CTI – Dezvoltarea Aplicațiilor Web – Laborator 5
Razor Pages – CRUD complet, rutare, Tag Helpers, paginare

Obiective

Laboratorul continuă proiectul **News Portal** din Lab 4. La finalul acestui laborator, aplicația va avea un **CRUD complet** pentru știri, cu validare, rutare cu parametri și paginare.

Obiectivele laboratorului:

- Înțelegerea Tag Helpers: `asp-for`, `asp-page`, `asp-route-{param}`, `asp-validation-for`
- Rutare cu parametri (`@page "{id:int}"`)
- Atributul `[BindProperty]` pentru model binding în formulare
- Tipul returnat `IActionResult` pentru `OnGet` / `OnPost`
- Implementare completă: Details, Create, Edit, Delete
- Paginare simplă cu `Skip()` / `Take()`
- Adăugarea modelului `User` și legătura cu `Article`

Recapitulare Lab 4

Din laboratorul anterior avem:

- Modelele `Article` și `Category` cu `DataAnnotations`
- `AppDbContext` cu `DbSet<Article>` și `DbSet<Category>`
- Connection string + DI configurat
- Migrations create și aplicate
- Pagina **Index** care afișează lista articolelor

Creați următoarea structură de directoare:

```
Models/
    Article.cs
    Category.cs
    User.cs                ← NOU
Pages/
    Index.cshtml           ← lista articolelor
    Index.cshtml.cs
    Articles/
        Details.cshtml
        Details.cshtml.cs
        Create.cshtml
        Create.cshtml.cs
        Edit.cshtml
        Edit.cshtml.cs
        Delete.cshtml
        Delete.cshtml.cs
```

Modelul User

Înainte de a implementa CRUD-ul, adăugăm un model nou: `User`. Acesta va fi folosit în laboratoarele viitoare pentru autentificare și pentru a identifica autorul fiecărui articol.

Models/User.cs

```
namespace Lab05.Models;
using System.ComponentModel.DataAnnotations;

public class User
{
    public int Id { get; set; }

    [Required]
    [MinLength(3)]
    public string Name { get; set; } = string.Empty;

    [Required]
    [EmailAddress]
    public string Email { get; set; } = string.Empty;

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

Actualizare Article: adăugare FK către User

Adăugați următoarele proprietăți în clasa `Article`:

```
public int? UserId { get; set; }
public User? User { get; set; }
```

Actualizare AppDbContext

Adăugați noul `DbSet` în `AppDbContext`:

```
public DbSet<User> Users { get; set; }
```

Migrare

După aceste modificări, creați și aplicați o migrare nouă:

```
Add-Migration AddUser
Update-Database
```

Tag Helpers

Tag Helpers permit scrierea de atribute speciale pe elementele HTML, care sunt procesate pe server.

Tag Helper	Rol	Exemplu
`asp-page`	Generează link către Razor Page	`<a asp-page="Details">`
`asp-route-{param}`	Trimite un parametru în rută	`<a asp-route-id="@article.Id">`
`asp-for`	Leagă un `<input>` de o proprietate a modelului	`<input asp-for="Article.Title" />`
`asp-validation-for`	Afișează mesajul de validare pentru o proprietate	`<span asp-validation-for="Article.Title"></span>`
`asp-validation-summary`	Afișează toate erorile de validare	`<div asp-validation-summary="All"></div>`
`asp-items`	Populează un `<select>` cu opțiuni	`<select asp-for="Article.CategoryId" asp-items="Model.Categories">`

Exemplu: link către Details cu id:

```
<a asp-page="Articles/Details" asp-route-id="@article.Id">@article.Title</a>
```

Exemplu: input legat de model:

```
<input asp-for="@article.Title" class="form-control" />
<span asp-validation-for="@article.Title" class="text-danger"></span>
```

Tag helper-ul `asp-for` generează automat atributele `name`, `id`, `value` și `type` pe baza proprietății modelului.

Rutare cu parametri

Directiva `@page` poate include parametri de rută cu constrângeri de tip:

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

Aceasta generează ruta `/Articles/Details/3`, unde `3` este valoarea parametrului `id`. Constrângerea `:int` asigură că doar valori numerice sunt acceptate.

Referințe absolute și relative la pagini

Când folosim `asp-page` sau `RedirectToPage`, calea poate fi relativă sau absolută:

- Relativă (`"Index"`) este rezolvată față de folderul paginii curente. Din `Pages/Articles/Edit.cshtml`, `"Index"` → `Pages/Articles/Index.cshtml`.
- Absolută (`"/Index"`) este rezolvată față de folderul `Pages/`. `"/Index"` → `Pages/Index.cshtml` indiferent de unde e apelat.

Deoarece pagina Index se află la `Pages/Index.cshtml`, iar paginile CRUD se află în `Pages/Articles/`, toate referințele spre Index trebuie să folosească calea absolută:

```
return RedirectToPage("/Index");   // corect — merge la Pages/Index.cshtml
                                   // return RedirectToPage("Index"); //
gresit — ar cauta Pages/Articles/Index.cshtml
```

Același principiu se aplică și în `.cshtml`:

```
<a asp-page="/Index">← Înapoi</a>
<a asp-page="/Articles/Details" asp-route-id="@article.Id">Detalii</a>
```

Parametrul din rută este mapat automat pe un parametru al metodei handler:

```
public IActionResult OnGet(int id)
{
    // id este extras din URL
}
```

[BindProperty]

Atributul `[BindProperty]` permite model binding automat din formular.

Când utilizatorul trimite un formular (POST), framework-ul populează automat proprietatea decorată cu `[BindProperty]` cu datele din formular.

```
[BindProperty]
public Article Article { get; set; }
```

Fără `[BindProperty]`, proprietatea ar rămâne `null` la POST.

IActionResult

Metodele `OnGet` și `OnPost` pot returna `IActionResult`, ceea ce permite returnarea de rezultate diferite:

Metodă	Efect
`Page()`	Returnează pagina curentă (afișează view-ul)
`RedirectToPage("Index")`	Redirect (HTTP 302) către pagina Index
`NotFound()`	Returnează HTTP 404

Exemplu:

```
public IActionResult OnGet(int id)
{
    Article = _context.Articles.Include(a => a.Category)
        .FirstOrDefault(a => a.Id == id);
    if (Article == null) return NotFound();
    return Page();
}
```

Pagina Details

Afișează o singură știre, identificată prin `id` din URL.

Details.cshtml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using Lab05.Data;
using Lab05.Models;

namespace Lab05.Pages.Articles;

public class DetailsModel : PageModel
{
    private readonly AppDbContext _context;

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

    public Article? Article { get; set; }
    public IActionResult OnGet(int id)
    {
        Article = _context.Articles
            .Include(a => a.Category)
            .FirstOrDefault(a => a.Id == id);
        if (Article == null)
            return NotFound();
        return Page();
    }
}
```

Details.cshtml

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

<h1>@Model.Article.Title</h1>

<p><strong>Category:</strong> @Model.Article.Category.Name</p>
<p><strong>Data publicării:</strong>
@Model.Article.PublishedAt.ToShortDateString()</p>
<p>@Model.Article.Content</p>

<a asp-page="/Index">← Înapoi la listă</a>
```

Pagina Create

Permite adăugarea unei știri noi.

List<SelectListItem> pentru categorii

Pentru dropdown-ul de categorii, construim manual o `List<SelectListItem>`:

```
public List<SelectListItem> Categories { get; set; } = default!;
```

Se inițializează atât în `OnGet` cât și în `OnPost` (în caz de erori de validare):

```
Categories = _context.Categories
.Select(c => new SelectListItem
{
    Value = c.Id.ToString(),
    Text = c.Name
})
.ToList();
```

Fiecare `SelectListItem` are `Value` (valoarea trimisă la POST) și `Text` (textul afișat în dropdown). Interogarea LINQ proiectează fiecare `Category` într-un `SelectListItem`.

Create.cshtml.cs

```
using Lab05.Data;
using Lab05.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace Lab05.Pages.Articles;

public class CreateModel : PageModel
{
    private readonly AppDbContext _context;

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

    [BindProperty]
    public Article Article { get; set; } = default!;

    public List<SelectListItem> Categories { get; set; } = [];

    private void LoadCategories()
    {
        Categories = _context.Categories
            .Select(c => new SelectListItem
            {
                Value = c.Id.ToString(),
```

