

## Dezvoltarea Aplicatiilor Web-Anul 2

### Curs 4

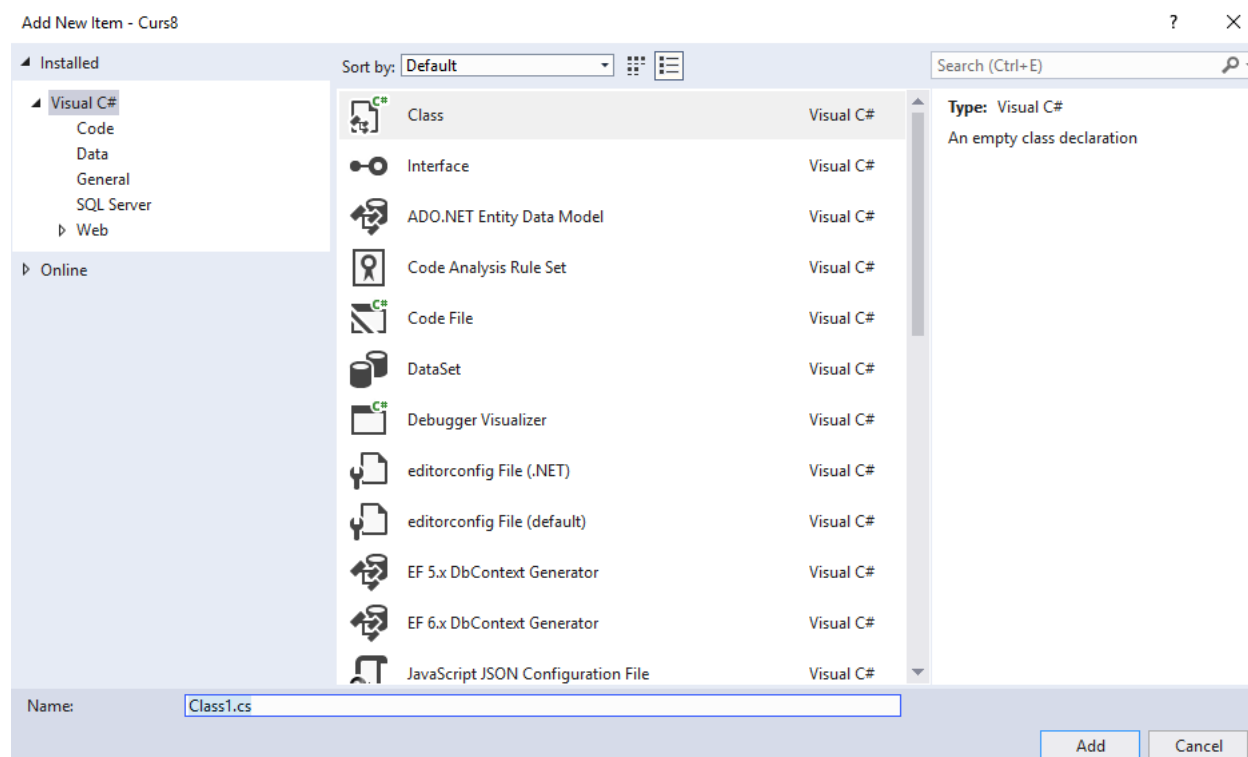
---

## Model. Model binding – binding intre Model si Controller. Entity Framework – Code First Database

### Ce este Modelul

In arhitectura MVC **Modelul** reprezinta datele aplicatiei si logica de business a acesteia. Acest strat utilizeaza si proceseaza datele aplicatiei. Datele sunt stocate intr-un mod persistent (baza de date) prin intermediul acestor obiecte. Modelul ofera accesul la date prin intermediul **atributelor publice ale clasei**. Locul acestora este in folderul “**Models**” din root-ul aplicatiei.

Adaugarea unui model se face prin click dreapta pe folderul Models > Add > Class.



Clasa nou creata va reprezenta modelul aplicatiei. In aceasta, vom defini proprietatile necesare:

```
public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string CNP { get; set; }
}
```

Astfel, am definit modelul Student care are urmatoarele proprietati:

- **StudentId** – de tip int care reprezinta ID-ul studentului
- **Name** - de tip string care reprezinta numele studentului
- **Email** – de tip string care reprezinta o adresa de e-mail a studentului
- **CNP** – de tip string care reprezinta CNP-ul studentului.  
(Aceasta proprietate a fost definita de tip STRING deoarece spatiul alocat pentru INT nu suporta valori de 13 caractere. De asemenea, definit ca STRING putem sa procesam caracter cu caracter aceasta proprietate pentru calcule ulterioare. Ex: extragere data de nastere, etc. )

## Model Binding

In ASP.NET MVC **model binding** ne permite sa facem legatura intre request-urile de tip HTTP si un model. Model binding este procesul de creare a obiectelor folosind datele trimise de browser printr-un request HTTP (prin intermediul formularelor din View).

