

**UNIVERSITATEA DIN CRAIOVA**  
**FACULTATEA DE ȘTIINTE**  
**SPECIALIZAREA INFORMATICĂ**

**LUCRARE DE DIPLOMĂ**

Îndrumător științific:  
**Asist. Dr. Tudorache Cristina**

Absolvent:  
**Stoentel Alexandru-Eduard**

**CRAIOVA**

2021

**UNIVERSITATEA DIN CRAIOVA**  
**FACULTATEA DE ȘTIINTE**  
**SPECIALIZAREA INFORMATICĂ**

**Platformă de blogging realizată cu tehnologii web și  
baze de date**

Îndrumător științific:

**Asist. Dr. Tudorache Cristina**

Absolvent:

**Stoentel Alexandru-Eduard**

**CRAIOVA**

2021

*„Nu există vânt favorabil decât pentru cei  
care știu în ce direcție se îndreaptă”*

**Seneca**

**DECLARAȚIE DE ORIGINALITATE**

Subsemnatul Stoentel Alexandru-Eduard, student la specializarea Informatică din cadrul Facultății de Științe a Universității din Craiova, certific prin prezenta că am luat la cunoștință de cele prezentate mai jos și că îmi asum, în acest context, originalitatea proiectului meu de licență:

- cu titlul Platformă de blogging realizată cu tehnologii web și baze de date,
- coordonată de Asist. Dr. Tudorache Cristina,
- prezentată în sesiunea IULIE 2024.

La elaborarea proiectului de licență, se consideră plagiat una dintre următoarele acțiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele și referința precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obținute sau a unor aplicații realizate de alți autori fără menționarea corectă a acestor surse,
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situații neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii și corespunzător în lista de referințe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, etc.,
- precizarea referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data,

Semnătura candidatului,



UNIVERSITATEA DIN CRAIOVA

Facultatea de Științe

Departamentul de Informatică

Aprobat la data de .....

Șef de departament,

Lect. Dr. Stoian

Gabriel

**LUCRARE DE DIPLOMĂ**

Numele și prenumele studentului/-ei:	Stoentel Alexandru-Eduard
Enunțul temei:	Platformă de blogging realizată cu tehnologii web și baze de date
Datele de pornire:	<p>Pentru dezvoltarea proiectului de licență am avut ca punct de plecare:</p> <ul style="list-style-type: none"> <li>- modele și informații din diversele aplicații web de blogging.</li> <li>- Secolul în care majoritatea populației caută informații și dorește să fie la curent cu știrile de actualitate</li> </ul>
Conținutul proiectului:	Structura proiectului va cuprinde următoarele secțiuni: introducere, tehnologii și framework-uri folosite, componente software utilizate, specificații și reprezentarea aplicației, dezvoltarea aplicației, implementarea aplicației și concluzii.
Material grafic obligatoriu:	Diagrame, scheme, capturi.
Consultații:	Periodice

Conducătorul științific (titlul, nume și prenume, semnătura):	Asist. Dr. Tudorache Cristina
Data eliberării temei:	1.12.2023
Termenul estimat de predare a proiectului:	20.06.2024
Data predării proiectului de către student și semnătura acestuia:	

***Cuvinte cheie:*** blog, știri, informație, Blazor, .NET, SQL Server.

## Recunoștință

În primul rând, doresc să îmi exprim recunoștința și mulțumirile întregului corp profesoral al acestei facultăți. De-a lungul acestor patru ani, prin implicarea și devotamentul lor, prin temele provocatoare și examenele semestriale, m-au ajutat să devin persoana harnică pe care am aspirat întotdeauna să o fiu.

De asemenea, doresc să mulțumesc coordonatorului meu de lucrare, doamnei Tudorache Cristina, căruia îi sunt profund recunoscător pentru sprijinul acordat. A fost mereu alături de mine, oferindu-mi sfaturi, idei și perspective noi. Domnul profesor a oferit întotdeauna feedback pozitiv și a fost întotdeauna disponibil pentru a răspunde întrebărilor mele. Vă sunt recunoscător și vă mulțumesc pentru tot sprijinul acordat!

Dar și profesorilor care mi-au fost mereu alături și nu au ezitat să mă susțină și să mă învețe tot ce știau ei și au avut o contribuție semnificativă în formarea mea profesională și personală. Fiecare dintre acești profesori mi-a oferit nu doar cunoștințe teoretice, ci și perspective practice valoroase, care m-au ajutat să îmi dezvolt abilitățile și să mă pregătesc pentru viitoarea mea carieră. Fără implicarea și dedicarea lor, nu aș fi reușit să ajung la acest nivel de pregătire. Le sunt profund recunoscător pentru tot ceea ce au făcut pentru mine.

Nu în ultimul rând, doresc să mulțumesc părinților și colegilor mei. Părinții mei m-au sprijinit necondiționat în orice situație, încurajându-mă mereu să fiu o persoană dedicată și devotată. Alături de colegi, am trăit cea mai bună experiență profesională și personală în acești trei ani.

Sprijinul academic pe care l-am primit din partea profesorului meu coordonator a fost esențial. El a fost un mentor de la care am învățat multe lucruri valoroase, inclusiv ce înseamnă motivația și dorința de succes.



## Cuprins

1. INTRODUCERE .....	11
1.1 Scop principal .....	11
1.2 Motivație .....	11
2. TEHNOLOGII ȘI FRAMEWORK-URI FOLOSITE.....	13
2.1 C#.....	13
2.2 .NET.....	14
2.3 ASP.NET Web API .....	14
2.4 Entity Framework .....	15
2.4 Identity .....	19
2.5 LINQ .....	19
2.5 HyperText Markup Language (HTML) .....	20
2.6 Cascading Style Sheets (CSS).....	21
2.7 Blazor .....	21
2.8 Blazorise .....	23
2.9 Bootstrap .....	23
2.10 Refit API .....	24
2.11 Structured query language (SQL) .....	24
2.12 NuGet Package Console .....	25
3. ELEMENTE SOFTWARE FOLOSITE .....	26
3.1 Visual Studio.....	26
3.2 SQL Server Management Studio 20 .....	26
3.3 Postman.....	27
4. SPECIFICAȚII ȘI REPREZENTAREA APLICAȚIEI.....	28
4.1 Specificații funcționale .....	28
4.2 Specificații tehnice.....	29
4.3 Diagramele cazurilor de utilizare .....	35
4.4 Organizarea bazei de date .....	36
5. DEZVOLTAREA APLICAȚIEI .....	38
5.1 Popularea bazei de date.....	38
5.2 Modele de baze de date Entity Framework.....	41
5.3 Sistemul de autentificarea .....	45
5.4 Sistemul de blogging.....	48
5.5 Sistemul de comentarii.....	52
6. DESFĂȘURAREA APLICAȚIEI .....	54
7. CONCLUZIE.....	61

8. BIBLIOGRAFIE.....	62
9. REFERINȚE.....	63

# 1. INTRODUCERE

## 1.1 Scop principal

Această documentație reprezintă rezultatul eforturilor mele de cercetare pe parcursul a trei ani de studiu în domeniul Informaticii, la Facultatea de Științe din Craiova. Scopul principal al acestei documentații este să ofere informații detaliate despre arhitectura proiectului, tehnologiile utilizate și funcționalitățile de bază ale acestuia.

Proiectul este un sistem blogging, bazat pe scrierea unei postări, cu scopul de a informa utilizatorii. Platforma este concepută pentru a oferi utilizatorilor o experiență simplă și intuitivă de redactare a articolelor, facilitând procesul de creare și publicare a conținutului pe internet. Scopul principal al acestui sistem este să ofere o modalitate eficientă de comunicare și distribuire a informațiilor, promovând schimbul de idei și experiențe între utilizatori și contribuind la dezvoltarea unei comunități active și informate.

Pe lângă aspectele conceptuale, documentația urmărește să explice și motivul tehnic al începerii proiectului, precum și modul în care acesta poate aduce valoare comunității noastre. Consider că această aplicație va testa abilitățile mele în diverse domenii ale dezvoltării software, oferindu-mi motivația necesară pentru a găsi soluții eficiente și pentru a adopta noi tehnologii atunci când este necesar. Este o provocare care ne împinge întotdeauna să ne dăm tot ce e mai bun din noi și să ne depășim limitele.

## 1.2 Motivație

Studiile și analizele desfășurate pe parcursul timpului au demonstrat că platformele de blogging au un impact semnificativ în mediul online, fiind considerate surse de informații esențiale și instrumente de comunicare eficiente pentru utilizatori.

Unul dintre principalele avantaje ale unui sistem de blogging este accesibilitatea și ușurința în a distribui informații. Comparativ cu alte forme de comunicare și distribuire a conținutului, un blog oferă posibilitatea de a crea și partaja conținut într-un mod rapid și simplu, fără a fi nevoie de cunoștințe tehnice avansate.

De asemenea, blogging-ul oferă oportunitatea de a construi și de a consolida comunități online, aducând împreună persoane cu interese comune și creând un spațiu propice pentru schimbul de idei, opinii și experiențe. Astfel, platforma noastră de blogging este concepută

pentru a sprijini acest proces de conectare și colaborare între utilizatori, facilitând interacțiunea și schimbul de cunoștințe într-un mediu virtual prietenos și deschis.

În final, platforma noastră de blogging este menită să ofere o experiență plăcută și valoroasă utilizatorilor, oferindu-le un spațiu dedicat pentru exprimare și comunicare, încurajându-i să împărtășească idei, să dezbată subiecte importante și să contribuie la dezvoltarea unei comunități active și informate.

## 2. TEHNOLOGII ȘI FRAMEWORK-URI FOLOSITE

### 2.1 C#

C# este un limbaj de programare modern, orientat pe obiect, dezvoltat de Microsoft ca parte a platformei .NET. Este utilizat pe scară largă pentru dezvoltarea de aplicații enterprise, aplicații web, jocuri și multe altele. În contextul dezvoltării acestei aplicații de blogging, C# joacă un rol central în realizarea componentelor back-end și a logicii aplicației, oferind o serie de avantaje.

C# este un limbaj complet orientat pe obiect, ceea ce înseamnă că suportă concepte OOP precum clase, obiecte, moștenire, polimorfism și încapsulare. Acest lucru facilitează scrierea unui cod modular, reutilizabil și ușor de întreținut.

C# este un limbaj puternic tipizat, ceea ce înseamnă că fiecare variabilă trebuie declarată cu un tip specific. Acest lucru ajută la detectarea erorilor la timp de compilare, îmbunătățind astfel fiabilitatea și siguranța codului.

Programarea Orientată pe Obiecte (OOP) este un paradigmă de programare care utilizează "obiecte" și "clase" pentru a structura și organiza codul. Aceasta paradigmă este esențială pentru dezvoltarea de software scalabil și modular. OOP este fundamentată pe patru principii de bază: încapsularea, moștenirea, polimorfismul și abstractizarea.

Principiile de baza ale programării orientată pe obiecte sunt:

- Încapsularea: se referă la restricționarea accesului la anumite componente ale unui obiect și la protejarea stării interne a acestuia. Aceasta se realizează prin declararea variabilelor de clasă ca private și furnizarea de metode publice pentru accesarea și modificarea acestor variabile.
- Moștenirea: permite crearea de noi clase (clase derivate) care împrumută atribute și metode de la o clasă existentă (clasa de bază)
- Polimorfismul: permite utilizarea unei metode într-o clasă de bază pentru a fi redefinită sau suprascrisă într-o clasă derivată. Acesta poate fi de două tipuri: polimorfism la timp de compilare (suprasarcină) și polimorfism la timp de execuție (suprascriere).

- .NET este un framework software dezvoltat de Microsoft care oferă un mediu unificat pentru dezvoltarea și rularea aplicațiilor. Acesta suportă o gamă largă de aplicații, inclusiv aplicații desktop, web, mobile, cloud, jocuri și IoT (Internet of Things). .NET este cunoscut pentru interoperabilitatea sa, performanța ridicată și suportul puternic pentru limbajele de programare moderne.
- Abstractizarea: se referă la ascunderea detaliilor complexe de implementare și la expunerea doar a funcționalităților esențiale prin intermediul claselor abstracte și al interfețelor

## **2.2 .NET**

### **2.3 ASP.NET Web API**

ASP.NET Web API <sup>1</sup>este un framework <sup>2</sup>pentru construirea și consumul de servicii HTTP care pot fi accesate de la diverse clienți, inclusiv browsere web, aplicații mobile și aplicații desktop. ASP.NET Web API este o parte a platformei .NET și oferă un set robust de instrumente și funcționalități pentru crearea API-urilor web RESTful. REST este un stil de arhitectură pentru a dezvolta servicii web, care utilizează protocolul HTTP ca interfață de comunicare pentru a transfera date prin metode HTTP.

ASP.NET Web API este proiectat pentru a construi servicii RESTful, care se bazează pe principiile de transfer de stare reprezentativ (REST). Aceste servicii sunt caracterizate prin utilizarea metodelor HTTP standard (GET, POST, PUT, DELETE) pentru operații CRUD (Create, Read, Update, Delete).

Framework-ul oferă mecanisme puternice pentru legarea datelor din cererile HTTP la modele de date și pentru validarea acestor date. Decorarea modelelor cu atribute de validare asigură că datele primite respectă regulile specificate înainte de a fi procesate.

ASP.NET Web API suportă diverse metode de autentificare și autorizare, inclusiv utilizarea de token-uri JWT (JSON Web Token), Identity, OAuth și alte mecanisme de securitate.

---

<sup>1</sup> Application Programming Interface (API) reprezintă un set de definiții de sub-programe, protocoale și unelte pentru programarea de aplicații și software.

<sup>2</sup> În dezvoltarea de software un framework este o structură conceptuală și reprezintă o arhitectură de software care modelează relațiile generale ale entităților domeniului (site-ului).

## **2.4 Entity Framework**

Entity Framework (EF) este un Object-Relational Mapper (ORM) open-source pentru .NET, care simplifică accesul la baze de date și manipularea datelor prin maparea obiectelor din aplicație la tabelele din baza de date<sup>3</sup>. EF permite dezvoltatorilor să interacționeze cu baza de date folosind obiecte .NET, eliminând necesitatea de a scrie cod SQL manual pentru majoritatea operațiunilor de date.

EF facilitează maparea obiectelor din modelul de date (entități) la tabelele din baza de date. Aceasta permite utilizarea limbajului de programare C# pentru a efectua operațiuni de date.

Entity Framework permite utilizarea Language Integrated Query (LINQ) pentru interogarea bazei de date. LINQ oferă o sintaxă intuitivă și puternică pentru scrierea interogărilor.

EF suportă trei metode principale de dezvoltare:

- Model First: Dezvoltarea începe cu un model conceptual care este utilizat pentru a genera baza de date.
- Database First: Baza de date existentă este utilizată pentru a genera modelul de date.
- Code First: Modelul de date este definit folosind clase C#, iar baza de date este generată din acest model.

Migrația este o caracteristică care permite evoluția schemei bazei de date în timp. Dezvoltatorii pot adăuga, modifica sau elimina tabele și coloane, iar EF va genera scripturi SQL necesare pentru aplicarea acestor modificări.

Entity Framework facilitează crearea relațiilor dintre entități, astfel avem relații: one-to-one, one-to-many și many-to-many.

O relație *one-to-one* apare atunci când fiecare rând dintr-o tabelă este asociat cu un singur rând dintr-o altă tabelă. De exemplu, un utilizator poate avea un profil unic asociat. În C# acest lucru se poate face fie prin proprietățile de navigare, fie prin referențierea virtuală dintre cele două entități.

---

<sup>3</sup> O bază de date este o colecție organizată de date care sunt stocate și accesate electronic. Bazele de date sunt esențiale pentru gestionarea și stocarea datelor în diverse aplicații și sisteme informatice, facilitând accesul rapid și eficient la informații.

```

public class Student
{
    public int StudentId { get; set; }
    public string StudentName { get; set; }

    public virtual StudentAddress Address { get; set; }
}

public class StudentAddress
{
    public int StudentAddressId { get; set; }
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public int Zipcode { get; set; }
    public string State { get; set; }
    public string Country { get; set; }

    public virtual Student Student { get; set; }
}

```

### 1. one-to-one folosind entitățile

```

public class Student
{
    public int StudentId { get; set; }
    public string StudentName { get; set; }

    public virtual StudentAddress Address { get; set; }
}

public class StudentAddress
{
    [ForeignKey("Student")]
    public int StudentAddressId { get; set; }

    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public int Zipcode { get; set; }
    public string State { get; set; }
    public string Country { get; set; }

    public virtual Student Student { get; set; }
}

```

### 2. one-to-one folosind entitățile și specificând ForeignKey-ul



```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    // Configure Student & StudentAddress entity
    modelBuilder.Entity<Student>()
        .HasOptional(s => s.Address) // Mark Address property optional in Student entity
        .WithRequired(ad => ad.Student); // mark Student property as required in StudentAddress entity. Cannot save Stu
}
```

### 3. one-to-one folosind Fluent API

O relație *one-to-many* apare atunci când un rând dintr-o tabelă poate fi asociat cu mai multe rânduri dintr-o altă tabelă. De exemplu, un user poate avea mai multe blog-uri.

