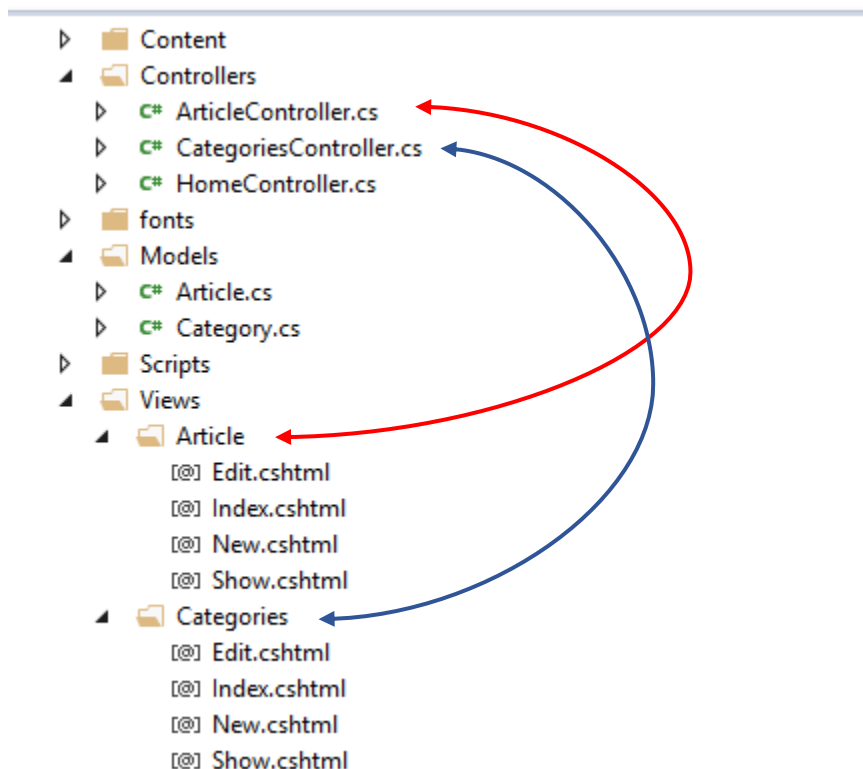


View. Razor. Trimiterea datelor catre View. Helpere pentru View. Validare. Layout View. Partial View.

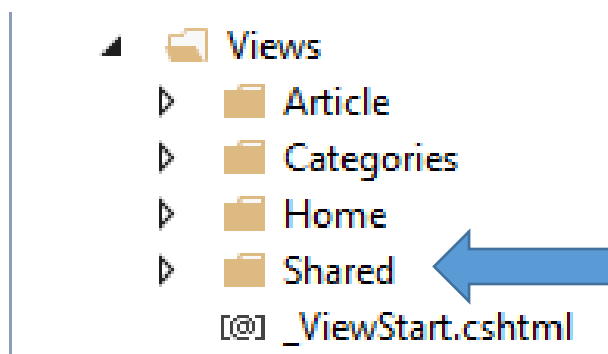
Ce este View-ul

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

Încă din ASP.NET MVC 3 motorul (engine-ul) de afisare 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 (acest lucru se întâmplă în ASP.NET WebForms prin intermediul caracterelor <% %>).

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 (oferă auto-completarea codului).

Exemple sintaxa Razor:

1. Se folosește @ pentru executarea codului server-side pe o singură 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

