Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC
Curs 8

Cuprins

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 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.

Pentru preluarea mesajelor de validare in View **VEZI Sectiunea urmatoare din curs – Preluarea Validarilor in View**. Dupa adaugarea validarilor in Model, urmeaza preluarea mesajelor de validare in View.

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 urmatoarea problema in cazul in care se incearca adaugarea unui articol fara completarea campurilor:

Pentru **Titlu** si **Continut** mesajele de eroare se afiseaza corect. In cazul ultimului input, dropdown-ul in care se selecteaza categoria, mesajul de eroare nu este cel scris in Modelul Article.

Se intampla acest lucru deoarece **CategoryId** este cheie externa. Cheia externa poate fi si null, acest lucru ducand la prezenta acelui mesaj de validare, mesaj implementat by default. Chiar daca acest atribut este obligatoriu, in implementarea lui interna se considera ca poate fi si null. In momentul in care este null, se afiseaza automat mesajul de validare intern “The value is invalid”.

In acest caz, este nevoie ca acel camp sa fie optional la nivel de Model, astfel incat in momentul in care inputul nu are o valoare selectata sa preia corect mesajul configurat la nivel de Model si sa paseze acel mesaj pentru afisare in View.

In acelasi mod trebuie implementate si proprietatile aflate in Modelul Article deoarece proprietatea Category, de exemplu, nu primeste o valoare exact in momentul in care pentru un articol se selecteaza categoria din care face parte. Acest pas are loc dupa asocierea unui articol cu o categorie. Asadar, si in cazul proprietatilor este nevoie ca acestea sa 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; }
...
```

In cazul in care Titlul este completat, dar are mai putin de 5 caractere, o sa se afiseze mesajul urmator de eroare, conform validarii.

Preluarea validarilor in View

Pentru preluarea validarilor si afisarea mesajelor asociate in View, se utilizeaza Helper-ul `@Html.ValidationMessageFor` astfel:

- Primul parametru este o lambda expresie care **selecteaza atributul modelului** pentru care se va afisa mesajul de validare

- Al doilea parametru este un string si reprezinta mesajul de validare afisat pe ecran

  In cazul in care acesta este gol sau null se va afisa **mesajul de validare aflat in Model** ca parametru al variabilei ErrorMessage.

  In cazul in care acest parametru are o valoare, atunci o sa fie suprascris mesajul din Model cu cel primit ca parametru in cadrul Helper-ului.
  In cazul in care in Model nu exista un mesaj configurat pentru o anumita validare, iar in View parametrul este null, atunci se va afisa un mesaj default, generat de framework.

- Al treilea parametru este optional si reprezinta o lista de atribute care poate fi adaugata mesajului afisat

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: FormMethod.NumeVerbHttp,
routeValues: new { id = Model.NumeId }))
```

In cazul in care se utilizeaza Helper-ul si este nevoie de trimiterea unui id catre Controller (de exemplu in cazul editarii) 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 paramentru 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: "Edit", 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:

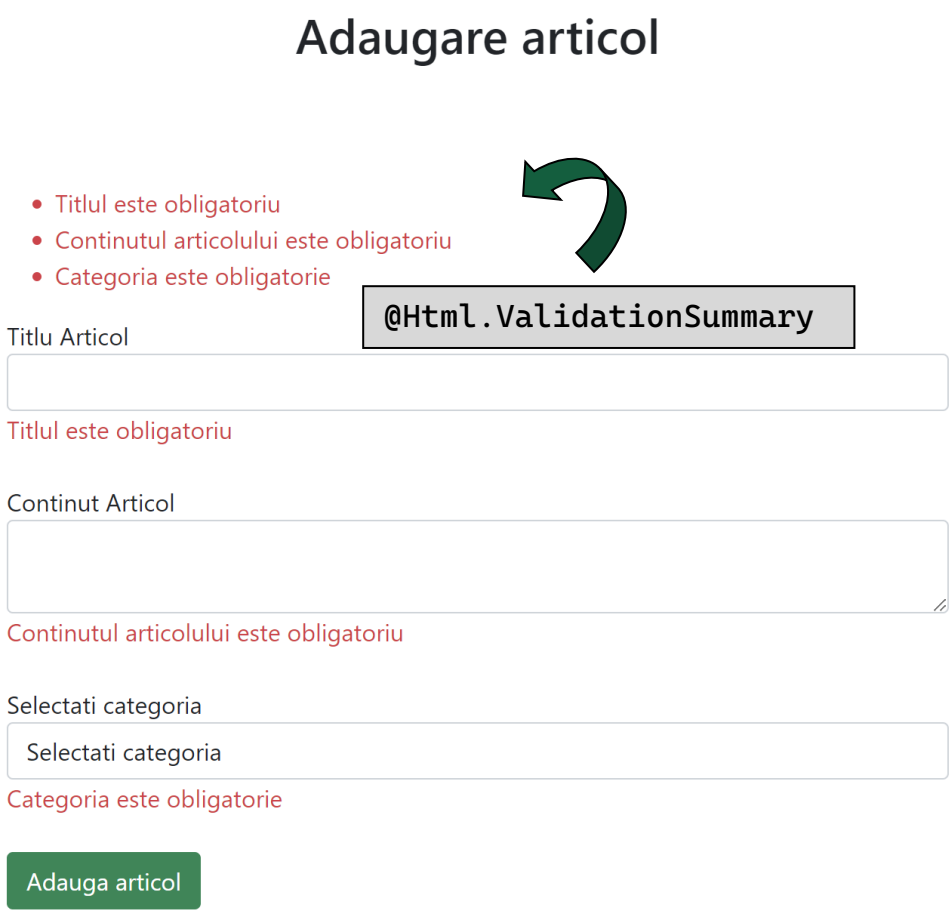
```
.field-validation-valid {
    display: none;
}
.validation-summary-valid {
    display: none;
}
```