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public Grade Grade { get; set; }
}

public class Grade
{
    public int GradeID { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Student { get; set; }
}
```

### 4. one-to-many folosind entitățile

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }

    public int GradeId { get; set; }
    public Grade Grade { get; set; }
}

public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }

    public ICollection<Student> Student { get; set; }
}
```

### 5. one-to-many folosind entitățile și specificând ForeignKey-ul

```

modelBuilder.Entity<Grade>()
    .HasMany<Student>(g => g.Students)
    .WithRequired(s => s.CurrentGrade)
    .HasForeignKey<int>(s => s.CurrentGradeId);

```

### 6. one-to-one folosind Fluent API

O relație *many-to-many* apare atunci când mai multe rânduri dintr-o tabelă pot fi asociate cu mai multe rânduri dintr-o altă tabelă. De exemplu, userii pot avea mai multe roluri, iar aceleași roluri pot fi atribuite mai multor useri.

```

public class Student
{
    public Student()
    {
        this.Courses = new HashSet<Course>();
    }

    public int StudentId { get; set; }
    [Required]
    public string StudentName { get; set; }

    public virtual ICollection<Course> Courses { get; set; }
}

public class Course
{
    public Course()
    {
        this.Students = new HashSet<Student>();
    }

    public int CourseId { get; set; }
    public string CourseName { get; set; }

    public virtual ICollection<Student> Students { get; set; }
}

```

### 7. one-to-many folosind entitățile

```

modelBuilder.Entity<Student>()
    .HasMany<Course>(s => s.Courses)
    .WithMany(c => c.Students)
    .Map(cs =>
    {
        cs.MapLeftKey("StudentRefId");
        cs.MapRightKey("CourseRefId");
        cs.ToTable("StudentCourse");
    });

```

### 8. one-to-one folosind Fluent API

## 2.4 Identity

ASP.NET Core Identity este un sistem de autentificare și autorizare care permite dezvoltatorilor să gestioneze utilizatorii, parolele, rolurile și permisiunile într-o aplicație web. Identity este proiectat pentru a fi extensibil și personalizabil, oferind în același timp o securitate robustă și funcționalități avansate pentru gestionarea identităților utilizatorilor.

Identity oferă mecanisme pentru autentificarea utilizatorilor în aplicație și autorizarea acestora pentru accesarea resurselor protejate. Aceste funcționalități sunt integrate cu middleware-ul de autentificare al ASP.NET Core.

Identity gestionează detaliile utilizatorilor, inclusiv numele de utilizator, parolele și alte informații relevante. Parolele sunt stocate în mod securizat folosind hashing și salting.

Identity pune la dispoziție un set prestabilit de endpoint-uri, disponibile pentru Web API. El vine cu un service-uri și repository-uri deja predefinite pentru a administra utilizatorii și rolurile:

```
16 private readonly SignInManager<DataBaseLayout.Models.User> _signInManager;  
17 private readonly UserManager<DataBaseLayout.Models.User> _userManager;
```

## 2.5 LINQ

*Language Integrated Query (LINQ)* este o componentă puternică a limbajului C# și a platformei .NET, care oferă o sintaxă coerentă și ușor de utilizat pentru interogarea și manipularea colecțiilor de date. LINQ permite dezvoltatorilor să scrie interogări puternice și eficiente direct în codul lor C#, fie că lucrează cu baze de date, colecții în memorie, XML, sau alte surse de date.

LINQ oferă o sintaxă unificată pentru interogarea diferitelor surse de date. Aceasta face ca interogările să fie mai intuitive și mai ușor de citit, indiferent de tipul de date cu care se lucrează.

LINQ este integrat direct în limbajul C#, ceea ce înseamnă că dezvoltatorii pot folosi operatori familiari și expresii lambda pentru a construi interogări.

LINQ oferă un set bogat de operatori standard pentru filtrare, proiecție, grupare, unire și alte operațiuni comune. Acești operatori includ *Where*, *Select*, *GroupBy*, *Join*, *OrderBy*, *Sum*, *Average*, *Count*, și altele.

## 2.5 HyperText Markup Language (HTML)

*HyperText Markup Language (HTML)* este limbajul standard folosit pentru a crea și structura paginile web<sup>4</sup>. HTML folosește un set de elemente (denumite și tag-uri) pentru a defini diferitele părți ale unei pagini web, cum ar fi paragrafele, imaginile, linkurile și multe altele. Aceste elemente sunt înconjurate de acolade unghiulare (<>) și sunt în general pereche, cu un tag de deschidere și unul de închidere.

Cele mai importante tag-uri care se regăsesc și în proiectul meu sunt:

- <html> - elementul rădăcină care înconjoară întregul document HTML.
- <head> - conține meta-informații despre document, cum ar fi titlul paginii, linkurile către foi de stil și scripturi.
- <title> - definește titlul paginii, afișat în bara de titlu a browserului sau pe fila paginii.
- <body> - conține conținutul vizibil al paginii web, cum ar fi textul, imaginile, formularele și alte elemente.
- <p> - definirea paragrafelor.
- <img> - includerea imaginilor. Atributul src specifică calea imaginii, iar alt oferă o descriere a acesteia.
- <div> și <span> - elemente generice pentru blocuri și inline, utilizate pentru a grupa alte elemente și a aplica stiluri CSS.

```
4
5 <!DOCTYPE html>
6 <html lang="en">
7 <head>
8   <meta charset="utf-8" />
9   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
10  <base href="/" />
11  <link href="content/Blazorise.Icons.FontAwesome/v6/css/all.min.css" rel="stylesheet">
12  <link href="content/Blazorise/blazorise.css" rel="stylesheet" />
13  <link href="content/Blazorise.Bootstrap/blazorise.bootstrap.css" rel="stylesheet" />
14  <link href="content/Blazorise.Snackbar/blazorise.snackbar.css" rel="stylesheet" />
15  <link href="content/Blazorise.Spinkit/blazorise.spinkit.css" rel="stylesheet" />
16  <link href="content/Blazorise.LoadingIndicator/blazorise.loadingindicator.css" rel="stylesheet" />
17
```

---

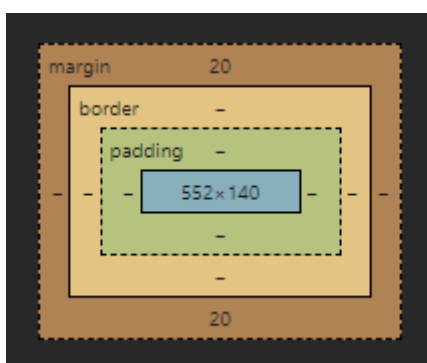
<sup>4</sup> World Wide Web (pe scurt web sau www) este un sistem hipertext care operează pe Internet. Hipertextul este vizualizat cu un program numit browser, care descarcă paginile web de pe un server web (sau site web) și îl afișează pe ecran.

## **2.6 Cascading Style Sheets (CSS)**

*Cascading Style Sheets (CSS)* este un limbaj de stil folosit pentru a descrie prezentarea unui document scris într-un limbaj de marcare, cum ar fi HTML. CSS controlează aspectul și formatarea elementelor HTML pe o pagină web, inclusiv layout-ul, culorile, fonturile și alte aspecte vizuale. Prin separarea conținutului de prezentare, CSS permite o gestionare mai ușoară și flexibilă a designului web.

Layout-ul unui tag HTML la care se aplică CSS este modul în care acesta este poziționat și aranjat în cadrul DOM-ului. Acesta este determinat de regulile CSS aplicate tag-ului, care pot controla dimensiunea, poziția, spațierea și alte aspecte ale elementului.

Layoutul este format din: margin, border, padding și content:



## **2.7 Blazor**

*Blazor* este un framework open-source dezvoltat de Microsoft pentru crearea de aplicații web interactive, single-page applications (SPA), utilizând .NET și C#. Blazor permite dezvoltatorilor să construiască interfețe de utilizator web moderne și interactive fără a folosi JavaScript pentru logica de client. Blazor folosește C# atât pe server, cât și pe client, împreună cu componente reutilizabile, care sunt asemănătoare cu componentele din alte framework-uri de frontend, cum ar fi React sau Angular.

Blazor permite dezvoltarea de aplicații web folosind C# și .NET, atât pe partea de client, cât și pe cea de server. Aceasta oferă avantajul de a utiliza un singur limbaj de programare pentru întregul stack al aplicației.

Blazor folosește un model bazat pe componente, unde fiecare componentă este o unitate reutilizabilă de interfață de utilizator, ce poate include atât markup HTML, cât și logică de interacțiune în C#.

Blazor oferă două modele principale de hosting:

- Blazor Server: Execută logica aplicației pe server și folosește SignalR<sup>5</sup> pentru a comunica actualizările UI cu browserul.
- Blazor WebAssembly: Execută aplicația complet în browser folosind WebAssembly. Aceasta permite aplicațiilor Blazor să ruleze direct pe client fără a depinde de server pentru logica de interacțiune.

Blazor suportă binding bidirecțional al datelor, permițând sincronizarea automată a datelor între componenta UI și modelul de date. În Blazor, o componentă este reprezentată printr-o clasă C# care extinde clasa *ComponentBase*. Această clasă reprezintă o unitate autonomă și reutilizabilă de interfață utilizator, care poate fi utilizată în cadrul aplicației Blazor pentru a afișa conținut și pentru a interacționa cu utilizatorul.

O componentă Blazor este definită într-un fișier cu extensia *.razor*<sup>6</sup>, care conține atât cod C# cât și markup HTML. Acest fișier combină logica și prezentarea componentei într-un singur loc. În interiorul fișierului *.razor*, este definită o clasă C# care extinde clasa *ComponentBase*. Această clasă conține logica componentei, inclusiv metodele de inițializare, manipulare a evenimentelor și alte funcționalități specifice. În fișierul *.razor*, este inclus și markup-ul HTML care reprezintă aspectul vizual al componentei. Acest markup poate conține elemente HTML standard, împreună cu directivele specifice Blazor pentru atașarea de evenimente, legături de date și alte funcționalități.

Proprietăți și Parametri: Componentele Blazor pot avea proprietăți și parametri, care sunt utilizate pentru a transmite date către componentă și pentru a personaliza comportamentul acesteia. Acestea pot fi definite ca proprietăți ale clasei componentei și pot fi utilizate în interiorul markup-ului HTML pentru a influența afișarea și funcționarea componentei.

```
[Parameter]
public EventCallback<Guid> ItemClicked { get; set; }

<BlogCard Item="@blog" ItemClicked="@BlogClicked"></BlogCard>
```

<sup>5</sup> SignalR este o bibliotecă inclusă în ASP.NET și ASP.NET Core care facilitează adăugarea de funcționalități de comunicație în timp real în aplicațiile web. Utilizând SignalR, dezvoltatorii pot crea aplicații care necesită o comunicare bidirecțională între server și client, cum ar fi chat-urile live, tablourile de bord de monitorizare, jocurile online și alte aplicații interactive.

<sup>6</sup> Razor este un motor de vizualizare folosit în cadrul ASP.NET pentru a genera pagini web dinamice. Este utilizat în principal în ASP.NET Core și ASP.NET MVC pentru a crea interfețe web prin combinarea codului C# cu markup-ul HTML

## 2.8 Blazorise

*Blazorise* este o bibliotecă de componente UI<sup>7</sup> pentru Blazor, care oferă un set de componente stilizate și interactive predefinite pentru dezvoltarea rapidă a aplicațiilor web Blazor. Blazorise simplifică crearea de interfețe de utilizator atractive și funcționale, eliminând nevoia de a scrie cod CSS și JavaScript personalizat pentru fiecare componentă.

Blazorise oferă o gamă largă de componente UI, cum ar fi butoane, casete de text, dropdown-uri, liste, paginatoare, meniuri și multe altele, toate stilizate și gata de utilizare.

Fiecare componentă Blazorise vine cu funcționalitate încorporată, cum ar fi gestionarea *evenimentelor de click, hover și focus, validarea datelor de intrare, gestionarea paginării și sortării*, facilitând dezvoltarea aplicațiilor web interactive.

Biblioteca Blazorise permite crearea de teme personalizate sau utilizarea unor teme predefinite, precum *Bootstrap* sau *Material Design*, oferind o experiență coerentă și stilizată pentru aplicațiile Blazor.

## 2.9 Bootstrap

*Bootstrap* este unul dintre cele mai populare framework-uri front-end utilizate pentru dezvoltarea rapidă a interfețelor de utilizator web. Creat inițial de către Twitter, Bootstrap este acum o resursă open-source și este utilizat pe scară largă de dezvoltatori din întreaga lume pentru construirea de site-uri web responsive și atrăgătoare.

Bootstrap oferă un sistem de grilă flexibil, bazat pe un sistem de coloane și rânduri, care facilitează crearea de layout-uri responsive și adaptabile pentru diverse dimensiuni de ecran.

---

<sup>7</sup> UI este prescurtarea de la user interface (interfață de utilizator). Este aspectul grafic al unei aplicații sau al unui website și se referă la toate elementele care vin în contact cu utilizatorii: butoane, text, imagini, câmpuri pentru introducere text, slide-uri și așa mai departe.

Bootstrap include o serie de componente UI predefinite, cum ar fi butoane, casete de text, dropdown-uri, bare de navigare, carduri, alerte și multe altele, care sunt stilizate și gata de utilizare.

Bootstrap oferă o gamă largă de clase utilitare CSS pentru gestionarea marginilor, padding-urilor, alinierii, culorilor și multe altele, care facilitează stilizarea și formatarea rapidă a elementelor HTML.

Bootstrap permite personalizarea aspectului și stilului componentelor prin intermediul temelor predefinite sau prin crearea de teme personalizate, permițând dezvoltatorilor să creeze designuri unice și distinctive.

Bootstrap este compatibil cu cele mai recente versiuni ale principalelor browsere web și asigură o experiență consistentă pentru utilizatorii din întreaga lume.

## **2.10 Refit API**

*Refit* este o bibliotecă .NET care simplifică integrarea API-urilor REST în aplicațiile .NET. Creată de Paul Betts, Refit oferă o modalitate elegantă și ușor de utilizat pentru definirea și apelarea serviciilor web bazate pe HTTP, fără a fi nevoie să se scrie cod boilerplate pentru comunicare sau serializare/deserializare a datelor JSON<sup>8</sup>.

Refit permite definirea interfețelor de serviciu în stilul .NET, care declară metode pentru fiecare endpoint al API-ului. Aceste interfețe servesc ca contracte pentru comunicarea cu serviciul web.

Folosind anotările de atribut, dezvoltatorii pot specifica detalii precum URL<sup>9</sup>-ul, metoda HTTP și alte opțiuni pentru fiecare metodă din interfața de serviciu.

Refit automat gestionează serializarea și deserializarea datelor JSON, fără a fi nevoie să se scrie cod suplimentar pentru aceste operațiuni.

Refit suportă toate metodele HTTP standard, inclusiv GET, POST, PUT, DELETE etc., facilitând comunicarea cu API-uri RESTful.

## **2.11 Structured query language (SQL)**

*Structured Query Language (SQL)* este un limbaj de programare specializat utilizat pentru gestionarea datelor în sisteme de management al bazelor de date relaționale (RDBMS).

---

<sup>8</sup> JSON este un format text, inteligibil pentru oameni, utilizat pentru reprezentarea obiectelor și a altor structuri de date și este folosit în special pentru a transmite date structurate prin rețea, procesul purtând numele de serializare

<sup>9</sup> URL - localizator uniform de resurse este o secvență de caractere standardizată, folosită pentru denumirea, localizarea și identificarea unor resurse de pe Internet, inclusiv documente text, imagini, clipuri video, expuneri de diapozitive etc.



SQL permite utilizatorilor să efectueze diverse operațiuni asupra datelor, inclusiv interogări, inserări, actualizări și ștergeri.

