabilă.

Totodată, nevoia de a schimba mediul de viață, pe termen scurt, este resimțită de orice individ. Turismul este mai mult decât esențial în viața umană – lucru resimțit cel mai mult în timpul pandemiei, când toată lumea simțea nevoia unei schimbări de peisaj.

## 2.2 Evaluarea Soluțiilor Existente în Turism

Cantitatea de informații disponibile pe World Wide Web și numărul de utilizatori au înregistrat o creștere enormă în ultimul deceniu. Toate aceste informații pot fi deosebit de utile pentru acei utilizatori care intenționează să viziteze o destinație necunoscută. Informațiile despre destinațiile de călătorie și resursele asociate acestora, cum ar fi cazarea, restaurantele, muzeele sau evenimentele, printre altele, sunt frecvent căutate de turiști pentru a planifica o călătorie.

De aici a plecat nevoia implementării sistemelor de recomandare în domeniul turistic.

În acest moment, printre cele mai mari platforme ce oferă aceste servicii în România se numără: Booking, Airbnb, Turistinfo, Travelminit, Undemergem și multe altele.

În 2023, Booking a raportat 120 de milioane de utilizatori activi care au plătit, prin intermediul aplicației, peste 150 miliarde de dolari [7]. Dat fiind faptul că turismul este în creștere, lucrul acesta se poate observa clar în încasările aplicației ce conduce piața când vine vorba de asta. Veniturile Booking fac parte dintr-un trend crescător și au parte de o creștere anuală de aproximativ 25%.

Majoritatea aplicațiilor moderne au integrat sisteme performante de recomandare, bazate pe filtrul cumulativ, algoritmi de ranking, similitudine, geolocație și cuvinte cheie. Printre acestea se numără și cele menționate anterior.

Principala problemă, după cum a fost observată și în studiul de caz, a fost acuratețea informațiilor și a pozelor, dar și integritatea proprietarilor. Totodată, toate aplicațiile menționate percep un comision comercial cuprins între 14% și 20%.

Aproximativ 85% din persoanele ce au completat studiul de caz respectiv folosesc aceste aplicații. Așadar își antrenează algoritmi cu o cantitate foarte mare de date, lucru ce duce la rezultate calitative. Putem observa că oamenii ce au optat pentru apelarea la o agenție de turism sau care s-au ocupat individual este mult mai mic (10% respectiv 5%). Studiul de caz a fost făcut pe 60 de persoane, cu vârste cuprinse între 19 și 51 de ani, media de vârstă fiind aproximativ 26 de ani.

Care este modul în care v-ați organizat vacanțele, în ultimul timp?

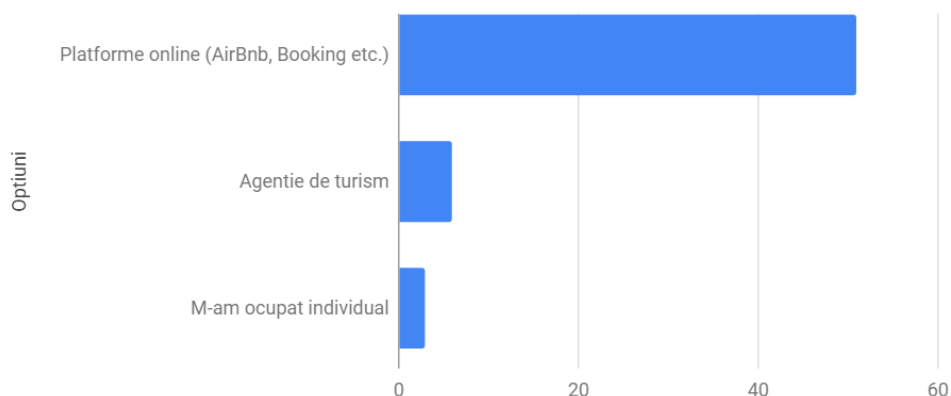


Figura 2.1 Studiu de caz realizat personal ce indică tendința folosirii platformelor de rezervări online

Deși aceste platforme au o acoperire globală și oferă soluții convenabile și unice pentru rezervarea cazărilor și planificarea călătoriilor, ele au și limitări. Acestea percep taxe comerciale substanțiale, care pot fi deranjante pentru proprietari. Există discrepanțe ocazionale în ceea ce privește acuratețea informațiilor și a fotografiilor furnizate, iar fiabilitatea anunțurilor și integritatea proprietarilor pot fi uneori puse la îndoială.

În plus, faptul că se bazează foarte mult pe recenziiile utilizatorilor, care pot fi părtinitoare sau manipulate, reprezintă o altă provocare. În ciuda acestor dezavantaje, opțiunile sofisticate de filtrare și de căutare de pe aceste platforme îi ajută pe utilizatori să găsească exact ceea ce au nevoie.

Cu toate acestea, există încă un loc semnificativ pentru îmbunătățiri, în special în ceea ce privește sporirea acurateței informațiilor, asigurarea integrității anunțurilor și reducerea taxelor comerciale. Proiectul nostru își propune să abordeze aceste lacune prin implementarea unui sistem de recomandare fiabil și ușor de utilizat pentru cazările de călătorie, valorificând algoritmi de recomandare multipli și caracteristici centrate pe utilizator pentru a îmbunătăți experiența și încrederea utilizatorilor.



### 3. Tehnologii și Arhitecturi

În această secțiune, va fi prezentată infrastructura tehnică și arhitectura propusă pentru acest sistem. Prin valorificarea tehnologiilor moderne și a celor mai bune practici arhitecturale, scopul acestui proiect este de a construi o aplicație robustă, scalabilă și ușor de utilizat, care să abordeze deficiențele identificate în soluțiile existente.

#### 3.1 Infrastructura tehnică

Infrastructura tehnică cuprinde diferitele tehnologii și instrumente care vor fi utilizate pentru a dezvolta, implementa și întreține sistemul de recomandare a călătoriilor. Aceasta include limbaje de programare, baze de date și alte instrumente și servicii de sprijin. O infrastructură tehnică bine gândită este esențială pentru a se asigura că sistemul este scalabil, poate fi întreținut și este capabil să gestioneze volume mari de trafic, oferind în același timp o experiență de utilizare fără probleme.

Ca limbaje de programare, au fost utilizate:

**Python** este principalul limbaj de programare utilizat pentru dezvoltarea backend datorită simplității, lizibilității și suportului său extins de biblioteci. Fiind un limbaj high-level, acesta permite dezvoltatorilor să scrie cod într-un mod intuitiv și ușor de înțeles, ceea ce reduce timpul necesar pentru dezvoltarea și întreținerea aplicațiilor. Structura sa clară și sintaxa minimalistă duc la accesibilitate atât pentru începători, cât și pentru programatori experimentați.

Adoptarea sa pe scară largă în comunitatea de dezvoltare a dus la crearea unei vaste game de biblioteci și cadre de lucru, cum ar fi Django, care simplifică și accelerează procesul de dezvoltare a aplicațiilor web. Aceste biblioteci oferă soluții predefinite pentru probleme comune, permițând dezvoltatorilor să se concentreze pe logica specifică a aplicației lor, în loc să reinventeze roata. De exemplu, Django include un sistem robust de autentificare, administrare și manipulare a bazelor de date, ceea ce economisește timp și efort considerabil – avantaje de care această aplicație a beneficiat.

De asemenea, Python este o aplicație ce beneficiază de un bun suport. Având o comunitate mare și activă, dezvoltatorii pot găsi rapid soluții la problemele lor prin intermediul forumurilor, grupurilor de discuții și resurselor online. Documentația este detaliată și numeroasele tutoriale disponibile fac ca învățarea și utilizarea Python să fie foarte accesibile. Această abundență de resurse este utilă pentru dezvoltarea și întreținerea pe termen lung a aplicațiilor.

Acest limbaj de programare este interpretat. Codul Python este executat linie cu linie, ceea ce facilitează detectarea și rezolvarea erorilor. Spre deosebire de limbajele compilate, unde întregul cod trebuie tradus în limbaj mașină înainte de execuție, un limbaj interpretat permite dezvoltatorilor să testeze și să modifice codul în mod iterativ. Aceasta reduce semnificativ timpul de dezvoltare și permite o abordare mai flexibilă și interactivă în crearea aplicațiilor. Cu toate acestea, fiind un limbaj interpretat prezintă și dezavantaje, principalul fiind timpul de rulare.

**Django** este un cadru web Python de nivel înalt care încurajează dezvoltarea rapidă și o proiectare curată și pragmatică. Fiind construit pe principiul "baterii incluse", Django oferă o gamă largă de caracteristici integrate, ceea ce permite dezvoltatorilor să se concentreze pe aspectele specifice ale aplicațiilor lor fără a fi necesar să reinventeze roata. Printre aceste caracteristici se numără autentificarea, Object-Relational Mapper (ORM) și gestionarea formularelor.

Autentificarea este un element critic pentru multe aplicații web, iar Django furnizează un sistem robust și personalizabil pentru gestionarea utilizatorilor și a sesiunilor. Aceasta include gestionarea parolilor, autentificarea pe mai multe niveluri și permisiuni complexe. ORM-ul din Django permite dezvoltatorilor să interacționeze cu baza de date într-un mod intuitiv, utilizând modele Python în locul scripturilor SQL. Aceasta nu doar simplifică codul, dar și face aplicațiile mai ușor de întreținut și de extins.

Gestionarea formularelor este o altă componentă esențială oferită de Django, care permite crearea, validarea și procesarea formularelor HTML într-un mod eficient.

Unul dintre punctele forte ale Django este accentul pus pe securitate. Caracteristicile robuste de securitate ale Django includ protecția împotriva injecției SQL, a scripturilor cross-site (XSS) și a falsificării cererilor cross-site (CSRF). Protecția împotriva injecției SQL previne atacurile care ar putea compromite baza de date prin injectarea de cod SQL rău intenționat. Protecția XSS împiedică executarea scripturilor malițioase în contextul unui site web, iar protecția CSRF asigură că cererile către aplicație provin de la surse legitime.

**HTML, CSS și JavaScript** sunt tehnologii esențiale pentru dezvoltarea front-end, jucând un rol crucial în crearea de interfețe receptivă și ușor de utilizat. HTML (HyperText Markup Language) asigură structura paginilor web, definind elementele de bază precum titlurile, paragrafele, linkurile, imaginile și formularele. Fiecare element HTML contribuie la organizarea și prezentarea conținutului într-un mod clar și logic, facilitând accesibilitatea și navigarea utilizatorului.

CSS (Cascading Style Sheets) se ocupă de aspectele vizuale și de proiectarea paginilor web. Prin utilizarea CSS, dezvoltatorii pot controla stilurile și aspectul elementelor HTML, inclusiv culorile, fonturile, spațierea, alinierea și aspectul. Acest lucru permite personalizarea și îmbunătățirea vizuală a paginilor web, creând o experiență de utilizare atractivă și coerentă. CSS permite, de asemenea, crearea de modele receptivă care se adaptează automat la diferite dimensiuni de ecran și dispozitive, asigurând accesibilitatea și utilizabilitatea optimă a aplicațiilor pe diferite platforme.

Împreună, HTML și CSS asigură faptul că o aplicație nu este doar funcțională, ci și atractivă din punct de vedere vizual. HTML stabilește fundația structurală a paginilor web, în timp ce CSS adaugă straturi de stil pentru a îmbunătăți aspectul vizual și interactivitatea. Separând structura de styling, dezvoltatorii pot menține un cod curat și organizat, facilitând întreținerea și actualizările site-ului.

Prin utilizarea corectă a etichetelor HTML semantice și a stilurilor CSS adecvate, dezvoltatorii pot crea site-uri web care sunt ușor de navigat pentru utilizatorii cu handicap. De exemplu, etichetele HTML semantice ajută tehnologiile de asistență precum cititoarele de ecran să interpreteze și să prezinte conținutul într-o manieră accesibilă. CSS poate fi utilizat pentru a ascunde elemente de afișare vizuală, menținându-le în același timp accesibile cititoarelor de ecran, asigurând o experiență web mai incluzivă.



Pentru managementul bazelor de date, s-a folosit SQLite. Aceasta este o bază de date relațională cunoscută pentru simplitatea și ușurința sa de utilizare, ceea ce o face potrivită pentru aplicațiile de mici dimensiuni sau pentru prototipuri. SQLite stochează datele într-un singur fișier pe disc, fiind ideal pentru proiectele care nu necesită un server de baze de date separat. Capacitatea sa de a gestiona tranzații rapide și operațiuni simple de citire și scriere îl face o alegere bună pentru aplicațiile care nu au cerințe ridicate de scalabilitate și gestionare a volumelor mari de date. Totodată, este foarte util când vine vorba de testare, folosind Locust.

În acest context, Django utilizează nativ SQLite ca backend pentru baza de date, ceea ce permite dezvoltatorilor să interacționeze cu baza de date folosind ORM-ul Django. Aceasta combină avantajele cadrului Django cu simplitatea și eficiența SQLite.

Implementarea API-ului Google Maps în cadrul aplicației permite proprietarilor să listeze hotelurile în locația exactă. Acel marker plasat pe hartă captează automat coordonatele geografice (latitudine și longitudine) ale locului selectat. Aceste coordonate sunt apoi utilizate pentru a completa câmpurile relevante din formularul de listare a hotelului, asigurând date exacte și fiabile privind locația.

Pentru a obține o cheie Google API pentru utilizarea Google Maps, trebuie creat un cont Google Cloud. Odată ce API-ul este activat, se poate genera o cheie API - Google va furniza apoi o cheie API ce se poate utiliza pentru a integra Google Maps în aplicația dezvoltată.

### **3.2 Arhitectura propusă**

Arhitectura sistemului de recomandare a călătoriilor urmează o abordare MVC (Model-View-Controller). Acest model împarte aplicația în trei componente distincte, dar interconectate: Modelul, Vizualizarea și Controlorul. Acesta separă logica aplicației de prezentare, asigurând organizarea eficientă și ușurința de întreținere a codului. MVC, prin principiul „separării preocupărilor”, nu doar oferă un cadru solid pentru dezvoltarea aplicațiilor web, dar facilitează și scalabilitatea ulterioară a acestora.

Experiența utilizatorului începe cu interacțiunile din cadrul interfeței de utilizator a aplicației. Aceste interacțiuni pot include trimiterea unui formular, apăsarea unui buton sau alte acțiuni complexe. View-ul capturează aceste acțiuni și transmite o cerere către Controller.

Controlorul, adesea considerat „creierul” arhitecturii MVC, primește aceste cereri și decide cum să le gestioneze. Acesta procesează logica și regulile de afaceri ale aplicației pentru a lua deciziile corespunzătoare.

După ce Controller-ul primește cererile, Modelul este responsabil de preluarea și gestionarea datelor. Modelul, asigurând integritatea datelor, formulează interogările necesare și le trimite către baza de date.

Baza de date procesează aceste interogări, extrage informațiile relevante și returnează rezultatele către Model.

Modelul procesează datele primite, asigurându-se că acestea respectă structura și cerințele aplicației. Ulterior, transmite aceste date către Controller.

Pe baza datelor primite de la Model, Controller-ul ia decizii fundamentate pe acțiunile utilizatorului și logica aplicației.

Atunci când datele trebuie prezentate utilizatorului, Controller-ul colaborează cu View-ul. View-ul folosește apoi aceste date pentru a reda corect interfața cu utilizatorul.

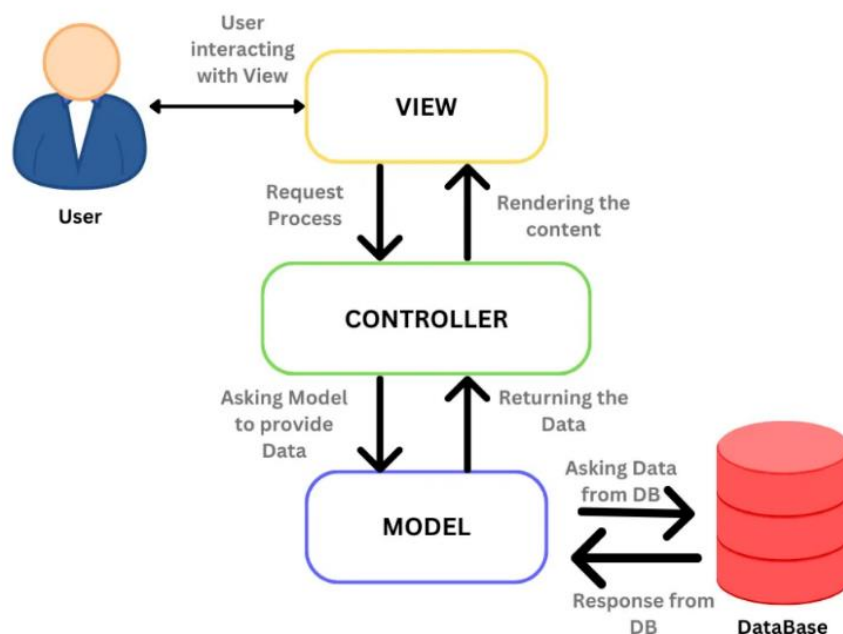


Figura 3.1 Prezentarea unui model MVC [8]

### 3.2.1 Stratul de interfață cu utilizatorul (UI):

Partea frontală a sistemului, dezvoltată cu ajutorul HTML și CSS oferă o experiență de utilizare receptivă și interactivă. Acest strat include diverse pagini, cum ar fi căutarea de cazări, vizualizarea detaliilor de cazare, înregistrarea utilizatorului, conectarea și gestionarea profilului.

