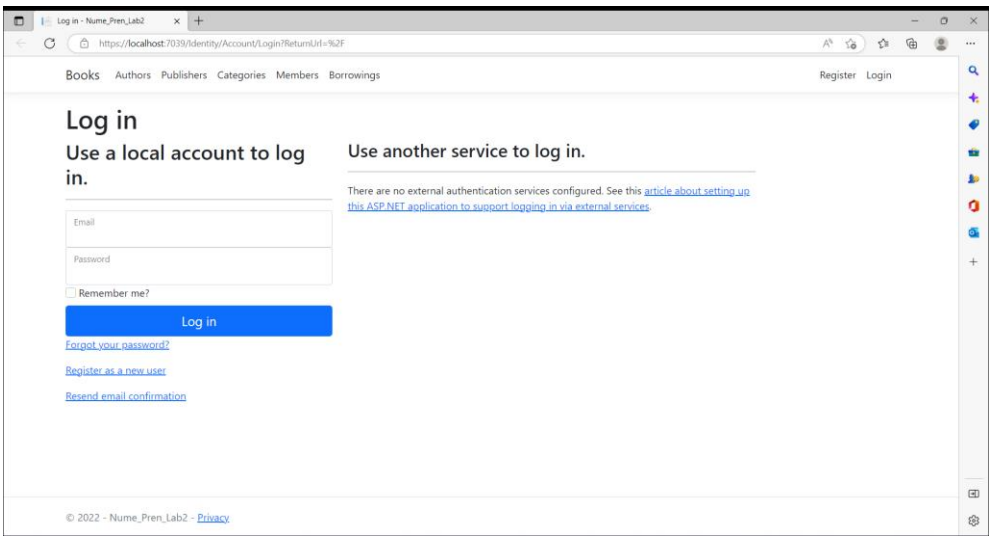
Laborator 6 - Aplicatii Web cu ASP.NET Core si Entity Framework Code First

1. In laboratorul curent, vom dezvolta aplicatia web creata la laboratorul anterior. Deschidem Visual Studio si apoi alegem optiunea **Open a project or solution** si cautam proiectul **Nume_Pren_Lab2,** creat anterior. Pentru a putea fi evaluata activitatea aferenta fiecarui laborator, laboratorul curent il vom dezvolta pe un branch nou, diferit de cel master care se afla deja creat. Utilizand pasii indicati in Lab 1, pct. 22 vom crea un branch nou (based on Laborator5) pe care il vom denumi Laborator6.
2. In laboratorul anterior am utilizat libraria Identity pentru a realiza autentificarea utilizatorilor. In laboratorul curent vom utiliza mai multe metode de autorizare pentru a asigura accesul utilizatorilor in sectiuni specifice ale aplicatiei. O modalitate de a controla accesul în aplicațiile Web cu Razor Pages este să utilizam convențiile de autorizare la pornirea aplicatiei. Aceste convenții ne permit să autorizam utilizatorii și să oferim posibilitatea ca utilizatori anonimi să acceseze pagini individuale sau foldere de pagini.
3. Utilizam convenția AuthorizeFolder pentru a permite accesul doar utilizatorilor autentificati, la paginile dintr-un folder cu calea specificată. In fisierul Program.cs adaugam urmatoarele instructiuni:

```
builder.Services.AddRazorPages(options =>
{
    options.Conventions.AuthorizeFolder("/Books");
});
```

Rulam aplicatia si obervam ca pe pagina \Books\Index nu mai avem acces decat daca ne autentificam



4. Vom modifica conventia de autorizare astfel incat sa pemitem accesul utilizatorilor anonimi pe pagina \Books\Index si Books\Details

```
builder.Services.AddRazorPages(options =>
{
options.Conventions.AuthorizeFolder("/Books");
options.Conventions.AllowAnonymousToPage("/Books/Index");
```

