CTI – Dezvoltarea Aplicațiilor Web – Laborator 4
Entity Framework Core + Razor Pages – Proiect: News Portal

Obiective

În acest laborator vom construi baza aplicației care va fi folosită pe parcursul semestrului.

Aplicația va fi un **portal simplu de știri** (News Portal), implementat cu **Razor Pages** și **Entity Framework Core**.

Obiectivele laboratorului:

- Configurarea Entity Framework Core
- Definirea modelelor și relațiilor între entități
- Utilizarea `DbContext` și `DbSet`
- Configurarea Dependency Injection
- Crearea bazei de date folosind migrations
- Afișarea datelor într-o pagină Razor Pages
- Introducerea conceptelor de bază pentru lucrul cu baza de date într-o aplicație web

Tehnologii utilizate

- ASP.NET Core
- Razor Pages
- Entity Framework Core
- SQL Server (LocalDB)

Crearea proiectului

Creați un proiect nou de tip **ASP.NET Core Web App (Razor Pages)**.

Selectați **.NET 8**. Rulați aplicația pentru a verifica că funcționează.

Instalarea pachetelor NuGet

Instalați următoarele pachete din `Tools → NuGet Package Manager → Manage NuGet Packages for Solution`:

- `Microsoft.EntityFrameworkCore 8.0.18`
- `Microsoft.EntityFrameworkCore.SqlServer 8.0.18`
- `Microsoft.EntityFrameworkCore.Tools 8.0.18`

Aceste pachete permit conectarea la baza de date, generarea migrărilor, și executarea comenzilor EF.

Structura proiectului

Adăugați următoarele foldere și fișiere:

```
Models/
    Article.cs
    Category.cs
Data/
    AppDbContext.cs
Pages/
    Articles/
        Index.cshtml
        Index.cshtml.cs
```

Modelele aplicației

Vom avea două tabele: **Categories** și **Articles**. O categorie poate avea mai multe articole.

Modelul `Category`

```
using System.ComponentModel.DataAnnotations;
public class Category
{
    public int Id { get; set; }

    [Required]
    [MinLength(2)]
    public string Name { get; set; } = string.Empty;

    public List<Article> Articles { get; set; } = [];
}
```

Modelul `Article`

```
 using System.ComponentModel.DataAnnotations;
 public class Article
 {
     public int Id { get; set; }

     [Required]
     [MinLength(5)]
     public string Title { get; set; } = string.Empty;

     [Required]
     [MinLength(20)]
     public string Content { get; set; } = string.Empty;

     public DateTime PublishedAt { get; set; } = DateTime.Now;

     public int CategoryId { get; set; }
     public Category? Category { get; set; }
 }
```

DbContext

`DbContext` reprezintă conexiunea dintre aplicație și baza de date. Este clasa principală prin care interacționăm cu baza de date în Entity Framework Core.

Adăugați fișierul `Data/AppDbContext.cs`:

```
using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
    {
    }

    public DbSet<Article> Articles { get; set; }

    public DbSet<Category> Categories { get; set; }
}
```

Clasa `AppDbContext` moștenește `DbContext`, clasa de bază din EF Core care gestionează conexiunea și operațiile cu baza de date.

Constructorul primește un obiect `DbContextOptions<AppDbContext>` care conține configurarea (tipul bazei de date, connection string-ul etc.) și îl pasează clasei de bază prin `: base(options)`.

Fiecare proprietate de tip `DbSet<T>` corespunde unui **tabel** din baza de date. `DbSet<Article>` → tabelul `Articles`, `DbSet<Category>` → tabelul `Categories`. Prin aceste proprietăți vom face query-uri și vom adăuga/modifica/șterge date.

Connection String

În fișierul `appsettings.json` adăugați un câmp pentru connection string:

```
"ConnectionStrings": {
  "Lab04": ""
}
```

Deschideți **SQL Server Object Explorer** (View → SQL Server Object Explorer) și selectați instanța **(localdb)\MSSQLLocalDB**.

Click dreapta → Properties și copiați valoarea **Connection String**.

Lipiți valoarea copiată în câmpul "Lab04" din `appsettings.json`, apoi setați valoarea de la `Initial Catalog` la `NewsDb`.

