

Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC

Curs 8

Cuprins

View (interfață cu utilizatorul).....	2
Validarea cu ajutorul atributelor de validare	2
Atribute de validare la nivel de Model (Data Annotations)	2
Error Message	5
Exemple de implementare la nivel de Model	5
Preluarea validărilor în View	8
Atributul <i>asp-validation-for</i>	8
Exemplu de implementare la nivel de View	8
Includerea librăriilor jQuery Validation și jQuery Unobtrusive Validation	13
Formularul de adăugare - <form>.....	14
Formularul de editare - <form>	14
Atributul <i>asp-validation-summary</i>	16
Vizualizarea mesajelor de validare în browser.....	18
Validări custom	18
Validare custom folosind atribute personalizate.....	19
Validare pe serviciu (IValidatableObject)	20
Implementarea validărilor la nivel de Controller.....	21
View-uri partajate	21
Layout View.....	22
Adăugarea unui nou Layout.....	24
Partial View.....	31

View (interfața cu utilizatorul)

Validarea cu ajutorul atributelor de validare

În ASP.NET Core MVC, validarea se poate realiza prin intermediul adăugării atributelor necesare în Model. Atributele de validare oferă posibilitatea integrării regulilor de validare pentru fiecare atribut/proprietate a modelului. Atributele de validare sunt, de regulă, utilizate împreună cu **Model Binding** și **Data Annotations**.

Atribute de validare la nivel de Model (Data Annotations)

Aceasta este cea mai utilizată metodă de validare. Se folosesc atrbute din namespace-ul **System.ComponentModel.DataAnnotations**.

Cele mai utilizate atrbute de validare sunt următoarele:

- **Required** – verifică dacă inputul este diferit de null;
- **StringLength** – verifică dacă lungimea unui string este mai mică sau egală decât limita impusă;
- **Range** – verifică dacă valoarea inputului se află într-un anumit interval;
- **RegularExpression** – verifică dacă valoarea respectă expresia regulată;
- **CreditCard** – verifică dacă valoarea are un format specific unui cod bancar;
- **CustomValidation** – reprezintă o validare custom (creată de dezvoltator pentru a valida un anumit atribut);

- **EmailAddress** – verifică dacă valoarea inputului are un format specific unei adrese de e-mail;
- **FileExtension** – verifică dacă valoarea corespunde unei denumiri de extensie;
- **MaxLength** – specifică valoarea maximă pe care o poate avea un array sau un string;
- **MinLength** – specifică valoarea minimă pe care o poate avea un array sau un string;
- **Phone** – verifică dacă valoarea reprezintă un număr de telefon valid;
- **DataType** – verifică tipul de date al inputului;

Diferența dintre StringLength și MaxLength

StringLength:

- Se aplică **doar pe proprietăți de tip string**
- Este folosit pentru **validare la nivel de model (Model Validation)**
- Permite setarea atât a **minimului**, cât și a **maximului** de caractere
- NU influențează schema bazei de date (în mod implicit)

Exemplu:

```
[StringLength(50, MinimumLength = 5)]
public string Nume { get; set; }
```

În acest caz:

- La model binding se validează ca stringul să aibă **între 5 și 50 de caractere**
- EF Core nu schimbă coloana automat în baza de date

MaxLength:

- Se aplică pe **string**, **array**, **colectii**, etc.
- NU validează direct inputul în model binding (pentru string se comportă similar, dar nu oferă opțiunea MinimumLength)
- Este folosit în special pentru **migrații EF Core**, indicând dimensiunea maximă a coloanei din bază
- În model validation, pentru string, funcționează similar cu un upper bound

Exemplu:

```
[MaxLength(50)]
public string Prenume { get; set; }
```

În acest caz:

- EF Core creează în baza de date un nvarchar(50) (SQL Server)
- Validarea modelului acceptă maxim 50 caractere, dar **nu permite MinimumLength**

Acstea validări se aplică automat în timpul ***model binding-ului***, iar dacă există erori, acestea sunt adăugate în ***ModelState***.

Exemplu:

```
if (ModelState.IsValid)
{
    // Dacă validările trec cu succes
}
else
{
    // Dacă validările nu trec cu succes
}
```

Error Message

Atributele de validare permit utilizarea parametrului **ErrorMessage** pentru afișarea către utilizatorul final a unui mesaj de eroare specific, în funcție de validarea utilizată pentru respectivul atribut.

Exemplu:

```
[Required(ErrorMessage = "Continutul articolului este obligatoriu")]
```

Exemple de implementare la nivel de Model

Exemplul 1 – validări asupra modelului **Student**

Se adaugă asupra modelului **Student** următoarele validări:

- **Numele** este obligatoriu;
- **Emailul** este obligatoriu și trebuie să aibă un format specific unei adrese de e-mail;
- **CNP-ul** este obligatoriu și trebuie să aibă exact 13 caractere;
- **Adresa** este obligatorie și nu poate avea mai mult de 50 de caractere;

```

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

    [Required(ErrorMessage = "Numele studentului este
obigatoriu")]
    public string Name { get; set; }

    [Required(ErrorMessage = "Campul e-mail este
obigatoriu")]
    [EmailAddress(ErrorMessage = "Adresa de e-mail nu este
valida")]
    public string Email { get; set; }

    [MinLength(13, ErrorMessage = "Lungimea minima trebuie
sa fie de 13 caractere")]
    [MaxLength (13, ErrorMessage = "Lungimea maxima trebuie
sa fie de 13 caractere")]
    [Required(ErrorMessage = "CNP-ul este obligatoriu")]
    public string CNP { get; set; }

    [StringLength(50, ErrorMessage = "Adresa nu poate avea
mai mult de 50 de caractere")]
    [Required(ErrorMessage = "Adresa este obligatorie")]
    public string Address { get; set; }
}

```

Exemplul 2 – validări asupra modelului **Article** (exemplul din cadrul laboratorului)

Se adaugă asupra modelului **Article** următoarele validări:

- **Titlul** articolului este obligatoriu, poate avea o lungime maximă de 100 de caractere și nu poate avea mai puțin de 5 caractere;
- **Conținutul** articolului este obligatoriu;
- **Categoria** din care face parte articolul este obligatorie;

```

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

    [Required(ErrorMessage = "Titlul este obligatoriu")]
    [StringLength(100, ErrorMessage = "Titlul nu poate avea
mai mult de 100 de caractere")]
    [MinLength(5, ErrorMessage = "Titlul trebuie sa aiba mai
mult de 5 caractere")]
    public string Title { get; set; }

    [Required(ErrorMessage = "Continutul articolului este
obligatoriu")]
    public string Content { get; set; }

    public DateTime Date { get; set; }

    [Required(ErrorMessage = "Categoria este obligatorie")]
    public int CategoryId { get; set; }

    public virtual Category Category { get; set; }

    public virtual ICollection<Comment> Comments {get; set; }

    [NotMapped]
    public IEnumerable<SelectListItem> Categ { get; set; }
}

```

!OBSERVAȚII

Mesajele de eroare aflate în Model, ca valoare a parametrului **ErrorMessage**, o să fie preluate în View pentru afișare.

Pentru preluarea mesajelor de validare în View **VEZI Secțiunea următoare din curs – Preluarea Validărilor în View**. După adăugarea validărilor în Model, urmează preluarea mesajelor de validare în View.

Preluarea validărilor în View

Atributul *asp-validation-for*

Pentru preluarea validărilor și afișarea mesajelor asociate în View, se utilizează atributul *asp-validation-for*.

