MINISTERUL EDUCAŢIEI ŞI CERCETĂRII ŞTIINŢIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

# La pas prin România

## Site web pentru explorare atractii turistice din România

Proiect realizat de : Cozma Casiana, Anastasiu Andreea

Grupa: 30237

# Cuprins

# Introducere

Site-ul nostru de turism oferă o platformă inovatoare și interactivă, concepută pentru a facilita descoperirea și planificarea călătoriilor. Proiectul este structurat pe două componente principale: interfața client și interfața de administrare, fiecare având funcționalități bine definite pentru a răspunde nevoilor utilizatorilor.

## Partea de client

Partea de client este dedicată utilizatorilor care își doresc să exploreze și să organizeze vizite la cele mai reprezentative atracții turistice din România. Fiecare vizitator are posibilitatea de a naviga printr-o hartă detaliată, care evidențiază atracțiile turistice importante. Odată înregistrat sau autentificat, utilizatorul beneficiază de funcționalități personalizate.

După accesarea contului, utilizatorii pot selecta atracțiile turistice pe care doresc să le viziteze, cele pe care le-au vizitat deja sau pe care nu le consideră interesante. Aceste informații sunt organizate în contul personal, sub forma unei liste clare și intuitive, structurată pe trei categorii: *Vreau să vizitez*, *Nu vreau să vizitez* și *Am vizitat*. De asemenea, platforma oferă o hartă interactivă personalizată, care evidențiază atracțiile turistice deja vizitate, oferind utilizatorului o perspectivă vizuală asupra progresului său.

## Partea de administrare

Interfața de administrare este concepută pentru a permite gestionarea eficientă a conținutului și a utilizatorilor de pe platformă. Administratorii au acces la un set complet de funcționalități pentru a asigura actualitatea și relevanța informațiilor afișate.

Aceștia pot adăuga, edita sau șterge atracții turistice, având posibilitatea de a include detalii precum descrieri, imagini și coordonate geografice pentru fiecare locație. În plus, interfața de administrare oferă un control complet asupra conturilor utilizatorilor, permițând monitorizarea activităților acestora și gestionarea preferințelor turistice.

Prin această abordare dual, site-ul nostru de turism asigură o experiență completă și optimizată, combinând interactivitatea și personalizarea pentru utilizatori cu eficiența și controlul pentru administratori. Proiectul este astfel pregătit să devină un punct de referință în planificarea călătoriilor și promovarea turismului local.

# Funcționalitate site

## Partea de client

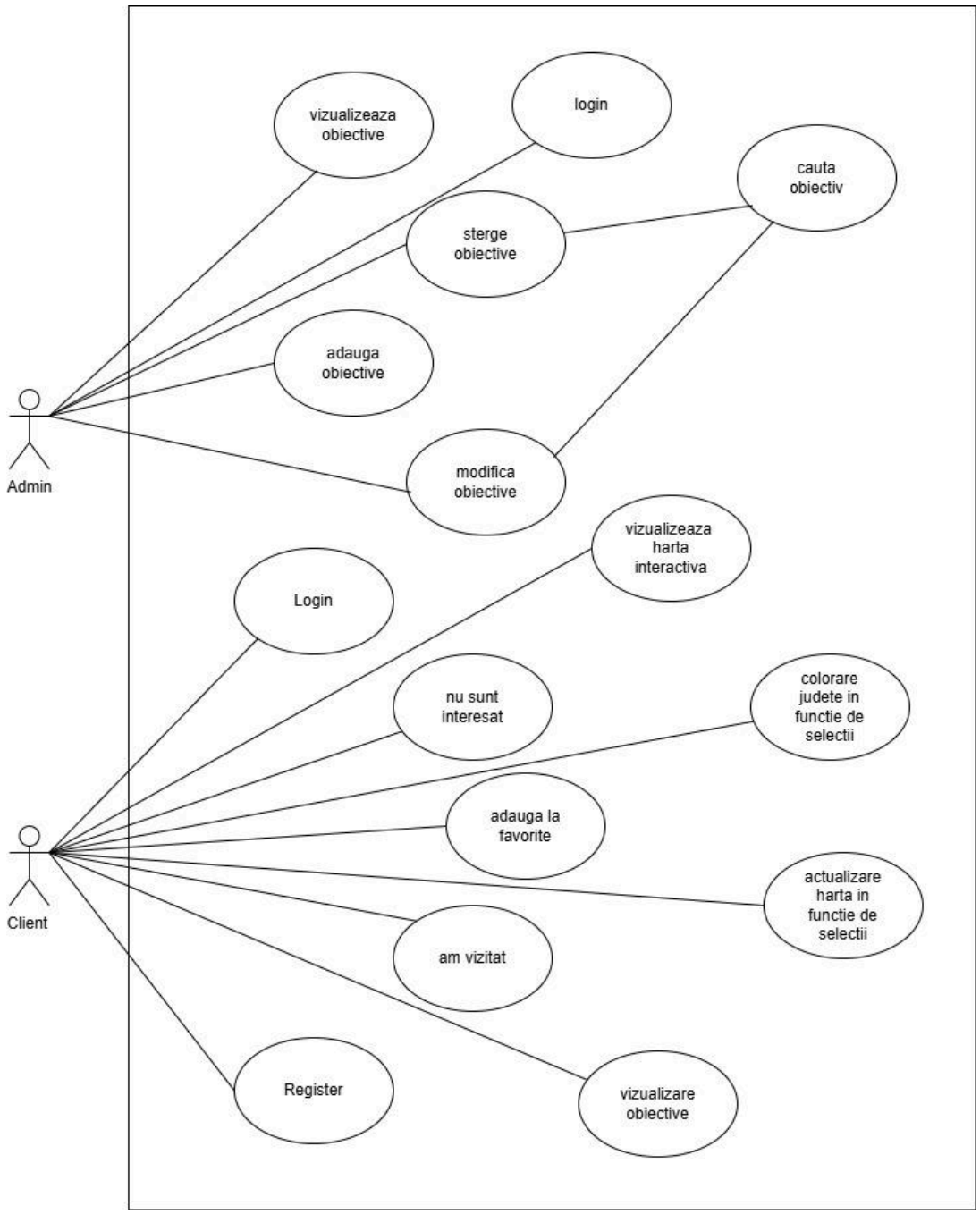
1. Explorarea atracțiilor turistice
   Vizitatorii pot explora cele mai importante atracții turistice din România, utilizând o hartă interactivă care evidențiază locațiile de interes.
2. Înregistrare și autentificare
   Utilizatorii pot crea conturi noi sau se pot autentifica în conturile existente pentru a accesa funcționalități personalizate.
3. Gestionarea preferințelor turistice
   După autentificare, utilizatorii pot selecta atracțiile turistice pe care:
   - doresc să le viziteze,
   - le-au vizitat deja,
   - nu sunt interesați să le viziteze.

