

Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC

Curs 8

Cuprins

View (interfata cu utilizatorul).....	2
Validarea cu ajutorul atributelor	2
Atribute de validare la nivel de Model	2
Error Message	3
Exemple de implementare la nivel de Model	3
Preluarea validarilor in View	8
Exemplu de implementare la nivel de View	9
Helper-ul @Html.BeginForm	10
@Html.ValidationMessage si @Html.ValidationMessageFor	11
Helper-ul @Html.ValidationSummary	11
Vizualizarea mesajelor de validare in browser	12
Implementarea validarilor la nivel de Controller.....	15
View-uri partajate	15
Layout View	16
Adaugarea unui nou Layout.....	18
Partial View.....	25

View (interfata cu utilizatorul)

Validarea cu ajutorul atributelor

In ASP.NET Core MVC validarea se poate realiza prin intermediul adaugarii atributelor necesare in Model. Atributele de validare ofera posibilitatea integrarii regulilor de validare pentru fiecare atribut/proprietate a modelului.

Atribute de validare la nivel de Model

Cele mai utilizate atribute de validare sunt urmatoarele:

- **Required** – verifica daca inputul este diferit de null;
- **StringLength** – verifica daca lungimea unui string este mai mica sau egala decat limita impusa;
- **Range** – verifica daca valoarea inputului se afla intr-un anumit interval;
- **RegularExpression** – verifica daca valoarea respecta expresia regulata;
- **CreditCard** – verifica daca valoarea are un format specific unui cod bancar;
- **CustomValidation** – reprezinta o validare custom (creata de dezvoltator pentru a valida un anumit atribut);
- **EmailAddress** – verifica daca valoarea inputului are un format specific unei adrese de e-mail;
- **FileExtension** – verifica daca valoarea corespunde unei denumiri de extensie;

- **MaxLength** – specifica valoarea maxima pe care o poate avea un array sau un string;
- **MinLength** – specifica valoarea minima pe care o poate avea un array sau un string;
- **Phone** – verifica daca valoarea reprezinta un numar de telefon valid;
- **DataType** – verifica tipul de date al inputului;

Error Message

Atributele de validare permit utilizarea parametrului **ErrorMessage** pentru afisarea catre utilizatorul final a unui mesaj de eroare specific, in functie de validarea utilizata pentru respectivul atribut.

Exemplu:

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

Exemple de implementare la nivel de Model

Exemplul 1 – validari asupra modelului Student

Se adauga asupra modelului Student urmatoarele validari:

- **Numele** este obligatoriu
- **Emailul** este obligatoriu si trebuie sa aiba un format specific unei adrese de e-mail
- **CNP-ul** este obligatoriu si trebuie sa aiba exact 13 caractere
- **Adresa** este obligatorie si nu poate avea mai mult de 30 de caractere

```

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

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

    [Required(ErrorMessage = "Campul e-mail este obligatoriu")]
    [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 – validari asupra modelului Article (exemplul din cadrul laboratorului)

Se adauga asupra modelului Article urmatoarele validari:

- **Titlul** articolului este obligatoriu, poate avea o lungime maxima de 100 de caractere si nu poate avea mai putin de 5 caractere
- **Continutul** 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 de 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; }
}

```

!OBSERVATII

Mesajele de eroare aflate in Model, ca valoare a parametrului ErrorMessage, o sa fie preluate in View pentru afisare.

In cazul in care in formularul de adaugare a unui nou articol nu se completeaza toate inputurile, o sa apara mesajele de eroare asociate, deoarece Title, Content si CategoryId sunt campuri obligatorii. In cazul in care titlul are o lungime mai mica de 5 caractere, atunci o sa apara mesajul de eroare asociat.

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

Adaugare articol

Titlu Articol

Titlul este obligatoriu

Continut Articol

Continutul articolului este obligatoriu

Selectati categoria

The value " " is invalid.

Adauga articol

Pentru **Titlu** și **Continut** mesajele de eroare se afișează corect. În cazul ultimului input, dropdown-ul în care se selectează categoria, mesajul de eroare nu este cel scris în Modelul Article.

Se întâmplă acest lucru deoarece CategoryId este cheie externă. Cheia externă poate fi și null, acest lucru ducând la prezenta acelui mesaj de validare, mesaj implementat by default. Chiar dacă acest atribut este obligatoriu, în implementarea lui internă se consideră că poate fi și null. În momentul în care este null, se afișează automat mesajul de validare intern “The value is invalid”. În acest caz, este nevoie ca acel câmp să fie optional la nivel de Model, astfel încât în momentul în care inputul nu are o valoare selectată să preia corect mesajul configurat la nivel de Model și să paseze acel mesaj pentru afișare în View.

În același mod trebuie implementate și proprietățile aflate în Modelul Article deoarece proprietatea Category, de exemplu, nu primește o valoare exact în momentul în care pentru un articol se selectează categoria din care face parte. Acest pas are loc după asocierea unui articol cu o categorie. Altfel, și în cazul proprietăților este nevoie ca acestea să fie declarate *optional* la nivel de Model.