Exemplu de implementare la nivel de View

```
<form asp-action="New" asp-controller="Articles" method="post">

    <div class="form-group">
        <!-- Label pentru Titlu -->
        <label asp-for="Title" class="form-label">Titlu Articol</label>
        <br />
        <!-- TextBox pentru Titlu -->
        <input asp-for="Title" class="form-control" />
        <!-- Mesaj de validare pentru Titlu -->
        <span asp-validation-for="Title" class="text-danger"></span>
    </div>

    <br /><br />

    <div class="form-group">
        <!-- Label pentru Conținut -->
        <label asp-for="Content" class="form-label">Conținut Articol</label>
        <br />
        <!-- TextArea pentru Conținut -->
        <textarea asp-for="Content" class="form-control"></textarea>
        <!-- Mesaj de validare pentru Conținut -->
        <span asp-validation-for="Content" class="text-danger"></span>
    </div>

    <br /><br />

    <div class="form-group">
        <!-- Label pentru Selectare Categorie -->
        <label asp-for="CategoryId" class="form-label">Selectați
        categoria</label>
        <!-- DropDownList pentru Categorii -->
        <select asp-for="CategoryId" asp-items="Model.Categ" class="form-
        control">
            <option value="">Selectați categoria</option>
        </select>
        <!-- Mesaj de validare pentru CategoryId -->
        <span asp-validation-for="CategoryId" class="text-danger"></span>
    </div>
```

```
<br /><br />
<!-- Buton de submit -->
<button type="submit" class="btn btn-success">Adaugă articol</button>
</form>
```

În acest moment, după adăugarea validărilor necesare în cadrul Modelului, pentru fiecare atribut care necesită o validare, dar și după preluarea acestor mesaje de validare în View, se poate observa o afișare incorectă a acestora, după cum urmează.

În cazul în care în formularul de adăugare a unui nou articol nu se completează toate inputurile, o să apară mesajele de eroare asociate deoarece **Title**, **Content** și **CategoryId** sunt câmpuri obligatorii. În cazul în care titlul are o lungime mai mică de 5 caractere, atunci o să apară mesajul de eroare asociat.

De asemenea, se poate observa următoarea problemă în cazul în care se încearcă adăugarea unui articol fără completarea câmpurilor:

The screenshot shows a web form titled "Adaugare articol". It has three fields: "Titlu Articol" (Title Article), "Continut Articol" (Article Content), and "Selectati categoria" (Select category). The "Titlu Articol" field is empty and has a red error message: "Titlul este obligatoriu". The "Continut Articol" field is empty and has a red error message: "Continutul articolului este obligatoriu". The "Selectati categoria" field contains the placeholder text "Selectati categoria" and has a red error message: "The value '' is invalid." A green arrow points to this error message. At the bottom is a green "Adauga articol" (Add article) button.

Pentru **Titlu** și **Continut** mesajele de eroare se afișează corect. În cazul ultimului input, dropdown-ul din care se selectează categoria, mesajul de eroare afișat nu este cel definit în Modelul Article, ci mesajul implicit “*The value is invalid*”.

Acest comportament apare deoarece proprietatea **CategoryId** este definită ca tip int (non-nullable). Atunci când utilizatorul nu selectează nicio valoare din dropdown, framework-ul încearcă să lege o valoare goală (" ") la un int, lucru care este imposibil. În acest caz, se declanșează mesajul de validare implicit al model binder-ului (“*The value is invalid*”), înainte ca validarea **[Required]** să poată afișa mesajul configurat în Model.

Pentru a permite mecanismului de validare să folosească mesajul definit în atributul **[Required]**, este nevoie ca atributul **CategoryId** să fie **optional** la nivel de Model, adică de tip **int?**. În acest fel, atunci când nu este selectată nicio valoare, proprietatea primește **null**, iar **[Required]** poate declanșa corect mesajul personalizat, care va fi apoi afișat în View.

În mod similar, și proprietățile de navigație din Modelul Article trebuie declarate **optionale**. De exemplu, proprietatea **Category** nu are o valoare în momentul inițial al creării articolului, ci doar după ce acesta este asociat efectiv cu o categorie. Din acest motiv, Category ar trebui declarată ca fiind **Category?**, iar colecțiile, precum **Comments**, pot fi inițializate ulterior sau din constructor.

Codul devine:

```
...
[Required(ErrorMessage = "Categoria este obligatorie")]
public int? CategoryId { get; set; }

public virtual Category? Category { get; set; }

// De obicei colecțiile se lasă nnullable și inițializate

public virtual ICollection<Comment> Comments { get; set; }
= new List<Comment>();
```

```
[NotMapped]
public IEnumerable<SelectListItem> Categ { get; set; } =
Enumerable.Empty<SelectListItem>();
```

La fel se procedează în cazul tuturor modelelor din cadrul unei aplicații.

Se observă afișarea corectă a mesajelor de validare.

Aduagare articol

Titlu Articol
Titlul este obligatoriu

Continut Articol
Continutul articolului este obligatoriu

Selectati categoria
Categoria este obligatorie

Adauga articol

În cazul în care Titlul este completat, dar are mai puțin de 5 caractere, o să se afișeze mesajul următor de eroare, conform validării.

Adaugare articol

Titlu Articol

Titlul trebuie sa aiba mai mult de 5 caractere

Continut Articol

Continutul articolului este obligatoriu

Selectati categoria

Selectati categoria

Categoria este obligatorie

Adauga articol

În continuare, se află o serie de exemple de implementare a View-urilor.

Codul din View, scris cu Tag Helpers:

```
<label asp-for="Title" class="form-label">Titlu Articol</label>
<br />

<input asp-for="Title" class="form-control" />

<span asp-validation-for="Title" class="text-danger"></span>
```

HTML generat automat (modul în care ASP.NET Core generează codul HTML):

```
<label for="Title">Titlu</label>

<input id="Title" name="Title" class="form-control" data-
val="true" data-val-required="Titlul este obligatoriu." />

<span class="text-danger" data-valmsg-for="Title" data-valmsg-
replace="true"></span>
```

Validarea în browser:

- ***data-val*** și celelalte atribute sunt interpretate de librăriile ***jQuery Unobtrusive Validation*** și ***jQuery Validation***.
- Când utilizatorul încearcă să trimită formularul fără să completeze datele corecte:
 - Mesajele de validare vor fi afișate direct în browser, oferind feedback instantaneu utilizatorului; acesta nu trebuie să mai aștepte răspunsul serverului;
 - Formularul **nu va fi trimis la server** până când erorile nu sunt rezolvate, iar datele sunt validate în browser înainte de a fi trimise. Pașii prin care se include librăriile se află în secțiunea următoare, numită **Includerea librăriilor jQuery Validation și jQuery Unobtrusive Validation**

Includerea librăriilor **jQuery Validation** și **jQuery Unobtrusive Validation**

În momentul în care se utilizează atributul ***asp-validation-for***, framework-ul ASP.NET Core:

1. Generează automat mesajele de eroare pe client:

- Mesajele de eroare definite în model (prin atribute de validare, cum sunt [Required], [MaxLength], etc.) sunt incluse în HTML-ul generat pentru formular.
- **Validarea pe client** înseamnă afișarea mesajelor instant (în browser), fără ca formularul să fie trimis și respins.

2. Integreză validarea cu librăriile jQuery Validation și jQuery Unobtrusive Validation:

- Aceste librării aplică regulile de validare direct în browser, pe baza datelor generate de ASP.NET Core.

3. Configurarea automată a atributelor data- :

- **asp-validation-for** generează attribute HTML specifice pentru validare (data-val, data-val-required, etc.) pe inputurile corespunzătoare, pe care jQuery Validation le interpretează.

Pentru a funcționa validarea pe client, este necesară adăugarea scripturilor necesare în Layout (Views → Shared → _Layout.cshtml), în secțiunea **Scripts**, imediat sub linia de cod care include **jquery**.