     Aceste preferințe sunt salvate într-o listă personală, organizată pe trei coloane: *Vreau să vizitez*, *Nu vreau să vizitez* și *Am vizitat*.
4. Hartă personalizată
   Utilizatorii au acces la o hartă interactivă care evidențiază atracțiile deja vizitate, oferind o reprezentare vizuală a progresului lor.

## Partea de administrare

1. Gestionarea atracțiilor turistice
   Administratorii pot adăuga, edita sau șterge atracții turistice, actualizând informații precum descrierea, imaginile și locația pe hartă.
2. Gestionarea utilizatorilor
   Administratorii pot vizualiza, modifica sau șterge conturile utilizatorilor, având acces la preferințele și activitățile acestora.
3. Control asupra conținutului
   Administratorii au control complet asupra informațiilor afișate pe site, asigurând actualitatea și relevanța acestora pentru utilizatori.

# Diagrame




Sistemul include doi actori principali, Adminul și Clientul, fiecare cu roluri și funcționalități bine definite:

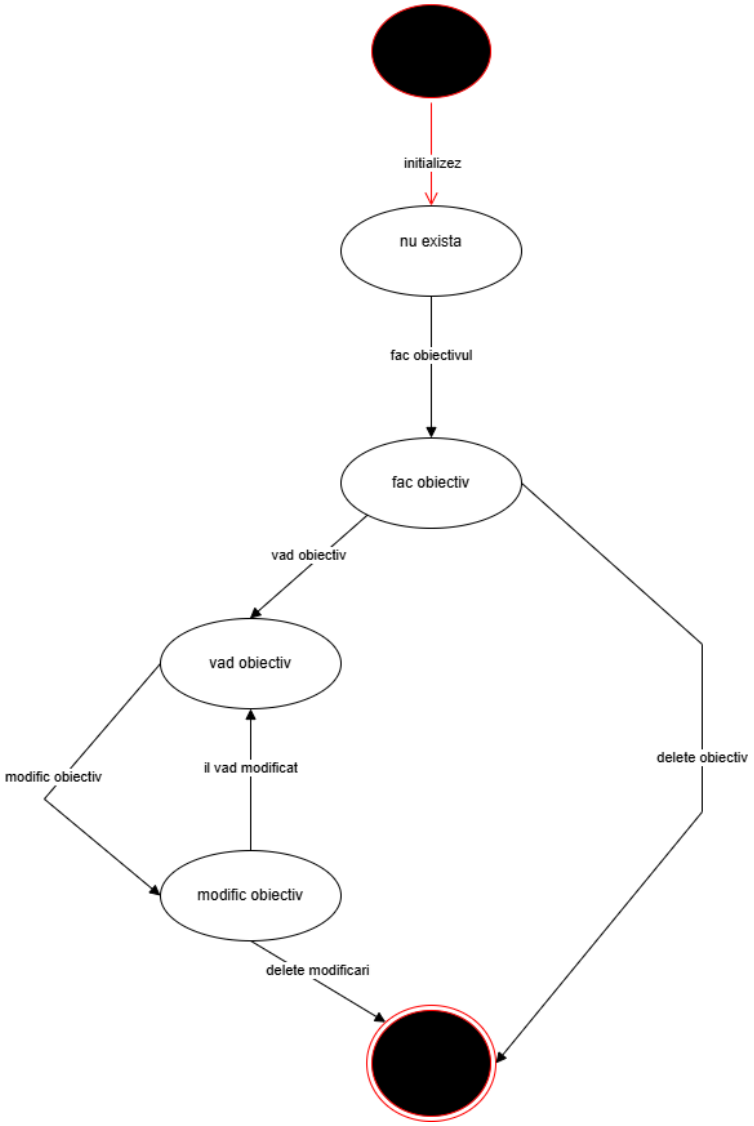
- Adminul are acces la funcții administrative avansate, precum:
    - Vizualizarea, adăugarea, modificarea și ștergerea obiectivelor.

