Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Curs 7

Cuprins

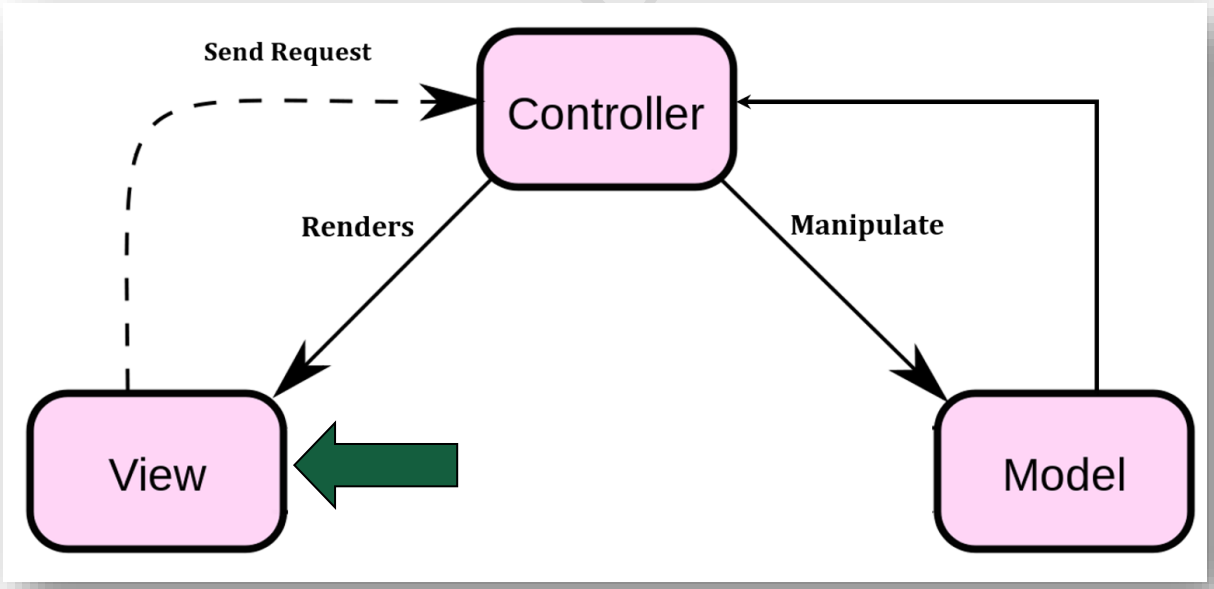
View (interfața cu utilizatorul)

Ce este View-ul

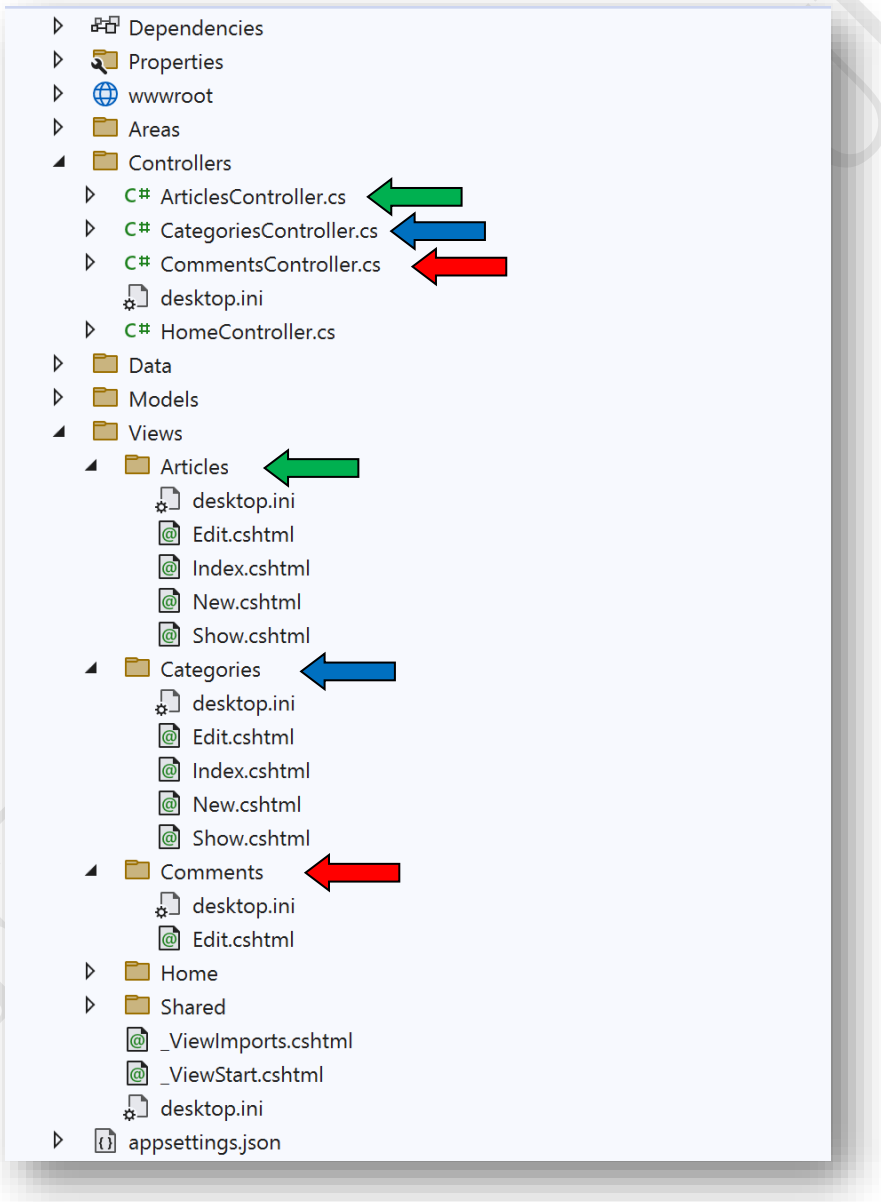
View-ul reprezintă interfața cu utilizatorul, fiind componenta arhitecturii MVC cu care utilizatorii interacționează, prin intermediul browser-ului.

