

Dezvoltarea Aplicațiilor Web - [ASP.NET](#) Core MVC

INDEX

A

Concept	Locație
Acțiuni (Actions)	Curs 4 - Acțiuni
ActionName (selector)	Curs 4 - Selectori
ActionResult / IActionResult	Curs 4 - Tipul de returnare
Add-Migration	Curs 5 - Migrații
appsettings.json	Curs 3 - Structura Proiectului
Array-uri	Curs 2 - Variabile și Tipuri
ASP.NET Core	Curs 1 - Framework-ul .NET
ASP.NET Core Identity	Curs 9 - Sistemul de Autentificare
asp-validation-for	Curs 8 - Validare în View
asp-validation-summary	Curs 8 - Validare în View
Atribute de validare	Curs 8 - Data Annotations
Authorize (atribut)	Curs 9 - Atributul Authorize

B

Concept	Locație
Base Class Library (BCL)	Curs 1 - Framework-ul .NET
Baze de date	Curs 6 - Noțiuni Generale
Bookmark (many-to-many)	Curs 10 - Relația many-to-many
Bootstrap	Curs 3 - wwwroot

C

Concept	Locație
C# (introducere)	Curs 2 - Introducere în C#
C++ vs C#	Curs 2 - Comparație
Căutare (funcționalitate)	Curs 10 - Funcționalitatea de căutare
Cheie Externă (FK)	Curs 6 - Concepte Cheie
Cheie Primară (PK)	Curs 6 - Concepte Cheie

Concept	Locație
Cheie Primară Compusă	Curs 6 - Concepte Cheie
Ciclul de viață pagină Web	Curs 1 - Ciclul de viață
Clase și Obiecte	Curs 2 - Concepte Avansate
CLR (Common Language Runtime)	Curs 1 - CLR
Code-first	Curs 5 - Entity Framework Core
Connection String	Curs 5 - Dependency Injection
Constructor	Curs 2 - Clase și Obiecte
ActionResult	Curs 4 - IActionResult
Controller	Curs 4 - Ce este Controller-ul
Conversii de tip	Curs 2 - Conversii
CRUD	Curs 5 - Operații CRUD
Cross-platform	Curs 1 - ASP.NET Core

D

Concept	Locație
Data Annotations	Curs 8 - Validare
Database-first	Curs 5 - Entity Framework Core
DbContext	Curs 5 - Dependency Injection
DbSet	Curs 7 - DbSet și EF Core
Dependency Injection	Curs 5 - DI
Design web	Curs 10 - Design și UX
Diagrama Conceptuală	Curs 6 - Diagrama Conceptuală
Diagrama E/R	Curs 6 - Diagrama E/R

E

Concept	Locație
Eager Loading (Include)	Curs 6 - Interogări LINQ
Editor de text (Summernote)	Curs 10 - Summernote
EmailAddress (validare)	Curs 8 - Atribute validare
Endpoint	Curs 11 - REST API
Entity Framework Core	Curs 5 - EF Core
Entitate	Curs 6 - Concepte Cheie

F

Concept	Locație
FileResult	Curs 4 - IActionResult
Fișiere în baza de date	Curs 9 - Stocare fișiere
Fluent API	Curs 10 - Configurare Model
Foreign Key	Curs 6 - Concepte Cheie
[FromBody], [FromForm], [FromQuery]	Curs 8 - Atribute sursa datelor

G

Concept	Locație
Garbage Collector	Curs 1 - CLR
GET (verb HTTP)	Curs 4 - Action Verbs
Google AI (Gemini)	Curs 12 - Switching to Google AI
Group By	Curs 6 - Interogări LINQ

H

Concept	Locație
Helpere pentru View	Curs 7 - Helpere
HtmlSanitizer	Curs 10 - Securitate XSS
HTML Helpers vs Tag Helpers	Curs 7 - Categoriile de Helpere
HTTP (protocol)	Curs 1 - Aplicație Web
HttpGet / HttpPost	Curs 4 - Action Verbs

I

Concept	Locație
IActionResult	Curs 4 - Tipul de returnare
Identity Framework	Curs 9 - Sistemul de Autentificare
IIS (Internet Information Server)	Curs 1 - Ciclul de viață
Include (Eager Loading)	Curs 6 - Interogări LINQ
Individual Accounts	Curs 6 - ASP.NET Core Identity
Intermediate Language (IL)	Curs 1 - CLR
IQueryable	Curs 7 - DbSet
IValidatableObject	Curs 8 - Validări Custom

J

Concept	Locație
JIT (Just In Time)	Curs 1 - CLR
Join-uri	Curs 6 - Interogări LINQ
jQuery Validation	Curs 8 - Validare Client
JSON	Curs 11 - REST API
JsonIgnore	Curs 11 - ArticlesController
JsonResult	Curs 4 - IActionResult
JWT (JSON Web Tokens)	Curs 11 - Gestionarea Utilizatorilor

L

Concept	Locație
Layout View	Curs 8 - View-uri Partajate
Lazy Loading	Curs 6 - Lazy Loading
Limbaj compilat vs interpretat	Curs 1 - Compilare
LINQ	Curs 5 - EF Core

M

Concept	Locație
Many-to-many (relație)	Curs 10 - Relația many-to-many
MapControllerRoute	Curs 3 - Sistemul de Rutare
MaxLength / MinLength	Curs 8 - Atribute validare
Metoda Main	Curs 2 - Structura program
Middleware	Curs 3 - Pipeline Middleware
Migrații	Curs 5 - Migrații
Model	Curs 5 - Ce este Modelul
Model Binding	Curs 5 - CRUD
ModelState	Curs 8 - Validare Controller
MVC (Model-View-Controller)	Curs 3 - Arhitectura MVC

N

Concept	Locație
Namespace	Curs 2 - Structura program
Naming Conventions	Curs 2 - Convenții de nume
NonAction	Curs 4 - Selectorii

Concept	Locație
[NotMapped]	Curs 7 - Dropdown Categorii
Nullable	Curs 2 - Concepte Avansate
NuGet Package Manager	Curs 5 - Instalarea EF Core

O

Concept	Locație
One-to-many (relație)	Curs 6 - Relații
One-to-one (relație)	Curs 6 - Relații
OpenAI API	Curs 12 - Crearea unui cont OpenAI
ORM (Object Relational Mapper)	Curs 5 - EF Core

P

Concept	Locație
Paginare	Curs 10 - Afișarea paginată
Partial View	Curs 8 - Partial View
PascalCase / camelCase	Curs 2 - Convenții
POST (verb HTTP)	Curs 4 - Action Verbs
Primary Key	Curs 6 - Concepte Cheie
Program.cs	Curs 3 - Structura Proiectului

R

Concept	Locație
Range (validare)	Curs 8 - Atribute validare
Razor	Curs 7 - Razor în ASP.NET
Redirect / RedirectToAction	Curs 4 - Redirecționarea
RegularExpression (validare)	Curs 8 - Atribute validare
Relații	Curs 6 - Concepte Cheie
RenderBody()	Curs 8 - Layout View
Required (validare)	Curs 8 - Atribute validare
REST API	Curs 11 - Introducere REST API
Roluri	Curs 9 - Roluri și Utilizatori
Rutare	Curs 3 - Sistemul de Rutare

S

Concept	Locație
SaveChanges()	Curs 5 - CRUD
Scaffolding	Curs 9 - Identity Framework
SeedData	Curs 9 - Popularea BD
SelectListItem	Curs 7 - Dropdown Categorii
Sentiment Analysis	Curs 12 - Implementare
SGBD/DBMS	Curs 6 - Noțiuni Generale
Skip() / Take()	Curs 10 - Paginare
SQL Server	Curs 5 - Conexiune BD
StatusCodeResult	Curs 4 - IActionResult
StringLength (validare)	Curs 8 - Atribute validate
Summernote	Curs 10 - Editor de Text
Swagger	Curs 11 - Accesarea Swagger

T

Concept	Locație
Tabel asociativ	Curs 6 - Diagrama Conceptuală
Tag Helpers	Curs 7 - Categorii de Helpere
TempData	Curs 7 - Trimiterea datelor
This	Curs 2 - Clase și Obiecte
Tipuri de date	Curs 2 - Variabile

U

Concept	Locație
Update-Database	Curs 5 - Migrații
UseAuthorization()	Curs 3 - Middleware
UseRouting()	Curs 3 - Middleware
User Experience (UX)	Curs 10 - Design și UX
userManager	Curs 9 - Implementare CRUD
using (directiva)	Curs 2 - Structura program

V

Concept	Locație
ValidationAttribute	Curs 8 - Atribute Personalizate

Concept	Locație
View	Curs 7 - View
ViewBag	Curs 7 - Trimiterea datelor
ViewData	Curs 7 - Trimiterea datelor
ViewResult	Curs 4 - IActionResult
virtual (proprietăți)	Curs 6 - Lazy Loading

W

Concept	Locație
wwwroot	Curs 3 - Structura Proiectului
WYSIWYG	Curs 10 - Summernote

X

Concept	Locație
XSS (Cross-Site Scripting)	Curs 10 - Securitate XSS

CURS 1 - Introducere în [ASP.NET](#) Core MVC

Cuprins

- Ce este o aplicație Web
- Arhitectura Web
- Avantajele aplicațiilor Web
- Introducere în [ASP.NET](#)
- Ce este CLR - Common Language Runtime
- Framework-ul .NET
- [ASP.NET](#) Core
- Introducere în C#
- Limbaj compilat vs limbaj interpretat
- Ciclul de viață al unei pagini Web

1. Ce este o aplicație Web?

O aplicație web rulează într-o arhitectură **Client-Server** și se bazează pe următoarele elemente:

Element	Descriere
Protocoale	HTTP (HyperText Transfer Protocol) și TCP/IP
Componente	Un browser web (client) și un server web
Tehnologii	PHP, ASP.NET , Python, HTML, CSS, JavaScript etc.

Element	Descriere
Interacțiune	Permit interacțiunea directă cu utilizatorul printr-o interfață grafică accesibilă de pe orice dispozitiv conectat la internet
Utilizare	E-commerce, sisteme educaționale, management și rețele sociale

Arhitectura Web (Diagrama de proces)

- Client (PC/Mac/Unix + Browser):** Trimite un **Request** (ex: `http://www.xo.com/default.aspx`)
 - Rețea:** Transportă datele prin HTTP și TCP/IP
 - Server Web:** Primește cererea și poate interacționa cu o **Bază de date** (SQL Request/Response)
 - Răspuns:** Serverul trimite un **Response** sub formă de cod HTML către client
-

2. Avantajele aplicațiilor Web

Avantaj	Descriere
Independența de SO	Pot fi accesate de pe Windows, macOS, Linux sau dispozitive mobile
Fără instalare	Rulează direct în browser, eliminând problemele de compatibilitate și spațiu
Actualizare facilă	Modificările se fac doar pe server și se propagă automat la toți utilizatorii
Accesibilitate	Pot fi utilizate de oriunde există o conexiune la internet
Costuri reduse	Mentenanță și distribuție eficientă
Scalabilitate	Adaptare ușoară pentru mulți utilizatori și integrare facilă cu API-uri
Securitate centralizată	Datele sunt gestionate pe server, reducând riscurile locale

3. Introducere în ASP.NET și CLR

ASP.NET este un framework open-source creat de Microsoft pentru dezvoltarea aplicațiilor și serviciilor web.