- Căutarea obiectivelor specifice.
- Gestionarea și explorarea hărții interactive.
- Autentificarea pentru acces securizat.

- Clientul beneficiază de funcții orientate spre explorare și personalizare, precum:
  - Vizualizarea obiectivelor disponibile.
  - Adăugarea obiectivelor la favorite sau marcarea acestora ca vizitate sau neinteresante.
  - Interacțiunea cu o hartă interactivă, ce permite colorarea județelor și actualizarea dinamică în funcție de preferințe.
  - Înregistrarea și autentificarea pentru o experiență personalizată.

Fiecare actor este conectat la funcționalitățile relevante prin relații logice, reflectând modul în care aceștia interacționează cu sistemul pentru a îndeplini diferite scopuri.

## **Diagrama de stare**- Anastasiu Andreea




Starea inițială (cercul negru din partea de sus)
Procesul începe din starea inițială, indicată de cercul negru. Aceasta semnifică faptul că sistemul este inițializat pentru a începe gestionarea unui obiectiv.
Starea „Nu există”
După inițializare, sistemul verifică dacă obiectivul există deja. Dacă acesta nu există, trece la starea „fac obiectivul”.
Starea „Fac obiectiv”
În această stare, utilizatorul sau administratorul creează un obiectiv nou. După finalizarea acestei acțiuni, se trece la starea în care obiectivul poate fi vizualizat.
Starea „Văd obiectiv”
Aici, obiectivul creat poate fi vizualizat de către utilizator. Din această stare există două opțiuni:

- Modificare obiectiv: Dacă utilizatorul decide să modifice obiectivul.
- Ștergere obiectiv: Dacă obiectivul este considerat inutil și trebuie eliminat.

Starea „Modific obiectiv”
În această stare, obiectivul este actualizat. După modificare, utilizatorul îl poate vizualiza din nou în forma actualizată sau poate șterge modificările.
Ștergerea modificărilor
Dacă modificările realizate nu sunt dorite, acestea pot fi șterse, revenindu-se la starea anterioară.
Starea finală (cercul negru cu contur roșu din partea de jos)
Procesul se încheie aici, fie prin ștergerea completă a obiectivului, fie prin finalizarea modificărilor. Aceasta semnifică terminarea ciclului de gestionare pentru un anumit obiectiv.

**Diagramă de activitate**- Cozma Casiana


Această diagramă de activitate descrie fluxul de utilizare a unui site web turistic, ilustrând interacțiunile posibile ale utilizatorilor, fie ca **client** sau **administrator**. Fluxul este organizat astfel încât să permită accesul și utilizarea funcționalitților relevante pentru fiecare rol.

## Etapele fluxului

### 1. Start

Procesul începe prin determinarea rolului utilizatorului: **client** sau **admin**.

### 2. Decizia: admin/client?

- Utilizatorul trebuie să aleagă dacă dorește să acceseze site-ul ca:
  - **Client** (utilizator obișnuit care navighează și interacționează cu atracțiile turistice).
  - **Administrator** (utilizator cu drepturi de gestionare a site-ului).

## 3. Fluxul pentru client

### a. Accesul fără cont:

- Clientul poate accesa informații despre atracțiile turistice, dar **nu are dreptul de a interacționa cu acestea** (de exemplu, nu poate salva atracții într-o listă personală).

### b. Accesul cu cont:

- Dacă utilizatorul dorește să interacționeze cu atracțiile turistice, verificăm dacă:
    - Are deja un cont:
        - Utilizatorul trebuie să se autentifice (logare).
    - Nu are cont:
        - Utilizatorul poate crea un cont nou (încregistrare).
- După autentificare sau încregistrare, clientul are posibilitatea de a:
    - **Adăuga atracții turistice în cele trei liste personale**:
        - „Vreau să vizitez”
        - „Nu vreau să vizitez”
        - „Am vizitat”

## 4. Fluxul pentru administrator

- Administratorul are acces direct la **interfața de administrare** a site-ului, care permite:
    - Gestionarea atracțiilor turistice (adăugare, modificare, ștergere).
    - Gestionarea utilizatorilor (de exemplu, aprobarea și ștergerea conturilor)

# Design Patterns

1. **Singleton- Anastasiu Andreea**
2. **Proxy- Cozma Casiana**

1.Singleton este un *Design Pattern* care asigură că o clasă are o singură instanță în tot programul și oferă un punct global de acces la această instanță. Este util în scenarii unde ai nevoie de un control centralizat, cum ar fi gestionarea unei conexiuni la bază de date, logarea evenimentelor, sau un manager de resurse.