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`



In cazul in care exista un mesaj de validare in View, acesta o sa suprascrie mesajul configurat la nivel de Model, dupa cum urmeaza:

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 ca mesajul de validare din View il suprascrie pe cel din Model doar la nivelul inputului. In cazul in care se realizeaza sumarul cu toate mesajele de validare, sunt preluate tot mesajele de validare existente la nivel de Model.

De aceea este recomandata utilizarea mesajelor de validare in Model.

In cazul in care sunt utilizate validari, fara scrierea mesajelor specifice, framework-ul o sa afiseze 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'.”

Implementarea validarilor la nivel de Controller

Pentru functionarea corecta a validarilor, cat si pentru identificarea corecta a datelor in partea de server (server-side) este necesar sa adaugam in Controller-ul care modifica datele, verificarea starii modelului. Astfel, prin intermediul variabilei **ModelState** putem sa aflam daca toate validarile 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);
	}
}
```

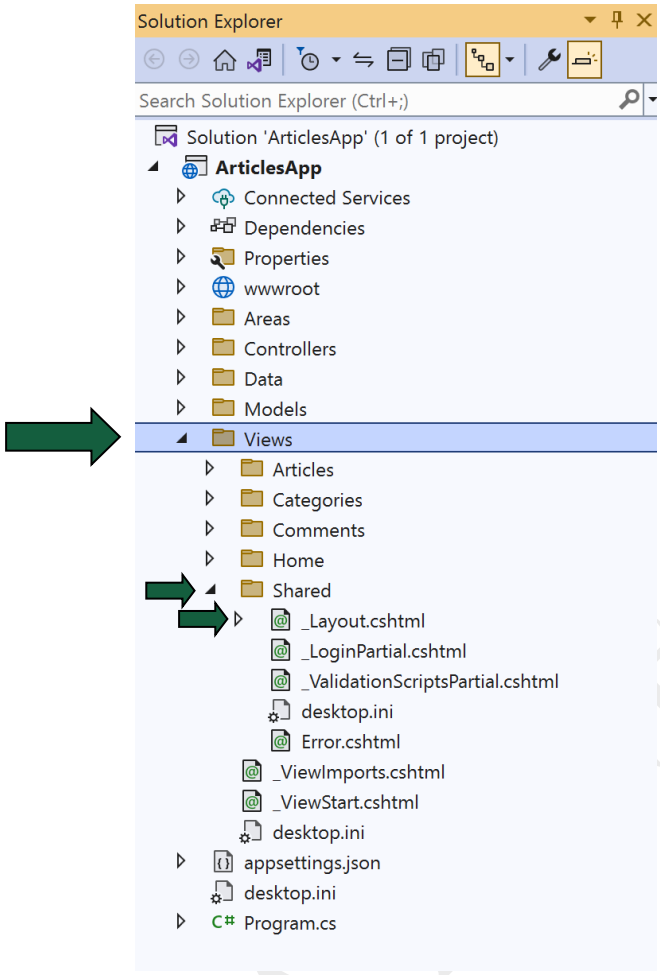
View-uri partajate

Interfata unei aplicatii, indiferent de tehnologia cu care este realizata, intotdeauna o sa contina foarte multe componente comune tuturor paginilor: Header, Footer, Meniuri, etc. Aceste componente nu se modifica de la o pagina la alta, iar repetarea scrierii aceluiasi cod devinde redundanta. 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.



Layout.cshtml

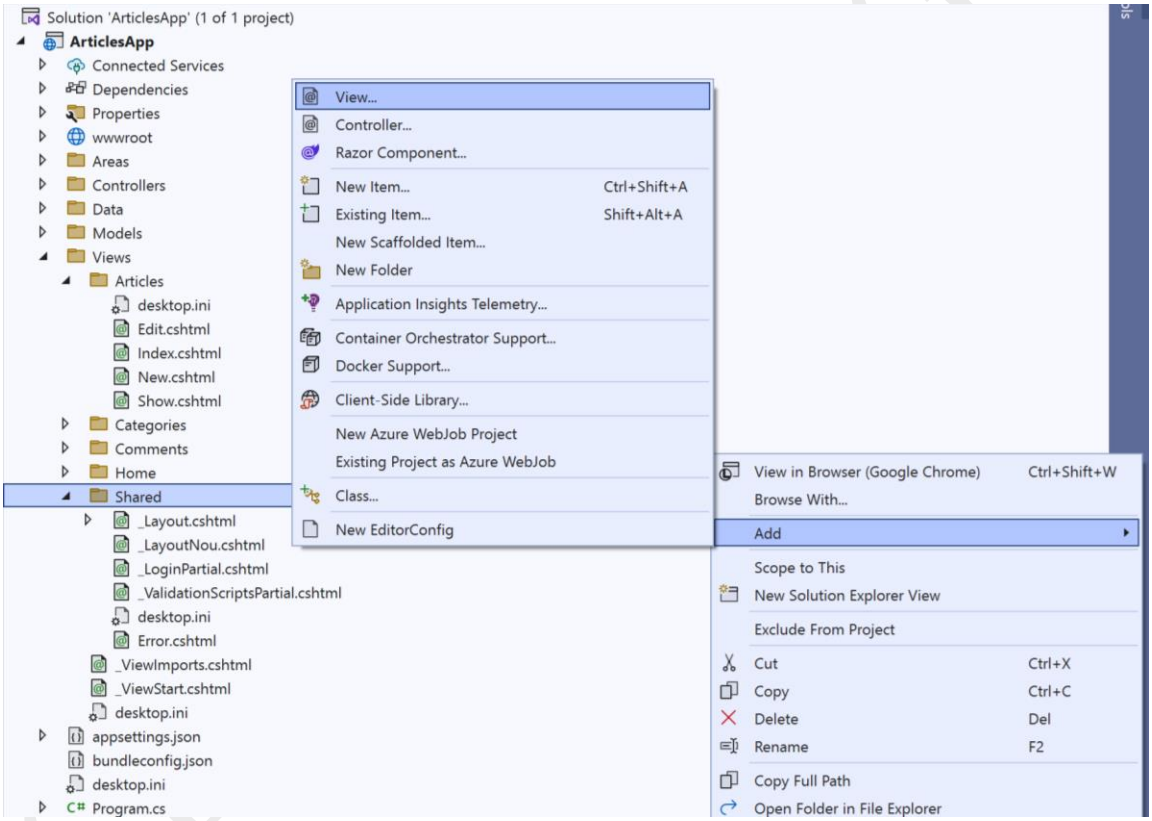
```
22                      <ul class="navbar-nav flex-grow-1">
23                          <li class="nav-item">
24                              <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-a
25                          </li>
26                          <li class="nav-item">