Designul interfeței cu utilizatorul se dorește a fi intuitiv, asigurând ușurința de utilizare pentru toate tipurile de utilizatori. Prin captarea preferințelor utilizatorilor, a criteriilor de căutare și a feedback-ului, frontend-ul joacă un rol crucial în personalizarea experienței utilizatorilor și în generarea de recomandări relevante.

Datele colectate din interacțiunile utilizatorilor sunt trimise către backend pentru procesare, permițând sistemului să își adapteze sugestiile în mod eficient.

### **3.2.2 Stratul de logică**

Stratul logic al programului este proiectat cu ajutorul aplicației Django. Acest strat este responsabil de gestionarea autentificării utilizatorilor, de gestionarea cazărilor, de funcționalitățile de căutare, de generarea de recomandări și de procesarea feedback-ului utilizatorilor. Centralizând aceste operațiuni, stratul de logică de afaceri asigură faptul că toate funcționalitățile aplicației funcționează perfect împreună și respectă regulile definite. Punctele finale RESTful API, expuse de aplicația Django, facilitează comunicarea între frontend și backend. Aceste puncte finale sunt concepute pentru a fi sigure, eficiente și ușor de utilizat, permițând o interacțiune fără probleme între diferitele părți ale sistemului.

### **3.2.3 Stratul de persistență a datelor**

SQLite este utilizată ca bază de date principală pentru stocarea tuturor datelor legate de utilizatori, cazări, recenzii și tranzacții. Utilizarea unei baze de date relaționale precum SQLite oferă o soluție simplă și eficientă pentru gestionarea datelor, fiind ideală pentru aplicații de mici dimensiuni sau prototipuri.

Deși nu este destinată gestionării unor volume extrem de mari de date, SQLite este suficient de robustă pentru a susține aplicația și pentru a asigura performanțe constante pe măsură ce baza de utilizatori crește. Acest strat joacă un rol crucial în menținerea integrității și accesibilității datelor, esențiale pentru generarea recomandărilor corecte și pentru a asigura o experiență de utilizare fluentă.



## 4. Proiectare și modelare

În această secțiune va fi prezentată proiectarea logică și modelarea componentelor sistemului de recomandare a călătoriilor. Aceasta explorează arhitectura și modelarea bazei de date pentru a asigura o integrare perfectă și o experiență optimizată a utilizatorului.

### 4.1 Logica și proiectarea componentelor

Obiectivul principal este de a asigura o experiență coerentă pe toate dispozitivele. Interfața include diverse pagini, cum ar fi căutarea de cazări, vizualizarea detaliată a cazărilor, înregistrarea utilizatorului, conectarea și gestionarea profilului. Fiecare pagină este concepută pentru a fi intuitivă, facilitând navigarea și interacțiunea utilizatorilor cu sistemul. Prin prioritizarea utilizabilității și a accesibilității, designul urmărește să se adreseze unei game largi de utilizatori, inclusiv celor cu abilități tehnice limitate.

Atunci când un utilizator interacționează cu aplicația, de obicei prin intermediul interfeței de utilizare, aceste interacțiuni pot varia de la acțiuni simple, cum ar fi trimiterea unui formular sau apăsarea unui buton, până la activități mai complexe. Componenta View captează aceste acțiuni ale utilizatorului și trimite o cerere către Controller. Această interacțiune stă la baza modelului MVC (Model-View-Controller), asigurându-se că interfața de utilizator rămâne receptivă și centrată pe utilizator.

Prin separarea preocupărilor, modelul MVC permite echipei de dezvoltare să lucreze la diferite părți ale aplicației în mod independent, sporind productivitatea și reducând probabilitatea de erori.

Controlorul acționează ca unitate centrală de procesare a modelului MVC. Acesta primește solicitările utilizatorului de la View și determină cursul de acțiune adecvat pe baza logicii de afaceri a aplicației.

Aceasta poate implica recuperarea datelor, procesarea intrărilor utilizatorului sau interacțiunea cu alte componente ale sistemului. Controlerul se asigură că toate cererile sunt tratate eficient și că starea aplicației este actualizată în consecință. Prin centralizarea procesului decizional și a procesării logice, controlorul asigură coerența și fiabilitatea comportamentului aplicației.

La primirea instrucțiunilor de la controler, modelul este responsabil de recuperarea și integritatea datelor. Acesta formulează interogări precise către baza de date, asigurându-se că datele recuperate sunt exacte și relevante. Modelul procesează apoi aceste date pentru a le alinia la structura aplicației înainte de a le înapoia controlorului.

Această separare a preocupărilor garantează că gestionarea datelor este tratată eficient și că aplicația se poate extinde pe măsură ce crește baza de utilizatori. Modelul este conceput pentru a fi robust și flexibil, capabil să gestioneze interacțiuni complexe de date, menținând în același timp integritatea și coerența acestora.

În cazul SQLite, baza de date nu necesită un server dedicat, eliminând complexitatea asociată cu administrarea unui server de baze de date. În schimb, datele sunt gestionate direct de aplicație, stocate într-un singur fișier pe disc. Această abordare simplifică atât dezvoltarea, cât și mentenanța aplicației, permițându-i să ofere performanțe consistente fără a depinde de un sistem server complex.

Deși SQLite nu este proiectată pentru a gestiona volume extrem de mari de date, capacitatea sa de a oferi interogări rapide și precise este suficientă pentru aplicații de dimensiuni medii sau pentru prototipuri. Fiind „serverless,” SQLite oferă un acces eficient la date, păstrând simplitatea și fiabilitatea.

În cele din urmă, controlorul decide cum să prezinte datele utilizatorului. Acesta colaborează cu View pentru a reda în mod corespunzător datele pe interfața cu utilizatorul. Acest lucru poate implica afișarea rezultatelor căutării, a detaliilor de cazare sau a informațiilor specifice utilizatorului. Asigurându-se că datele sunt prezentate într-un mod clar și organizat, sistemul sporește satisfacția și implicarea utilizatorului.

Componenta View, responsabilă de redarea interfeței de utilizator, utilizează tehnologii web moderne pentru a crea o experiență de utilizare interactivă și fără cusur. Menținând interfața cu utilizatorul curată și intuitivă, sistemul urmărește să faciliteze găsirea de către utilizatori a informațiilor de care au nevoie și efectuarea acțiunilor dorite.

## **4.2 Modelarea bazei de date**

Baza de date este concepută pentru a stoca și gestiona eficient toate datele relevante, inclusiv informațiile despre utilizatori, listele de cazare, recenziile și înregistrările tranzacțiilor. Utilizarea SQLite, o bază de date relațională „serverless,” oferă o soluție simplă și eficientă pentru gestionarea datelor structurale într-un format bine definit. Deși SQLite nu este concepută pentru gestionarea unor volume mari de date sau structuri complexe, flexibilitatea sa și ușurința de utilizare o fac ideală pentru aplicații de mici dimensiuni sau prototipuri. Modelul de stocare bazat pe tabele al SQLite este bine potrivit pentru aplicații în care structura datelor este bine definită și stabilă.

Entități precum CustomUsers (din care sunt derivați Landlords și Renters), Hotels, și Sejur formează nucleul modelului bazei de date. Fiecare entitate are attribute și relații specifice cu alte entități. De exemplu, entitatea Users conține informații despre fiecare utilizator, cum ar fi numele, adresa de e-mail, parola și rolul, în timp ce entitatea Sejur stochează detalii despre fiecare cazare, inclusiv ID-ul proprietarului, descrierea, locația și prețul. Aceste entități sunt interconectate, permițând sistemului să gestioneze eficient relații și dependențe complexe. Proiectarea bazei de date asigură că toate informațiile relevante sunt ușor accesibile și pot fi interogate într-un mod performant.

Asigurarea integrității și securității datelor este de o importanță capitală. Proiectarea bazei de date include mecanisme de validare și curățare a datelor pentru a preveni intrările rău intenționate. Sunt implementate procese de autentificare și autorizare a utilizatorilor pentru a proteja informațiile sensibile. În SQLite, identificarea unică a fiecărei înregistrări este realizată prin utilizarea de chei primare, ceea ce asigură că datele sunt stocate și accesate în siguranță. Backup-uri regulate și măsuri de redundanță sunt implementate pentru a asigura disponibilitatea datelor și capacitatea de recuperare în caz de defecțiuni.

Deși SQLite nu este proiectată pentru scalare la nivelul unei aplicații de dimensiuni mari, este suficient de robustă pentru a susține aplicații de dimensiuni medii, cu un număr rezonabil de utilizatori. Indexarea este utilizată pentru a îmbunătăți performanța interogărilor, asigurând un acces rapid la datele necesare. Deși scalarea orizontală nu este o caracteristică nativă a SQLite, arhitectura sa simplă și integrată permite dezvoltatorilor să gestioneze eficient resursele disponibile. Acest lucru s-a dovedit a fi

esențial în cazul unei aplicații pentru recomandări de unități de cazare, deoarece volumul de date ce trebuie procesate este mare.

Structura bazei de date este proiectată cu atenție pentru a echilibra performanța, scalabilitatea și mentenabilitatea. Sistemul poate gestiona și prelua eficient informațiile prin organizarea datelor în entități și relații bine definite. Această abordare structurată garantează că sistemul de recomandare a călătoriilor este robust, ușor de întreținut și capabil să ofere performanțe constante. Punând accentul pe interfețe ușor de utilizat, pe o gestionare eficientă a datelor și pe practici sigure, sistemul își propune să ofere utilizatorilor săi o experiență calitativă, din toate punctele de vedere.

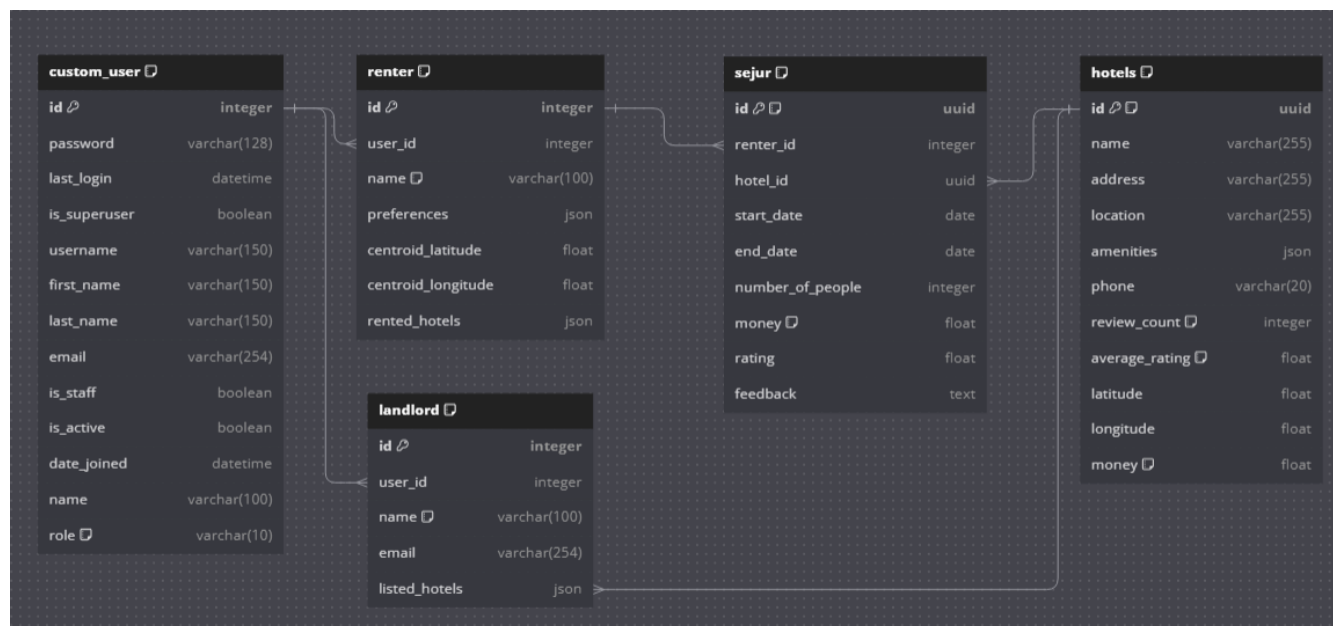


Figura 4.1 Schema logică a bazelor de date

Între entitățile „users” și „landlords” este o relație de one-to-one prin „user\_id”. Între entitățile „users” și „sejur” este o relație de one-to-many prin „user\_id”. Între entitățile „landlords” și „hotels” este o relație de one-to-many prin „ListedHotels”. Între entitățile „hotel” și „sejur” este o relație de one-to-many prin „hotel\_id”.





## 5. Implementare și Dezvoltare

În acest capitol, se dorește evidențierea aspectelor practice ale construirii aplicației.

Acesta acoperă atât dezvoltarea interfeței cu utilizatorul, cât și execuția diferitelor componente, detaliind modul în care a fost dezvoltată și integrată fiecare parte a sistemului - domeniile cheie includ crearea mediului de dezvoltare, configurarea serverului, implementarea funcționalităților de bază și abordarea provocărilor întâlnite în timpul dezvoltării.

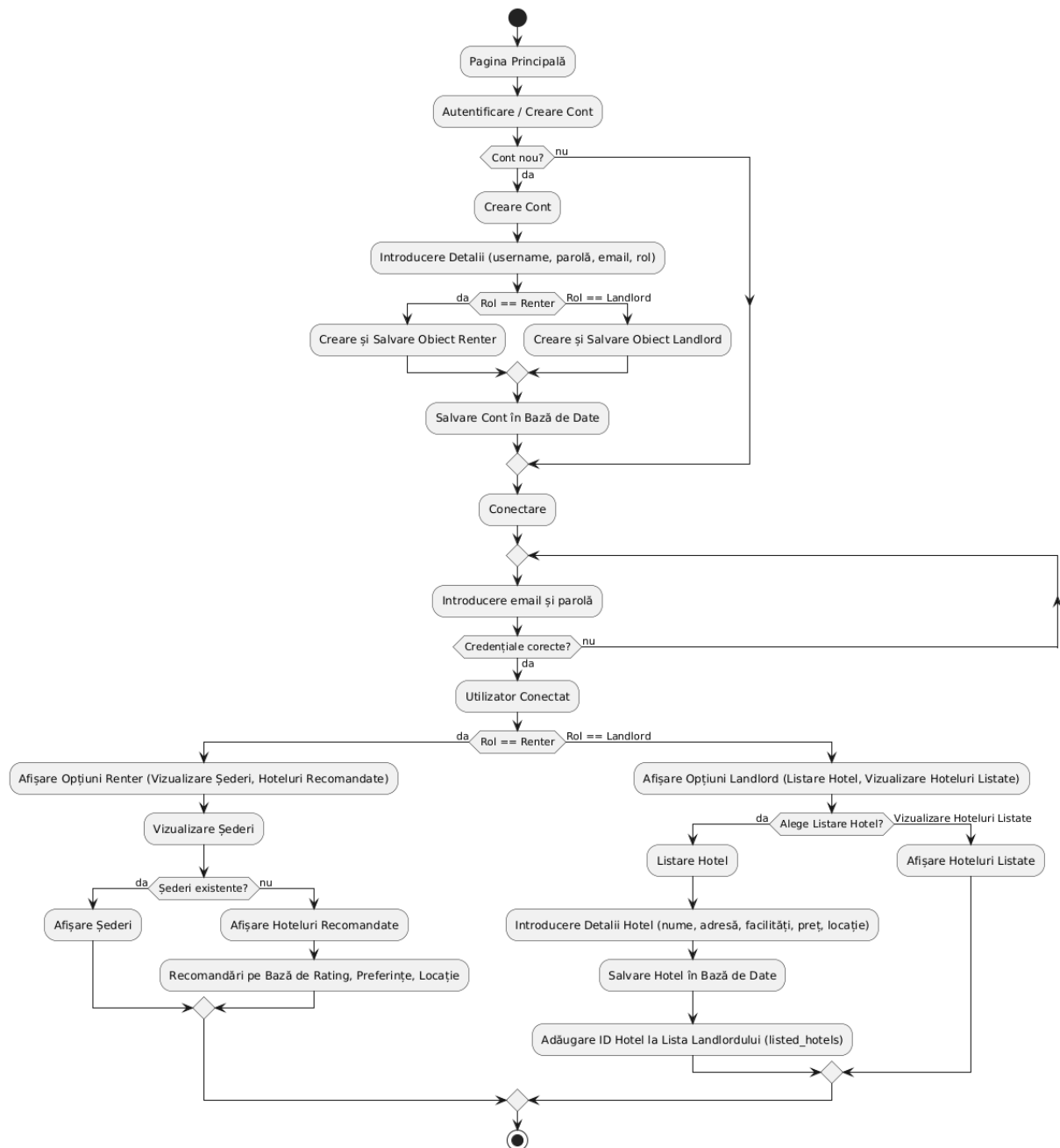


Figura 5.1 Diagrama UML

## 5.1 Dezvoltare Frontend

Partea de frontend a fost dezvoltată folosind HTML, CSS și elemente de JavaScript.