Cum funcționează Singleton?

- Constructorul clasei este privat pentru a preveni instanțierea externă.
- O instanță unică este creată fie în momentul încărcării clasei (eager initialization), fie la prima utilizare (lazy initialization).
- Este furnizată o metodă statică pentru accesarea acestei instanțe.

ex de cod in care folosim singletone implicit:

```
package com.example.demo.service.impl;

import com.example.demo.model.Admin;

import com.example.demo.model.DTO.AdminLoginDTO;

import com.example.demo.repository.AdminRepository;

import com.example.demo.service.AdminService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class AdminServiceImpl implements AdminService {

 @Autowired

 AdminRepository adminRepository;

 @Override

  public Admin create(Admin admin) {

     return adminRepository.save(admin);

  }
```

```
@Override
  public List<Admin> getAllAdmins() {
      return adminRepository.findAll();
  }
@Override
  public Admin getAdminById(Integer idAdmin) {
      return adminRepository.findById(idAdmin).get();
  }
@Override
  public Admin update(Admin admin) {
      return adminRepository.save(admin);
  }
 @Override
  public String deleteAdmin(Integer idAdmin) {
      adminRepository.deleteById(idAdmin);
      return "Admin deleted successfully!";
  }
@Override
  public Boolean validateAdmin(AdminLoginDTO adminLoginDTO) {
      Admin admin = adminRepository.findByUsernameAndPassword(adminLoginDTO.getUsername(),
adminLoginDTO.getPassword()); // Folosește metoda corectă
      return admin != null; // Returnează true dacă admin este găsit
  }
}
```

Singleton-ul este gestionat automat de containerul de Spring și este utilizat pentru a gestiona cererile HTTP la endpoint-urile definite.

2.Proxy este un *Design Pattern* structural care furnizează un substitut sau un înlocuitor pentru un alt obiect. Este utilizat pentru a controla accesul la acel obiect, oferind o funcționalitate suplimentară, cum ar fi verificări de securitate, caching sau comunicare remote.

Cum funcționează Proxy?

- Proxy-ul implementează aceeași interfață ca obiectul real (real subject).
- Clasa proxy gestionează accesul la obiectul real și poate adăuga logică înainte sau după apelarea metodelor acestuia.

exemplu de cod:

```
import axios from 'axios';
import { useParams, useNavigate } from 'react-router-dom';
import './MyAccountVis.css';


const MyAccountVis = () => {
 const [visitedAttractions, setVisitedAttractions] = useState([]);
 const [toVisitAttractions, setToVisitAttractions] = useState([]);
 const [notInterestedAttractions, setNotInterestedAttractions] = useState([]);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState(null);


 const { locationName } = useParams();
 const navigate = useNavigate();
 const clientEmail = localStorage.getItem('email');
 const location = locationName || localStorage.getItem('locationName');


 const fetchData = async () => {
  if (!clientEmail) {
   setError('Nu sunteți autentificat. Conectați-vă pentru a vedea datele.');
   setLoading(false);
   return;
  }
```

```
  try {
    const response = await axios.get(
      `http://localhost:8080/attraction-user-status/get-all-user-visited-attractions/${clientEmail}/${location}`
    );

    setVisitedAttractions(response.data.visited || []);
    setToVisitAttractions(response.data.toVisit || []);
    setNotInterestedAttractions(response.data.notInterested || []);
    setLoading(false);
  } catch (err) {
    setError('A apărut o eroare la încărcarea datelor.');
    setLoading(false);
  }
 };

 useEffect(() => {
  fetchData();
 }, [clientEmail, location]);

 const handleDelete = async (attractionName, status) => {
  try {
     await axios.delete(
        `http://localhost:8080/attraction-user-status/delete/${clientEmail}/${location}/${attractionName}`
     );
```

```
public class Config {

  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
     http
```

```
        .cors(Customizer.withDefaults()) // Activează CORS

        .csrf(csrf -> csrf.disable()) // Deactivate CSRF
        .authorizeHttpRequests(auth -> auth

                .anyRequest().permitAll()
        )
        .httpBasic(Customizer.withDefaults());

    return http.build();
}
```

Folosim aici pattern-ul de proxy pentru comunicare dintre baza de date si frontend (cu “axios”).

# Testare

În cadrul acestui proiect, procesul de testare a fost structurat în două categorii principale: testarea backend-ului și testarea frontend-ului.

