View. Validare. Layout View. Partial View.

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 Curs6.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 parametri, 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 atribute care poate fi adaugata mesajului afisat;

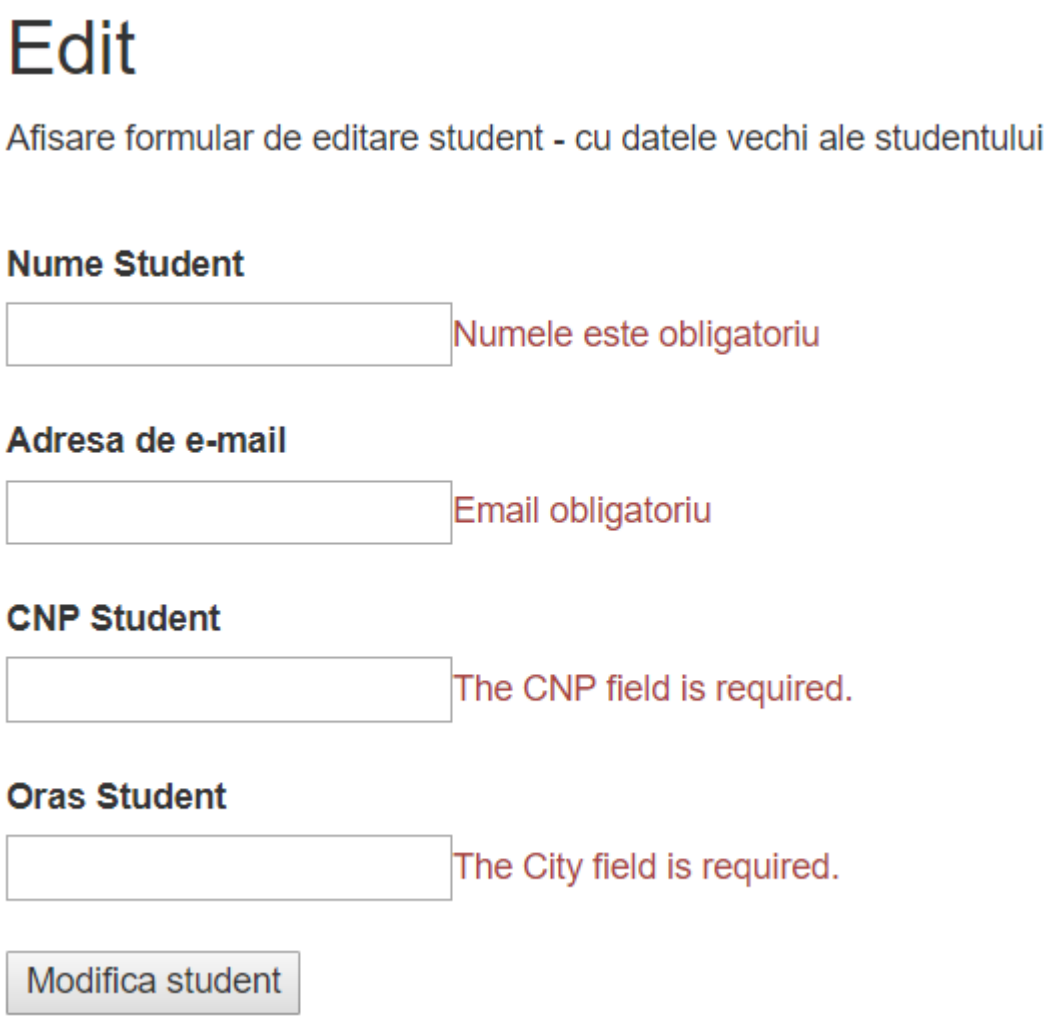
/!\OBSERVATIE

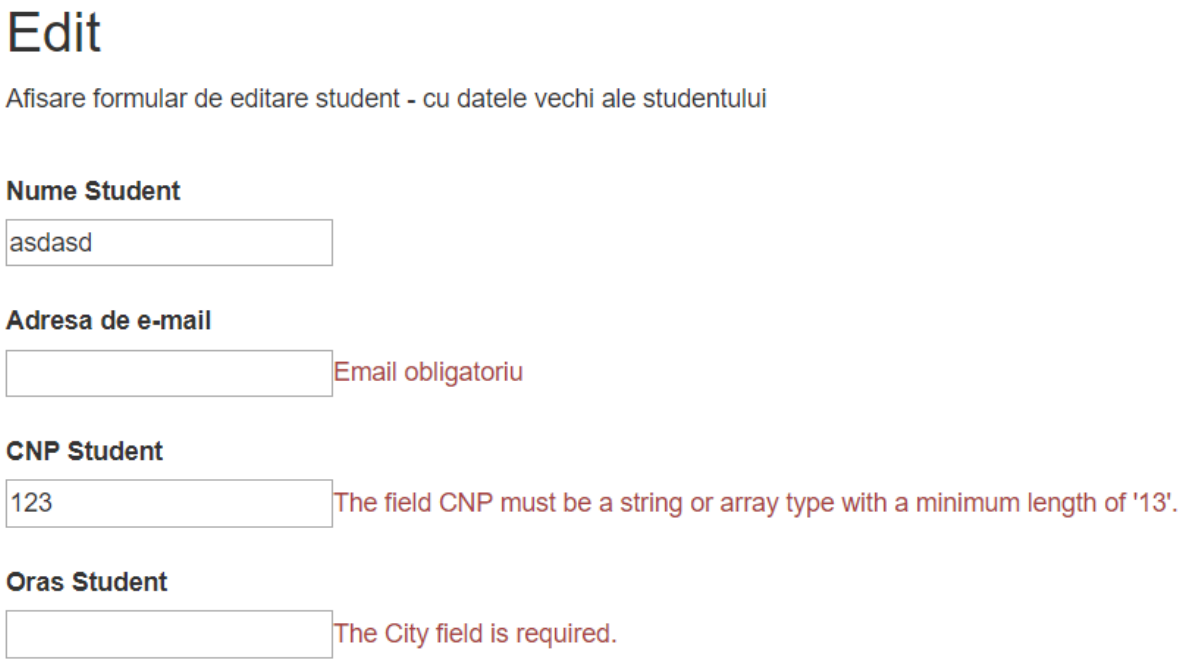
`@Html.Editor("Name")` si `@Html.EditorFor(m => m.Name)` sunt echivalente si se poate utiliza orice varianta, acest lucru aplicandu-se tuturor helpere-lor.