27                              <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          </nav>
34      </header>
35      <div class="container">
36          <main role="main" class="pb-3">
37              @RenderBody()
38          </main>
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>
```

Adaugarea unui nou Layout

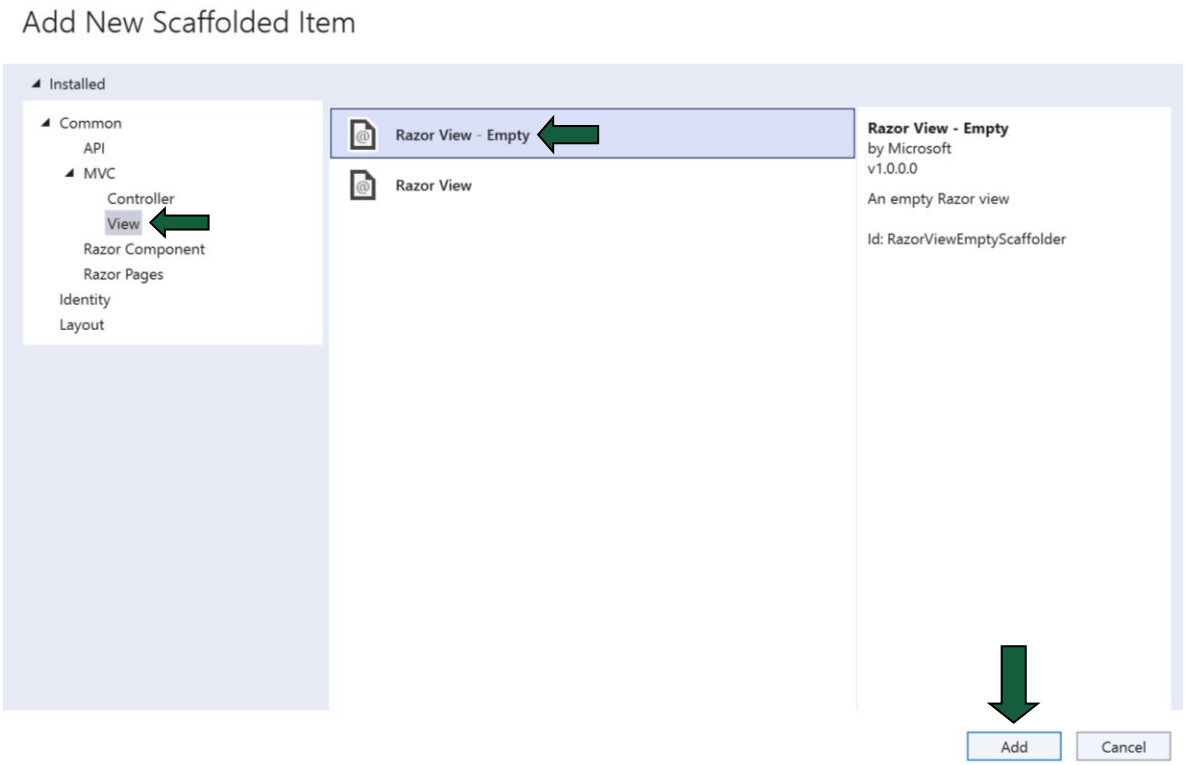
Implicit, proiectul ASP.NET Core are in folderul Shared din View un Layout. Acest Layout este utilizat in cadrul tuturor paginilor existente in proiect. In momentul in care se doreste utilizarea unui alt Layout, se poate crea si utiliza dupa cum urmeaza:

PASUL 1 – crearea noului Layout

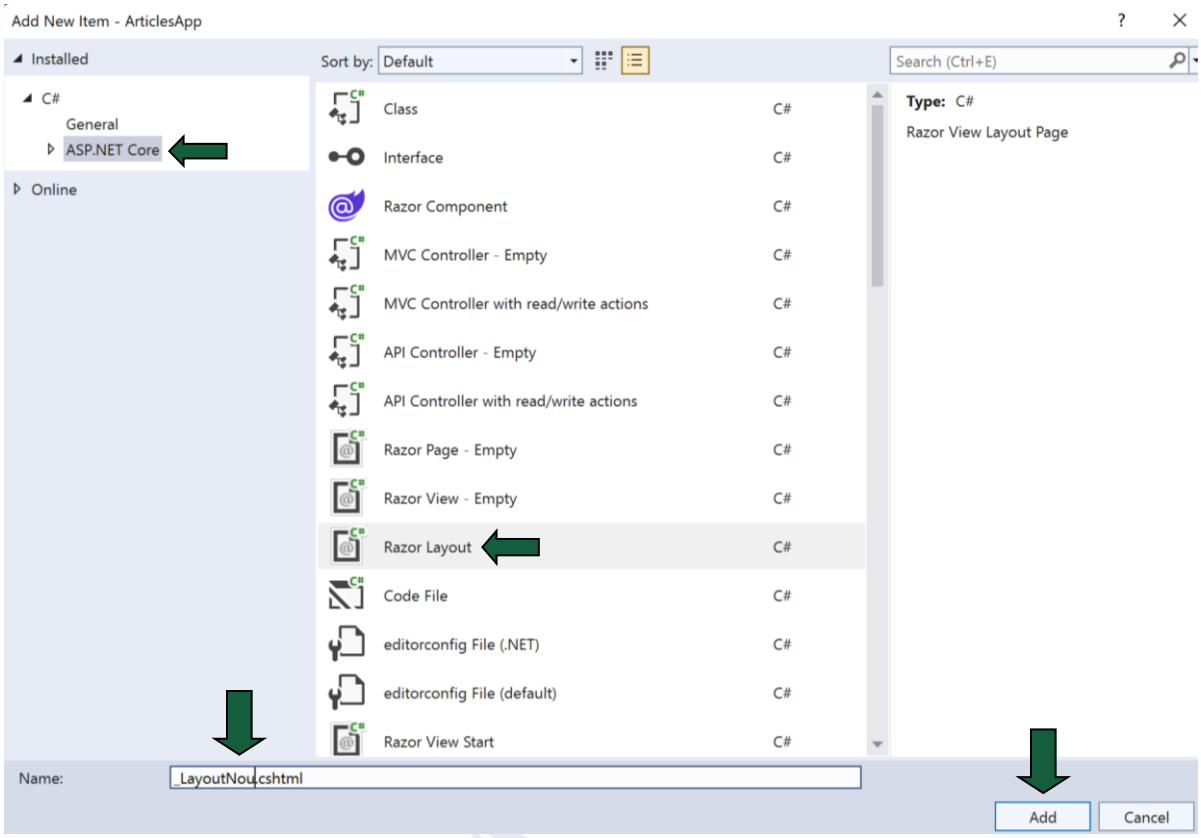
Views → Shared → click dreapta → Add → View



Se selecteaza Razor View – Empty



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



View-ul generat de framework o sa arate astfel:

_LayoutNou.cshtml