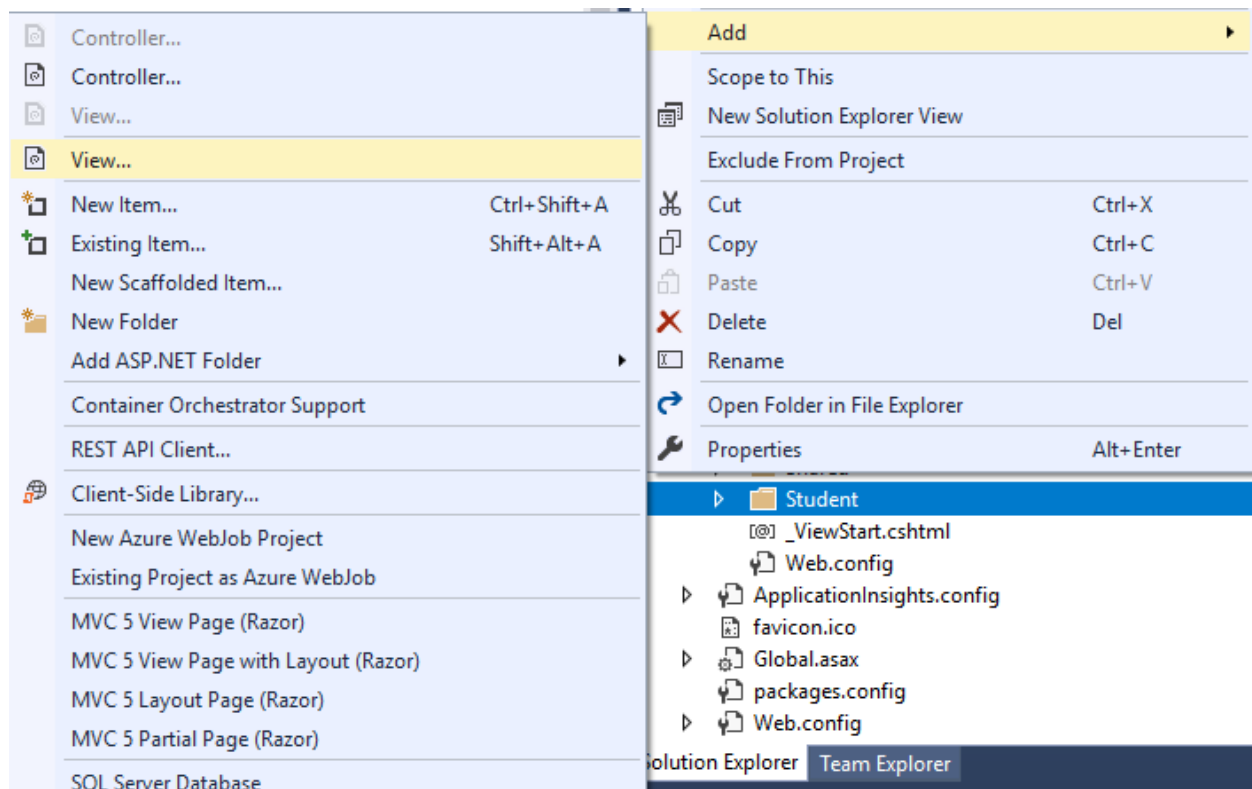
Exemplu:

```
@model Curs8.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:

Template:

Model class:

Data context class:

Options:

☐ Create as a partial view

☐ Reference script libraries

☒ Use a layout page:

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

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:

```
1  @model Curs8.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

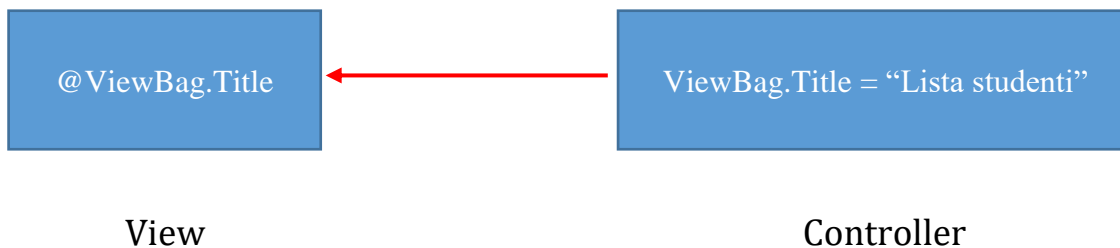
Folosim @Model pentru preluarea si afisarea datelor