- Integrează HTML, CSS și JavaScript
- Folosește **sintaxa Razor** (C# + HTML) pentru pagini dinamice
- Include librării pentru autentificare, baze de date și lucrul cu fișiere

CLR (Common Language Runtime)

Este mediul de execuție care se ocupă de programele .NET:

Caracteristică	Descriere
Compilare IL	Codul C# este transformat inițial în Intermediate Language (IL) , salvat în fișiere <code>.exe</code> sau <code>.dll</code>
Compilator JIT	Transformă codul IL în cod nativ (instrucțiuni pentru procesor) chiar în momentul rulării
Servicii CLR	Gestionarea memoriei (Garbage Collector), tratarea excepțiilor și securitatea codului

4. Framework-ul .NET și ASP.NET Core

Framework-ul .NET este compus din:

- 1. **CLR:** Mediul de execuție
- 2. **Base Class Library (BCL):** Bibliotecă de clase pentru interfețe, baze de date și date

ASP.NET Core este o versiune complet rescrisă:

Caracteristică	Descriere
Cross-platform	Rulează pe Windows, macOS și Linux
Performanță	Design modular, optimizat pentru cloud (Azure, AWS)
Tehnologii suportate	MVC, Razor Pages, Blazor (C# în browser) și Minimal APIs
Dependency Injection	Sistem integrat pentru o arhitectură curată

5. Limbajul C# și Compilarea

C# este un limbaj orientat pe obiecte, derivat din C++ și conceput ca un concurent pentru Java.

Tip Limbaj	Caracteristici	Exemple
Compilat	Tradus complet în cod executabil înainte de rulare. Performanță ridicată.	C, C++, Rust
Interpretat	Fiecare linie este tradusă și executată la momentul rulării. Feedback rapid.	PHP, Python, Ruby

C# ca limbaj hibrid: Este considerat compilat (în IL), dar folosește și compilarea JIT la execuție, ceea ce îi oferă portabilitate și optimizare hardware.

6. Ciclul de viață al unei pagini Web ASP.NET

Procesul prin care trece o pagină pe serverul IIS (Internet Information Server):

Etapă	Descriere
1. Page Request	Clientul cere pagina
2. Start	Se încarcă obiectele Request/Response; se determină tipul cererii (GET/POST)
3. Initialization	Se inițializează controalele și se aplică Master Page
4. Load	Se încarcă valorile în controale (dacă este postback)
5. Postback Events	Se execută codul pentru evenimente (ex: apăsarea unui buton) și validarea datelor
6. Rendering	Se construiește codul HTML final
7. Unload	Se eliberează resursele de pe server

CURS 2 - Introducere în C#

Cuprins

- Introducere în C#
- C++ vs C#

- Structura unui program
- Variabile și Tipuri de date
- Conversii de tip
- Nullable
- Instrucțiuni de control
- Array-uri
- Clase și Obiecte
- Constructor
- This
- Convenții de nume

Introducere în C#

C# este un limbaj de programare popular dezvoltat de **Microsoft** care rulează pe arhitectura **.NET**. Este un limbaj de **nivel înalt**, orientat pe obiecte, utilizat pentru aplicații web, mobile, desktop, jocuri, VR și servicii web.

C++ vs C#

Aspect	C++	C#
Sintaxa	Complexă	Simplă și intuitivă
Nivel	Mediu (combină low-level cu high-level)	Înalt (abstract, apropiat de limbajul uman)
Memorie	Gestionată manual (destructori manuali)	Gestionată automat prin Garbage Collector
Compilare	Direct în cod mașină	În limbaj intermediar (IL) prin CLR, apoi JIT în cod mașină
Moștenire	Suportă moștenire multiplă	Nu suportă moștenire multiplă

Structura unui program

Pentru dezvoltarea în C# se utilizează **Visual Studio 2022**.

Crearea unui proiect nou:

1. Selectați "Create a new project"
2. Alegeți template-ul **Console App** (pentru Windows, Linux, macOS)
3. Configurați numele proiectului (ex: `Lab2App`) și locația
4. Scrieți codul în `Program.cs`

Elemente Cheie:

Element	Descriere
Namespace	O colecție de clase și funcții utilizată pentru organizarea codului și prevenirea conflictelor de nume
Directiva <code>using</code>	Permite importul altor namespace-uri (ex: <code>using System</code> oferă acces la clasa <code>Console</code>)
Metoda <code>Main</code>	Este punctul de intrare în orice program C#; execuția începe întotdeauna aici

Variabile și Tipuri de date

C# este **case sensitive** și fiecare instrucțiune se încheie cu punct și virgulă (;).

Tipuri de date comune:

Tip	Descriere
int	Numere întregi
double	Numere cu virgulă
char	Un singur caracter
string	Text
bool	Valori true sau false
object	Clasa de bază pentru toate tipurile

Conversii de tip

Tip conversie	Descriere
Implicite	Realizate automat de compilator (ex: de la int la double)
Explicite (Casting)	Necesare când se trece de la un tip mai mare la unul mai mic (ex: int nInt = (int)nDouble;)
Parse()	Utilizat pentru tipuri incompatibile, cum ar fi transformarea unui string în int
Clasa Convert	Metode precum ToBoolean() , ToDouble() , ToString()

Concepte Avansate

Nullable

Permite tipurilor de date să stocheze valoarea null .

Sintaxa: <tip>? <nume> = null; (ex: int? num1 = null;)

Clase și Obiecte

Concept	Descriere
Clasa	O structură de date care conține proprietăți și metode; se comportă ca un tip de date
Obiectul	O instanță a unei clase; la creare, se alocă memorie specifică
Constructor	Trebuie să aibă același nume ca și clasa, nu returnează valori și poate fi supraîncărcat
This	Referință la instanța curentă a clasei

Convenții de nume (Naming Conventions)

Convenție	Utilizare
PascalCase	Folosit pentru Clase, Constructori și Metode
camelCase	Folosit pentru Variabile și Argumentele metodelor

Laborator 2 - Exerciții

Exercițiul 1: Crearea unui proiect

Să se creeze un nou proiect conform instrucțiunilor din Cursul 2.

Exercițiul 2: Implementare curs

Să se implementeze exemplele studiate în cadrul cursului 2.

Exercițiul 3: Problema Palindrom

Să se creeze un nou proiect de tip **Console App**, numit **Laborator2**, pentru a rezolva următoarea cerință:

- Se citește un număr întreg de la tastatură
- Să se verifice dacă numărul este palindrom și să se afișeze un mesaj corespunzător
- Se vor trata cazurile particulare (ex: toate numerele formate dintr-o singură cifră sunt palindrom)
- Numărul se citește de la tastatură utilizând `Console.ReadLine()`

Varianta 1: Implementare directă în metoda `Main` :

```
class Program
{
    static void Main(string[] args)
    {
        // Implementare aici
    }
}
```

Varianta 2: Crearea unei metode dedicate numită `Palindrom` :

```
class Program
{
    void Palindrom(int n)
    {
        // Implementare logică
    }

    static void Main(string[] args)
    {
        // Apelare metodă
    }
}
```

Exercițiul 4: Problema Paritate Alternativă

Se citește un vector cu n elemente numere naturale.

- Verificați dacă oricare două elemente alăturate alternează ca paritate (au paritate diferită)
- Să se afișeze un mesaj corespunzător
- **Exemplu:** `[8, 73, 2]` -> Da

CURS 3 - Arhitectura MVC și Rutare

Acest curs prezintă arhitectura **MVC (Model-View-Controller)**, structura unui proiect în **ASP.NET Core 9.0** și funcționarea **sistemului de rutare**.

1. Arhitectura MVC

Model-View-Controller este un model arhitectural care izolează logica aplicației de interfața cu utilizatorul, facilitând modificarea aspectului vizual sau a regulilor de business fără a afecta celelalte niveluri.

Componentă	Responsabilitate
Model	Gestionarea și manipularea datelor; reprezintă nucleul aplicației și realizează legătura cu baza de date
Controller	Controlează accesul la aplicație, procesează request-urile HTTP, citește datele de la utilizator și comunică cu Modelul și View-ul
View	Interfața cu utilizatorul; afișează datele și înregistrările din baza de date prin intermediul browser-ului

Notă: În Controller, fiecare metodă publică este considerată un **Action**. Acestea trebuie să fie publice pentru ca framework-ul MVC să le poată accesa și invoca în urma unei cereri HTTP.

2. Structura Proiectului (Sistemul de fișiere)

Un proiect MVC nou creat în Visual Studio 2022 include următoarele elemente esențiale:

Element	Descriere
wwwroot	Conține fișiere statice (biblioteci precum Bootstrap, imagini, scripturi CSS/JS)
Controllers	Include clasele C# care gestionează cererile; numele acestora trebuie să se termine obligatoriu în <code>Controller</code>
Models	Folderul care stochează modelele de date ale aplicației
Views	Conține interfețele utilizator (.cshtml)
appsettings.json	Fișier de configurare (format JSON) pentru detalii precum conexiunea la baza de date sau setările de logging
Program.cs	Punctul de pornire al aplicației; aici se configurează serviciile, serverul web (IIS) și pipeline-ul middleware

3. Sistemul de Rutare

Rutarea elimină necesitatea mapării URL-urilor cu fișiere fizice, folosind URL-uri logice pentru a invoca metode din Controllere.

Formatul de bază al rutelor:

```
{NumeController}/{NumeActiune}/{Parametri}
```

Definirea în Program.cs:

Rutele se configurează folosind metoda `MapControllerRoute()`. În [ASP.NET Core 9.0](#), aceasta se integrează cu noul sistem unificat de **Static Assets** prin `.WithStaticAssets()`.