Model binding este o legatura intre request-urile HTTP si metodele unui Controller (Actiuni). Deoarece datele trimise prin POST sau GET ajung intotdeauna la Controller, acest mecanism de binding **leaga in mod automat variabilele de request** cu **atributele publice ale modelului**. Aceasta mapare se va face dupa **numele atributelor modelului**. **Este necesar ca numele campurilor din View sa coincida cu numele atributelor**, ca binding-ul sa functioneze.

Pentru a exemplifica model binding presupunem ca avem o actiune de creare a unui student. Astfel, in Controller-ul **StudentsController** vom avea doua actiuni:

- **New** (prin metoda **GET**) care va afisa formularul de adaugare a datelor unui student
- **New** (prin metoda **POST**) care va salva in baza de date datele studentului

```
// implicit GET
public ActionResult New()
{
    return View();
}

[HttpPost]
public ActionResult New(Student student)
{
    // ... cod creare student ...
    var name = student.Name;
    var cnp = student.CNP;
    var email = student.Email;

    return View("NewPostMethod");
}
```

Observam ca in metoda New (prin POST) parametrul primit de metoda este un obiect de tipul **Student** (de tipul modelului creat anterior). Astfel, in momentul trimiterii request-ului catre server, model binding va face legatura in mod automat a parametrilor formularului cu attributele modelului.

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



Parametrii care se vor trimite prin request la controller

Deoarece parametrii trimisi coincid cu numele atributelor modelului **Student** si datorita faptului ca metoda New primeste ca parametru un obiect de tipul Student, model binding face automat corelarea acestora:

## Creare

Afisare formular de adaugare student

Nume

Adresa e-mail

CNP

Adauga student

| student   | {Curs1.Models.Student}   | Curs1.Models.Student |
|-----------|--------------------------|----------------------|
| CNP       | "123456789012"           | string               |
| Email     | "student@university.tld" | string               |
| Name      | "Student 1"              | string               |
| StudentId | 0                        | int                  |

Imaginea de mai sus ilustreaza efectul model binding. Datele trimise prin intermediul formularului au instantiat in mod automat un obiect de tipul Student.

Exista cazuri in care vrem **sa includem** sau **sa excludem** anumite attribute din binding. Pentru a face acest lucru, framework-ul ASP.NET MVC ofera posibilitatea specificarii elementelor pentru care se va face bind cu ajutorul atributului **Bind**:

```
public ActionResult New([Bind(Include = "Name, Email")] Student student)
```

Linia anterioara de cod va instantia obiectul student doar cu attributele Name si Email. Restul atributelor (CNP, StudentId) nu vor fi instantiate folosind valorile din request.

Analog, pentru **excluderea** unuia sau a mai multor attribute se poate folosi directiva **Exclude**:

```
public ActionResult New([Bind(Exclude = "CNP")] Student student)
```