Prin adaugarea acestor helpere, formularul anterior devine:

```
@using (Html.BeginForm(actionName:"Edit", controllerName:"Student"))
{
    @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:





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:

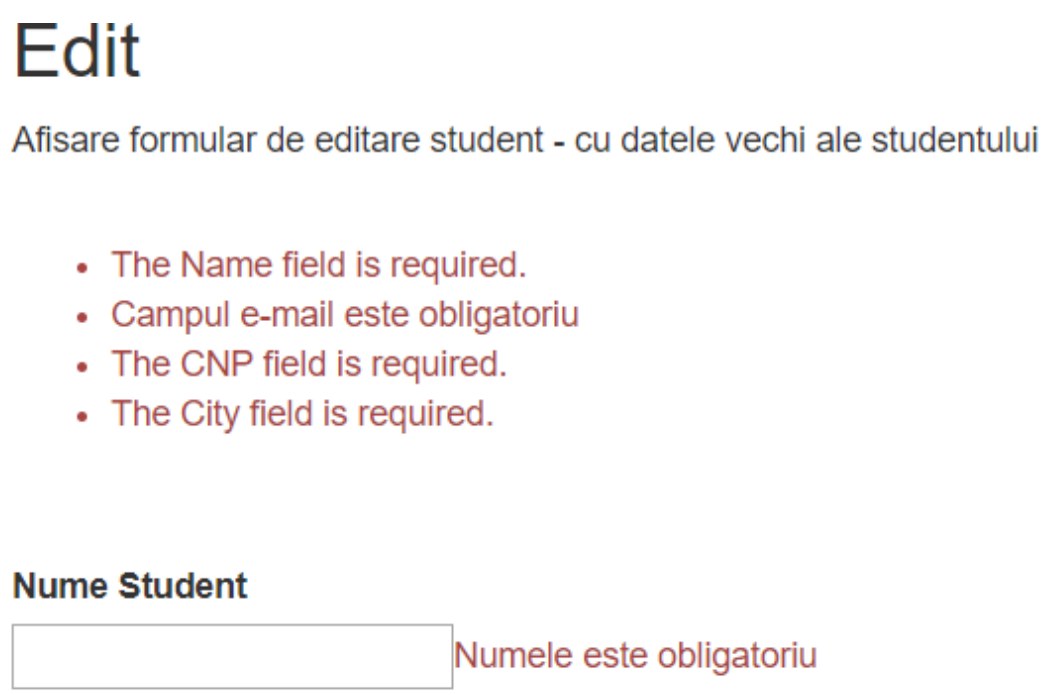
```
@Html.ValidationMessageFor(m => m.Name, "Numele este obligatoriu", new {
@class = "text-danger " })
```

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" })
```