Spre exemplu, pagina index.html este structurată folosind HTML pentru a oferi un aspect clar și ușor de utilizat pentru utilizatori. Documentul începe cu o declarație standard HTML5 doctype și include meta tag-uri necesare în secțiunea <head>, cum ar fi setul de caractere, setările viewport-ului pentru design responsiv și titlul paginii, "Acasă". Aceasta apelează la un fișier CSS extern (styles.css) pentru a stiliza pagina în mod consecvent.

Secțiunea de <body> conține câteva elemente cheie. În partea superioară, un element <header> prezintă un mesaj de bun venit, "Bine ați venit pe site-ul nostru de rezervări hoteliere", încapsulat într-un tag <h1>. Acest antet include și o bară de navigare implementată folosind un element <nav>. Bara de navigare include câteva tag-uri <a> care oferă linkuri către diferite secțiuni ale site-ului, cum ar fi "Acasă", "Autentificare", "Profil" și "Hoteluri", permițând utilizatorilor o navigare ușoară.

Conținutul principal al paginii este înfășurat într-un element <div> cu clasa main-container. Acest container oferă o zonă centrală pentru afișarea mesajului principal și a oricărui conținut recomandat. Este stilizat pentru a fi vizibil distinct și central pe pagină, adesea cu padding, culori de fundal sau borduri pentru a-l separa de restul conținutului. În interiorul acestui container principal, titlurile și paragrafele suplimentare introduc site-ul, evidențiază caracteristicile cheie și ghidează utilizatorul asupra acțiunilor pe care le poate întreprinde în continuare.

```
200 <body>
201   <header>
202     <h1>Hoteluri Disponibile și Recomandate</h1>
203     <nav>
204       <a href="{% url 'profile-view' %}">Profilul tău</a>
205       <a href="{% url 'acasa-view' %}" class="active">Fă o rezervare</a>
206       <a href="{% url 'logout-view' %}">Deconectare</a>
207     </nav>
208   </header>
209
210   <div class="main-container">
211     <h2>Lista Hotelurilor Disponibile și Recomandate</h2>
212     <ul class="hotel-list">
213       {% for hotel in hotels %}
214       <li class="hotel-item">
```

Figura 5.1.1 Exemplu 1 HTML

```

215 <div class="hotel-info">
216   <h3>{{ hotel.name }}</h3>
217   <p>Adresă: {{ hotel.address }}</p>
218   <p>Facilități:
219     {% if hotel.amenities|first == "[" %}
220       {{ hotel.amenities|safe|slice:"1:-1"|join:", " }}
221     {% elif hotel.amenities is list %}
222       {{ hotel.amenities|join:", " }}
223     {% else %}
224       {{ hotel.amenities }}
225     {% endif %}
226   </p>

```

Figura 5.1.2 Exemplu 2 HTML

```

227   <p>Telefon: {{ hotel.phone }}</p>
228   <p>Preț: {{ hotel.money }} RON/noapte</p>
229 </div>
230 <div class="hotel-actions">
231   <div class="hotel-rating" data-reviews="{{ hotel.review_count }}">{{ hotel.average_rating|floatformat:1 }} ★</div>
232   <button class="reserve-button" onclick="showForm('{{ hotel.id }}')">Rezervă acum</button>
233 </div>
234 </li>
235 {% endfor %}
236 </ul>
237 </div>

```

Figura 5.1.3 Exemplu 3 HTML

```

239 <!-- Formular de rezervare -->
240 <div id="reserveModal" class="modal">
241   <div class="modal-content">
242     <span class="close" onclick="hideForm()">&times;</span>
243     <h2>Rezervă Hotelul</h2>
244     <form method="post" action="{% url 'acasa-view' %}">
245       {% csrf_token %}
246       {{ form.as_p }}
247       <input type="hidden" id="id_hotel" name="hotel" value="">
248     </form>
249     <button type="submit">Trimite Rezervarea</button>
250   </div>
251 </div>
252 </div>

```

Figura 5.1.4 Exemplu 4 HTML

```

254 <script>
255   function showForm(hotelId) {
256     console.log("Button clicked for hotel: " + hotelId);
257     document.getElementById('id_hotel').value = ''; // Resetează valoarea înainte de a seta noul hotelId
258     document.getElementById('id_hotel').value = hotelId;
259     document.getElementById('reserveModal').style.display = 'block';
260   }

```

Figura 5.1.5 Exemplu 5 HTML

### 5.1.1 Pagina de autentificare

Această pagină de autentificare are un design simplist și atrăgător din punct de vedere vizual, având pe fundal un peisaj de tip „Travel Stock Image”. Pagina utilizează un formular de autentificare centralizat cu câmpuri pentru "Nume utilizator" (Username) și "Parolă" (Password). Câmpurile de intrare sunt stilizate cu un aspect modern, păstrând simplitatea și funcționalitatea. Butonul "Autentificare" (Login) este afișat în mod vizibil sub câmpurile de introducere a datelor.

Stilurile CSS aplicate asigură că elementele sunt bine aliniate și că textul iese în evidență în mod clar pe fundal. Utilizarea umbrelor și a border-radius-ului în elementele de design împunătățește estetica generală a paginii. Formularul și legăturile de navigare sunt funcționale, oferind utilizatorilor o interfață intuitivă și plăcută din punct de vedere vizual.

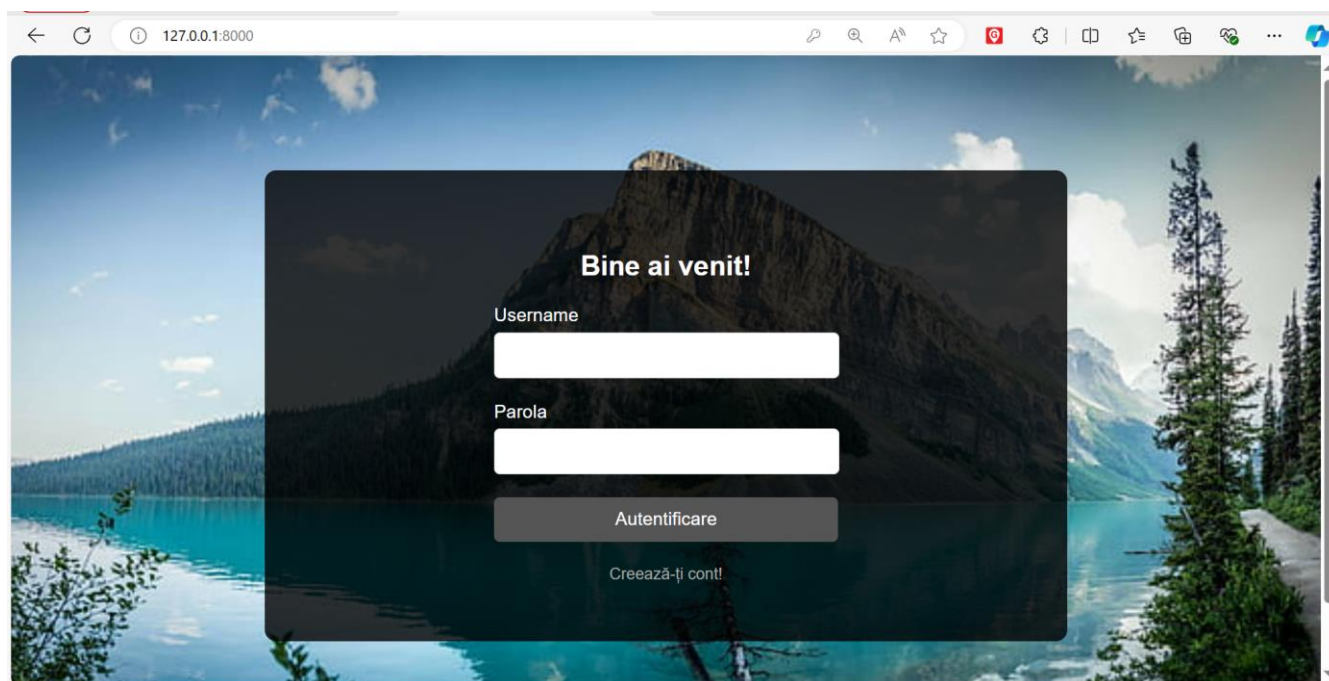


Figura 5.1.6 Pagina Autentificare

În cazul în care un cont de utilizator nu a fost creat încă, avem opțiunea de a o face apăsând pe butonul „Înregistrează-te”. Putem observa cum apare un formular prin care se face înscrierea unui nou utilizator, putând alege dacă va fi de tip „Renter” sau de tip „Landlord”. Am facut acest lucru pentru a stabili o delimitare clară între tipurile de utilizatori, fiecare având un meniu personalizat.

De asemenea, fiecare tip de user servește funcționalități diferite. Un „Renter” poate să caute unități de cazare și poate să beneficieze de recomandări calitative.

Un „Landlord” poate să își listeze propria unitate de cazare și să o managerieze din interiorul unei interfețe specifice.z

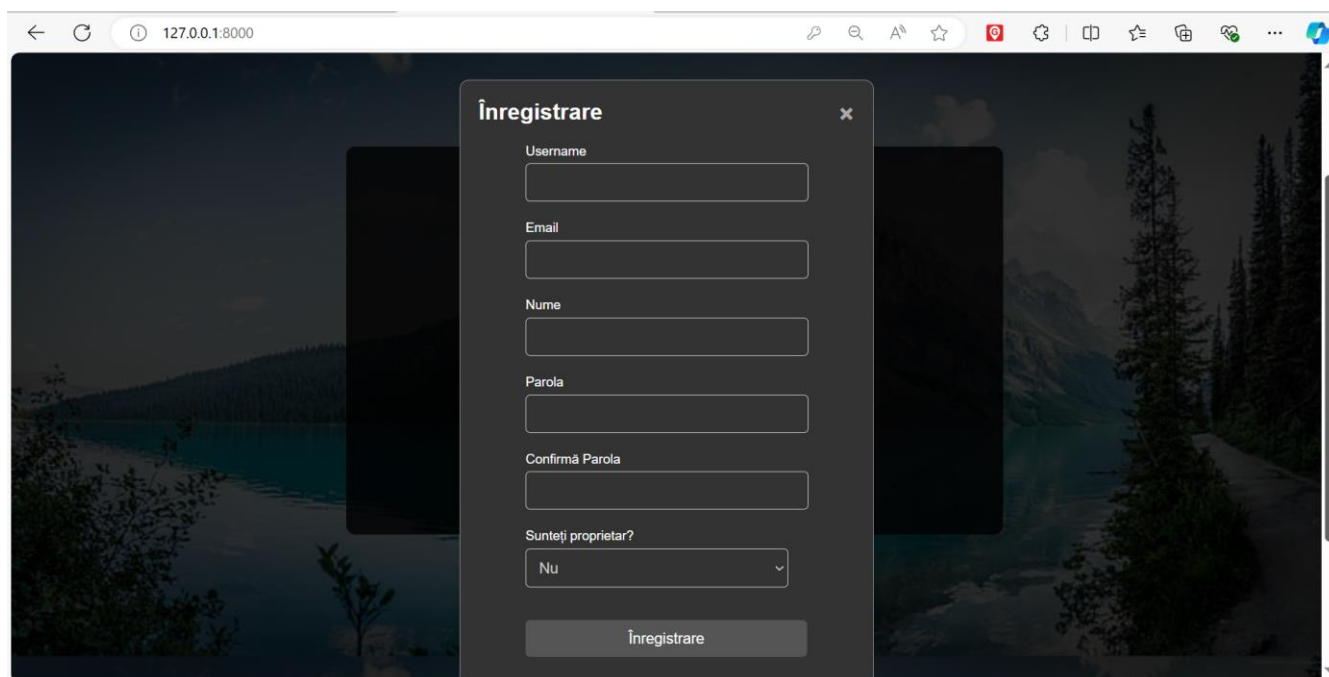


Figura 5.1.7 Formularul de Înregistrare

### 5.1.2 Pagina profilului de utilizator tip „Renter”

Această pagină prezintă utilizatorul ce dorește să găsească o unitate de cazare, în acest caz, "Renter", și îi permite să vizualizeze informațiile din profil.

Accentul se pune pe afișarea istoricului de sejururi hoteliere ale utilizatorului, enumerate într-un format structurat. Fiecare intrare prezintă informații detaliate despre hoteluri, cum ar fi numele, adresa, facilitățile, datele de contact și un număr de stele.

Lista oferă o imagine de ansamblu clară a sejururilor anterioare, ajutând utilizatorii să își reamintească experiențele. Acest design asigură faptul că utilizatorii pot accesa și revizui cu ușurință rezervările lor anterioare, promovând un sentiment de continuitate și servicii personalizate.

De asemenea, în meniul intuitiv se poate lăsa o recenzie pentru un hotel la care un utilizator a fost cazat. Odată selectată o notă pentru un hotel, aceasta nu poate să fie schimbată. O recenzie se poate lăsa doar pentru un hotel din cadrul unui sejur ce are loc în prezent sau ce a avut loc în trecut.

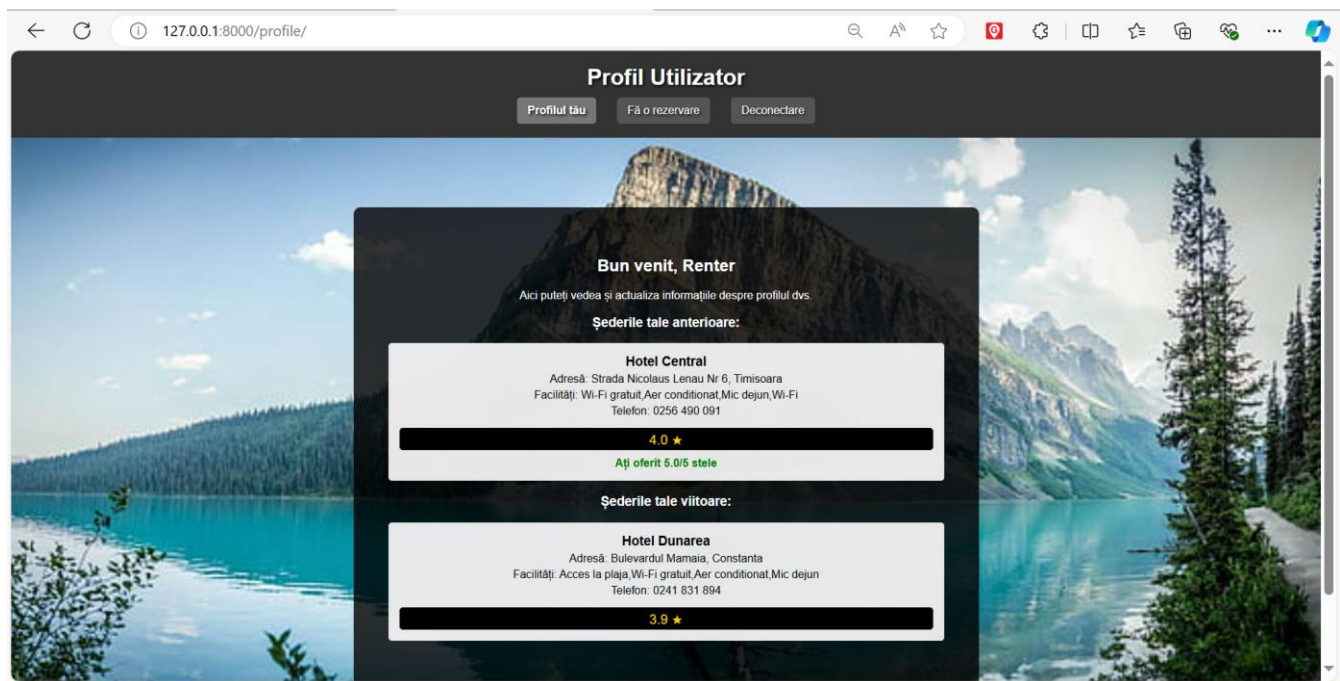


Figura 5.1.8 Pagina Profilului de Utilizator

### 5.1.3 Pagina pentru recomandarea de hoteluri

Pagina de recomandare a hotelurilor este construită pentru a oferi utilizatorului o experiență personalizată, făcând procesul de rezervare de hoteluri mai ușor și eficient. În spate, sistemul utilizează preferințele și istoricul utilizatorului pentru a genera o listă de hoteluri recomandate, bazându-se pe factori precum ratingurile anterioare, locațiile preferate și facilitățile de care un utilizator a ales să beneficieze în trecut.

