Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC
Curs 7

Cuprins

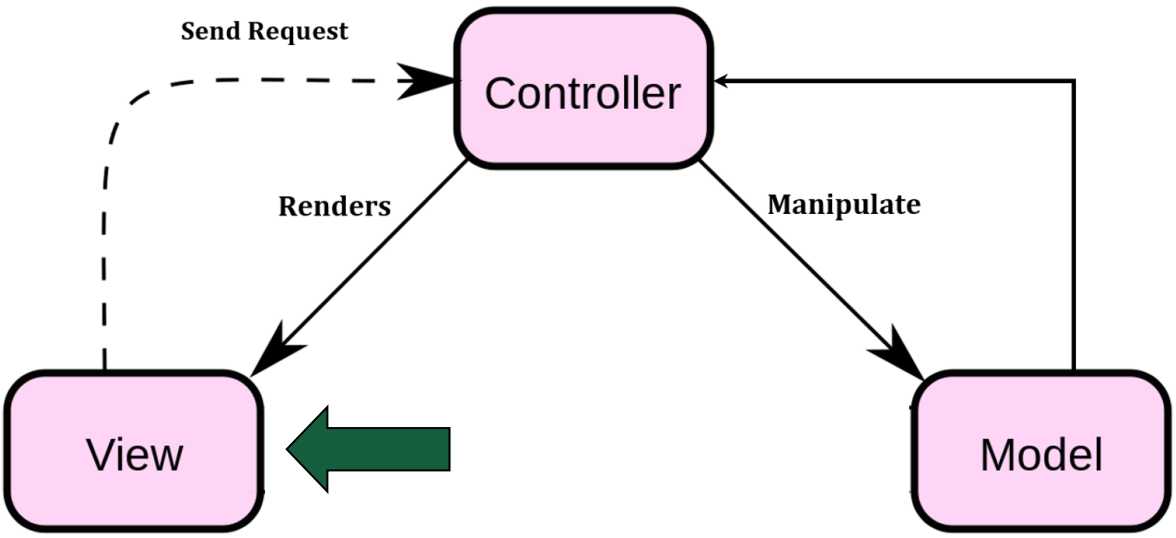
View (interfata cu utilizatorul)

Ce este View-ul

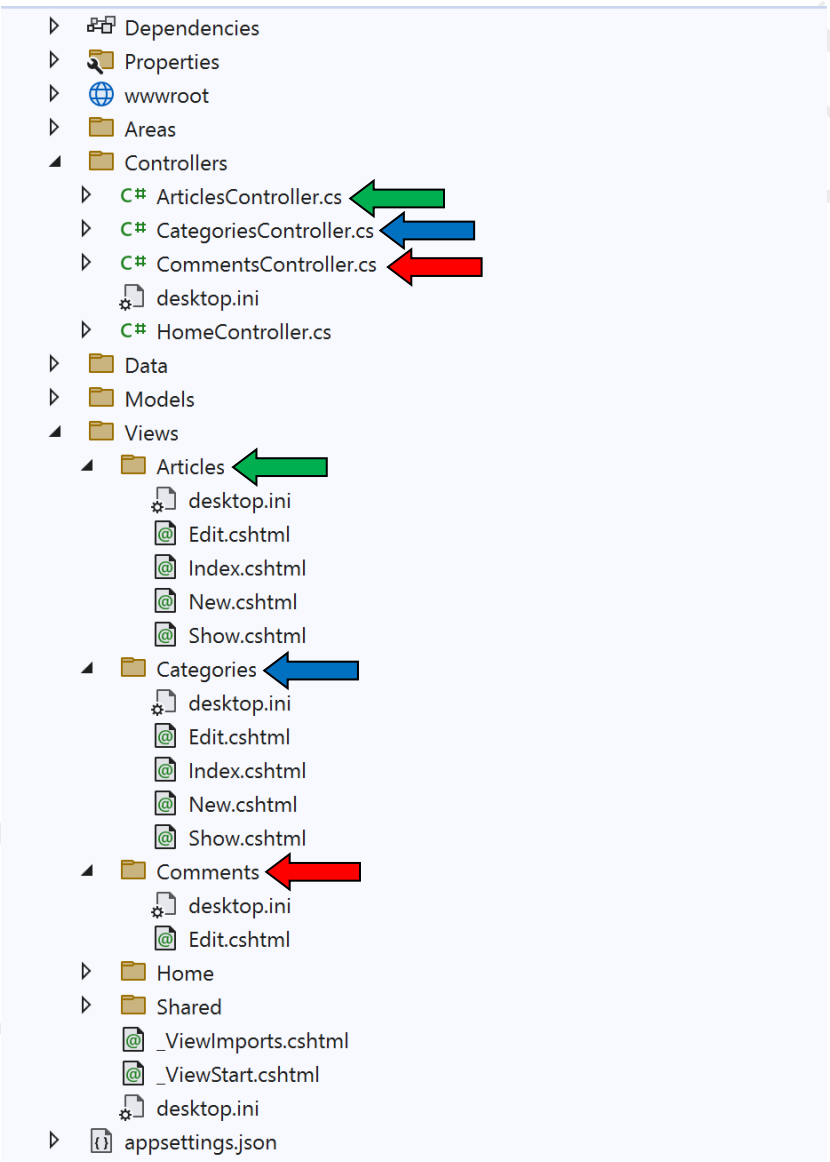
View-ul reprezinta interfata cu utilizatorul, fiind componenta arhitecturii MVC cu care utilizatorii interactioneaza prin intermediul browser-ului.