În cadrul unei baze de date relaționale, relațiile dintre tabele sunt definite prin intermediul cheilor străine și primare. Iată câteva tipuri de relații comune:

- *One-to-One*: Acest tip de relație apare atunci când fiecare înregistrare dintr-o tabelă este asociată cu exact o înregistrare dintr-o altă tabelă și invers.
- *One-to-Many*: Într-o relație unu-la-mulți, fiecare înregistrare dintr-o tabelă este asociată cu mai multe înregistrări dintr-o altă tabelă.
- *Many-to-Many*: Într-o relație mulți-la-mulți, mai multe înregistrări dintr-o tabelă sunt asociate cu mai multe înregistrări dintr-o altă tabelă. Pentru a implementa această relație, este necesară o tabelă de legătură care să asocieze înregistrările din cele două tabele.

*Foreign Keys* și *Primary Keys* sunt concepte fundamentale în proiectarea bazelor de date relaționale și sunt utilizate pentru a stabili relațiile între tabele.

*Primary Key* este o coloană sau un set de coloane într-o tabelă care identifică unic fiecare înregistrare din acea tabelă. Cheia primară este folosită pentru a asigura integritatea datelor și pentru a permite accesul rapid și eficient la înregistrările din tabelă. O cheie primară trebuie să fie unică pentru fiecare înregistrare din tabelă.

*Foreign Key* este o coloană sau un set de coloane într-o tabelă care stabilește o legătură între două tabele. Foreign Key este folosită pentru a crea relații între tabele, permițând asocierea datelor dintr-o tabelă cu înregistrările corespunzătoare dintr-o altă tabelă. O Foreign Key este de obicei legată de Primary Key a unei alte tabele, stabilind astfel relația dintre ele.

## **2.12 NuGet Package Console**

*NuGet Package Console* este o interfață de linie de comandă (CLI) integrată în Visual Studio care permite dezvoltatorilor să instaleze, să actualizeze și să gestioneze pachetele NuGet în proiectele lor. Este o unealtă utilă pentru lucrul cu dependențele și bibliotecile externe în aplicațiile lor .NET.

### 3. ELEMENTE SOFTWARE FOLOSITE

#### 3.1 Visual Studio

*Visual Studio* este un mediu integrat de dezvoltare (IDE) dezvoltat de Microsoft, utilizat pentru dezvoltarea diverselor tipuri de aplicații, de la aplicații desktop și web la aplicații mobile și jocuri. Este una dintre cele mai populare și puternice suite de dezvoltare pentru platforma .NET și alte tehnologii.

Visual Studio oferă o interfață utilizator prietenoasă, cu un set bogat de instrumente și opțiuni de personalizare, permițând dezvoltatorilor să lucreze eficient și confortabil.

Visual Studio oferă un puternic set de instrumente pentru debugging<sup>10</sup> și profilare a aplicațiilor, inclusiv suport pentru breakpoint-uri<sup>11</sup>, evaluarea expresiilor, analiza performanței și depanarea la distanță.

IDE-ul are integrată suportul pentru controlul versiunilor cu Git, permițând dezvoltatorilor să gestioneze și să colaboreze la codul lor folosind funcționalități precum controlul versiunilor, ramificările și solicitările de tragere.

Visual Studio suportă o varietate de limbaje de programare, inclusiv C#, VB.NET, F#, C++, Python, JavaScript, TypeScript și altele, oferind dezvoltatorilor o platformă unificată pentru dezvoltarea diferitelor tipuri de aplicații.

Visual Studio oferă un set complet de instrumente pentru dezvoltarea aplicațiilor web și mobile, inclusiv șabloane de proiect, instrumente de testare, emulatoare și suport pentru tehnologii precum ASP.NET, Blazor, Angular, React, Xamarin și multe altele.

#### 3.2 SQL Server Management Studio 20

*SQL Server Management Studio (SSMS)* este o aplicație de gestionare și dezvoltare a bazelor de date, dezvoltată de Microsoft, care servește ca interfață grafică pentru administrarea și manipularea bazelor de date Microsoft SQL Server. Este una dintre cele mai utilizate și puternice aplicații pentru administrarea bazelor de date SQL Server.

SSMS oferă o interfață utilizator familiară și ușor de utilizat, cu un set bogat de instrumente și opțiuni pentru administrarea bazelor de date.

---

<sup>10</sup> Debugging, în inginerie, este procesul de găsire a cauzei rădăcină și a soluțiilor și a posibilelor remedieri pentru erori.

<sup>11</sup> Breakpoint-ul în dezvoltarea de software, este un loc de oprire sau pauză intenționată într-un program, pus în aplicare în scopuri de depanare. De asemenea, uneori este denumită pur și simplu o pauză.

Utilizatorii pot crea, edita și executa interogări SQL direct în SSMS, beneficiind de facilități precum colorarea sintaxei, completarea automată a codului și sugestii de comenzi. SSMS permite gestionarea obiectelor bazei de date, cum ar fi tabelele, vizualizările, funcțiile, procedurile stocate, indecșii și altele, prin intermediul unei interfețe grafice intuitive.

SSMS oferă capacități puternice de scripting și automatizare, permițând utilizatorilor să scrie și să execute scripturi SQL complexe și să automatizeze sarcini comune de administrare a bazei de date.

### **3.3 Postman**

*Postman* este o platformă de colaborare pentru dezvoltatorii de API-uri care permite testarea, dezvoltarea și documentarea API-urilor. Este o aplicație puternică și ușor de utilizat, disponibilă ca aplicație desktop și extensie de browser, care oferă un set complet de instrumente pentru gestionarea și testarea API-urilor.

Postman oferă o interfață utilizator prietenoasă, cu un design intuitiv și o navigare simplă, permițând utilizatorilor să lucreze eficient și să acceseze rapid funcționalitățile aplicației.

Utilizatorii pot crea și trimite cereri HTTP personalizate, inclusiv cereri GET<sup>12</sup>, POST<sup>13</sup>, PUT<sup>14</sup>, DELETE<sup>15</sup> etc., folosind un editor de cereri ușor de utilizat, cu suport pentru parametri, antete, corpuri de cerere și multe altele. Postman permite crearea și gestionarea mediilor de testare, permițând dezvoltatorilor să organizeze și să execute seturi de teste pentru a valida și a verifica funcționalitatea API-urilor lor.

Postman oferă funcționalități avansate de colaborare și partajare, permițând dezvoltatorilor să lucreze împreună la dezvoltarea și testarea API-urilor și să partajeze rapid și ușor cererile și seturile de teste cu alte echipe.

---

<sup>12</sup> Metoda GET este folosită pentru a prelua date de la o adresa URL

<sup>13</sup> Metoda HTTP POST trimite date către server.

<sup>14</sup> Metoda PUT creează o nouă resursă sau înlocuiește o reprezentare a resursei țintă cu sarcina utilă de solicitare.

<sup>15</sup> Metoda HTTP DELETE șterge resursa specificată.

## 4. SPECIFICAȚII ȘI REPREZENTAREA APLICAȚIEI

### 4.1 Specificatii functionale

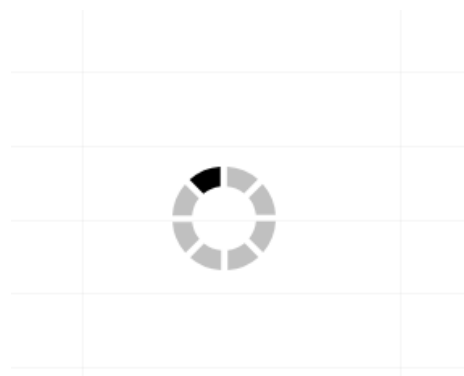
Platforma mea de blogging sau *Your blog*, sub numele în care se regăsește în proiect, este o aplicație web ce permite utilizatorilor să creeze, să publice și să vizualizeze postări ce pot conține o descriere, un titlu și o imagine. Acestea sunt grupate pe categorii, cum ar fi: fotbal, IT, music, etc. Utilizatorii sunt de 2 tipuri: useri normali și administratori. Administratorii au, în plus de utilizatorii normali, posibilitatea de a șterge blogurile și comentariile celorlalți și posibilitatea de a crea categorii de bloguri noi.

În aplicație există un singur administrator. Ceilalți useri pot fi creați prin sistemul de înregistrare al unui cont nou.

Detaliile despre utilizatori sunt: nume de utilizator (username), email, poză de profil, rol și parolă. Toate informațiile private sunt codate în baza de date.

Un user poate să adauge și comentarii la blogurile celorlalți, poate să vizualizeze profilul celorlalți, să își vizualizeze propriul profil dar și să îl editeze. Dacă utilizatorul nu își alege o imagine de profil, platforma o să îi atribuie una predefinită.

Toate comunicările dintre interfață și API sunt așteptate, timp în care utilizatorului nu îi este permis să facă vreo modificare pe pagină, prin apariția unui *ecran de încărcare*.



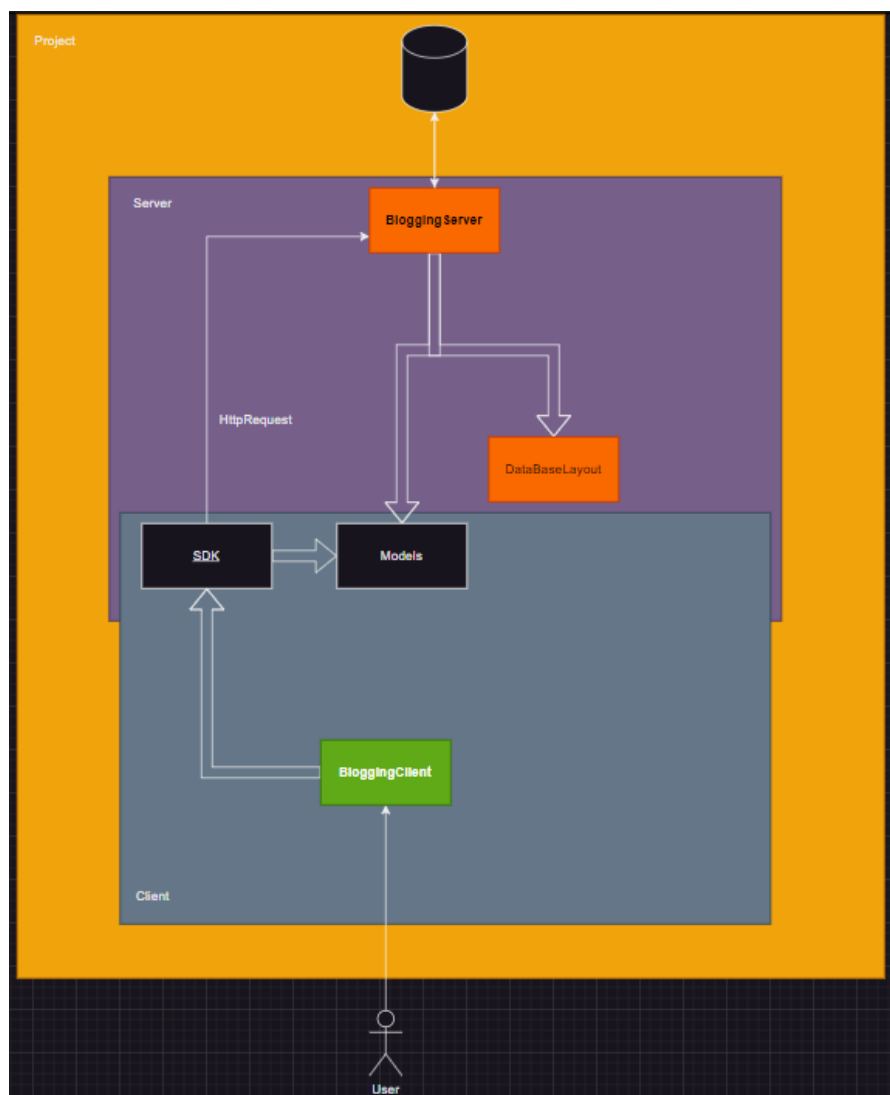
Utilizatorii sunt informați despre eventualele erori sau informări despre cererile lor către API printr-un *toast*, care apare în jocul paginii:

Current time is: 6/6/2024 3:03:02 PM

Current time is: 6/6/2024 3:03:05 PM

## 4.2 Specificații tehnice

Aplicația este împărțită în două proiecte: *BloggingServer* și *BloggingClient*.



*BloggingServer* reprezintă API-ul, proiectul prin care interfața comunică cu baza de date. Comunicarea se face prin Refit. Această soluție conține mai multe proiecte formând o structură care facilitează extinderea funcționalităților. Acesta comunică cu baza de date folosind Entity Framework. Toate setările referitoare legate de baza de date, dar și partea de token se găsesc în fișierul *appsettings.json*:

```

na: <No Schema Selected>
1  {
2  |
3  |   "Logging": {
4  |     |
5  |     |   "LogLevel": {
6  |     |     |
7  |     |     |   "Default": "Information",
8  |     |     |   "Microsoft.AspNetCore": "Warning"
9  |     |     | }
10 |     |   }
11 |   },
12 |   "ConnectionStrings": {
13 |     |
14 |     |   "DefaultConnection": "Server=localhost;Database=Blogging;Trusted_Connection=True;Encrypt=False;"
15 |     |   }
16 |   },
17 |   "AppTokenSettings": {
18 |     |
19 |     |   "Token": "token settings/config key123456789987654321 qwertyuiopasdfghjklzxcvbnm",
20 |     |   "RefreshToken": "refresh settings/config keyksadjoaosidjasdghjklzxcvbnmasdasdasdtgh",
21 |     |   "Audience": "https://localhost:7134;http://localhost:5068",
22 |     |   "Issuer": "https://localhost:7134;http://localhost:5068"
23 |     | }
24 |   }
25 | }

```

Entity Framework folosește un *DbContext* pentru a accesa toate entitățile. Restul serviciilor nu trebuie să aibă acces la toate informațiile, de aceea se folosește *Repository Pattern*<sup>16</sup>. Avem implementată o interfață generică care acoperă toate nevoile consumatorilor:

```

9  public interface IRepositoryBase<T> where T : class
10 {
11     /// <summary>
12     4 references
13     void Add(T objModel);
14     /// <summary>
15     1 reference
16     void AddRange(IEnumerable<T> objModel);
17     /// <summary>
18     1 reference
19     T Get(Expression<Func<T, bool>> predicate, Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null);
20     /// <summary>
21     5 references
22     Task<T> GetAsync(Expression<Func<T, bool>> predicate, Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null);
23     /// <summary>
24     1 reference
25     IEnumerable<T> GetList(Expression<Func<T, bool>> predicate, Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null);
26     /// <summary>
27     3 references
28     Task<IEnumerable<T>> GetListAsync(Expression<Func<T, bool>> predicate, Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null);
29     /// <summary>
30     1 reference
31     IEnumerable<T> GetAll(Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null);
32     /// <summary>
33     4 references
34     Task<IEnumerable<T>> GetAllAsync(Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null);
35     /// <summary>
36     1 reference
37     int Count();
38     /// <summary>
39     1 reference
40     Task<int> CountAsync();
41     /// <summary>
42     1 reference
43     void Update(T objModel);
44     /// <summary>
45     4 references
46     void Remove(T objModel);

```

Implementarea acestei interfețe este o clasă generică care folosește un *TEntity* ce poate fi populat cu orice model ce mapează o entitate din baza de date:

<sup>16</sup> Repository pattern este un set de reguli și practici care îmbunătățesc software-ul, oferă o abstractizare a persistenței datelor, astfel încât aplicația să poată funcționa cu o abstracție simplă (pe care o deține modelul de domeniu) care are o interfață care se aproximează pe cea a unei colecții. Adăugarea, eliminarea, actualizarea și selectarea elementelor din această colecție se face printr-o serie de metode simple, fără a fi nevoie să se ocupe de preocupările bazei de date, cum ar fi conexiunile, comenzile, cursorii sau cititorii.

```

12 public class RepositoryBase<TEntity> : IRepositoryBase<TEntity> where TEntity : class
13 {
14     protected readonly Context Context;
15
16     0 references
17     public RepositoryBase(IContext context)
18     {
19         Context = context as Context;
20     }
21     /// <inheritdoc />
22     4 references
23     public void Add(TEntity model)
24     {
25         Context.Set<TEntity>().Add(model);
26         Context.SaveChanges();
27     }
28     /// <inheritdoc />
29     1 reference
30     public void AddRange(IEnumerable<TEntity> model)
31     {
32         Context.Set<TEntity>().AddRange(model);
33         Context.SaveChanges();
34     }
35     /// <inheritdoc />
36     1 reference
37     public TEntity Get(Expression<Func<TEntity, bool>> predicate, Func<IQueryable<TEntity>, IQueryable<TEntity>> include = null)
38     {
39         IQueryable<TEntity> query = Context.Set<TEntity>();
40         if (include != null)
41         {
42             query = include(query);
43         }
44         return query.FirstOrDefault(predicate);
45     }
46     /// <inheritdoc />
47     5 references
48     public async Task<TEntity> GetAsync(Expression<Func<TEntity, bool>> predicate, Func<IQueryable<TEntity>, IQueryable<TEntity>> include = null)
49     {
50         IQueryable<TEntity> query = Context.Set<TEntity>();
51         if (include != null)
52         {
53             query = include(query);
54         }
55         return await query.FirstOrDefaultAsync(predicate);

```

Toate serviciile și repository-urile sunt înregistrate folosind *dependency injection*.