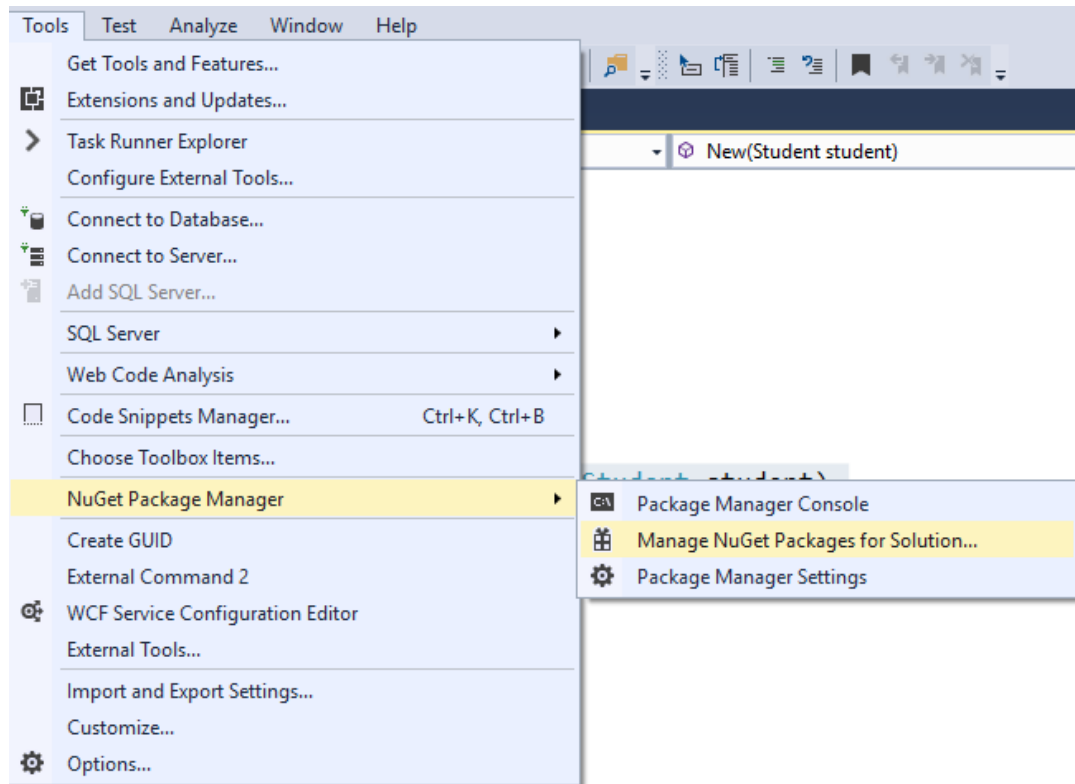
Astfel, se va face binding pentru toti parametrii, mai putin pentru atributul CNP.

## Entity Framework

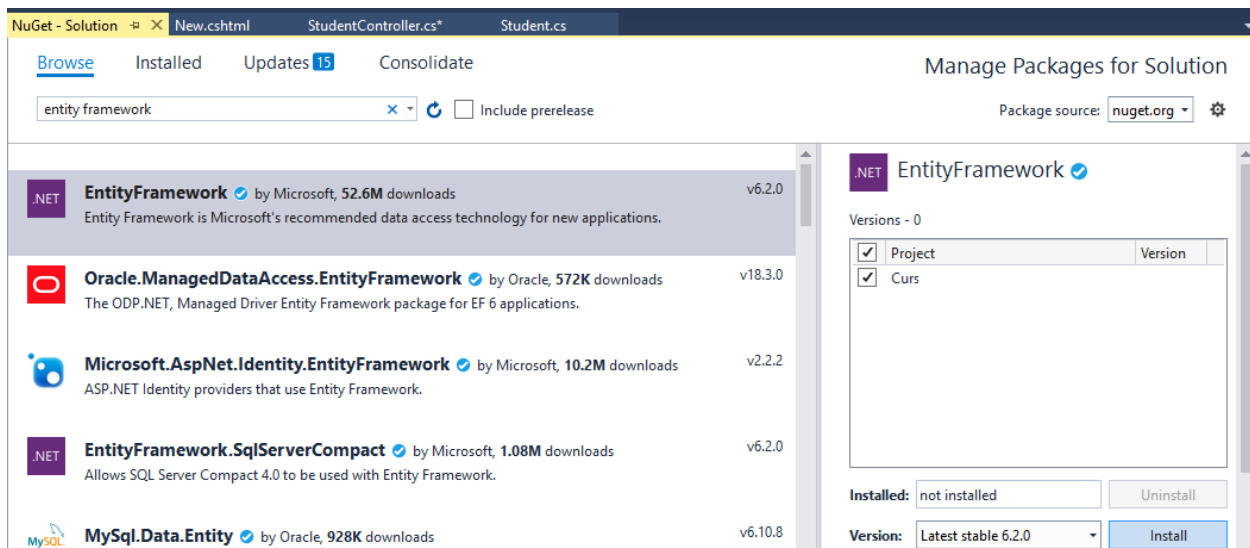
Pentru stocarea datelor in ASP.NET MVC se foloseste o tehnologie numita **Entity Framework**. Aceasta tehnologie este necesara pentru a corela modelele cu o baza de date.

EntityFramework este un pachet care poate fi instalat folosind **NuGet** si care suporta **tehnica Code First**. Tehnica Code First ofera posibilitatea dezvoltatorilor de a scrie clase (modele) prin intermediul carora baza de date va fi generata automat. Acest lucru duce la o dezvoltare curata si rapida a aplicatiilor cu baze de date.