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="~/lib/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.min.js"></script>
```

Formularul de adăugare - <form>

Formularul de adăugare se implementează astfel, utilizând tagul **<form>**

```
<form asp-action="NumeMetoda" asp-controller="NumeController"
method="post">
```

Formularul de editare - <form>

Formularul de adăugare se implementează astfel, utilizând tagul **<form>**

```
<form asp-action="NumeMetoda" asp-controller="NumeController"
asp-route-id="id-ul elementului pe care dorim sa-l editam "
method="post">
```

Pentru trimitera id-ului există două variante de implementare

1. Trimiterea id-ului în URL, la fel ca în exemplul anterior

- Atributul ***asp-route-id*** adaugă id-ul articolului în URL-ul formularului

Dacă, de exemplu, id-ul preia valoarea 10, URL-ul formularului va devein → /Articles/Edit/10

2. Transmiterea id-ului ca un câmp ascuns

```
<form asp-action="NumeMetoda" asp-
controller="NumeController" method="post">
    <!-- Input ascuns pentru ID -->
    <input type="hidden" asp-for="Id" />
    ...
</form>
```

- Câmpul ascuns include id-ul ca parte a datelor trimise prin POST. Acesta nu apare în URL, ci este inclus în corpul cererii HTTP.

Când se alege fiecare soluție?

1. Transmiterea id-ului în URL (***asp-route-id***):

- Este utilă pentru scenarii RESTful, unde id-ul este parte a rutei;
- URL-ul rezultat este mai descriptiv (exemplu: /Articles/Edit/10);

2. Transmiterea id-ului ca un câmp ascuns:

- Este utilă dacă se dorește păstrarea URL-ului curat, fără să se expună în adresă;
- Ideal pentru aplicații în care securitatea datelor este prioritată;

Atributul **asp-validation-summary**

Atributul ***asp-validation-summary*** oferă posibilitatea afișării unui **sumar cu toate erorile apărute în timpul validării**.

Acsta se adaugă în formularul din cadrul View-ului, astfel:

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
```

Atributul ***asp-validation-summary*** poate avea următoarele valori:

- **ModelOnly** – afișează mesajele de validare generale asociate modelului, în cadrul Controller-ului (nu cele specifice proprietăților individuale);
- **All** – afișează toate mesajele de validare, inclusiv cele legate de câmpuri;
- **None** - nu afișează nimic;

Exemplu:

Validare atașată pe proprietatea individuală **Title**

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

Validare atașată în Controller-ul specific:

```
[HttpPost]
public IActionResult NumeMetoda(Article model)
{
    if (string.IsNullOrEmpty(model.Title))
    {
        // Adauga o eroare generală asociată modelului
        ModelState.AddModelError(string.Empty, "Titlul trebuie completat");
    }
}
```

Dacă View-ul corespunzător metodei are valoarea **ModelOnly** (**asp-validation-summary="ModelOnly"**), atunci se va afișa mesajul din Controller → **Titlul trebuie completat**.

Erorile legate de câmpuri individuale (ex: "**Titlul este obligatoriu**") vor fi afișate separat lângă câmpurile respective, prin intermediul atributului **asp-validation-for**.

Vizualizarea mesajelor de validare în browser

The screenshot shows a web page titled "Agregare articol". A green curved arrow points from the title to a red box labeled "Atributul asp-validation-summary". Inside this box, there is a bulleted list of validation errors:

- Titlul este obligatoriu
- Continutul articolului este obligatoriu
- Categoria este obligatorie

Below the errors, there are three input fields with their respective validation messages:

- Titlu Articol**: An empty input field with the message "Titlul este obligatoriu" below it.
- Continut Articol**: An empty input field with the message "Continutul articolului este obligatoriu" below it.
- Selectati categoria**: An empty dropdown menu with the message "Selectati categoria" below it.

At the bottom left is a green button labeled "Agrega articol".

Validări custom

În **ASP.NET Core** și **Razor**, validările pot fi realizate în mai multe moduri. Cele 3 moduri în care se pot efectua validări sunt:

- Validare cu atributele de validare (Data Annotations) – validare prezentată anterior;
- Validare custom folosind atrbute personalizate;
- Validare pe serviciu (IValidatableObject);

Validare custom folosind attribute personalizate

În acest context se pot crea propriile attribute de validare pentru scenarii specifice.

Exemplu de implementare:

Se poate crea un nou folder în cadrul proiectului. Se va numi sugestiv **Validations**. În acest folder se creează un fișier de tip **.cs**. Acesta a fost numit **Max200CharsValidation.cs**.

Codul sursă este:

```
using System.ComponentModel.DataAnnotations;

namespace ArticlesApp.Validations
{
    public class Max200CharsValidation : ValidationAttribute
    {
        protected override ValidationResult IsValid(object value,
ValidationContext validationContext)
        {
            if (value is string stringValue && stringValue.Length <
200)
            {
                return ValidationResult.Success;
            }
            return new ValidationResult("Continutul trebuie sa fie sub
200 de caractere");
        }
    }
}

public class Article
{
    ...
    [Required(ErrorMessage = "Continutul articolului este
obligatoriu")]
    [Max200CharsValidation]
    public string Content { get; set; }
    ...
}
```

Validare pe serviciu (*IValidableObject*)

Clasa poate implementa *IValidableObject* pentru a realiza validări mai complexe care necesită acces la mai multe proprietăți.

```
public class Article : IValidableObject
{
    ...
    [Required(ErrorMessage = "Continutul articolului este obligatoriu")]
    public string Content { get; set; }

    public DateTime Date { get; set; }

    ...
    public IEnumerable<ValidationResult> Validate(ValidationContext
validationContext)
    {
        if(Content.Length < 200 && Date > DateTime.Parse("12/24/2024"))
            // DateTime.ParseExact("24/12/2024", "dd/MM/yyyy",
System.Globalization.CultureInfo.InvariantCulture);
        {
            yield return ValidationResult.Success;
        }

        yield return new ValidationResult("Continutul trebuie sa fie de
maxim 200 de caractere");
    }
}
```

În **ASP.NET Core**, cuvântul cheie **yield** nu este specific framework-ului, ci face parte din limbajul **C#**. Acesta este utilizat pentru a genera în mod incremental elemente dintr-o colecție, fără a necesita ca întreaga colecție să fie creată în memorie.

Cuvântul cheie **yield** este folosit în metodele care returnează un tip enumerabil, cum ar fi *IEnumerable<T>* sau *IAsyncEnumerable<T>*. Aceasta permite generarea elementelor secvențial, la cerere, ceea ce economisește memorie și realizează execuția doar atunci când este necesar.

Implementarea validărilor la nivel de Controller

Pentru funcționarea corectă a validărilor, cât și pentru identificarea corectă a datelor în partea de server (server-side), este necesar să adăugăm, în Controller-ul care modifică datele, verificarea stării modelului. Astfel, prin intermediul variabilei **ModelState**, putem să aflăm dacă toate validările au trecut cu succes.

```
[HttpPost]
public IActionResult New(Article article)
{
    article.Date = DateTime.Now;
    article.Categ = GetAllCategories();

    if (ModelState.IsValid)
    {
        db.Articles.Add(article);
        db.SaveChanges();
        TempData["message"] = "Articolul a fost
adaugat";
        return RedirectToAction("Index");
    }
    else
    {
        return View(article);
    }
}
```

View-uri partajate

Interfața unei aplicații, indiferent de tehnologia cu care este realizată, întotdeauna o să conțină foarte multe componente comune tuturor paginilor: Header, Footer, Meniuri, etc. Aceste componente nu se modifică de la o pagină la alta, iar repetarea scrierii aceluiasi cod devinde redundantă. Pentru a facilita implementarea se pot utiliza View-uri globale.

Layout View

Layout View – permite scrierea unui cod comun pentru toate paginile, cât și un **Placeholder** în care se va include conținutul celorlalte pagini. Acest placeholder este definit prin intermediul variabilei **@RenderBody()**. Locul în care este plasată această variabilă în Layout, va fi locul în care se va afișa conținutul View-urilor aferente.

De exemplu, în momentul în care creăm un nou proiect, acesta generează, în mod automat, un layout care include toate resursele necesare: Head, Stiluri CSS, JavaScript, Header, Footer, etc. În acest layout se află metoda **RenderBody()**, metodă prin care toate View-urile create sunt incluse.