Rezultatul final ar trebui să arate astfel:

```
"ConnectionStrings": {
  "Lab04": "Data Source=(localdb)\\MSSQLLocalDB;Initial
Catalog=NewsDb;Integrated Security=True;Connect Timeout=30;Encrypt=False;Trust
Server Certificate=False;Application Intent=ReadWrite;Multi Subnet
Failover=False"
}
```

Configurarea DbContext

În `Program.cs` adăugați după `builder.Services.AddRazorPages()`:

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("Lab04")));
```

Această linie înregistrează `AppDbContext` în **containerul de Dependency Injection**. Framework-ul va ști cum să creeze instanțe ale contextului oriunde este cerut.

- `AddDbContext<AppDbContext>(...)` înregistrează contextul ca serviciu disponibil prin DI.
- `UseSqlServer(...)` specifică providerul de baze de date (SQL Server).
- `GetConnectionString("Lab04")` citește connection string-ul cu cheia `"Lab04"` din `appsettings.json`.

Migrations

Migrations sunt mecanismul prin care EF Core traduce modelele C# în structura bazei de date. La fiecare modificare a modelelor, creăm o nouă migrare care descrie schimbările.

În **Package Manager Console** (Tools → NuGet Package Manager → Package Manager Console):

```
Add-Migration InitialCreate
Update-Database
```

Comanda `Add-Migration InitialCreate` generează un fișier cu instrucțiunile SQL necesare pentru a crea tabelele (pe baza DbSet-urilor din `AppDbContext`).

`Update-Database` aplică migrarea și creează efectiv baza de date și tabelele în SQL Server.

Introducerea datelor inițiale

Înainte de a continua, vom adăuga manual câteva înregistrări pentru a avea date de test.

Deschideți **SQL Server Object Explorer** → expandați baza de date **NewsDb** → Tables. Click dreapta pe tabelul `Categories` → **View Data** și adăugați câteva categorii:

Id	Name
1	Tehnologie
2	Sport
3	Cultură

Apoi faceți același lucru pentru tabelul `Articles`:

Id	Title	Content	PublishedAt	CategoryId
1	Inteligența artificială în educație	Universitățile experimentează noi metode de predare.	2026-03-10	1
2	Startul sezonului de Formula 1	Echipele prezintă noile monoposturi pentru sezon.	2026-03-15	2
3	Festival de film european	Proiecții speciale și regizori invitați.	2026-03-09	3
4	Noua generație de procesoare	Performanțe mai bune și consum redus de energie.	2026-03-12	1
5	Turneu internațional de tenis	Jucători din topul mondial participă la competiție.	2026-03-11	2

Notă: `CategoryId` trebuie să corespundă unui `Id` existent din tabelul `Categories`.

Dependency Injection

Deoarece am înregistrat `AppDbContext` în containerul de DI (în `Program.cs`), îl putem primi automat prin constructor în orice clasă gestionată de framework.

În clasa `IndexModel` (code-behind-ul paginii Razor) adăugăm:

```
public class IndexModel : PageModel
{
    private readonly AppDbContext _context;

    public IndexModel(AppDbContext context)
    {
        _context = context;
    }
}
```

Framework-ul creează automat instanța contextului și o pasează constructorului.

Razor Pages – Pagina Index

Pagina Index va afișa lista știrilor.

Index.cshtml.cs

```
namespace Lab04.Pages;

using Lab04.Data;
using Lab04.Models;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;

public class IndexModel : PageModel
{
    private readonly AppDbContext _context;

    public IndexModel(AppDbContext context)
    {
        _context = context;
    }

    public List<Article> Articles { get; set; } = [];

    public void OnGet()
    {
        Articles = _context.Articles
            .Include(a => a.Category)
            .ToList();
    }
}
```

Eager loading reprezintă încărcarea entităților asociate în același query cu entitatea principală, folosind metoda `Include()`. Entity Framework generează de obicei un `JOIN` pentru a obține toate datele necesare dintr-o singură interogare.

Proprietatea `Articles` este publică, astfel valorile setate aici vor fi accesibile din fișierul `.cshtml` prin `Model.Articles`.

Metoda `OnGet()` este apelată automat când pagina primește un request `GET`.

Trimiterea datelor suplimentare cu ViewData / ViewBag

Pe lângă proprietățile publice, putem trimite date către pagină folosind `ViewData`, un dicționar `string → object`.

În metoda `OnGet()` putem popula `ViewData`:

```
ViewData["Title"] = "Lista de stiri";
ViewData["Count"] = _context.Articles.Count();
```

În `.cshtml` accesăm valorile astfel:

```
<h1>@ViewData["Title"]</h1>
<p>Total: @ViewData["Count"] articole</p>
```

`ViewBag` este un wrapper dinamic peste `ViewData`. Permite acces cu sintaxă de proprietate în loc de dicționar:

```
// in .cshtml.cs
ViewData["Title"] = "Stiri";       // ViewData - dictionar

// in .cshtml - ambele variante functioneaza:
@ViewData["Title"]
@ViewBag.Title
```

Primirea parametrilor din URL (query string)

Metoda `OnGet` poate primi parametri direct din URL. De exemplu, pentru `/Articles?category=sport`:

```
public void OnGet(string? category)
{
    // category va avea valoarea "sport" din query string
}
```

Framework-ul face automat legătura între numele parametrului din metodă și cel din URL (**model binding**).

Rutele în Razor Pages

În laboratorul anterior am definit rute explicit pe controllere cu `[Route()]` și `[HttpGet()]`.
În **Razor Pages**, rutele sunt generate **automat** pe baza structurii de foldere din `Pages/`:

Fișier	Ruta generată
Pages/Index.cshtml	`/` sau `/Index`
Pages/Articles/Index.cshtml	`/Articles` sau `/Articles/Index`
Pages/Articles/Details.cshtml	`/Articles/Details`

Nu trebuie să configurăm nimic, framework-ul mapează automat calea fișierului la URL.

Pentru a adăuga **parametri în rută**, folosim directiva `@page` cu un template:

```
@page "{id:int}"
```

Aceasta face ca pagina să răspundă la `/Articles/Details/5`, unde `5` este valoarea parametrului `id`. Parametrul va fi disponibil în metoda `OnGet(int id)`.

Index.cshtml

```
@page
@model IndexModel

<h1>Stiri</h1>

<table>
    <thead>
        <tr>
            <th>Title</th>
            <th>Date</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var article in Model.Articles)
        {
            <tr>
                <td>@article.Title</td>
                <td>@article.PublishedAt.ToShortDateString()</td>
            </tr>
        }
    </tbody>
</table>
```

`@page` este directivă obligatorie care marchează fișierul ca **Razor Page**. Fără ea, fișierul nu va fi tratat ca pagină și nu va răspunde la request-uri.

`@model IndexModel` specifică **tipul modelului** asociat paginii. Prin Model (cu M mare) accesăm instanța clasei `IndexModel` și proprietățile ei publice (ex: `Model.Articles`).

`@foreach`, `@article.Title` folosesc sintaxa Razor: prefixul `@` permite scrierea de expresii C# direct în HTML.

Fluxul unei cereri de la Browser la baza de date

Componentă	Acțiune (Request)	Acțiune (Response)
Browser	Trimite HTTP Request (GET/POST)	Primește și afișează HTML
Razor Page (.cshtml)	Primește cererea (prin rute)	Randează codul C# în HTML final
PageModel (.cshtml.cs)	Procesează input-ul, apelează metode	Pregătește datele pentru View (Model)
DbContext (EF Core)	Traduce interogarea LINQ în SQL	Mapează rezultatele SQL în obiecte C#
SQL Server	Execută interogarea pe tabele	Returnează rândurile de date

Referință — Metode EF Core utilizate

Metodă	Rol
DbContext	Conexiunea la baza de date
DbSet<T>	Reprezintă un tabel
Include()	Încarcă relații (eager loading)
ToList()	Execută query-ul și returnează o listă
SaveChanges()	Salvează modificările

Referință — Metode Razor Pages utilizate

Metodă	Rol
PageModel	Code-behind pentru pagină
OnGet()	Handler pentru request GET
Page()	Returnează pagina curentă
RedirectToPage()	Redirect către altă pagină
ModelState.IsValid	Verifică validarea modelului

Documentație EF Core: https://learn.microsoft.com/en-us/ef/core/

Documentație Razor Pages: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/

Razor Pages + EF Core tutorial: https://learn.microsoft.com/en-us/aspnet/core/data/ef-rp/

Exerciții

1. **(1p)** Completați pagina Index astfel încât să afișeze și categoria articolului.
2. **(1p)** Ordonați articolele descrescător după PublishedAt.
3. **(2p)** Creați o pagină **Details** care afișează un singur articol.
    - Ruta trebuie să fie /Articles/Details/{id}
    - Dacă articolul nu există, returnați NotFound()
4. **(2p)** Creați o pagină **Create** pentru adăugarea unui articol.
    - Folosiți ModelState.IsValid pentru validare
    - După salvare, faceți redirect către Index
5. **(1p)** Adăugați un link **Details** în tabelul din pagina Index.