În View se afișează datele, adică înregistrările din baza de date și informațiile generate de aplicație.

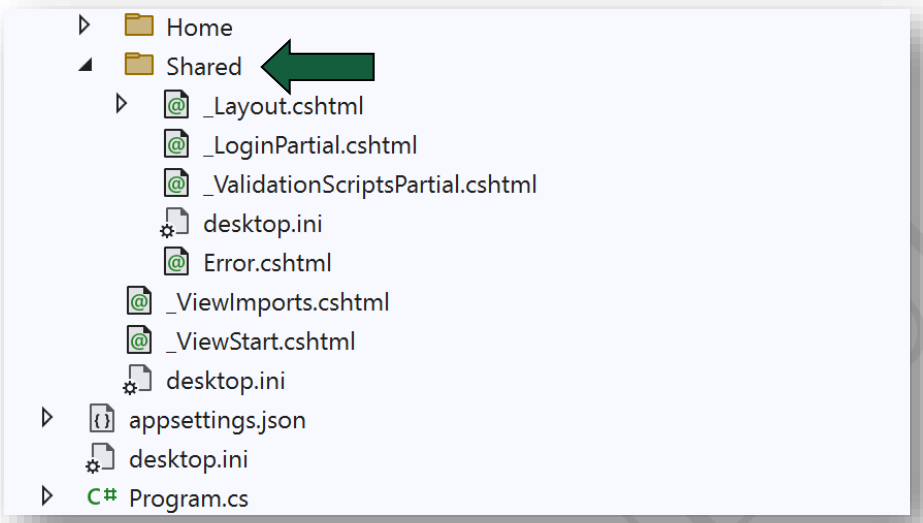
View-ul poate conține HTML static sau chiar HTML trimis din Controller (HTML dinamic). În cadrul arhitecturii MVC, View-ul comunică doar cu Controller-ul, iar cu Modelul comunică indirect, tot prin intermediul Controller-ului.



View-urile în ASP.NET Core MVC se află în folderul **Views**. Diferitele acțiuni (metode) implementate într-un Controller randează diferite View-uri, ceea ce înseamnă că folderul Views conține câte un folder separat pentru fiecare Controller, cu același nume ca și Controller-ul, după cum se observă în imaginea următoare:



Folderul **Shared** conține View-urile (layout-uri, view-uri parțiale) care sunt partajate (folosite) în cadrul mai multor View-uri existente în aplicație.



Razor în cadrul unui proiect ASP.NET Core MVC

Încă din ASP.NET MVC 3 motorul (engine-ul) de afișare a View-urilor în acest framework este **Razor**. Acest motor ne oferă posibilitatea de a mixa tag-uri HTML cu cod C#. În Razor se utilizează caracterul @ pentru a începe o secvență de cod **server-side**.

Sintaxa Razor este simplă și a fost construită cu scopul de a minimiza lungimea codului scris. Aceasta este foarte compactă, ușor de învățat și este suportată de editorul Visual Studio.

Exemple sintaxă Razor:

- Se folosește @ pentru executarea codului server-side, pe o singură linie (de exemplu: pentru afișarea valorii unei variabile).

Exemplu – se afișează valoarea variabilei `ViewBag.Article` (Article este numele ales pentru această variabilă), trimisă din Controller în View, preluându-se atributul ***Title*** din modelul ***Article*** (conform exemplului implementat în cadrul laboratorului)

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

- Parcurgerea colecției **ViewBag.Articles** pentru afișarea articolelor

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

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

Exemplu – variabila ViewBag.Title primește valoarea "Index"

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

Cu alte cuvinte, atunci când se integrează cod C# în HTML, se utilizează simbolul @ pentru o singură linie de cod și @{} pentru o secvență de cod.

- In cadrul unei secvente de cod se poate include cod HTML și text, utilizând simbolul @:

Exemplul 1 – pentru includerea textului:

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

În acest exemplu, se observă cum se poate afișa o secvență de text în cadrul unui bloc de cod. Secvența de mai sus va afișa pe ecran mesajul “este adevărat”.