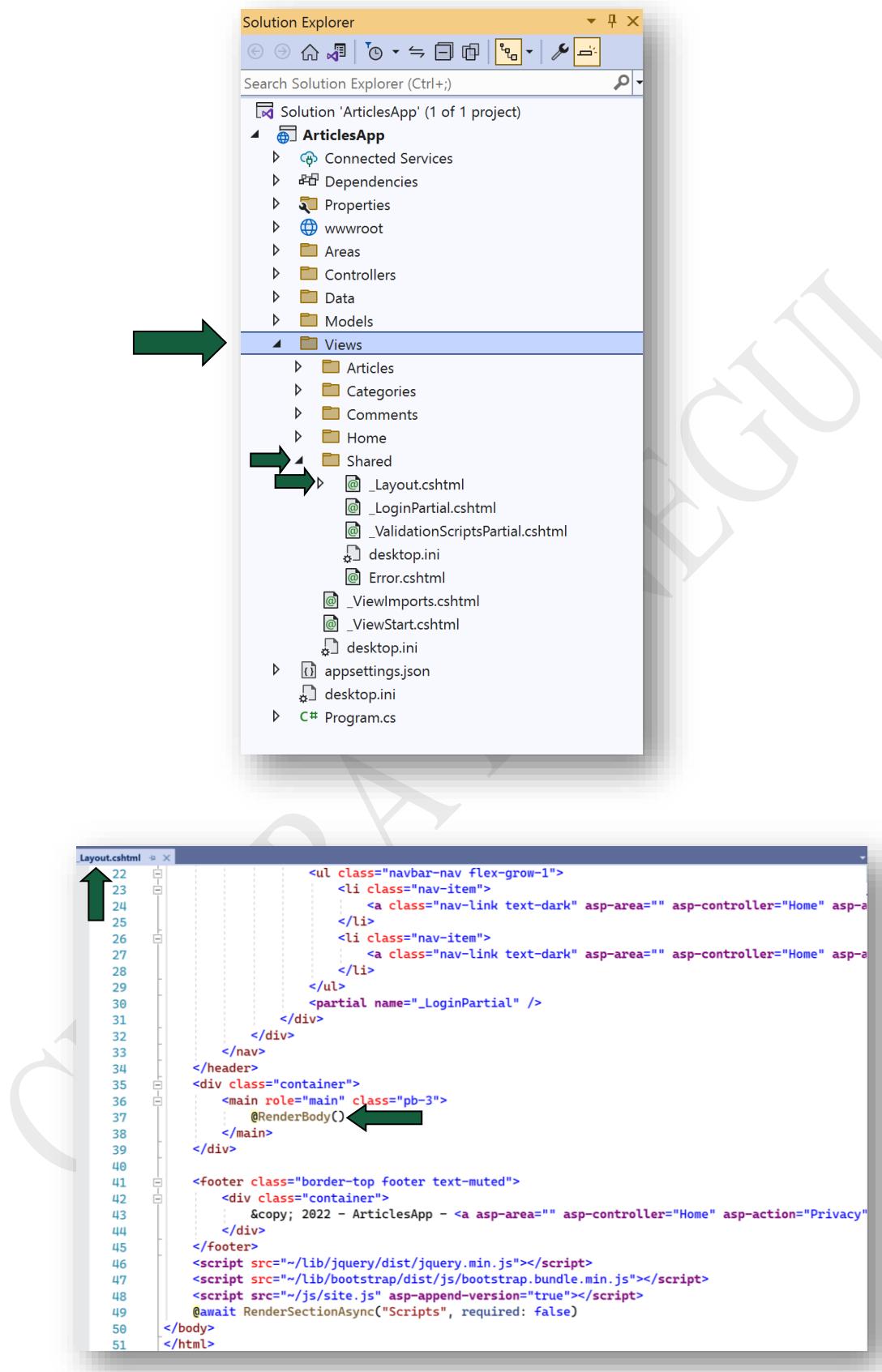
The screenshot shows the Visual Studio IDE interface. On the left is the code editor window titled '_Layout.cshtml' containing the following C# code:

```

22 <ul class="navbar-nav flex-grow-1">
23   <li class="nav-item">
24     <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-a
25     </li>
26   <li class="nav-item">
27     <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-a
28     </li>
29   </ul>
30   <partial name="_LoginPartial" />
31 </div>
32 </div>
33 </header>
34 <div class="container">
35   <main role="main" class="pb-3">
36     @RenderBody()
37   </main>
38 </div>
39 </div>
40
41 <footer class="border-top footer text-muted">
42   <div class="container">
43     &copy; 2022 - ArticlesApp - <a asp-area="" asp-controller="Home" asp-action="Privacy"
44   </div>
45 </footer>
46 <script src="~/lib/jquery/dist/jquery.min.js"></script>
47 <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
48 <script src="/js/site.js" asp-append-version="true"></script>
49   @await RenderSectionAsync("Scripts", required: false)
50 </body>
51 </html>

```

A green arrow points from the line containing `@RenderBody()` to the 'Views' folder in the Solution Explorer window on the right. A large green downward-pointing arrow is positioned below the code editor.