*Dependency Injection* este un concept de programare care implică furnizarea dependențelor necesare unei componente din exterior, în loc să le creeze intern. Este folosit pentru a separa responsabilitățile, crește flexibilitatea și ușurează testarea aplicațiilor software.

Codul respectă principiile *SOLID* și *design patterns*<sup>17</sup>, pentru a ușura extinderea, înțelegerea și complexitatea acestuia.

- **Single Responsibility Principle** sau Principiul Responsabilității Unice:
  - Fiecare clasă ar trebui să aibă o singură responsabilitate și, prin urmare, un singur motiv pentru a se schimba. Aceasta înseamnă că o clasă ar trebui să aibă o responsabilitate bine definită și să nu fie suprasolicitată cu multiple funcționalități.

Exemplu: În aplicația noastră de blogging, clasa Blog este responsabilă de manipularea datelor legate de postări, nu și pentru gestionarea comentariilor sau a utilizatorilor.

- **Open/Closed Principle** sau Principiul Deschiderii/Închiderii:

<sup>17</sup> Design pattern-urile reprezintă soluții generale și reutilizabile ale unei probleme comune în design-ul software. Un design pattern este o descriere a soluției sau un template ce poate fi aplicat pentru rezolvarea problemei, nu o bucată de cod ce poate fi aplicată direct. În general pattern-urile orientate pe obiect arată relațiile și interacțiunile dintre clase sau obiecte, fără a specifica însă forma finală a claselor sau a obiectelor implicate.

- Entitățile (clase, module, funcții) ar trebui să fie deschise pentru extensie, dar închise pentru modificare. Acest principiu impune dezvoltatorii să extindă funcționalitățile platformei fără a modifica codul existent.

Exemplu: Dacă dorim să adăugăm un nou tip de autentificare în aplicație, putem crea o nouă clasă care extinde funcționalitatea de autentificare fără a modifica clasele existente.

- **Liskov Substitution Principle** sau **Principiul Substituirii Liskov**:
  - Obiectele de tipul unei clase derivate ar trebui să poată înlocui obiectele de tipul clasei de bază fără a strica funcționalitatea aplicației.

Exemplu: Clasa derivată *User* ar trebui să poată fi folosită oriunde se folosește clasa de bază *IdentityUser* fără a afecta funcționalitatea.

- **Interface Segregation Principle** sau **Principiul Segregării Interfeței**:
  - O interfață ar trebui să fie specifică unui client, și clienții nu ar trebui să fie forțați să implementeze interfețe pe care nu le folosesc.

Exemplu: În loc să avem o interfață mare *IBlogOperations* care include metode pentru postări, comentarii și utilizatori, ar fi mai bine să avem interfețe mai mici și mai specifice: *IBlogService*, *ICommentService* și *IUserService*.

- **Dependency Inversion Principle** sau **Principiul Inversării Dependentei**:
  - Modulele de nivel înalt nu ar trebui să depindă de modulele de nivel scăzut. Atât modulele de nivel înalt, cât și cele de nivel scăzut ar trebui să depindă de abstracții (interfețe). Abstracțiile nu ar trebui să depindă de detalii. Detaliile ar trebui să depindă de abstracții.

Exemplu: În loc ca clasa *BlogController* să depindă direct de o clasă *BlogService*, ar trebui să depindă de o interfață *IBlogService*. Acest lucru permite schimbarea implementării *IBlogService* fără a afecta *BlogController*.

*Proiectul SDK* conține toate endpoint-urile disponibile în API (*BloggingServer*). Toate modelele de cerere sau de răspuns sunt puse în proiectul de *Models*.

*BloggingClient* injectează un *ApiClient* de fiecare dată când are nevoie să facă un request către Server:

```
27  
28 [Inject]  
29 private IBloggingApiClient BloggingApiClient { get; set; }
```

Pentru a pune în așteptare pagina, până se execută cererea către API, dar și pentru a notifica rezultatul în urma finalizării request-ului, se folosesc *state*-uri. Clase injectate *singleton*, care folosesc un *Action* pentru a notifica toate componentele care sunt abonate la acestea:



```

6
7 public class SnackbarState
8 {
9     public SnackbarStack Snackbar { get; set; }
10
11     public event Action OnStateChange;
12
13     public async Task PushAsync(string message, bool isError = false)
14     {
15         await Snackbar.PushAsync(
16             message,
17             isError ? SnackbarColor.Danger : SnackbarColor.Info,
18             options: options => { options.IntervalBeforeClose = 3000; }); // Task
19
20         NotifyStateChanged();
21     }
22
23     private void NotifyStateChanged() => OnStateChange?.Invoke();
24 }

```

În momentul în care are loc o acțiune, componentele folosesc metoda *StateHasChanged* pentru a notifica contextul despre schimbări. Folosim interfața *IDisposable*, care apelează ca un destructor metoda *Dispose*, pentru a dezabona componentele de la state-uri și pentru a evita *memory leaks*.

```

44 public void Dispose()
45 {
46     SnackbarState.OnStateChange -= StateHasChanged;
47     LoadingState.OnStateChange -= StateHasChanged;
48 }
49
50 protected override async Task OnInitializedAsync()
51 {
52     SnackbarState.OnStateChange += StateHasChanged;
53     LoadingState.OnStateChange += StateHasChanged;
54
55     await GetBlogCategoriesAsync();
56 }

```

Proiectul începe de componenta *\_Host.cshtml* care folosește ca și layout: *\_Layout.cshtml*. Aici se injectează toate referințele de la toate librăriile și se randează un *body*. *Body*, care este reprezentat de un routing pus la dispoziție de către Blazor, prin care trec toate paginile:

```

1 @inherits ComponentBase
2 @implements IDisposable
3
4 <SnackbarStack @ref="@SnackbarState.Snackbar"></SnackbarStack>
5
6 <Router AppAssembly="@typeof(App).Assembly">
7     <Found Context="routeData">
8         <AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
9         <FocusOnNavigate RouteData="@routeData" Selector="h1" />
10     </Found>
11     <NotFound>
12         <PageTitle>Not found</PageTitle>
13         <LayoutView Layout="@typeof(MainLayout)">
14             <p role="alert">Sorry, there's nothing at this address.</p>
15         </LayoutView>
16     </NotFound>
17 </Router>
18

```

Această componentă apelează un *MainLayout* predefinit, ce centrează conținutul folosind clasa din bootstrap *container*.

```

1  @inherits LayoutComponentBase
2
3  <PageTitle>Blogging</PageTitle>
4
5  <LoadingIndicator @ref="LoadingState.Loading">
6      <ChildContent>
7          <div class="bg-black text-light">
8              <div class="container">
9                  <header>
10                     <NavMenu></NavMenu>
11                 </header>
12                 <div class="background-grey background-height ">
13
14                     @Body
15                 </div>
16                 <footer>
17                     <Footer></Footer>
18                 </footer>
19             </div>
20         </div>
21     </ChildContent>
22     <IndicatorTemplate>
23         <Animate Animation="Animations.ZoomIn" Auto Duration="TimeSpan.FromMilliseconds(700)">
24             <Div>
25                 <SpinKit Type="SpinKitType.Circle" Size="100px" />
26             </Div>
27         </Animate>
28     </IndicatorTemplate>
29 </LoadingIndicator>

```

Pe toate paginile există o bară de navigare și un footer, unde putem naviga către pagina de scriere a unui blog, de a vizualiza toate blogurile, profilul utilizatorului, dar și de a te loga și a te înregistra:

```

1  <nav class="navbar pt-3">
2      <div class="container d-flex flex-row flex-align-between">
3          <Link class="navbar-brand text-light fw-bold logo" To="/">Your Blog</Link>
4          <div class="navbar-nav fs-3 fw-bold d-flex flex-row gap-3">
5              <AuthorizeView>
6                  <Authorized>
7                      <li class="nav-item">
8                          <Link class="nav-link text-light" To="/write">Write</Link>
9                      </li>
10                     <li class="nav-item">
11                         <Link class="nav-link text-light" To="/search">Search</Link>
12                     </li>
13                 </Authorized>
14             </AuthorizeView>
15             <li class="nav-item">
16                 <Link class="nav-link text-light" To="/about">About</Link>
17             </li>
18             <li class="nav-item">
19                 <Link class="nav-link text-light" To="/faq">FAQ</Link>
20             </li>
21         </div>
22         <ul class="navbar-nav d-flex flex-row gap-3">
23             <AuthorizeView>
24                 <Authorized>
25                     <li class="nav-item">
26                         <Button To="@($account / @authState.User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.Name).Value)" Type="ButtonType.Link" Color="Color.Light"
27                     </li>
28                 </Authorized>
29                 <NotAuthorized>
30                     <li class="nav-item">
31                         <Button To="/login" Type="ButtonType.Link" Color="Color.Light" TextWeight="TextWeight.Bold" TextColor="TextColor.Dark" TextSize="TextSize.Heading3" Class="
32                     </li>
33                     <li class="nav-item">
34                         <Button To="/register" Type="ButtonType.Link" Color="Color.Light" TextWeight="TextWeight.Bold" TextColor="TextColor.Dark" TextSize="TextSize.Heading3" Class="
35                     </li>
36                 </NotAuthorized>
37             </AuthorizeView>
38         </ul>
39     </div>
40 </nav>

```

Din footer putem accesa și termenii și condițiile aplicației.

```

1 <Div Flex="Flex.Row.JustifyContent.Between.AlignItems.Center" Height="@Height.Px(size:107)">
2   <Div Flex="Flex.Column.JustifyContent.Center" Gap="Gap.Is3">
3     <Div Flex="Flex.Row" Gap="Gap.Is5">
4       <a href="/privacy-policy" class="text-light">Privacy Policy</a>
5       <a href="/terms-and-conditions" class="text-light">Terms & Conditions</a>
6     </Div>
7     <Paragraph>© 2024 Your Blog Inc. All Rights Reserved.</Paragraph>
8   </Div>
9   <Div Flex="Flex.Row.JustifyContent.Between.AlignItems.Center" Gap="Gap.Is4">
10    <span class="fs-4">Share our platform with the world!</span>
11    <Div Flex="Flex.Row" Gap="Gap.Is2">
12      <i class="bi bi-linkedin fs-4"></i>
13      <i class="bi bi-facebook fs-4"></i>
14      <i class="bi bi-instagram fs-4"></i>
15    </Div>
16  </Div>
17 </Div>

```

### 4.3 Diagramele cazurilor de utilizare

O diagramă a cazurilor de utilizare (use case diagram) prezintă o colecție de cazuri de utilizare și actori care:

- oferă o descriere generală a modului în care va fi utilizat sistemul
- furnizează o privire de ansamblu a funcționalităților ce se doresc a fi oferite de sistem
- arată cum interacționează sistemul cu unul sau mai mulți actori
- asigură faptul că sistemul va produce ceea ce s-a dorit.



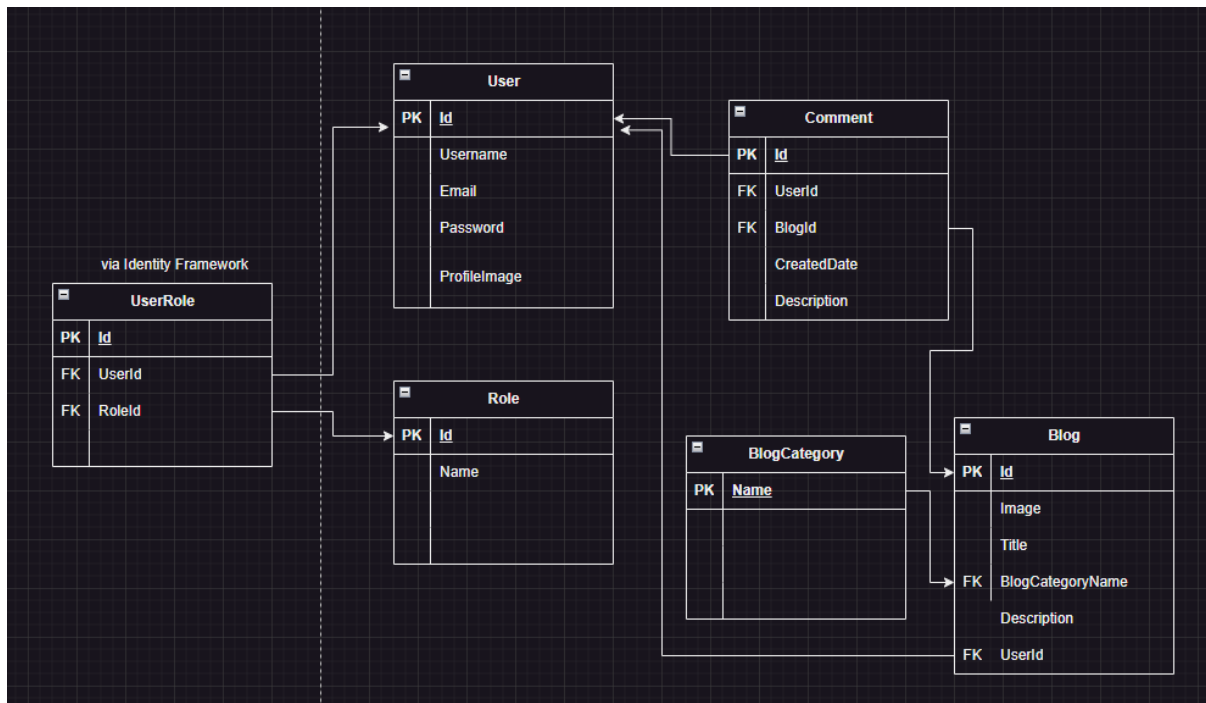
Un actor este un stereotip al unei clase. Actorii sunt reprezentați de utilizatori sau entități care pot interacționa cu sistemul. Ei nu fac parte din sistem și definesc mulțimi de roluri în comunicarea cu acesta.

Un actor se reprezintă sub formă unui *omuleț* sub care se trece numele acestuia.



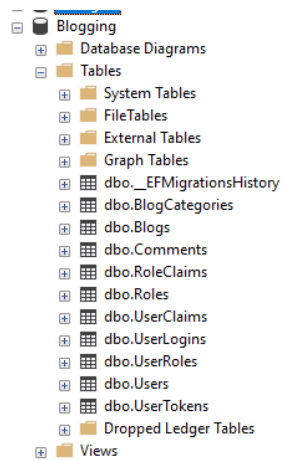
#### 4.4 Organizarea bazei de date

*Diagrama ERD* este o diagramă care prezintă structura unei baze de date în termeni de entități și relațiile dintre ele. Entitățile sunt obiectele sau conceptele distincte despre care se stochează date în baza de date, iar relațiile definesc modul în care aceste entități sunt conectate sau interacționează între ele. Diagrama ERD este utilă pentru proiectarea și modelarea bazei de date, oferind o vedere vizuală asupra entităților și relațiilor lor.



Cu această structură orice entitate poate avea acces la referința ei: un blog își poate vedea creatorul, comentariile, rolul creatorului, categoria, dar și celelalte componente pot face același lucru.

Baza de date o să arate astfel:



## 5. DEZVOLTAREA APLICAȚIEI

### 5.1 Popularea bazei de date

Popularea bazei de date este esențială în configurarea aplicației. În cazul de față folosim SQL Server împreună cu Entity Framework și Identity pentru a gestiona autentificarea și operațiile pe date.

*Entity Framework (EF)* este un ORM (Object-Relational Mapper) dezvoltat de Microsoft care permite dezvoltatorilor să lucreze cu o bază de date folosind obiecte .NET. Una dintre cele trei abordări principale ale EF este Code First, care permite dezvoltatorilor să definească modelul de date folosind clase C# obișnuite (POCO - Plain Old CLR Objects) și apoi să genereze schema bazei de date pe baza acestor clase.

O entitate în Entity Framework (EF) reprezintă o clasă C# care este mapată la o tabelă din baza de date. Fiecare instanță a acestei clase corespunde unei rând din tabelă.