```
                Text = c.Name
            })
            .ToList();
    }

    public IActionResult OnGet()
    {
        LoadCategories();
        return Page();
    }

    public IActionResult OnPost()
    {
        if (!ModelState.IsValid)
        {
            LoadCategories();
            return Page();
        }

        Article.PublishedAt = DateTime.Now;

        _context.Articles.Add(Article);
        _context.SaveChanges();

        return RedirectToPage("/Index");
    }
}
```

Create.cshtml

```
@page
@model CreateModel

<h1>Adaugă articol</h1>

<form method="post">
    <div>
        <label asp-for="Article.Title"></label>
        <input asp-for="Article.Title" class="form-control" />
        <span asp-validation-for="Article.Title" class="text-danger"></span>
    </div>

    <div>
        <label asp-for="Article.Content"></label>
        <textarea asp-for="Article.Content" class="form-control"></textarea>
        <span asp-validation-for="Article.Content" class="text-
danger"></span>
    </div>

    <div>
        <label asp-for="Article.CategoryId"></label>
```

```
        <select asp-for="Article.CategoryId" asp-items="Model.Categories"
class="form-control">
            <option value="">-- Selectați categoria --</option>
        </select>
        <span asp-validation-for="Article.CategoryId" class="text-
danger"></span>
    </div>

    <button type="submit" class="btn btn-primary">Salvează</button>
</form>

<a asp-page="/Index">← Înapoi la listă</a>
```

Câmpul `PublishedAt` este setat automat în `OnPost`, nu apare în formular.

Pagina Edit

Permite modificarea unei știri existente. Formularul este similar cu Create, dar precompletează câmpurile cu valorile existente.

Edit.cshtml.cs

```
using Lab05.Data;
using Lab05.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace Lab05.Pages.Articles;
public class EditModel : PageModel
{
    private readonly AppDbContext _context;

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

    [BindProperty]
    public Article? Article { get; set; }

    public List<SelectListItem> Categories { get; set; } = [];

    private void LoadCategories()
    {
        Categories = _context.Categories
            .Select(c => new SelectListItem
            {
                Value = c.Id.ToString(),
                Text = c.Name
            })
```

```
            .ToList();
    }

    public IActionResult OnGet(int id)
    {
        Article = _context.Articles.Find(id);

        if (Article == null)
        {
            return NotFound();
        }

        LoadCategories();
        return Page();
    }

    public IActionResult OnPost()
    {
        if (!ModelState.IsValid)
        {
            LoadCategories();
            return Page();
        }
        _context.Update(Article!);
        _context.SaveChanges();

        return RedirectToPage("/Index");
    }
}
```

Atenție: `[BindProperty]` populează obiectul `Article` din formular, inclusiv `Id`-ul. Fără câmpul hidden pentru `Id`, EF Core ar crea o resursă nouă în loc să o actualizeze pe cea existentă.

Metoda `_context.Update()` marchează entitatea ca modificată. La apelul `SaveChanges()`, EF generează un `UPDATE` SQL.

Edit.cshtml

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

<h1>Editează articolul</h1>

<form method="post">
    <input type="hidden" asp-for="Article.Id" />

    <div>
        <label asp-for="Article.Title"></label>
        <input asp-for="Article.Title" class="form-control" />
        <span asp-validation-for="Article.Title" class="text-danger"></span>
    </div>
```

```
    <div>
        <label asp-for="Article.Content"></label>
        <textarea asp-for="Article.Content" class="form-control"></textarea>
        <span asp-validation-for="Article.Content" class="text-
danger"></span>
    </div>

    <div>
        <label asp-for="Article.CategoryId"></label>
        <select asp-for="Article.CategoryId" asp-items="Model.Categories"
class="form-control">
            <option value="">-- Selectați categoria --</option>
        </select>
        <span asp-validation-for="Article.CategoryId" class="text-
danger"></span>
    </div>

    <button type="submit" class="btn btn-success">Actualizează</button>
</form>

<a asp-page="/Index">← Înapoi la listă</a>
```

Câmpul `<input type="hidden" asp-for="Article.Id" />` este esențial: fără el, `Article.Id` ar fi `0` la POST, iar EF Core ar insera o entitate nouă în loc să o actualizeze.