Caracteristică	Descriere
Ruta Default	Identifică în mod implicit <code>HomeController</code> și acțiunea <code>Index</code>
Parametri Opționali	Un parametru urmat de semnul întrebării (ex: <code>{id?}</code>) indică faptul că acesta poate fi omis din URL
Constrângeri	Se pot impune reguli asupra parametrilor (tip de date, lungime, interval sau expresii regulate)

Exemple de constrângeri:

- Range: `pattern: "Verify/{id:range(10,100)}"`
- Regex: `pattern: "VerifyDigits/{id:regex(\\d+)}"`

4. Pipeline Middleware

Reprezintă un lanț de componente software prin care trece o cerere HTTP. Fiecare componentă (Middleware) poate procesa cererea sau decide trimiterea ei mai departe.

Exemple de middleware în pipeline:

Middleware	Funcție
UseRouting()	Activează mecanismul de potrivire a cererii cu rutele definite
UseAuthorization()	Verifică permisiunile utilizatorului
MapControllerRoute()	Definește efectiv rutele și finalizează cererea

Diferențe .NET 8 vs .NET 9

În **.NET 8**, fișierele statice erau servite separat prin `app.UseStaticFiles()`. În **.NET 9**, sistemul este unificat pentru o performanță mai mare, oferind optimizare automată și cache inteligent prin utilizarea hash-urilor în URL.

Laborator 3 - Exerciții practice

Exercițiul 1: Implementare ExamplesController

Se cere crearea unui proiect nou și adăugarea unui Controller numit `ExamplesController` cu următoarele acțiuni și rute:

1. Rută Concatenare:

- **URL:** `/concatenare/{param1}/{param2}`
- **Funcționalitate:** Primește doi parametri de tip string și afișează valorile lor concatenate

2. Rută Produs:

- **URL:** `/produs/param1/param2`
- **Parametri:** Două numere întregi; al doilea este opțional
- **Funcționalitate:** Afișează produsul celor două numere sau mesajul "Introduceți ambele valori" dacă lipsește parametrul opțional

3. Rută Operație:

- **URL:** `/operatie/param1/param2/op`
- **Parametri:** Doi întregi și un string (`op`) reprezentând operația (plus, minus, ori, div)
- **Funcționalitate:** Afișează parametrii, operația și rezultatul. Dacă un parametru lipsește, afișează "Introduceți parametrul 1/2/3"

Exercițiul 2: Operații C.R.U.D. pentru Studenți

Se implementează `StudentsController` pentru a gestiona resursa **Student** prin rute specifice:

Acțiune	Rută (Name)	URL Pattern	Mesaj/Funcționalitate
Index (v1)	StudentsIndex	<code>/Students/Index</code>	Afișarea tuturor studenților

Acțiune	Rută (Name)	URL Pattern	Mesaj/Funcționalitate
Index (v2)	StudentsAll	students/all	Variantă alternativă pentru afișare
Show	StudentShow	students/{id}	"Afișare student cu ID: {id}" (id opțional)
Create	StudentCreate	students/new	"Creare student în baza de date"
Edit	StudentEdit	students/{id}	"Editare student cu ID: {id}" (id opțional)
Delete	StudentDelete	students/{id}	"Ștergere student cu ID: {id}" (id opțional)

Observație: Utilizarea aceluiași pattern URL (students/{id}) pentru Show, Edit și Delete poate crea ambiguitate dacă nu sunt diferențiate prin metode HTTP sau constrângeri suplimentare.

Exercițiul 3: Căutare cu Constrângeri (SearchController)

Implementarea unei pagini de căutare pentru numere de telefon cu validări la nivel de Controller și de rută.

Constrângeri și Validări:

- **Format Rută:** search/telefon/{telef:regex() }
- **Reguli:** Lungime exactă de 10 cifre, începând obligatoriu cu cifra 0

Logica din Controller:

- Dacă parametrul lipsește: "Introduceți numărul de telefon"
- Dacă lungimea < 10: "Numarul de telefon nu are suficiente cifre"
- Dacă lungimea > 10: "Numarul de telefon are prea multe cifre"
- Valid format: "Cautare pentru nr de telefon: {telef}"

Exemplu Configurare Rută (regex):

```
app.MapControllerRoute(
    name: "NrTelefon",
    pattern: "search/telefon/{telef:regex(^0\\d{9}$)?}",
    defaults: new { controller = "Search", action = "NumarTelefon" });
```

Expresia regulată ^0\\d{9}\$ asigură începerea cu 0 urmat de încă 9 cifre.

CURS 4 - Controller în ASP.NET Core MVC

Cursul se concentrează pe rolul, structura și funcționalitățile **Controller-ului** în arhitectura MVC.

1. Ce este Controller-ul?

În arhitectura MVC, **Controller-ul** este componenta responsabilă pentru procesarea URL-urilor aplicației.

- Este o clasă care derivă din clasa de bază Microsoft.AspNetCore.Mvc
- Conține metode publice numite **Acțiuni (Actions)**

Roluri principale:

- Procesează request-urile venite de la browser
- Apelează modelele și procesează datele

- Trimite răspunsul final către utilizator prin browser

Caracteristică	Descriere
Convenții de denumire	Numele trebuie să se termine obligatoriu cu sufixul <code>Controller</code> (ex: <code>HomeController</code>)
Locație	Controller-ele se adaugă în folderul <code>Controllers</code> al proiectului

2. Crearea și Adăugarea unui Controller

Pentru a începe un proiect nou, se utilizează șablonul **ASP.NET Core Web App (Model-View-Controller)**.

Pașii pentru adăugarea unui Controller nou:

1. **Click dreapta** pe folderul `Controllers` -> **Add** -> **Controller**
2. Din fereastra de dialog, se selectează **MVC Controller - Empty**
3. Se alege **ASP.NET Core** din meniul stâng, apoi din nou **MVC Controller - Empty**
4. Se introduce numele (ex: `StudentsController.cs`), asigurându-vă că păstrați sufixul `Controller`

3. Acțiuni (Actions)

Acțiunile sunt metodele publice dintr-un Controller care răspund cererilor HTTP.

Structura unei metode

```
public class StudentsController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

Caracteristică	Descriere
Publicitate	Metodele trebuie să fie <code>public</code> pentru a fi accesate de framework. Metodele <code>private</code> sunt folosite doar intern
Supraîncărcare	Acțiunile pot fi supraîncărcate doar dacă au parametri diferiți (număr sau tip); semnătura trebuie să fie distinctă

Tipul de returnare: `IActionResult`

- Este o interfață abstractă ce permite diferite tipuri de răspunsuri HTTP
- Clasa `ActionResult` : Implementează `IActionResult` și este clasa de bază pentru toate rezultatele acțiunilor

Tip Rezultat	Metodă Controller	Descriere
<code>ViewResult</code>	<code>View()</code>	Returnează conținut HTML (fișier <code>.cshtml</code>)
<code>EmptyResult</code>	-	Returnează un răspuns gol (Status 200)
<code>ContentResult</code>	<code>Content()</code>	Returnează text simplu
<code>JsonResult</code>	<code>Json()</code>	Returnează date serializate în format JSON
<code>RedirectResult</code>	<code>Redirect()</code>	Redirecționare către un URL (string)
<code>FileResult</code>	<code>File()</code>	Manipulare și descărcare de fișiere

Tip Rezultat	Metodă Controller	Descriere
StatusCodeResult	StatusCode()	Returnează coduri de status (400, 401, 404 etc.)

4. Selectorii și Verbe HTTP

Selectorii sunt atribute care ajută sistemul de rutare să aleagă acțiunea corectă.

Selector	Descriere
ActionName	Permite definirea unui nume de rută diferit de numele metodei (ex: <code>[ActionName("afisare")]</code>)
NonAction	Indică faptul că o metodă publică nu este o acțiune și nu poate fi accesată prin URL

Action Verbs: Specifică verbul HTTP la care răspunde metoda. Verbul implicit este **GET**.

Verb	Utilizare
GET	Accesare resursă/pagină
POST	Creare resursă sau trimitere date prin formular
PUT/PATCH/DELETE	În ASP.NET Core MVC, pentru editare și ștergere se folosește adesea tot <code>[HttpPost]</code>

5. Redirecționarea (Redirect)

Există mai multe metode pentru a trimite utilizatorul către o altă pagină:

Metodă	Descriere
Redirect	Redirecționare către un URL specific (HTTP 302)
RedirectToRoute	Redirecționare folosind numele rutei definit în <code>Program.cs</code>
RedirectToAction	Redirecționare către o altă metodă (din același Controller sau unul diferit)
Redirect Permanent (HTTP 301)	<code>RedirectPermanent</code> , <code>RedirectToRoutePermanent</code> . Răspunsul este stocat în cache-ul browserului. Se folosește pentru schimbări definitive de URL sau optimizare SEO

Laborator 4 - Exerciții

Exercițiul 1: Crearea proiectului și a modelului Article

Se va crea un proiect numit **Laborator4** și se va adăuga un Controller numit `ArticlesController`.

1. Definirea Modelului

În folderul `Models`, se adaugă clasa `Article.cs` cu următoarea structură:

```
public class Article
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime Date { get; set; }
}
```

2. Metoda NonAction GetArticles()

Deoarece nu se utilizează o bază de date în acest laborator, se va crea o metodă marcată cu atributul `[NonAction]` pentru a genera date de test.

- **Rol:** Returnează un array de obiecte `Article`
- **Implementare:** Se instanțiază un array de 3 articole, populând câmpurile `Id`, `Title`, `Content` și `Date` într-un ciclu repetitiv

Operații C.R.U.D. și Vizualizare

3. Metoda Index

- **Controller:** Apelează `GetArticles()` și transmite datele către View folosind `ViewBag.Articles`
- **View (`Index.cshtml`):** Trebuie să afișeze lista tuturor articolelor și să ofere link-uri pentru vizualizarea și editarea fiecăruia

4. Metoda Show (Vizualizare)

- **Funcționalitate:** Afișează detaliile unui singur articol bazat pe `id`
- **Gestionarea erorilor:** Dacă articolul nu este găsit, se returnează un View numit `Error` care afișează mesajul "Articolul căutat nu poate fi găsit" și detalii despre excepție
- **Rutare:** Se va configura o rută personalizată `ArticlesShow` pentru a accesa metoda prin `/articole/show/{id}` în loc de `/Articles/Show/{id}`

Metodele pentru Adăugare, Editare și Ștergere

5. New (Adăugare)

Se folosesc două metode în Controller:

- `[HttpGet] New()` : Afișează formularul de creare
- `[HttpPost] New(Article article)` : Primește datele de la server. Momentan, returnează un View (`NewPostMethod`) care confirmă succesul operațiunii printr-un mesaj

6. Edit (Editare)

- `[HttpGet] Edit(int? id)` : Încarcă datele articolului existent într-un formular
- `[HttpPost] Edit(Article article)` : Trimite modificările către server. Poate redirecționa către `Index` sau afișa un mesaj de succes

7. Delete (Ștergere)

- Se implementează în `ArticlesController` folosind verbul `[HttpPost]`
 - Butonul de ștergere este inserat într-un formular în pagina `Show`
 - După procesare, se returnează un mesaj de confirmare: "Articolul a fost sters din baza de date!"
-

Exerciții avansate și Selectori

- **Redenumire:** Redenumiți metoda `Index` în `Listare` folosind selectori (`ActionName`)
- **Configurare rută:** Modificați ruta implicită în `Program.cs` astfel încât aplicația să pornească direct cu pagina de listare a articolelor

CURS 5 - Model și Entity Framework Core

Cuprins

- Model (Stratul business - prelucrarea datelor)
- Entity Framework Core
- Instalare Entity Framework Core
- Migrații
- Conexiunea cu Baza de Date
- Dependency Injection
- C.R.U.D. utilizând Entity Framework

1. Ce este Modelul?

Modelul este responsabil pentru **gestionarea și manipularea datelor** din cadrul aplicației. Acesta reprezintă nucleul aplicației, realizând legătura directă cu baza de date și procesând cererile venite de la View prin intermediul Controller-ului. Accesul la date se face prin atributele publice ale claselor.

2. Entity Framework Core (EF Core)

Entity Framework este un **ORM (Object Relational Mapper)** open source pentru .NET.

Caracteristică	Descriere
Scop	Permite dezvoltatorilor să se concentreze pe aplicație, nu pe baza de date, corelând clasele din model cu tabelele din baza de date
EF Core	Este o versiune "light" și cross-platform (Windows, Linux) care suportă atât tehnici database-first , cât și code-first
LINQ	Permite scrierea de interogări direct în C# pentru diverse surse de date (SQL Server, Oracle, XML, etc.)

3. Instalarea EF Core

Instalarea se face prin **NuGet Package Manager** urmând pașii:

1. **Tools** → **NuGet Package Manager** → **Manage NuGet Packages for Solution**
2. Căutarea și instalarea pachetului `Microsoft.EntityFrameworkCore`
3. Acceptarea termenilor de licență
4. Verificarea instalării în consolă sau în secțiunea **Dependencies** a proiectului

Notă: Sunt necesare și pachetele `Microsoft.EntityFrameworkCore.SqlServer` , `.Design` și `.Tools` pentru gestionarea bazei de date și a migrațiilor.

4. Conexiunea cu Baza de Date și Dependency Injection

Dependency Injection (DI)

DI este un design pattern care **abstractizează implementarea**, injectând automat dependențele (servicii, obiecte) într-o componentă (ex: Controller) în loc ca aceasta să le creeze manual.

Metode de conectare:

Varianta 1 (Fără DI)	Varianta 2 (Cu DI)
Necesită instanțierea manuală a contextului în fiecare Controller: <code>private AppDbContext db = new AppDbContext();</code>	Contextul este injectat prin constructor: <code>public ArticlesController(AppDbContext context) { db = context; }</code>
Stringul de conexiune este adesea hardcodat în metoda <code>OnConfiguring</code>	Stringul de conexiune este stocat în <code>appsettings.json</code> și configurat în <code>Program.cs</code>

5. Migrații

Migrațiile sunt necesare pentru a **sincroniza modificările din cod** (clase noi, proprietăți noi) cu baza de date.

Comandă	Funcție
Add-Migration [Nume]	Compară modelul curent cu cel anterior și generează codul necesar
Update-Database	Aplică modificările în baza de date

6. Operații C.R.U.D.

Implementarea operațiilor folosind clasa de context (ex: `AppDbContext db`):

Operație	Implementare
Index (Read all)	Preluarea studenților ordonați: <code>from student in db.Students orderby student.Name select student</code>
Show (Read one)	Găsirea după ID: <code>db.Students.Find(id)</code>
New (Create)	Adăugarea unui obiect nou: <code>db.Students.Add(s); db.SaveChanges();</code>
Edit (Update)	Modificarea atributelor unui obiect existent urmată de <code>db.SaveChanges()</code>
Delete (Delete)	Ștergerea unui obiect: <code>db.Students.Remove(student); db.SaveChanges();</code>

Model Binding: Procesul automat prin care datele trimise prin HTTP POST/GET sunt mapate pe atributele modelului, condiția fiind ca numele câmpurilor din View să coincidă cu cele din Model.

Laborator 5 - Exerciții

Objective

- Crearea modelului de date în cadrul stratului business (Model)

- Configurarea Entity Framework Core și conectarea la baza de date SQL Server
- Utilizarea Dependency Injection pentru gestionarea serviciilor
- Crearea și aplicarea migrațiilor pentru generarea bazei de date

Scenariu

Se va realiza o aplicație [ASP.NET](#) Core MVC pentru gestionarea de **articole** și **categorii**. Relația dintre acestea este de tipul: o categorie conține mai multe articole, iar fiecare articol aparține unei singure categorii. Baza de date va fi construită de la zero folosind tehnica code-first.

Exercițiul 1 - Crearea proiectului și configurarea EF Core

1. Creați un proiect numit **Laborator5** de tip **ASP.NET Core Web App (Model-View-Controller)**
2. Adăugați pachetele NuGet necesare pentru Entity Framework Core și SQL Server

Exercițiul 2 - Crearea modelelor

Adăugați modelele `Article` și `Category` cu următoarele proprietăți:

Entitate	Proprietăți
Article	<code>Id</code> (int, PK), <code>Title</code> (string), <code>Content</code> (string), <code>Date</code> (DateTime)
Category	<code>Id</code> (int, PK), <code>CategoryName</code> (string)

Exemplu implementare `Article`:

```
public class Article
{
    [Key]
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime Date { get; set; }
}
```

Exercițiul 3 - Configurare DbContext și Dependency Injection

4. Realizați conexiunea cu baza de date utilizând **Dependency Injection**
5. Adăugați baza de date SQL Server cu Connection String-ul numit **ArticlesDB**

Exercițiul 4 - Migrații

6. Integrați sistemul de migrații pentru a genera structura bazei de date

Exercițiul 5 - Implementare C.R.U.D.

Creați controllerele `ArticlesController` și `CategoriesController` pentru a implementa operațiile:

- **Index:** Afișarea tuturor înregistrărilor
- **Show:** Afișarea unei singure entități
- **New:** Adăugarea unei noi înregistrări
- **Edit:** Editarea unei înregistrări existente
- **Delete:** Ștergerea unei înregistrări

CURS 6 - Baze de Date și Diagrame E/R

Acest curs acoperă noțiuni fundamentale despre baze de date, proiectarea diagramei Entitate/Relație (E/R) și Conceptuale, precum și implementarea acestora folosind **Entity Framework Core** și **LINQ** în **ASP.NET Core**.

1. Baze de Date - Noțiuni Generale

Ce este o bază de date?

O bază de date este un ansamblu structurat de date coerente, fără redundanță inutilă, permițând prelucrarea eficientă de către mai mulți utilizatori simultan.

Sistemul de Gestiune a Bazelor de Date (SGBD/DBMS)

Aspect	Descriere
Definiție	Un produs software care asigură interacțiunea cu o bază de date
Funcționalități	Stochează date în tabele, le accesează folosind limbajul SQL și asigură securitatea, integritatea și consistența acestora
Exemple	Oracle Database, Microsoft SQL Server, MySQL

Cerințe Minimale

- Redundanță minimă
- Sincronizarea datelor pentru utilizarea simultană
- Securitatea și integritatea datelor (recuperare în caz de erori)
- Furnizare rapidă și flexibilitate

2. Concepte Cheie în Modelarea Datelor

Concept	Descriere
Cheie Primară (PK)	Identificator UNIC pentru intrările dintr-un tabel; nu poate fi <code>null</code>
Cheie Externă (FK)	Referă o cheie primară dintr-un tabel de legătură; poate fi <code>null</code> sau trebuie să corespundă unei valori PK existente
Cheie Primară Compusă	Creată prin combinarea a două sau mai multe coloane pentru a forma un identificator unic
Entitate	Reprezintă un loc, acțiune sau persoană (ex: Studenți, Cursuri). În modelul E/R sunt substantive, iar în cel relațional devin tabele
Relație	Asociere între entități (reprezentate prin verbe). Tipuri: one-to-one (1:1) , one-to-many (1:m) , many-to-many (m:m)
Atribut	O proprietate care descrie o entitate, având nume, tip de date și constrângeri

3. Proiectarea Diagramei Entitate/Relație (E/R)

Regulile de proiectare includ:

1. Identificarea entităților
2. Identificarea relațiilor dintre entități
3. Stabilirea cardinalităților minime și maxime
4. Identificarea atributelor asociate fiecărei entități
5. Stabilirea cheilor primare

Exemplu Practic: Engine de Știri

Pentru un sistem de știri, entitățile identificate sunt:

- **USER**: Utilizatorii sistemului
 - **ROLE**: Tipuri de utilizatori (Vizitator, Înregistrat, Editor, Administrator)
 - **ARTICLE**: Știrile postate
 - **COMMENT**: Comentariile la știri
 - **CATEGORY**: Categoriile știrilor (Știință, Tehnologie, etc.)
-

4. Diagrama Conceptuală

Transformarea din modelul E/R în cel Conceptual urmează reguli specifice:

- Entitățile devin **tabele**
 - Relațiile **one-to-many** devin chei externe plasate în tabelul cu cardinalitatea "many"
 - Relațiile **many-to-many** devin **tabele asociative**
-

5. ASP.NET Core Identity

Framework-ul oferă un sistem preconfigurat pentru autentificare și autorizare:

- Gestionarea utilizatorilor și rolurilor
 - Autentificare cu user/parolă sau externă (Google, Facebook)
 - Pentru activare, se alege **Individual Accounts** la crearea proiectului
 - **Comandă esențială**: `Update-Database` în Package Manager Console pentru a crea tabelele de identitate
-

6. Implementare cu Entity Framework Core (EF Core)

Definirea Claselor (Models)

În EF Core, clasele sunt la singular (ex: `Article.cs`), iar tabelele rezultate sunt la plural (`Articles`).