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

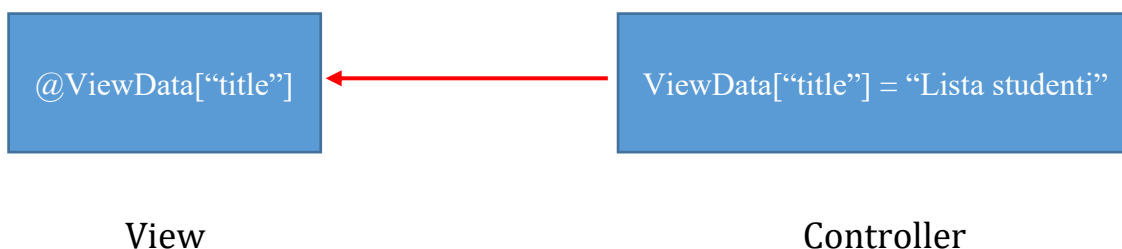
```
public ActionResult Show(int id)
{
    Student student = db.Students.Find(id);
    return View(student);
}
```

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. Ex: string-uri, array-uri, obiecte care nu fac parte din model)



- **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.



⚠ Atentie: 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 stergerii din baza de date si redirectioneaza catre metoda Index.

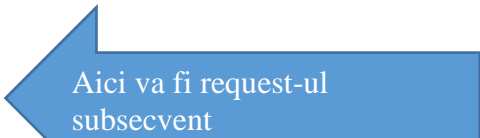
Pentru stergerea unui student, se vor executa 2 request-uri HTTP dupa cum urmeaza:

- Show (care afiseaza datele unui student si butonul de stergere) – aceasta este pagina unde apasam pe 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 stingerii 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");
}
```



Aici va fi request-ul subsecvent

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

```
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>
```

Lista tuturor studentilor:

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



Lista tuturor studentilor:

Se va executa metoda **delete** care va sterge studentul si va seta variabila temporara care va fi disponibila si afisata din nou 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

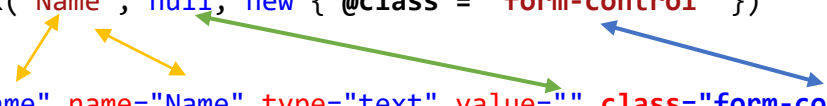
Pentru label, se genereaza urmatoarele elemente:

```
@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 vechi din 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 de 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 helper-elor 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 de acest cod 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 urmatoarii parametrii:

- 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 **dar va si selecta categoria** deja existenta pentru modelul utilizat.

Selectati categoria

Natura	▼
Selectati categoria	
Stiinta	
Natura	
Animale	

Validare

Validarea in ASP.NET MVC se face prin intermediul adaugarii atributelor necesare in Model. Atributele pentru validare sunt:

- Required
- StringLength
- Range
- RegularExpression
- CreditCard
- CustomValidation
- EmailAddress
- FileExtension
- MaxLength
- MinLength
- Phone
- DataType

Exemplu:

```
public class Student
{
    [Key]
    public int StudentId { get; set; }
    [Required]
    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)][MaxLength(13)][Required]
    public string CNP { get; set; }
    [StringLength(20)][DataType(DataType.Text)][Required]
    public string City { get; set; }

    public virtual ICollection<Marks> Marks { get; set; }
}
```

Pentru afisarea mesajelor in View, consideram urmatorul formular necesar pentru editarea unui student:

```
@model Curs8.Models.Student
<form method="post" action="/Student/Edit/@Model.StudentId">
    @Html.HttpMethodOverride(HttpVerbs.Put)
    @Html.HiddenFor(m => m.StudentId)
    @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 />
    @Html.Label("City", "Oras Student")
    <br />
    @Html.Editor("City")
    <br /><br />
    <button type="submit">Modifica student</button>
</form>
```

Pentru afisarea unui mesaj de validare pentru fiecare camp in parte, se poate folosi helper-ul **@Html.ValidationMessageFor**. Acesta primeste 3 parametrii, 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 default de validare (in functie de tipul validarii care a esuat)
- Al treilea parametru este optional si reprezinta o lista de attribute care poate fi adaugata mesajului afisat

@Html.Editor("Name") si **@Html.EditorFor(m => m.Name)** sunt echivalente si se poate utiliza orice varianta, acest lucru aplicandu-se tuturor helper-elor.