Codul devine:

```
...
[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; }
...
```

Adaugare 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 să aibă mai mult de 5 de caractere



Continut Articol

Continutul articolului este obligatoriu

Selectați categoria

Categoria este obligatorie

Preluarea validărilor în View

Pentru preluarea validărilor și afișarea mesajelor asociate în View, se utilizează Helper-ul `@Html.ValidationMessageFor` astfel:

- Primul parametru este o lambda expresie care **selectează atributul modelului** pentru care se va afișa mesajul de validare
- Al doilea parametru este un string și reprezintă mesajul de validare afișat pe ecran

În cazul în care acesta este gol sau null se va afișa **mesajul de validare aflat în Model** ca parametru al variabilei ErrorMessage.

În cazul în care acest parametru are o valoare, atunci o să fie suprascris mesajul din Model cu cel primit ca parametru în cadrul Helper-ului.

În cazul în care în Model nu există un mesaj configurat pentru o anumită validare, iar în View parametrul este null, atunci se va afișa un mesaj default, generat de framework.

- Al treilea parametru este opțional și reprezintă o listă de atribute care poate fi adăugată mesajului afișat

Exemplu de implementare la nivel de View

```
@using (Html.BeginForm(actionName: "New", controllerName:
"Articles"))
{
    @Html.Label("Title", "Titlu Articol")
    <br />
    @Html.TextBox("Title", null, new { @class = "form-control"
})

    @Html.ValidationMessageFor(m => m.Title, null, new {
@class = "text-danger" })
    <br /><br />

    @Html.Label("Content", "Continut Articol")
    <br />
    @Html.TextArea("Content", null, new { @class = "form-
control" })

    @Html.ValidationMessage("Content", null, new { @class =
"text-danger" })
    <br /><br />

    <label>Selectati categoria</label>
    @Html.DropDownListFor(m => m.CategoryId, new
SelectList(Model.Categ, "Value", "Text"),
"Selectati categoria", new { @class = "form-control" })

    @Html.ValidationMessageFor(m => m.CategoryId, null, new {
@class = "text-danger" })
    <br /><br />

    <button class="btn btn-success" type="submit">Adauga
articol</button>
}
```

Helper-ul @Html.BeginForm

Se poate observa utilizarea unui nou Helper → `Html.BeginForm`

Acest Helper inlocuieste tagul `<form>` utilizat pana in acest moment in cazul formularelor din View.

Implementarea unui formular utilizand Helper-ul:

```
@using (Html.BeginForm (actionName: "NumeMetoda",
    controllerName: "NumeController",
    method: RequestMethod.NumeVerbHttp,
    routeValues: new { id = Model.NumeId })))
```

In cazul in care se utilizeaza Helper-ul si este nevoie de trimiterea unui id catre Controller, exista urmatoarele variante de implementare:

1. Trimiterea id-ului prin intermediul Helper-ului `Html.BeginForm` cu ajutorul parametrului `routeValues`, unde Model este Modelul primit ca parametru din Controller si inclus in View prin intermediul Helper-ului `@model` → `@model NumeProiect.Models.NumeClasa` (EX din laborator: `@model ArticlesApp.Models.Article`)

```
@using (Html.BeginForm(actionName: "New", controllerName:
    "Articles", routeValues: new { id = Model.Id })))
```

2. Trimiterea id-ului in interiorul formularului prin intermediul Helper-ului `@Html.HiddenFor`

`@Html.HiddenFor(m => m.Id)` -> unde m este un parametru de tipul Modelului primit ca parametru din Controller si inclus in View la fel ca in exemplul anterior

@Html.ValidationMessage si @Html.ValidationMessageFor

!OBSERVATIE

Implementarile urmatoare sunt echivalente, acest lucru fiind valabil in cazul tuturor Helperelor:

```
@Html.ValidationMessageFor(m => m.Title, null, new { @class  
= "text-danger" })
```

si

```
@Html.ValidationMessage("Title", null, new { @class =  
"text-danger" })
```

Helper-ul @Html.ValidationSummary

Helperul @Html.ValidationSummary ofera posibilitatea afisarii unui sumar cu toate erorile aparute in timpul validarii.

Acesta se adauga in formularul din cadrul View-ului:

```
@Html.ValidationSummary(false, "", new { @class = "text-  
danger" })
```

Pentru afisarea corecta a mesajelor de eroare, doar in momentul in care validarea datelor nu este corecta, este necesar sa adaugam urmatoarele linii de cod in fisierul **site.css** aflat in wwwroot → folderul css

```
.field-validation-valid {  
    display: none;  
}  
  
.validation-summary-valid {  
    display: none;  
}
```

Este necesar sa procedam asa doar in momentul in care mesajele de validare se afla doar in View, in cadrul parametrului doi.

```
@Html.ValidationMessageFor(m => m.Title, "Titlul este obligatoriu", new { @class = "text-danger" })
```

Vizualizarea mesajelor de validare in browser

Se pot observa mesajele de validare preluate din cadrul Modelului Article, atat la nivelul fiecarui input prin intermediul Helper-ului `@Html.ValidationMessageFor`, dar si mesajele preluate prin intermediul Helper-ului `@Html.ValidationSummary`

Adaugare articol



The diagram illustrates the flow of validation messages. A green curved arrow points from the `@Html.ValidationSummary` box to a list of validation messages. Another green curved arrow points from the list of messages to the individual input fields. A third green curved arrow points from the `@Html.ValidationSummary` box to the individual input fields.