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 validarile au trecut cu succes si putem salva modificarile.

```
[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(requestStudent);
        }

    } catch(Exception e) {
        return View(requestStudent);
    }
}
```

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, iar repetarea scrierii aceluiasi cod devinde redundanta. Pentru a facilita acest lucru se poate folosi un View global numit **Layout**.

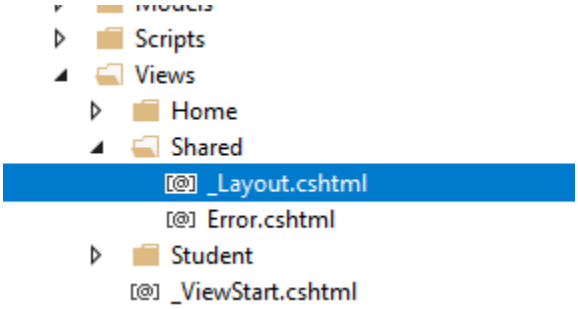
Acest view este identic cu un MasterPage din WebForms. Permite scrierea unui cod comun pentru toate paginile, cat si un Placeholder in care se va include continutul celorlalte pagini. Acest placeholder este definit prin intermediul variabilei **@RenderBody()**. Locul in care este plasata aceasta variabila 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

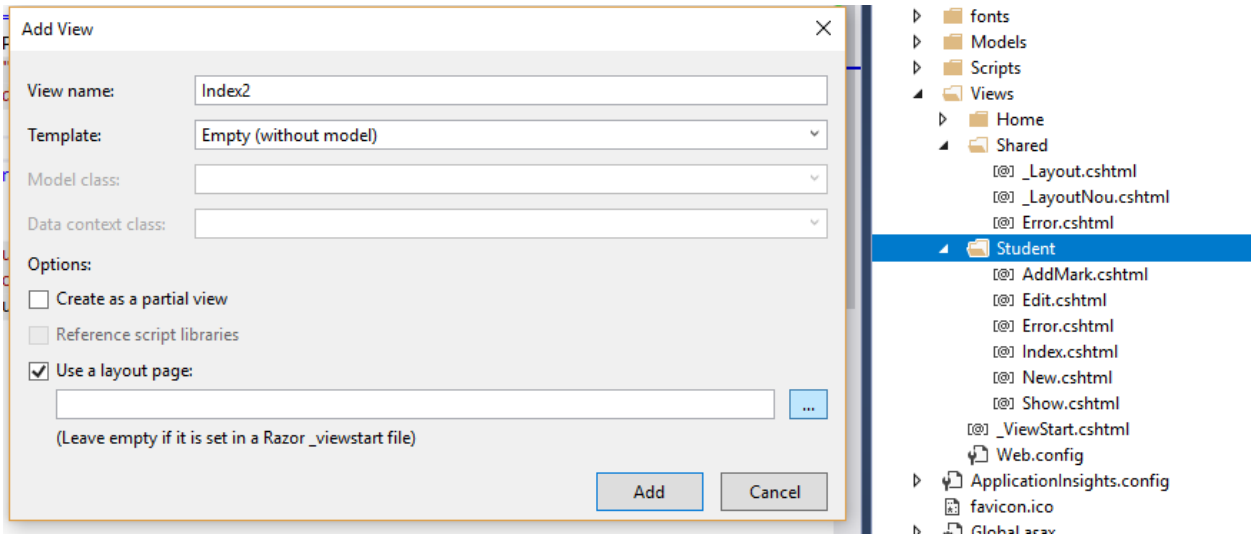


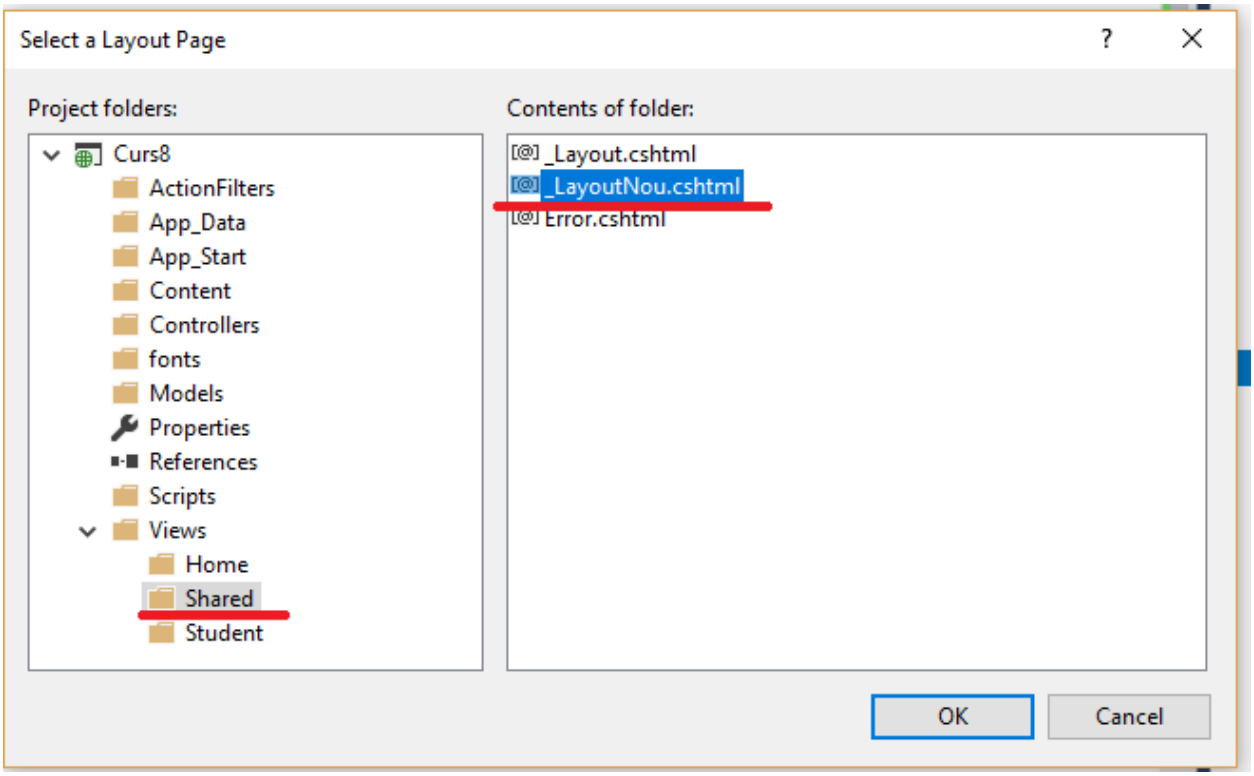
Dupa cum putem observa, in folderul Views exista un fisier numit **_ViewStart.cshtml**. Acest fisier este folosit de motorul Razor pentru a seta layout-ul default pentru toate View-urile.