Prin adaugarea acestor helpere formularul de mai sus devine:

```
@using (Html.BeginForm())
{
    @Html.HttpMethodOverride(HttpVerbs.Put)
    @Html.HiddenFor(model => model.StudentId)
    <br />
    @Html.Label("Name", "Nume Student")
    <br />
    @Html.EditorFor(m => m.Name)
    @Html.ValidationMessageFor(m => m.Name, "Numele este obligatoriu", new {
@class = "text-danger " })

    <br /><br />
    @Html.Label("Email", "Adresa de e-mail")
    <br />
    @Html.EditorFor(m => m.Email)
    @Html.ValidationMessageFor(model => model.Email, "Email obligatoriu", new
{ @class = "text-danger " })
    <br /><br />
    @Html.Label("CNP", "CNP Student")
    <br />
    @Html.EditorFor(m => m.CNP)
    @Html.ValidationMessageFor(model => model.CNP, null, new { @class =
"text-danger " })
    <br /><br />
    @Html.Label("City", "Oras Student")
    <br />
    @Html.EditorFor(m => m.City)
    @Html.ValidationMessageFor(model => model.City, null, new { @class =
"text-danger " })
    <br /><br />
    <button type="submit">Modifica student</button>
}
}
```

In ecranele de mai jos putem sa vedem diferite mesaje de validare pentru acest View:

Edit

Afisare formular de editare student - cu datele vechi ale studentului

Nume Student

Numele este obligatoriu

Adresa de e-mail

Email obligatoriu

CNP Student

The CNP field is required.

Oras Student

The City field is required.

Modifica student

Edit

Afisare formular de editare student - cu datele vechi ale studentului

Nume Student

Adresa de e-mail

Email obligatoriu

CNP Student

The field CNP must be a string or array type with a minimum length of '13'.

Oras Student

The City field is required.

Pentru a schimba mesajul de validare, acesta se poate face din model conform declaratiei urmatoare:

```
[Required(ErrorMessage = "Campul e-mail este obligatoriu")][EmailAddress(ErrorMessage = "Adresa de e-mail nu este valida")]
```

Sau se poate face in view prin intermediul helper-ului astfel:

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

Pentru a afisa corect mesajul de eroare, doar cand validarea datelor nu este corecta, este necesar sa adaugam urmatoarele linii de cod in fisierul Site.css

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


Exista posibilitatea afisarii unui sumar cu toate erorile intervenite in timpul validarii. Acest lucru se face prin intermediul helper-ului **Html.ValidationSummary** astfel:

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

Edit

Afisare formular de editare student - cu datele vechi ale studentului

- The Name field is required.
- Campul e-mail este obligatoriu
- The CNP field is required.
- The City field is required.

Nume Student

Numele este obligatoriu

Pentru functionarea corecta a validatorului cat si pentru identificarea corecta a datelor in partea de 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 validările au trecut cu succes si putem salva modificările.

```
[HttpPut]
public ActionResult Edit(int id, Student requestStudent)
{
    try
    {
        if (ModelState.IsValid)
        {
            Student student = db.Students.Find(id);
            if (TryUpdateModel(student))
            {
                student.Name = requestStudent.Name;
                student.Email = requestStudent.Email;
                student.CNP = requestStudent.CNP;
                student.City = requestStudent.City;
                db.SaveChanges();
            }
            return RedirectToAction("Index");
        }
        else
        {
            return View();
        }
    }
    catch (Exception e) {
        return View();
    }
}
```

Layout View (Master page)

O aplicatie web MVC contine foarte multe componente comune tuturor paginilor: Header, Footer, Meniuri, etc. Aceste componente nu se modifica de la pagina la pagina si trebuie incluse. Pentru a facilita acest lucru se poate folosi un View global numit **Layout**.