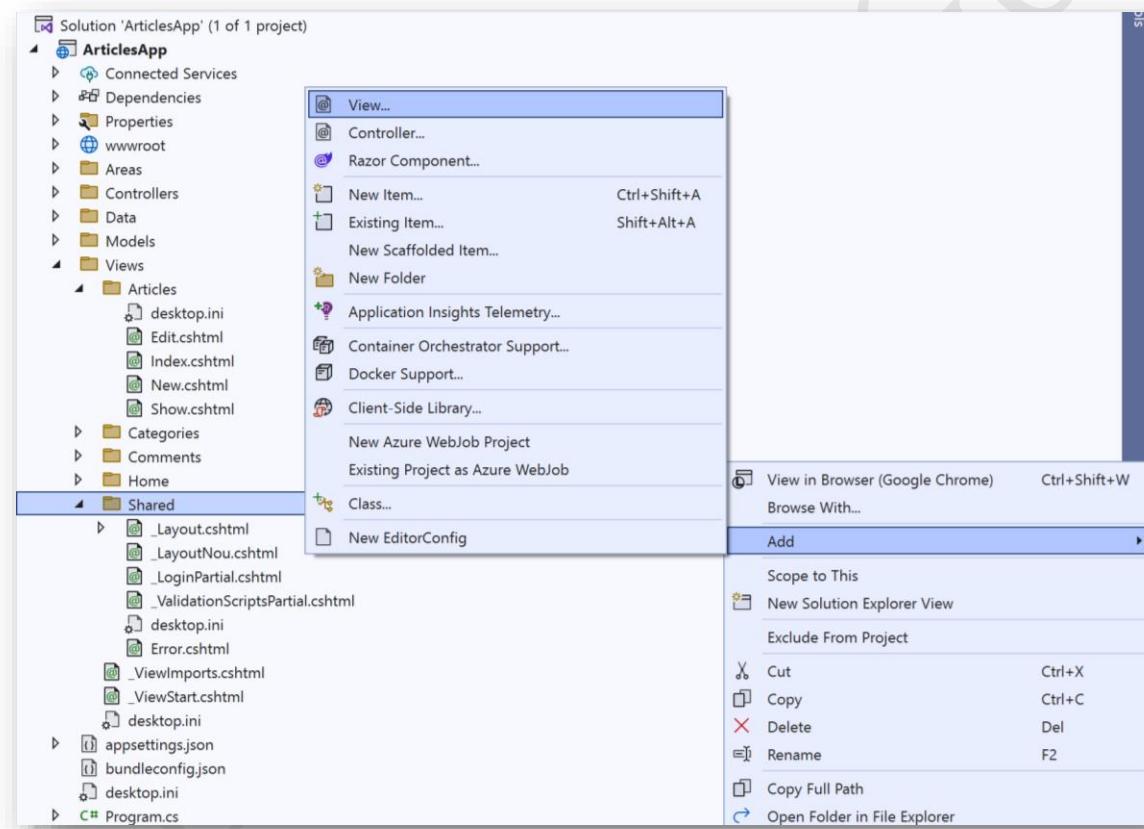


Adăugarea unui nou Layout

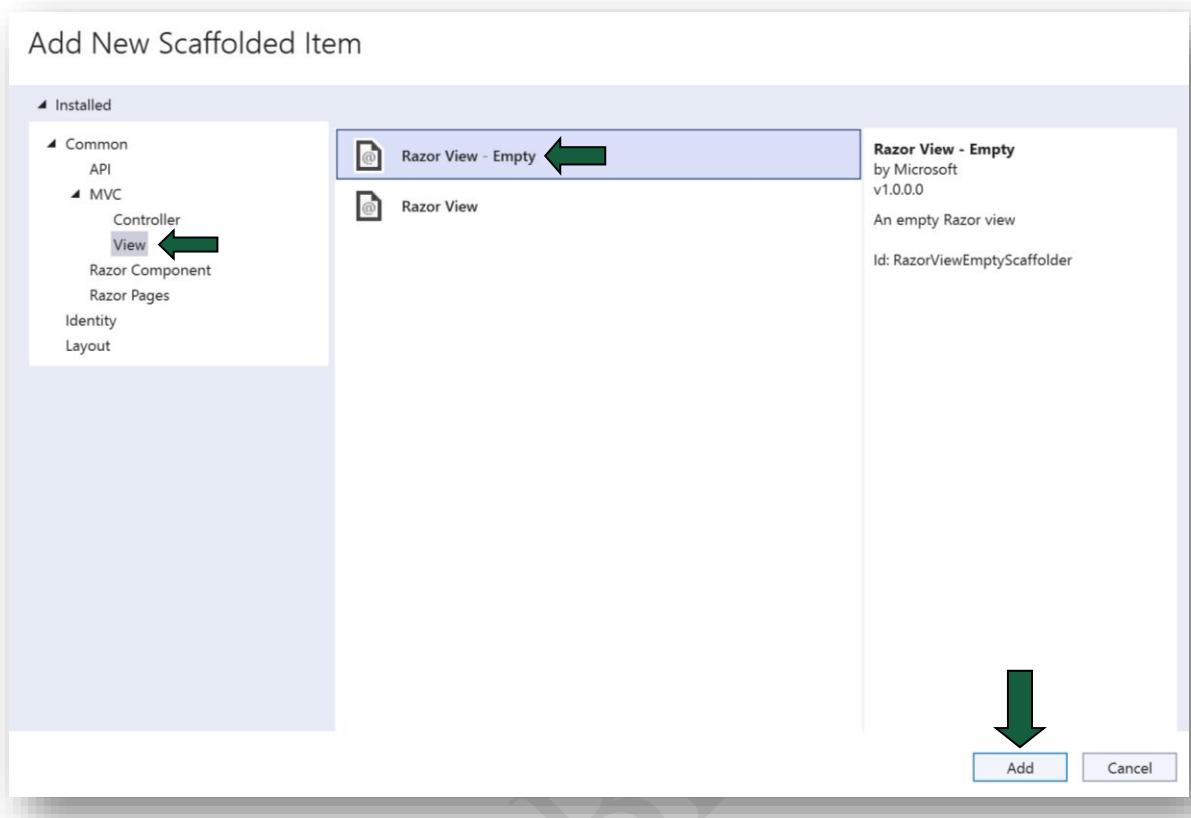
Implicit, proiectul ASP.NET Core are în folderul **Shared** din **Views** un Layout. Acest Layout este utilizat în cadrul tuturor paginilor existente în proiect. În momentul în care se dorește utilizarea unui alt Layout, se poate crea și utiliza după cum urmează:

PASUL 1 – crearea noului Layout

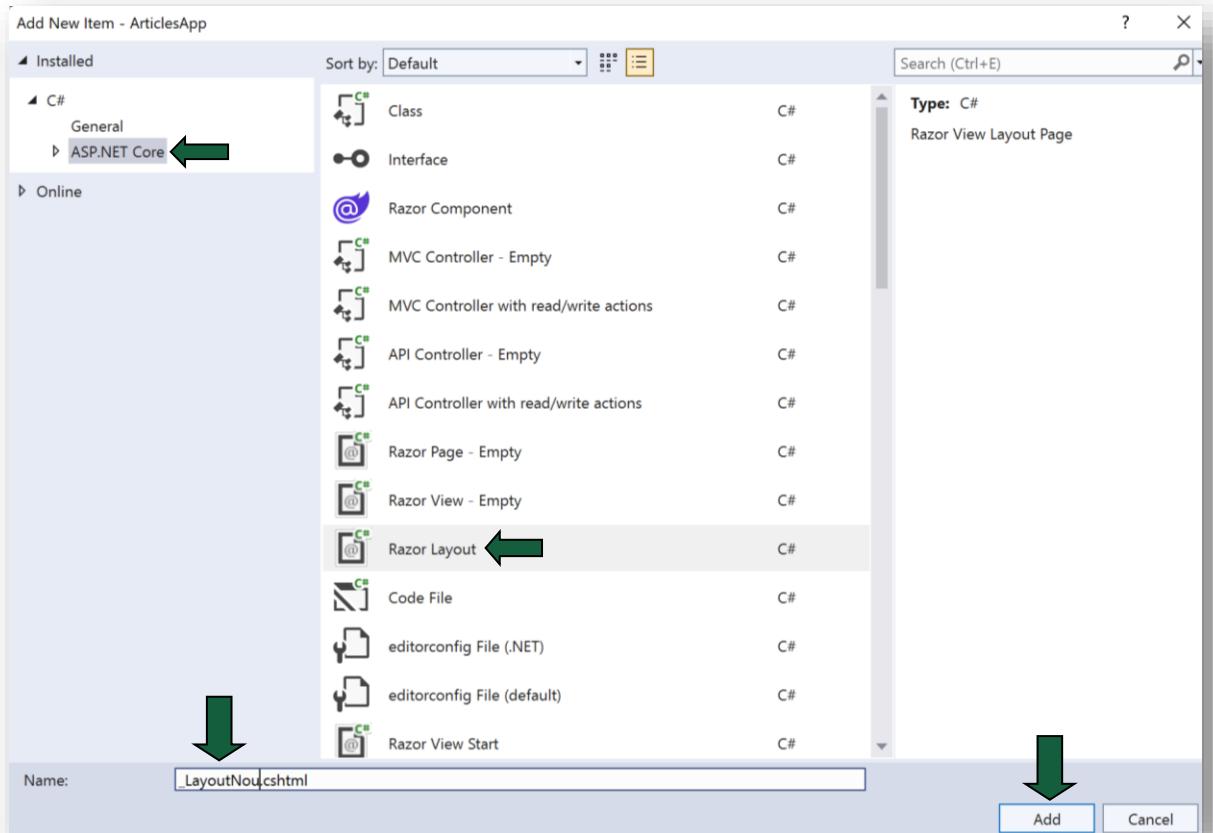
Views → Shared → click dreapta → Add → View



Se selectează Razor View – Empty



**Se selectează ASP.NET Core → Razor Layout
Se redenumește Layout-ul → Add**



View-ul generat de framework o să arate astfel:

```
_LayoutNou.cshtml  □ X
1   <!DOCTYPE html>
2
3   <html>
4     <head>
5       <meta name="viewport" content="width=device-width" />
6       <title>@ViewBag.Title</title>
7     </head>
8     <body>
9       <div>
10      @RenderBody()
11    </div>
12  </body>
13</html>
14
```