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** începe cu: `@if{ … }`

Exemplu:

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

- Loop-ul are următoarea sintaxă: **@for{ … }**

Exemplu:

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

- **@model** oferă posibilitatea afișării valorilor modelului, oriunde în View

Exemplu:

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

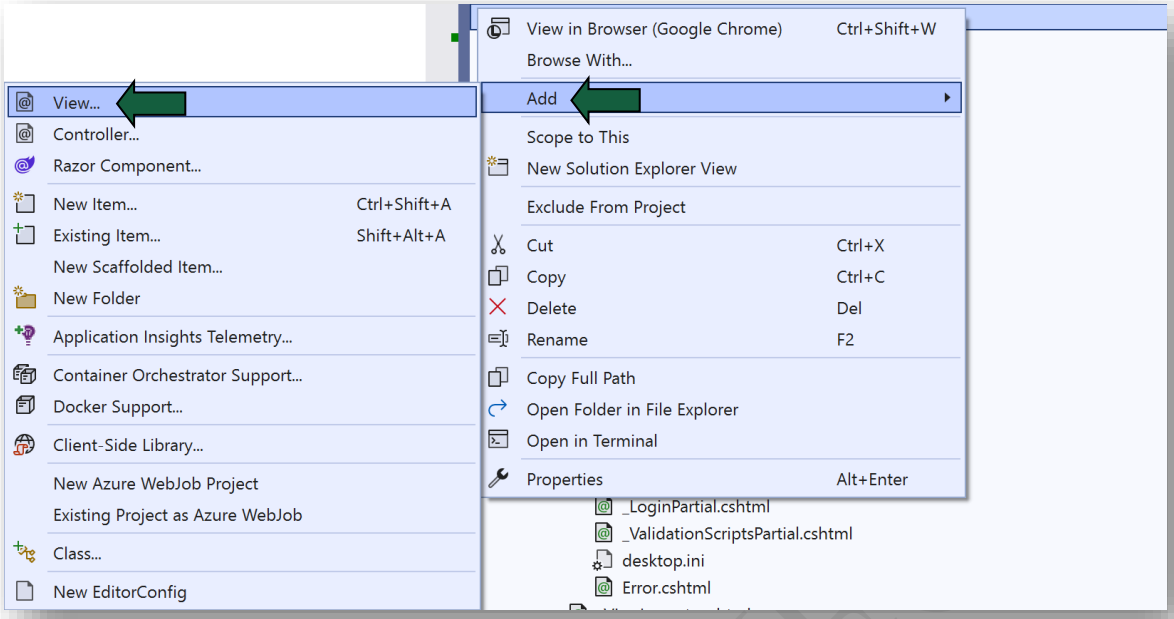
Crearea unui View în cadrul unui proiect

Se creează un View, după cum urmează:

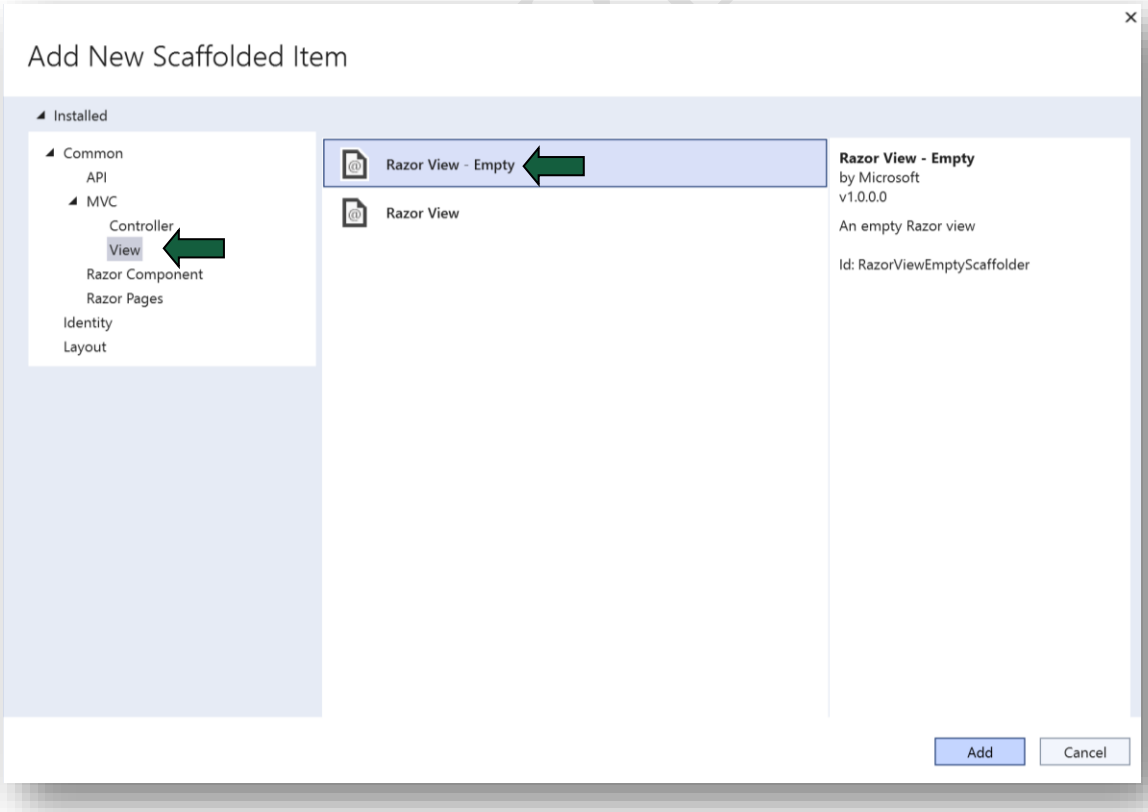
PASUL 1:

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

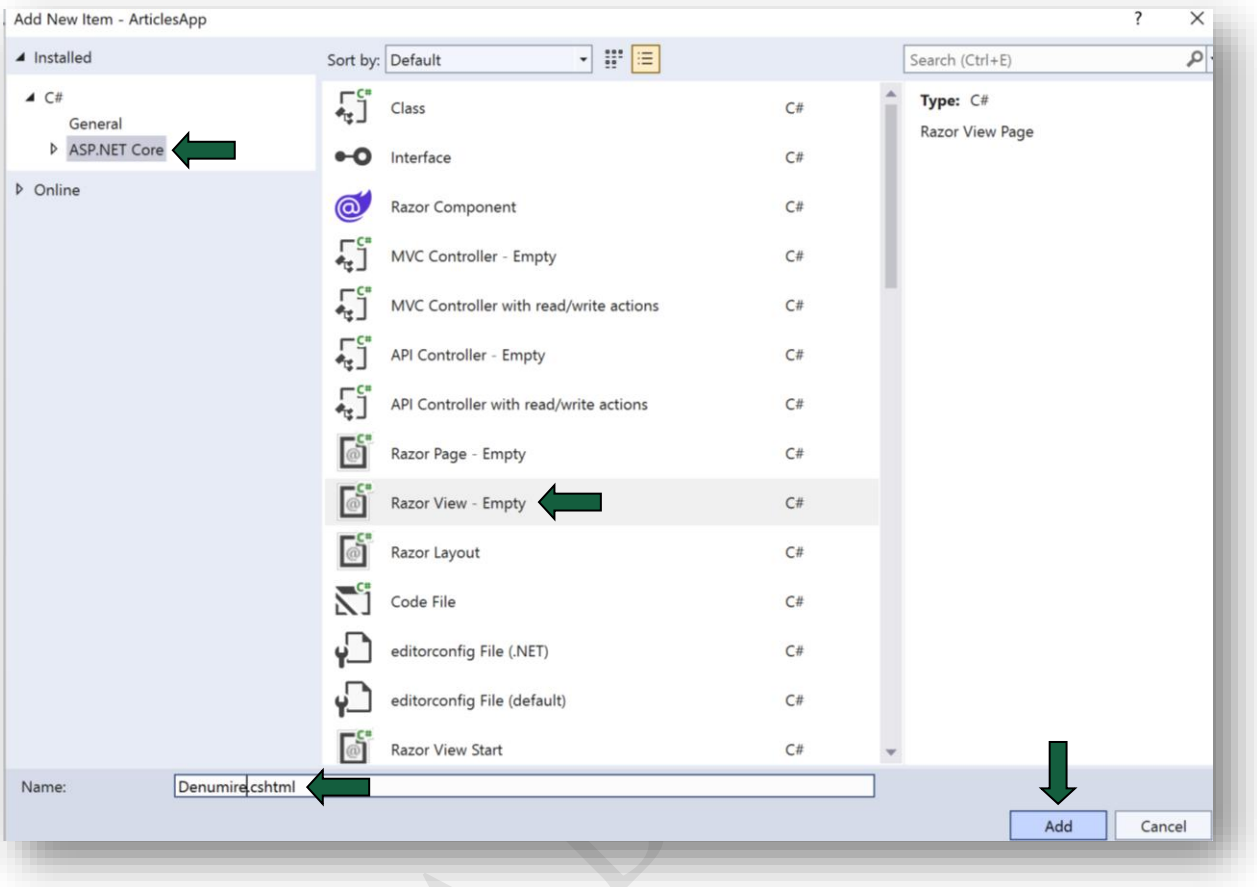
De exemplu: pentru prelucrarea articolelor din baza de date (**VEZI Laborator 6**) o să avem Controller-ul ***ArticlesController***, iar în folderul Views trebuie creat folderul ***Articles*** asociat. În acesta, o să existe View-urile asociate cu metodele din Controller.


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

De foarte multe ori, este necesar să trimitem diferite date către View pentru afișarea acestora în browser. Pe langa informațiile primite din baza de date (prin intermediul modelului), există cazuri în care vom trimite și alte tipuri de date.