Acest view este identic cu un MasterPage din WebForms. 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 MVC, 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.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home", new {
area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
```

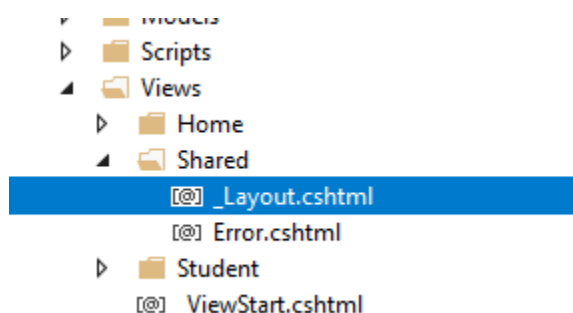
```

        <li>@Html.ActionLink("About", "About", "Home")</li>
        <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
    </ul>
</div>
</div>
</div>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
    </footer>
</div>

@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
</body>
</html>

```

Locul in care se includ View-urile create



Dupa cum putem observa, in folderul Views exista un fisier numit **_ViewStart.cshtml**. Acest fisier este folosit de motorul Razor pentru a seta layout-ul default pentru toate view-urile.

```

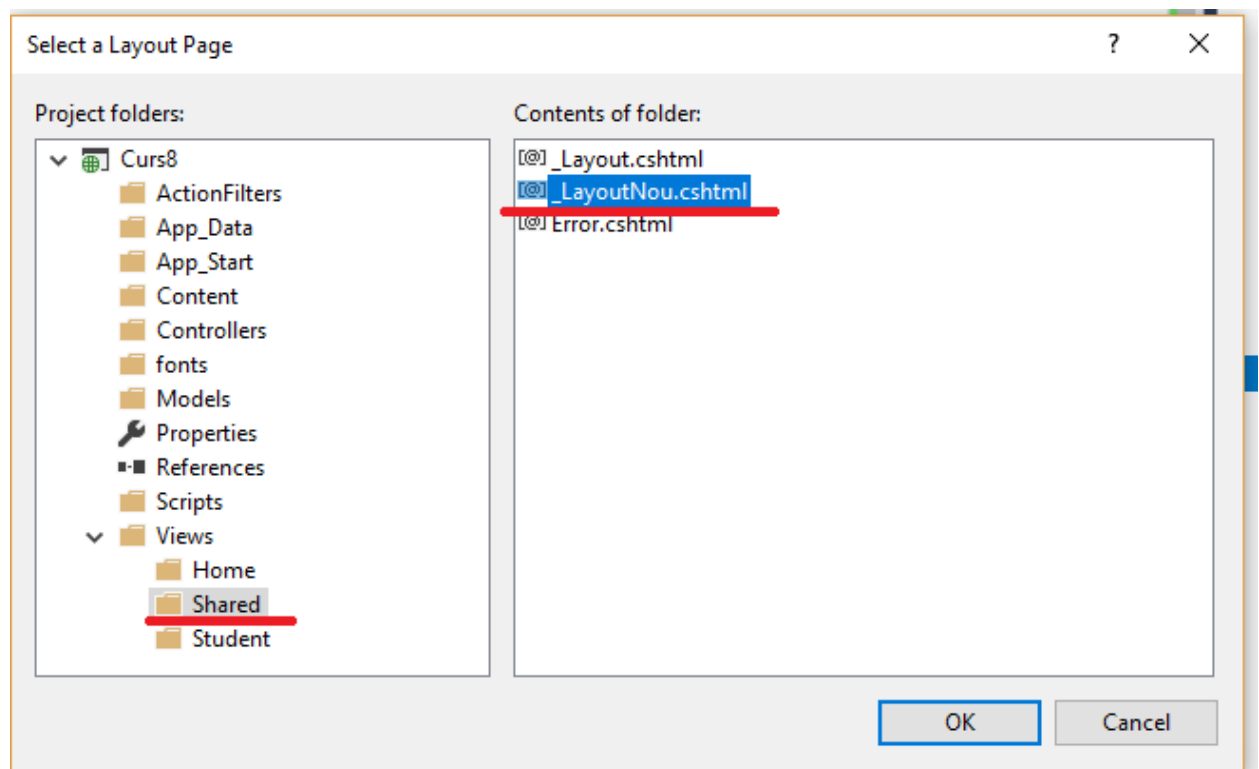
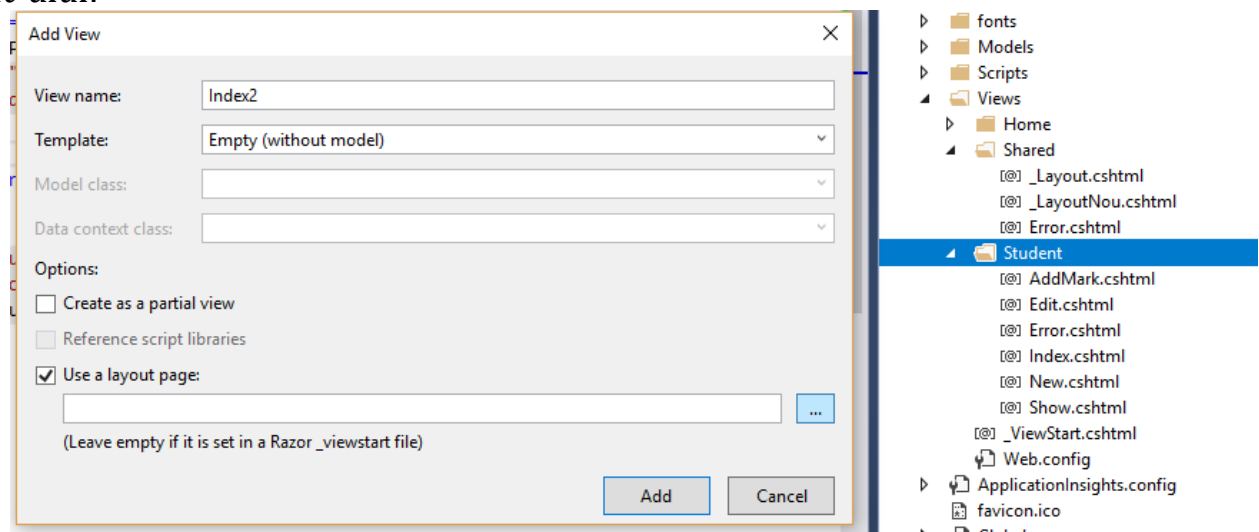
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

```

Astfel, layout-ul **_Layout.cshtml** din folderul **Shared** va fi layout-ul prestabilit pentru toate view-urile. Acest lucru se poate suprascrie in momentul adaugarii unui nou View (prin selectarea layout-ului conform print screen-ului de mai jos) sau intr-un view existent prin suprascrierea valorii variabilei Layout.