Sistemul stochează datele de rezervare într-o bază de date și actualizează automat informațiile legate de utilizator, precum locația preferată și facilitățile frecvent utilizate, pentru a îmbunătăți viitoarele recomandări. Această integrare între recomandări și rezervare asigură o experiență ce dorește a fi adaptată continuu, după comportamentul utilizatorului.

Aici hotelurile sunt așezate în ordinea scorului oferit de algoritmi de recomandare. Putem observa cum, pentru userul „Renter”, primul hotel este „Hotel Central”. Acest lucru este datorat faptului că acesta a oferit acelui hotel o recenzie de 5 stele. De asemenea, fiind doar un singur hotel recomandat, putem observa că recomandările sunt destul de slabe.

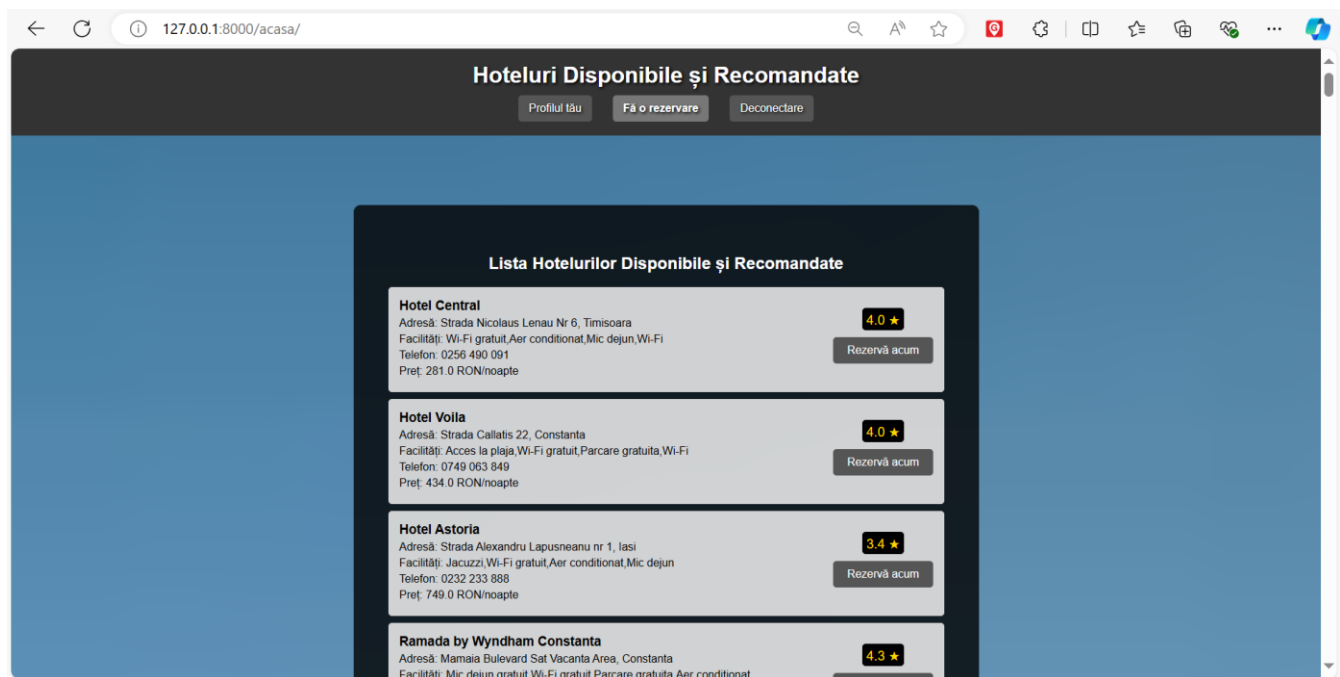


Figura 5.1.9 Pagina de Recomandări

#### 5.1.4 Pagina profilului de utilizator tip „Landlord”

Pagina profilului de utilizator pentru un "Landlord" este locul unde proprietarii de hoteluri pot gestiona și vizualiza cu ușurință toate proprietățile pe care le administrează. În acest caz, avem contul „LandlordTest”. La accesarea paginii, utilizatorul este întâmpinat cu un mesaj personalizat de bun venit, urmat de o listă a hotelurilor adăugate în sistem. Fiecare hotel este prezentat cu detalii esențiale, precum adresa, locația, facilitățile disponibile, numărul de telefon și prețul pe noapte, oferind astfel o privire de ansamblu clară și completă.

În plus, utilizatorul are opțiunea de a edita informațiile hotelului direct din această pagină, facilitând astfel actualizarea rapidă a datelor. Această funcționalitate este esențială pentru menținerea informațiilor corecte și actualizate, asigurându-se că potențialii clienți primesc mereu detalii precise și relevante despre hotelurile listate. Astfel, pagina profilului "Landlord" oferă un control facil asupra managementului proprietăților și centralizează informațiile esențiale.



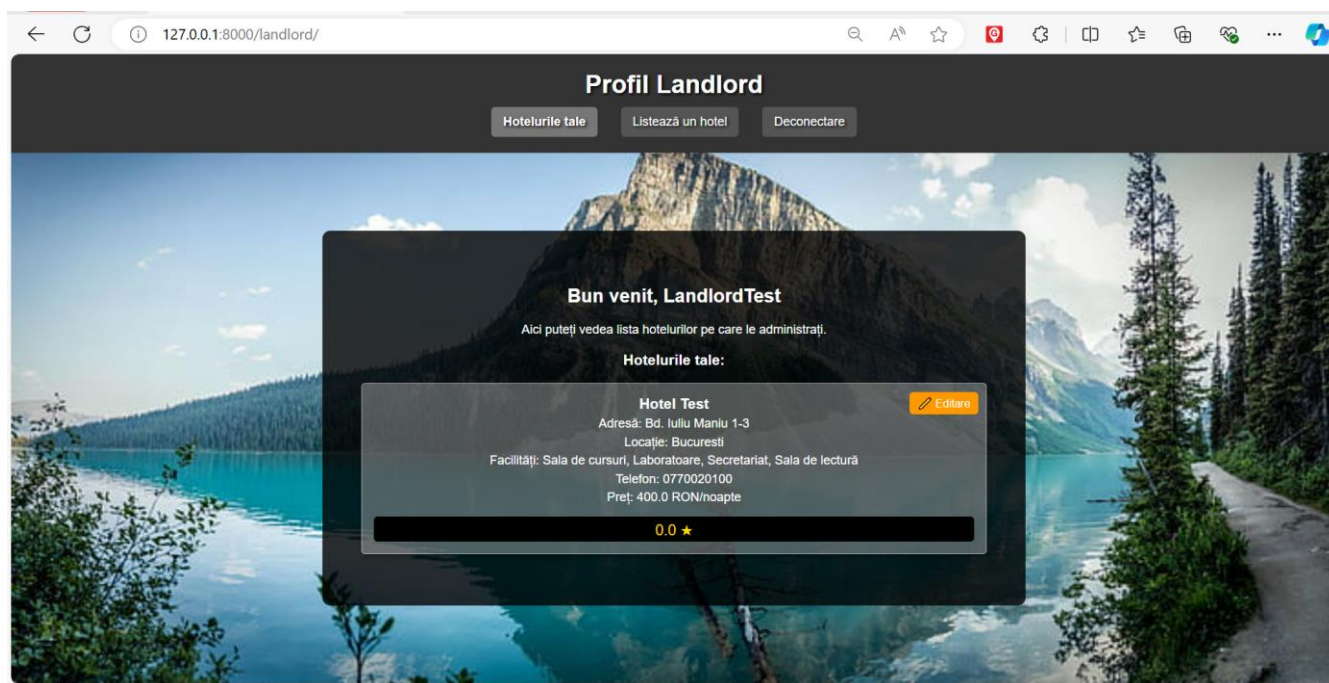


Figura 5.1.10 Pagina Profilului de Proprietar

Editarea datelor se face prin intermediul unui form, ce poate fi accesat într-un mod intuitiv, prin apăsarea butonului „Editare”.

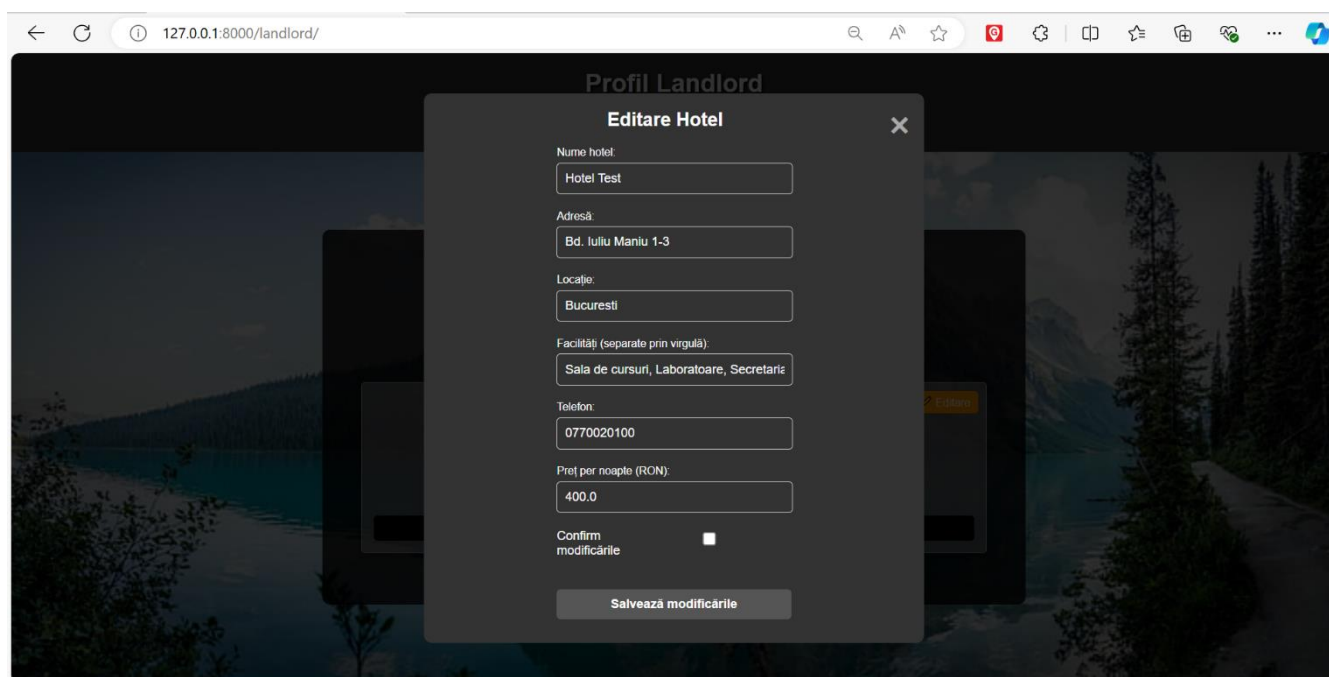


Figura 5.1.11 Formularul de Editare a Caracteristicilor Unui Hotel



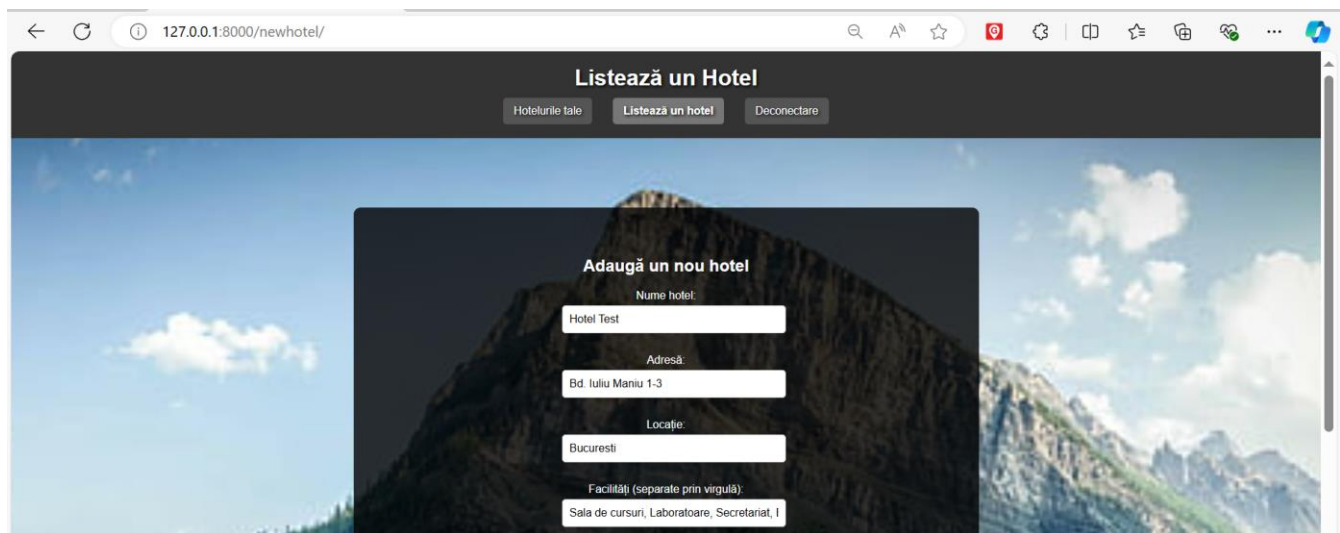
### 5.1.5 Pagina pentru listat unități de cazare

În această pagină, o persoană poate să își listeze o unitate de cazare.

Pagina de listare a hotelurilor permite utilizatorilor, în rol de proprietari, să adauge rapid și eficient noi hoteluri în platformă. Formularul este simplu și intuitiv, solicitând informații esențiale precum numele hotelului, adresa, locația, facilitățile disponibile, numărul de telefon și prețul pe noapte. După completarea detaliilor, utilizatorul poate trimite formularul pentru a adăuga hotelul în baza de date.

Un element cheie al acestei pagini este integrarea cu Google Maps, care permite utilizatorilor să selecteze precis locația hotelului pe hartă. Prin plasarea unui marker pe hartă, coordonatele geografice ale hotelului sunt preluate automat și incluse în formular, asigurând astfel acuratețea locației. Această funcționalitate nu doar simplifică procesul de adăugare a unui hotel, dar și îmbunătățește precizia datelor.

Mai apoi, aceste coordonate geografice sunt folosite pentru a putea oferi recomandări bazate pe geolocație unui utilizator de tip „Renter”.



The screenshot displays a web browser window with the address bar showing '127.0.0.1:8000/newhotel/'. The page has a dark header with the title 'Listează un Hotel' and three buttons: 'Hotelurile tale', 'Listează un hotel', and 'Deconectare'. The main content area features a background image of a mountain range. Overlaid on this is a form titled 'Adaugă un nou hotel'. The form contains the following fields: 'Nume hotel:' with the value 'Hotel Test', 'Adresă:' with 'Bd. Iuliu Maniu 1-3', 'Locație:' with 'Bucuresti', and 'Facilități (separate prin virgulă):' with 'Sala de cursuri, Laboratoare, Secretariat, I'.

Figura 5.1.12 Formular de Înregistrare a Unui Hotel

Figura 5.1.13 Formular de Înregistrare a Unui Hotel 2

## 5.2 Dezvoltare backend

În acest capitol se va prezenta atât configurarea mediului de lucru, cât și logica din spatele algoritmilor de recomandare.

### 5.2.1 Configurarea Mediului de Dezvoltare și a Bazelor de Date

Modulul *venv* permite crearea unui mediu virtual, cu propriul set independent de pachete instalate. Un mediu virtual este creat având la bază o instanță de Python existentă.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Facultate\LicentaPROIECT\Proiect> python -m venv

```

Figura 5.2.1 Pornire a Mediului Virtual (VENV – Virtual Environment)

Fiind utilizate dintr-un mediu virtual, instrumentele de instalare obișnuite, cum ar fi *pip*, instalează pachetele Python în directorul curent.

Configurarea serverului implică configurarea unui proiect Django, inclusiv crearea aplicațiilor necesare în cadrul proiectului. În această secțiune, se descrie modul în care setările Django sunt adaptate pentru a integra SQLite ca bază de date. SQLite este o bază de date relațională "serverless", integrată nativ în Django, ceea ce face configurarea simplă și eficientă. Se discută, de asemenea, procesul de aplicare a migrărilor bazei de date, asigurându-se astfel că schema este configurată corect și că toate tabelele și relațiile sunt create conform modelului definit în aplicație. Aceasta facilitează o dezvoltare rapidă și o administrare ușoară a bazei de date, fără necesitatea unui server de baze de date separat.

