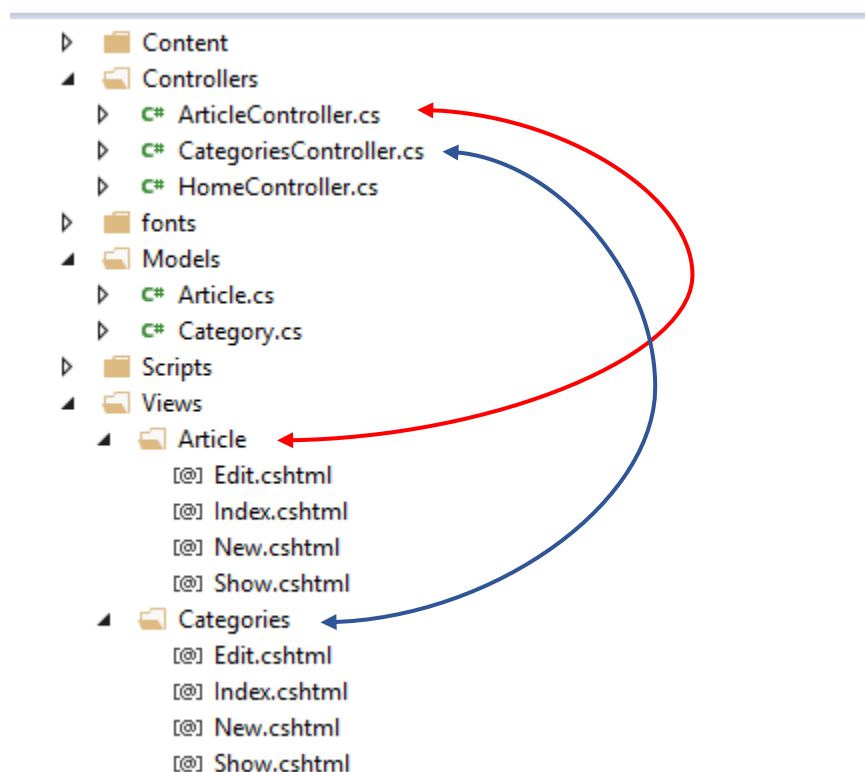


View. Razor. Trimiterea datelor catre View. Helpere pentru View.

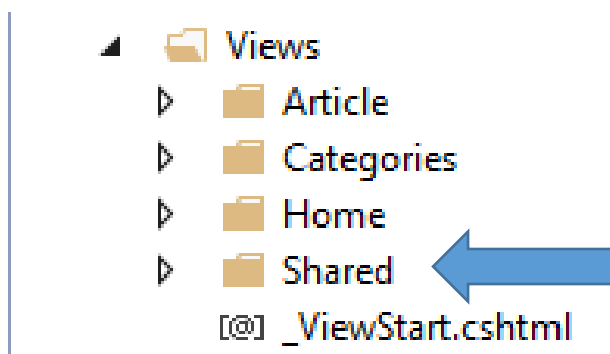
Ce este View-ul

View-ul reprezinta interfata cu utilizatorul. Acesta afiseaza date de la Model catre utilizatori si de asemenea, le ofera acestora posibilitatea modificarii datelor.

View-urile, in ASP.NET 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-urile si partialele care sunt partajate (folosite) de mai multe view-uri.



Razor

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** (acest lucru se intampla in ASP.NET WebForms prin intermediul caracterelor `<% %>`).

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 (ofera auto-completarea codului).

Exemple sintaxa Razor:

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

Exemplu:

```
<h2>@ViewBag.Title</h2>
```

2. Pentru a executa un bloc intreg de cod (mai multe linii de cod) este necesar sa folosim acolade, astfel: `@{ /* cod */ }`

Exemplu:

```
@{  
    ViewBag.Title = "Lista studenti";  
    ...  
}
```

3. In cadrul unui bloc de cod pentru a afisa o secventa de text se folosesc caracterele `@:` sau tag-urile `<text></text>`