In View se afiseaza datele, adica inregistrarile din baza de date si informatiile generate de aplicatie.

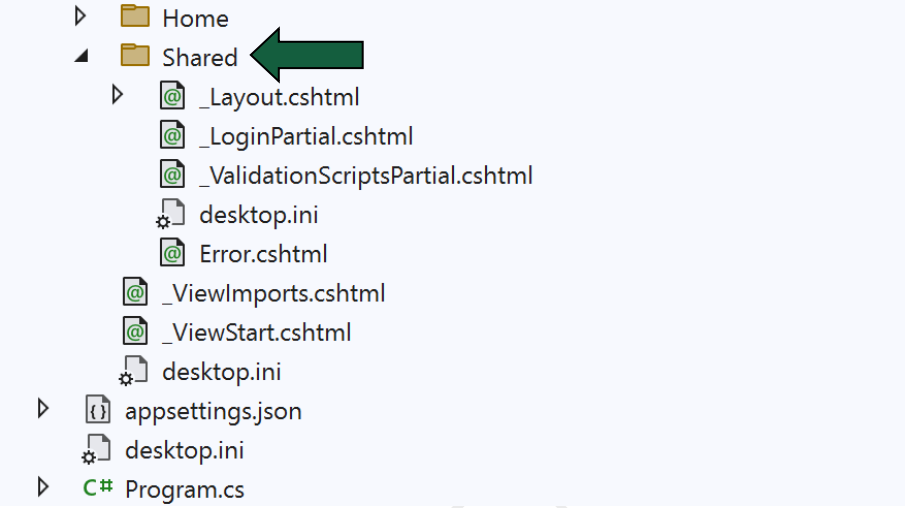
View-ul poate contine HTML static sau chiar HTML trimis din Controller (HTML dinamic). In cadrul arhitecturii MVC, View-ul comunica doar cu Controller-ul, iar cu Modelul comunica indirect tot prin intermediul Controller-ului.



View-urile in ASP.NET Core MVC se afla in folderul **Views**. Diferitele actiuni (metode) implementate intr-un Controller randeaza diferite View-uri, ceea ce inseamna ca folderul Views contine cate un folder separat pentru fiecare Controller, cu acelasi nume ca si Controller-ul, dupa cum se observa in imaginea urmatoare:



Folderul **Shared** contine View-urile (layout-uri, view-uri partiale) care sunt partajate (folosite) in cadrul mai multor View-uri existente in aplicatie.



Razor in cadrul unui proiect ASP.NET Core MVC

Inca din ASP.NET MVC 3 motorul (engine-ul) de afisare a View-urilor in acest framework este **Razor**. Acest motor ne ofera posibilitatea de a mixa tag-uri HTML cu cod C#. In Razor se utilizeaza caracterul **@** pentru a incepe o secventa de cod **server-side**.

Sintaxa Razor este simpla si a fost construita cu scopul de a minimiza lungimea codului scris. Aceasta este foarte compacta, usor de invatat si este suportata de editorul Visual Studio.

Exemple sintaxa Razor:

- Se foloseste **@** pentru executarea codului server-side pe o singura linie (de exemplu: pentru afisarea valorii unei variabile).

Exemplu – se afiseaza valoarea variabilei `ViewBag.Article` trimisa din Controller in View, preluandu-se atributul Title din modelul Article (conform exemplului implementat in cadrul laboratorului)

```
<h3>@ViewBag.Article.Title</h3>
```

- Parcurgerea colectiei **ViewBag.Articles** pentru afisarea articolelor

```
@foreach (var article in ViewBag.Articles)
```

- Pentru a executa un bloc intreg de cod (mai multe linii de cod) sau pentru a da o valoare unei variabile este necesar sa folosim acolade, astfel: `@{ /* cod */ }`

Exemplu – variabila ViewBag.Title primeste valoarea “Index”

```
@{
      ViewBag.Title = "Index";
}
```

Cu alte cuvinte, atunci cand se integreaza cod C# in HTML, se utilizeaza simbolul @ pentru o singura linie de cod si @{ pentru o secventa de cod.

- In cadrul unei secvente de cod se poate include cod HTML si text utilizand simbolul @:

Exemplul 1 – pentru includerea textului:

```
@if(1 > 0)
{
    @:este adevarat
}
```

In acest exemplu, putem vedea cum se poate afisa o secventa de text in cadrul unui bloc de cod. Secventa de mai sus va afisa pe ecran mesajul “este adevarat”.

Exemplul 2 – pentru includerea codului HTML (exemplu preluat din Laborator 6):

```
<select name="CategoryId">

   @foreach (var item in ViewBag.Categories)
   {
       <option value="@item.Id">@item.CategoryName</option>
   }

 </select>
```

- Conditia **if** incepe cu: **@if{ … }**

Exemplu:

```
@if(1 > 10)
{
    @:este fals
}
```

- Loop-ul are urmatoarea sintaxa: **@for{ … }**

Exemplu:

```
@for (int i = 0; i < 10; i++)
{
    @i.ToString()
}
```

- **@model** ofera posibilitatea afisarii valorilor modelului oriunde in View

Exemplu:

```
@model Curs7.Models.Student – includerea se realizeaza in View-ul asociat metodei
. . .
<h1>@Model.Name</h1>
<p>@Model.Email</p>
<p>@Model.CNP</p>
```