Model

Trimiterea datelor din baza de date se face prin intermediul helper-ului **@model**. Astfel, pentru a afișa informațiile stocate în proprietățile unui model, se utilizează următoarea secvență de cod:

```
@model Curs7.Models.Student

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

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

Includerea modelului necesar pentru afișare

Se utilizează @Model pentru preluarea și afișarea datelor

Pentru a putea trimite **Modelul** către **View** și pentru a putea fi folosit, este necesară următoarea secvență de cod în Controller:

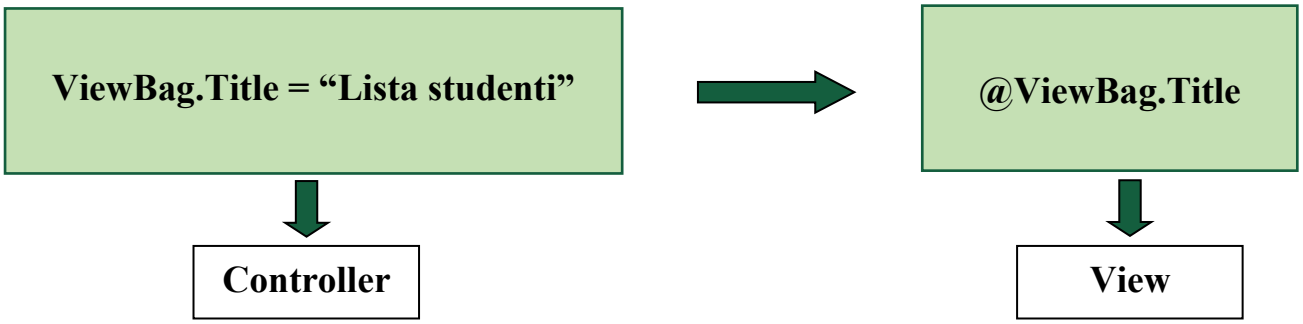
```
public IActionResult Show(int id)
   {
       Student student = db.Students.Find(id);

       return View( student );
   }
```

Această secvență de cod transmite obiectul de tip **Model** (în exemplul acesta, un obiect de tipul clasei **Student**) către View.

ViewBag

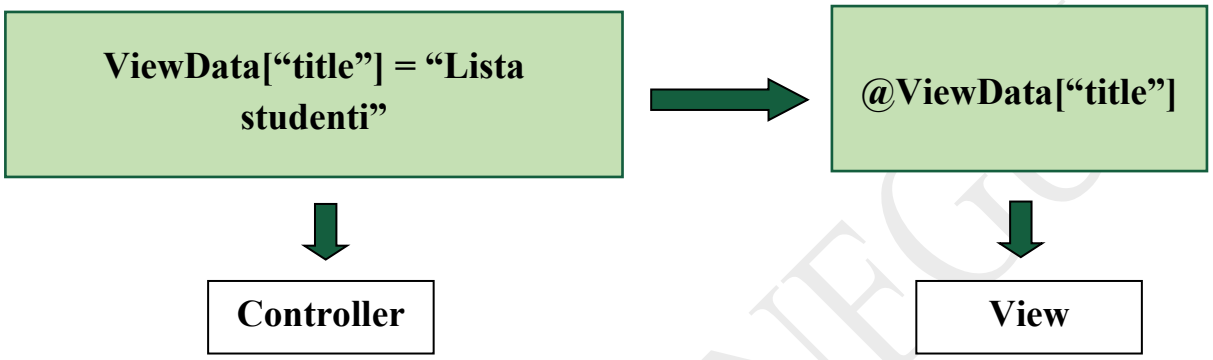
Helperul ViewBag – oferă posibilitatea transferului de date între Controller și View. Aceste date sunt cele care nu se regăsesc în Model.

Array-urile care conțin obiecte de tipul unui Model se pot trimite către View tot prin intermediul unei variabile de tipul ViewBag.



ViewData

Helperul ViewData – este similar cu ViewBag, singura diferență dintre acestea fiind că **ViewData** este reprezentat de un **Dicționar** și nu de un obiect, similar cu un array cheie-valoare. Fiecare cheie trebuie să fie de tip string.



/!\ OBSERVAȚIE

ViewBag inserează datele alocate proprietăților sale în dicționarul asociat variabilei **ViewData**. Acest lucru necesită ca numele cheilor (pentru ViewData) și numele proprietăților (pentru ViewBag) să fie **diferite**.

Exemplu:

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

Acest lucru conduce la suprascrierea valorii atributului “Title” din ViewBag (adică Titlu), cu ultima valoare alocată în cheia din ViewData (și anume “Titlu 2”)

Astfel, în View, atât **@ViewBag.Title**, cât și **@ViewData[“Title”]** vor afișa “Titlu 2”.

TempData

Helperul TempData – poate seta o valoare care va fi disponibilă într-un request subsecvent. Astfel, dacă valoarea a fost setată în Acțiunea1, iar această acțiune va face un redirect către Acțiunea2, valoarea setată în TempData va fi disponibilă în Acțiunea2 (**ATENȚIE! Doar la prima accesare**).

Exemplu: Să presupunem că avem C.R.U.D. pentru obiectul Student. Controller-ul are metoda Delete, care va șterge studentul din baza de date, prin verbul HTTP Post, neafișând în acest caz un View. Această metodă execută codul aferent ștergerii din baza de date și redirectionează către metoda Index.

Pentru ștergerea unui student, se vor executa două request-uri HTTP, după cum urmează:

- **View-ul Show** (care afișează datele unui student și butonul de ștergere) – aceasta este pagina unde are loc request-ul, prin apăsarea butonului de ștergere. La apăsarea butonului se va face un ***Request*** către metoda Delete din Controller.

- **[Request 1]**: Se va accesa metoda Delete și se va executa codul aferent ștergerii din baza de date. După ștergere, metoda redirecționează către Index.

- **[Request 2]:** Browser-ul va ajunge în metoda Index și va prelua lista studenților, împreună cu mesajul primit din cadrul primului request, prin intermediul variabilei TempData. Aceste valori sunt apoi trimise în View-ul Index pentru afișarea către utilizatorul final.

Pentru o mai bună experiență de utilizare a aplicației (**UX – User Experience**) este necesară afișarea, către utilizatorul final, a unui mesaj. Mesajul are ca scop înștiințarea utilizatorului cu privire la acțiunea pe care tocmai a executat-o (resursa a fost ștearsă cu succes). Acest lucru trebuie să se întâmple în pagina Index și se poate realiza prin intermediul helper-ului **TempData** (deoarece avem două request-uri subsecvente).

Implementare:

View-ul Show – din acest View o să se execute request-ul (ștergerea unui student din baza de date, în funcție 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>
```