Exemplu:

```
@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”.

4. Conditia IF incepe cu: `@if{ ... }`

Exemplu:

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

5. Loop-ul are urmatoarea sintaxa: `@for{ ... }`

Exemplu:

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

6. **@model** ofera posibilitatea afisarii valorilor modelului oriunde in View

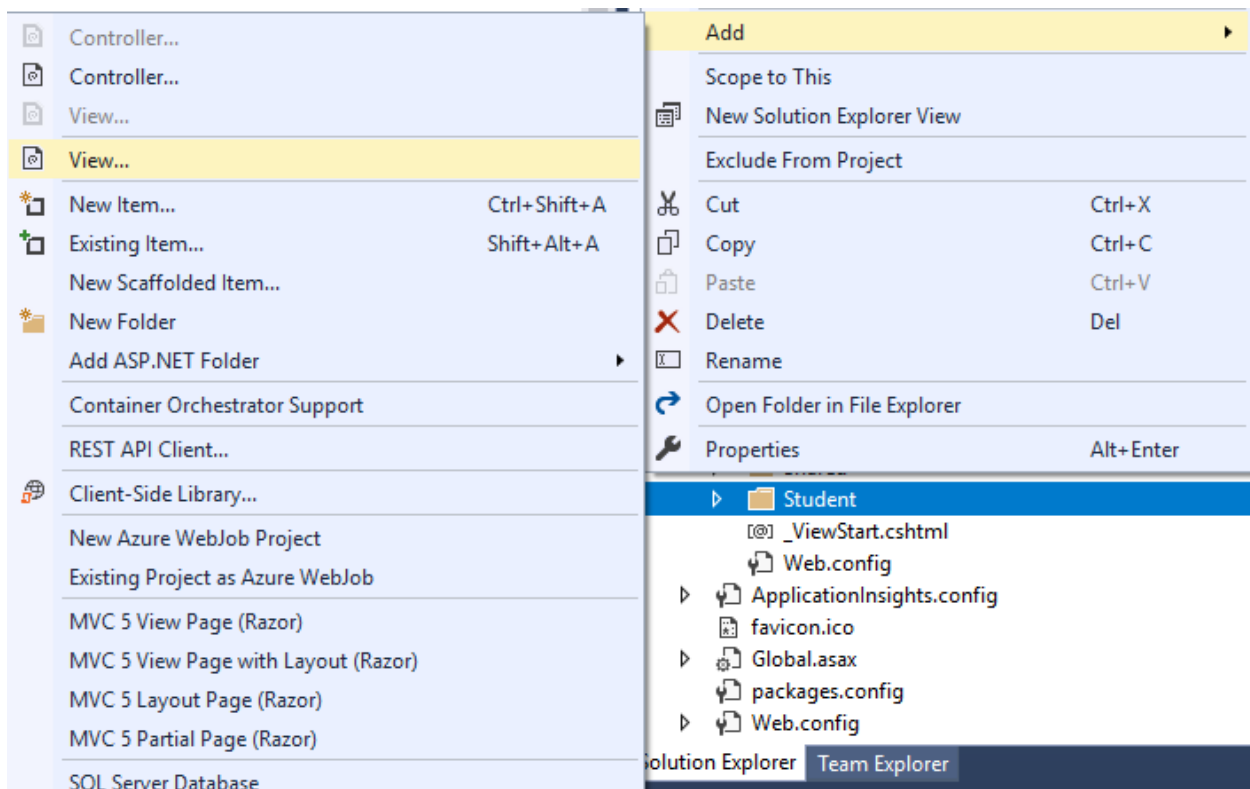
Exemplu:

```
@model Curs5.Models.Student
...
<h1>@Model.Name</h1>
<p>@Model.Email</p>
<p>@Model.CNP</p>
```

Crearea unui View

Se creeaza un View dupa cum urmeaza:

Click dreapta pe folderul corespunzator din folderul View (in exemplul urmator avem un folder Student, in folderul View, unde o sa cream toate view-urile necesare prelucrarii unei entitati Student) > **Add** > **View**.



Add View

View name: Index

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

Dupa crearea noului View se genereaza fisierul in folderul specificat.

Trimiterea datelor catre View (Model, ViewBag, ViewData, TempData)

De foarte multe ori este necesar sa trimitem informatii 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.

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

The diagram illustrates the data flow between a View and a Controller. On the left, a code snippet for a View shows the use of `@model` to include the `Student` model and the use of `@Model` to display its properties. On the right, a code snippet for a Controller shows the retrieval of a `Student` object from the database and its passing to the `View` method. Two blue arrows point from the Controller code to the View code: one points to the `@model` line, and the other points to the `@Model` lines in the Razor view.

```

1  @model Curs.Models.Student
2
3  @{
4      ViewBag.Title = "Afisare student";
5  }
6
7  <h1>@Model.Name</h1>
8  <p>@Model.Email</p>
9  <p>@Model.CNP</p>
10 <br />

```

Includem modelul necesar pentru afisare

Folosim @Model pentru preluarea si afisarea datelor

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

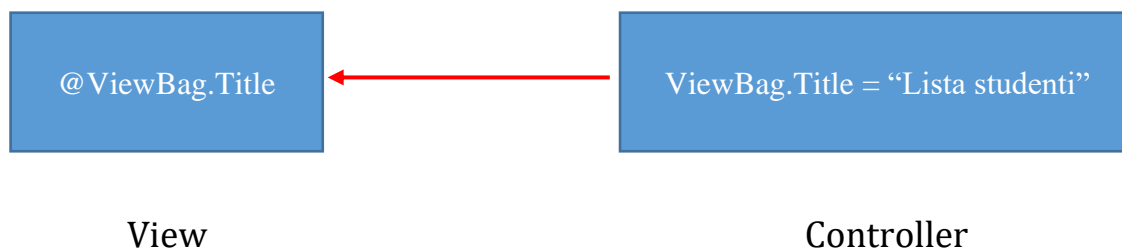
```

public ActionResult Show(int id)
{
    Student student = db.Students.Find(id);
    return View(student);
}