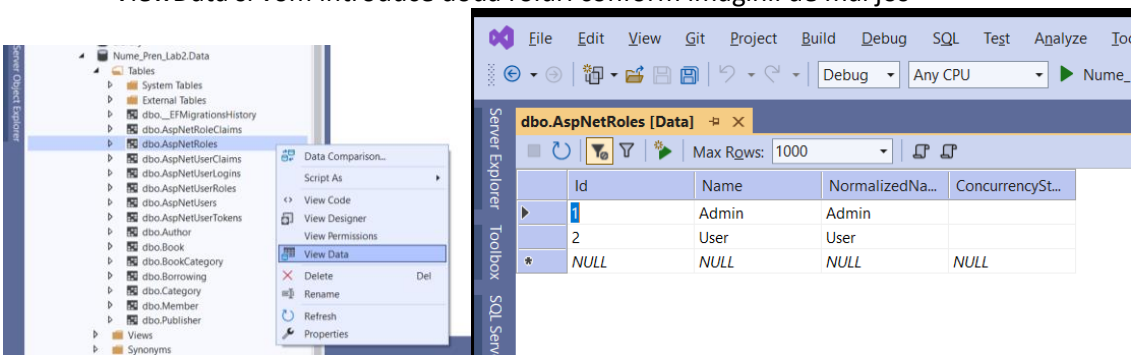
```
    options.Conventions.AllowAnonymousToPage("/Books/Details");
    });
```

Rulam din nou aplicatia si observam ca putem accesa atat pagina \Books\Index si pagina Books\Details, lucru care ofera posibilitate potentialilor viitori membrii ai biblioteci sa poata vizualiza cartile existente in biblioteca, respectiv sa poata cauta daca exista o anumita carte in biblioteca noastra.

5. Când se creează un utilizator, aceasta poate aparține unuia sau mai multor roluri. De exemplu, poate aparține rolurilor de Administrator și/sau Utilizator. Vom inregistra serviciile de autorizare bazate pe roluri în Program.cs apelând AddRoles:

```
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<LibraryIdentityContext>();
```

6. Vom crea doua tipuri de roluri in baza de date. ASP.NET Identity gestioneaza rolurile in tabelul AspNetRoles. Deschidem SQL Server Object Explorer, Click dreapta pe tabelul AspNetRoles->ViewData si vom introduce doua roluri conform imaginii de mai jos



7. In fisierul Register.cshtml.cs, in metoda OnPostAsync vom adauga codul de mai jos care va insera o noua inregistrare in tabelul AspNetUserRoles care va asocia id-ul utilizatorului cu rolul de User

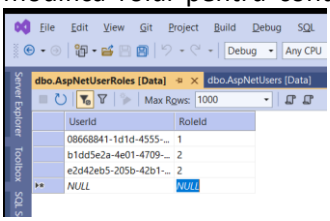
```
.....

if (result.Succeeded)
      {
            _logger.LogInformation("User created a new account with
password.");

    var role = await _userManager.AddToRoleAsync(user,"User");
    var userId = await _userManager.GetUserIdAsync(user);
    var code = await userManager.GenerateEmailConfirmationTokenAsync(user);

….
```

8. Pentru a pastra datele consistente vom seta ca fiind readonly controlul de tip input pentru Email. Deschidem fisierul \Members\Edit.cshtml si adaugam proprietatea readonly pentru Email

```
…..
 <div class="form-group">
                <label asp-for="Member.Email" class="control-
label"></label>
                <input asp-for="Member.Email" readonly class="form-control"
/>
```

```
                <span asp-validation-for="Member.Email" class="text-
danger"></span>
            </div>
…..
```

9. Rulam aplicatia si accesand optiunea Register vom crea doua conturi ion.popescu@gmail.com si admin@gmail.com. Datorita codului adaugat mai sus, ambii utilizatori vor avea rol de User. Vom modifica rolul pentru contul admin@gmail.com in tabelul AspNetUsers modificand RoleId=1



10. Vom autoriza accesul pentru paginile \Books\Create, Books\Edit, Books\Delete doar pentru utilizatori autentificati cu rol de Admin. Deschidem fisierul \Books\Create.cshtml.cs si vom adauga atributul [Authorize] astfel:

```
….
namespace Nume_Pren_Lab2.Pages.Books
{
    [Authorize(Roles = "Admin")]
    public class CreateModel : BookCategoriesPageModel
    {
        private readonly Nume_Pren_Lab2.Data.Nume_Pren_Lab2Context _context;

        public CreateModel(Nume_Pren_Lab2.Data.Nume_Pren_Lab2Context context)
        {
            _context = context;
        }
…..
```

Deschidem apoi fisierele \Books\Edit.cshtml.cs si \Books\Delete.cshtml.cs si procedam la fel ca mai sus.