Presupunem ca am adaugat un nou layout in folderul Views/**Shared** cu numele **_LayoutNou.cshtml**. Pentru adaugarea unui view care sa contina

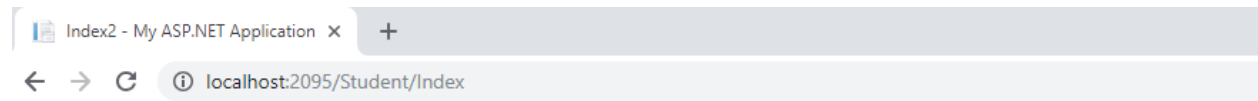
acest layout, selectam folderul dorit pentru adaugarea view-ului iar in dialog-ul aparut selectam **"Use a layout page"** si apasam pe butonul de cautare a layout-ului:



Putem sa observam codul generat pentru noul view. Acesta contine variabila Layout a carei valoare este calea catre noul layout.

```
@{  
    ViewBag.Title = "Index2";  
    Layout = "~/Views/Shared/_LayoutNou.cshtml";  
}
```

```
<h2>Index2</h2>
```



Index2

Student 1

user@test.com

1121123123123

[Afisare student](#)

[Adauga alt student](#)

Partial View

Partialele reprezinta bucati de view care pot fi refolosite in una sau mai multe pagini. In aplicatiile MVC codul poate fi reutilizat pentru a optimiza timpul de scriere si pentru a obtine aceleasi rezultate in toate paginile unde anumite informatii se afiseaza in acelasi mod.

Acestea reprezinta bucati de view care contin o anumita secventa de cod. Ele pot fi incluse intr-un view sau intr-un layout pentru a fi afisate. Sa consideram exemplul listarii tuturor studentilor. Pentru fiecare student, avem de afisat numele, e-mail-ul si CNP-ul.