```
PS D:\Facultate\LicentaPROIECT\Proiect> pip install django
Requirement already satisfied: django in d:\facultate\licentaproiect\proiect\.venv\lib\site-packages (4.1.13)
Requirement already satisfied: asgiref<4,>=3.5.2 in d:\facultate\licentaproiect\proiect\.venv\lib\site-packages (from dja
ngo) (3.7.2)
Requirement already satisfied: sqlparse>=0.2.2 in d:\facultate\licentaproiect\proiect\.venv\lib\site-packages (from djang
o) (0.2.4)
Requirement already satisfied: tzdata in d:\facultate\licentaproiect\proiect\.venv\lib\site-packages (from django) (2024.
1)
```

Figura 5.2.2 Instalarea Modulului Django

Dupa configurarea mediului, am creat bazele de date specifice, pentru Renters, Landlords, Hotels si Sejur. Renter si Landlord sunt utilizatori derivați din clasa custom\_user. In timp ce se creeaza un cont, acesta poate fi ales a fi de Renter sau de Landlord, în funcție de tipul de utilizator dorit.

Câmpurile ce descriu un custom\_user sunt urmatoarele:

Field Name	Data Type
id	Int32
password	String
last_login	String
is_superuser	String
username	String
first_name	String
last_name	String
email	String
is_staff	boolean
Is_active	boolean
date_joined	datetime
name	String
role	String

Tabel 5. 1 Câmpurile ce descriu un utilizator

Persoanele de tip „Renter” sunt reprezentate de utilizatori comuni, oamenii care vor să își caute o unitate de cazare și să beneficieze de recomandări.

Figura 5.2.3 Baza de Date Renters

Modulul "Renter" se concentrează pe gestionarea preferințelor și a istoricului de călătorii al utilizatorului. Un utilizator de tip "Renter" are acces la câmpuri esențiale care descriu profilul său, cum ar fi name, preferences, centroid\_latitude, centroid\_longitude, și rented\_hotels. Aceste câmpuri permit sistemului să urmărească preferințele utilizatorului în scopul recomandărilor de hoteluri pe baza locațiilor anterioare și să gestioneze lista hotelurilor închiriate.

Câmpurile ce descriu un „Renter” sunt următoarele:

Field Name	Data Type
id	Int32
user_id	Int32
name	String
preferences	String
centroid_latitude	Float
centroid_longitude	Float
rented_hotels	JSON

Tabel 5.2 Câmpurile ce descriu un Renter

Modulul "Landlord" se concentrează pe gestionarea unităților de cazare listate de către proprietari. Un utilizator de tip "Landlord" are acces la câmpuri esențiale care descriu profilul său, cum ar fi name, email, și listed\_hotels. Aceste câmpuri permit sistemului să gestioneze eficient proprietățile deținute de proprietar și să asocieze fiecare cont de utilizator cu unitățile de cazare administrate, facilitând astfel procesul de listare și actualizare a hotelurilor pe platformă.

Câmpurile ce descriu un „Landlord” sunt următoarele:

Field Name	Data Type
id	Int32
user_id	Int32
name	String
email	String
listed_hotels	json

Tabel 5.3 Câmpurile ce descriu un Landlord

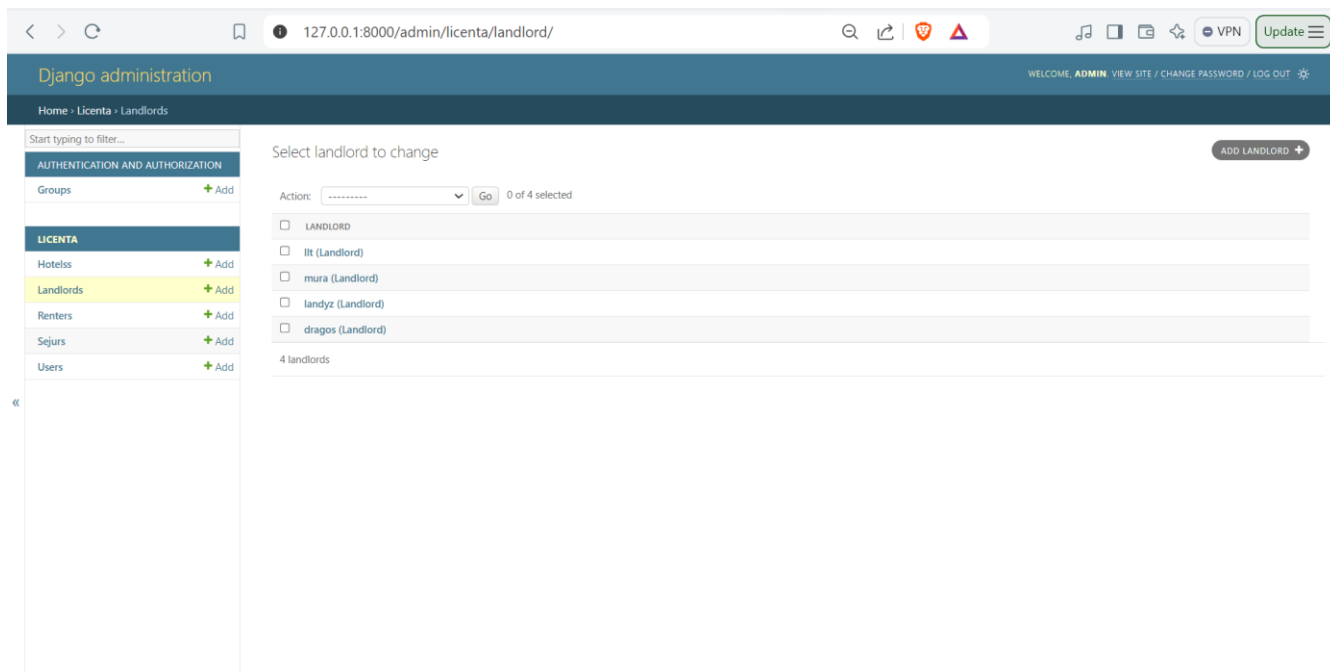


Figura 5.2.4 Baza de Date Landlords

În „hotels” se găsesc toate unitățile de cazare. Pentru a avea o bună bază, am adăugat 130 de unități deja existente. Datele privind hotelurile au fost colectate cu ajutorul instrumentului G Maps Extractor, care extrage informații din Google Maps (Procedeu numit web-scraping). Au fost obținute mai multe fișiere CSV pentru diferite regiuni, fiecare conținând detalii precum nume de hoteluri, adrese, facilități, numere de telefon și recenzii.

Aceste fișiere au fost ulterior concatenate, curățate și alterate, încât să nu apară duplicate, câmpuri inutile și date de care nu este nevoie. Caracterele speciale din adrese au fost înlocuite cu echivalentele lor standard, iar coloanele inutile au fost eliminate. Setul de date curățat a fost structurat și pregătit pentru a fi utilizat în scopul mării numărului total de hoteluri, pentru a putea genera recomandări calitative.

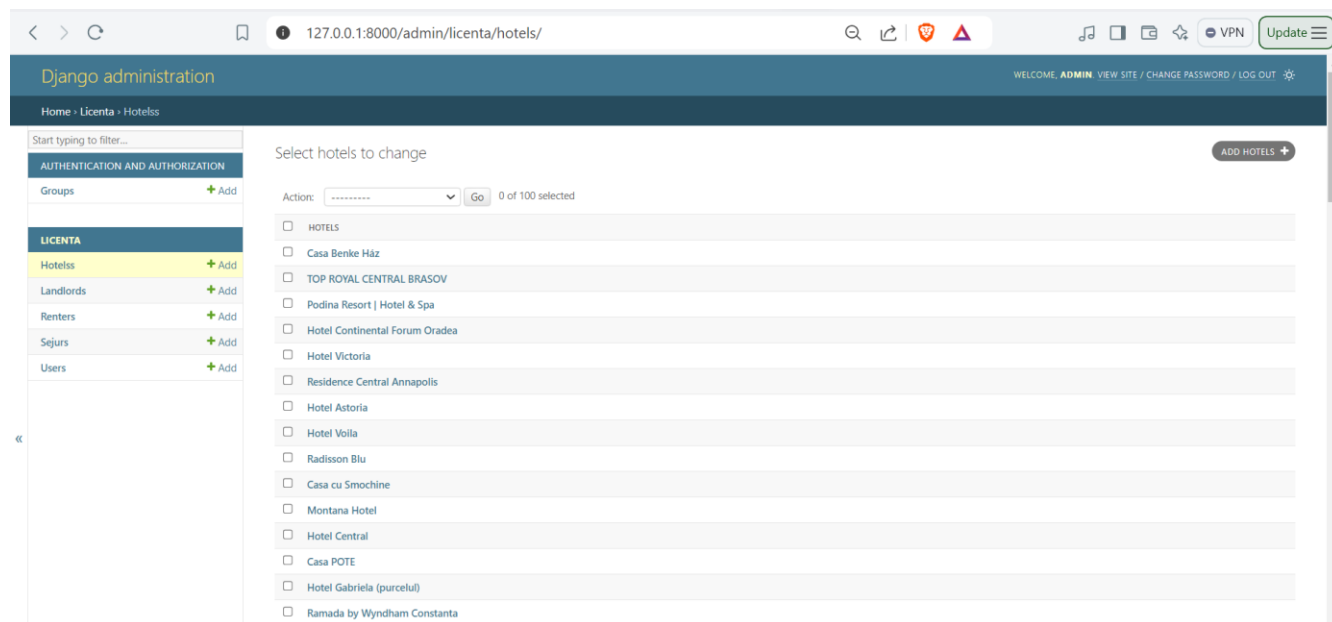


Figura 5.2.5 Baza de Date Hotels

Câmpurile ce descriu un hotel sunt următoarele:

Field Name	Data Type
id	Int32
name	String
address	String
location	String
amenities	JSON
phone	String
review_count	Int32
average_rating	Float
latitude	Float
longitude	Float

money	Float
-------	-------

Tabel 5.4 Câmpurile ce descriu un hotel

În proiectarea bazelor de date, o bază de date de tip mulți-la-mulți (M:N) apare atunci când mai multe înregistrări dintr-un tabel sunt asociate cu mai multe înregistrări dintr-un alt tabel. În scenariul nostru, fiecare user se poate caza la mai multe hoteluri dar fiecare hotel poate caza mai multe persoane simultan.

Pentru a soluționa această problemă, vom adăuga câmpul „Sejur”:

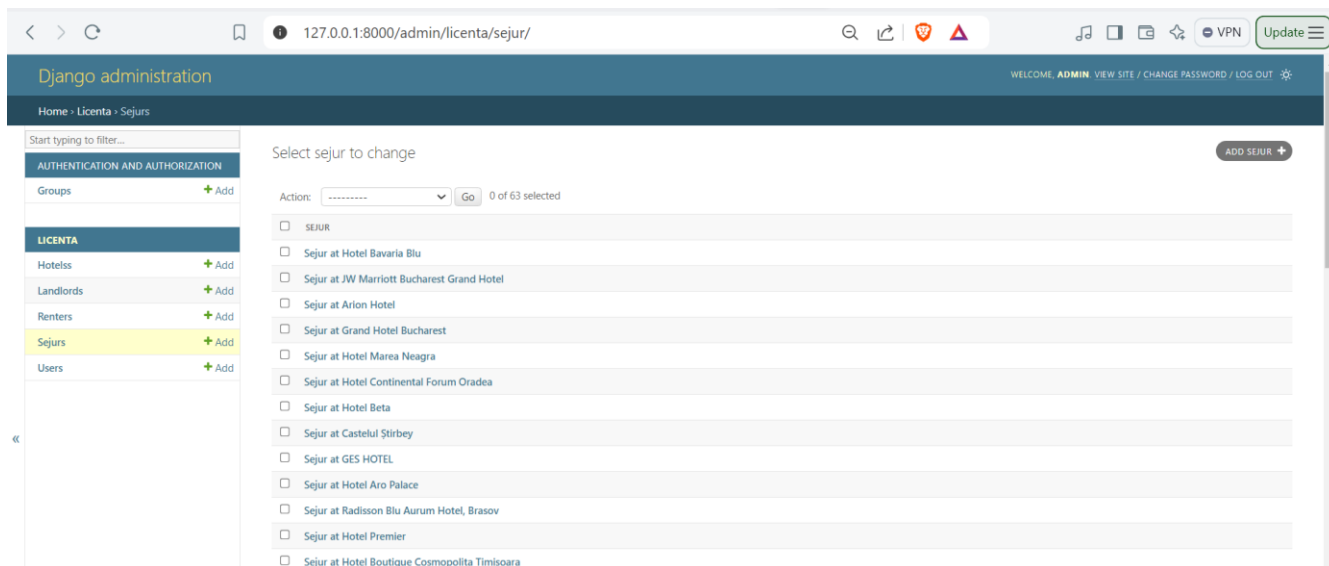


Figura 5.2.6 Baza de Date Sejururi

Câmpurile ce descriu un sejur sunt următoarele:

Name	Data Type
id	Int32
name	String
address	String
location	String
amenities	JSON
phone	String
review_count	Int32
average_rating	Float
latitude	Float
longitude	Float
money	Float

Tabel 5.5 Campurile ce descriu un sejur

## 5.2.2 Implementarea Algoritmilor de Recomandare

Această secțiune cuprinde logica de implementare a algoritmilor folosiți în scopul recomandării unei unități de cazare.

Primul pas a fost definirea modelelor conform cu schema bazei de date.

```
1 from django.contrib.auth.models import AbstractUser
2 from django.db import models
3
4 class CustomUser(AbstractUser):
5     RENTER = 'renter'
6     LANDLORD = 'landlord'
7
8     ROLE_CHOICES = [
9         (RENTER, 'Renter'),
10        (LANDLORD, 'Landlord'),
11    ]
12    email = models.EmailField(unique=True)
13    name = models.CharField(max_length=100, null=True, blank=True)
14    role = models.CharField(max_length=10, choices=ROLE_CHOICES, default=RENTER)
15
16    def save(self, *args, **kwargs):
17        super().save(*args, **kwargs)
18
19    def __str__(self):
20        return self.username
```

Figura 5.2.7 Definirea Modelelor 1

```
23 class Renter(models.Model):
24     user = models.OneToOneField(CustomUser, on_delete=models.CASCADE, related_name='renter')
25     name = models.CharField(max_length=100, default="")
26     preferences = models.JSONField(null=True, blank=True)
27     centroid_latitude = models.FloatField(null=True, blank=True)
28     centroid_longitude = models.FloatField(null=True, blank=True)
29     rented_hotels = models.JSONField(null=True, blank=True)
30
31     def __str__(self):
32         return f"{self.user.username} (Renter)"
33
34     def update_preference(self, preference):
35         """Actualizează frecvența unei preferințe specifice."""
36         if not self.preferences:
37             self.preferences = {}
38
39         if preference in self.preferences:
40             self.preferences[preference] += 1
41         else:
42             self.preferences[preference] = 1
43
44         self.save()
```

Figura 5.2.8 Definirea Modelelor 2



```

46 def update_location(self, latitude, longitude):
47     """Actualizează centroidul geografic pe baza unui nou hotel."""
48     if self.centroid_latitude is None or self.centroid_longitude is None:
49         self.centroid_latitude = latitude
50         self.centroid_longitude = longitude
51     else:
52         # Calculul centroidului folosind media aritmetică
53         n = len(self.rented_hotels) if self.rented_hotels else 1
54         self.centroid_latitude = (self.centroid_latitude * n + latitude) / (n + 1)
55         self.centroid_longitude = (self.centroid_longitude * n + longitude) / (n + 1)
56
57     self.save()
58
59
60 class Landlord(models.Model):
61     user = models.OneToOneField(CustomUser, on_delete=models.CASCADE, related_name='landlord')
62     name = models.CharField(max_length=100, default="")
63     email = models.EmailField(unique=True)
64     listed_hotels = models.JSONField(null=True, blank=True)

```

Figura 5.2.9 Definirea Modelelor 3

```

66 def __str__(self):
67     return f"{self.user.username} (Landlord)"
68
69 import uuid
70
71 class Hotels(models.Model):
72     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False) # Utilizează UUIDField
73     name = models.CharField(max_length=255)
74     address = models.CharField(max_length=255)
75     location = models.CharField(max_length=255)
76     amenities = models.JSONField()
77     phone = models.CharField(max_length=20)
78     review_count = models.IntegerField(default=0)
79     average_rating = models.FloatField(default=0.0)
80     latitude = models.FloatField()
81     longitude = models.FloatField()
82     money = models.FloatField(default=0.0)
83
84     def __str__(self):
85         return self.name
86
87     class Meta:
88         db_table = 'hotels'

```

Figura 5.2.10 Definirea Modelelor 4

```

90 class Sejur(models.Model):
91     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
92     renter = models.ForeignKey(Renter, on_delete=models.CASCADE)
93     hotel = models.ForeignKey(Hotels, on_delete=models.CASCADE)
94     start_date = models.DateField()
95     end_date = models.DateField()
96     number_of_people = models.IntegerField()
97     money = models.FloatField(default=0.0)
98     rating = models.FloatField(null=True, blank=True)
99     feedback = models.TextField(null=True, blank=True)
100
101     def __str__(self):
102         return f'Sejur at {self.hotel.name}'
103
104     class Meta:
105         db_table = 'sejur'

```

Figura 5.2.11 Definirea Modelelor 5

### 5.2.2.1. Filtrul Bazat pe Conținut (CBF)

Într-un sistem de recomandare a unităților de cazare, filtrarea bazată pe conținut sugerează hoteluri pe baza caracteristicilor acestora și a preferințelor anterioare ale utilizatorilor. Fiecare hotel este descris de un set de atribute, cum ar fi locația, facilitățile (de exemplu, Wi-Fi, piscină, spa), intervalul de preț și tipul de cameră. Profilurile utilizatorilor sunt create pe baza sejururilor lor anterioare la hotel sau a preferințelor explicite pe care le-au furnizat, care surprind gusturile lor în materie de caracteristici hoteliere.

Filtrarea bazată pe conținut în sistemele de recomandare a hotelurilor este avantajoasă, deoarece poate recomanda hoteluri de nișă sau mai puțin populare care corespund exact preferințelor unui utilizator, evitând astfel un timp mare de căutare a unui loc potrivit. Cu toate acestea, necesită date detaliate și precise despre hoteluri și s-ar putea să nu surprindă pe deplin diversitatea preferințelor utilizatorilor noi, deoarece aceștia nu au interacționat cu o gamă largă de hoteluri.

Primul pas constă în importarea pachetelor necesare.