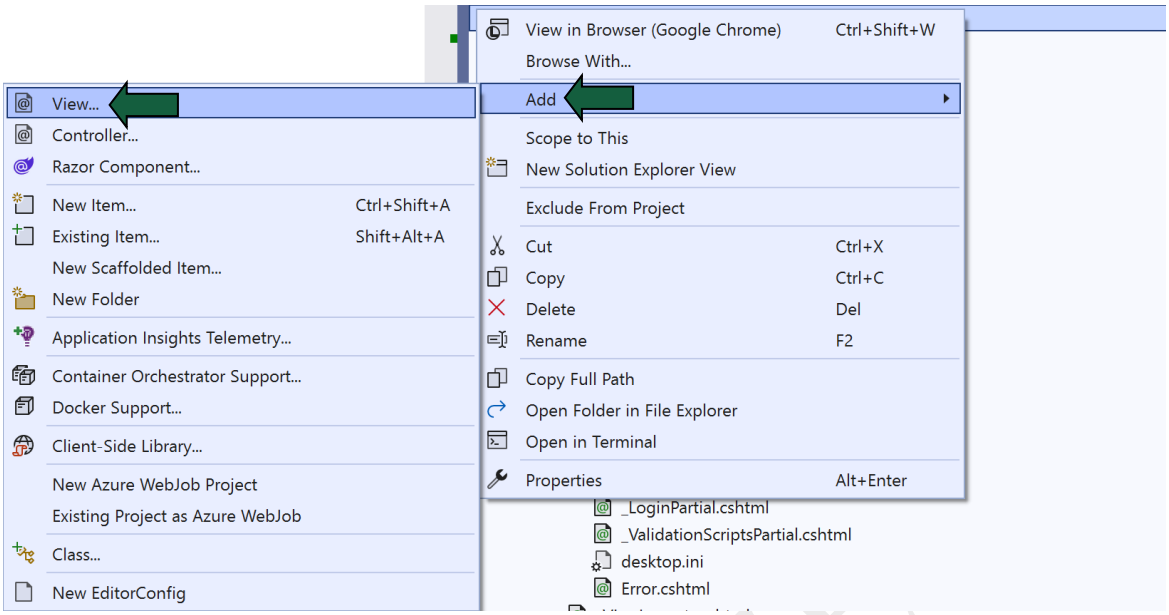
Crearea unui View in cadrul unui proiect

Se creeaza un View dupa cum urmeaza:

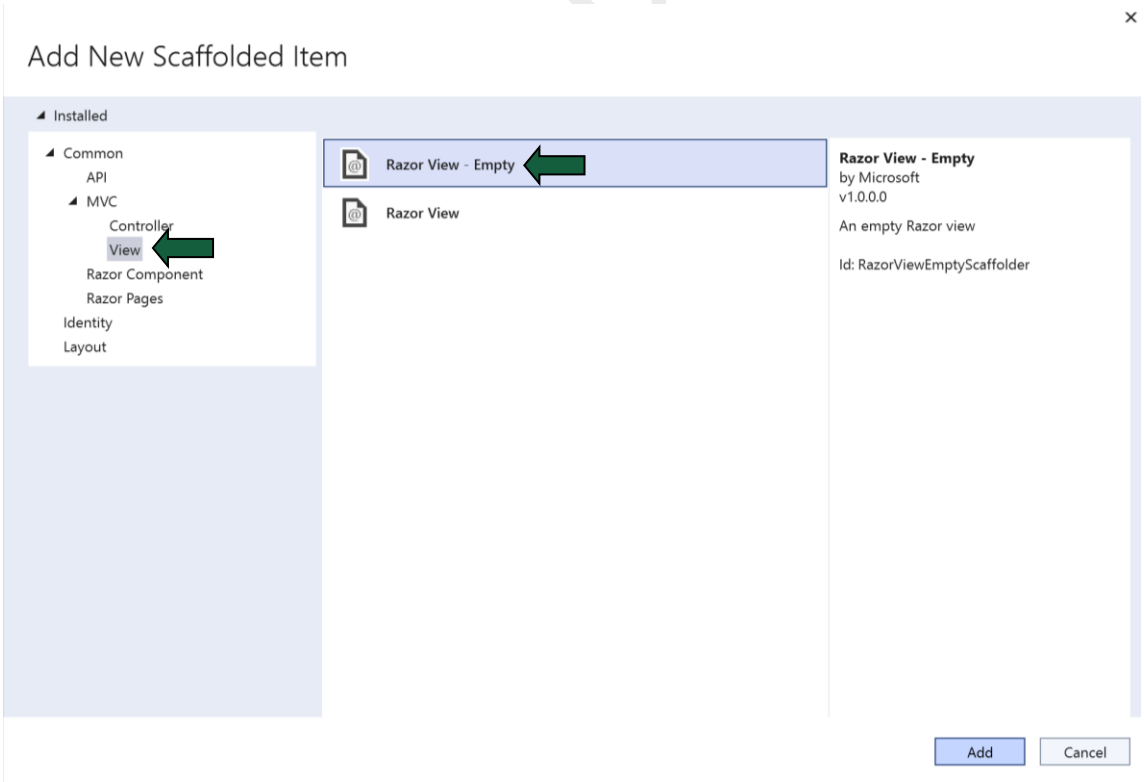
PASUL 1:

Click dreapta pe folderul corespunzator din folderul View → **Add** → **View**.