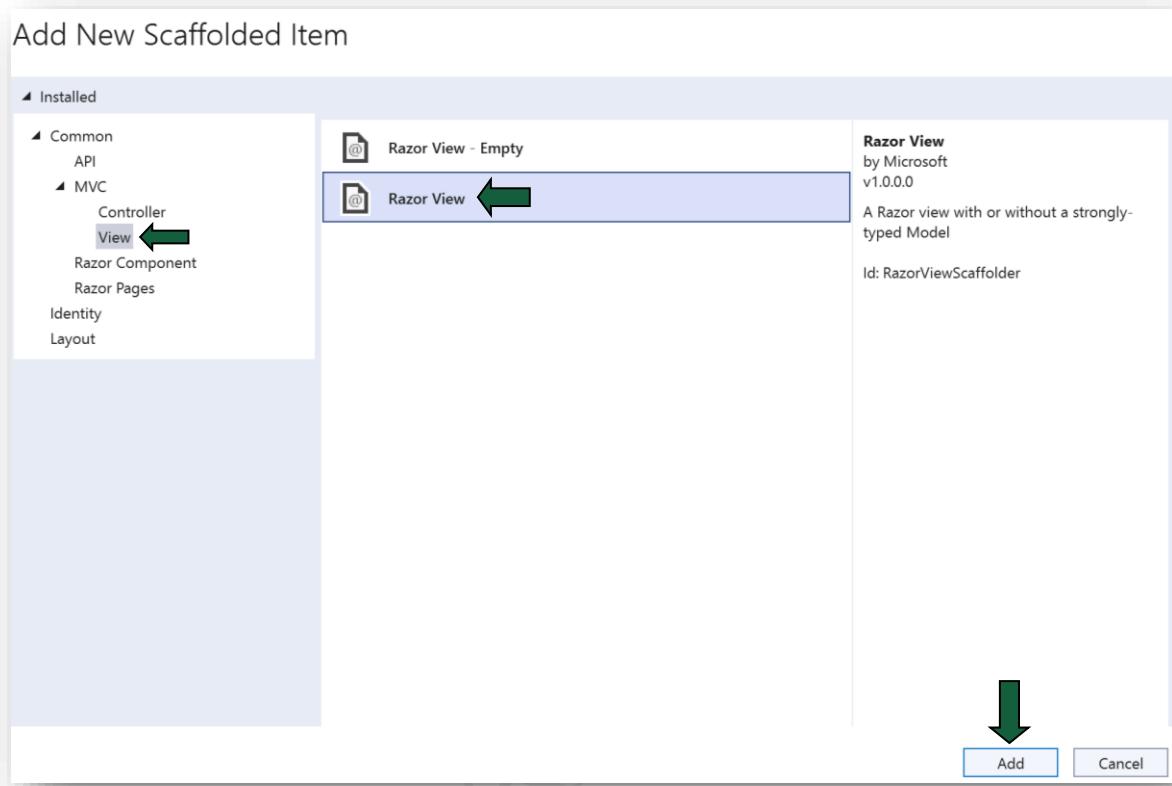
PASUL 2 – modificarea noului Layout, adăugând header și footer



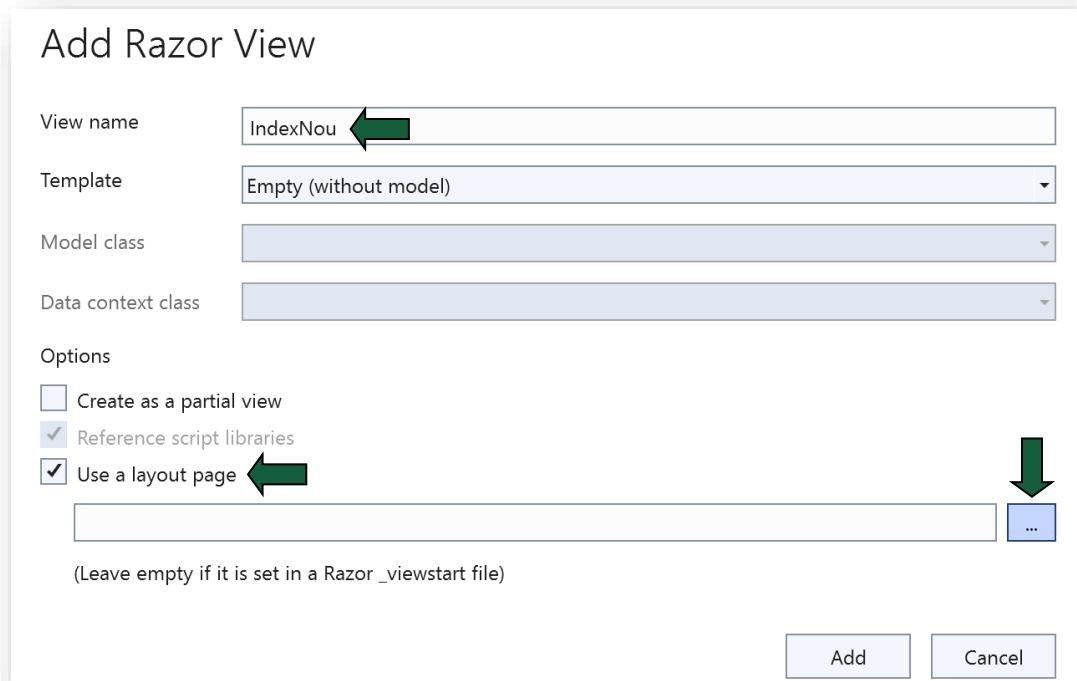
```
IndexNou.cshtml      _LayoutNou.cshtml
13  <header>
14    <div class="container">
15      <div class="col-md-6 offset-3 mt-5">
16
17        <h2>Header</h2>
18
19      </div>
20    </div>
21  </header>
22
23  <body>
24    <div class="container">
25      <div class="col-md-6 offset-3">
26
27        @RenderBody()
28
29      </div>
30    </div>
31  </body>
32
33  <footer>
34    <div class="container">
35      <div class="col-md-6 offset-3">
36
37        <h2>Footer</h2>
38
39      </div>
40    </div>
41  </footer>
42
43  </html>
```

PASUL 3 – adăugarea unui nou View care o să aibă ca Layout noul Layout creat:

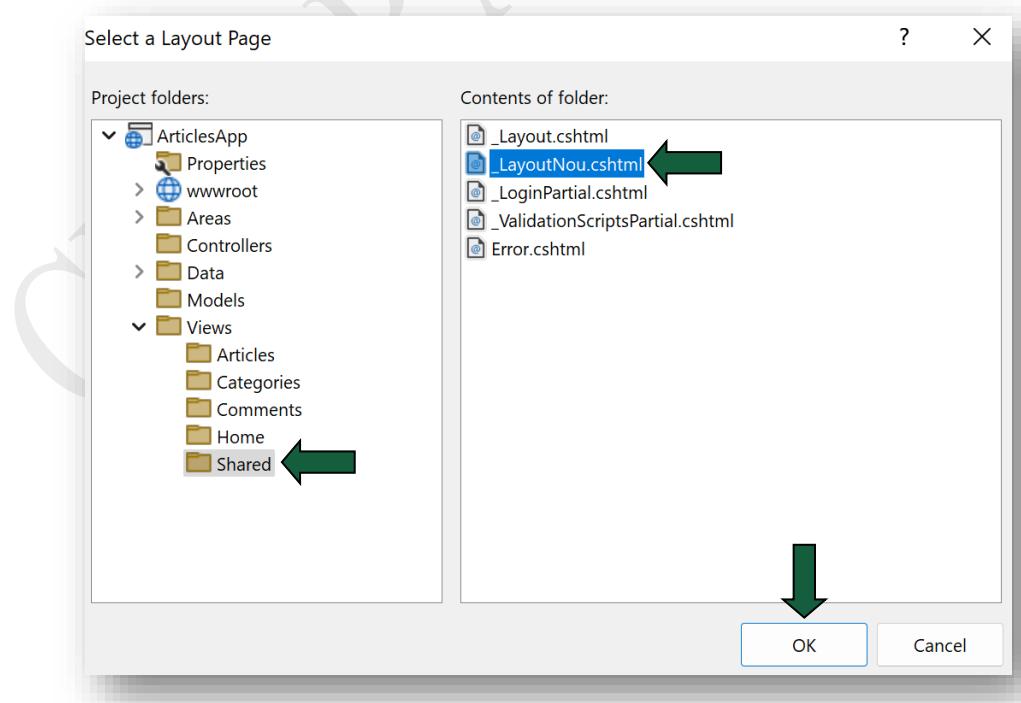
Se adaugă un nou View de tipul Razor View