```

1  import math
2  import unicodedata
3  import pandas as pd
4  import numpy as np
5  from collections import Counter
6  from sklearn.feature_extraction.text import TfidfVectorizer
7  from sklearn.metrics.pairwise import linear_kernel
8  from .models import CustomUser, Hotels, Sejur

```

Figura 5.2.12 Importarea Bibliotecilor Necesare

A fost creată funcția „get\_recommendations”, care primește ca argument ID-ul unui utilizator și are ca scop returnarea unei liste de hoteluri recomandate, ordonate după similaritatea cu preferințele utilizatorului.

Funcția începe prin extragerea preferințelor utilizatorului și combinarea acestora cu locațiile preferate într-un șir de text unificat. Dacă utilizatorul are preferințe definite, aceste informații sunt utilizate pentru a construi un profil de preferințe care va fi folosit în etapa de recomandare.

Următorul pas implică colectarea datelor despre hoteluri din baza de date și construirea unui DataFrame care conține ID-urile hotelurilor și descrierile acestora, combinate din facilitățile oferite și locațiile hotelurilor.

Aceste descrieri sunt apoi vectorizate folosind TF-IDF, eliminând cuvintele comune în limba română pentru a preveni influențarea negativă a rezultatelor. Se calculează apoi similitudinea cosinusului între vectorul de preferințe al utilizatorului și vectorii de descriere ai hotelurilor. Hotelurile sunt ordonate în funcție de scorurile de similaritate, iar lista finală de hoteluri recomandate este obținută prin filtrarea ID-urilor hotelurilor cu cele mai mari scoruri de similitudine, care sunt returnate sub forma unui queryset Django.

```
30 def get_recommendations(user_id):
31     """Oferă recomandări bazate pe cuvinte cheie pentru un utilizator specific."""
32     user = CustomUser.objects.get(pk=user_id)
33
34     if hasattr(user, 'renter') and user.renter.preferences:
35         user_preferences = preferences_to_string(user.renter.preferences)
36         user_locations = ""
37
38         combined_string = f"{user_preferences} {user_locations}"
39     else:
40         return Hotels.objects.none()
41
42     hotels = Hotels.objects.all()
43
44     hotel_data = [{
45         'hotel_id': hotel.id,
46         'description': remove_diacritics(f"{preferences_to_string(hotel.amenities)} {hotel.location}")
47     } for hotel in hotels]
48     df = pd.DataFrame(hotel_data)
49
50     tfidf = TfidfVectorizer(stop_words=romanian_stop_words)
51     tfidf_matrix = tfidf.fit_transform(df['description'])
52
53     user_pref_vector = tfidf.transform([remove_diacritics(combined_string)])
54
55     cosine_sim = linear_kernel(user_pref_vector, tfidf_matrix)
56
57     sim_scores = list(enumerate(cosine_sim[0]))
58     sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
59     top_hotels = [df.iloc[i[0]].hotel_id for i in sim_scores]
60
61     recommended_hotels = Hotels.objects.filter(id__in=top_hotels)
62
63     return recommended_hotels
```

Figura 5.2.13 Logica Filtrului Bazat pe Conținut

### 5.2.2.2. Algorimi de recomandare bazați pe rating și feedback

Algoritmii bazați pe ratinguri și feedback reprezintă un pilon central în sistemele de recomandare, având capacitatea de a influența puternic deciziile utilizatorilor. Totuși, pentru a oferi recomandări precise, este esențial să nu ne bazăm doar pe valoarea brută a ratingurilor, ci și pe numărul de recenzii. De exemplu, un hotel cu un rating de 4.7/5, obținut din 29 de recenzii, nu este la fel de relevant sau de încredere precum un hotel cu un rating de 4.5, dar susținut de 10.000 de recenzii.

Pentru a ține cont de acest aspect, am implementat în cod funcția "weighted\_rating", care ajustează scorurile pentru a reflecta atât ratingul mediu, cât și volumul de feedback. Această funcție primește ca argument un user\_id și, în primul pas, verifică dacă utilizatorul a fost cazat la anumite hoteluri, extrăgând o listă de hoteluri relevante. Apoi, folosind un dictionary comprehension, funcția extrage recenziile acordate fiecărui hotel.

Așadar, se calculează un rating ponderat folosind formula de ajustare Bayesiană, care ia în considerare atât ratingul mediu al hotelului, cât și numărul de recenzii. Hotelurile cu mai multe recenzii primesc un scor mai de încredere, chiar dacă ratingul lor mediu este puțin mai scăzut, în timp ce hotelurile cu puține recenzii nu sunt supraevaluate. Acest sistem de ponderare ajută la evitarea situațiilor în care hotelurile noi sau cele cu puține recenzii obțin scoruri fals ridicate. Rezultatul este o listă de hoteluri recomandate, ordonată în funcție de relevanță și încredere.

```
68 def get_rating_based_recommendations(user_id, min_reviews=5):
69     """Recomandă hoteluri bazate pe rating-urile medii ponderate și pe rating-urile date de utilizator."""
70     user = CustomUser.objects.get(pk=user_id)
71     user_stays = Sejur.objects.filter(renter=user.renter)
72     user_ratings = {stay.hotel.id: stay.rating for stay in user_stays if stay.rating is not None}
73
74     hotels = Hotels.objects.all()
75
76     # Parametrii pentru media Bayesiana
77     k = 10 # Factor de ponderare
78     C = np.mean([hotel.average_rating for hotel in hotels if hotel.review_count > 0])
79
80     hotel_ratings = {}
81     for hotel in hotels:
82         if hotel.review_count > 0:
83             weighted_rating = (hotel.review_count / (hotel.review_count + k)) * hotel.average_rating + (k / (hotel.review_count + k)) * C
84         else:
85             weighted_rating = hotel.average_rating
86
87         # Ajustează în funcție de user rating
88         if hotel.id in user_ratings:
89             personal_rating = user_ratings[hotel.id]
90             combined_rating = 0.7 * personal_rating + 0.3 * weighted_rating
91         else:
92             combined_rating = weighted_rating
93
94         hotel_ratings[hotel.id] = combined_rating
95
96     topRatedHotels = sorted(hotel_ratings.items(), key=lambda x: x[1], reverse=True)
97     top_hotels_ids = [hotel_id for hotel_id, rating in topRatedHotels]
98
99     recommended_hotels = Hotels.objects.filter(id__in=top_hotels_ids)
100
101     return recommended_hotels
```

Figura 5.2.14 Logica Algoritmilor ce Decid În Funcție de Rating și Feedback

### 5.2.2.3. Algorimi de recomandare bazați pe geolocație

Algoritmii bazați pe geolocalizare pot fi utilizați în mod eficient într-un sistem de recomandare a hotelurilor prin valorificarea coordonatelor geografice ale hotelurilor, pentru a oferi recomandări personalizate.

Analizând aceste informații spațiale, sistemul poate acorda prioritate recomandării hotelurilor care se află în anumite zone de interes. În plus, algoritmi de geolocalizare pot încorpora preferințele utilizatorului pentru anumite locații, cum ar fi centrele orașelor, atracțiile turistice sau cartierele de afaceri, sporind astfel relevanța recomandărilor.

Algoritmi pot, de asemenea, să ia în considerare datele istorice de localizare pentru a identifica tipare în comportamentul utilizatorului, cum ar fi destinațiile frecvente de călătorie, oferind astfel sugestii de hoteluri mai precise și mai adecvate din punct de vedere contextual.

Algoritmul implementat se folosește de poziția geografică a unității de cazare, pentru a putea recomanda unei persoane o unitate de cazare similară cu istoricul acesteia, din punct de vedere geografic.

Distanța este calculată cu ajutorul formulei euclidiene.

```
#calculam functia de distanta euclidiană
def calculate_distance(lat1, lon1, lat2, lon2):
    return math.sqrt((lat2 - lat1)**2 + (lon2 - lon1)**2)
```

Figura 5.2.15 Definirea Unei Funcții ce Calculează Distanța Euclidiană

Funcția `recommend_hotels_based_on_geolocation` returnează o listă de obiecte de tip `Hotels`, sortate în funcție de distanța lor față de centroidul geografic al utilizatorului (`renter`). Această listă conține hotelurile cele mai apropiate de locațiile pe care utilizatorul vizitat anterior, oferind astfel recomandări bazate pe proximitate.

```
120 def recommend_hotels_based_on_geolocation(renter):
121     """Recomandă hoteluri bazate pe proximitatea față de centroidul geografic al utilizatorului."""
122     if renter.centroid_latitude is None or renter.centroid_longitude is None:
123         return Hotels.objects.none()
124
125     hotels = Hotels.objects.all()
126     distances = []
127
128     for hotel in hotels:
129         distance = calculate_distance(renter.centroid_latitude, renter.centroid_longitude, hotel.latitude, hotel.longitude)
130         distances.append((hotel, distance))
131
132     distances.sort(key=lambda x: x[1]) # Sortează după distanță
133
134     recommended_hotels = [hotel for hotel, _ in distances]
135
136     return recommended_hotels
```

Figura 5.2.16 Calcularea Unui Centroid Geografic

#### 5.2.2.4. Integrarea algoritmilor de recomandare

Ultimul pas în finalizarea algoritmului de recomandari este integrarea celor 3 algoritmi.

Acest lucru se face în doua etape: normalizarea și combinarea propriu-zisă.

Funcția `normalize_scores` este responsabilă pentru ajustarea scorurilor brute într-un interval standardizat între 0 și 1. Acest proces de normalizare este esențial deoarece permite compararea echitabilă a scorurilor provenite din surse diferite, cum ar fi preferințele utilizatorului, ratingurile hotelurilor și distanța geografică.

Normalizarea asigură că fiecare tip de scor contribuie în mod proporțional la calculul final al scorului combinat, evitând astfel situațiile în care un scor cu o valoare foarte mare sau foarte mică ar putea domina în mod nejustificat rezultatul final.

```
139 def normalize_scores(scores):
140     """Normalizează o listă de scoruri între 0 și 1."""
141     if not scores:
142         return [0] * len(scores)
143
144     min_score = min(scores)
145     max_score = max(scores)
146
147     if min_score == max_score:
148         return [1] * len(scores)
149
150     return [(score - min_score) / (max_score - min_score) for score in scores]
```

Figura 5.2.17 Normalizarea

Funcția `combined_recommendations` aduce împreună scorurile obținute din trei surse principale: preferințele de conținut (CBF - Content-Based Filtering), ratingurile hotelurilor, și poziția geografică.

Fiecare dintre aceste scoruri este normalizat și ponderat în funcție de importanța sa relativă, specificată de parametrii „weights” – aceștia pot fi ajustați, în scop de fine-tuning. În funcție de scorurile normalizate, se calculează un scor combinat pentru fiecare hotel, care reflectă cât de bine se potrivește hotelul respectiv cu preferințele și nevoile utilizatorului.

Acest scor combinat este ajustat ulterior în funcție de ratingurile date de utilizator hotelurilor, aplicând bonusuri sau penalizări pentru a reflecta în mod corect experiențele anterioare.

În final, hotelurile sunt ordonate după acest scor combinat, iar utilizatorului îi sunt prezentate cele mai relevante opțiuni, îmbunătățind semnificativ calitatea și personalizarea recomandărilor.

```

155 def combined_recommendations(user_id, weights=(0.4, 0.3, 0.3), min_user_rating=3):
156     renter = CustomUser.objects.get(pk=user_id).renter
157     user_stays = Sejur.objects.filter(renter_id=renter.id)
158     rated_hotels = {stay.hotel.id: stay.rating for stay in user_stays if stay.rating is not None}
159
160     # Apply default rating if hotel is not rated
161     all_hotel_ids = Hotels.objects.values_list('id', flat=True)
162     for hotel_id in all_hotel_ids:
163         if hotel_id not in rated_hotels:
164             rated_hotels[hotel_id] = 4 # Default to 4 - Daca da 4 stele, inmulțirea se face cu 1
165
166     cbf_hotels = get_recommendations(user_id)
167     rating_hotels = get_rating_based_recommendations(user_id)
168     geo_hotels = recommend_hotels_based_on_geolocation(renter)
169
170     cbf_weight, rating_weight, geo_weight = weights
171
172     cbf_scores = {hotel.id: index for index, hotel in enumerate(cbf_hotels)} if cbf_hotels else {}
173     cbf_norm = normalize_scores(list(cbf_scores.values())) if cbf_scores else []
174
175     rating_scores = {hotel.id: index for index, hotel in enumerate(rating_hotels)} if rating_hotels else {}
176     rating_norm = normalize_scores(list(rating_scores.values())) if rating_scores else []
177
178     geo_scores = {hotel.id: index for index, hotel in enumerate(geo_hotels)} if geo_hotels else {}
179     geo_norm = normalize_scores(list(geo_scores.values())) if geo_scores else []

```

Figura 5.2.18 Crearea Unui Model Combinat 1

```

181 combined_scores = {}
182 for hotel in Hotels.objects.filter(id__in=set(cbf_scores.keys()) | set(rating_scores.keys()) | set(geo_scores.keys())):
183     cbf_score = cbf_norm[cbf_scores[hotel.id]] if hotel.id in cbf_scores else 0
184     rating_score = rating_norm[rating_scores[hotel.id]] if hotel.id in rating_scores else 0
185     geo_score = geo_norm[geo_scores[hotel.id]] if hotel.id in geo_scores else 0
186
187     combined_score = (cbf_weight * cbf_score) + (rating_weight * rating_score) + (geo_weight * geo_score)
188     rating = rated_hotels.get(hotel.id, 4) # Default to 4 if no explicit rating
189     if rating == 4:
190         pass
191     elif rating < 4:
192         if rating == 3:
193             combined_score *= 0.7
194         elif rating == 2:
195             combined_score *= 0.5
196         elif rating == 1:
197             combined_score *= 0.3
198         combined_score *= 20
199
200     combined_scores[hotel.id] = combined_score
201
202 sorted_hotels = sorted(combined_scores.items(), key=lambda x: x[1], reverse=True)
203 top_hotels_ids = [hotel_id for hotel_id, score in sorted_hotels]
204
205 # Creating a Case for ordering
206 ordering = Case(
207     *[When(id=hotel_id, then=pos) for pos, hotel_id in enumerate(top_hotels_ids)],
208     output_field=IntegerField()
209 )
210
211 # Return the hotels ordered by the calculated score
212 recommended_hotels = Hotels.objects.filter(id__in=top_hotels_ids).order_by(ordering)
213
214 return recommended_hotels

```

Figura 5.2.19 Crearea Unui Model Combinat 2

## 6. Testare

Testarea este un aspect fundamental al procesului de dezvoltare software, esențial pentru asigurarea calității, fiabilității și performanței aplicațiilor.

Testarea implică evaluarea sistematică a software-ului pentru a detecta defectele, a verifica funcționalitatea și a valida faptul că software-ul îndeplinește cerințele specificate și atinge așteptările dezvoltatorului.

Prin identificarea și abordarea preventivă a problemelor, testarea ajută la prevenirea escaladării problemelor, reduce costurile și asigură un produs mai stabil și mai robust. În general, testarea este esențială pentru furnizarea de software de înaltă calitate, care îndeplinește așteptările utilizatorilor și standardele industriei.

### 6.1 Metode de Testare

Ca o primă metodă de testare, am folosit testarea manuală. Am rugat două persoane să testeze aplicația, să își facă un profil și să încerce să găsească eventuale erori sau bug-uri. Acest lucru s-a dovedit a fi foarte folositor deoarece s-au găsit erori vizuale și elemente centrate necorespunzător.

Apoi, am implementat testarea automată, folosind teste de stres cu Locust. Locust este un instrument open-source dedicat testării performanței și a încărcării aplicațiilor web, care suportă protocoale precum HTTP. Acesta este conceput să fie flexibil și ușor de utilizat, fiind foarte popular printre dezvoltatori, deoarece testele pot fi scrise în Python, ceea ce permite o personalizare detaliată a scenariilor de testare.

Unul dintre avantajele Locust este faptul că poate simula un număr mare de utilizatori care accesează simultan aplicația, oferind informații esențiale despre cum aceasta gestionează un volum mare de trafic. Testele pot fi rulate fie din linia de comandă (CLI), fie folosind interfața grafică web a Locust, care permite monitorizarea în timp real a unor metrici cheie, precum rata de transfer, timpii de răspuns și erorile întâlnite.

În plus, rezultatele pot fi exportate pentru o analiză ulterioară detaliată, oferind o imagine clară asupra comportamentului aplicației sub stres. Instalarea și configurarea Locust sunt simple și rapide, putând fi realizate direct din terminal. Instalarea acestuia se face simplu, în terminal.