```
@{
    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.

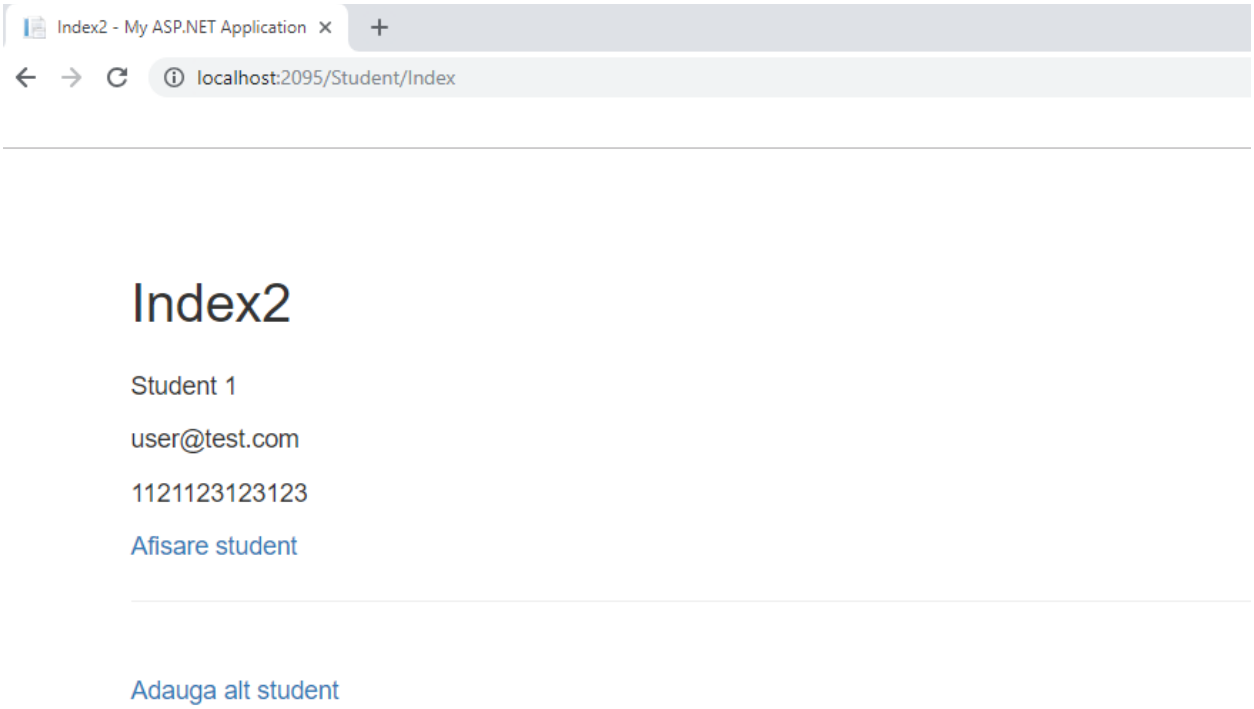
Presupunem ca am adaugat un nou layout in folderul Views/**Shared** cu numele **_LayoutNou.cshtml**. Pentru adaugarea unui View care sa contina acest layout, selectam folderul dorit pentru adaugarea View-ului, iar in dialog-ul aparut selectam **"Use a layout page"** si apasam pe butonul de cautare a 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>
```