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

`@Html.ValidationSummary`

Titlu Articol

Titlul este obligatoriu

Continut Articol

Continutul articolului este obligatoriu

Selectati categoria

Selectati categoria

Categoria este obligatorie

Adauga articol

În cazul în care există un mesaj de validare în View, acesta o să suprascrie mesajul configurat la nivel de Model, după cum urmează:

Mesajul de validare din View-ul New:

```
@Html.TextBox("Title", null, new { @class = "form-control"
})

@Html.ValidationMessageFor(m => m.Title, "OBLIGATORIU", new
{ @class = "text-danger" })
```

Mesajul de validare din Modelul Article:

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

Se poate observa faptul că mesajul de validare din View îl suprascrie pe cel din Model doar la nivelul inputului. În cazul în care se realizează sumarul cu toate mesajele de validare, sunt preluate tot mesajele de validare existente la nivel de Model.

De aceea este recomandată utilizarea mesajelor de validare în Model.

În cazul în care sunt utilizate validări, fără scrierea mesajelor specifice, framework-ul o să afișeze mesaje default.

De exemplu: “The Title field is required.”

“The field Title must be a string or array type with a minimum length of '5'.”

Adaugare articol

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

Titlu Articol

OBLIGATORIU ←

Continut Articol

Continutul articolului este obligatoriu

Selectati categoria

Categoria este obligatorie

Adauga articol

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

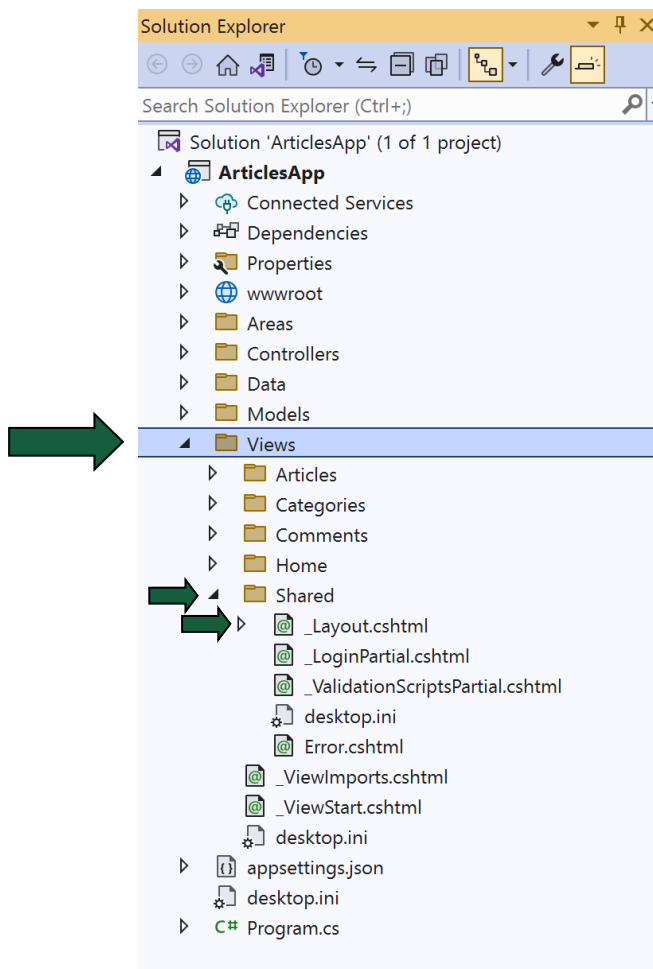
Interfața unei aplicații, indiferent de tehnologia cu care este realizată, întotdeauna o să conțină foarte multe componente comune tuturor paginilor: Header, Footer, Meniuri, etc. Aceste componente nu se modifică de la o pagină la alta, iar repetarea scrierii aceluiași cod devine redundantă. Pentru a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

Layout View

Layout View – permite scrierea unui cod comun pentru toate paginile, cat si un Placeholder in care se va include continutul celorlalte pagini. Acest placeholder este definit prin intermediul variabilei **@RenderBody()**. Locul in care este plasata aceasta variabila in Layout, va fi locul in care se va afisa continutul View-urilor aferente.

De exemplu, in momentul in care cream un nou proiect, acesta genereaza in mod automat un layout care include toate resursele necesare: Head, Stiluri CSS, JavaScript, Header, Footer, etc. In acest layout se afla metoda **RenderBody()** prin care toate View-urile create sunt incluse.