```
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
```

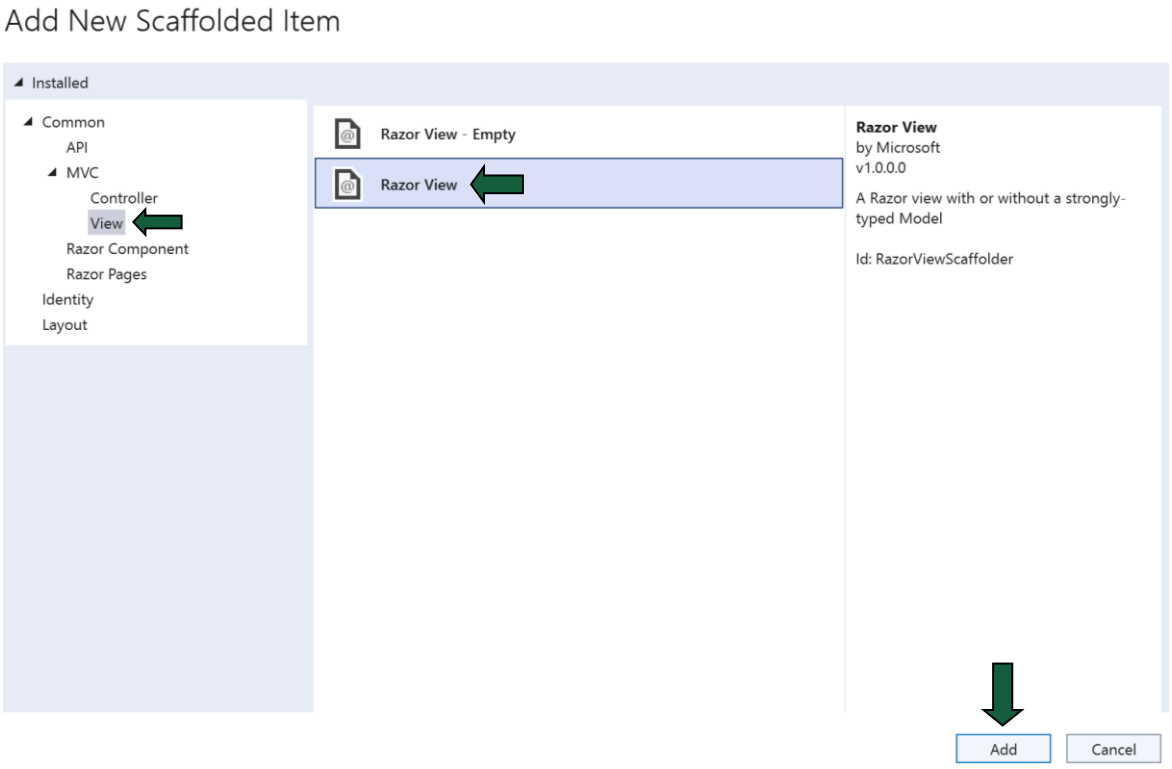
PASUL 2 – modificarea noului Layout, adaugand header si footer

IndexNou.cshtml _LayoutNou.cshtml

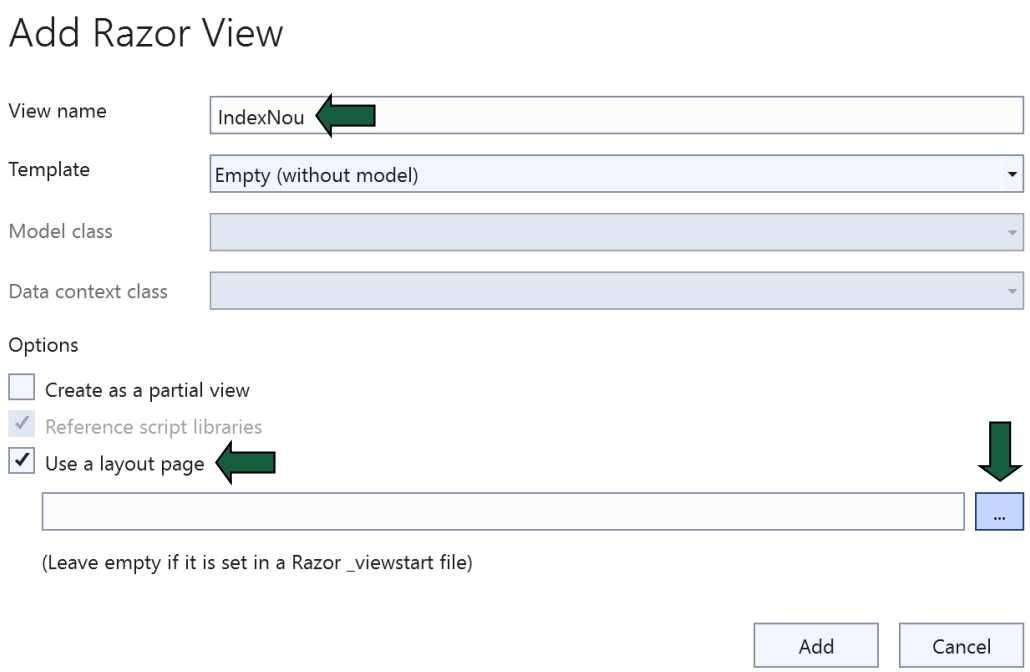
```
13  <header>
14      <div class="container">
15          <div class="col-md-6 offset-3 mt-5">
16
17              <h2>Header</h2>
18
19          </div>
20      </div>
21  </header>
22
23  <body>
24      <div class="container">
25          <div class="col-md-6 offset-3">
26
27              @RenderBody()