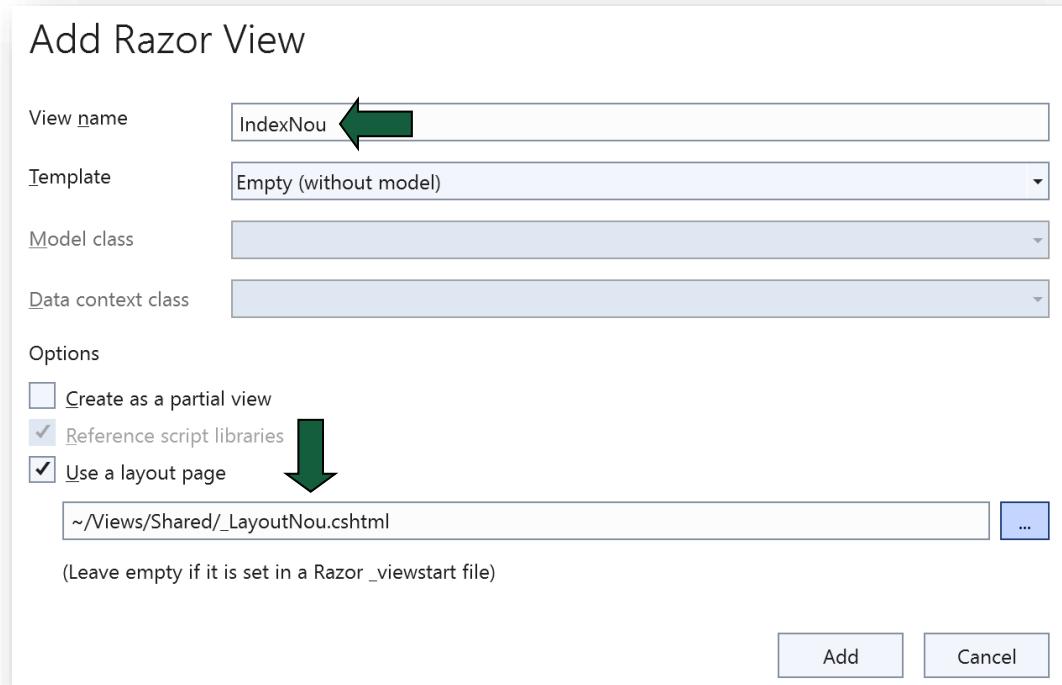
Se adaugă o denumire pentru respectivul View, după care se alege Layout-ul



Se selectează Layout-ul



În cadrul acestui pas se observă Layout-ul pe care tocmai l-am selectat



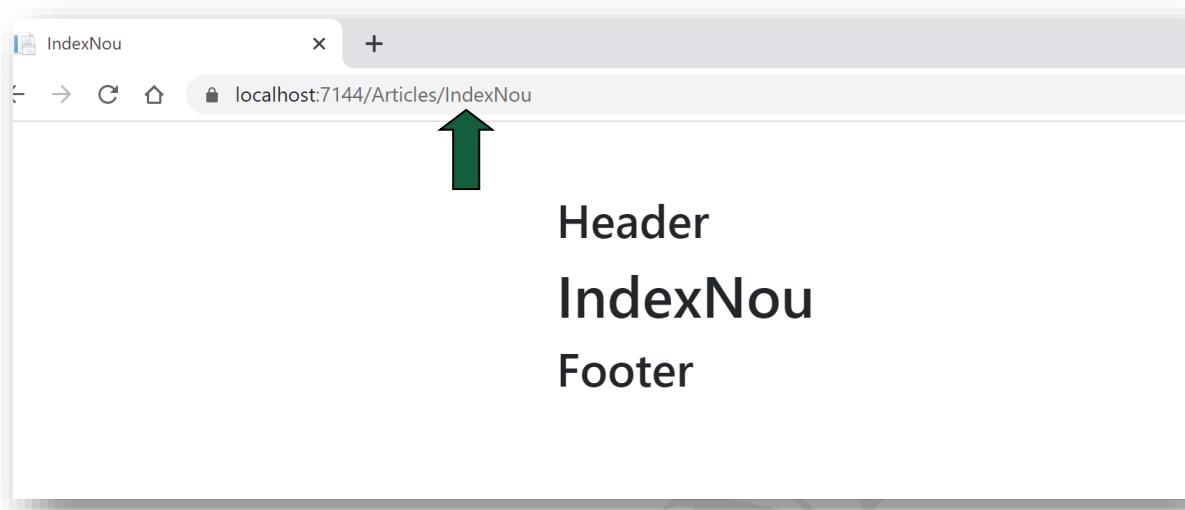
După executarea pașilor anteriori, se poate observa cum în cadrul View-ului numit **IndexNou**, Layout-ul numit **LayoutNou** se include prin intermediul parametrului **Layout**, care primește ca valoare calea din sistemul de fișiere. Mai jos se poate observa secvența de cod generată:

```

1
2     @{
3         ViewData["Title"] = "IndexNou";
4         Layout = "~/Views/Shared/_LayoutNou.cshtml";
5     }
6
7     <h1>IndexNou</h1>
8
9

```

După rulare și accesarea URL-ului → **/Articles/IndexNou** se poate observa includerea cu succes a noului Layout (noul Layout nu are stilizare, exemplul fiind realizat cu scop demonstrativ).



Partial View

Partialele reprezintă secvențe de cod specifice View-urilor, care pot fi refolosite în una sau mai multe pagini. În cadrul dezvoltării aplicațiilor web, codul poate fi reutilizat pentru a optimiza timpul de scriere și pentru a nu include același cod în mod repetitiv.

Secvențele de cod care se repeta în cadrul mai multor pagini pot fi incluse într-un **View** sau într-un **Layout** pentru a fi afișate.

În cadrul aplicației dezvoltate în laborator, afișarea unui articol conține același cod, atât în cazul afișării tuturor articolelor din baza de date → **View-ul Index**, cât și în cazul afișării unui singur articol → **View-ul Show**. În acest caz, pentru a elimina redundanța, și anume scrierea repetitivă a aceluiași cod în cadrul ambelor View-uri, o să se utilizeze un **Partial View**, după cum urmează:

Secvența de cod, cu ajutorul căreia se afișează informațiile corespunzătoare unui articol, se află inclusă în ambele View-uri, ***Index.cshtml*** și ***Show.cshtml***.

```
<div class="card-body">

    <h3 class="card-title alert-success py-3 px-3 rounded-2">@Model.Title</h3>

    <div class="card-text">@Model.Content</div>

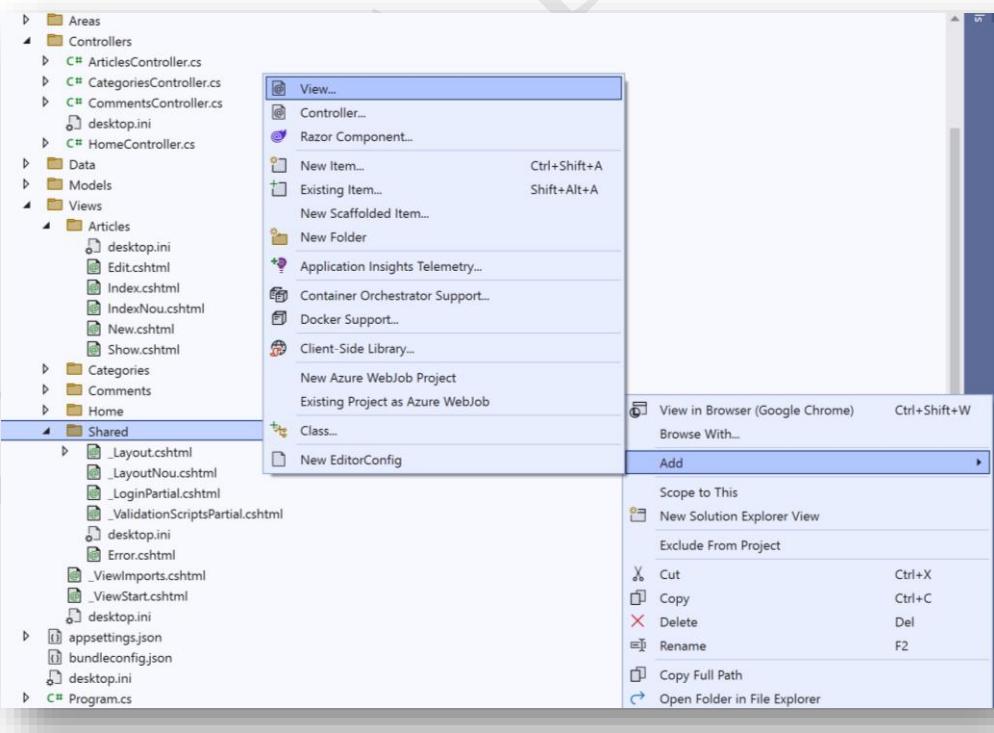
    <div class="d-flex justify-content-between flex-row mt-5">

        <div><i class="bi bi-globe"></i>
@Model.Category.CategoryName</div>

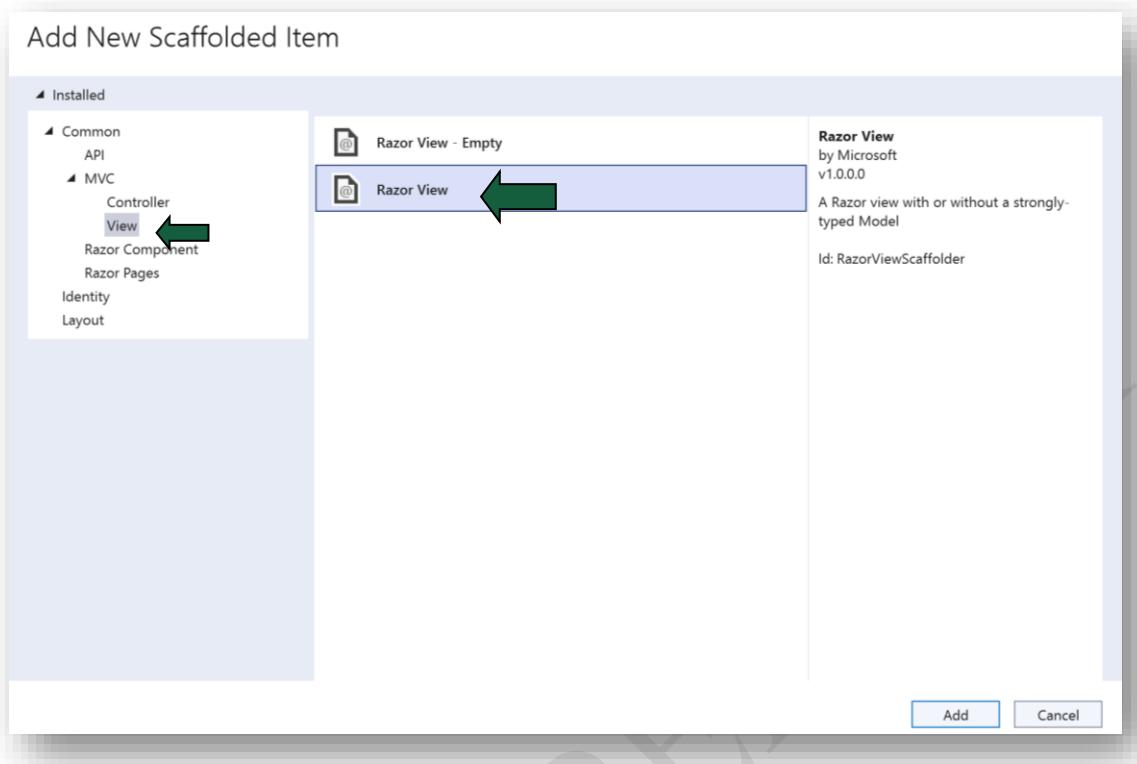
        <span class="alert-success">@Model.Date</span>

    </div>
</div>
```

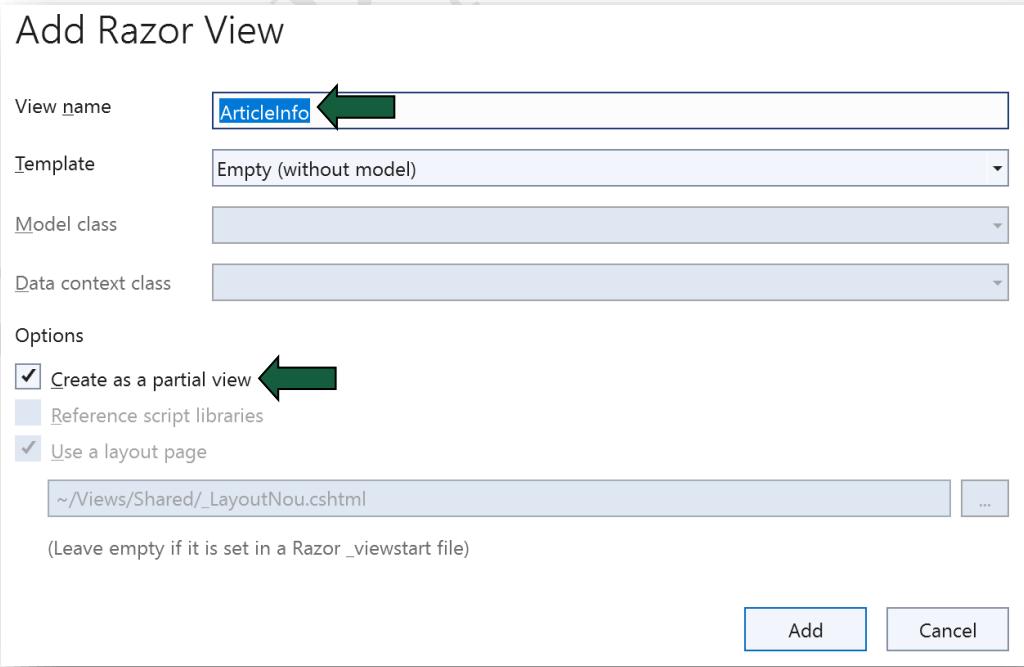
Pentru eliminarea redundanței, se adaugă un nou View de tip **Partial View**. Acest View o să conțină secvența de cod care se repetă.



Se selectează următoarele opțiuni:



View-ul o să fie de tipul **Partial View**, bifând opțiunea
“Create as a partial view”



View-ul numit “**ArticleInfo**” o să conțină secvența de cod care se repetă în ambele View-uri, atât în Index.cshtml, cât și în Show.cshtml.

```

1
2
3
4 <div class="card-body">
5
6   <h3 class="card-title alert-success py-3 px-3 rounded-2">@Model.Title</h3>
7
8   <div class="card-text">@Model.Content</div>
9
10  <div class="d-flex justify-content-between flex-row mt-5">
11
12    <div><i class="bi bi-globe"></i> @Model.Category.CategoryName</div>
13
14    <span class="alert-success">@Model.Date</span>
15
16  </div>
17
18 </div>

```

Pentru includerea parțialului se utilizează atributul specific

```
<partial name="NumePartialView" model="Model" />
```

Parametrii:

➤ **name:**

- Reprezintă numele fișierului pentru Partial View-ul necesar (fără extensia .cshtml);
- **Exemplu:** "ArticleInfo";

➤ **model:**

- Transmite un model către Partial View;

Pentru apelarea parțialului din **View-ul Index** se utilizează pentru primul parametru numele parțialului, iar pentru al doilea parametru obiectul **article** de tipul **Model**. Astfel, în loop trebuie să declarăm tipul modelului și să pasăm acest parametru la parțial. Prin intermediul acestui cod, în parțial putem să folosim variabila **@Model** pentru afișarea datelor.

```
@foreach (ArticlesApp.Models.Article article in
ViewBag.Articles)
{
    ...
    <partial name="ArticleInfo" model="article" />
    ...
}
```

În cazul modificării parțialului, modificările se vor reflecta asupra tuturor View-urilor care utilizează parțialul respectiv, nefiind nevoie de modificări în fiecare View în parte. Acest lucru eficientizează atât timpul de scriere a codului, cât și mențenanța ulterioară a acestuia în cazul în care apar modificări în timp.