Code First permite dezvoltatorilor să creeze modelul de date prin scrierea de cod C#. Aceasta înseamnă că nu este necesar să existe o bază de date preexistentă, deoarece EF poate crea baza de date și tabelele pe baza modelului definit în cod. Acest lucru oferă flexibilitate și control complet asupra designului modelului de date.

Folosind *Entity Framework*, avem definit contextul de date, prin clasa *Context*, care extinde *IdentityDbContext* pentru gestionarea utilizatorilor, folosind *ASP.NET Core Identity*. Contextul de date definește seturile de entități care vor fi folosite în baza de date:

```
7 namespace DataBaseLayout.DbContext;
8
9 references
9 public class Context : IdentityDbContext<User, Role, string>, IContext
10 {
11     1 reference
12     public DbSet<Blog> Blogs { get; set; }
13
14     1 reference
15     public DbSet<BlogCategory> BlogCategories { get; set; }
16
17     1 reference
18     public DbSet<Comment> Comments { get; set; }
19
20     0 references
21     public Context(DbContextOptions<Context> options)
22         : base(options) { }
23
24     1 reference
25     public async Task<int> SaveChangesAsync()
26     {
27         return await base.SaveChangesAsync();
28     }
29 }
```

Pentru a genera baza de date, mai întâi trebuie generate migrațiile. Migrațiile în Entity Framework (EF) sunt un mecanism prin care modificările aduse modelului de date în cod sunt reflectate în schema bazei de date. Acestea permit gestionarea și aplicarea modificărilor astfel încât baza de date este sincronizată cu modelul de date definit în cod. Pentru acest lucru se vor executa în Package Manager Console următoarele comenzi:

```
Package Manager Console
Package source: All | Default project: BloggingServer
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it
dependencies.

Package Manager Console Host Version 6.9.1.3

Type 'get-help NuGet' to see all available NuGet commands.

PM> add-migration Initial
```

Iar pentru a rula aceste migrații se va executa:

```
Package Manager Console
Package source: All | Default project: BloggingServer
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it
dependencies.

Package Manager Console Host Version 6.9.1.3

Type 'get-help NuGet' to see all available NuGet commands.

PM> update-database
```

Toate entitățile din aplicație sunt populate din partea de Client a proiectului, de către toți utilizatorii care și-au creat un cont. În aplicație există 2 roluri: *User* și *Admin*. Acestea sunt adăugate automat de fiecare dată când server-ul rulează. Cu rolul *Admin* există un singur cont, care, de asemenea, este generat la fiecare runtime:

```
57
58 var app: WebApplication = builder.Build();
59
60 if (app.Environment.IsDevelopment())
61 {
62     app.UseSwagger();
63     app.UseSwaggerUI();
64 }
65
66 app.UseHttpsRedirection();
67
68 app.UseAuthorization();
69
70 app.MapControllers();
71
72 await DefaultDataAsync();
73
74 app.Run();
75
76 return;
```

```
1 reference
78 async Task DefaultDataAsync()
79 {
80     var serviceProvider = builder.Services.BuildServiceProvider();
81     var roleManager = serviceProvider.GetService<RoleManager<Role>>();
82     var userManager = serviceProvider.GetService<UserManager<User>>();
83
84     var userRole = await roleManager.Roles.FirstOrDefaultAsync(x:Role => x.Id == Roles.User);
85     if (userRole == null)
86     {
87         var result = await roleManager.CreateAsync(
88             new Role()
89             {
90                 Id = Roles.User,
91                 Name = Roles.User
92             }); // Task<IdentityResult>
93         if (!result.Succeeded)
94         {
95             throw new Exception(result.Errors.First().Description);
96         }
97     }
98
99     var adminRole = await roleManager.Roles.FirstOrDefaultAsync(x:Role => x.Id == Roles.Admin);
100    if (adminRole == null)
101    {
102        var result = await roleManager.CreateAsync(
103            new Role()
104            {
105                Id = Roles.Admin,
106                Name = Roles.Admin
107            }); // Task<IdentityResult>
108        if (!result.Succeeded)
109        {
110            throw new Exception(result.Errors.First().Description);
111        }
112    }
113 }
```



```

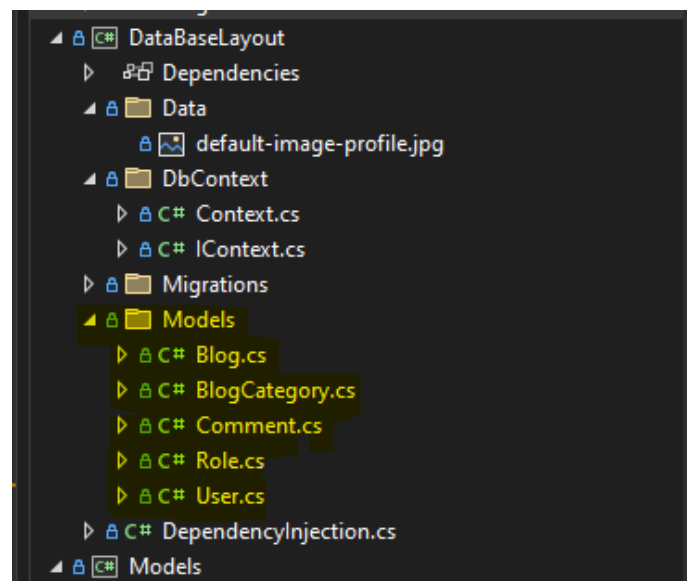
113
114 var adminUser: IList<User> = await userManager.GetUsersInRoleAsync(Roles.Admin);
115 if (!adminUser.Any())
116 {
117     var profileImage: byte[] = await File.ReadAllBytesAsync(path: @".\DataBaseLayout\Data\default-image-profile.jpg");
118     var user = new User
119     {
120         Id = "admin",
121         UserName = "admin",
122         Email = "admin@admin.ro",
123         EmailConfirmed = true,
124         PhoneNumber = "0111111111",
125         PhoneNumberConfirmed = true,
126         TwoFactorEnabled = false,
127         ProfileImage = profileImage,
128         JoinedDate = DateTime.UtcNow,
129         AcceptTerms = true,
130     };
131     var result = await userManager.CreateAsync(user, password: "Admin1234!");
132
133     if (!result.Succeeded)
134     {
135         throw new Exception(result.Errors.First().Description);
136     }
137
138     result = await userManager.AddToRolesAsync(user, roles: new List<string>() { Roles.User, Roles.Admin });
139
140     if (!result.Succeeded)
141     {
142         throw new Exception(result.Errors.First().Description);
143     }
144 }
145
146

```

Se adaugă rolurile: User și Admin, în bază, dacă nu există, la fel și user-ul cu rolul *Admin*.

## 5.2 Modele de baze de date Entity Framework

Modelele de date în Entity Framework (EF) sunt reprezentări ale entităților din aplicație, utilizate pentru a interacționa cu baza de date. Aceste modele sunt adesea clase C# care definesc structura și relațiile datelor pe care le gestionăm. Prin intermediul acestor modele, EF poate traduce operațiunile efectuate asupra obiectelor în comenzi SQL care, mai apoi sunt trimise către baza de date și interpretate. Aplicația noastră dispune de cinci astfel de modele:



```

1  using Microsoft.EntityFrameworkCore;
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel.DataAnnotations;
5
6  namespace DataBaseLayout.Models;
7
8  [PrimaryKey(nameof(Id))]
9
10 14 references
11  public class Blog
12  {
13      7 references
14      public Guid Id { get; set; }
15
16      [Required]
17      4 references
18      public string Title { get; set; }
19
20      5 references
21      public string BlogCategoryName { get; set; }
22
23      [Required]
24      1 reference
25      public BlogCategory BlogCategory { get; set; }
26
27      [Required]
28      4 references
29      public byte[] Image { get; set; }
30
31      4 references
32      public string Description { get; set; }
33
34      4 references
35      public DateTime CreatedTime { get; set; }
36
37      3 references
38      public string UserId { get; set; }
39
40      [Required]
41      5 references
42      public User User { get; set; }
43
44      2 references
45      public ICollection<Comment> Comments { get; set; }
46  }

```

Un blog conține id-ul, titlul, o categorie, o imagine, o descriere, data creării, utilizatorul care a inițiat crearea blogului dar și mai multe comentarii.

```

4  namespace DataBaseLayout.Models;
5
6  [PrimaryKey(nameof(Name))]
7
8  public class BlogCategory
9  {
10      4 references
11      public string Name { get; set; }
12
13      2 references
14      public ICollection<Blog> Blogs { get; set; }
15  }

```

Categoria unui blog conține doar numele și blogurile care aparțin acelei categorii. Acestea sunt unice prin nume.

```

1  using System;
2  using System.ComponentModel.DataAnnotations;
3  using Microsoft.EntityFrameworkCore;
4
5  namespace DataBaseLayout.Models;
6
7  [PrimaryKey(nameof(Id))]
8
9  public class Comment
10 {
11
12     [Required]
13     public string Description { get; set; }
14
15     [Required]
16     public DateTime CreatedDate { get; set; }
17
18     public string UserId { get; set; }
19
20     [Required]
21     public User User { get; set; }
22
23     public Guid BlogId { get; set; }
24
25     [Required]
26     1 reference
27     public Blog Blog { get; set; }
28 }

```

Un comentariu conține un Id, o descriere, data creării, utilizatorul care a creat comentariul și blog-ul la care a fost atribuit.

```

1  using Microsoft.AspNetCore.Identity;
2
3  namespace DataBaseLayout.Models;
4
5  6 references
6  public class Role : IdentityRole
7  {
8
9  }

```

*Role* este un *IdentityRole*, ce conține numele rolului. Acesta are referințe de many-to-many cu *User*, făcute de către Identity. Această referință este suprascrisă în *Context*, pentru a păstra convențiile de nume din proiect:

```

25 protected override void OnModelCreating(ModelBuilder modelBuilder)
26 {
27     base.OnModelCreating(modelBuilder);
28
29     modelBuilder.Entity<User>(entity =>
30     {
31         entity.ToTable(name: "Users");
32     });
33     modelBuilder.Entity<Role>(entity =>
34     {
35         entity.ToTable(name: "Roles");
36     });
37     modelBuilder.Entity<IdentityUserClaim<string>>(entity =>
38     {
39         entity.ToTable(name: "UserClaims");
40     });
41     modelBuilder.Entity<IdentityUserRole<string>>(entity =>
42     {
43         entity.ToTable(name: "UserRoles");
44     });
45     modelBuilder.Entity<IdentityRoleClaim<string>>(entity =>
46     {
47         entity.ToTable(name: "RoleClaims");
48     });
49     modelBuilder.Entity<IdentityUserToken<string>>(entity =>
50     {
51         entity.ToTable(name: "UserTokens");
52     });
53     modelBuilder.Entity<IdentityUserLogin<string>>(entity =>
54     {
55         entity.ToTable(name: "UserLogins");
56     });

```

```

1  using System;
2  using System.Collections.Generic;
3  using Microsoft.AspNetCore.Identity;
4
5  namespace DataBaseLayout.Models;
6
7  public class User : IdentityUser
8  {
9      public byte[] ProfileImage { get; set; }
10
11     public bool AcceptTerms { get; set; }
12
13     public DateTime JoinedDate { get; set; }
14
15     public ICollection<Comment> Comments { get; set; }
16
17     public ICollection<Blog> Blogs { get; set; }
18 }

```

*User* definește utilizatorul și conține în plus, pe lângă proprietățile standard oferite de *IdentityUser* (username, email, etc.), imaginea de profil, dacă a acceptat termeni și condițiile, data la care s-a înregistrat în platformă și relația de one-to-many dintre acesta și *Comments*, plus *Blogs*. Astfel, un utilizator poate scrie mai multe comentarii și bloguri, însă un comentariu, respective un blog poate fi scris de mai multi utilizatori.

În caz de ștergere sau actualizare, *Context*-ul este configurat să nu facă operația în cascadă pentru că toate entitățile noastre pot fi create și pot exista independent.

```

57
58     modelBuilder.Entity<Blog>().HasOne(navigationExpression: x:Blog => x.User) // ReferenceNavigationBuilder<Blog,User>
59     .WithMany(navigationExpression: x:User => x.Blogs)
60     .OnDelete(DeleteBehavior.NoAction)
61     .HasForeignKey(x:Blog => x.UserId);
62
63     modelBuilder.Entity<Blog>().HasOne(navigationExpression: x:Blog => x.BlogCategory) // ReferenceNavigationBuilder<Blog,BlogCategory>
64     .WithMany(navigationExpression: x:BlogCategory => x.Blogs)
65     .OnDelete(DeleteBehavior.NoAction)
66     .HasForeignKey(x:Blog => x.BlogCategoryId);
67
68     modelBuilder.Entity<Comment>().HasOne(navigationExpression: x:Comment => x.Blog) // ReferenceNavigationBuilder<Comment,Blog>
69     .WithMany(navigationExpression: x:Blog => x.Comments)
70     .OnDelete(DeleteBehavior.NoAction)
71     .HasForeignKey(x:Comment => x.BlogId);
72
73     modelBuilder.Entity<Comment>().HasOne(navigationExpression: x:Comment => x.User) // ReferenceNavigationBuilder<Comment,User>
74     .WithMany(navigationExpression: x:User => x.Comments)
75     .OnDelete(DeleteBehavior.NoAction)
76     .HasForeignKey(x:Comment => x.UserId);

```

Prin aceste modele se evidențiază tipul pe care fiecare proprietate ar trebui să îl aibă, valorile pe care le pot avea, dar și relațiile dintre entități: one-to-one, one-to-many, many-to-many. Astfel se facilitează mult mai ușor accesarea acestora în cod.

Pentru ca aceste referințe să fie accesate se folosește noțiunea de *AutoInclude*. *AutoInclude* este o caracteristică introdusă în Entity Framework Core 6 care permite încorporarea automată a relațiilor la interogările LINQ fără a fi nevoie de includerea explicită a acestora în cod.

```

77
78     modelBuilder.Entity<Blog>().Navigation(t:Blog => t.Comments).AutoInclude();
79
80     modelBuilder.Entity<Blog>().Navigation(t:Blog => t.User).AutoInclude();
81
82     modelBuilder.Entity<User>().Navigation(t:User => t.Blogs).AutoInclude();
83
84     modelBuilder.Entity<BlogCategory>().Navigation(t:BlogCategory => t.Blogs).AutoInclude();
85

```

### 5.3 Sistemul de autentificare

Sistemul de autentificare este partea esențială a aplicației deoarece verifică identitatea utilizatorilor și le acordă acces la diferite resurse ale aplicației în funcție de permisiunile lor. Acesta asigură securitatea și confidențialitatea datelor prin autentificarea utilizatorilor și gestionarea sesiunilor.

Atât autentificarea cât și autorizarea se realizează prin *Bearer Token*.

*Autentificarea Bearer* (numită și autentificare cu token) este o schemă de autentificare HTTP care implică jetoane de securitate numite Bearer Token. Tokenul este un șir criptic, generat de obicei de server ca răspuns la o solicitare de conectare. Clientul trebuie să trimită acest token în header-ul *Authorization* atunci când face cereri către API.