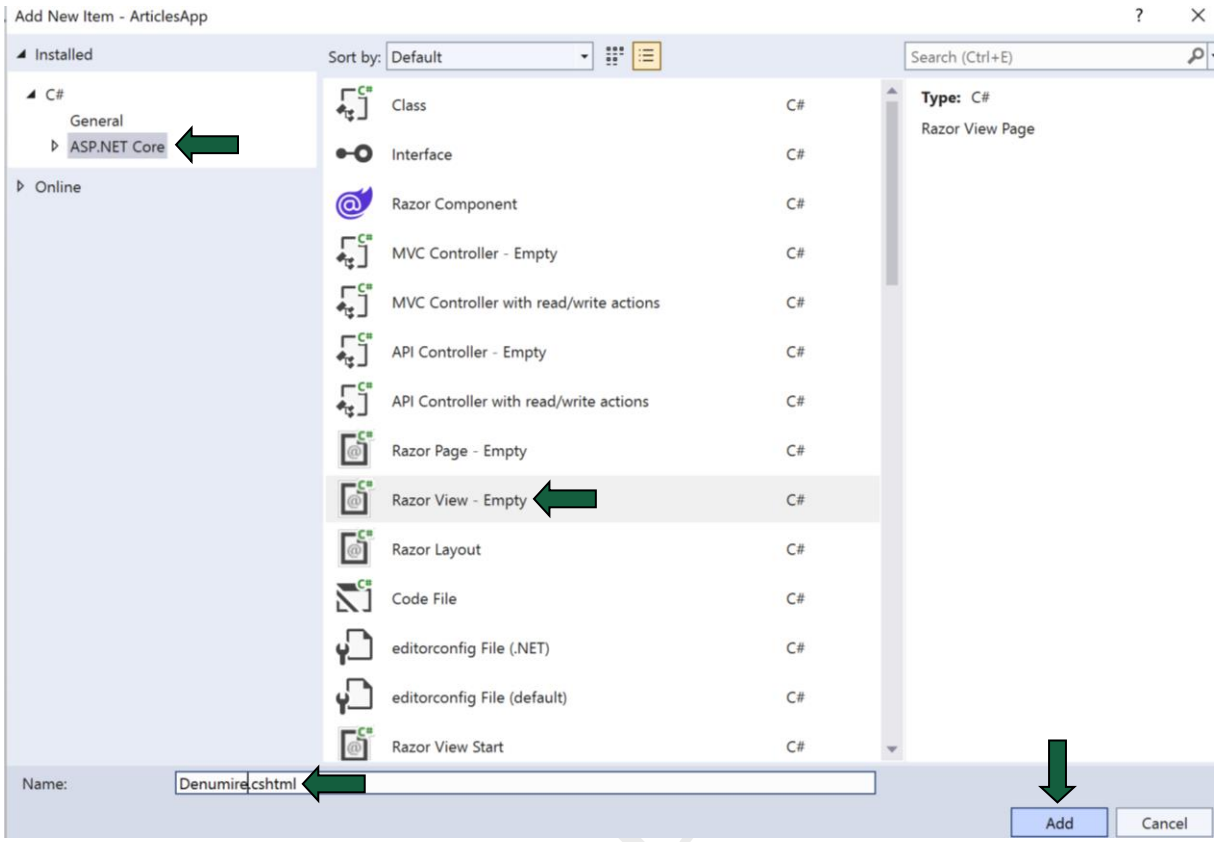
De exemplu: pentru prelucrarea articolelor din baza de date (**VEZI Laborator 6**) o sa avem Controller-ul ArticlesController, iar in folderul Views trebuie creat folderul Articles asociat. In acesta o sa existe View-urile asociate cu merocele din Controller.


View in Browser (Google Chrome) Ctrl+Shift+W
Browse With...
Add
Scope to This
New Solution Explorer View
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Delete Del
Rename F2
Copy Full Path
Open Folder in File Explorer
Open in Terminal
Properties Alt+Enter
View...
Controller...
Razor Component...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Application Insights Telemetry...
Container Orchestrator Support...
Docker Support...
Client-Side Library...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
New EditorConfig
_LoginPartial.cshtml
_ValidationScriptsPartial.cshtml
desktop.ini
Error.cshtml

PASUL 2:


Add New Scaffolded Item
Installed
Common
API
MVC
Controller
View
Razor Component
Razor Pages
Identity
Layout
Razor View - Empty
Razor View
Razor View - Empty
by Microsoft
v1.0.0.0
An empty Razor view
Id: RazorViewEmptyScaffolder
Add
Cancel

PASUL 3:



Trimiterea datelor catre View

De foarte multe ori este necesar sa trimitem diferite date catre View pentru afisarea acestora in browser. Pe langa informatiile primite din baza de date (prin intermediul modelului) exista cazuri in care vom trimite si alte tipuri de date.

Model

Trimiterea datelor din baza de date se face prin intermediul helper-ului **@model**. Astfel, pentru a afisa informatiile stocate in proprietatile unui model, se utilizeaza urmatoarea secventa de cod:

```
@model Curs7.Models.Student

@{
    ViewBag.Title = "Afisare student";
}

<h1>@Model.Name</h1>
<p>@Model.Email</p>
<p>@Model.CNP</p>
```

Pentru a putea trimite Modelul catre View si pentru a putea fi folosit este nevoie de urmatoarea secventa de cod in Controller:

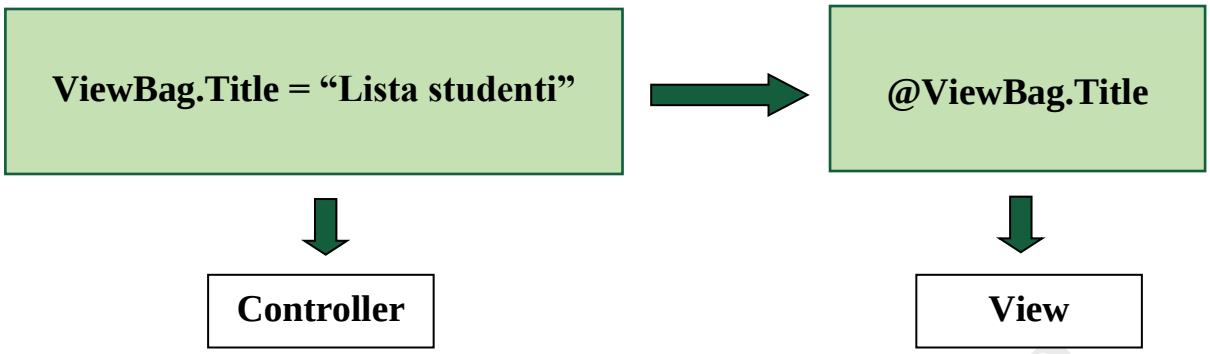
```
public IActionResult Show(int id)
   {
        Student student = db.Students.Find(id);

        return View( student );
   }
```

Aceasta secventa de cod paseaza obiectul de tip **Model** (in exemplul acesta, un obiect de tipul clasei **Student**) catre View.

ViewBag

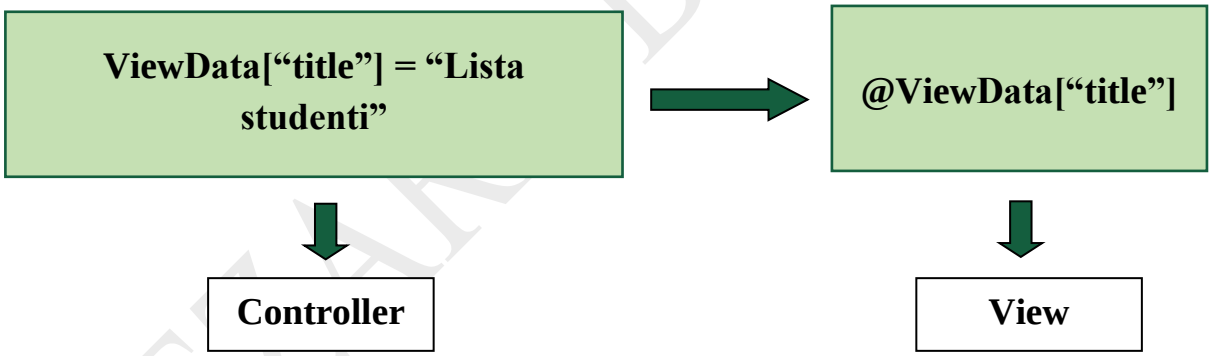
Helperul ViewBag – ofera posibilitatea transferului de date intre Controller si View. Aceste date sunt cele care nu se regasesc in Model.

Array-urile care contin obiecte de tipul unui Model, se pot trimite catre View tot prin ViewBag.



ViewData

Helperul ViewData – este similar cu ViewBag, singura diferenta dintre acestea fiind ca **ViewData** este reprezentat de un Dictionar si nu de un obiect, similar cu un array cheie-valoare. Fiecare cheie trebuie sa fie de tip string.



/!\ OBSERVATIE

ViewBag insereaza datele alocate proprietatilor in dictionarul asociat variabilei **ViewData**. Acest lucru necesita ca numele cheilor (pentru ViewData) respectiv numele proprietatilor (pentru ViewBag) sa fie **diferite**.

Exemplu:

ViewBag.Title = “Titlu”;
ViewData[“Title”] = “Titlu 2”;

Acest lucru conduce la suprascrierea valorii atributului “Title” din ViewBag (adica Titlu), cu ultima valoare alocata in cheia din ViewData (si anume “Titlu 2”)

Astfel, in View, atat **@ViewBag.Title**, cat si **@ViewData[“Title”]** vor afisa “Titlu 2”.

TempData

Helperul TempData – poate seta o valoare care va fi disponibila intr-un request subsecvent. Astfel, daca valoarea a fost setata in Actiunea1, iar aceasta actiune va face un redirect catre Actiunea2, valoarea setata in TempData va fi disponibila in Actiunea2 (**ATENTIE! Doar la prima accesare**).

Exemplu: Sa presupunem ca avem C.R.U.D. pentru obiectul Student. Controller-ul are metoda Delete care va sterge studentul din baza de date, prin verbul HTTP Post, neafisand in acest caz un View. Aceasta metoda executa codul aferent stergerii din baza de date si redirectioneaza catre metoda Index.

Pentru stergerea unui student, se vor executa 2 request-uri HTTP dupa cum urmeaza:

- **View-ul Show** (care afiseaza datele unui student si butonul de stergere) – aceasta este pagina unde are loc request-ul, prin apasarea butonului de stergere. La apasarea butonului se va face un Request catre metoda Delete din Controller

- **[Request 1]**: Se va accesa metoda Delete si se va executa codul aferent stergerii din baza de date. Dupa stergere, metoda redirectioneaza catre Index.

- **[Request 2]:** Browser-ul va ajunge in metoda Index si va prelua lista studentilor, impreuna cu mesajul primit din cadrul primului request, prin intermediul variabilei TempData. Aceste valori sunt apoi trimise in View-ul Index pentru afisarea catre utilizatorul final.

Pentru o mai buna experienta de utilizare a aplicatiei (**UX – User Experience**) este necesara afisarea catre utilizatorul final a unui mesaj. Mesajul are ca scop instiintarea utilizatorului cu privire la actiunea pe care tocmai a executat-o (resursa a fost stearsa cu succes). Acest lucru trebuie sa se intample in pagina Index si se poate realiza prin intermediul helper-ului **TempData** (deoarece avem doua request-uri subsecvente).