```
PS D:\Facultate\LicentaPROIECT\Proiect\licenta> pip install locust
Collecting locust
  Downloading locust-2.29.1-py3-none-any.whl.metadata (7.4 kB)
```

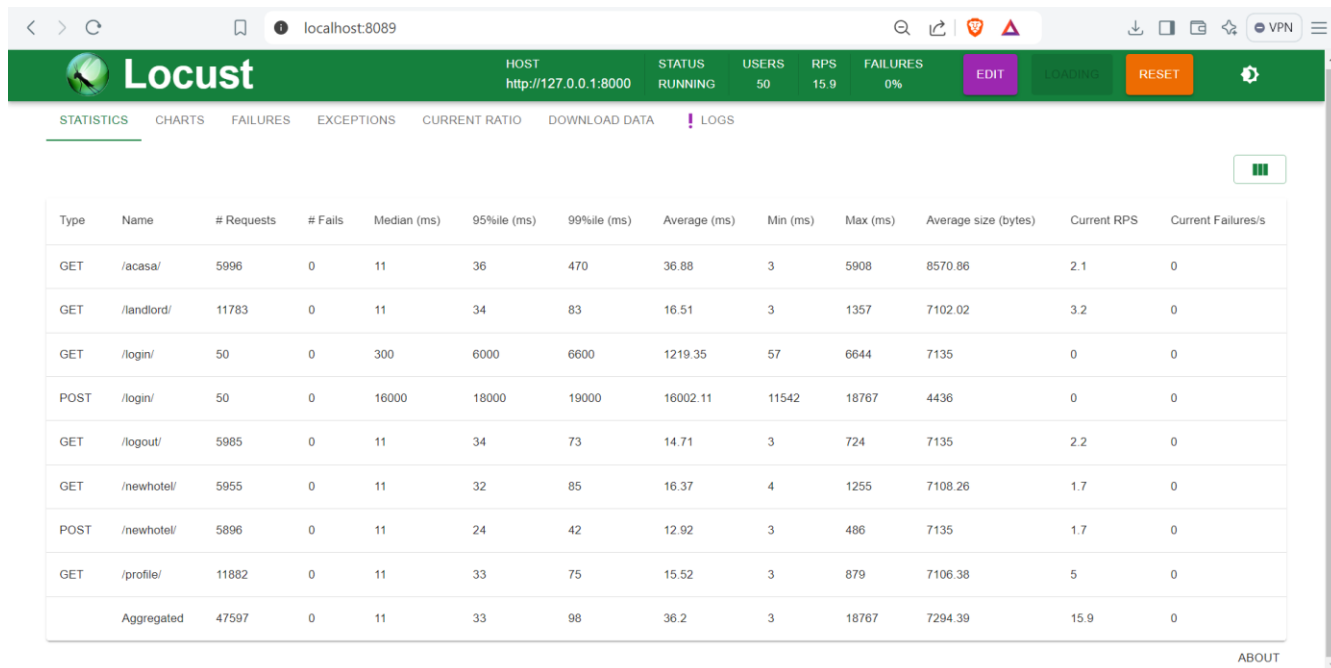
Figura 6.1 Descărcarea Uneltei Locust

După configurarea și instalarea Locust, utilizarea sa s-a dovedit a fi destul de intuitivă. Un aspect esențial al testelor a fost setarea numărului de utilizatori care vor accesa simultan aplicația, precum și ajustarea unui parametru important numit Ramp-Up. Acest parametru controlează creșterea treptată a sarcinii asupra sistemului, simulând un scenariu realist în care numărul de cereri crește progresiv în loc să fie aplicat brusc.

Această abordare ajută la observarea modului în care aplicația reacționează la o încărcare care se dezvoltă în timp, reflectând comportamente reale ale utilizatorilor.



Pentru aceste teste a fost folosită interfața grafică a Locust, alegând să se evalueze exclusiv secțiunile de înregistrare și autentificare ale aplicației. Acest lucru a permis să monitorizăm performanța acestor componente critice în condiții de încărcare, fără a supune întregul sistem la teste inutile în această etapă.



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/acasa/	5996	0	11	36	470	36.88	3	5908	8570.86	2.1	0
GET	/landlord/	11783	0	11	34	83	16.51	3	1357	7102.02	3.2	0
GET	/login/	50	0	300	6000	6600	1219.35	57	6644	7135	0	0
POST	/login/	50	0	16000	18000	19000	16002.11	11542	18767	4436	0	0
GET	/logout/	5985	0	11	34	73	14.71	3	724	7135	2.2	0
GET	/newhotel/	5955	0	11	32	85	16.37	4	1255	7108.26	1.7	0
POST	/newhotel/	5896	0	11	24	42	12.92	3	486	7135	1.7	0
GET	/profile/	11882	0	11	33	75	15.52	3	879	7106.38	5	0
Aggregated		47597	0	11	33	98	36.2	3	18767	7294.39	15.9	0

Figura 6.2 Rezultatul Testelor de Stres

Rezultatele testelor de performanță efectuate cu Locust au evidențiat comportamentul diferitelor endpoint-uri din aplicație sub sarcină. Testele au arătat că endpoint-ul /acasa/ a gestionat un total de 5.996 cereri, fără eșecuri, cu un timp de răspuns median de 11 ms și un timp de răspuns mediu de 36,88 ms. Rata de cereri per secundă (RPS) a fost de 2,1, ceea ce indică o capacitate stabilă de a răspunde cererilor de acest tip fără întârzieri semnificative.

Endpoint-ul /landlord/ a procesat 11.783 cereri, de asemenea fără eșecuri. Timpul de răspuns median a fost similar, de 11 ms, cu un timp de răspuns mediu de 16,51 ms, iar timpul de răspuns pentru 99% dintre cereri a fost sub 83 ms. Rata curentă de cereri a fost de 3,2 cereri pe secundă, ceea ce indică o performanță foarte bună și o scalabilitate eficientă.

În ceea ce privește endpoint-ul /login/, testele au arătat o diferență notabilă în timpii de răspuns. Acesta a gestionat 50 de cereri GET, cu un timp de răspuns median de 300 ms și un timp de răspuns mediu de 1.219,35 ms. Cea mai mare întârziere a fost de 6.644 ms, ceea ce sugerează că ar putea exista oportunități de optimizare în acest punct. Cererile POST pentru același endpoint au avut un timp de răspuns median mult mai mare, de 16.000 ms, cu o medie similară, ceea ce arată o latență foarte ridicată în procesul de autentificare.

Endpoint-ul /logout/ a procesat 5.985 cereri, cu un timp de răspuns median foarte mic, de 11 ms, și un timp de răspuns mediu de 14,71 ms, indicând o funcționare eficientă și rapidă. Endpoint-ul /newhotel/, atât pentru cererile GET, cât și POST, a avut timpi de răspuns asemănători. Cererile GET au avut un timp de răspuns median de 11 ms și o medie de 16,37 ms, iar cererile POST au fost procesate cu

un timp median de 11 ms și o medie de 12,92 ms, ceea ce sugerează un comportament consistent și stabil pentru această funcționalitate.

Pentru endpoint-ul /profile/, testele au arătat o gestionare a 11.882 cereri, cu un timp de răspuns median de 11 ms și o medie de 15,52 ms. Acest endpoint s-a comportat eficient sub sarcină, având o rată de cereri per secundă de 5, fără eșecuri.

În total, au fost procesate 47.597 cereri fără eșecuri, cu un timp de răspuns median agregat de 11 ms și un timp mediu de răspuns de 36,2 ms. Performanța generală a sistemului este stabilă, cu excepția endpoint-ului de autentificare, care necesită optimizare pentru a reduce timpii de răspuns ridicați în cererile POST.

Testele au fost rulate pentru puțin peste două ore. Pentru o interpretare vizuală, se observă Figura 6.3.



Figura 6.3 Reprezentarea Grafică a Testelor de Stres



## 7. Concluzii

În concluzie, sistemul inteligent de recomandări implementat în această aplicație reprezintă o inovație importantă în domeniul rezervărilor online din sectorul turismului. Acesta valorifică algoritmi avansați de recomandare și date istorice, geografice și preferințele utilizatorilor, oferind soluții personalizate care îmbunătățesc considerabil experiența utilizatorilor. Platforma se remarcă printr-o interfață intuitivă și ușor de utilizat, care permite accesul simplu și rapid la opțiunile de cazare recomandate, indiferent de nivelul tehnic al utilizatorului.

Testele de performanță au confirmat eficiența sistemului, demonstrând capacitatea sa de a genera recomandări relevante și precise, bazate pe criterii diverse. În plus, rezultatele obținute în faza de testare sunt promițătoare, ceea ce sugerează că sistemul poate deveni și mai performant pe măsură ce numărul opțiunilor de cazare crește.

Scalabilitatea rămâne o prioritate importantă pentru viitor, iar capacitatea de a integra noi funcționalități asigură flexibilitatea necesară pentru adaptarea la cerințele viitoare ale pieței. Astfel, acest proiect deschide drumul către o utilizare tot mai largă a tehnologiei în turism, oferind o platformă stabilă, sigură și eficientă, capabilă să răspundă în mod dinamic nevoilor utilizatorilor și să ofere o experiență optimizată de rezervare.

## Bibliografie

- [1]. [Thiengburanathum, P. \(2018\). \*An Intelligent Destination Recommendation System for Tourists\*. U.K.: Department of Computing and Informatics, Faculty of Science and Technology.](#)
- [2]. [Nica, M. \(2023\). Vacanțele reprezintă în continuare un răsfăț pentru români. \*Agerpres\*.](#)
- [3]. [Evan P., David O.; CORRELATING ECOMMERCE WEB PAGE LOAD TIMES WITH REVENUE.](#)
- [4]. [Gang Li, Haiyan Song, and Stephen F. Witt\(2005\); Recent Developments in Econometric Modeling and Forecasting](#)
- [5]. Haiyan Song, Gang Li; [Tourism demand modelling and forecasting—A review of recent research](#)
- [6]. Haywantee Ramkissoon; [Perceived social impacts of tourism and quality-of-life: a new conceptual model](#)
- [7]. [David Curry; Booking Revenue and Usage Statistics \(2024\)](#)
- [8]. [Sadikarah M., \(2023; The MVC Architecture](#)
- [9]. [Batchelder N., \(2023\); venv - Creation of Virtual Environments](#)

ANEXA:

Froms.py:

```
from django import forms
```

```
from .models import Sejur
```

```
class SejurForm(forms.ModelForm):
```

```
    class Meta:
```

```
        model = Sejur
```

```
        fields = ['hotel', 'start_date', 'end_date', 'number_of_people']
```

```
        widgets = {
```

```
            'start_date': forms.DateInput(attrs={'type': 'date'}),
```

```
        'end_date': forms.DateInput(attrs={'type': 'date'}),  
    }
```

Models.py

```
from django.contrib.auth.models import AbstractUser
```

```
from django.db import models
```

```
class CustomUser(AbstractUser):
```

```
    RENTER = 'renter'
```

```
    LANDLORD = 'landlord'
```

```
    ROLE_CHOICES = [  
        (RENTER, 'Renter'),  
        (LANDLORD, 'Landlord'),  
    ]
```

```
    email = models.EmailField(unique=True)
```

```
    name = models.CharField(max_length=100, null=True, blank=True)
```

```
    role = models.CharField(max_length=10, choices=ROLE_CHOICES, default=RENTER)
```

```
    def save(self, *args, **kwargs):
```

```
        super().save(*args, **kwargs)
```

```
    def __str__(self):
```

```
        return self.username
```

```
class Renter(models.Model):
```

```

user = models.OneToOneField(CustomUser, on_delete=models.CASCADE, related_name='renter')
name = models.CharField(max_length=100, default="")
preferences = models.JSONField(null=True, blank=True)
centroid_latitude = models.FloatField(null=True, blank=True)
centroid_longitude = models.FloatField(null=True, blank=True)
rented_hotels = models.JSONField(null=True, blank=True)

def __str__(self):
    return f"{self.user.username} (Renter)"

def update_preference(self, preference):
    """Actualizează frecvența unei preferințe specifice"""
    if not self.preferences:
        self.preferences = {}

    if preference in self.preferences:
        self.preferences[preference] += 1
    else:
        self.preferences[preference] = 1

    self.save()

def update_location(self, latitude, longitude):
    """Actualizează centroidul geografic """
    if self.centroid_latitude is None or self.centroid_longitude is None:
        self.centroid_latitude = latitude
        self.centroid_longitude = longitude
    else:
        # Calculul centroidului folosind media aritmetică
        n = len(self.rented_hotels) if self.rented_hotels else 1

```

```
self.centroid_latitude = (self.centroid_latitude * n + latitude) / (n + 1)
self.centroid_longitude = (self.centroid_longitude * n + longitude) / (n + 1)
```

```
self.save()
```

```
class Landlord(models.Model):
```

```
    user = models.OneToOneField(CustomUser, on_delete=models.CASCADE, related_name='landlord')
```

```
    name = models.CharField(max_length=100, default="")
```

```
    email = models.EmailField(unique=True)
```

```
    listed_hotels = models.JSONField(null=True, blank=True)
```

```
    def __str__(self):
```

```
        return f"{self.user.username} (Landlord)"
```

```
import uuid
```

```
class Hotels(models.Model):
```

```
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False) # Utilizează UUIDField
```