Pentru generarea acestui token, se folosește *JwtSecurityToken*:

```

31  /// <inheritdoc />
32  3 references
33  public async Task<string> GenerateTokenAsync(string username, int durationMin)
34  {
35      var user = _userManager.Users.FirstOrDefault(u => u.UserName == username);
36      var roles = await _userManager.GetRolesAsync(user);
37      var claims = new List<Claim>()
38      {
39          new Claim(ClaimTypes.Name, username),
40          new Claim(ClaimTypes.Email, user.Email),
41      };
42      claims.AddRange(collection: roles.Select(role => new Claim(ClaimTypes.Role, role)));
43
44      var key = new SymmetricSecurityKey(
45          Encoding.UTF8.GetBytes(_configuration.GetSection(key: "AppTokenSettings:Token").Value!));
46      var credential = new SigningCredentials(key, algorithm: SecurityAlgorithms.HmacSha512Signature);
47
48      var token = new JwtSecurityToken(claims: claims, expires: DateTime.UtcNow.AddMinutes(durationMin),
49          signingCredentials: credential);
50      return new JwtSecurityTokenHandler().WriteToken(token);
51  }

```

Acesta este apelat de către controller, în momentul în care utilizatorul inițiază operația de *Login*:

```

30  /// <inheritdoc />
31  2 references
32  public async Task<LoginResponse> SignInAsync(string userName, string password)
33  {
34      var user = await _userManager.FindByNameAsync(userName);
35
36      var isLoggedIn = await _signInManager.CheckPasswordSignInAsync(user, password, lockoutOnFailure: false);
37
38      if (isLoggedIn.Succeeded)
39      {
40          var token = await _tokenService.GenerateTokenAsync(userName, durationMin: 2);
41          var refreshToken = await _tokenService.GenerateTokenAsync(userName, durationMin: 8);
42
43          var responseLogin = new LoginResponse
44          {
45              AccessToken = token,
46              RefreshToken = refreshToken
47          };
48
49          return responseLogin;
50      }
51
52      throw new Exception(message: "Email or password is incorrect!");
53  }

```

Toate rolurile utilizatorului se criptează ca și Claims<sup>18</sup> în token. Toți utilizatorii care vor să facă diferite operații sau să acceseze resurse din platformă trebuie să ofere un token pentru a le verifica identitatea. În *backend* acest lucru se face printr-un atribut definit la fiecare request:

<sup>18</sup> Claims sunt informații afirmate despre un subiect. De exemplu, un simbol ID (care este întotdeauna un JWT) poate conține o revendicare numită *nume* care afirmă numele utilizatorului ce se autentifică

```

8 | 2 references
9 | public class BloggingAuthorizationHandler : AuthorizationHandler<AuthorizationRequirement>
10 | {
11 |     private readonly IHttpContextAccessor _httpContextAccessor;
12 |     private readonly ITokenService _tokenService;
13 |
14 |     0 references
15 |     public BloggingAuthorizationHandler(IHttpContextAccessor httpContextAccessor, ITokenService tokenService)
16 |     {
17 |         _httpContextAccessor = httpContextAccessor;
18 |         _tokenService = tokenService;
19 |     }
20 |
21 |     0 references
22 |     protected override Task HandleRequirementAsync
23 |     (AuthorizationHandlerContext context, AuthorizationRequirement requirement)
24 |     {
25 |         var httpRequest = _httpContextAccessor.HttpContext!.Request;
26 |         var token :string = httpRequest.Headers["Authorization"].ToString().Replace(oldValue: "Bearer ", newValue: string.Empty);
27 |
28 |         if (!_tokenService.IsValidToken(token, requirement.RoleName))
29 |         {
30 |             context.Fail();
31 |             return Task.CompletedTask;
32 |         }
33 |
34 |         context.Succeed(requirement);
35 |         return Task.CompletedTask;
36 |     }
37 | }

```

Acesta are o perioadă de expirare. Dacă este expirat accesul este restricționat. Utilizatorul își poate folosi un al doilea token, numit și *refreshToken* pentru a regenera un alt token, fără a fi nevoie să repete pașii de login. Dacă și acest refreshToken este expirat atunci sesiunea se încheie și utilizatorul este nevoi să se logheze din nou. Toate acestea se fac prin intermediul Refit-ului, care în spate pune la dispoziție un *HttpClient*, prin intermediul căruia *frontend*-ul apelează controller-ele din API.

În client-side, token-ul este salvat în *Local Storage*. Local Storage este o tehnologie de stocare web care permite aplicațiilor web să stocheze date local, direct în browser-ul utilizatorului. Aceasta este parte a specificației Web Storage și oferă o modalitate simplă și eficientă de a păstra datele pe partea clientului fără a fi nevoie de servere sau baze de date externe.

Tot acest flow este susținut de un *AuthenticationStateProvider*. Acesta este folosit ca un *CascadeParameter*, prin care notifică toate componentele că *state*-ul s-a schimbat. Blazor pune la dispoziție un tag care verifică dacă user-ul este autentificat și autorizat:

```

23 | <AuthorizeView>
24 | <Authorized>
25 |     <li class="nav-item">
26 |         <button type="button" class="btn btn-link" href="/account/{@authState.User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.Name).Value}" Type="ButtonType.Link" Color="Color.Light" TextWeight="TextWeight.Normal" TextColor="TextColor.Dark" TextSize="TextSize.Default" Class="rounded-4">Log Out
27 |     </li>
28 | </Authorized>
29 | <NotAuthorized>
30 |     <li class="nav-item">
31 |         <button type="button" class="btn btn-link" href="/login" Type="ButtonType.Link" Color="Color.Light" TextWeight="TextWeight.Bold" TextColor="TextColor.Dark" TextSize="TextSize.Default" Class="rounded-4">Log In
32 |     </li>
33 |     <li class="nav-item">
34 |         <button type="button" class="btn btn-link" href="/register" Type="ButtonType.Link" Color="Color.Light" TextWeight="TextWeight.Bold" TextColor="TextColor.Dark" TextSize="TextSize.Default" Class="rounded-4">Register
35 |     </li>
36 | </NotAuthorized>
37 | </AuthorizeView>

```

Refit-ul este configurat ca la începutul fiecărui request să acceseze acest storage și să atașeze în header-ul *Authorization* token-ul de acces:



```

RefitSettings refitSettings = new()
{
    AuthorizationHeaderValueGetter = (_, cancellationToken) => AuthBearerTokenFactory.GetBearerTokenAsync(cancellationToken)
};
services.AddRefitClient<IBloggingApi>(refitSettings)
    .ConfigureHttpClient(c => c.BaseAddress = url);

services.AddSingleton<IBloggingApiClient, BloggingApiClient>();

```

Utilizatorul poate face o cerere de înregistrare din interfață. Toate informațiile private sunt codificate folosind inversarea caracterelor, iar parola folosește un HASH prestabilit:

```

5
6 public class PersonalDataProtector : IPersonalDataProtector
7 {
8     public string Protect(string data)
9     {
10         return new string(data?.Reverse().ToArray());
11     }
12
13     public string Unprotect(string data)
14     {
15         return new string(data?.Reverse().ToArray());
16     }
17 }

```

În baza de date, aceste valori o să arate de forma:

Results		Messages									
	Id	ProfileImage	AcceptTerms	JoinedDate	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash	Security
1	admin	0xFFD0FFE000104A46494600010100025802580000FFD800...	1	2024-06-02 19:55:20.9895163	nimda	NIMDA	or.nimda@nimda	OR.NIMDA@NIMDA	1	AQAAAAIAAYagAAAAEMddAhoab81qDh5j7Zhb4WtEaaNG+XOo...	DM2E1

## 5.4 Sistemul de blogging

Sistemul de blogging oferă utilizatorilor să vizualizeze, dar și să creeze blog-uri noi. Pentru toate acestea, *user*-ul trebuie să fie autentificat dar și autorizat cu rolul de User sau Admin:



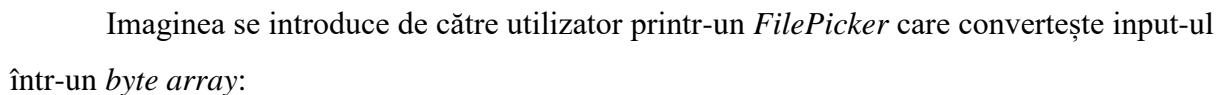
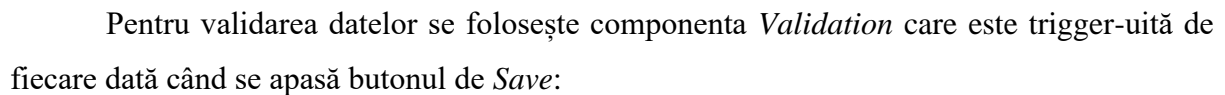
```

23
24 [HttpGet]
25 [Authorize(policy: Roles.User)]
26
27 public async Task<IActionResult> GetBlogsAsync()
28 {
29     try
30     {
31         var result :List<Blog> = await _blogService.GetBlogsAsync();
32         return ApiServiceResponse.ApiServiceResult(new ServiceResponse<List<Blog>>(result.ToList()));
33     }
34     catch (Exception ex)
35     {
36         return ApiServiceResponse.ApiServiceResult(new ServiceResponse<List<Blog>>(ex));
37     }
38 }
39
40 [HttpGet(template: "username/{username}")]
41
42 [Authorize(policy: Roles.User)]
43
44 public async Task<IActionResult> GetBlogsByUserAsync(string username)
45 {
46     try
47     {
48         var result :List<Blog> = await _blogService.GetBlogsByUserAsync(username);
49         return ApiServiceResponse.ApiServiceResult(new ServiceResponse<List<Blog>>(result.ToList()));
50     }
51     catch (Exception ex)
52     {
53         return ApiServiceResponse.ApiServiceResult(new ServiceResponse<List<Blog>>(ex));
54     }
55 }
56
57 [HttpGet(template: "{id}")]
58 [Authorize(policy: Roles.User)]
59
60 public async Task<IActionResult> GetBlogAsync(string id)
61 {
62     try
63     {
64         var result :Blog = await _blogService.GetBlogAsync(id);
65         return ApiServiceResponse.ApiServiceResult(new ServiceResponse<Blog>(result));
66     }
67     catch (Exception ex)
68     {
69         return ApiServiceResponse.ApiServiceResult(new ServiceResponse<Blog>(ex));
70     }
71 }

```

Admin-ul, în acest proces, are dreptul de a șterge și blog-urile celorlalți Useri.

Pentru scrierea unui blog, utilizatorul trebuie să ofere un titlu, o imagine, o descriere și o categorie. Categoriile pot fi adăugate doar de către admini și sunt unice prin nume. Aceste intrări sunt create folosind componentele de care dispune librăria Blazorise:



```

75 private async Task OnImageUploaded(FileUploadEventArgs e)
76 {
77     try
78     {
79         using var result = new MemoryStream();
80         await e.File.OpenReadStream(maxAllowedSize: long.MaxValue).CopyToAsync(result);
81
82         _addBlogModel.Image = result.ToArray();
83     }
84     catch (Exception exc)
85     {
86         Console.WriteLine(exc.Message);
87     }
88     finally
89     {
90         StateHasChanged();
91     }
92 }
93

```

Metoda *StateHasChanged()* notifică toate componentele că s-au efectuat modificări pe contextul curent.

Toate blog-urile se găsesc sub pagina */search/{parameters}* care, parcurge toate blog-urile înregistrate în platformă, și le afișează într-un format prietenos pentru utilizator:

```

UserSettings.razor.cs Write.razor Account.razor Write.razor.cs Register.razor.cs Search.razor* Blo
BloggingClient
1 @page "/search"
2 @inherits ComponentBase
3 @implements IDisposable
4
5 @attribute [Authorize(Roles = Roles.User)]
6
7 <PageTitle>Blogs</PageTitle>
8
9
10 <div class="d-flex flex-row flex-nowrap justify-content-start w-100 bg-white">
11     @foreach (var blogCategory : FilterCheck in _blogCategories)
12     {
13         <div class="filter-cell">
14             <span class="fs-4">@blogCategory.Name</span>
15             <Check class="check-box" TValue="bool" Checked="@blogCategory.IsChecked"></Check>
16         </div>
17     }
18 </div>
19
20 <div class="d-flex flex-row p-5 flex-wrap w-100">
21     @if (!_blogs.Any())
22     {
23         <div class="d-flex flex-column gap-1">
24             <div class="fs-1 fw-bold align-items-center">Blogs not found!</div>
25             <Anchor class="fs-3 fw-bold text-dark" To="/write">Write something!</Anchor>
26         </div>
27     }
28     @foreach (var blog in _blogs)
29     {
30         <div class="blog-card">
31             <BlogCard Item="@blog" ItemClicked="@BlogClicked"></BlogCard>
32         </div>
33     }
34 </div>
35

```

Pentru a pastra principiul Single Responsibility, s-a creat o componentă separată pentru un card blog:

```

1 @inherits ComponentBase
2
3 <Div Flex="Flex.Column" Width="Width.Px(size: 300)" Position="Position.Relative" Gap="Gap.Is3">
4   <Image Width="@Width.Px(size: 300)" Height="@Height.Px(size: 200)" Source="@($data:image/png;base64,{Item.Image})" class="rounded-5"/>
5   <div class="text-wrap">
6     <Span class="fs-3 fw-bold text-wrap me-2">@Item.Title</Span><Span class="fs-5">by <u>@Item.UserName</u></Span>
7   </div>
8
9   <Link Clicked="@CellClick" Stretched class="cursor-pointer">
10
11 </Link>
12 </Div>

```

Această componentă conține un *Link* care este *Stretched*, ceea ce înseamnă că oriunde s-ar apăsa click pe acel card, se va face un fire event către acel *Link*. Acesta v-a naviga user-ul către pagina de detaliu al unui blog.

În pagina de detaliu al unui blog se regăsește în plus, descrierea și comentariile lăsate de către utilizatori:

```

7 <PageTitle>Blog</PageTitle>
8
9 <div class="d-flex flex-column align-items-center gap-3 pt-5">
10   <span class="text-wrap fw-bold fs-1">@_blog.Title</span>
11   <div class="d-flex flex-row justify-content-evenly container-fluid">
12     <span class="fs-3">by <Anchor Class="text-black" To="@($"/account/{_blog.UserName})">@_blog.UserName</Anchor></span>
13     <span class="fs-3 fw-bold">Category: @_blog.BlogCategory</span>
14   </div>
15   <Image Width="@Width.Px(size: 400)" Height="@Height.Px(size: 300)" Source="@($data:image/png;base64,{_blog.Image})" class="rounded-5"/>
16   <div class="fs-4 fw-bold text-wrap container w-75">
17     <pre>@_blog.Description</pre>
18   </div>
19
20   <Divider Shadow="Shadow.Default" Class="w-75 fs-3 fw-bold bg-black" style="height: 2px" />
21   <div class="d-flex flex-column container w-75 gap-3 mb-4">
22     <span class="fs-3 fw-bold">Comments</span>
23     <div class="d-flex flex-column overflow-auto h-50 w-100">
24       @foreach (var comment in _comments)
25       {
26         <div class="d-flex flex-row gap-3">
27           <Image class="rounded-circle"
28             Width="@Width.Px(size: 50)"
29             Height="@Height.Px(size: 50)"
30             Source="@($data:image/png;base64,{comment.UserImage})">
31           </Image>
32           <div class="d-flex flex-column w-100">
33             <span class="fw-bold">@comment.UserName</span>
34             <div class="d-flex flex-row rounded-3 bg-white container align-items-start justify-content-start w-100">
35               <span class="text-wrap fw-bold">@comment.Description</span>
36             </div>
37           </div>
38         </div>
39       }
40     </div>
41     <div class="d-flex flex-row gap-2 align-items-center">
42       <TextEdit class="rounded-pill" Placeholder="Leave a comment..." Role="TextRole.Text" @bind-Text="_comment"></TextEdit>
43       <Button
44         Color="Color.Light"
45         TextWeight="TextWeight.Bold"
46         Clicked="@AddCommentAsync"
47         TextColor="TextColor.Dark"
48         TextSize="TextSize.Default"