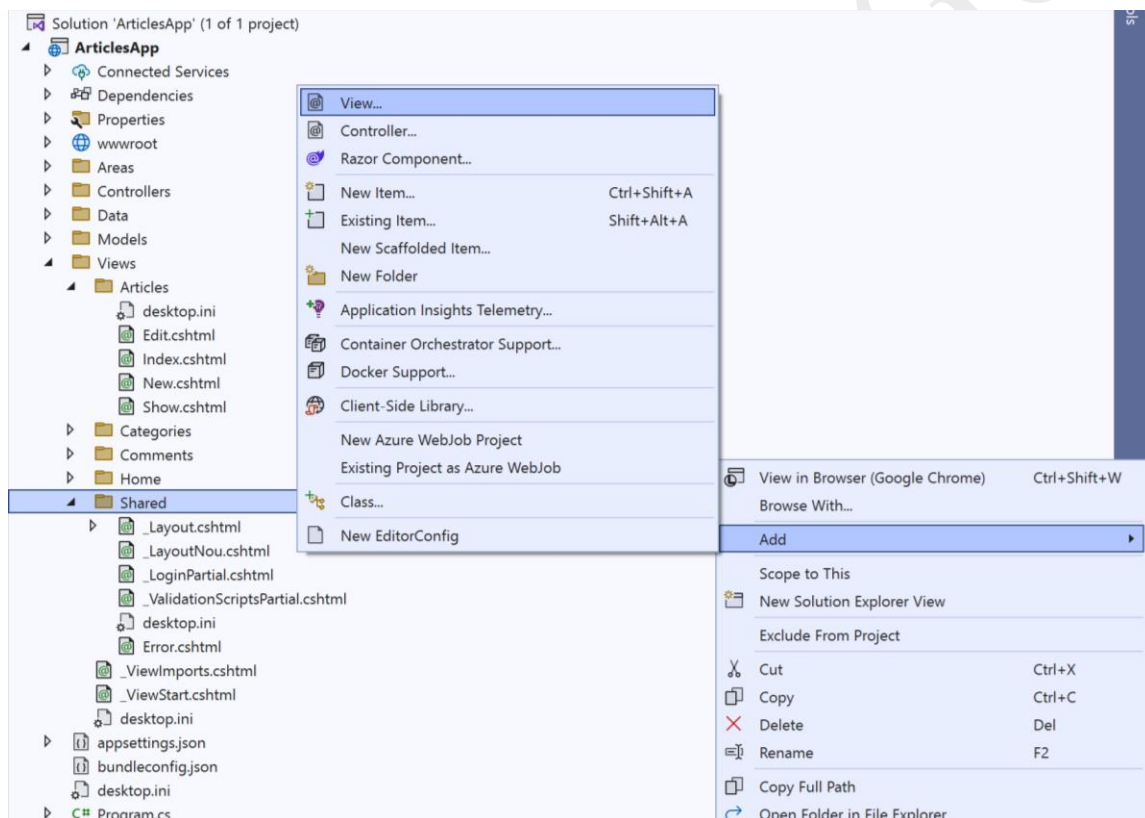


Adaugarea unui nou Layout

Implicit, proiectul ASP.NET Core are in folderul Shared din View un Layout. Acest Layout este utilizat in cadrul tuturor paginilor existente in proiect. In momentul in care se doreste utilizarea unui alt Layout, se poate crea si utiliza dupa cum urmeaza:

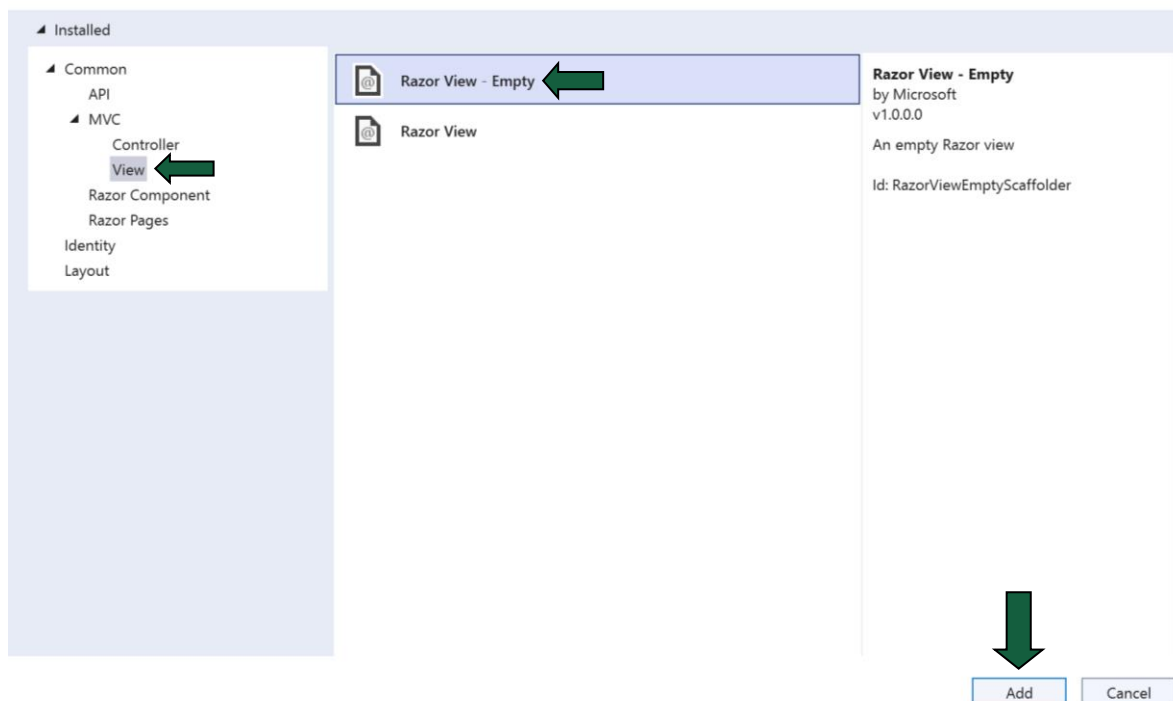
PASUL 1 – crearea noului Layout

Views → Shared → click dreapta → Add → View

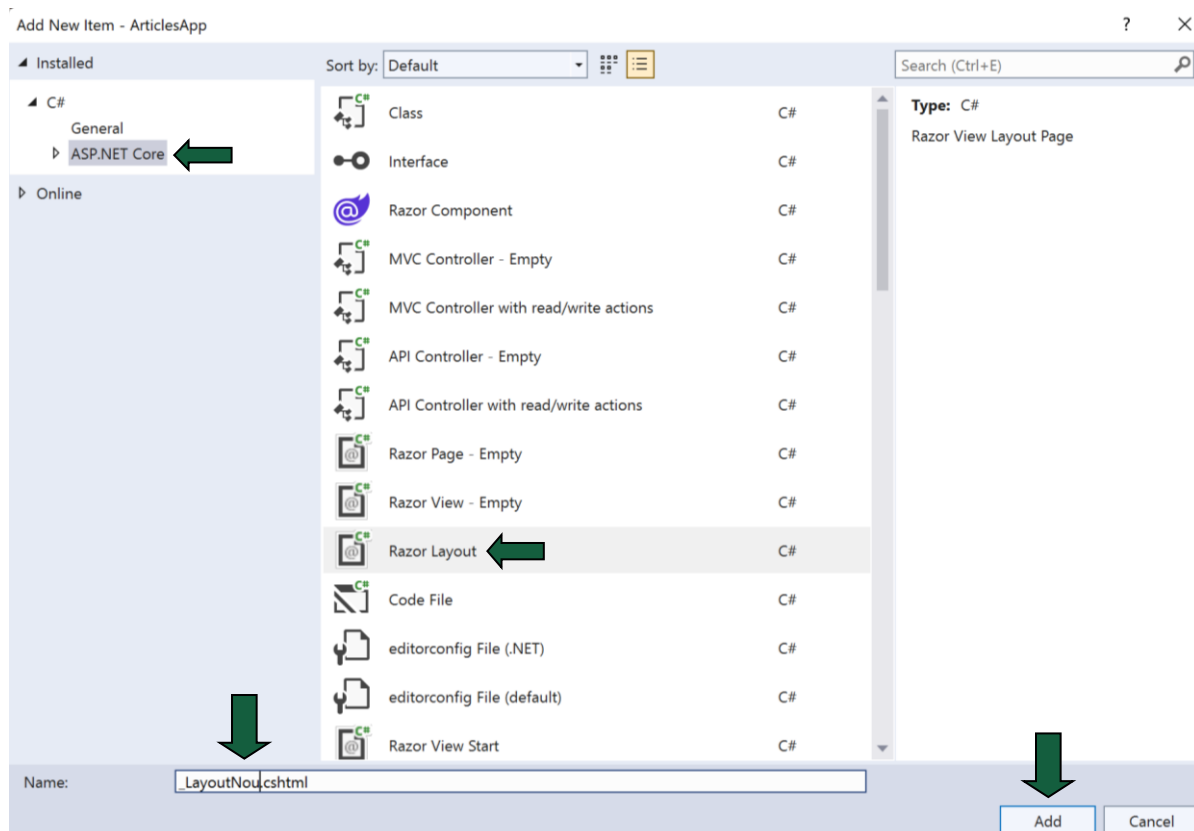


Se selecteaza Razor View – Empty

Add New Scaffolded Item



Se selecteaza ASP.NET Core → Razor Layout
 Se redenumeste Layout-ul → Add

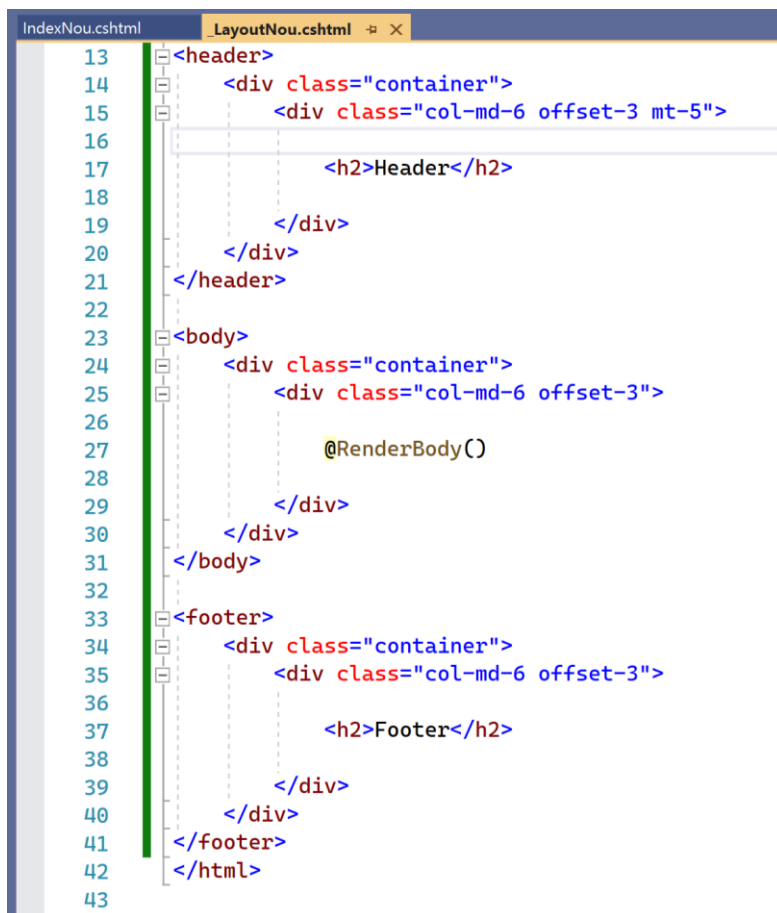


View-ul generat de framework o sa arate astfel:

```

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, adaugand header si footer

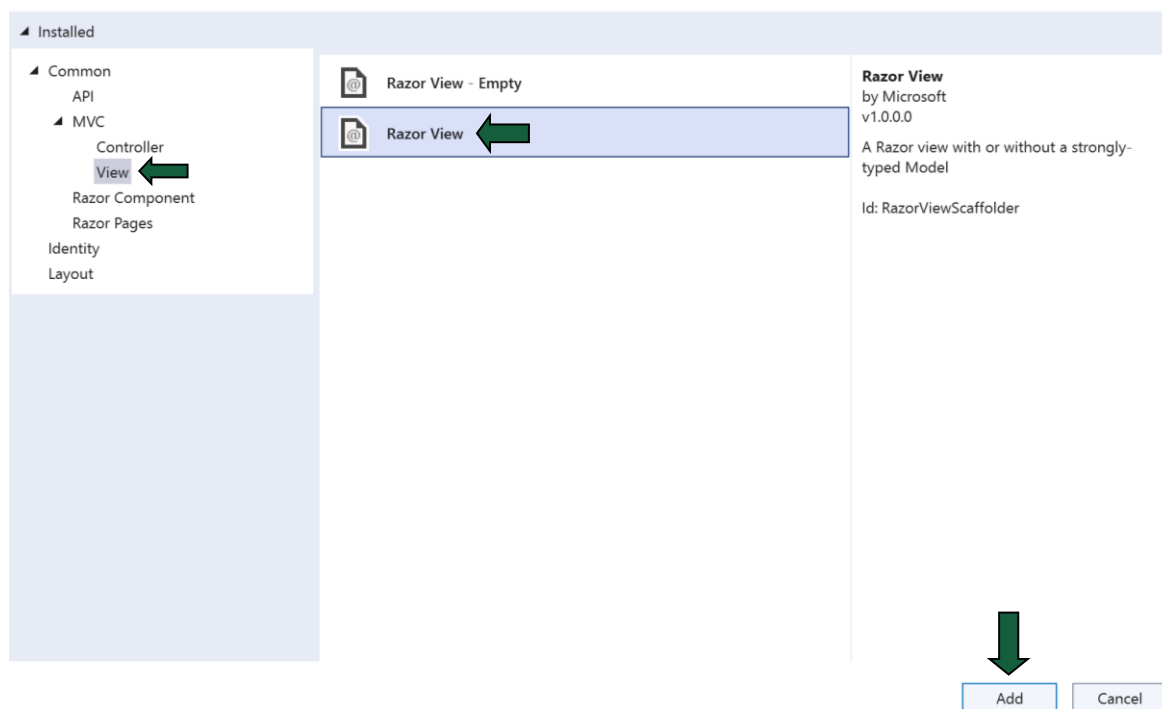


```
13 <header>
14   <div class="container">
15     <div class="col-md-6 offset-3 mt-5">
16       <h2>Header</h2>
17     </div>
18   </div>
19 </header>
20
21 <body>
22   <div class="container">
23     <div class="col-md-6 offset-3">
24       @RenderBody()
