

# **Platformă de Rezervări Turistice**

## Documentație

Purenciu Diana  
30236

14 ianuarie 2026

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>2</b>
1.1	Descrierea proiectului . . . . .	2
1.2	Obiectivele proiectului . . . . .	2
<b>2</b>	<b>Tehnologii Utilizate</b>	<b>3</b>
2.1	Arhitectura Generală . . . . .	3
2.2	Backend: .NET Core și C# . . . . .	3
2.3	Frontend: React.js . . . . .	3
2.4	Baza de date . . . . .	3
<b>3</b>	<b>Analiza și Proiectarea Sistemului</b>	<b>4</b>
3.1	Cerințe Funcționale . . . . .	4
3.2	Diagrama Use Case . . . . .	4
3.3	Diagrama de Stări (State Diagram) . . . . .	5
3.4	Diagrama de Secvență . . . . .	7
3.5	Diagrama de Comunicare . . . . .	8
<b>4</b>	<b>Implementarea Soluției</b>	<b>9</b>
4.1	Implementarea Backend-ului . . . . .	9
4.1.1	Configurarea bazei de date și Seeder-ul . . . . .	9
4.1.2	Design Pattern-uri: Singleton . . . . .	9
4.1.3	API Controllers . . . . .	9
4.2	Implementarea Frontend-ului . . . . .	10
4.2.1	Componente și State . . . . .	10
4.2.2	Comunicarea cu serverul . . . . .	10
<b>5</b>	<b>Concluzii și Utilizare</b>	<b>11</b>
5.1	Concluzii . . . . .	11
5.2	Dezvoltări ulterioare . . . . .	11
5.3	Scurt Read Me (Ghid de Pornire) . . . . .	11
	<b>Bibliografie</b>	<b>12</b>

# Capitolul 1

## Introducere

### 1.1 Descrierea proiectului

Acest proiect reprezintă o aplicație web de tip *Full Stack* dedicată rezervărilor turistice. Scopul principal este digitalizarea procesului de planificare a vacanțelor, oferind o alternativă modernă la agențiile de turism clasice.

Aplicația funcționează ca un catalog interactiv unde utilizatorii pot explora destinații, pot consulta detalii și recenzii verificate și pot verifica disponibilitatea cazării în timp real. Accentul a fost pus pe o experiență de utilizare simplă și intuitivă.

### 1.2 Obiectivele proiectului

Obiectivul central a fost simularea unui scenariu real de dezvoltare software, urmărind întregul ciclu de viață al aplicației.

Obiectivele specifice au inclus:

- **Arhitectura Client-Server:** Separarea logicii de business (Backend) de interfața cu utilizatorul (Frontend).
- **Dezvoltarea API-ului:** Utilizarea .NET Core pentru a crea un sistem de gestionare a datelor.
- **Interactivitate:** Implementarea unei interfețe dinamice cu React.js.
- **Persistența datelor:** Proiectarea unei baze de date relaționale eficiente.
- **Optimizare:** Utilizarea design pattern-ului *Singleton* pentru gestionarea centralizată a rezervărilor.

# Capitolul 2

## Tehnologii Utilizate

### 2.1 Arhitectura Generală

Sistemul utilizează o arhitectură în care Frontend-ul și Backend-ul sunt proiecte independente care comunică prin protocolul HTTP. Transferul datelor între cele două componente se realizează exclusiv în format **JSON**.

### 2.2 Backend: .NET Core și C#

Pentru partea de server s-a ales platforma .NET Core datorită performanței ridicate. Serverul funcționează ca un API RESTful, expunând rute (endpoints) pe care interfața le apelează pentru a citi sau scrie date.

### 2.3 Frontend: React.js

Interfața vizuală este o aplicație de tip *Single Page Application (SPA)* construită cu React.js. Aceasta permite navigarea între pagini instantaneu, fără reîncărcarea browserului.

### 2.4 Baza de date

Stocarea datelor se face într-o bază de date relațională (SQL Server), gestionată prin **Entity Framework Core**. Această abordare a permis lucrul direct cu obiecte și clase în C#, eliminând necesitatea scrierii manuale de cod SQL.

# Capitolul 3

## Analiza și Proiectarea Sistemului

### 3.1 Cerințe Funcționale

1. **Vizualizare catalog:** Afișarea listei cu destinații turistice.
2. **Filtrare:** Posibilitatea de a sorta ofertele după preț sau categorie.
3. **Detalii:** Pagină dedicată fiecărei locații (poze, descriere, preț).
4. **Rezervare:** Formular pentru selectarea perioadei și salvarea rezervării.
5. **Recenzii:** Posibilitatea de a adăuga feedback (text și rating).