```

## 5.5 Sistemul de comentarii

Sistemul de comentarii permite oricărui utilizator să își exprime idei despre o anumită postare. Oricine este înregistrat poate să lase sau să vadă un comentariu din secțiunea de detaliu al unui blog:

```

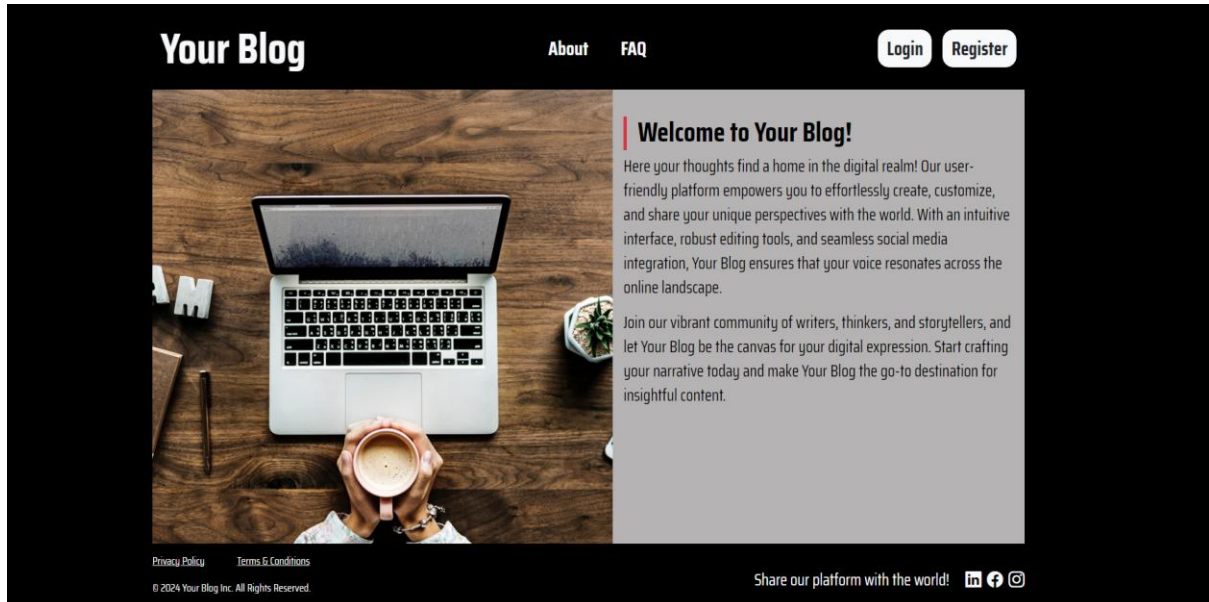
@foreach (var comment in _comments)
{
    <div class="d-flex flex-row gap-3">
        <Image class="rounded-circle"
            Width="@Width.Px(size: 50)"
            Height="@Height.Px(size: 50)"
            Source="@($"data:image/png;base64,{comment.UserImage}")">
        </Image>
        <div class="d-flex flex-column w-100">
            <span class="fw-bold">@comment.UserName</span>
            <div class="d-flex flex-row rounded-3 bg-white container align-items-start justify-content-start w-100">
                <span class="text-wrap fw-bold">@comment.Description</span>
            </div>
        </div>
    </div>
}

40 </div>
41 <div class="d-flex flex-row gap-2 align-items-center">
42     <TextEdit class="rounded-pill" Placeholder="Leave a comment..." Role="TextRole.Text" @bind-Text="_comment"></TextEdit>
43     <Button
44         Color="Color.Light"
45         TextWeight="TextWeight.Bold"
46         Clicked="@AddCommentAsync"
47         TextColor="TextColor.Dark"
48         TextSize="TextSize.Default"
49         Class="rounded-4">
50         <Blazorise.Icon Name="@FontAwesomeIcons.PaperPlane"></Blazorise.Icon>
51     </Button>
52 </div>
53 </div>

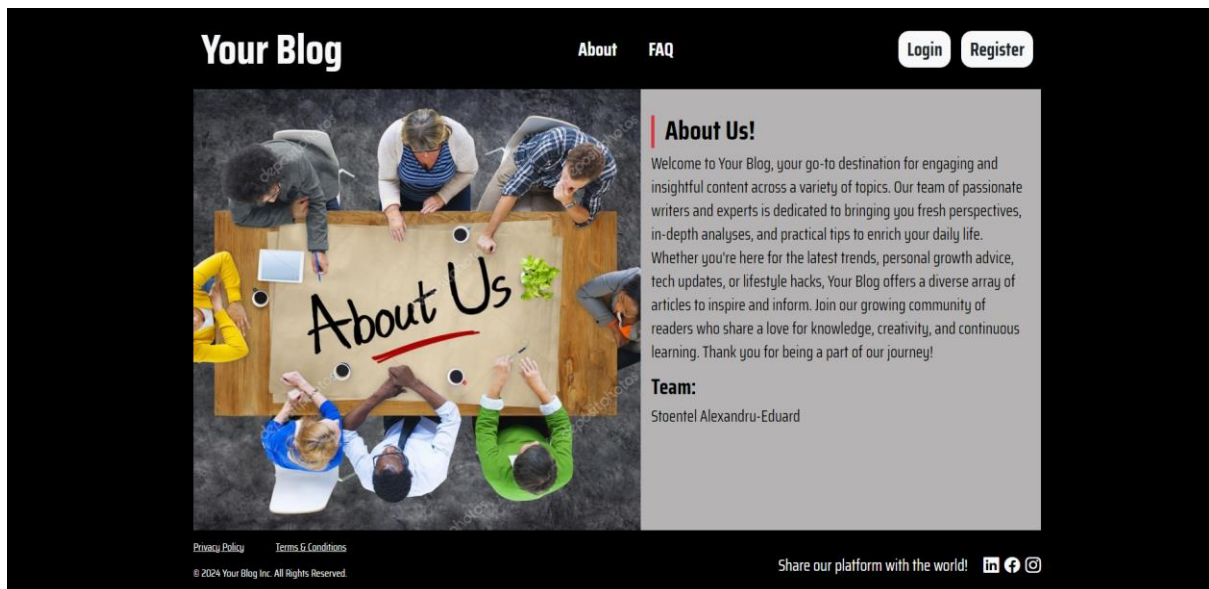
```

## 6. DESFĂȘURAREA APLICAȚIEI

În primă fază, când utilizatorul intră pe platformă, este redirecționat pe pagina principală unde găsește un scurt rezumat al proiectului:



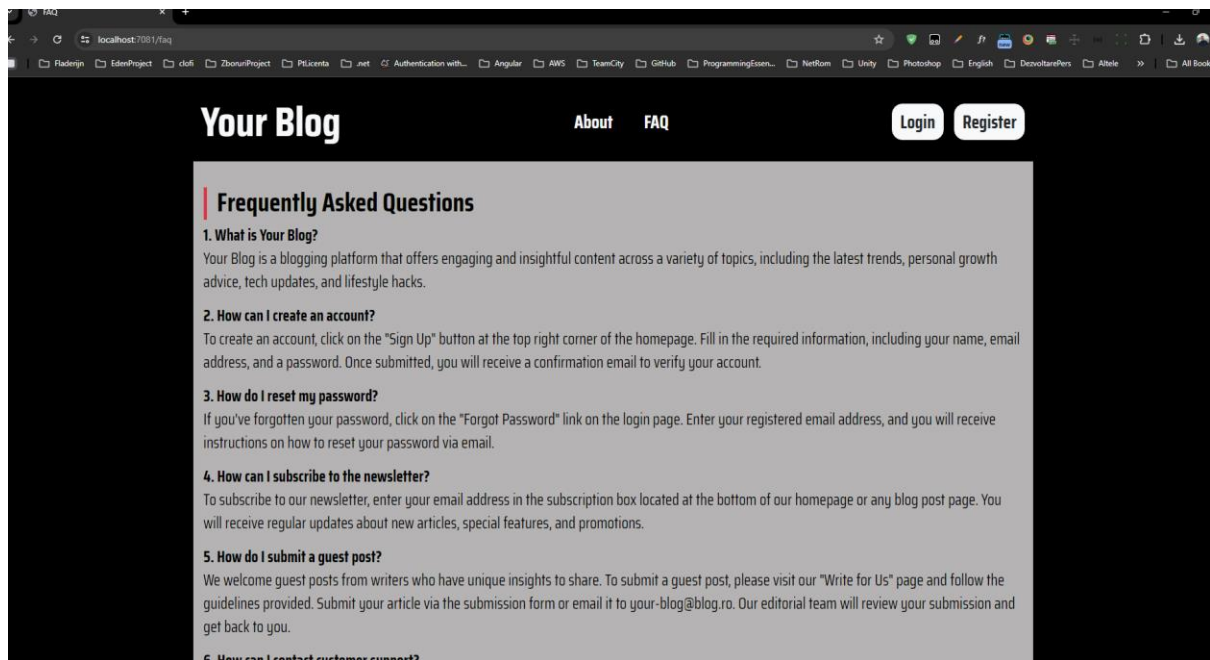
El poate vedea mult mai multe informații la secțiunea *About*:



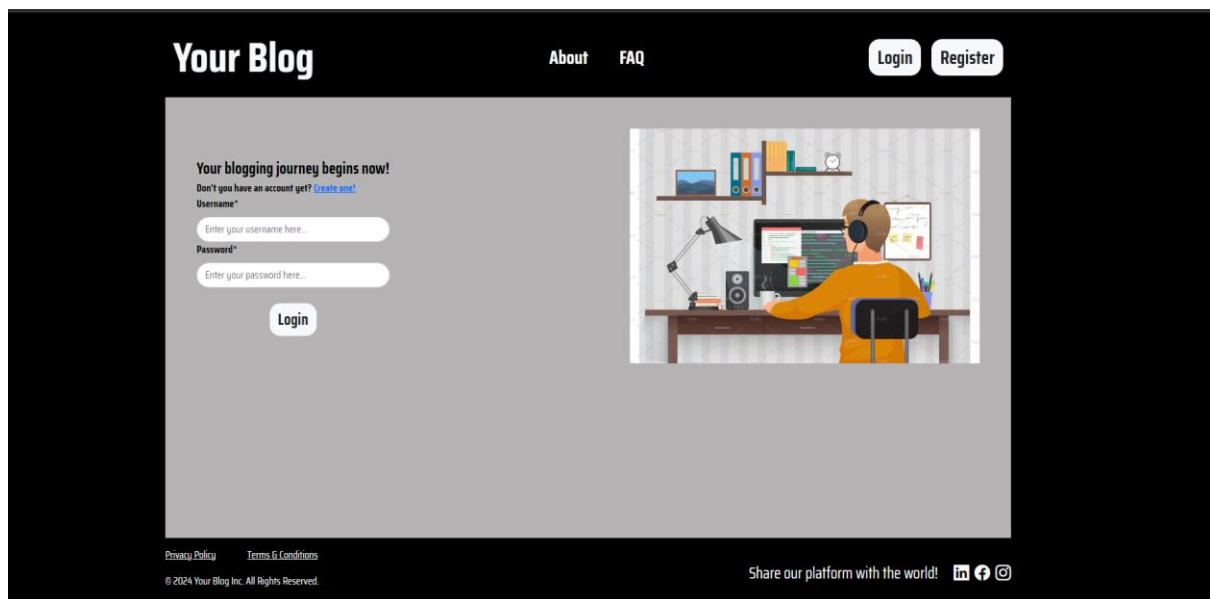
Sau poate vedea întrebările și răspunsurile adminilor, la secțiunea *FAQ*<sup>19</sup>

<sup>19</sup> FAQ - Frequently Asked Questions





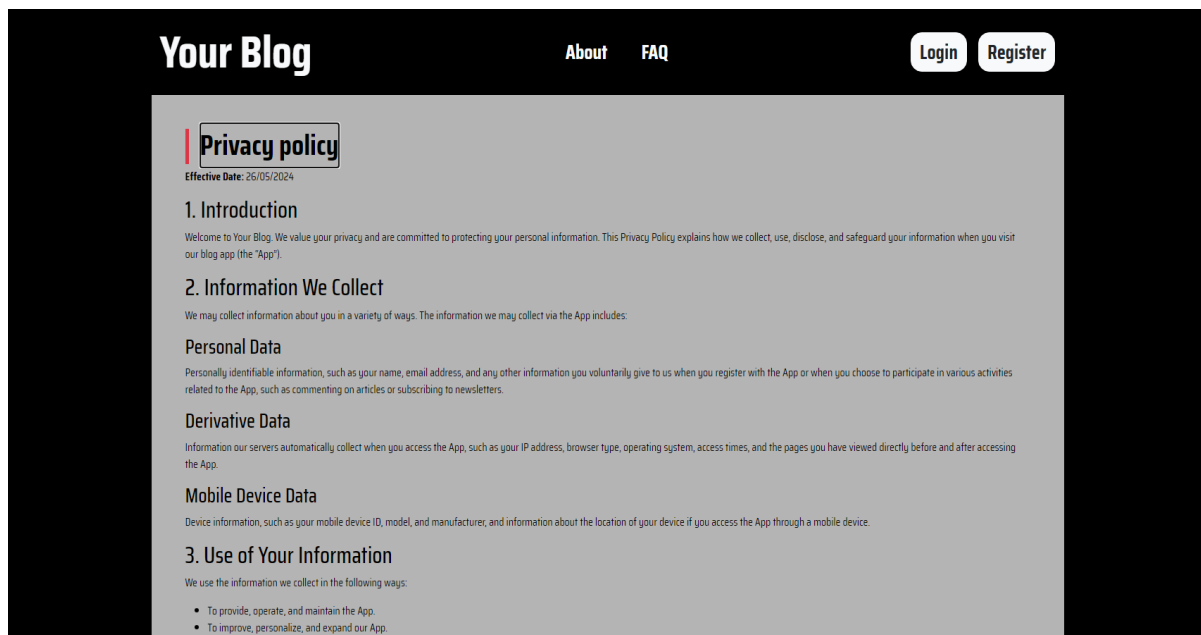
Pentru a se autentifica poate naviga către pagina *Login*:



Dacă utilizatorul dorește să își creeze un cont, trebuie să navigheze către *Register* și să completeze informațiile de acolo. Imaginea de profil este opțională, caz în care este completată cu una predefinită, dacă este omisă.

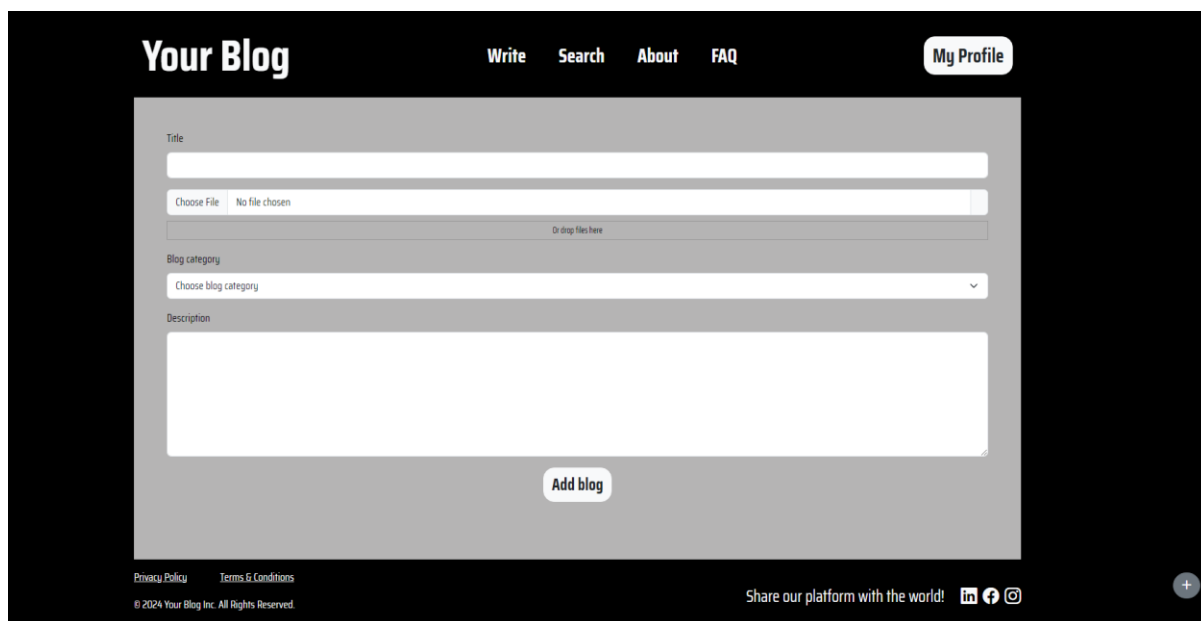
Utilizatorul trebuie să accepte termenii și condițiile aplicației. Poate găsi mai multe informații despre aceștia făcând click pe link-ul din pagina de register sau din secțiunea *footer*.





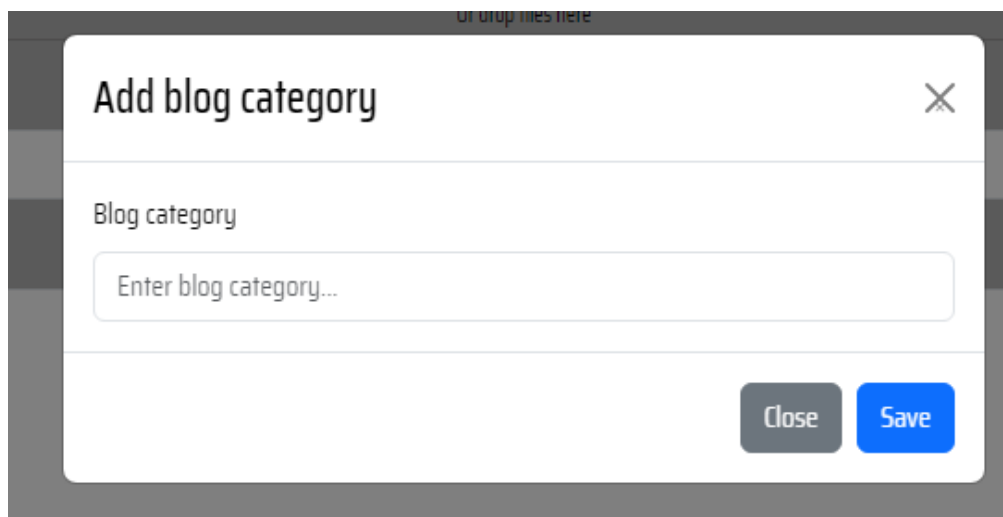
După înregistrare, utilizatorul este redirecționat către pagina de Login pentru a se autentifica. După autentificare, utilizatorului i se deblochează posibilitatea de a vedea profilul lui sau al celorlalți utilizatori, de își edita profilul, de a vedea bloguri, de a crea bloguri și de a comenta.