25     </div>
26   </div>
27 </body>
28
29 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

PASUL 3 – adaugarea unui nou View care o sa aiba ca Layout noul Layout creat:

Se adauga un nou View de tipul Razor View

Add New Scaffolded Item



Se adauga un nume pentru respectivul View, dupa care se alege Layout-ul

Add Razor View

View name: IndexNou

Template: Empty (without model)

Model class:

Data context class:

Options

☐ Create as a partial view

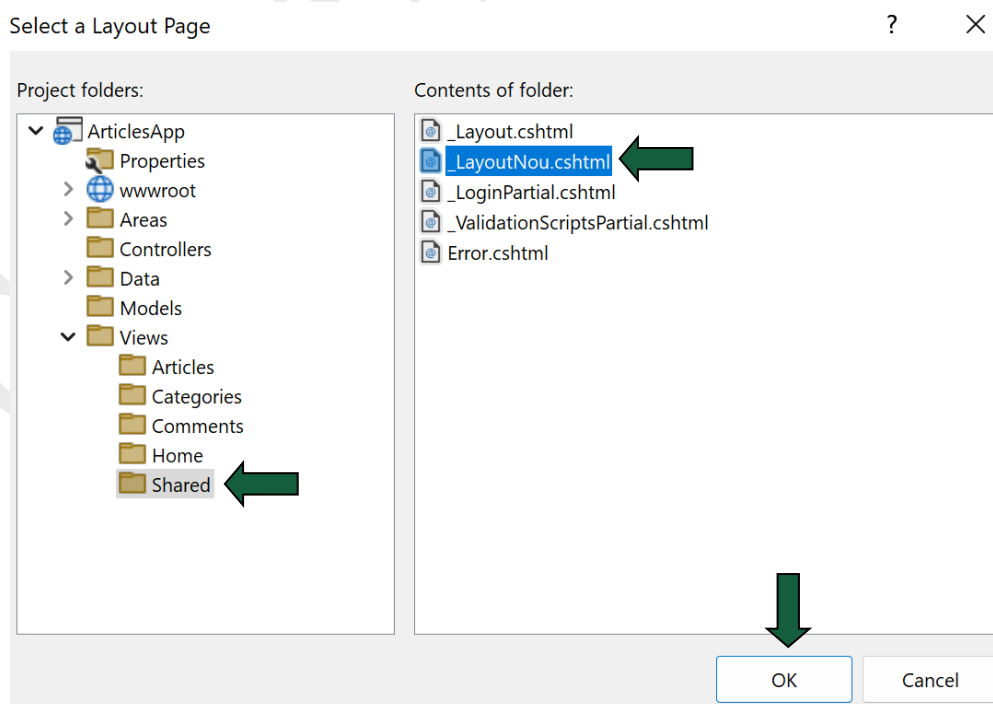
☒ Reference script libraries

☒ Use a layout page

(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

Se selecteaza Layout-ul



In acest pas se observa Layout-ul pe care tocmai l-am selectat

Add Razor View

View name: IndexNou

Template: Empty (without model)