Testarea backend-ului

Pentru backend, testarea a fost realizată utilizând Postman, un instrument versatil care ne-a permis să verificăm funcționalitatea API-urilor. Am evaluat următoarele aspecte:

1. Endpoint-uri funcționale:
   Fiecare endpoint a fost testat pentru a valida răspunsurile corecte la cereri de tip GET, POST, PUT și DELETE.
2. Validarea datelor:
   Am testat datele trimise către server pentru a ne asigura că respectă structura așteptată și pentru a verifica comportamentul aplicației în cazul datelor incomplete sau incorecte (ex: email invalid, ID-uri inexistente).
3. Erori și coduri de răspuns:
   Codurile de răspuns HTTP au fost verificate pentru a ne asigura că fiecare cerere primește răspunsul corespunzător:

- 200 OK pentru cereri reușite;
- 400 Bad Request pentru date incorecte;
- 401 Unauthorized pentru cereri fără autentificare validă;
- 404 Not Found pentru resurse inexistente;
- 500 Internal Server Error pentru erori interne.

## Testarea frontend-ului

Pentru frontend, testarea a fost realizată utilizând consola browserului, cu accent pe următoarele aspecte:

1. Interfața utilizatorului (UI):
   S-a verificat afișarea corectă a tuturor componentelor grafice pe diferite rezoluții și dispozitive, inclusiv paginile cu atracțiile turistice și harta interactivă.
2. Interacțiunile utilizatorului:
   Funcționalitățile interactive au fost testate, cum ar fi:

- Navigarea către pagina unui județ prin click pe hartă;
- Selectarea opțiunilor „vreau să vizitez” sau „am vizitat”;
- Modificarea preferințelor utilizatorilor.

3. Validarea datelor:
   Formularele aplicației au fost testate pentru a ne asigura că utilizatorii primesc mesaje

de eroare adecvate în cazul introducerii unor date greșite (ex: câmpuri goale, parolă incorectă).

4. Conexiunea cu backend-ul:
   S-a monitorizat comunicarea dintre frontend și backend, analizând cererile și răspunsurile API prin consola browserului pentru a identifica eventuale probleme.

## Rezultatele testării

1. Backend-ul funcționează conform specificațiilor, iar erorile sunt gestionate corespunzător. Datele sunt salvate și prelucrate corect.
2. Frontend-ul oferă o experiență de utilizare fără erori, iar toate interacțiunile sunt funcționale și intuitive.

# Mod de rulare al aplicației

Aplicația este dezvoltată utilizând React pentru frontend și Java Spring Boot pentru backend. Pornirea aplicației presupune configurarea ambelor componente, asigurându-se că toate dependențele și setările necesare sunt corect implementate.

1. **Configurarea mediului de lucru**

1. Cerințe preliminare:
    - Node.js (pentru frontend) – descărcați și instalați de pe Node.js.
    - Java Development Kit (JDK) – versiunea 21 sau mai recentă.
    - Maven (opțional, dacă nu este inclus în IDE).
    - MySQL sau alt sistem de gestionare a bazelor de date compatibil.
    - Postman pentru testarea API-urilor backend (opțional).
    - IntelliJ IDEA
    - Visual Studio Code pentru frontend.
2. Baza de date:
    - Instalați și configurați serverul bazei de date (ex. MySQL).
    - Creați o bază de date nouă (ex: tourism_app).
    - Importați schema bazei de date folosind fișierul SQL furnizat (ex. schema.sql).

2. **Instalarea aplicației**

Frontend (React)

Navigați în directorul proiectului React: **cd frontend**

Instalați dependențele utilizând npm: **npm install**
Porniți serverul React: **npm run dev**

Aplicația va rula pe **http://localhost:5173.**

Backend (Java Spring Boot)

Deschideți directorul backend în IntelliJ IDEA

Configurați fișierul application.properties din src/main/resources:

**spring.datasource.url=jdbc:mysql://localhost:5173/tourism_app**

**spring.datasource.username=your_db_username**

**spring.datasource.password=your_db_password**

**spring.jpa.hibernate.ddl-auto=update**

**server.port=8080**

Verificați configurațiile Maven și actualizați dependențele (în IntelliJ sau folosind comanda):
**bash**
**mvn clean install**

Porniți serverul backend rulând clasa principală din src/main/java, de obicei clasa Application. Backend-ul va asculta cererile pe **http://localhost:8080.**

### 3. Pornirea aplicației

1. Lansați backend-ul utilizând IntelliJ IDEA.
2. Lansați frontend-ul din terminal folosind comanda npm run dev.
3. Accesați aplicația în browser pe **http://localhost:5173.**

La lansarea aplicației, utilizatorul va fi direcționat către pagina de start. Aici, utilizatorul poate selecta rolul pe care dorește să îrl joace: **Administrator** sau **Client**.