Pagina Delete

Afișează o confirmare înainte de ștergere. GET afișează detaliile, POST execută ștergerea.

Delete.cshtml.cs

```
namespace Lab05.Pages.Articles;

using Lab05.Data;
using Lab05.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;

public class DeleteModel : PageModel
{
    private readonly AppDbContext _context;

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

    public Article Article { get; set; } = default!;

    public IActionResult OnGet(int id)
    {
```

```
        var article = _context.Articles
            .Include(a => a.Category)
            .FirstOrDefault(a => a.Id == id);

        if (article == null)
        {
            return NotFound();
        }
        Article = article;
        return Page();
    }

    public IActionResult OnPost(int id)
    {
        var article = _context.Articles.Find(id);
        if (article == null)
        {
            return NotFound();
        }

        _context.Articles.Remove(article);
        _context.SaveChanges();

        return RedirectToPage("/Index");
    }
}
```

Delete.cshtml

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

<h1>Ștergere articol</h1>

<p>Sigur doriți să ștergeți acest articol?</p>

<div>
    <h3>@Model.Article.Title</h3>
    <p><strong>Category:</strong> @Model.Article.Category!.Name</p>
    <p><strong>Data:</strong>
@Model.Article.PublishedAt.ToShortDateString()</p>
    <p>@Model.Article.Content</p>
</div>

<form method="post">
    <button type="submit" class="btn btn-danger">Confirmă ștergerea</button>
</form>
<a asp-page="/Index">← Înapoi la listă</a>
```

Delete-ul folosește **POST** pentru acțiunea de ștergere, nu GET. O ștergere prin GET ar fi o vulnerabilitate (un link simplu ar putea șterge date).

Actualizarea paginii Index

Actualizăm pagina Index pentru a include linkuri către Details, Edit, Delete, precum și un link către Create.

Index.cshtml

```
@page
@model IndexModel

<h1>Articles</h1>

<p><a asp-page="/Articles/Create" class="btn btn-primary">+ Adaugă
articol</a></p>

<table class="table">
    <thead>
        <tr>
            <th>Title</th>
            <th>Category</th>
            <th>Data</th>
            <th>Acțiuni</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var article in Model.Articles)
        {
            <tr>
                <td>@article.Title</td>
                <td>@article.Category!.Name</td>
                <td>@article.PublishedAt.ToShortDateString()</td>
                <td>
                    <a asp-page="/Articles/Details" asp-route-
id="@article.Id">Detalii</a> |
                    <a asp-page="/Articles/Edit" asp-route-
id="@article.Id">Editează</a> |
                    <a asp-page="/Articles/Delete" asp-route-
id="@article.Id">Șterge</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Paginare simplă cu Skip / Take

Pentru a nu afișa toate știrile pe o singură pagină, implementăm o paginare simplă.

Index.cshtml.cs cu paginare

```
using Lab05.Data;
using Lab05.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;

namespace Lab05.Pages;

public class IndexModel : PageModel
{
    private readonly AppDbContext _context;
    private const int PageSize = 5;

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

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

    public void OnGet(int? pageNumber)
    {
        CurrentPage = pageNumber ?? 1;

        var totalItems = _context.Articles.Count();
        TotalPages = (int)Math.Ceiling(totalItems / (double)PageSize);

        Articles = _context.Articles
            .Include(a => a.Category)
            .OrderByDescending(a => a.PublishedAt)
            .Skip((CurrentPage - 1) * PageSize)
            .Take(PageSize)
            .ToList();
    }
}
```

`Skip()` sare peste elementele din paginile anterioare, `Take()` selectează doar elementele paginii curente.

Parametrul `pageNumber` vine din query string: `/?pageNumber=2`.