Model class:

Data context class:

Options

☐ Create as a partial view

☒ Reference script libraries

☒ Use a layout page

~/Views/Shared/_LayoutNou.cshtml

(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

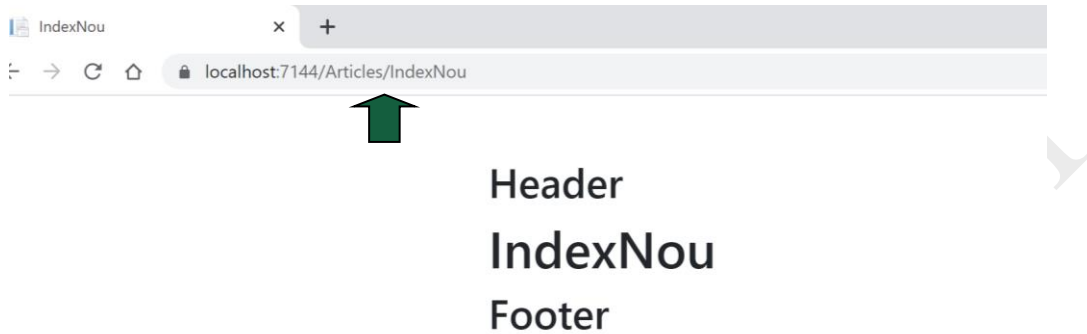
Dupa executarea pasilor anteriori, se poate observa cum in cadrul View-ului numit *IndexNou*, Layout-ul numit *LayoutNou* se include prin intermediul parametrului *Layout* care primeste ca valoare calea din sistemul de fisiere. Mai jos se poate observa secventa de cod generata:

```

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

```


Dupa rulare si 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 reprezinta secvente de cod specifice View-urilor care pot fi refolosite in una sau mai multe pagini. In cadrul dezvoltarii aplicatiilor web, codul poate fi reutilizat pentru a optimiza timpul de scriere si pentru a nu include acelasi cod in mod repetitiv.

Secventele de cod care se repeta in cadrul mai multor pagini pot fi incluse intr-un View sau intr-un Layout pentru a fi afisate.

In cadrul aplicatiei dezvoltate in laborator, afisarea unui articol contine acelasi cod, atat in cazul afisarii tuturor articolelor din baza de date → View-ul Index, cat si in cazul afisarii unui singur articol → View-ul Show. In acest caz, pentru a elimina redundanta, si anume scrierea repetitiva a aceluasi cod in cadrul ambelor View-uri, o sa utilizam un **Partial View**, dupa cum urmeaza:

Secventa de cod, cu ajutorul careia se afiseaza informatiile corespunzatoare unui articol, se afla inclusa in ambele View-uri, *Index.cshtml* si *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>

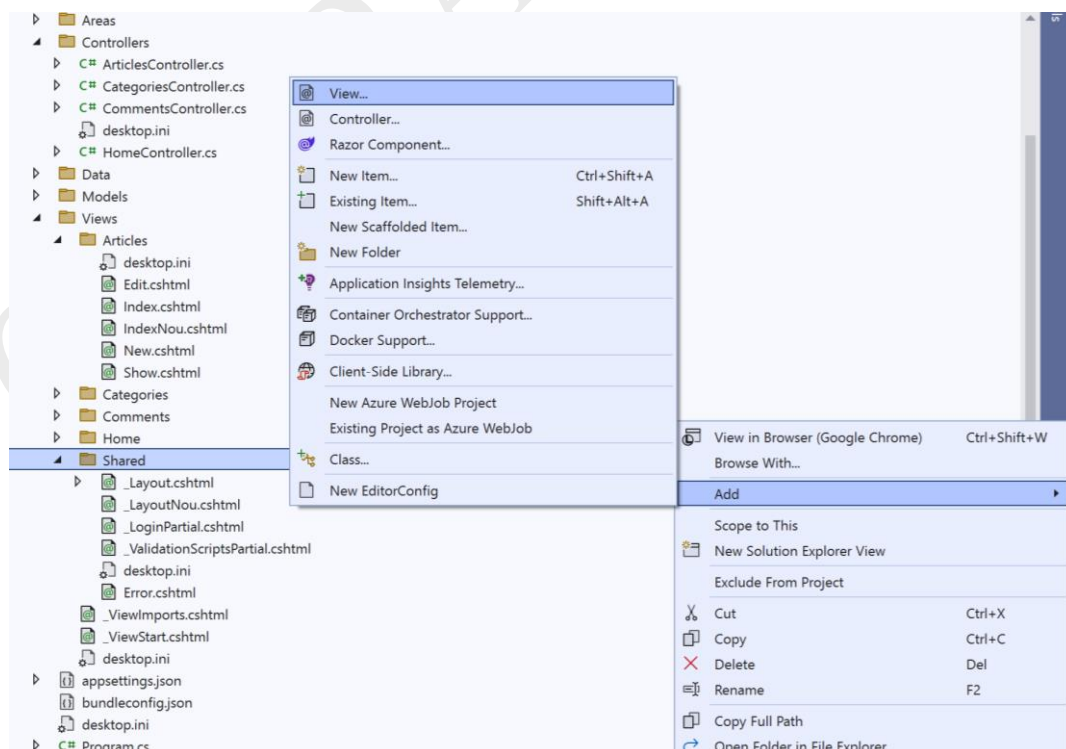
    <div class="d-flex justify-content-between flex-row mt-5">