Implementare:

View-ul Show – din acest View o sa se execute request-ul (stergerea unui student din baza de date, in functie de id-ul acestuia)

```
@model Curs7.Models.Student


@{
    ViewBag.Title = "Afisare student";
}

<h1>@Model.Name</h1>
<p>@Model.Email</p>
<p>@Model.CNP</p>

<form method="post" action="/Students/Delete/@Model.Id">

    <button class="btn btn-danger"  type="submit">Stergere
student</button>

</form>
```

In StudentsController –> metoda Delete – in aceasta metoda se executa codul aferent **request-ului 1**, si anume se va sterge un student din baza de date. Dupa stergere, metoda redirectioneaza catre Index. In acest request variabila TempData[“message”] o sa stocheze stringul in cheia message, dupa care o sa trimita valoarea catre request-ul 2 → Index.

```
[HttpPost]
 public ActionResult Delete(int id)
 {
        Student student = db.Students.Find(id);
        TempData["message"] = "Studentul cu numele " +
student.Name + " a fost sters din baza de date";

        db.Students.Remove(student);
        db.SaveChanges();
        return RedirectToAction("Index");
  }
```

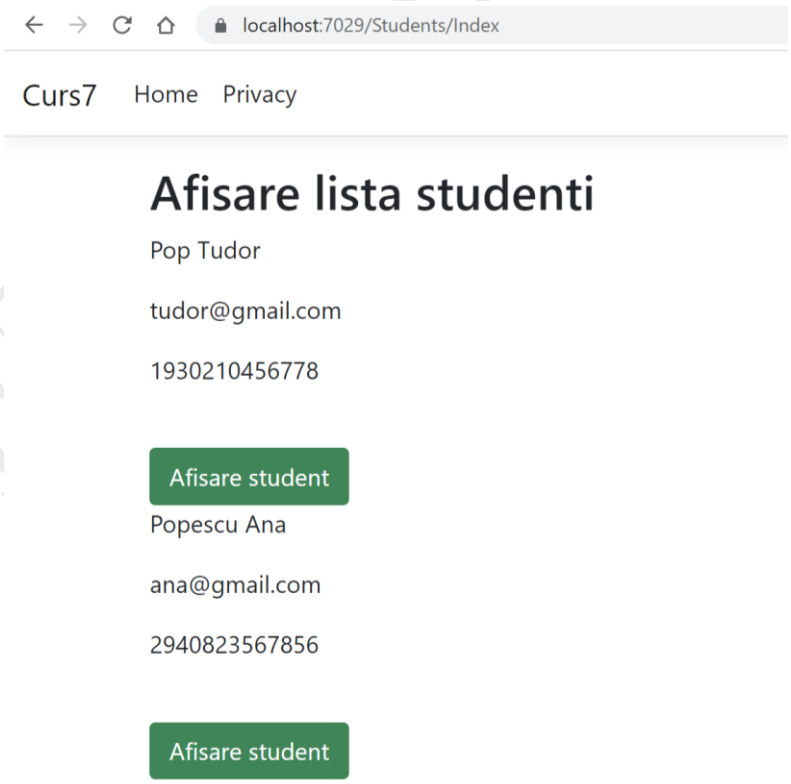
In StudentsController –> metoda Index – browser-ul va ajunge in metoda Index si va prelua lista studentilor, impreuna cu mesajul primit din cadrul primului request, prin intermediul variabilei `TempData`. Aceste valori sunt apoi trimise in View-ul Index pentru afisarea catre utilizatorul final.

```
public ActionResult Index()
{
        var students = from student in db.Students
                       select student;

        ViewBag.Students = students;

        if (TempData.ContainsKey("message"))
        {
            ViewBag.Msg = TempData["message"].ToString();
        }
        return View();
 }
```

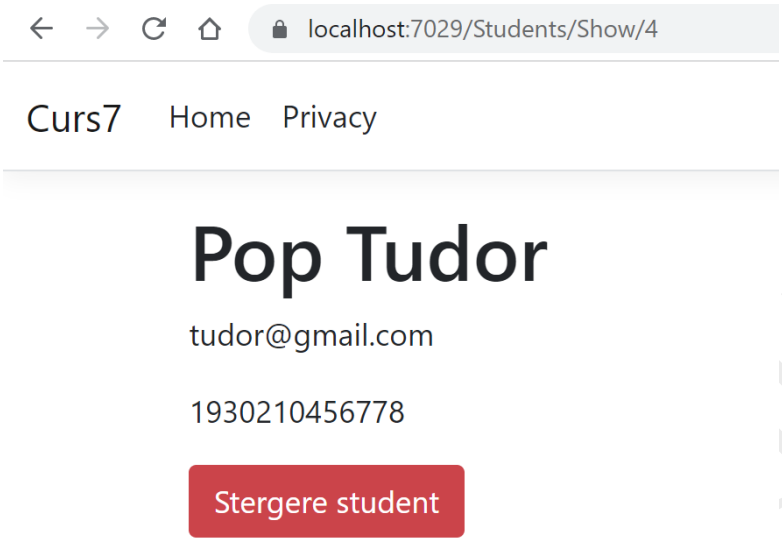
In View-ul Index – sunt afisati toti studentii din baza de date, dar si mesajul provenit din TempData.

```
<h2>Afisare lista studenti</h2>

<h2>@ViewBag.Msg</h2>

@foreach (var student in ViewBag.Students)
{
    <p>@student.Name</p>
    <p>@student.Email</p>
    <p>@student.CNP</p>
    <br />
    <a class="btn btn-success"
href="/Students/Show/@student.Id">Afisare student</a>
    <br />
}
```