Linkuri de paginare în Index.cshtml

Adăugați la finalul paginii, după `</table>`:

```
<nav>
    @if (Model.CurrentPage > 1)
    {
        <a asp-page="/Index" asp-route-pageNumber="@(Model.CurrentPage -
1)">← Pagina anterioară</a>
    }

    <span>Pagina @Model.CurrentPage din @Model.TotalPages</span>

    @if (Model.CurrentPage < Model.TotalPages)
    {
        <a asp-page="/Index" asp-route-pageNumber="@(Model.CurrentPage +
1)">Pagina următoare →</a>
    }
</nav>
```

Fișiere statice și imagini pentru articole

Aplicația noastră stochează imagini în directorul `wwwroot/images/`. ASP.NET Core servește automat fișierele din `wwwroot/` prin middleware-ul `UseStaticFiles()`, configurat în `Program.cs`:

```
app.UseStaticFiles();
```

Un fișier la calea `wwwroot/images/articol.jpg` devine accesibil la URL-ul `/images/articol.jpg`.

Adăugarea proprietății `ImagePath` în model

Adăugați proprietatea în clasa `Article`:

```
public string? ImagePath { get; set; }
```

Tipul `string?` (nullable) înseamnă că imaginea este opțională, articolele fără imagine au `ImagePath = null`.

Migrare

Creați și aplicați o migrare pentru noua coloană:

```
Add-Migration AddImagePath
Update-Database
```

La repornirea aplicației, seed data va popula automat câmpul `ImagePath` pentru articolele care au imagine asociată.

Afișarea imaginii în pagina Details

În `Details.cshtml`, adăugați blocul de imagine după titlu:

```
@if (Model.Article.ImagePath != null)
{
    <img src="@Model.Article.ImagePath"
         alt="@Model.Article.Title"
         class="img-fluid rounded mb-3"
         style="max-height: 400px; object-fit: cover; width: 100%;" />
}
```

`src="@Model.Article.ImagePath"` generează un URL de tipul `/images/chuck-norris.png`. ASP.NET Core servește fișierul din `wwwroot/images/chuck-norris.png`.

Blocul `@if` asigură că tag-ul `<img>` apare doar pentru articolele care au o imagine setată.

Thumbnail în lista de articole

În `Index.cshtml`, adăugați o coloană de thumbnail. În `<thead>`:

```
<tr>
    <th></th>
    <th>Title</th>
    ...
</tr>
```

În `<tbody>`, în fiecare `<tr>`:

```
<tr>
    <td>
        @if (article.ImagePath != null)
        {
            <img src="@article.ImagePath" alt="@article.Title"
                 style="width: 80px; height: 60px; object-fit: cover;
border-radius: 4px;" />
        }
    </td>
    <td>@article.Title</td>
    ...
</tr>
```

Upload imagine la crearea unui articol

Formularul din `Create.cshtml` trebuie să suporte trimiterea de fișiere. Modificați tag-ul `<form>`:

```
<form method="post" enctype="multipart/form-data">
```

Fără `enctype="multipart/form-data"`, browser-ul nu trimite conținutul fișierului, trimite doar numele.

Adăugați câmpul de upload înainte de butonul Submit:

```
<div>
    <label class="form-label">Imagine articol (opțional)</label>
    <input type="file" name="Upload" class="form-control" accept="image/*"
/>
</div>
```

Procesarea fișierului în Create.cshtml.cs

`IFormFile` reprezintă un fișier trimis prin formular. `IWebHostEnvironment` oferă calea fizică spre `wwwroot/`.

```
using Lab05.Data;
using Lab05.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace Lab05.Pages.Articles;

public class CreateModel : PageModel
{
    private readonly AppDbContext _context;
    private readonly IWebHostEnvironment _env;

    public CreateModel(AppDbContext context, IWebHostEnvironment env)
    {
        _context = context;
        _env = env;
    }

    [BindProperty]
    public Article Article { get; set; } = default!;

    [BindProperty]
    public IFormFile? Upload { get; set; }

    public List<SelectListItem> Categories { get; set; } = [];

    private void LoadCategories()
    {
        Categories = _context.Categories
            .Select(c => new SelectListItem
            {
                Value = c.Id.ToString(),
                Text = c.Name
            })
            .ToList();
    }

    public IActionResult OnGet()
    {
```

```
        LoadCategories();
        return Page();
    }

    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            LoadCategories();
            return Page();
        }

        if (Upload != null)
        {
            var fileName = Path.GetFileName(Upload.FileName);
            var savePath = Path.Combine(_env.WebRootPath, "images",
fileName);
            using var stream = System.IO.File.Create(savePath);
            await Upload.CopyToAsync(stream);
            Article.ImagePath = $"/images/{fileName}";
        }

        Article.PublishedAt = DateTime.Now;
        _context.Articles.Add(Article);
        await _context.SaveChangesAsync();

        return RedirectToPage("/Index");
    }
}
```