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

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

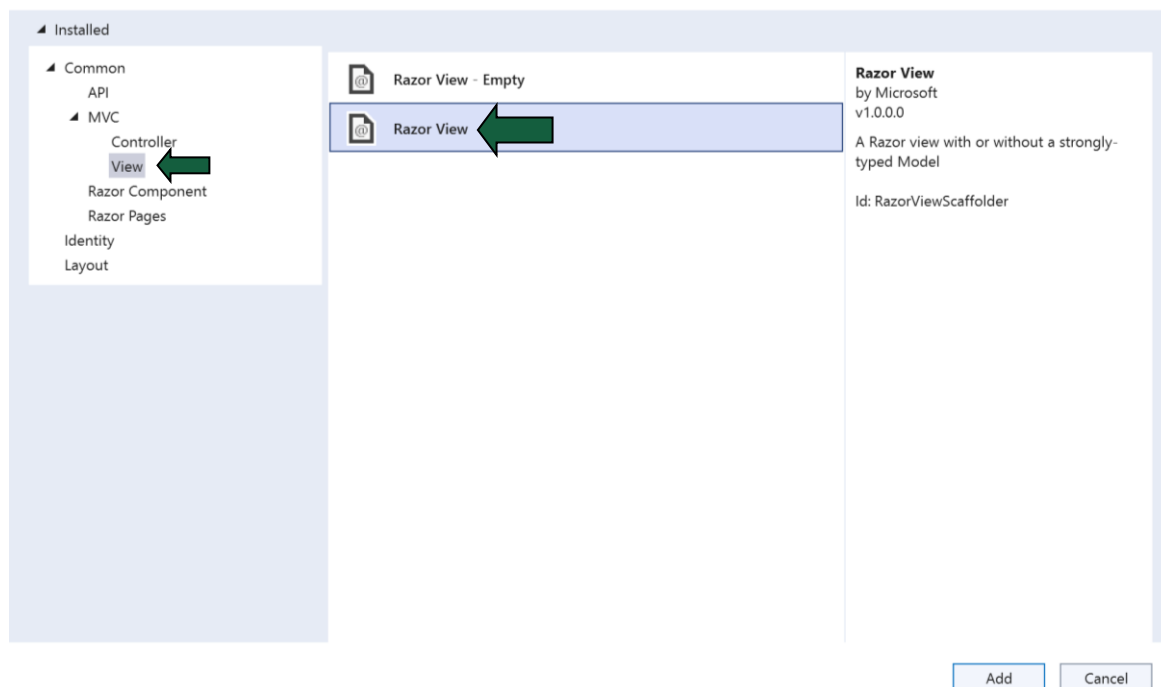
    </div>
</div>
```

Pentru eliminarea redundantei, se adauga un nou View de tip Partial View. Acest View o sa contina secventa de cod care se repeat.



Se selecteaza urmatoarele optiuni:

Add New Scaffolded Item



View-ul o sa fie de tipul Partial View, bifand optiunea
“Create as a partial view”

Add Razor View

View name: ←

Template:

Model class:

Data context class:

Options

☒ Create as a partial view ←

☐ Reference script libraries

☒ Use a layout page

...

(Leave empty if it is set in a Razor _viewstart file)

View-ul numit *“ArticleInfo”* o sa contina secventa de cod care se repeta in ambele View-uri, atat in Index.cshtml, cat si in Show.cshtml.

```

ArticleInfo.cshtml  Index.cshtml  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 partialului se utilizeaza Helper-ul specific `@Html.Partial` sau tag-ul `<partial></partial>`

Exemplu utilizand aplicatia dezvoltata in cadrul laboratorului:

Pentru apelarea partialului din **View-ul Show** se utilizeaza pentru primul parametru numele partialului, iar pentru al doilea parametru **Modelul**.

```
@Html.Partial("ArticleInfo", Model)
```

Pentru apelarea partialului din **View-ul Index** se utilizeaza pentru primul parametru numele partialului, iar pentru al doilea parametru obiectul **article** de tipul **Model**. Astfel, in loop trebuie sa declaram tipul modelului si sa pasam acest parametru la partial. Prin intermediul acestui cod, in partial putem sa folosim variabila **@Model** pentru afisarea datelor.

```
@foreach (ArticlesApp.Models.Article article in
 ViewBag.Articles)
{
    ...
    @Html.Partial("ArticleInfo", article)
    ...
}
```

In cazul modificarii partialului, modificarile se reflecta asupra tuturor View-urilor care utilizeaza partialul respectiv, nefiind nevoie de modificari in fiecare View in parte. Acest lucru eficientizeaza atat timpul de scriere a codului, cat si mentenanta ulterioara a acestuia in cazul in care apar modificari in timp.