```
public class Article
{
    [Key]
    public int Id { get; set; }

    [Required(ErrorMessage = "Titlul este obligatoriu")]
    public string Title { get; set; }

    public virtual Category Category { get; set; } // Lazy Loading
    public virtual ICollection<Comment> Comments { get; set; }
}
```

Lazy Loading

Proprietățile sunt declarate ca `virtual` pentru a permite **Lazy Loading**. Acest pattern întârzie încărcarea resurselor până în momentul utilizării lor efective pentru a îmbunătăți performanța.

Interogări LINQ și Join-uri

Metodă	Descriere
Include	Folosit pentru a încărca datele relaționate (Eager Loading). Exemplu: <code>db.Articles.Include("Category")</code>
Group By	Utilizat pentru funcții de grup, cum ar fi numărarea articolelor per categorii folosind <code>.Count()</code>

Migrații

După modificarea modelelor, se execută:

- 1. `Add-Migration NumeMigratie`
- 2. `Update-Database`

Laborator 6 - ArticlesApp

Acest laborator se concentrează pe configurarea unui proiect practic numit **ArticlesApp**, implementarea modelelor de date și realizarea operațiilor CRUD într-un mediu cu autentificare.

1. Configurarea Proiectului

Pas	Descriere
Creare	Se creează un nou proiect ASP.NET Core Web App (MVC) cu numele ArticlesApp
Autentificare	Proiectul trebuie să includă sistemul de autentificare (Individual Accounts)
Pachete necesare	<code>Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore</code> , <code>Microsoft.AspNetCore.Identity.EntityFrameworkCore</code> , <code>Microsoft.AspNetCore.Identity.UI</code> , <code>Microsoft.EntityFrameworkCore.SqlServer</code> , <code>Microsoft.EntityFrameworkCore.Tools</code>

2. Baza de Date și Migrări

- **Update-Database:** Deoarece migrația inițială pentru Identity există deja, se rulează doar comanda `Update-Database`
- **Verificare:** Tabelele pot fi vizualizate în **SQL Server Object Explorer**
- **Testare:** Se rulează proiectul și se înregistrează un cont nou; acesta va apărea în tabelul `dbo.AspNetUsers`

3. Implementarea Modelelor

Article:

- **Proprietăți:** `Id` (PK), `Title` (obligatoriu), `Content` (obligatoriu), `Date`, `CategoryId` (FK)
- **Relații:** Referință către `Category` și o colecție de `Comments`

Category:

- **Proprietăți:** `Id` (PK), `CategoryName` (obligatoriu)
- **Relații:** O colecție de `Articles`

Comment:

- **Proprietăți:** `Id` (PK), `Content` (obligatoriu), `Date`, `ArticleId` (FK)

4. Operații CRUD

Pentru fiecare entitate se adaugă un **Controller** (`ArticlesController`, `CategoriesController`, `CommentsController`) și folderele corespunzătoare în **Views**.

Entitatea Article:

- **Index:** Afișarea tuturor articolelor folosind componente **Bootstrap Cards**
- **Show:** Vizualizarea unui singur articol detaliat
- **New/Edit:** Adăugarea sau editarea unui articol, utilizând un **dropdown** pentru selecția categoriei
- **Delete:** Ștergerea articolului

Entitatea Category:

- Implementarea operațiilor CRUD; la ștergerea unei categorii, se vor șterge automat toate articolele asociate (Cascade Delete)

6. Temă (Comments)

Se cere implementarea CRUD pentru entitatea **Comment**:

- Afișarea comentariilor în pagina de `Show` a unui articol
- Funcționalități de adăugare, editare și ștergere pentru comentarii

CURS 7 - View în **ASP.NET** Core MVC

Cuprins

- View (interfața cu utilizatorul)
- Ce este View-ul
- Razor în cadrul unui proiect **ASP.NET** Core MVC
- Crearea unui View în cadrul unui proiect
- Trimiterea datelor către View (Model, ViewBag, ViewData, TempData)
- Helpere pentru View
- Exemple pentru implementarea alternativă a Helperelor

View (interfața cu utilizatorul)

View-ul reprezintă componenta arhitecturii **MVC** cu care utilizatorii interacționează direct prin intermediul browser-ului.

Caracteristici principale:

Caracteristică	Descriere
Afișarea datelor	În View se afișează înregistrările din baza de date și informațiile generate de aplicație
Tipuri de conținut	Poate conține HTML static sau HTML dinamic (trimis din Controller)
Comunicare	În cadrul arhitecturii MVC, View-ul comunică direct doar cu Controller-ul . Comunicarea cu Modelul se realizează indirect, tot prin intermediul Controller-ului

Structura folderelor

- View-urile se află în folderul principal **Views**
- Fiecare Controller are un folder separat în interiorul `Views`, purtând același nume ca și Controller-ul respectiv
- Folderul **Shared** conține elemente comune, precum layout-uri sau view-uri parțiale, folosite în mai multe pagini ale aplicației

Razor în ASP.NET Core MVC

Razor este motorul de afișare utilizat încă din [ASP.NET MVC 3](#). Acesta permite mixarea tag-urilor HTML cu cod C#.

Sintaxa Razor:

Element	Descriere
Simbolul <code>@</code>	Se folosește pentru a începe o secvență de cod server-side
Cod pe o singură linie	Se utilizează <code>@</code> urmat de variabilă (ex: <code><h3>@ViewBag.Article.Title</h3></code>)
Blocuri de cod	Pentru mai multe linii de instrucțiuni se folosesc acoladele: <code>@{ /* cod */ }</code>
Text în cod	În cadrul unui bloc de cod, se poate afișa text folosind <code>@:</code> (ex: <code>@:este adevărat</code>)

Structuri de control:

- **If:** `@if(conditie) { ... }`
- **For:** `@for (int i=0; i<10; i++) { ... }`
- **Foreach:** `@foreach (var item in colectie) { ... }`

Trimiterea datelor către View

Există mai multe modalități de a transfera date de la Controller către View:

Metodă	Descriere
Model	Folosit pentru trimiterea obiectelor din baza de date prin helper-ul <code>@model</code>
ViewBag	Obiect dinamic ce permite transferul datelor care nu se regăsesc în Model
ViewData	Similar cu ViewBag, dar sub formă de Dicționar (cheie-valoare), unde cheia este un string
TempData	Stochează valori disponibile pentru un request subsecvent (ex: după un redirect). Datele sunt șterse după prima accesare

Observație: ViewBag inserează datele în dicționarul ViewData. Dacă se folosește același nume pentru o proprietate în ambele, ultima valoare alocată va suprascrie valorile anterioare.

Helpere pentru View

ASP.NET Core MVC oferă Helpere pentru a genera automat elemente HTML și pentru a facilita procesul de **model binding**.

Categorii de Helpere:

Categorie	Descriere
Tag Helpers	Arhitectură modernă, procesate la compilare, oferă suport mai bun pentru IDE (autocomplete)
HTML Helpers	Procesate la runtime, pot genera o ușoară scădere de performanță în aplicații foarte mari

Exemple de corespondență:

Funcționalitate	HTML Helper	Tag Helper
Link	Html.ActionLink	<a asp-action="..." asp-controller="...">
Input text	Html.TextBox	<input asp-for="Property" />
Parolă	Html.Password	<input type="password" asp-for="Property" />
Editor	Html.Editor	Generează automat tipul de input în funcție de tipul proprietății din Model

Crearea unui View (Pași)

- PASUL 1:** Click dreapta pe folderul asociat din `Views` -> **Add**
- PASUL 2:** Selectare **Razor View - Empty** sau **Razor View**
- PASUL 3:** Introducerea numelui fișierului cu extensia `.cshtml` și apăsarea butonului **Add**

Laborator 7 - Tag Helpers și TempData

Obiective și Configurare Inițială

Laboratorul se concentrează pe utilizarea **Tag Helpers** și gestionarea datelor între Controller și View folosind modele și variabile de tip **TempData**.

- Modele utilizate:** `Article.cs` , `Category.cs` și `Comment.cs`
- Configurare:**
 - Modificarea stringului de conexiune în `appsettings.json`
 - Executarea migrației: `AddUserArticleCategoryCommentModels`
 - Aplicarea modificărilor în baza de date: `Update-Database`

1. Afișarea unui articol (Metoda Show)

- Controller:** Se modifică metoda `Show` pentru a realiza o operație de **join** și se trimite obiectul de tip `Article` direct ca Model către View
- View:** Trebuie să includă direct modelul (`@model ArticlesApp.Models.Article`) pentru a afișa proprietățile acestuia

Tag Helpers pentru Link-uri:

- Link inițial:* ``

- *Link cu Tag Helpers:* `<a asp-controller="Articles" asp-action="Edit" asp-route-id="@Model.Id">`

2. Adăugarea unui articol (Metoda New)

- **Dropdown Categorii:** Se utilizează o metodă `GetAllCategories()` care returnează o listă de tip `IEnumerable<SelectListItem>` pentru a popula elementul de tip dropdown
- **Proprietate specială:** În modelul `Article`, se adaugă proprietatea `Categ` marcată cu atributul `[NotMapped]` pentru a stoca lista de categorii fără a o salva în baza de date

Sintaxă Dropdown:

```
<select asp-for="CategoryId" asp-items="Model.Categ" class="form-control">
    <option value="">Selectati categoria</option>
</select>
```

- **Mesaje Feedback:** Se utilizează `TempData` pentru a afișa mesaje precum "Articolul a fost adăugat" după acțiunea de tip POST

3. Editarea și Ștergerea

- **Edit (GET):** Preia categoriile și trimite modelul `Article` către View-ul de editare
- **Edit (POST):** Salvează modificările și notifică utilizatorul via `TempData["message"] = "Articolul a fost modificat"`
- **Delete:** După ștergere, se afișează mesajul "Articolul a fost șters"

Noțiuni Teoretice: DbSet și Entity Framework Core

Ce este DbSet?

`DbSet<T>` este o proprietate din `ApplicationDbContext` care reprezintă o colecție de entități ce permite operații de interogare (query) și modificare.

Concept	Descriere
IQueryable	<code>DbSet<T></code> implementează <code>IQueryable<T></code> , permițând utilizarea LINQ pentru a scrie interogări în C# care sunt traduse ulterior în SQL
Execution (Materializare)	Interogările LINQ nu se execută imediat (deferred execution). Acestea se execută doar la materializare, de exemplu prin apelarea metodelor <code>.ToList()</code> , <code>.First()</code> , sau prin iterare într-un <code>foreach</code>

Procesul la Runtime

1. **Construire Query:** Metodele `.Include()` sau `.Where()` construiesc un "expression tree"
2. **Traducere:** EF Core traduce acest arbore în instrucțiuni SQL (ex: `SELECT`, `LEFT JOIN`)
3. **Change Tracker:** `DbContext` monitorizează starea entităților (`Unchanged`, `Added`, `Modified`, `Deleted`) până la apelarea `SaveChanges()`, moment în care se execută efectiv comenzile în baza de date

CURS 8 - Validare și View-uri Partajate

Cuprins

- Validarea cu ajutorul atributelor de validare
- Atribute de validare la nivel de Model (Data Annotations)
- Preluarea validărilor în View

- Validări custom
- Implementarea validărilor la nivel de Controller
- View-uri partajate (Layout View, Partial View)

1. Validarea în ASP.NET Core MVC

Validarea se realizează prin adăugarea atributelor necesare direct în **Model**. Aceste attribute integrează regulile de validare pentru fiecare proprietate a modelului și sunt utilizate împreună cu **Model Binding** și **Data Annotations**.

Atribute de validare comune (Data Annotations)

Cele mai utilizate attribute din namespace-ul `System.ComponentModel.DataAnnotations` includ:

Atribut	Descriere
Required	Verifică dacă inputul este diferit de null
StringLength	Verifică lungimea unui string (permite setarea minimului și maximului)
Range	Verifică dacă valoarea se află într-un anumit interval
RegularExpression	Verifică respectarea unei expresii regulate
EmailAddress	Verifică formatul unei adrese de e-mail
MaxLength / MinLength	Specifică limitele pentru array-uri sau string-uri

Diferența cheie: `StringLength` este folosit pentru validarea modelului și nu influențează implicit baza de date, în timp ce `MaxLength` este utilizat în special pentru migrații EF Core pentru a seta dimensiunea coloanei în baza de date.

2. Implementarea Validării în View

Pentru a afișa erorile în interfața utilizatorului, se folosesc Tag Helper-e specifice:

Atributul asp-validation-for

Acesta este utilizat pentru a afișa mesaje de eroare individuale lângă fiecare câmp.

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

Atributul asp-validation-summary

Afișează un sumar al tuturor erorilor de validare. Valorile posibile includ:

Valoare	Descriere
ModelOnly	Afișează doar mesajele generale adăugate în Controller prin <code>ModelState.AddModelError(string.Empty, "...")</code>
All	Afișează toate erorile, inclusiv cele de la nivel de câmp
None	Nu afișează nimic

Validarea pe Client

Pentru a oferi feedback instantaneu fără a reîncărca pagina, trebuie incluse librăriile jQuery în layout-ul paginii:

- jquery.validate.min.js
- jquery.validate.unobtrusive.min.js

3. Validări Custom

Există două modalități principale pentru validări personalizate:

A. Atribute Personalizate

Se creează o clasă care moștenește `ValidationAttribute` și se suprascrie metoda `IsValid`.

Exemplu: Un atribut care limitează conținutul la 200 de caractere.

B. Validare pe Serviciu (`IValidatableObject`)

Clasa modelului implementează interfața `IValidatableObject` pentru validări complexe care implică mai multe proprietăți. Aceasta folosește cuvântul cheie `yield` pentru a genera erorile secvențial.

4. View-uri Partajate și Reutilizarea Codului

Layout View

Permite definirea unei structuri comune (Header, Footer) pentru toate paginile.

Element	Descriere
@RenderBody()	Placeholder-ul unde va fi injectat conținutul View-ului specific
@await RenderSectionAsync("Scripts", ...)	Permite injectarea scripturilor specifice dintr-un View în Layout

Partial View

Reprezintă secvențe de cod refolosibile în mai multe pagini (ex: afișarea detaliilor unui articol atât în lista `Index`, cât și în pagina `Show`).

Apelarea unui Partial View:

```
<partial name="ArticleInfo" model="article" />
```

Parametrii includ `name` (numele fișierului fără extensie) și `model` (datele transmise către parțial).

Laborator 8 - Validări și Partial Views

1. Modificarea Modelelor (Data Annotations)

Se adaugă atribute de validare pentru a impune reguli asupra datelor introduse:

Modelul Article:

- Titlu:** Obligatoriu (`Required`), lungime maximă de 100 de caractere (`StringLength`) și minim 5 caractere (`MinLength`)

- **Conținut:** Obligatoriu (Required)
- **Categorie:** Selectarea unei categorii este obligatorie (Required)

Modelul Category:

- **Nume:** Obligatoriu (Required)

Modelul Comment:

- **Conținut:** Obligatoriu (Required)
-

2. Preluarea Validărilor în View

Mesajele de validare definite în Model nu apar automat în browser; ele trebuie preluate în fișierele `.cshtml`.

- **asp-validation-for** : Se utilizează pentru a afișa eroarea specifică fiecărui câmp lângă input-ul respectiv
 - **asp-validation-summary** : Se implementează în View-urile `New` și `Edit` pentru a afișa un sumar al tuturor erorilor de validare apărute
-

3. Validarea la nivel de Controller

În metodele de tip `HttpPost`, este esențială verificarea stării modelului prin variabila `ModelState`.

- **Logică:** Dacă `ModelState.IsValid` este `true`, datele sunt salvate în baza de date
 - **Gestionarea erorilor:** Dacă validările eșuează (ramura `else`), metoda trebuie să returneze View-ul trimițând înapoi obiectul modelului pentru a păstra datele introduse și a afișa erorile
-

4. Optimizarea Interfeței (Partial View & Layout)

- **Partial View (ArticleInfo):** Se creează un fișier separat pentru codul HTML utilizat pentru afișarea unui articol, eliminând redundanța între View-urile `Index` și `Show`
 - **Actualizarea Layout-ului:** Se modifică `_Layout.cshtml` pentru a include link-uri de navigare către:
 - Afișarea tuturor articolelor
 - Afișarea tuturor categoriilor
 - Adăugarea unui articol nou
 - **Redirecționare Logo:** Logo-ul aplicației va redirecționa acum către `/Articles/Index` în loc de `/Home/Index`
-

5. Cazul special: Adăugarea Comentariilor

Deoarece formularul de adăugare a unui comentariu se află în pagina `Show` a unui articol, se aplică o logică specifică:

- **Mutarea Metodei:** Metoda de adăugare (`HttpPost`) este mutată în `ArticlesController` sub numele `Show`
 - **Atributul [FromBody]** : Utilizat pentru a specifica faptul că datele obiectului `comment` trebuie preluate exclusiv din formularul HTML
 - **Păstrarea Contextului:** Dacă validarea comentariului eșuează, se reîncarcă articolul cu toate comentariile sale pentru a oferi o experiență de utilizare continuă
-

Resurse: Atribute pentru sursa datelor

Când datele nu sunt transmise corect automat, se pot folosi atribute pentru a indica sursa lor explicită:

Atribut	Sursa datelor
[FromBody]	Corpul cererii HTTP (ex: JSON)
[FromForm]	Datele unui formular HTML (cerere POST)
[FromQuery]	Parametrii din query string (URL)
[FromRoute]	Parametrii de rută din URL
[FromHeader]	Antetele cererii HTTP (ex: token-uri)
[FromServices]	Injectare directă din containerul de dependențe

CURS 9 - Autentificare și Autorizare

Cuprins

- Sistemul de autentificare. Identity Framework
- [ASP.NET](#) Core Identity
- Crearea unui proiect integrând Identity Framework
- Roluri. Asocierea dintre roluri și utilizatori
- Atributul [Authorize]
- Implementare New, Edit și Delete utilizând roluri
- Stocarea fișierelor în baza de date

Sistemul de Autentificare

Framework-ul [ASP.NET](#) Core permite integrarea unui sistem de autentificare prin intermediul **Identity Framework**. Acesta este un sub-framework integrat cu Entity Framework Core pentru operațiunile de bază de date.

Caracteristici Identity Framework:

Caracteristică	Descriere
Autentificare complexă	Permite utilizarea combinației user/parolă sau conturi third-party (Google, Facebook, Twitter etc.)
Managementul Rolurilor	Include posibilitatea creării și manipulării rolurilor utilizatorilor
Stocare	Utilizează SQL Server pentru stocarea utilizatorilor, rolurilor și asocierilor dintre aceștia

ASP.NET Core Identity

Pentru a genera un proiect cu Identity, trebuie selectată opțiunea **Individual Accounts** la crearea proiectului. Această opțiune permite utilizatorilor să își creeze propriile conturi stocate în baza de date a aplicației.

Beneficii:

Beneficiu	Descriere
Securitate	Implementează hashing-ul pentru protejarea parolelor

Beneficiu	Descriere
Integrare externă	Suport pentru servicii third-party
Autorizare	Sistem de roluri și permisiuni integrat
Interfață	Management ușor pentru adăugarea, editarea sau ștergerea utilizatorilor

Crearea unui proiect integrând Identity Framework

Instrucțiuni pe Platforme:

- Windows:** Se utilizează template-ul *ASP.NET Core Web App (Model-View-Controller)* și se selectează *Individual Accounts*
- MAC și Linux:** Se utilizează comanda: `dotnet new webapp --auth Individual -o NumeProiect`

Pachete necesare (Terminal):

```
dotnet tool install --global dotnet-ef --version 9.0.11
dotnet add package Pomelo.EntityFrameworkCore.MySql
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

Roluri. Asocierea dintre roluri și utilizatori

Structura Bazei de Date (Exemplu: Engine de știri)

Diagrama entităților include:

- USERS:** `user_id` (PK), `user_name`, `pass`
- ROLES:** `role_id` (PK), `role_name`
- USER_IN_ROLE:** Tabel de legătură între utilizatori și roluri
- ARTICLES, COMMENTS, CATEGORIES:** Entități corelate cu utilizatorii prin chei externe (`user_id`)

Pași pentru Configurarea Rolurilor:

PASUL 1: Extinderea clasei User

Se creează o clasă `ApplicationUser` în folderul `Models` care moștenește `IdentityUser` .

```
public class ApplicationUser : IdentityUser
{
    // Se pot adăuga atribute suplimentare (data nașterii, bio, etc.)
}
```

PASUL 2: Adăugarea serviciilor în `Program.cs`

Este necesară adăugarea serviciului de management al rolurilor: `.AddRoles<IdentityRole>()`

PASUL 3: Modificarea Contextului Bazei de Date

Clasa `ApplicationDbContext` trebuie să moștenească `IdentityDbContext<ApplicationUser>`

PASUL 4 & 5: Popularea Bazei de Date (SeedData)

Se creează o clasă `SeedData` cu o metodă `Initialize` pentru a insera rolurile implicite (Admin, Editor, User) și utilizatorii inițiali. Aceasta se apelează în `Program.cs` folosind un scope temporar de servicii.

PASUL 9: Atribuirea rolului la înregistrare

În codul sursă generat prin *Scaffolding* pentru pagina de înregistrare, se adaugă:

```
await _userManager.AddToRoleAsync(user, "User");
```

Atributul `[Authorize]`

Se utilizează pentru a restricționa accesul la metodele unui Controller.

Exemplu: `[Authorize(Roles = "User, Editor, Admin")]` permite accesul doar utilizatorilor cu acele roluri specifice.