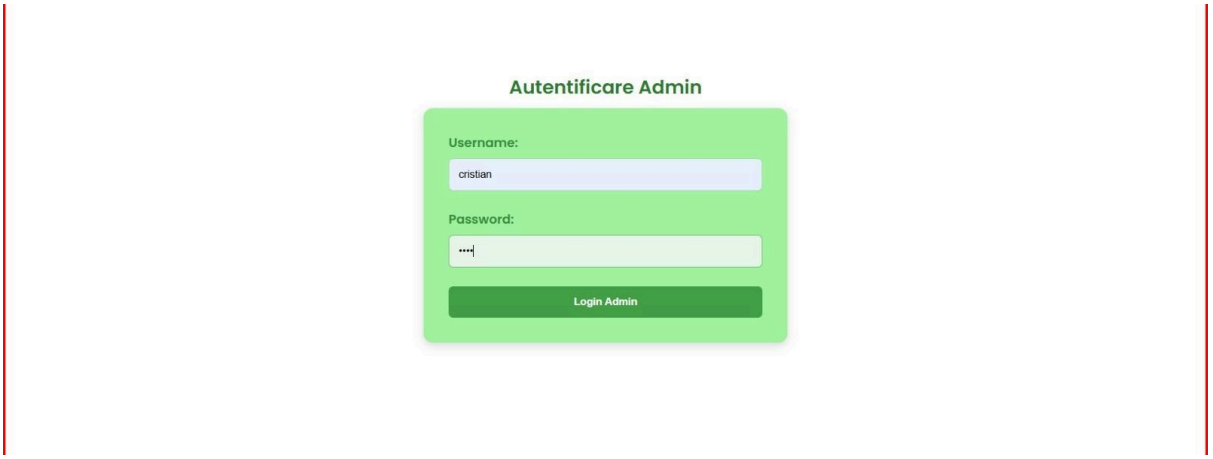
#### Selectarea Rolului

După selectarea rolului, aplicația oferă acces la două tipuri de pagini:

1. **Pagina de Administrator** - unde administratorul se poate autentifica.
2. **Pagina de Utilizator** - unde utilizatorul neînregistrat poate accesa secțiunea principală a aplicației.