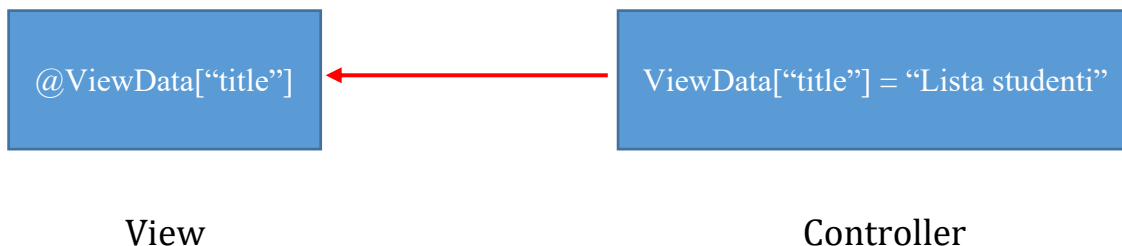
```

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

- **ViewBag** – acest helper ofera posibilitatea transferului de date intre Controller si View. Aceste date sunt reprezentate de 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** – acest helper 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 "titlu" cu ultima valoare alocata ("Titlu 2") pentru ambele variabile

Astfel, in View, **@ViewBag.title** cat si **@ViewData["title"]** vor afisa "Titlu 2".

- **TempData** – acest helper 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 (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 Delete, neafisand in acest caz un View. Aceasta metoda executa codul aferent stingerii din baza de date si redirectioneaza catre metoda Index.

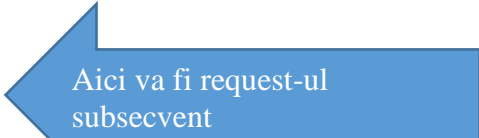
Pentru stergerea unui student, se vor executa 2 request-uri HTTP dupa cum urmeaza:

- Show (care afiseaza datele unui student si butonul de stergere) – aceasta este pagina unde apasam butonul de stergere. La apasarea butonului se va face un Request catre metoda Delete
- **[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 pe metoda Index si va afisa lista studentilor.

Pentru o mai buna experienta de utilizare (User Experience, prescurtat UX) va trebui sa afisam utilizatorului un mesaj ca resursa a fost stearsa cu succes. Acest lucru trebuie sa se intample in pagina Index. Acest lucru se poate realiza prin intermediul helper-ului **TempData** (deoarece avem doua request-uri subsecvente).

Codul se va modifica astfel:

```
[HttpDelete]
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");
}
```



De asemenea, in actiunea Index va trebui sa verificam daca variabila TempData["message"] este disponibila si are o valoare si apoi sa o afisam in View.



```

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

    ViewBag.Students = students;

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

```



In View-ul aferent metodei Index, vom afisa mesajul:

```
<h1>@ViewBag.message</h1>
```

Exemplu:

Lista studenti

Student 1

user@test.com

1121123123123

[Afisare student](#)

Pe pagina de student apasam butonul "Sterge studentul":

Student 1

user@test.com

1121123123123

[Modifica student](#)

Sterge studentul



Se va executa metoda **Delete** care va sterge studentul si va seta variabila temporara care va fi disponibila si afisata in metoda Index:

Lista studenti

Studentul cu numele Student 1 a fost sters din baza de date

Student 2

user@test.com

1121123123123

Afisare student

Helpere pentru View

ASP.NET MVC, 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 se folosesc de sistemul de binding** pentru a afisa valorile modelului in elementele de tip HTML (**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.RadioButton** – 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="/Student/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>CPN</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="/Student/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="/Student/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>
```

Creare student

Afisare formular de adaugare student

Nume Student

Adresa de e-mail

CNP Student

Adauga student

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 pastrarea sau modificarea acestora, putem apela la helperul **Html.Editor** pentru a genera in mod automat campurile de editare si pentru preluarea automata a acestor 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="/Student/Edit/@ViewBag.Student.StudentId">
    @Html.HttpMethodOverride(HttpVerbs.Put)
    <label>Nume</label>
    <br />
    <input type="text" name="Name" value="@ViewBag.Student.Name" />
    <br /><br />
    <label>Adresa e-mail</label>
    <br />
    <input type="text" name="Email" value="@ViewBag.Student.Email" />
    <br /><br />
    <label>CPN</label>
    <br />
    <input type="text" name="CNP" value="@ViewBag.Student.CNP" />
    <br />
    <button type="submit">Modifica student</button>
</form>
```

Acesta poate fi rescris prin intermediul helpere-lor astfel:

```
@model Cours8.Models.Student
<form method="post" action="/Student/Edit/@Model.StudentId">
    @Html.HttpMethodOverride(HttpVerbs.Put)
    @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="/Student/Edit/1">
  <input name="X-HTTP-Method-Override" type="hidden" value="PUT" />
  <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 attributele **value** asociate elementelor formularului au primit valorile modelului in mod automat (**prin Binding**).

Edit

Afisare formular de editare student - cu datele vechi ale studentului

Nume Student

Adresa de e-mail

CNP Student

Pe langa codul generat pentru formularul de editare impreuna cu valorile aferente din model, helperul `Html.Editor` a adaugat si attributele necesare pentru **validarea datelor modelului**, conorm definitiei acestuia.

Un alt exemplu pentru aceste helpere, este exemplul pentru generarea de elemente de tip **Dropdown** (element select din html). Sa consideram o aplicatie in care avem modelele **Article** si **Category**. Category reprezinta o lista de categorii care pot fi alocate unui articol. La crearea unui articol trebuie sa selectam categoria din care acesta face parte.

Folosind cod HTML, avand o lista de categorii din baza de date, putem sa generam elementul de tip select (dropdown) dupa cum urmeaza:

```
<select name="CategoryId">
  @foreach (var category in ViewBag.Categories)
  {
    <option
      value="@category.CategoryId">@category.CategoryName
    </option>
  }
</select>
```

Acest lucru poate fi simplificat folosind helper-ul, astfel:

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

Folosind helper-ul **@model** in View pentru modelul Article, putem adauga **@Html.DropDownListFor** pentru acest model. Acest helper (DropDownListFor) primeste urmatorii parametri:

- Primul parametru este o lambda expresie prin care se alege atributul modelului pentru care se va genera elementul de tip select (in acest caz pentru **CategoryId**)
- Al doilea parametru trebuie sa fie o lista de tipul **SelectList** cu elementele pentru care se va genera acest dropdown
- Al treilea element reprezinta o optiune default (Selectati categoria)
- Al patrulea element, este optional si reprezinta o lista de attribute aditionale pentru acest element generat

Pentru a putea obtine lista de categorii care va fi afisata in dropdown va fi necesar sa facem modificari asupra Modelului si a Controller-ului.

Astfel, in **Model** se adauga o noua proprietate in care se poate instantia o lista de categorii (aceasta trebuie sa fie de tipul

IEnumerable<SelectListItem>

```
// Se adauga acest atribut pentru a putea prelua toate categoriile unui  
//model in helper  
public IEnumerable<SelectListItem> Categories { get; set; }
```

De asemenea, se va modifica Controller-ul pentru a instantia aceasta proprietate cu lista tuturor categoriilor din baza de date:

```
public ActionResult New()  
{  
    Article article = new Article();  
    // preluam lista de categorii din metoda GetAllCategories()  
    article.Categories = GetAllCategories();  
    return View(article);  
}
```

Metoda GetAllCategories va returna o lista de tipul **IEnumerable<SelectListItem>** dupa cum urmeaza:

```
[NonAction]
public IEnumerable<SelectListItem> GetAllCategories()
{
    // generam o lista goala
    var selectList = new List<SelectListItem>();
    // Extragem toate categoriile din baza de date
    var categories = from cat in db.Categories select cat;
    // iteram prin categorii
    foreach(var category in categories)
    {
        // Adaugam in lista elementele necesare pentru dropdown
        selectList.Add(new SelectListItem
        {
            Value = category.CategoryId.ToString(),
            Text = category.CategoryName.ToString()
        });
    }
    // returnam lista de categorii
    return selectList;
}
```

Astfel, codul utilizat mai sus, prin intermediul helper-ului **Html.DropDownListFor** va genera urmatoarea secventa de cod HTML:

```
<select class="form-control" data-val="true" data-val-number="The field
CategoryId must be a number." data-val-required="The CategoryId field
is required." id="CategoryId" name="CategoryId">
    <option value="">Selectati categoria</option>
    <option value="2">Stiinta</option>
    <option value="8">Natura</option>
    <option value="9">Animale</option>
</select>
```

Analog, in cazul editarii, acest helper va genera automat lista de categorii posibile pentru alegere **si va selecta si categoria** deja existenta pentru modelul utilizat.

Selectati categoria

Natura ▼
Selectati categoria
Stiinta
Natura
Animale