Ce se întâmplă pas cu pas:

1. `Upload != null` — verificăm că utilizatorul a selectat un fișier
2. `Path.GetFileName()` — extrage doar numele fișierului (fără cale)
3. `_env.WebRootPath` — calea fizică spre `wwwroot/` (ex: `C:\...\wwwroot`)
4. `Path.Combine(...)` — construiește calea completă de salvare
5. `System.IO.File.Create()` + `CopyToAsync()` — scrie fișierul pe disc
6. `Article.ImagePath = $"/images/{fileName}"` — salvează URL-ul relativ în baza de date

Metoda devine `async` (returnează `Task<IActionResult>`) pentru a folosi `await` la operațiile I/O.

Fluxul CRUD complet

Operație	Pagina	Metoda HTTP	Handler	EF Core
Listare	Index	GET	`OnGet`	`ToList()`
Detalii	Details	GET	`OnGet(int id)`	`FirstOrDefault()`
Creare	Create	GET + POST	`OnGet` + `OnPost`	`Add()` + `SaveChanges()`
Editare	Edit	GET + POST	`OnGet(int id)` + `OnPost`	`Update()` + `SaveChanges()`
Ștergere	Delete	GET + POST	`OnGet(int id)` + `OnPost(int id)`	`Remove()` + `SaveChanges()`

Referință — Tag Helpers utilizate

Tag Helper	Descriere
`asp-page`	Link către o Razor Page
`asp-route-{param}`	Parametru de rută
`asp-for`	Binding input ↔ proprietate model
`asp-items`	Populează `<select>` cu opțiuni
`asp-validation-for`	Mesaj de eroare per câmp
`asp-validation-summary`	Sumar erori de validare

Referință — Metode utilizate

Metodă	Rol
`Find(id)`	Caută o entitate după cheie primară
`FirstOrDefault()`	Primul element sau `null`
`Add(entity)`	Marchează entitatea pentru inserare
`Update(entity)`	Marchează entitatea ca modificată
`Remove(entity)`	Marchează entitatea pentru ștergere
`Skip(n)`	Sare peste primele n elemente
`Take(n)`	Selectează următoarele n elemente

Documentație Tag Helpers: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro
Documentație Razor Pages: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/

Exerciții

1. Deschideți tabelul Articles în SQL Server Object Explorer (View → SQL Server Object Explorer → (localdb)\MSSQLLocalDB → Databases → ... → Tables → dbo.Articles → View Data) și completați manual coloana ImagePath pentru articolele care au o imagine corespunzătoare în wwwroot/images/.
   O imagine salvată la wwwroot/images/foto.jpg se accesează la URL-ul /images/foto.jpg.
   Alternativă (dacă nu lucrați în Visual Studio): actualizați câmpul ImagePath direct în SeedData.cs, resetați baza de date și rulați din nou aplicația.

2. Adăugați în pagina Index o coloană suplimentară care afișează primele 50 de caractere din `Continut` (sau tot conținutul dacă e mai scurt).

3. Stilizați pagina Index. Fiecare știre sa fie afișată ca un card, nu ca rând în tabel:
   Titlu (bold)
   Imagine
   Primele 100 caractere din conținut
   Categorie — Data publicării (italic)
   Linkuri: Detalii | Editează | Șterge

4. Adăugați filtrare pe pagina Index. Adăugați un dropdown cu categoriile și un buton „Filtrează”. Doar știrile din categoria selectată sunt afișate. Dacă nu e selectată nicio categorie, se afișează toate.
   Indicații: folosiți un query string `/?categoryId=2` și un parametru opțional `int? categoryId` în `OnGet`.

5. Modificați pagina Delete astfel încât ștergerea să funcționeze doar dacă utilizatorul confirmă prin tastarea titlului știrii. Adăugați un `<input>` în formularul de confirmare și validați pe server că textul introdus coincide cu `Article.Title`.