Pentru a scrie un blog, user-ul trebuie să facă click, din bara de navigare pe *Write*.



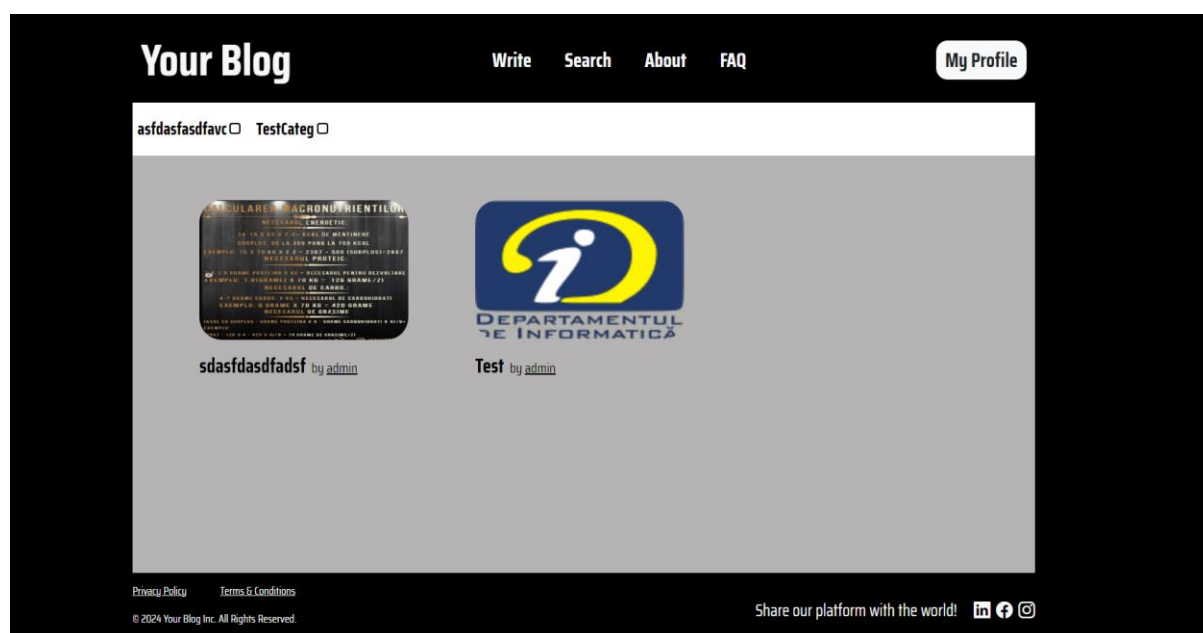
Aici, editorul mimează un *WYSIWYG*<sup>20</sup> editor și oferă posibilitatea user-ului să adauge un titlu, o imagine, o descriere și să selecteze o categorie. Pentru *Admini*, apare un buton suplimentar, de unde poate crea o categorie:

<sup>20</sup> WYSIWYG - what you see is what you get

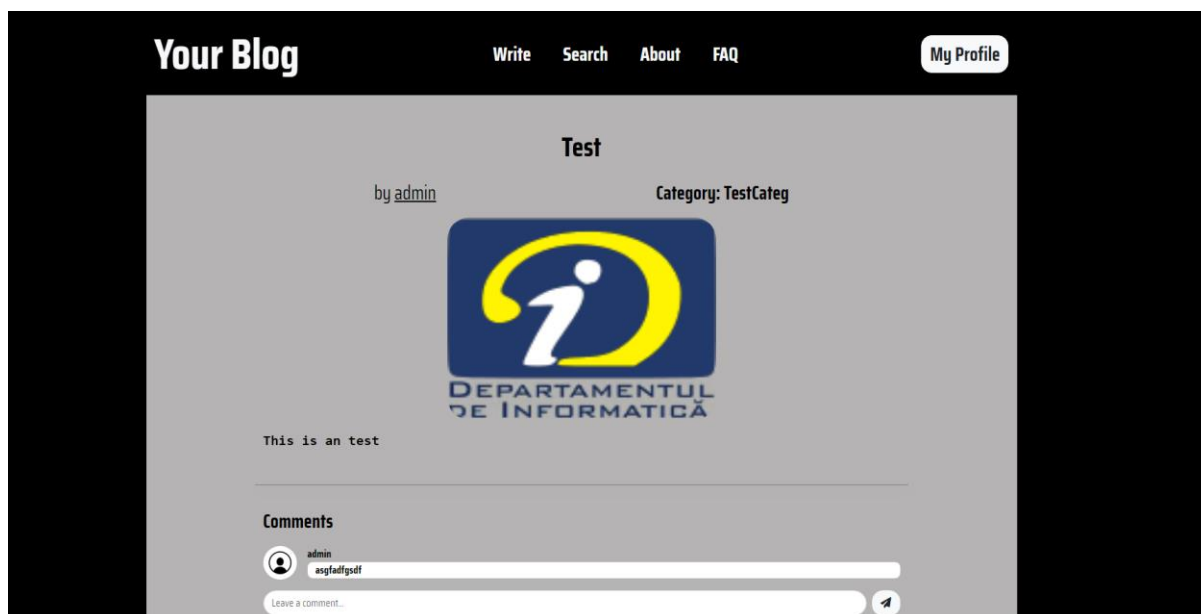


A modal form titled "Add blog category" with a close button (X) in the top right corner. Inside the form, there is a label "Blog category" above a text input field with the placeholder text "Enter blog category...". At the bottom right of the form, there are two buttons: "Close" and "Save".

După completarea informațiilor utilizatorul apasă butonul *Add blog* și este notificat că blogul a fost adăugat cu succes. Apoi este redirecționat către pagina de search, unde poate vedea toate blogurile existente. Acestea pot fi filtrate după categorie sau pot fi căutate din bara de navigare după titlu sau utilizatorul care le-a postat:

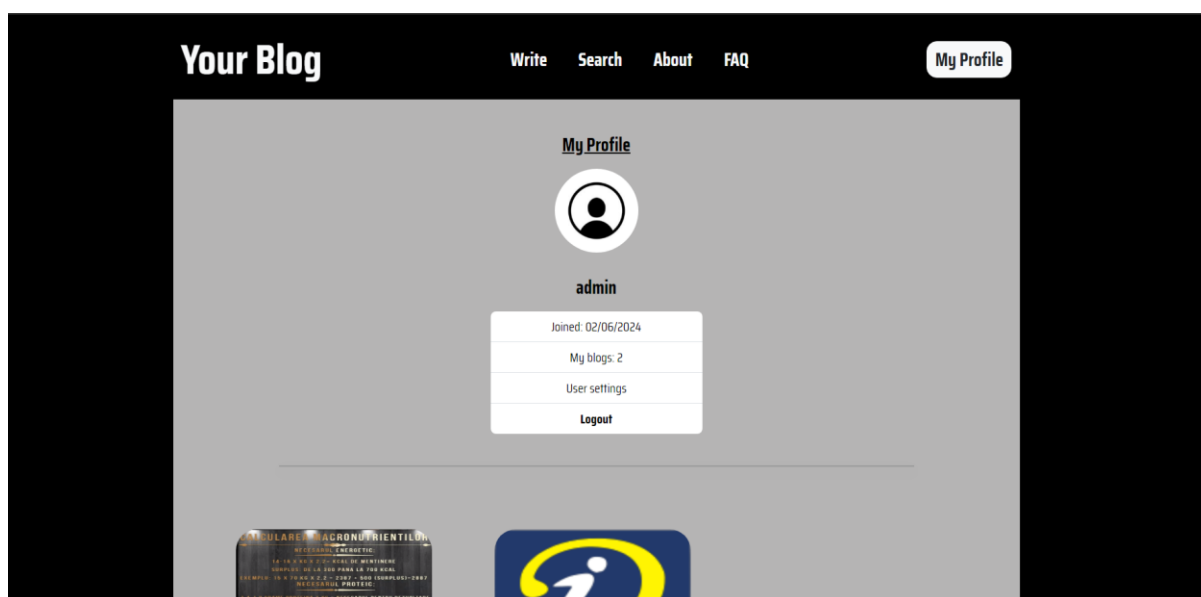


Aici se găsește titlul, imaginea și creatorul fiecărui blog. Prin apăsarea pe unu din bloguri, se deschide o pagina cu detalii blog-ului, unde, în plus, se regăsește descrierea blog-ului și comentariile:



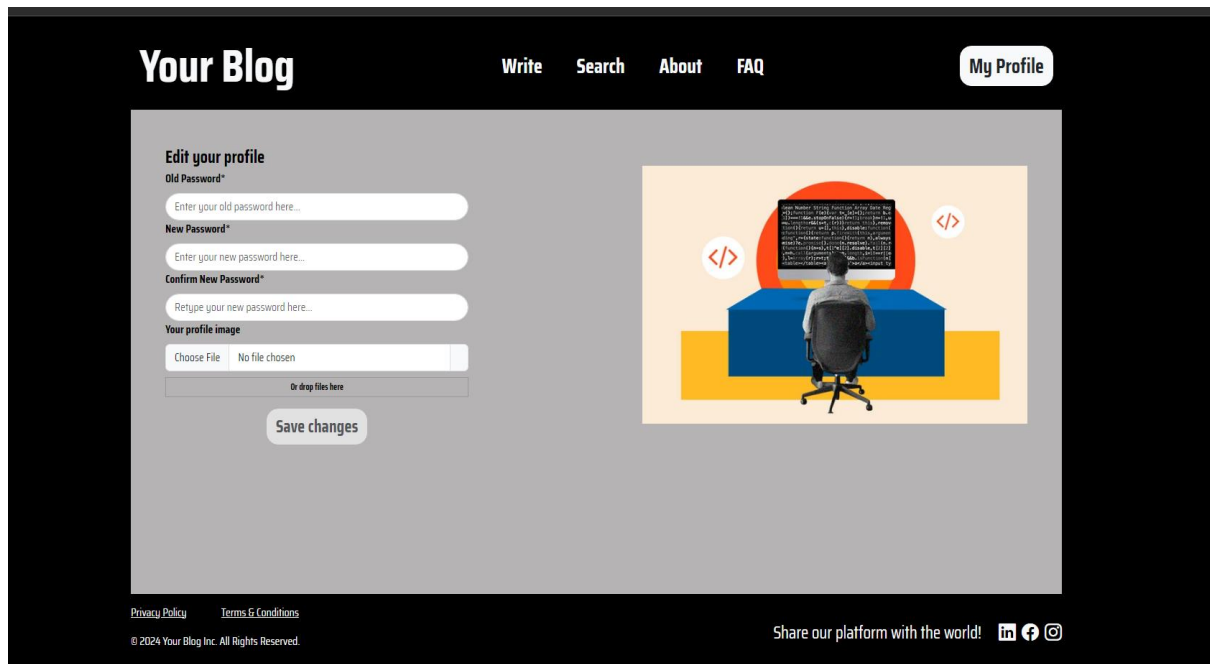
Aici se pot plasa comentarii și vizualiza celelalte comentarii referitoare la această postare.

Pentru a vedea detalii despre utilizatorul care a postat blog-ul, utilizatorul poate face click pe username și este navigat către pagina de profil, unde regăsește informații despre utilizator și toate blog-urile postate de acesta. Dacă utilizatorul este chiar persoana care este logată atunci aceasta poate să și editeze informațiile.



Din această interfață, user-ul se poate deloga.

Pentru a edita informațiile despre profilului, utilizatorul trebuie să facă click pe *User settings* și poate schimba imaginea de profil și parola. Pentru parolă, acesta trebuie să reintroducă parola veche, parola nouă și confirmarea parolei noi pentru a nu exista riscul de typo.



## **7. CONCLUZIE**

În concluzie, subliniez că realizarea acestui proiect a fost o adevărată provocare pentru mine. Pe parcursul întregului proces, am întâmpinat diverse situații care m-au pus la încercare. Cu toate acestea, prin îmbunătățiri constante, multe căutări și documentări, am reușit să duc la bun sfârșit acest proiect.

Pe parcursul anilor de facultate, prin studiu și efort constant, dar și prin realizarea acestui proiect, am reușit să-mi îndeplinesc un scop personal: acela de a acumula cât mai multă experiență profesională și personală.

**SFÂRȘIT.**

## **8. BIBLIOGRAFIE**

C#: C# in Depth, Fourth Edition, Jon Skeet, martie 2019

FullStack: Full Stack Web Development: The Comprehensive Guide, Ackermann Philip, 2023

Database: Learning SQL: Generate, Manipulate, and Retrieve Data, Alan Beaulieu, 2020

## 9. REFERINȚE

*.NET* . (fără an). Preluat de pe <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>.

*ASP.NET Web API* . (fără an). Preluat de pe <https://www.tutorialsteacher.com/webapi/what-is-web-api>.

*Autentificarea Bearer* . (2024). Preluat de pe <https://swagger.io/docs/specification/authentication/bearer-authentication/>.

*AutoInclude* . (fără an). Preluat de pe <https://learn.microsoft.com/en-us/ef/core/querying/related-data/eager>.

*Blazor* . (2024). Preluat de pe <https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0>.

*Blazorise* . (2024). Preluat de pe <https://blazorise.com/docs>.

*Bootstrap* . (2024). Preluat de pe <https://getbootstrap.com/docs/5.3/getting-started/introduction/>.

*C#* . (2024). Preluat de pe <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview>.

*CascadeParameter* . (2024). Preluat de pe <https://learn.microsoft.com/en-us/aspnet/core/blazor/components/cascading-values-and-parameters?view=aspnetcore-8.0>.

*Cascading Style Sheets (CSS)* . (2024). Preluat de pe <https://developer.mozilla.org/en-US/docs/Web/CSS>.

*Code First* . (fără an). Preluat de pe <https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database>.

*Dependency Injection* . (fără an). Preluat de pe <https://stackify.com/dependency-injection/>.

*Diagrama ERD* . (fără an). Preluat de pe <https://www.mindonmap.com/ro/blog/relationship-diagram/>.

*Entity Framework (EF)* . (2024). Preluat de pe <https://learn.microsoft.com/en-us/ef/core/>.  
*entitate* . (2024). Preluat de pe [https://learn.microsoft.com/en-us/previous-versions/office/developer/sharepoint-2010/ee536692\(v=office.14\)](https://learn.microsoft.com/en-us/previous-versions/office/developer/sharepoint-2010/ee536692(v=office.14)).

*HyperText Markup Language (HTML)* . (2024). Preluat de pe <https://en.wikipedia.org/wiki/HTML>.

*Language Integrated Query (LINQ)* . (2024). Preluat de pe <https://www.tutorialsteacher.com/linq/what-is-linq>.

*Local Storage* . (2024). Preluat de pe [https://www.w3schools.com/html/html5\\_webstorage.asp](https://www.w3schools.com/html/html5_webstorage.asp).

- many-to-many* . (2024). Preluat de pe <https://www.entityframeworktutorial.net/code-first/configure-many-to-many-relationship-in-code-first.aspx>.
- Migrațiile*. (2024). Preluat de pe <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli>.
- NuGet Package Console*. (2024). Preluat de pe <https://www.c-sharpcorner.com/UploadFile/22da8c/package-manager-console-in-visual-studio/>.
- one-to-many*. (2024). Preluat de pe <https://www.entityframeworktutorial.net/code-first/configure-one-to-many-relationship-in-code-first.aspx>.
- one-to-one*. (2024). Preluat de pe <https://www.entityframeworktutorial.net/code-first/configure-one-to-one-relationship-in-code-first.aspx>.
- Postman*. (fără an). Preluat de pe [https://en.wikipedia.org/wiki/Postman\\_\(software\)](https://en.wikipedia.org/wiki/Postman_(software)).
- Refit*. (2024). Preluat de pe <https://mwaseemzakir.substack.com/p/ep-32-using-refit-to-consume-apis>.
- SOLID*. (2024). Preluat de pe <https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>.
- SQL Server Management Studio (SSMS)* . (2024). Preluat de pe [https://en.wikipedia.org/wiki/SQL\\_Server\\_Management\\_Studio](https://en.wikipedia.org/wiki/SQL_Server_Management_Studio).
- Structured Query Language (SQL)* . (2024). Preluat de pe <https://www.techtarget.com/searchdatamanagement/definition/SQL>.
- Visual Studio* . (2024). Preluat de pe [https://en.wikipedia.org/wiki/Visual\\_Studio](https://en.wikipedia.org/wiki/Visual_Studio).