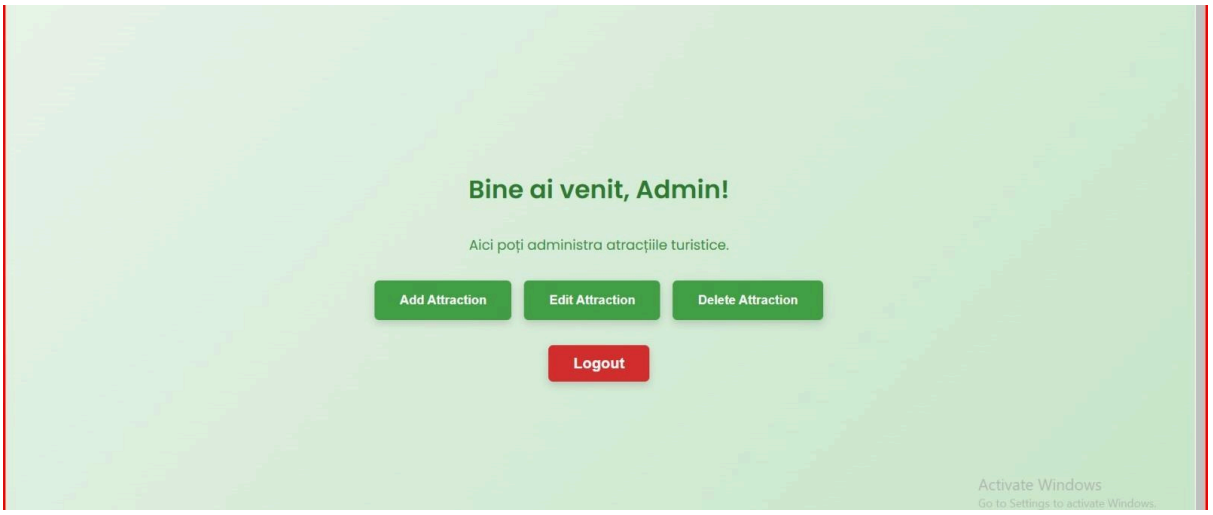
# Partea de Administrator

### Autentificarea Administratorului



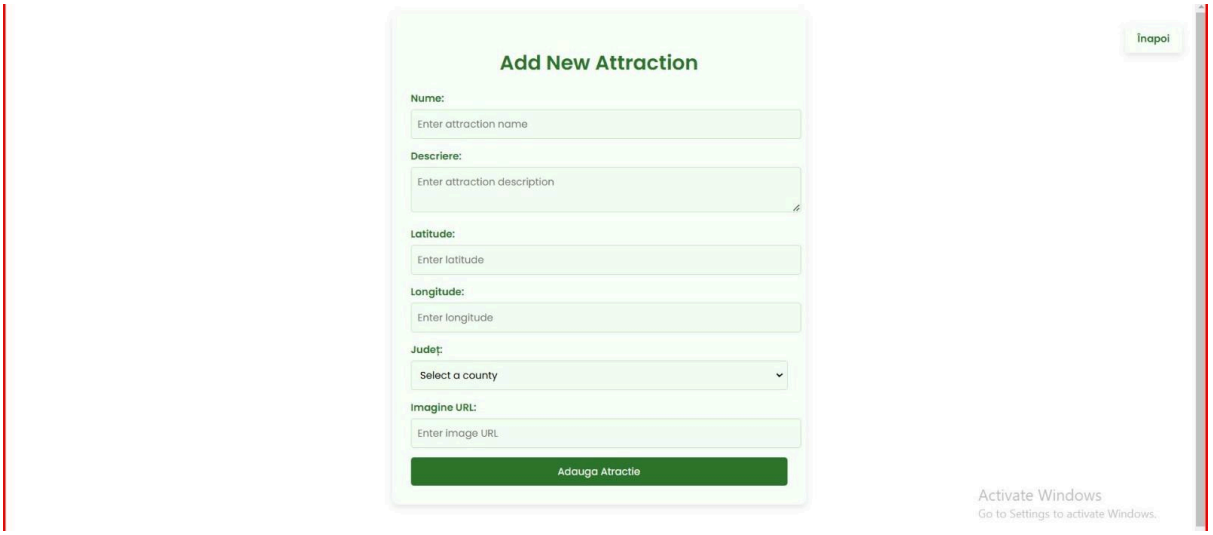

După autentificare, administratorul este direcționat către o pagină principală (**Home Admin**), unde poate alege una dintre următoarele operații:

- **Adăugare Obiectiv**
- **Editare Obiectiv**
- **Ștergere Obiectiv**
- **Deconectare**




### Operațiunile Administratorului

Fiecare operație are o pagină dedicată, cu o interfață intuitivă, care permite gestionarea ușoară a obiectivelor turistice.




---

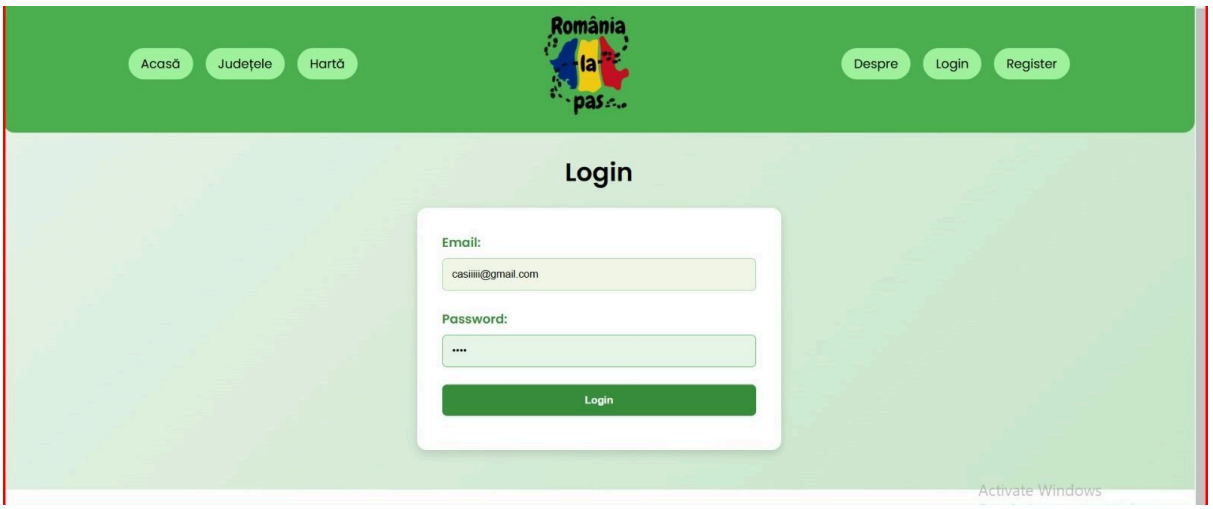
## Partea de Utilizator


### Login și Register

Utilizatorul are opțiunea de a:

- **Logare** – pentru accesarea unui cont existent.
- **Înregistrare** – pentru crearea unui cont nou.

Dacă utilizatorul dorește să creeze un cont care există deja, aplicația va afișa un mesaj de avertizare și va redirecționa utilizatorul către pagina de logare.