```
@foreach (var student in ViewBag.Students)
{
    <p>@student.Name</p>
    <p>@student.Email</p>
    <p>@student.CNP</p>
    <a href="/Student/Show/@student.StudentId">Afisare student</a>
    <br />
    <hr />
    <br />
}
```

Acest cod, poate fi folosit si pe pagina "Show" pentru afisarea informatiilor studentului. Pentru reutilizarea codului, mutam aceasta secventa de afisare a informatiilor in cadrul unui partial.

Adaugarea unui partial se face prin click dreapta pe folderul "Shared" (nu este necesar ca partialul sa fie plasat in folderul Shared, poate sa fie plasat in orice folder) > Add > View. In dialogul de adaugare a View-ului selectam "Create as partial view"

În partialul creat adaugam codul necesar pentru afisarea informatiilor studentului:

```
<p>@Model.Name</p>
<p>@Model.Email</p>
<p>@Model.CNP</p>
<a href="/Student/Show/@Model.StudentId">Afisare student</a>
<br />
<hr />
```

În metoda Index, modificam codul pentru loop astfel incat sa includa partialul pentru fiecare student din baza de date:

```
@foreach (Curs8.Models.Student student in ViewBag.Students)
{
    @Html.Partial("StudentInfo", student);
}
```

Partialul primeste al doilea parametru, un obiect 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.

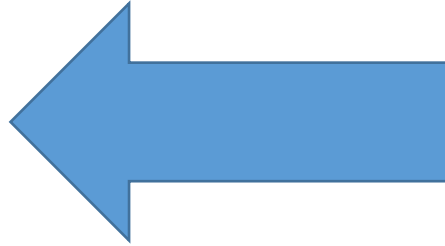
Lista studenti

Student 1

user@test.com

1121123123123

[Afisare student](#)

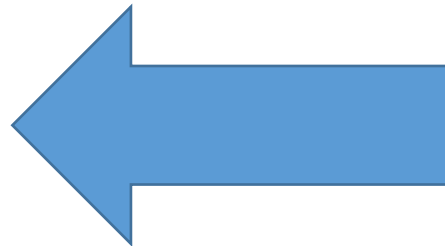


Student 2

student@university.tld

1121123123123

[Afisare student](#)



In cazul modificarii partialului, modificarile se reflecta asupra tuturor intrarilor:

```
<div class="panel panel-default">
  <div class="panel-heading">@Model.Name</div>
  <div class="panel-body">
    Studentul are CNP <strong>@Model.CNP</strong>
    <br />
    <span class="label label-success">@Model.Email</span>
    <br />
    <i class="glyphicon glyphicon-globe"></i> @Model.City
  </div>
  <div class="panel-footer">
    <a class="btn btn-sm btn-success"
href="/Student/Show/@Model.StudentId">Afisare student</a>
  </div>
</div>
<br />
```

Lista studenti

Student 1
Studentul are CNP 1121123123123 user@test.com Bucharest
Afisare student

Student 2
Studentul are CNP 1121123123123 student@university.tld Bucharest
Afisare student

Acelasi partial poate fi folosit si in pagina de afisare a studentului.
Pagina de afisare a studentului inainte de adaugarea partialului:

Student 1


user@test.com

1121123123123

[Modifica student](#)

Sterge studentul

Dupa includerea partialului in pagina de afisare a informatiilor studentului obtinem urmatorul rezultat:

Student 1
Studentul are CNP 1121123123123
user@test.com
 Bucharest
Afisare student

[Modifica student](#)

Sterge studentul

[Inapoi la lista studentilor](#)

[Adauga alt student](#)