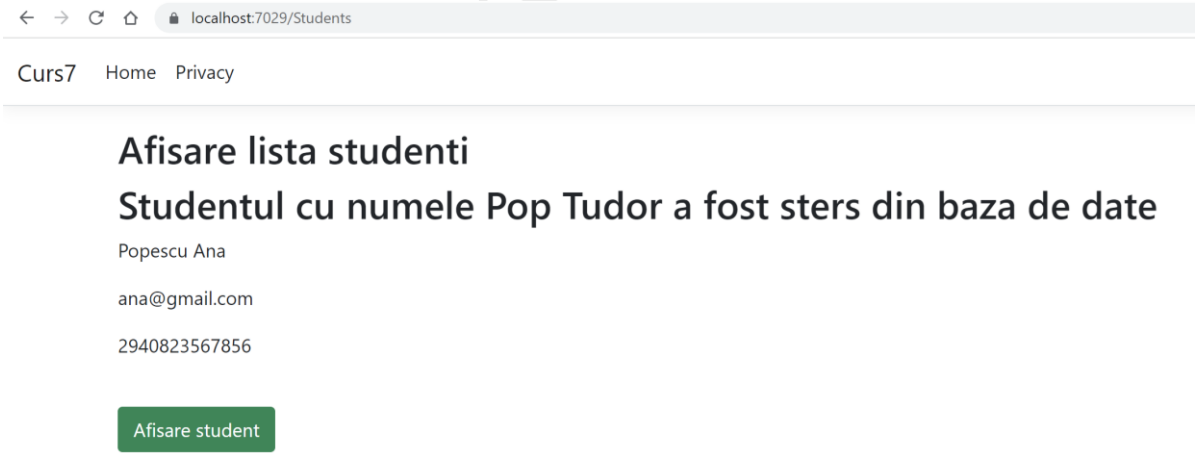
/Students/Index



/Students/Show/id



/Students/Index



Helpere pentru View

ASP.NET MVC Core, prin intermediul motorului Razor ofera o lista de **Helpere** care pot genera elemente de tip HTML. Aceste Helpere sunt folosite in combinatie cu un Model pentru a usura munca dezvoltatorului in generarea formularelor de procesare a datelor acestuia.

Aceste **Helpere utilizeaza sistemul de binding** pentru a afisa valorile modelului in elementele de tip HTML (**de exemplu**: in cazul editarii datelor unui model) cat si pentru trimiterea acestora catre Controller. Toate Helperele se acceseaza prin intermediul obiectului **@Html** disponibil in View.

Printre cele mai importante **Helpere** se enumera:

- **Html.ActionLink** – genereaza un URL
- **Html.TextBox** – genereaza un element de tipul TextBox
- **Html.TextArea** – genereaza un element de tipul TextArea
- **Html.CheckBox** – genereaza un element de tipul Check-box, util pentru valorile de tip Boolean
- **Html.RadioBox** – genereaza un element de tipul Radio button
- **Html.DropDownList** –genereaza un element de tipul Dropdown, util pentru valorile de tip Enum
- **Html.ListBox** – genereaza un element de tipul Dropdown cu selectie multipla
- **Html.Hidden** – genereaza un input field ascuns
- **Html.Password** – genereaza un camp pentru introducerea parolelor (textul introdus in camp este ascuns)
- **Html.Display** – este util pentru afisarea textelor
- **Html.Label** – genereaza un label pentru un element mentionat anterior
- **Html.Editor** – acest helper genereaza unul din elementele de mai sus in functie de tipul proprietatii modelului. Astfel, daca editorul este alocat unui camp de tip **int** va genera un input de tip numeric; daca editorul este alocat unui camp de tip string va genera un textbox, etc.

Folosirea acestor Helpere este similara cu scrierea manuala a codului HTML aferent formularelor, insa, Helperele **ofera si posibilitatea de binding a datelor in mod automat**.

Exemplu: sa consideram formularul urmator pentru **adaugarea unui student in baza de date**.

```
<form method="post" action="/Students/New">
    <label>Nume</label>
    <br />
    <input type="text" name="Name" />
    <br /><br />
    <label>Adresa e-mail</label>
    <br />
    <input type="text" name="Email" />
    <br /><br />
    <label>CNP</label>
    <br />
    <input type="text" name="CNP" />
    <br />
    <button type="submit">Adauga student</button>
</form>
```

Acesta se va rescrie, folosind **Helperele Html** dupa cum urmeaza:

```
<form method="post" action="/Students/New">

    @Html.Label("Name", "Nume Student")
    <br />

    @Html.TextBox("Name", null, new { @class = "form-control" })
    <br /><br />

    @Html.Label("Email", "Adresa de e-mail")
    <br />

    @Html.TextBox("Email", null, new { @class = "form-control"
})
    <br /><br />

    @Html.Label("CNP", "CNP Student")
    <br />

    @Html.TextBox("CNP", null, new { @class = "form-control" })
    <br />

    <button type="submit">Adauga student</button>

</form>
```

Generarea unui label pentru atributul "Name"

Generarea unui input field pentru proprietatea Email

Folosind aceste Helpere, codul HTML generat de View este urmatorul:

```
<form method="post" action="/Students/New">
    <label for="Name">Nume Student</label>
    <br />
    <input class="form-control" id="Name" name="Name" type="text"
value="" />
    <br /><br />
    <label for="Email">Adresa de e-mail</label>
    <br />
    <input class="form-control" id="Email" name="Email"
type="text" value=""/>
    <br /><br />
    <label for="CNP">CNP Student</label>
    <br />
    <input class="form-control" id="CNP" name="CNP" type="text"
value="" />
    <br />
    <button type="submit">Adauga student</button>
</form>
```