Implementare New, Edit și Delete

În cadrul metodelor CRUD, acțiunile sunt limitate pe baza identității:

Acțiune	Implementare
Adăugare	Se preia ID-ul utilizatorului curent folosind <code>_userManager.GetUserId(User)</code>
Editare/ Ștergere	Se verifică dacă articolul aparține utilizatorului sau dacă acesta are rolul de Admin folosind <code>User.IsInRole("Admin")</code>

Stocarea fișierelor în baza de date

Pentru a salva imagini/video asociate articolelor:

- Model:** Se adaugă o proprietate `string Image` pentru a salva calea fișierului
- View:** Formularul trebuie să aibă `enctype="multipart/form-data"` și un input de tip `file`
- Controller:**
 - Se injectează `IWebHostEnvironment` pentru a accesa folderul `wwwroot`
 - Se verifică extensiile permise (ex: .jpg, .png, .mp4)
 - Fișierul se salvează fizic în `wwwroot/images`, iar calea este salvată în baza de date
 - Se utilizează `ModelState.Remove` și `TryValidateModel` pentru a revalida modelul după actualizarea căii imaginii

Laborator 9 - Implementare Sistem Autentificare

Exercițiul 1: Implementarea Sistemului de Autentificare

Pornind de la versiunea din laboratorul anterior, implementați sistemul de autentificare și asocierea dintre roluri și utilizatori, urmând toți pașii descriși în **Cursul 9**.

Exercițiul 2: Cerințe Funcționale pentru Aplicație

Aplicația trebuie să îndeplinească următoarele specificații:

Cerință	Descriere
Tipuri de utilizatori	Trebuie să existe cel puțin 4 tipuri: vizitator neînregistrat, utilizator înregistrat, editor și administrator
Vizualizarea știrilor	Orice utilizator poate vizualiza știrile. Pagina principală va afișa cele mai recente știri, ordonate după data postării. Știrile vor fi afișate paginat
Categorii	Știrile sunt împărțite pe categorii create dinamic de către administrator
Editor de text	Publicarea știrilor noi va include un editor de text ce permite utilizarea elementelor de markup
Motor de căutare	Știrile pot fi căutate după titlu, conținut sau conținutul comentariilor

Restricții de acces:

- Utilizatorii fără cont pot vedea doar pagina principală
- Administratorii pot activa sau revoca drepturile utilizatorilor și editorilor

Implementare - Modificări pe Entități

Entitatea Article:

- Index:** Permite preluarea tuturor articolelor și a categoriilor lor. Accesul este restricționat utilizatorilor înregistrați (`[Authorize]`). Trebuie afișat și utilizatorul care a publicat articolul
- Show:** Afișează un singur articol cu detaliile complete (autor, categorie). Toate rolurile autentificate au acces
- New:** Permis doar pentru **Editor** și **Admin**. La salvare (POST), se reține ID-ul utilizatorului care postează
- Edit & Delete:** Editorii pot edita/șterge doar propriile știri. Adminii pot edita/șterge orice articol

Entitatea Category:

- Doar utilizatorii cu rolul **Admin** pot efectua operațiuni CRUD pe categorii

Entitatea Comment:

- Toți utilizatorii autentificați pot adăuga comentarii
- Utilizatorii pot edita/șterge doar comentariile proprii; Administratorul le poate gestiona pe toate

Interfața Utilizator (View-uri)

Personalizarea Meniului (`_Layout.cshtml`)

Link-urile din meniu vor fi vizibile în funcție de rol:

Rol	Link-uri vizibile
User	Vede doar "Afișare articole"
Editor	Vede "Afișare articole" și "Adăugare articol"
Admin	Vede "Afișare articole", "Adăugare articol" și "Afișare categorii"

Se utilizează verificări de tipul: `@if (User.IsInRole("Admin")) { ... }`

Observație Tehnică

Clasa `ApplicationUser` (care moștenește `IdentityUser`) trebuie să definească colecțiile pentru relațiile cu celelalte entități:

```
public virtual ICollection<Comment>? Comments { get; set; }
public virtual ICollection<Article>? Articles { get; set; }
```

CURS 10 - Funcționalități Avansate

Cuprins

- Implementarea relației many-to-many
- Implementarea claselor
- Afișarea paginată
- Includerea unui editor de text
- Funcționalitatea de căutare
- Design-ul într-o aplicație Web
- User Experience (UX)
- Alegerea culorilor potrivite

1. Implementarea relației many-to-many

Pentru exemplificarea relației **many-to-many**, se extinde diagrama proiectată anterior prin adăugarea tabelului **Bookmark**.

Specificații ale relației:

- Relația se stabilește între **Article** și **Bookmark**
- Utilizatorii (rolurile User, Editor, Admin) își pot crea propriile colecții (Bookmarks)
- Utilizatorii cu rolurile User/Editor au acces doar la propriile colecții, în timp ce Admin-ul are acces la toate
- Un articol poate face parte din mai multe colecții, iar o colecție poate conține mai multe articole

Structura Claselor:

Clasă	Proprietăți
Bookmark	<code>Id</code> (PK), <code>Name</code> (Required), și <code>UserId</code> (FK către creator)
ArticleBookmark (Tabel asociativ)	Cheie primară compusă din <code>Id</code> , <code>ArticleId</code> și <code>BookmarkId</code> . Include și <code>BookmarkDate</code> pentru a înregistra momentul adăugării

2. Implementarea claselor și a Contextului

Configurare Model (Fluent API)

În `ApplicationDbContext.cs`, se definește cheia primară compusă și relațiile pentru tabelul asociativ:

```
modelBuilder.Entity<ArticleBookmark>()  
    .HasKey(ab => new { ab.Id, ab.ArticleId, ab.BookmarkId });  
  
modelBuilder.Entity<ArticleBookmark>()  
    .HasOne(ab => ab.Article)  
    .WithMany(ab => ab.ArticleBookmarks)  
    .HasForeignKey(ab => ab.ArticleId);
```

Observație: După modificarea claselor, este obligatorie executarea comenzilor `Add-Migration` și `Update-Database`.

3. Afișarea paginată

Paginarea este esențială pentru gestionarea unui volum mare de date. În exemplul oferit, se afișează **3 articole pe pagină**.

Aspect	Implementare
Logica în Controller	Se calculează <code>offset</code> -ul pe baza paginii curente (<code>(currentPage - 1) * _perPage</code>) și se utilizează metodele <code>.Skip(offset).Take(_perPage)</code> pentru a prelua subsetul corect de date
Interfață	Se utilizează componenta Pagination din Bootstrap

4. Integrarea unui Editor de Text (Summernote)

Summernote este un editor de tip **WYSIWYG** (What You See Is What You Get) bazat pe jQuery și Bootstrap.

Pași de integrare:

- 1. Includerea fișierelor CSS și JS în `_Layout.cshtml` (via CDN)
- 2. Inițializarea editorului într-un fișier JavaScript separat prin selectarea clasei `.summernote`
- 3. Utilizarea atributului `class="summernote"` pe elemente `<textarea>`

Securitate și Prevenire XSS:

- Pentru afișarea conținutului HTML din baza de date se folosește `@Html.Raw()`, dar acest lucru expune aplicația la atacuri **Cross-Site Scripting (XSS)**
- **Soluție:** Instalarea pachetului **HtmlSanitizer** din NuGet pentru a filtra tag-urile periculoase înainte de salvarea datelor în baza de date

5. Funcționalitatea de căutare

Căutarea este implementată pentru a filtra articolele după **titlu**, **conținut** și **conținutul comentariilor**.

Aspect	Implementare
Implementare	Se preia șirul de căutare prin parametrul <code>search</code> din URL (<code>GET</code>)
Integrare cu Paginarea	URL-ul rezultat trebuie să păstreze ambii parametri: <code>/Articles/Index/?search=valoare&page=numar</code>

6. Design și User Experience (UX)

Reguli de bază în Design:

Regulă	Descriere
Simplitatea	Evitarea supraîncărcării paginii cu informații
Consistența	Mentținerea aceluiași stil vizual în toată aplicația
Responsiveness	Utilizarea <i>media queries</i> pentru adaptarea la orice rezoluție
Fonturi	Se recomandă fonturi Sans Serif (Arial, Verdana) la dimensiunea de 16px pentru lizibilitate

Principii UX:

Principiu	Descriere
Tu nu ești utilizatorul final	Testarea cu utilizatori reali (<i>usability testing</i>) este esențială
Feedback	Informarea constantă a utilizatorului (mesaje de succes, loading bars, erori de validare)
Spații albe	Ajută la scanarea rapidă a informației de către utilizator

Alegerea Culoilor:

Tehnică	Descriere
Tehnica 60-30-10	60% culoare dominantă, 30% secundară, 10% accente
Simbolică	Roșu (erori/încredere), Verde (succes/calm), Alb (puritate/simplitate)

Laborator 10 - Exerciții

1. Gestiunea Colecțiilor (Bookmarks)

Pornind de la diagrama conceptuală, se va implementa funcționalitatea de adăugare a articolelor în colecții:

- Creare și Acces:** Utilizatorii pot crea colecții proprii. Aceștia au acces exclusiv la colecțiile create de ei
- Roluri:** Utilizatorii cu rolurile **User**, **Editor** și **Admin** pot crea colecții. Admin-ul are acces la toate colecțiile din platformă
- Relație:** Un articol poate face parte din mai multe colecții, iar o colecție poate conține mai multe articole (relație many-to-many)

2. Administrare Utilizatori

- Administratorii au dreptul de a activa sau revoca drepturile celorlalți utilizatori
- Aceștia pot, de asemenea, să editeze și să șteargă utilizatori din aplicație

3. Integrare Funcționalități Curs 10

- Paginare:** Implementarea afișării paginate a articolelor
- Editor Text:** Integrarea editorului **Summernote**
- Căutare:** Implementarea motorului de căutare pentru articole după titlu, conținut sau conținutul comentariilor

Design și Stilizare

4. Fonturi și Tipografie

Pentru integrarea fonturilor (ex: Roboto, Montserrat):

- 1. Se utilizează `@import` din Google Fonts direct în `wwwroot/css/site.css`
- 2. Utilizarea claselor CSS personalizate pentru aplicarea `font-family`

5. Logo și Footer

- **Logo:** Se stilizează folosind fontul *Permanent Marker* în `_Layout.cshtml` și `site.css`
- **Footer:** Trebuie să fie fixat în partea de jos a paginii (`position: fixed`). Afișează logo-ul și anul curent utilizând `@DateTime.Now.Year`

Pagina Principală (Home/Index)

Logica de Afișare pentru Utilizatori Neautentificați

Dacă un utilizator nu este autentificat, pagina principală afișează doar 3 articole:

- **Primul articol:** Afișat pe un rând și o coloană. Se preia folosind `.First()`
- **Următoarele două articole:** Afișate pe același rând, dar pe două coloane. Se preiau folosind `.Skip(1).Take(2)`
- **Restricții Conținut:** Nu se afișează tot conținutul; se utilizează `line-clamp` pentru a limita rândurile afișate

Meniu și Accesibilitate (Layout.cshtml)

Link-urile din meniu sunt condiționate de rolul utilizatorului:

Rol	Link-uri accesibile
Neautentificat	Doar logo-ul aplicației (redirecționează către Home/Index)
User	Afișare articole, Colecții
Editor	Afișare articole, Adăugare articol, Colecții
Admin	Toate cele de mai sus + Afișare categorii, Afișare utilizatori

Exemplu Implementare: HomeController

```
public IActionResult Index()
{
    // Dacă utilizatorul este autentificat, merge la lista completă de articole
    if (User.Identity.IsAuthenticated)
    {
        return RedirectToAction("Index", "Articles");
    }

    var articles = from article in db.Articles select article;

    // Preluare articole pentru pagina de start (vizitatori)
    ViewBag.FirstArticle = articles.First();
    ViewBag.Articles = articles.OrderBy(o => o.Date).Skip(1).Take(2);

    return View();
}
```

CURS 11 - REST API

Cuprins

- REST API și Implementarea unei Aplicații CRUD cu [ASP.NET](#) Core și Identity
- Crearea Proiectului în Visual Studio
- Introducere în REST API
- Sistemul de fișiere
- Sistemul de Rutare
- Configurarea Identity Framework
- Implementarea CRUD pentru ArticlesController
- Gestionarea Utilizatorilor

Crearea Proiectului în Visual Studio

Pentru a crea o aplicație REST API cu [ASP.NET](#) Core și Identity:

1. **Deschideți Visual Studio** și selectați **Create a new project**
2. Alegeți **ASP.NET Core Web API**
3. Introduceți un nume pentru proiect (ex: `Laborator11RESTAPI`)
4. Selectați **.NET 9** ca framework

Opțiuni de activat:

- **Enable OpenAPI support:** Pentru documentarea API-ului cu **Swagger**. Swagger oferă o interfață interactivă pentru a testa API-ul fără unelte externe precum Postman
- **Configure for HTTPS**
- **Use controllers:** Această opțiune configurează proiectul să folosească Controllers pentru gestionarea logicii API-ului

Introducere în REST API

REST (Representational State Transfer) este un stil arhitectural pentru comunicarea client-server prin HTTP. Spre deosebire de MVC-ul tradițional care returnează View-uri HTML, un REST API returnează date în format **JSON** sau **XML**.

Avantaje:

- Separare completă între client și server
- Poate fi consumat de diverse platforme: web, mobile, IoT
- Scalabilitate și reutilizare

Concepte Cheie:

Concept	Descriere
Endpoint	Un punct final de comunicare (URL specific) unde clienții trimit cereri

Metode HTTP (Verbe):

Verb	Utilizare
GET	Afișează informații
POST	Creează o resursă

Verb	Utilizare
PUT	Actualizează o resursă
DELETE	Șterge o resursă

Sistemul de Rutare și Structura Fișierelor

Structura Recomandată:

Folder	Conținut
/Controllers	Metodele pentru endpoint-uri
/Models	Definirea entităților (ex: <code>Article.cs</code> , <code>Product.cs</code>)
/Data	Contextul bazei de date (<code>AppDbContext</code>)
/Services	Logica de business (ex: <code>ProductService.cs</code>)

Exemplu de Rutare:

Utilizarea adnotărilor `[Route("api/[controller]")]` și `[ApiController]` permite configurarea automată a rutelor. De exemplu, un controller numit `ArticlesController` va avea ruta de bază `api/articles` .

Configurarea Identity Framework

Pentru a gestiona utilizatorii în .NET 9 (versiunea 9.0.9), sunt necesare următoarele pachete:

- `Microsoft.AspNetCore.Identity.EntityFrameworkCore`
- `Microsoft.EntityFrameworkCore.SqlServer`
- `Microsoft.EntityFrameworkCore.Tools`

Configurarea în Program.cs:

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

builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<AppDbContext>()
    .AddDefaultTokenProviders();
```

Gestionarea Utilizatorilor cu JWT

Într-un API RESTful, autentificarea se face de obicei prin **JSON Web Tokens (JWT)**.

Fluxul de lucru JWT:

Pas	Descriere
1. Login	Clientul trimite email-ul și parola

Pas	Descriere
2. Generare	Serverul validează datele și returnează un JWT
3. Stocare	Clientul salvează token-ul și îl trimite în antetul <code>Authorization: Bearer <token></code> la fiecare cerere ulterioară
4. Validare	Serverul verifică semnătura și valabilitatea token-ului înainte de a procesa cererea

Notă Importantă: Cheile secrete pentru semnarea JWT nu trebuie păstrate direct în codul sursă. Se recomandă utilizarea unor sisteme precum **Azure Key Vault** sau **dotnet Secret Manager**.

Implementarea ArticlesController

Pentru a preveni ciclurile de serializare (recursivitate infinită) cauzate de relațiile bidirecționale din Entity Framework, se utilizează anotarea `[JsonIgnore]` pe proprietățile virtuale ale colecțiilor.

Exemplu Metodă Index (GET):

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Article>>> Index()
{
    var articles = await db.Articles
        .Include(a => a.Category)
        .Include(a => a.User)
        .ToListAsync();

    if (articles == null || articles.Count == 0)
        return NotFound();

    return Ok(articles);
}
```

Accesarea Swagger

Pentru a testa API-ul, Swagger este accesibil la adresa: `https://localhost:7020/swagger/`. Asigurați-vă că `launchBrowser` este setat pe `True` în `launchSettings.json`.

CURS 12 - Integrare AI pentru Analiza Sentimentelor

This course covers the integration of AI services for sentiment analysis within an **ASP.NET** Core MVC application, specifically focusing on OpenAI and Google AI (Gemini).

1. Creating an OpenAI Account and Obtaining an API Key

To use OpenAI services, follow these steps:

- Step 1: Access the Registration Page:** Navigate to `https://platform.openai.com/signup`
- Step 2: Complete Registration:** Create an account using an email address or through Google, Apple, or Microsoft
- Step 3: Access API Keys Section:** After logging in, navigate to the "API keys" section under "Manage" in the left sidebar
- Step 4: Create a New Key:** Click "+ Create new secret key"
 - Owned by:** "You" or "Service account"

- **Name:** A descriptive name (e.g., "App Web Production")
 - **Permissions:** Select "All" for full access
5. **Step 5: Save the Key:** Copy and save the key immediately

Important: Never share your API key or expose it in client-side code. Use Environment Variables, User Secrets, or Azure Key Vault for production.

2. Implementing Sentiment Analysis

The application implements an automated sentiment analysis system for comments on the `ArticlesApp` platform.

Main Features:

Feature	Description
Automatic Analysis	Analyzes sentiment when a comment is posted
Classification	Categorizes comments as "positive", "neutral", or "negative"
Confidence Score	Provides a score between 0% and 100%
Admin Visibility	Sentiment labels are only visible to users with the Admin role

Data Flow:

1. User posts a comment
2. `ArticlesController` receives the request
3. `SentimentAnalysisService` sends the text to the AI API
4. API returns `{label, confidence}`
5. Comment is saved in the Database with sentiment data
6. View displays the sentiment label for Admins

3. Technical Implementation Steps

Step 1: Modify the Comment Model

Add fields to store sentiment results in `Models/Comment.cs` :

```
public string? SentimentLabel { get; set; } // positive, neutral, negative
public double? SentimentConfidence { get; set; } // 0.0 - 1.0
public DateTime? SentimentAnalyzedAt { get; set; } // Audit timestamp
```

Step 2: Create the Sentiment Analysis Service

The service uses `ISentimentAnalysisService` for Dependency Injection, allowing easy implementation swaps (e.g., switching from OpenAI to Google AI).

Key Prompting Rules:

The service uses a "system prompt" to force the AI to respond **only** with a fixed JSON format to ensure successful automated parsing.

Step 3: Configure the API Key

Store the key in `appsettings.json` :

```
"OpenAI": {
  "ApiKey": "sk-proj-____"
}
```

Step 4: Register Service in DI Container

In `Program.cs` , register the service as Scoped (new instance per HTTP request).

Step 5: Update Database

Create and apply migrations using the terminal:

- **Windows:** `Add-Migration AddSentimentToComments` then `Update-Database`
- **Mac/Linux:** `dotnet ef migrations add AddSentimentToComments` then `dotnet ef database update`

4. Switching to Google AI (Gemini)

The architecture allows switching providers by implementing the same `ISentimentAnalysisService` interface.

1. **Obtain Key:** Get a key from Google AI Studio (`https://aistudio.google.com/`)
2. **Add Configuration:** Add `"GoogleAI": { "ApiKey": "... " }` to `appsettings.json`
3. **Update Registration:** Change the implementation in `Program.cs` :

```
builder.Services.AddScoped<ISentimentAnalysisService, GoogleSentimentAnalysisService>();
```

ANEXĂ - Sumar Rapid pentru Examen

Comenzi Esențiale Entity Framework

Comandă	Descriere
<code>Add-Migration [Nume]</code>	Creează o nouă migrație
<code>Update-Database</code>	Aplică migrațiile în baza de date
<code>Remove-Migration</code>	Șterge ultima migrație

Atribute de Validare Frecvent Folosite

Atribut	Utilizare
<code>[Required]</code>	Câmp obligatoriu
<code>[StringLength(max)]</code>	Lungime maximă
<code>[MinLength(min)]</code>	Lungime minimă
<code>[Range(min, max)]</code>	Interval numeric
<code>[EmailAddress]</code>	Format email
<code>[Key]</code>	Cheie primară

Atribut	Utilizare
[NotMapped]	Nu se salvează în BD

Tipuri de Rezultate Controller

Metodă	Ce returnează
View()	HTML (ViewResult)
Json()	JSON (JsonResult)
Content()	Text simplu
Redirect()	Redirecționare
NotFound()	HTTP 404
Ok()	HTTP 200 + date

Verificări Roluri în View

```
@if (User.Identity.IsAuthenticated) { ... }
@if (User.IsInRole("Admin")) { ... }
```

Tag Helpers Frecvent Folosite

```
<a asp-controller="Articles" asp-action="Index">Link</a>
<input asp-for="Title" />
<span asp-validation-for="Title"></span>
<select asp-for="CategoryId" asp-items="Model.Categories"></select>
```