### 3.2 Diagrama Use Case

Diagrama Use Case definește comportamentul aplicației din punctul de vedere al actorului principal, adică utilizatorul.

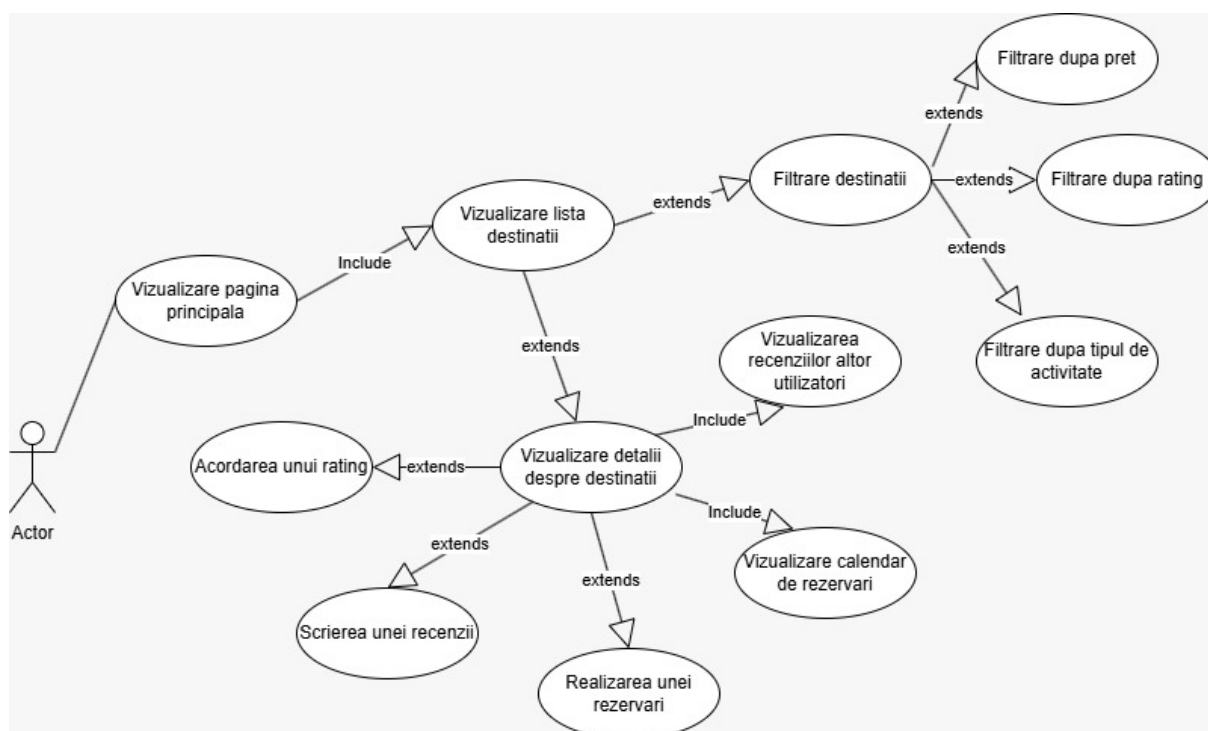


Figura 3.1: Diagrama Use Case a aplicației

Conform diagramei prezentate în Figura 3.1, fluxul de utilizare este structurat astfel:

- **Actorul:** Este reprezentat de utilizatorul obișnuit care accesează site-ul.
- **Pagina Principală și Lista (Relația Include):** Punctul de pornire este *Vizualizarea paginii principale*. Aceasta include în mod obligatoriu *Vizualizarea listei de destinații*, deoarece catalogul de oferte este conținutul central al primei pagini.
- **Filtrarea (Relația Extend):** Din lista de destinații, utilizatorul poate alege să restrângă rezultatele. Aceasta este o acțiune opțională (*Extend*). Filtrarea se poate realiza după criterii specifice precum preț sau rating.
- **Detaliile Destinației:** Selectarea unei oferte deschide cazul de utilizare *Vizualizare detalii despre destinații*. Această acțiune declanșează automat alte două funcționalități obligatorii (*Include*):
  - *Vizualizarea recenziilor altor utilizatori*;
  - *Vizualizarea calendarului de rezervări*.
- **Acțiuni Opționale pe pagina de detalii (Extend):** Odată ajuns pe pagina unei destinații, utilizatorul are libertatea de a efectua acțiuni suplimentare: poate realiza o rezervare, poate scrie o recenzie text sau poate acorda un rating.