Rulam aplicatia si observam ca doar utilizatorul admin@gmail.com cu rol de Admin, poate sa acceseze cele trei pagini.

11. Dorim ca sectiunea members sa poata fi accesata doar de utilizatorii cu rol de Admin. Vom crea si configura o politica de autorizare astfel:

```
    ….
    var builder = WebApplication.CreateBuilder(args);


    builder.Services.AddAuthorization(options =>
    {
        options.AddPolicy("AdminPolicy", policy =>
    policy.RequireRole("Admin"));
    });

    // Add services to the container.
    builder.Services.AddRazorPages(options =>
    {
        options.Conventions.AuthorizeFolder("/Books");
        options.Conventions.AllowAnonymousToPage("/Books/Index");
        options.Conventions.AllowAnonymousToPage("/Books/Details");
```

```
        options.Conventions.AuthorizeFolder("/Members", "AdminPolicy");
    });
    ….
```

Rulam aplicatia si observam ca doar utilizatorul admin@gmail.com cu rol de Admin, poate sa acceseze sectiunea Members.

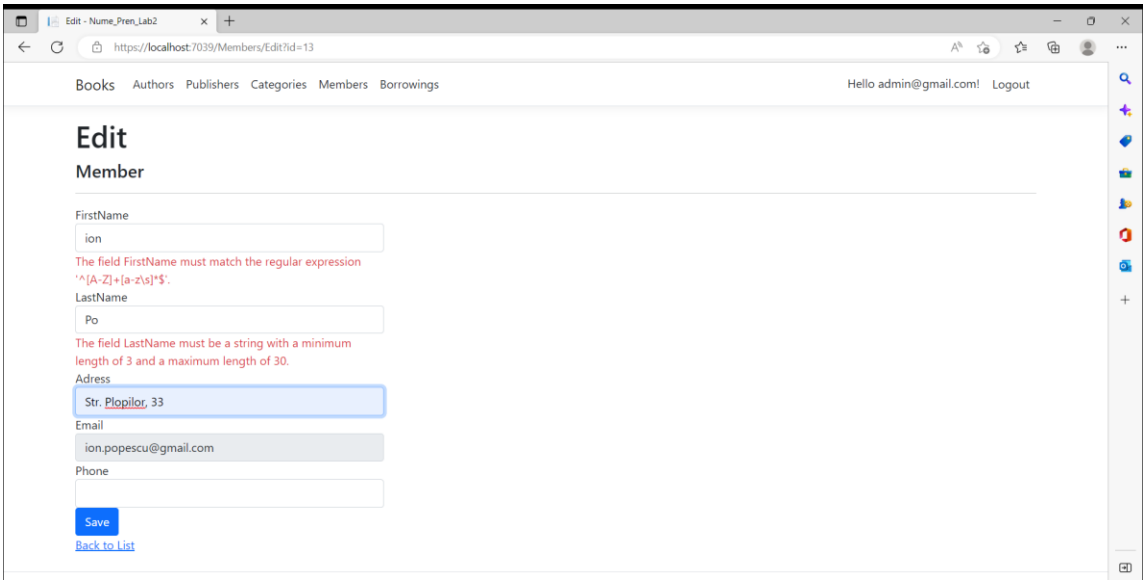
12. Pentru a valida datele de intrare introduse de utilizatori vom utiliza suportul de validare oferit de Razor Pages și Entity Framework. Regulile de validare sunt specificate declarativ într-un singur loc, în clasa modelului si sunt propagate in intreaga aplicatie.

Deschidem fisierul \Models\Member.cs si adaugam urmatoarele adnotari:

```
public class Member
    {

        public int ID { get; set; }

        [RegularExpression(@"^[A-Z]+[a-z\s]*$")]
        [StringLength(30, MinimumLength = 3)]
        public string? FirstName { get; set; }

       [RegularExpression(@"^[A-Z]+[a-z\s]*$")]
       [StringLength(30, MinimumLength = 3)]
        public string? LastName { get; set; }

        [StringLength(70)]
        public string? Adress { get; set; }

        public string Email { get; set; }

        public string? Phone { get; set; }

        [Display(Name = "Full Name")]
        public string? FullName
        {
            get
            {
                return FirstName + " " + LastName;
            }
        }
        public ICollection<Borrowing>? Borrowings { get; set; }

    }
```

Aceste adnotari specifica faptul ca Prenumele/Numele trebuie sa aiba o lungime minima de 3 carcate, maxim 30 si sa respecte expresia regulata specificata (prima litera sa fie majuscula, iar apoi urmeaza o succesiune de listere mici sau spatiu.