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  </html>
43
```

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

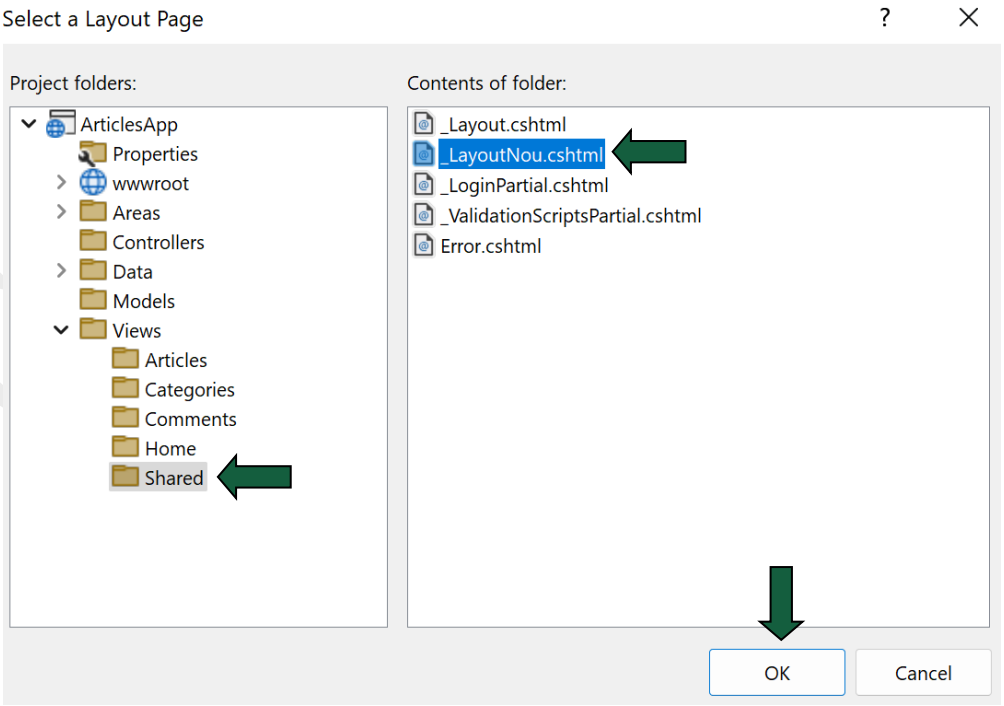
Se adauga un nou View de tipul Razor View



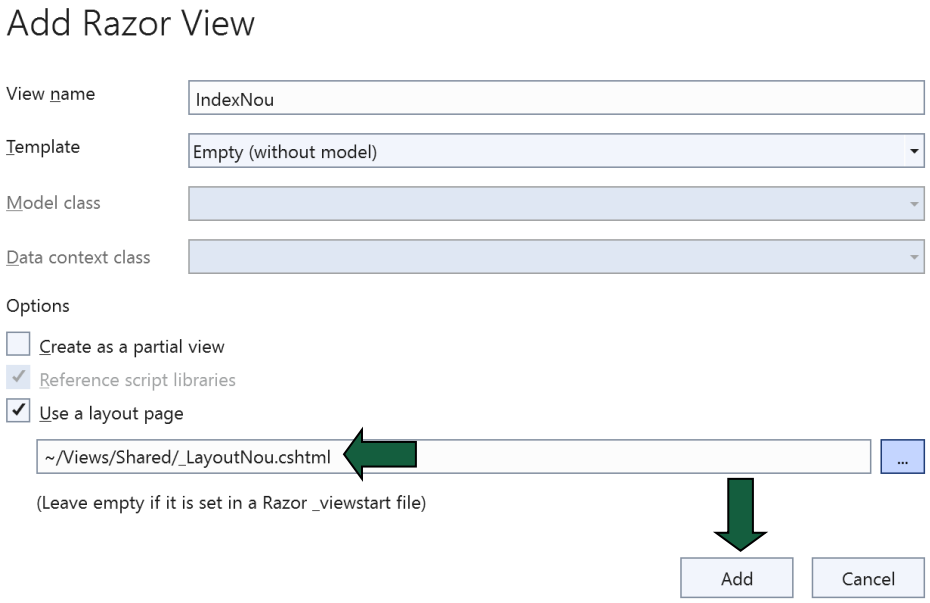
Se adauga un nume pentru respectivul View, dupa care se alege Layout-ul



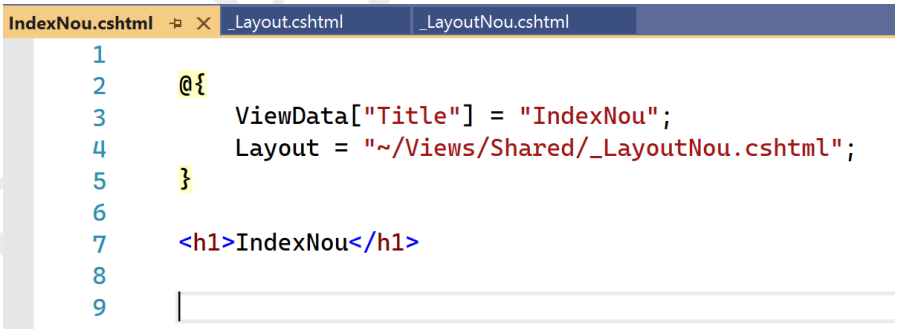
Se selecteaza Layout-ul



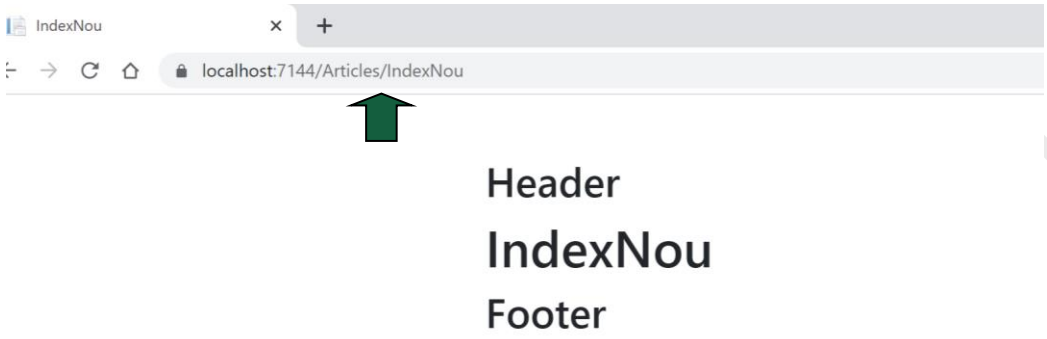
In acest pas se observa Layout-ul pe care tocmai l-am selectat



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:



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.

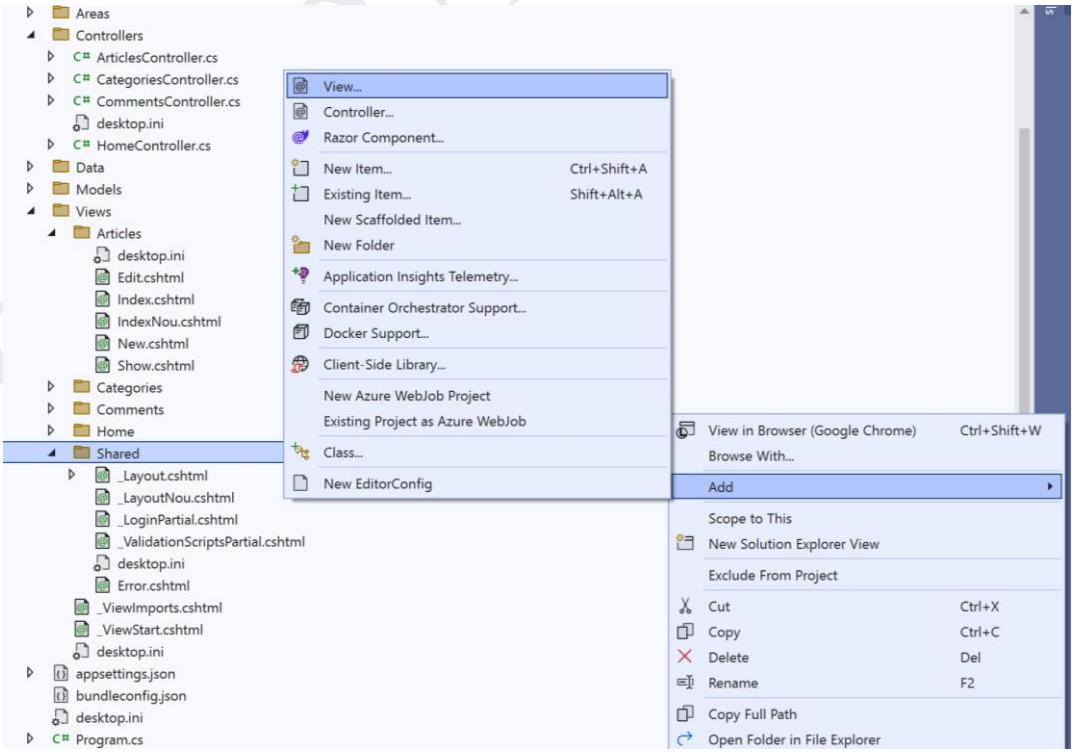
Secventele de cod care se repeta in cadrul mai multor pagini pot fi incluse intr-un View sau intr-un Layout pentru a fi afisate.

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 aceluiasi cod in cadrul ambelor View-uri, o sa utilizam un **Partial View**, dupa cum urmeaza:

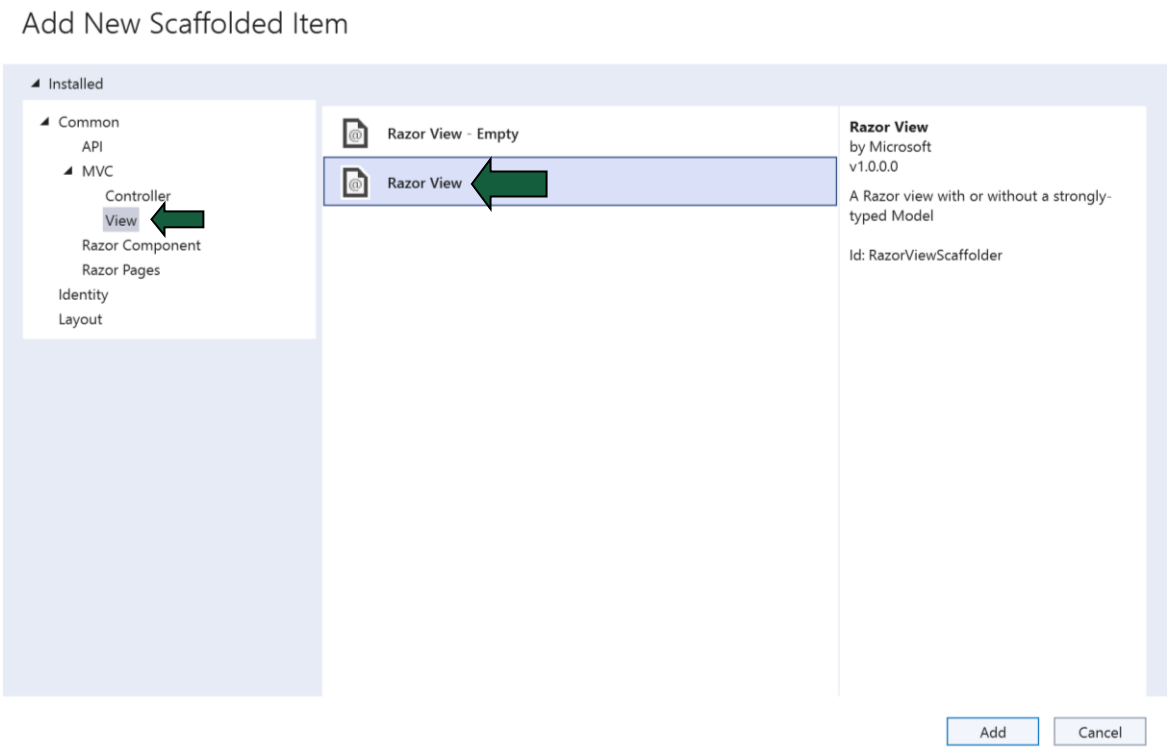
Secventa de cod, cu ajutorul careia se afiseaza informatiile corespunzatoare unui articol, se afla inclusa in ambele View-uri, *Index.cshtml* si *Show.cshtml*.