**Navigarea Utilizatorului**

După logare sau înregistrare, utilizatorul este direcționat către pagina principală (**Home**). De aici, utilizatorul poate:

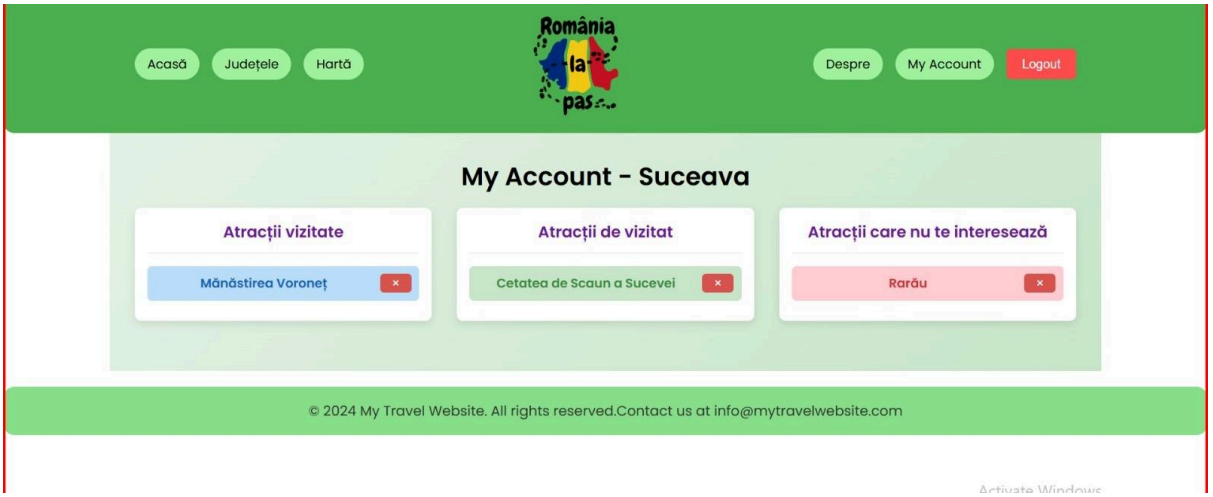
- **Accesa pagina de județe** prin intermediul butonului "Județe" din antet.
- Vizualiza o listă cu toate județele din țară.

**Lista de Județe**

Selectând un județ, utilizatorul va fi redirecționat către o pagină dedicată care afișează:

- **Numele atracției**
- **Descriere scurtă**
- **Poză reprezentativă**
- **Trei butoane opționale:**
  - *"Vreau să vizitez"*
  - *"Nu vreau să vizitez"*
  - *"Am vizitat"*

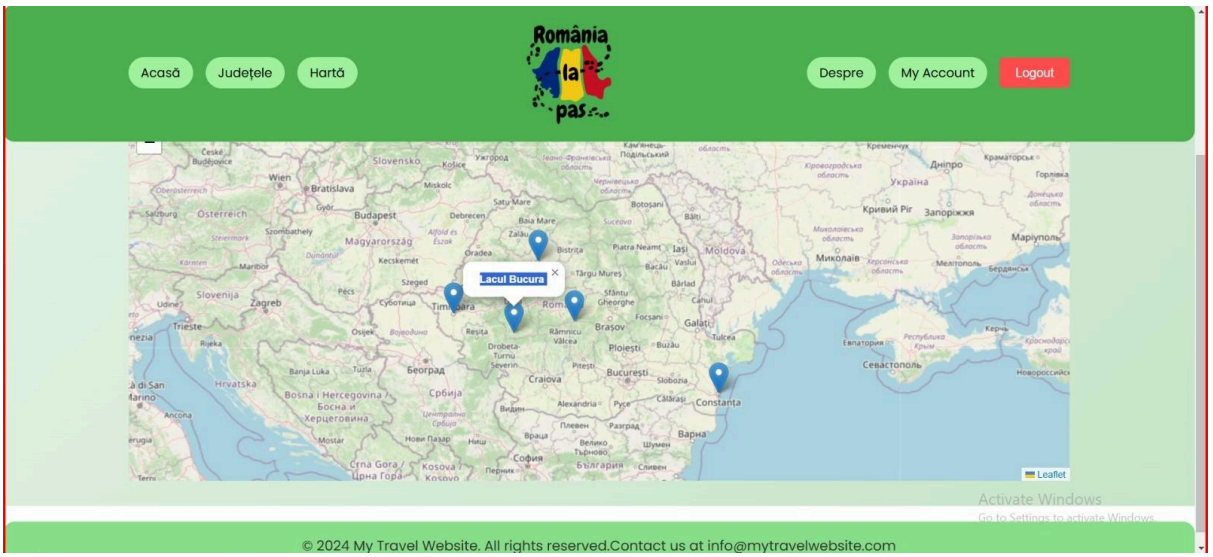
**Pagina "My Account"**




Pe baza selecțiilor utilizatorului din paginile județelor:

- Informațiile sunt organizate în trei coloane, fiecare corespunzând uneia dintre opțiuni (*"Vreau să vizitez"*, *"Nu vreau să vizitez"*, *"Am vizitat"*).
- Harta aplicației se actualizează automat pentru a indica atracțiile deja vizitate.

Această funcționalitate ajută utilizatorul să-și planifice și să-și urmărească progresul în explorarea obiectivelor turistice din țară.



**Rol in echipa:**

**Cozma Casiana: Backend, introducere date în baza de date, frontend**

**Anastasiu Andreea: frontend, documentație, descriere și imagini site**