### 3.3 Diagrama de Stări (State Diagram)

Diagrama de stări modelează comportamentul dinamic al sistemului pentru funcționalitatea de adăugare a unei recenzii.

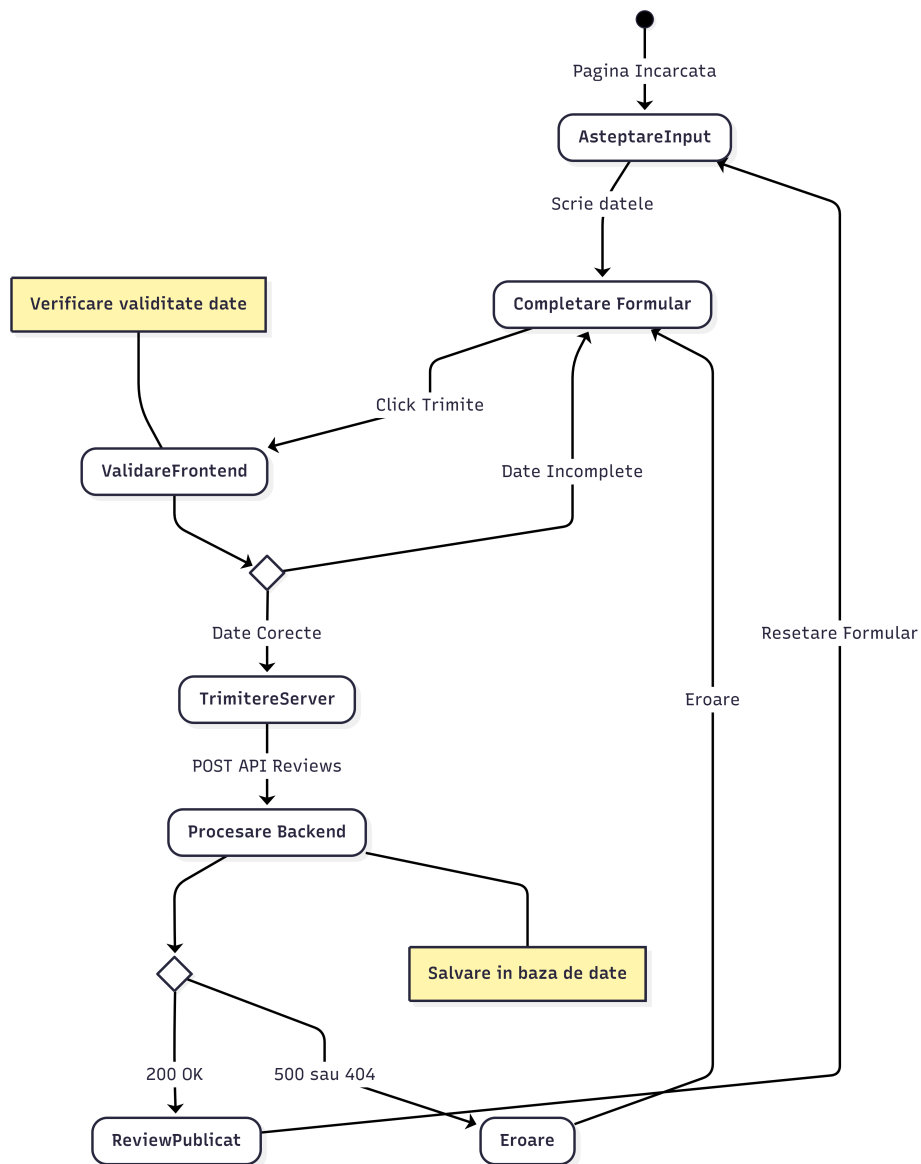


Figura 3.2: Diagrama de Stări - Fluxul de adăugare a unei recenzii

Conform diagramei, procesul de publicare a unei recenzii respectă următoarele etape:

- **Inițiere:** Utilizatorul se află pe pagina unei destinații (starea *Pagina Încărcată*) și interacționează cu formularul de feedback (*AșteptareInput*).
- **Validare Frontend:** După apăsarea butonului de trimitere, aplicația verifică local dacă datele introduse sunt valide.
  - Dacă datele sunt incomplete, utilizatorul este notificat și sistemul revine în starea de completare.
  - Dacă datele sunt corecte, cererea este trimisă către server.
- **Comunicarea cu API-ul:** Cererea este transmisă prin metoda HTTP POST către server. Acesta intră în etapa de *Procesare Backend* și încearcă să salveze recenzia în baza de date.

- **Finalizare:**

- **Succes:** Dacă salvarea reușește (cod 200 OK), sistemul trece în starea *ReviewPublicat*.
- **Eroare:** În cazul unei probleme tehnice, sistemul intră în starea de *Eroare*, permițând utilizatorului să încerce din nou.