```
<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>
```

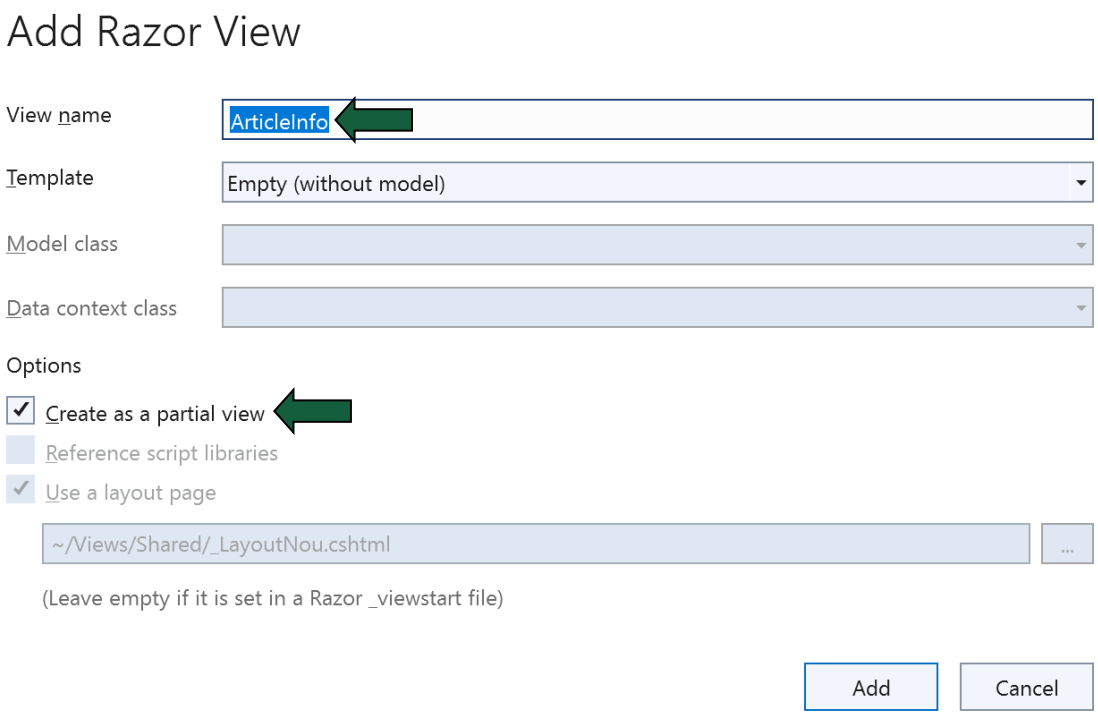
Pentru eliminarea redundantei, se adauga un nou View de tip Partial View. Acest View o sa contina secventa de cod care se repeat.



Se selecteaza urmatoarele optiuni:



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



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 paramentru **Modelul**.

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

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