```
    name = models.CharField(max_length=255)
```

```
    address = models.CharField(max_length=255)
```

```
    location = models.CharField(max_length=255)
```

```
    amenities = models.JSONField()
```

```
    phone = models.CharField(max_length=20)
```

```
    review_count = models.IntegerField(default=0)
```

```
    average_rating = models.FloatField(default=0.0)
```

```
    latitude = models.FloatField()
```

```
    longitude = models.FloatField()
```

```
    money = models.FloatField(default=0.0)
```

```
    def __str__(self):
```



```
    return self.name
```

```
class Meta:
```

```
    db_table = 'hotels'
```

```
class Sejur(models.Model):
```

```
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
```

```
    renter = models.ForeignKey(Renter, on_delete=models.CASCADE)
```

```
    hotel = models.ForeignKey(Hotels, on_delete=models.CASCADE)
```

```
    start_date = models.DateField()
```

```
    end_date = models.DateField()
```

```
    number_of_people = models.IntegerField()
```

```
    money = models.FloatField(default=0.0)
```

```
    rating = models.FloatField(null=True, blank=True)
```

```
    feedback = models.TextField(null=True, blank=True)
```

```
    def __str__(self):
```

```
        return f'Sejur at {self.hotel.name}'
```

```
class Meta:
```

```
    db_table = 'sejur'
```

```
recommendations.py:
```

```
import math
```

```
import unicodedata
```

```
import pandas as pd
```

```

import numpy as np

from collections import Counter

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import linear_kernel

from .models import CustomUser, Hotels, Sejur

# Listă de stop words pentru limba română
romanian_stop_words = [
    'în', 'și', 'de', 'la', 'pe', 'cu', 'pentru', 'din', 'sa', 'să',
    'a', 'este', 'fost', 'care', 'fi', 'că', 'ca', 'o', 'nu', 'un', 'lui',
    'acest', 'acesta', 'al', 'sau', 'ea', 'între', 'între', 'nici', 'mai',
    'decât', 'doar', 'decât', 'fără', 'fara'
]

def remove_diacritics(text):
    """Elimină diacriticele dintr-un text"""
    normalized_text = unicodedata.normalize('NFD', text)
    return ''.join([c for c in normalized_text if unicodedata.category(c) != 'Mn'])

def preferences_to_string(preferences):
    """Transformă dicționarul de preferințe într-un string"""
    if isinstance(preferences, dict):
        return ' '.join([f"{key} " * value for key, value in preferences.items()])
    return str(preferences)

def get_recommendations(user_id):
    """Oferă recomandări bazate pe cuvinte cheie pentru un utilizator specific"""
    user = CustomUser.objects.get(pk=user_id)

    if hasattr(user, 'renter') and user.renter.preferences:

```

```

user_preferences = preferences_to_string(user.renter.preferences)
user_locations = ""

combined_string = f"{user_preferences} {user_locations}"
else:
    return Hotels.objects.none()

hotels = Hotels.objects.all()

hotel_data = [{
    'hotel_id': hotel.id,
    'description': remove_diacritics(f"{preferences_to_string(hotel.amenities)} {hotel.location}")
} for hotel in hotels]
df = pd.DataFrame(hotel_data)

tfidf = TfidfVectorizer(stop_words=romanian_stop_words)
tfidf_matrix = tfidf.fit_transform(df['description'])

user_pref_vector = tfidf.transform([remove_diacritics(combined_string)])

cosine_sim = linear_kernel(user_pref_vector, tfidf_matrix)

sim_scores = list(enumerate(cosine_sim[0]))
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
top_hotels = [df.iloc[i[0]].hotel_id for i in sim_scores]

recommended_hotels = Hotels.objects.filter(id__in=top_hotels)

return recommended_hotels

```

```

def get_rating_based_recommendations(user_id, min_reviews=5):
    """Recomandă hoteluri bazate pe rating-urile medii ponderate și pe rating-urile date de utilizator"""
    user = CustomUser.objects.get(pk=user_id)
    user_stays = Sejur.objects.filter(renter=user.renter)
    user_ratings = {stay.hotel.id: stay.rating for stay in user_stays if stay.rating is not None}

    hotels = Hotels.objects.all()

    # Parametrii pentru media Bayesiană
    k = 10 # Factor de ponderare
    C = np.mean([hotel.average_rating for hotel in hotels if hotel.review_count > 0])

    hotel_ratings = {}
    for hotel in hotels:
        if hotel.review_count > 0:
            weighted_rating = (hotel.review_count / (hotel.review_count + k)) * hotel.average_rating + (k /
(hotel.review_count + k)) * C
        else:
            weighted_rating = hotel.average_rating

    # Ajustează în funcție de user rating
    if hotel.id in user_ratings:
        personal_rating = user_ratings[hotel.id]
        combined_rating = 0.7 * personal_rating + 0.3 * weighted_rating
    else:
        combined_rating = weighted_rating

    hotel_ratings[hotel.id] = combined_rating

```

```

topRatedHotels = sorted(hotel_ratings.items(), key=lambda x: x[1], reverse=True)
topHotelsIds = [hotel_id for hotel_id, rating in topRatedHotels]

recommendedHotels = Hotels.objects.filter(id__in=topHotelsIds)

return recommendedHotels

def calculate_distance(lat1, lon1, lat2, lon2):
    """Calculează distanța Euclidiană între două puncte geografice"""
    return math.sqrt((lat2 - lat1)**2 + (lon2 - lon1)**2)

def recommend_hotels_based_on_geolocation(renter):
    """Recomandă hoteluri bazate pe proximitatea față de centroidul geografic"""
    if renter.centroid_latitude is None or renter.centroid_longitude is None:
        return Hotels.objects.none()

    hotels = Hotels.objects.all()
    distances = []

    for hotel in hotels:
        distance = calculate_distance(renter.centroid_latitude, renter.centroid_longitude, hotel.latitude,
        hotel.longitude)
        distances.append((hotel, distance))

    distances.sort(key=lambda x: x[1]) # Sortează după distanță

    recommendedHotels = [hotel for hotel, _ in distances]

    return recommendedHotels

```

```

def normalize_scores(scores):
    """Normalizează o listă de scoruri între 0 și 1"""
    if not scores:
        return [0] * len(scores)

    min_score = min(scores)
    max_score = max(scores)

    if min_score == max_score:
        return [1] * len(scores)

    return [(score - min_score) / (max_score - min_score) for score in scores]

from django.db.models import Case, When, Value, IntegerField

def combined_recommendations(user_id, weights=(0.4, 0.3, 0.3), min_user_rating=3):
    renter = CustomUser.objects.get(pk=user_id).renter
    user_stays = Sejur.objects.filter(renter_id=renter.id)
    rated_hotels = {stay.hotel.id: stay.rating for stay in user_stays if stay.rating is not None}

    # Apply default rating if hotel is not rated
    all_hotel_ids = Hotels.objects.values_list('id', flat=True)
    for hotel_id in all_hotel_ids:
        if hotel_id not in rated_hotels:
            rated_hotels[hotel_id] = 4 # Default to 4 - Daca da 4 stele, inmulțirea se face cu 1

    cbf_hotels = get_recommendations(user_id)
    rating_hotels = get_rating_based_recommendations(user_id)
    geo_hotels = recommend_hotels_based_on_geolocation(renter)

```

```
cbf_weight, rating_weight, geo_weight = weights
```

```
cbf_scores = {hotel.id: index for index, hotel in enumerate(cbf_hotels)} if cbf_hotels else {}
```

```
cbf_norm = normalize_scores(list(cbf_scores.values())) if cbf_scores else []
```

```
rating_scores = {hotel.id: index for index, hotel in enumerate(rating_hotels)} if rating_hotels else {}
```

```
rating_norm = normalize_scores(list(rating_scores.values())) if rating_scores else []
```

```
geo_scores = {hotel.id: index for index, hotel in enumerate(geo_hotels)} if geo_hotels else {}
```

```
geo_norm = normalize_scores(list(geo_scores.values())) if geo_scores else []
```

```
combined_scores = {}
```

```
for hotel in Hotels.objects.filter(id__in=set(cbf_scores.keys()) | set(rating_scores.keys()) |  
set(geo_scores.keys())):
```

```
    cbf_score = cbf_norm[cbf_scores[hotel.id]] if hotel.id in cbf_scores else 0
```

```
    rating_score = rating_norm[rating_scores[hotel.id]] if hotel.id in rating_scores else 0
```

```
    geo_score = geo_norm[geo_scores[hotel.id]] if hotel.id in geo_scores else 0
```

```
    combined_score = (cbf_weight * cbf_score) + (rating_weight * rating_score) + (geo_weight * geo_score)
```

```
    rating = rated_hotels.get(hotel.id, 4) # Default to 4 if no explicit rating
```

```
    if rating == 4:
```

```
        pass
```

```
    elif rating < 4:
```

```
        if rating == 3:
```

```
            combined_score *= 0.7
```

```
        elif rating == 2:
```

```
            combined_score *= 0.5
```

```
        elif rating == 1:
```

```
            combined_score *= 0.3
```

```

combined_score *= 20

combined_scores[hotel.id] = combined_score

sorted_hotels = sorted(combined_scores.items(), key=lambda x: x[1], reverse=True)
top_hotels_ids = [hotel_id for hotel_id, score in sorted_hotels]

# Creating a Case for ordering
ordering = Case(
    *[When(id=hotel_id, then=pos) for pos, hotel_id in enumerate(top_hotels_ids)],
    output_field=IntegerField()
)

# Return the hotels ordered by the calculated score
recommended_hotels = Hotels.objects.filter(id__in=top_hotels_ids).order_by(ordering)

return recommended_hotels

```

urls.py:

```

from django.contrib import admin
from django.urls import path
from .views import (
    home_view, signup_view, login_view, profile_view, hotels_view,
    logout_view, edit_hotel_view, newhotel_view, landlord_view, acasa_view,
    submit_review
)

```



```
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", home_view, name='home-view'),
    path('signup/', signup_view, name='signup-view'),
    path('login/', login_view, name='login-view'),
    path('profile/', profile_view, name='profile-view'),
    path('hotels/', hotels_view, name='hotels-view'),
    path('logout/', logout_view, name='logout-view'),
    path('acasa/', acasa_view, name='acasa-view'),
    path('submit_review/<uuid:stay_id>/', submit_review, name='submit_review'),
    path('landlord/', landlord_view, name='landlord-view'),
    path('newhotel/', newhotel_view, name='newhotel-view'),
    path('edit-hotel/<uuid:hotel_id>/', edit_hotel_view, name='edit-hotel-view'),
]
```

Views.py:

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.forms import AuthenticationForm
from django.db import DatabaseError
from django.contrib.auth.decorators import login_required
from django.views.decorators.http import require_POST
from datetime import date

from .models import CustomUser, Hotels, Sejur, Renter, Landlord
from .forms import SejurForm
from .recommendations import combined_recommendations
```

```

# User registration view
def signup_view(request):
    if request.method == "POST":
        username = request.POST['user_name']
        email = request.POST['email']
        name = request.POST['name']
        password1 = request.POST['password1']
        password2 = request.POST['password2']
        role = request.POST['role']

        # Check if username exists
        if CustomUser.objects.filter(username=username).exists():
            return redirect('home-view')

        # Check if email exists
        if CustomUser.objects.filter(email=email).exists():
            return redirect('home-view')

        # Check username length
        if len(username) > 20:
            return redirect('home-view')

        # Check password match
        if password1 != password2:
            return redirect('home-view')

        # Check alphanumeric username
        if not username.isalnum():
            return redirect('home-view')

```

```
# Create user

myuser = CustomUser.objects.create_user(username=username, email=email, password=password1)

myuser.name = name

myuser.role = role

myuser.is_active = True

myuser.save()
```

```
# Associate with Renter or Landlord
```

```
if myuser.role == CustomUser.RENTER:
```

```
    Renter.objects.create(user=myuser, name=myuser.name)
```

```
elif myuser.role == CustomUser.LANDLORD:
```

```
    Landlord.objects.create(user=myuser, name=myuser.name, email=myuser.email)
```

```
return redirect('home-view')
```

```
return render(request, 'index.html')
```

```
# User login view
```

```
def login_view(request):
```

```
    if request.method == "POST":
```

```
        try:
```

```
            form = AuthenticationForm(request, data=request.POST)
```

```
            if form.is_valid():
```

```
                username = form.cleaned_data.get('username')
```

```
                password = form.cleaned_data.get('password')
```

```
                user = authenticate(request, username=username, password=password)
```

```
                if user is not None:
```

```
                    login(request, user)
```

```
                    return redirect('profile-view')
```

```
            else:
```

```

        return redirect('login-view')
    else:
        return redirect('login-view')
except DatabaseError as e:
    print(f"Database error during login: {type(e).__name__} - {e}")
    return redirect('login-view')
else:
    form = AuthenticationForm()
    return render(request, 'index.html', {'form': form})

```

# User logout view

```

def logout_view(request):
    try:
        logout(request)
        return redirect('home-view')
    except Exception as e:
        print(f"Error during logout: {e}")
        return redirect('home-view')

```

# Home view with user redirect

```

def home_view(request):
    if request.user.is_authenticated:
        if request.user.role == 'renter':
            return redirect('profile-view')
        elif request.user.role == 'landlord':
            return redirect('landlord-view')
    return render(request, 'index.html')

```

# View to create a new hotel

@login\_required

```

def newhotel_view(request):
    if not hasattr(request.user, 'landlord'):
        return redirect('profile-view')

    if request.method == 'POST':
        name = request.POST['name']
        address = request.POST['address']
        location = request.POST['location']
        amenities = request.POST['amenities']
        phone = request.POST['phone']
        money = request.POST['money']
        latitude = request.POST['latitude']
        longitude = request.POST['longitude']

        # Create new hotel
        hotel = Hotels.objects.create(
            name=name,
            address=address,
            location=location,
            amenities=amenities.split(','),
            phone=phone,
            money=float(money),
            latitude=float(latitude),
            longitude=float(longitude)
        )

        # Add hotel to landlord's list
        landlord = request.user.landlord
        if landlord.listed_hotels is None:
            landlord.listed_hotels = []

```

```

        landlord.listed_hotels.append(str(hotel.id))

        landlord.save()

    return redirect('landlord-view')

return render(request, 'newhotel.html')

# Hotels view for landlords
@login_required
def hotels_view(request):
    if not hasattr(request.user, 'landlord'):
        return redirect('profile-view')
    return render(request, 'hotels.html')

# Home view for renters
@login_required
def acasa_view(request):
    if not hasattr(request.user, 'renter'):
        return redirect('landlord-view')

    recommended_hotels = combined_recommendations(request.user.id)

    if request.method == 'POST':
        mutable_post = request.POST.copy()
        hotel_ids = mutable_post.getlist('hotel')
        hotel_id = next((h for h in hotel_ids if h), None)

        if hotel_id:
            mutable_post.setlist('hotel', [hotel_id])
    else:

```

```
form = SejurForm(request.POST)
form.add_error('hotel', 'Invalid hotel selection.')
return render(request, 'acasa.html', {'hotels': recommended_hotels, 'form': form})
```

```
form = SejurForm(mutable_post)
```

```
if form.is_valid():
```

```
    sejur = form.save(commit=False)
    sejur.renter = request.user.renter
    sejur.hotel_id = hotel_id
    sejur.save()
```

```
    renter = request.user.renter
    hotel = Hotels.objects.get(id=hotel_id)
```

```
    if renter.rented_hotels is None:
        renter.rented_hotels = []
```

```
    if hotel.name not in renter.rented_hotels:
        renter.rented_hotels.append(hotel.name)
        renter.save()
```

```
    # Update user location
    renter.update_location(hotel.latitude, hotel.longitude)
```

```
    # Update preferences based on hotel amenities
    amenities = hotel.amenities if isinstance(hotel.amenities, list) else hotel.amenities.split(',')
```

```
    for amenity in amenities:
        renter.update_preference(amenity.strip())
```

```

        return redirect('profile-view')
    else:
        print("Form errors:", form.errors)

    else:
        form = SejurForm()

    return render(request, 'acasa.html', {'hotels': recommended_hotels, 'form': form})

# Submit review for a stay
@login_required
@require_POST
def submit_review(request, stay_id):
    if not hasattr(request.user, 'renter'):
        return redirect('landlord-view')

    stay = get_object_or_404(Sejur, id=stay_id, renter=request.user.renter)

    rating = int(request.POST.get('rating', 0))
    if 1 <= rating <= 5:
        stay.rating = rating
        stay.save()

    hotel = stay.hotel

    k = 5
    total_reviews = hotel.review_count
    initial_total = hotel.average_rating * total_reviews
    new_total = initial_total + rating

```



```

    total_reviews += 1

    hotel.average_rating = (initial_total + k * rating) / (total_reviews + k)
    hotel.review_count = total_reviews
    hotel.save()
else:
    return redirect('profile-view')

return redirect('profile-view')

# Profile view for renters
@login_required
def profile_view(request):
    if not hasattr(request.user, 'renter'):
        return redirect('l-view')

    user = request.user
    current_date = date.today()

    past_stays = Sejur.objects.filter(renter=user.renter, end_date__lt=current_date)
    future_stays = Sejur.objects.filter(renter=user.renter, start_date__gte=current_date)

    context = {
        'user': user,
        'past_stays': past_stays,
        'future_stays': future_stays,
    }

    return render(request, 'profile.html', context)

```

```

# View for landlords to see their hotels

@login_required
def landlord_view(request):
    if not hasattr(request.user, 'landlord'):
        return redirect('profile-view')

    landlord = request.user.landlord
    hotel_ids = landlord.listed_hotels or []
    hotels = Hotels.objects.filter(id__in=hotel_ids)

    context = {
        'user': request.user,
        'hotels': hotels,
    }

    return render(request, 'landlord.html', context)

# Edit hotel view for landlords

@login_required
def edit_hotel_view(request, hotel_id):
    if not hasattr(request.user, 'landlord'):
        return redirect('profile-view')

    hotel = get_object_or_404(Hotels, id=hotel_id)

    if request.method == 'POST':
        hotel.name = request.POST['name']
        hotel.address = request.POST['address']
        hotel.location = request.POST['location']
        hotel.amenities = request.POST['amenities'].split(',')

```

```
hotel.phone = request.POST['phone']
hotel.money = float(request.POST['money'])
hotel.latitude = float(request.POST.get('latitude', hotel.latitude))
hotel.longitude = float(request.POST.get('longitude', hotel.longitude))
hotel.save()

return redirect('landlord-view')

context = {
    'hotel': hotel,
}
return render(request, 'newhotel.html', context)
```