Partial View

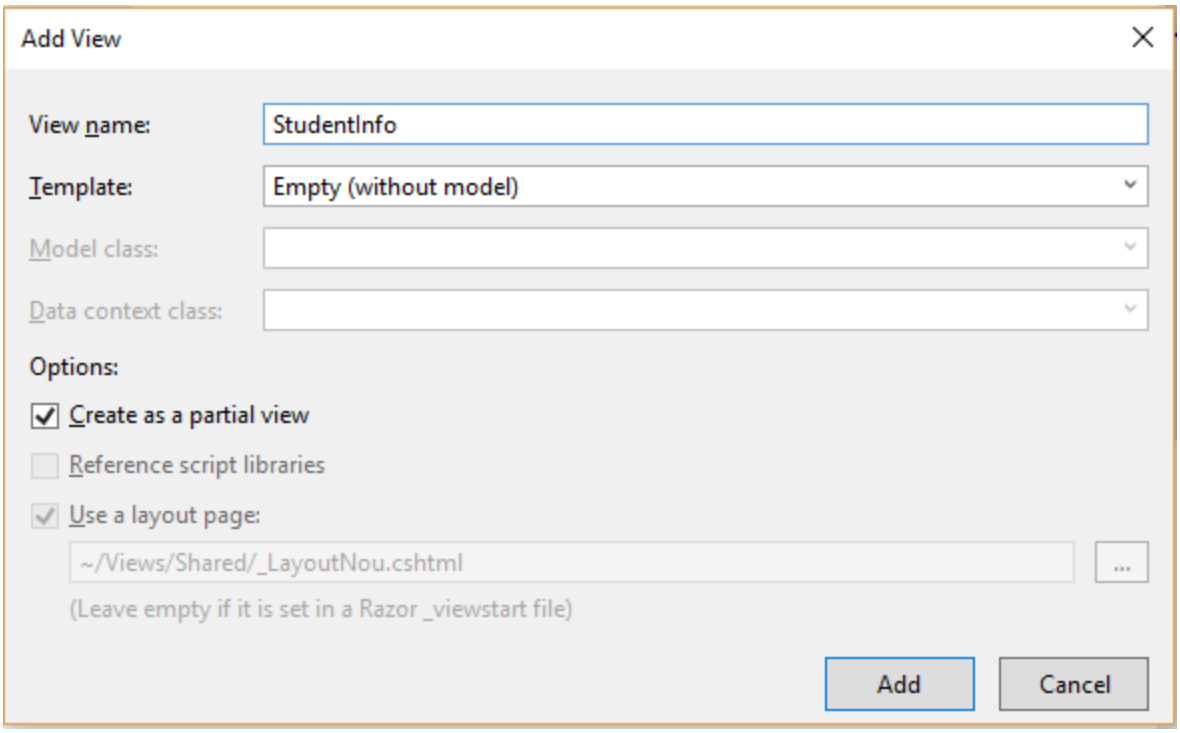
Partialele reprezinta bucati de View care pot fi refolosite in una sau mai multe pagini. In aplicatiile MVC codul poate fi reutilizat pentru a optimiza timpul de scriere si pentru a obtine aceleasi rezultate in toate paginile unde anumite informatii se afiseaza in acelasi mod.

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 fereastra de adaugare a View-ului selectam “**Create as partial view**”.



In 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 />
```

In 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.

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 />
```

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

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

Tips and Tricks

- Feedback-ul de validare trebuie implementat vizibil. Daca un camp are eroare este foarte bine ca acesta sa fie evidentiat cu un border rosu si o culoare de fundal rosie

- **/!\ Pentru toate erorile trebuie folosit font de culoare rosie. Erorile pot fi insotite de o iconita reprezentativa sub forma unui semn de exclamare**

- **Pentru mesajele de informare trebuie folosit un font de culoare albastra. Mesajele de informare pot fi insotite de o iconita reprezentativa sub forma unui (i)**

- **✓ Pentru mesajele de succes este necesara folosirea unui font verde. Acestea pot fi insotite de o iconita sub forma de bifa.**

- Pentru erorile de validare se va folosi aceeasi pagina din care utilizatorul face un request. O pagina universala pentru toate erorile din aplicatie inseamna un user experience slab

- Validarea instanta (in client, prin intermediul JavaScript – atunci cand mesajele de eroare apar imediat dupa completarea unui camp) este foarte buna

- Implementarea Captcha ajuta la eliminarea spam-ului. Desi este un pas in plus in experienta utilizatorilor finali, acesta imbunatateste considerabil informatiile ajunse la server