### 3.4 Diagrama de Secvență

Diagrama de secvență detaliază interacțiunile tehnice care au loc în momentul în care un utilizator decide să rezerve o vacanță.

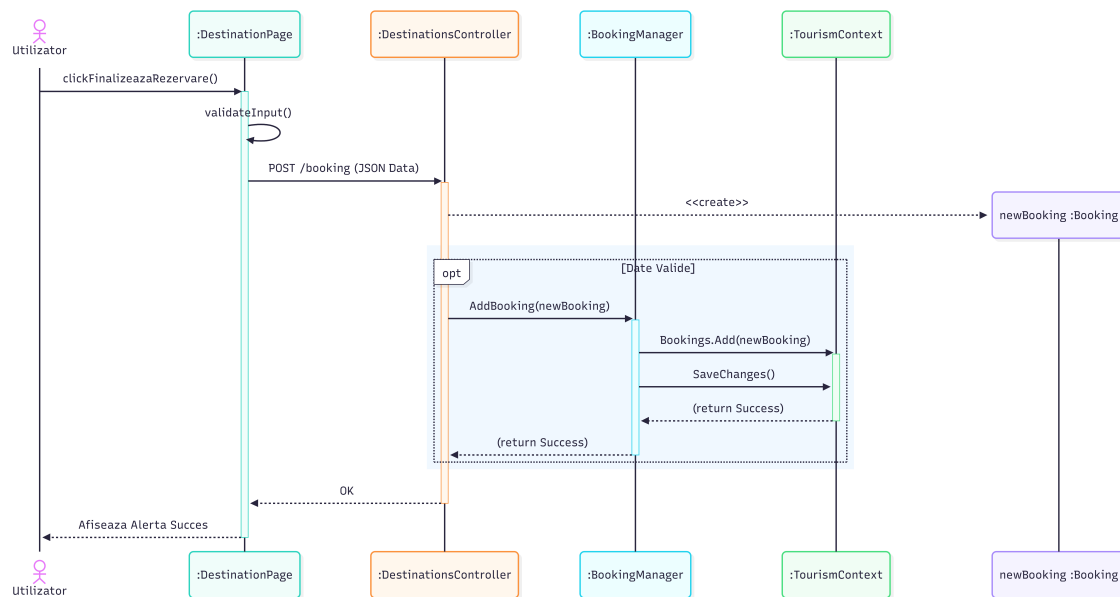


Figura 3.3: Diagrama de Secvență - Procesul de Rezervare

Fluxul de execuție este următorul:

- **Inițierea acțiunii:** Totul începe când utilizatorul apasă butonul „Finalizează Rezervare” în interfața grafică.
- **Validare locală:** Înainte de a trimite datele, pagina verifică intern dacă totul este completat corect.
- **Trimiterea datelor:** Dacă validarea trece, aplicația trimite o cerere de tip POST către server.
- **Procesarea pe Server:**
  - *DestinationsController* primește cererea.
  - Acesta apelează **BookingManager** (Singleton) pentru a procesa comanda.
  - *BookingManager* adaugă rezervarea atât în memorie, cât și în contextul bazei de date (*TourismContext*), apelând funcția *SaveChanges*.

- **Confirmarea:** Baza de date confirmă salvarea, serverul trimite răspunsul „OK”, iar utilizatorul vede o alertă de succes.

### 3.5 Diagrama de Comunicare

Diagrama de comunicare evidențiază structura legăturilor dintre componente și mesajele schimbate pentru a afișa pagina unei singure destinații.

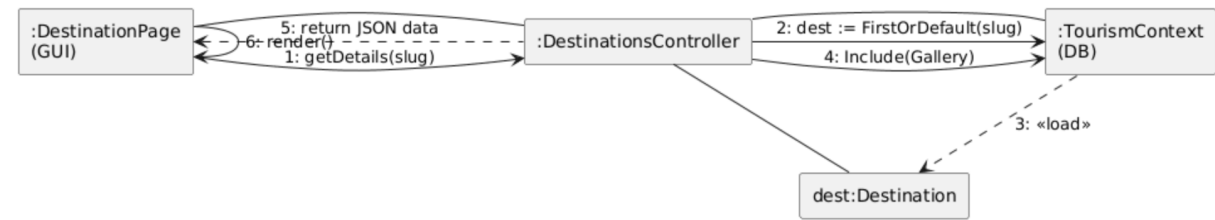


Figura 3.4: Diagrama de Comunicare - Obținerea detaliilor

Procesul ilustrat respectă următoarea logică:

- **1. Solicitarea (getDetails):** Interfața trimite o cerere către controller, furnizând slug-ul destinației.
- **2. Interogarea (FirstOrDefault):** Controllerul comunică cu contextul bazei de date pentru a găsi destinația.
- **3. Încărcarea datelor (load):** Baza de date identifică înregistrarea și încarcă obiectul în memorie.
- **4. Includerea relațiilor (Include):** Controllerul solicită explicit includerea datelor asociate (Galeria foto).
- **5. Răspunsul (return JSON):** După ce datele au fost colectate, controllerul le trimite înapoi către interfață.
- **6. Afișarea (render):** Interfața primește datele și actualizează pagina cu titlul, descrierea și imaginile.

# Capitolul 4

## Implementarea Soluției

### 4.1 Implementarea Backend-ului

#### 4.1.1 Configurarea bazei de date și Seeder-ul

Nucleul aplicației din punct de vedere al datelor este reprezentat de clasa `TourismContext`, care moștenește funcționalitățile **Entity Framework Core**. S-a implementat mecanismul de *Data Seeding* prin clasa `DbSeeder`, care populează automat baza de date cu destinații demonstrative la prima rulare, prevenind duplicarea datelor dacă acestea există deja.

#### 4.1.2 Design Pattern-uri: Singleton

**Problema:** Era necesară o metodă centralizată pentru gestionarea rezervărilor, care să asigure unicitatea și consistența datelor, fără a crea obiecte noi pentru fiecare cerere.

**Soluția:** S-a utilizat modelul de proiectare **Singleton** pentru clasa `BookingManager`.

**Implementare:**

- Aplicația menține o **singură instanță** a managerului în memorie.
- Toate cererile de rezervare venite din Controllere sunt direcționate către acest Manager.
- Managerul salvează datele simultan în două locuri: în memoria RAM (pentru acces rapid) și în baza de date SQL (pentru persistență), folosind un mecanism de blocare (*lock*) pentru siguranță.

#### 4.1.3 API Controllers

Serverul comunică cu interfața grafică prin Controllere care expun rute specifice:

- **GET /api/destinations:** Returnează lista ofertelor.
- **GET /api/destinations/{slug}:** Returnează detaliile complete ale unei oferte.
- **POST /api/bookings:** Primește datele din formular și apelează Singleton-ul pentru salvare.

## 4.2 Implementarea Frontend-ului

### 4.2.1 Componente și State

Aplicația React este modulară, folosind componente reutilizabile precum `DestinationCard`, `Navbar` sau `BookingForm`. Pentru gestionarea datelor dinamice, s-au utilizat Hooks:

- **useState:** Pentru memorarea listei de destinații și a stării formularelor.
- **useEffect:** Pentru încărcarea automată a datelor de la server la deschiderea paginii.

### 4.2.2 Comunicarea cu serverul

Legătura dintre interfață și server se realizează asincron, folosind funcția **fetch**. Acest lucru permite ca interfața să rămână receptivă în timp ce datele sunt încărcate în fundal.

# Capitolul 5

## Concluzii și Utilizare

### 5.1 Concluzii

Realizarea acestui proiect a constituit o etapă esențială în înțelegerea procesului complet de creare a unei aplicații web. Rezultatul final este o platformă funcțională, care îmbină un Backend performant cu o interfață modernă.

S-a demonstrat importanța separării logicii de afișare de cea de procesare (Client-Server). Utilizarea pattern-ului Singleton a asigurat o gestionare eficientă a rezervărilor, centralizând logica de salvare și eliminând redundanța codului.

### 5.2 Dezvoltări ulterioare

- Implementarea unui sistem de autentificare pentru utilizatori.
- Integrarea plăților online.
- Crearea unui panou de administrare pentru gestionarea ofertelor.

### 5.3 Scurt Read Me (Ghid de Pornire)

Pentru a rula aplicația local:

1. **Backend:** Configurați conexiunea la SQL Server în `appsettings.json` și rulați proiectul .NET. Serverul va popula automat baza de date.
2. **Frontend:** În folderul proiectului React, rulați `npm install` urmat de `npm run dev`. Aplicația va fi accesibilă la `http://localhost:5173`.

# Bibliografie

- Microsoft Learn - Documentația oficială .NET:  
<https://learn.microsoft.com/en-us/dotnet/>
- React.js - Documentația Oficială:  
<https://react.dev/>
- Entity Framework Core - Tutoriale:  
<https://learn.microsoft.com/en-us/ef/core/>
- Pexels - Resurse grafice:  
<https://www.pexels.com/>