În **StudentsController –> metoda Delete** – în această metodă se execută codul aferent **Request-ului 1**, și anume se va șterge un student din baza de date. După ștergere, metoda redirecționează către Index. În acest request, variabila TempData[“message”] va stoca stringul în cheia `"message"`, după care va trimite valoarea către request-ul 2 → Index.

```
[HttpPost]
 public ActionResult Delete(int id)
 {
       Student student = db.Students.Find(id);
       db.Students.Remove(student);
       db.SaveChanges();

        TempData["message"] = "Studentul cu numele " +
     student.Name + " a fost sters din baza de date";

       return RedirectToAction("Index");
  }
```

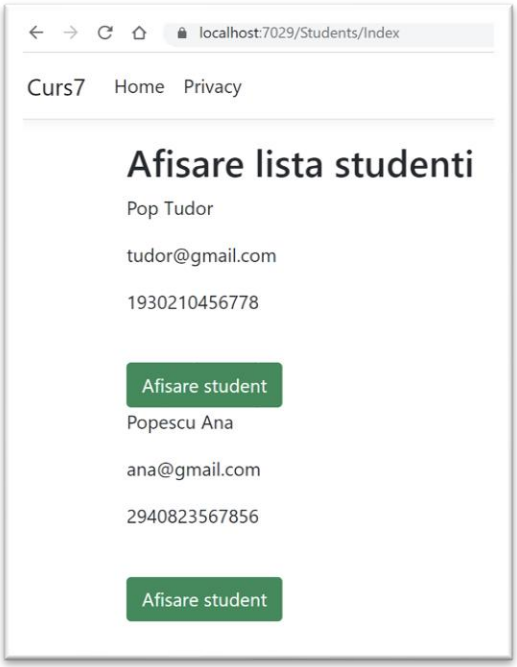
În **StudentsController –> metoda Index –** browser-ul va ajunge în metoda Index și va prelua lista studenților, împreună cu mesajul primit din cadrul primului request, prin intermediul variabilei `TempData`. Aceste valori sunt apoi trimise în View-ul Index pentru a fi afișate utilizatorului final.

```
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();
 }
```

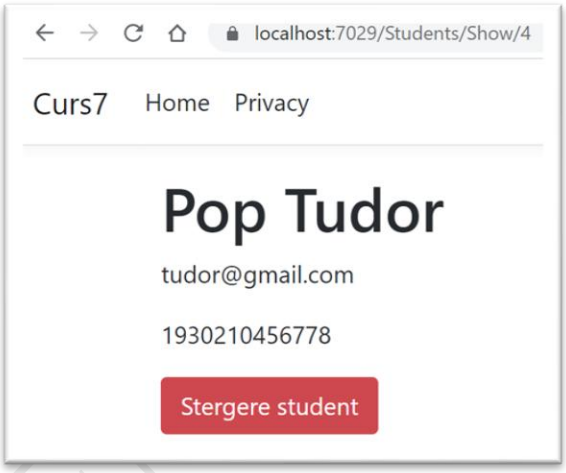
În **View-ul Index** – sunt afisați toți studenții din baza de date, dar și mesajul provenit din TempData.