Pentru **label** se genereaza urmatoarele elemente:

```
@Html.Label("Name", "Nume Student")

<label for="Name">Nume Student</label>
```

Pentru **TextBox** se genereaza urmatoarele elemente:

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

<input id="Name" name="Name" type="text" value="" class="form-
control" />
```

Continutul HTML generat de Helpere este similar cu cel scris manual. Insa, in cazul **editarii**, **unde este necesar sa preluam valorile existente in Model** pentru modificarea ulterioara a acestora, putem apela la helperul **Html.Editor** pentru a genera in mod automat campurile de editare si pentru a prelua automat aceste valori.

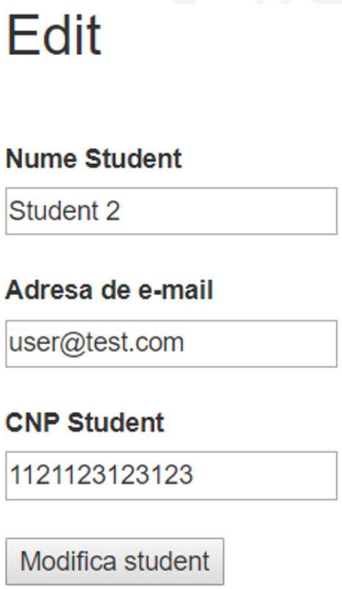
Formularul initial pentru **editarea unui student**, implementat manual cu preluarea manuala a valorilor si alocarea acestora inputurilor HTML prin intermediul atributului **value** are urmatorul continut:

```
<form method="post" action="/Students/Edit/@ViewBag.Student.Id">

    <label>Nume</label>

    <input type="text" name="Name" value="@ViewBag.Student.Name" />

    <label>Adresa e-mail</label>

    <input type="text" name="Email" value="@ViewBag.Student.Email" />

    <label>CPN</label>
    <input type="text" name="CNP" value="@ViewBag.Student.CNP" />

    <button type="submit">Modifica student</button>

</form>
```

Acesta poate fi rescris prin intermediul helperelor astfel:

```
@model Curs7.Models.Student

<form method="post" action="/Students/Edit/@Model.Id">


    @Html.Label("Name", "Nume Student")
    <br />
    @Html.Editor("Name")
    <br /><br />

    @Html.Label("Email", "Adresa de e-mail")
    <br />
    @Html.Editor("Email")
    <br /><br />

    @Html.Label("CNP", "CNP Student")
    <br />
    @Html.Editor("CNP")
    <br /><br />

    <button type="submit">Modifica student</button>

</form>
```

Acest helper genereaza in mod automat campul necesar



Codul generat este urmatorul:

```
<form method="post" action="/Students/Edit/1">

    <label for="Name">Nume Student</label>

    <br />

    <input class="text-box single-line" id="Name" name="Name"
type="text" value="Student 2" />

    <br /><br />

    <label for="Email">Adresa de e-mail</label>

    <br />

    <input class="text-box single-line" data-val="true" data-
val-required="The Email field is required." id="Email"
name="Email" type="text" value="user@test.com" />

    <br /><br />

    <label for="CNP">CNP Student</label>

    <br />

    <input class="text-box single-line" data-val="true" data-
val-maxlength="The field CNP must be a string or array type with
a maximum length of &#39;13&#39;." data-val-maxlength-max="13"
data-val-minlength="The field CNP must be a string or array type
with a minimum length of &#39;13&#39;." data-val-minlength-
min="13" id="CNP" name="CNP" type="text" value="1121123123123"
/>

    <br /><br />

    <button type="submit">Modifica student</button>

</form>
```

Putem observa ca **Helper-ul Html.Editor** a generat toate campurile necesare pentru formularul de editare a studentului, conform definitiei modelului Student. De asemenea, in formular se observa ca toate atributele **value** asociate elementelor formularului au primit valorile modelului in mod automat (**prin Binding**).

Implementarea alternativa a Helperelor

Html.TextBox – genereaza un element de tipul TextBox → Devine input in noua versiune de utilizare a helperelor. In continuare sunt prezentate ambele variante:

- **Varianta 1 → Helper @Html**

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

- **Varianta 2 → Helper in-line**

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

Html.DropDownList – genereaza un element de tipul Dropdown →Devine select

- **Varianta 1 → Helper @Html**

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

- **Varianta 2 → Helper in-line**

```
<select class="form-control" asp-for="CategoryId" asp-
items="Model.Categories">
        <option disabled selected value="">Selectati
Categoria</option>
</select>
```

Html.Label →Devine Label

- **Varianta 1 → Helper @Html**

```
@Html.LabelFor(m => m.Title, "Titlu Articol")
```

- **Varianta 2 → Helper in-line**

```
<label asp-for="Title"></label>
```

Fiecare Helper are un atribut numit **asp-for** folosit pentru a indica proprietatea pe care modelul o utilizeaza pentru procesul de model binding.

Helperele pot avea si alte atribute care incep cu **asp-** si care sunt folosite pentru diverse procese, cum ar fi:

- populatarea unui select cu o lista de optiuni (**asp-items**)
- setarea unei rute pentru un form **asp-controller** si **asp-action**
- setarea unui url pentru un un tag <a>: **asp-controller** si **asp-action**