13. Dorim sa personalizam mesajul care este afisat daca nu se respecta expresia regulata specificata astfel incat mesajul sa fie intuitiv pentru utilizator. Rulam aplicatia si observam mesajele care apar in cazul in care nu sunt respectate criteriile de validare.



14. Vom adauga un mesaj personalizat astfel:

```
public class Member
    {

        public int ID { get; set; }

          [RegularExpression(@"^[A-Z]+[a-zA-Z\s-]*$", ErrorMessage =
"Prenumele trebuie sa inceapa cu majuscula (ex. Ana sau Ana Maria sau Ana-
Maria")]
        [StringLength(30, MinimumLength = 3)]
        public string? FirstName { get; set; }

       [RegularExpression(@"^[A-Z]+[a-z\s]*$")]
       [StringLength(30, MinimumLength = 3)]
        public string? LastName { get; set; }

        [StringLength(70)]
        public string? Adress { get; set; }

        public string Email { get; set; }

        public string? Phone { get; set; }

        [Display(Name = "Full Name")]
        public string? FullName
        {
            get
            {
                return FirstName + " " + LastName;
            }
        }
        public ICollection<Borrowing>? Borrowings { get; set; }

    }
```

15. Pentru a ne asigura ca numerele de telefon respecta un format predefinit vom adauga o expresie regulata pentru atributul Phone . Vor pute fi introduse numere de telefon de forma '0722-123-123' sau '0722.123.123' sau '0722 123 123'

```
public class Member
    {

        public int ID { get; set; }

        [RegularExpression(@"^[A-Z]+[a-zA-Z\s-]*$", ErrorMessage = "Prenumele
trebuie sa inceapa cu majuscula (ex. Ana sau Ana Maria sau Ana-Maria")]
        [StringLength(30, MinimumLength = 3)]
        public string? FirstName { get; set; }

       [RegularExpression(@"^[A-Z]+[a-z\s]*$")]
       [StringLength(30, MinimumLength = 3)]
        public string? LastName { get; set; }

        [StringLength(70)]
        public string? Adress { get; set; }

        public string Email { get; set; }

        [RegularExpression(@"^\(?([0-9]{4})\)?[-. ]?([0-9]{3})[-. ]?([0-
9]{3})$", ErrorMessage = "Telefonul trebuie sa fie de forma '0722-123-123' sau
'0722.123.123' sau '0722 123 123'")]

        public string? Phone { get; set; }

        [Display(Name = "Full Name")]
        public string? FullName
        {
            get
            {
                return FirstName + " " + LastName;
            }
        }
        public ICollection<Borrowing>? Borrowings { get; set; }

    }
```

16. Pentru a ne asigura validarea datelor pentru clasa Book, deschidem fisierul Book.cs si adaugam urmatoarele adnotari care ne vor asigura ca pretul pentru o carte este un numar zecimal cuprins cu doua zecimale cuprins intre 0.01 si 500, data publicarii este de format Date (in loc de DateTime)

```
public class Book
    {
        public int ID { get; set; }

        public string Title { get; set; }

        [Column(TypeName = "decimal(6, 2)")
        [Range(0.01, 500)]
        public decimal Price { get; set; }

        [DataType(DataType.Date)]
        public DateTime PublishingDate { get; set; }
```

```
        public int? AuthorID { get; set; }
        public Author? Author { get; set; }
        public int? PublisherID { get; set; }
        public Publisher? Publisher { get; set; }

        public Borrowing? Borrowing { get; set; }

        public ICollection<BookCategory>? BookCategories { get; set; }

    }
```

Sarcina laborator :

1. Pentru sectiunile Publishers si Categories confirgurati accesul autorizat doar pentru utilizatorii care au rol de Admin
2. Adaugati pentru Book noi reguli de validare : modificati clasa Book astfel incat la salvarea datelor titlul cartii sa fie completat obiligatoriu, lungimea maxima pentru titlu sa fie de 150 de caractere si o lungime minima de 3 caractere
3. Modificati expresia regulata de la clasa Members pentru atributul Phone astfel incat sa obligam utilizatorul ca prima cifra introdusa sa fie 0