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

@if(ViewBag.Msg != null){
     <h2 class="alert-success p-3 rounded-3 text-center
     mb-5">@ViewBag.Message</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 />
}
```

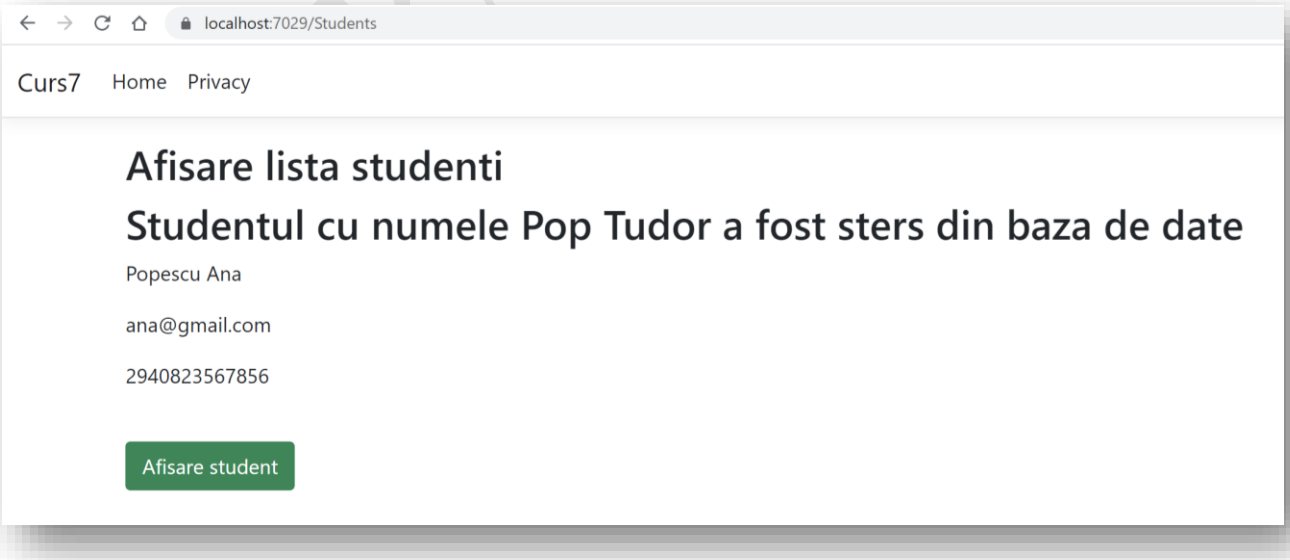
/Students/Index



/Students/Show/id



/Students/Index



Helpere pentru View

ASP.NET MVC Core, prin intermediul motorului Razor, oferă o listă de **Helpere** care pot genera elemente de tip HTML. Aceste Helpere sunt folosite în combinație cu un Model pentru a ușura munca dezvoltatorului în generarea formularelor de procesare a datelor acestuia.

Aceste **Helpere utilizează sistemul de model binding** pentru a afișa valorile modelului în elementele de tip HTML.

Helperele se împart în două categorii: **HTML Helpers** și **Tag Helpers**. Ambele tipuri se comportă la fel, cu mici diferențe de performanță.

Tag Helpers sunt implementate pe baza unei arhitecturi mai moderne și sunt procesate la nivelul **compilării** Razor. Așadar, sunt procesate mai devreme în timpul compilării paginii Razor, ceea ce poate oferi un avantaj în ceea ce privește optimizarea codului. De asemenea, au suport mai bun pentru completarea automată și refactoring în IDE-uri.

HTML Helpers sunt procesate la **runtime** și generează HTML direct în codul View-ului. Acest lucru poate genera o ușoară scădere de performanță la execuție comparativ cu **Tag Helpers**. Diferențele sunt minime și nu vor fi semnificative decât în cazul unor aplicații de mari dimensiuni.

În continuare vor fi prezentate **HTML Helpers** și echivalentul lor în **Tag Helpers**:

1. **Html.ActionLink** – generează un URL
    - **Tag Helper**: <a asp-action="ActionName" asp-controller="ControllerName">Link text
2. **Html.TextBox** – generează un element de tipul TextBox
    - **Tag Helper**: <input asp-for="Property" class="form-control" />

3. **Html.TextArea** – generează un element de tipul TextArea

    - **Tag Helper**: <textarea asp-for="Property" class="form-control"></textarea>

4. **Html.CheckBox** – generează un element de tipul Check-box, util pentru valorile de tip Boolean

    - **Tag Helper**: <input asp-for="Property" type="checkbox" class="form-check-input" />

5. **Html.RadioBox** – generează un element de tipul Radio button

    - **Tag Helper**: <input asp-for="Property" type="radio" class="form-check-input" />

6. **Html.DropDownList** –generează un element de tipul Dropdown, util pentru valorile de tip Enum

    - **Tag Helper**:

      ```
      <select asp-for="Property" asp-items="Model.PropertyList" class="form-control">
        <option value="">Select</option>
      </select>
      ```

7. **Html.ListBox** – generează un element de tipul Dropdown cu selecție multiplă

    - **Tag Helper**:

      ```
      <select asp-for="Property" asp-items="Model.PropertyList" multiple="multiple" class="form-control"> <option value="">Select</option> </select>
      ```

8. **Html.Hidden** – generează un input field ascuns

    - **Tag Helper**: <input asp-for="Property" type="hidden" />

9. **Html.Password** – generează un câmp pentru introducerea parolelor (textul introdus în câmp este ascuns)

    - **Tag Helper**: <input asp-for="Property" type="password" class="form-control" />

10. **Html.Display** – este util pentru afișarea textelor

    - **Tag Helper**:

11. **Html.Label** – generează un label pentru un element menționat

    - **Tag Helper**: <label asp-for="Property" class="form-label"></label>

12. **Html.Editor** – acest helper generează unul din elementele de mai sus, în funcție de tipul proprietății modelului. Astfel, dacă editorul este alocat unui câmp de tip **int**, va genera un input de tip numeric; dacă editorul este alocat unui camp de tip string, va genera un textbox, etc.

    - **Tag Helper**: <input asp-for="Property" class="form-control" />

Attribute	Description
asp-controller	The name of the controller.
asp-action	The name of the action method.
asp-area	The name of the area.
asp-page	The name of the Razor page.
asp-page-handler	The name of the Razor page handler.
asp-route	The name of the route.
asp-route-{value}	A single URL route value. For example, `asp-route-id="1234"`.
asp-all-route-data	All route values.
asp-fragment	The URL fragment.

Folosirea acestor helpere este similară cu scrierea manuală a codului HTML aferent formularelor, însă helperele oferă și posibilitatea de binding a datelor în mod automat.

Atât în cazul **HTML Helpers**, cât și a **Tag Helpers**, sunt direct integrate cu sistemul de **model binding** al framework-ului. Atunci când sunt utilizate Tag Helpers în formulare, acestea se leagă automat de proprietățile modelului, la fel ca și HTML Helpers.

Exemplu: să considerăm formularul următor pentru **adăugarea unui student în 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 **HTML Helpers**, după cum urmează:

```
<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 />
```

Generarea unui label pentru atributul “Name”

```
    @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 input field pentru proprietatea Email

Folosind aceste Helpere, codul HTML generat de View este următorul (acesta se poate vedea în consola – inspect element (F12)):

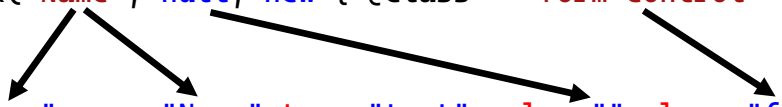
```
<form method="post" action="/Students/New">
   <label for="Name">Nume Student</label>
   <br />
   <input class="form-control" id="Name" name="Name" type="text"
value="" />
   <br /><br />
   <label for="Email">Adresa de e-mail</label>
   <br />
   <input class="form-control" id="Email" name="Email"
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 generează următoarele elemente:

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

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

Pentru **TextBox** se generează următoarele elemente:

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



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

În cazul utilizării **Tag Helpers**, codul este următorul:

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

    <div class="form-group">
        <label asp-for="Name" class="form-label">Nume Student</label>
        <input asp-for="Name" class="form-control" />
    </div>

    <div class="form-group">
        <label asp-for="Email" class="form-label">Adresa de e-
mail</label>
        <input asp-for="Email" class="form-control" />
    </div>

    <div class="form-group">
        <label asp-for="CNP" class="form-label">CNP Student</label>
        <input asp-for="CNP" class="form-control" />
    </div>

    <button type="submit" class="btn btn-primary">Adauga
student</button>

</form>
```

Explicații:

- **asp-action="New"** – reprezintă **Acțiunea** care va procesa formularul
- **asp-controller="Students"** – reprezintă **Controller-ul** în care se va trimite formularul
- **asp-for="Property"** – leagă fiecare element din formular de o proprietate a modelului (model binding).
  De exemplu, **asp-for="Name"** va lega câmpul (atributul) Name din modelul care va fi transmis.

În cazul **editării**, **unde este necesar să preluam valorile existente în Model** pentru modificarea ulterioară a acestora, putem apela la helperul **Html.Editor** pentru a genera în mod automat câmpurile de editare și pentru a prelua automat aceste valori.

Formularul inițial pentru **editarea unui student**, implementat manual cu preluarea manuală a valorilor și alocarea acestora inputurilor HTML, prin intermediul atributului **value**, are următorul conținut:

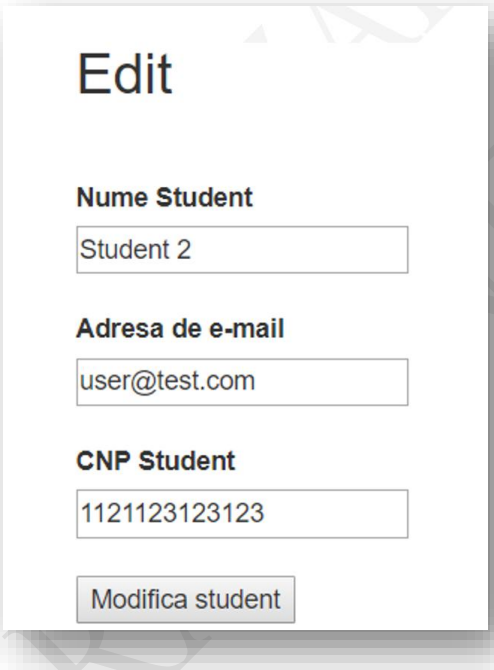
```
<form method="post" action="/Students/Edit/@ViewBag.Student.Id">

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

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

</form>
```

Acesta poate fi rescris prin intermediul **HTML Helpers**, 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")
```

Acest helper generează în mod automat câmpul necesar

```
    <br /><br />

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

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

</form>
```



Codul generat în consolă este următorul:

```
<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 că **Helper-ul Html.Editor** a generat toate câmpurile necesare pentru formularul de editare a studentului, conform definiției modelului **Student**. De asemenea, în formular se observă că toate atributele **value** asociate elementelor formularului au primit valorile modelului în mod automat (prin **Model Binding**).

Exemple pentru implementarea alternativă a Helperelor

Html.TextBox – generează un element de tipul TextBox → Devine input în noua versiune de utilizare a helperelor. În 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 – generează 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 utilizează pentru procesul de **Model Binding**.

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

- populareа unui select cu o listă de opțiuni (**asp-items**)
- setarea unei rute pentru un form **asp-controller** și **asp-action**
- setarea unui url pentru un un tag <a>: **asp-controller** și **asp-action**