Pentru instalarea Entity Framework (EF) accesam meniul **Tools > NuGet package manager > Manage NuGet Packages for solution**



In meniul aparut, cautam (selectam Browse) **Entity Framework**, alegem framework-ul, iar in partea dreapta selectam proiectul activ si apasam pe butonul Install:



In urmatoarea fereastra apasam “OK” si “I Agree” in momentul in care ne apare confirmarea pentru licenta pachetului.

Dupa instalarea corecta a acestuia, in consola vom avea urmatorul rezultat:

```
Successfully installed 'EntityFramework 6.2.0' to Curs  
Executing nuget actions took 11.68 sec  
Time Elapsed: 00:00:12.6348380  
===== Finished =====
```

In acest moment, proiectul nostru contine framework-ul Entity pe care il vom utiliza in continuare.

## Adaugarea Entity Framework in cadrul modelelor

Pentru a putea adauga layer-ul de conexiune cu baza de date in cadrul unui model, este necesara adaugarea unei noi clase in codul acestuia.

```
namespace Curs .Models  
{  
    public class Student  
    {  
        public int StudentId { get; set; }  
        public string Name { get; set; }  
        public string Email { get; set; }  
        public string CNP { get; set; }  
    }  
  
    public class StudentDbContext : DbContext  
    {  
        public StudentDbContext() : base("DBConnectionString") { }  
        public DbSet<Student> Students { get; set; }  
    }  
}
```

Clasa **StudentDbContext** mosteneste clasa de baza **DbContext** din Entity Framework. Aceasta clasa va face in mod automat conexiunea cu baza de date, crearea tabelului daca acesta nu exista si contine o metoda

**DbSet<Student> Students { get; set; }** prin intermediul careia vom avea acces la intrarile din baza de date.

Metoda DbSet va trebui sa primeasca tipul modelului in care a fost declarata (adica Student in cazul curent) si numele pluralizat al modelului.

Constructorul (StudentDbContext) apeleaza constructorul clasei de baza si trimite ca parametru numele unui connection string (**base("DBConnectionString")**). Acesta este responsabil pentru conectarea la baza de date corecta.

Pentru adaugarea bazei de date, facem **click dreapta pe folderul App\_Data > Add > New Item** si selectam **Sql Server Database**. Introducem numele bazei de date si apasam Add.

Adaugarea unui Connection String se realizeaza in fisierul **Web.config**. Deoarece am folosit in constructor numele DBConnectionString, in fisierul Web.config trebuie sa adaugam un connection string cu acest nume. Exemplu:

```
<connectionStrings>
  <add name="DBConnectionString" providerName="System.Data.SqlClient"
    connectionString="Data
    Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='C:\...\App_Data\Student
    Db.mdf';Integrated Security=True"/>
</connectionStrings>
```

Connection string-ul se preia dupa deschiderea bazei de date in Server Explorer. In Server Explorer se face:

> **click dreapta pe baza de date > Properties:**



## Folosirea Entity Framework. C.R.U.D.

Pentru a folosi modelul impreuna cu Entity Framework, in Controller-ul in care avem nevoie de acesta trebuie sa declaram o variabila de tipul contextului modelului:

```
public class StudentsController : Controller
{
    private StudentDbContext db = new StudentDbContext();
    . . .
```

Variabila este privata (deci accesibila doar la nivelul Controller-ului curent) si poate fi folosita in metodele acestuia. Db este un obiect de tipul DbContext, dar toate operatiile se fac prin variabila DbSet<NumeModel> care are urmatoarea structura interna:

```
IInternalSetAdapter where TEntity : class
{
    ...protected DbSet();

    ...public virtual ObservableCollection<TEntity> Local { get; }

    ...public virtual TEntity Add(TEntity entity);
    ...public virtual IEnumerable<TEntity> AddRange(IEnumerable<TEntity>
        entities);
    ...public virtual TEntity Attach(TEntity entity);
    ...public virtual TEntity Create();
    ...public virtual TDerivedEntity Create<TDerivedEntity>() where
        TDerivedEntity : class, TEntity;
    ...public override bool Equals(object obj);
    ...public virtual TEntity Find(params object[] keyValues);
    ...public virtual Task<TEntity> FindAsync(CancellationToken
        cancellationToken, params object[] keyValues);
    ...public virtual Task<TEntity> FindAsync(params object[] keyValues);
    ...public override int GetHashCode();
    ...public Type GetType();
    ...public virtual TEntity Remove(TEntity entity);
    ...public virtual IEnumerable<TEntity> RemoveRange(IEnumerable<TEntity>
        entities);
    ...public virtual DbSqlQuery<TEntity> SqlQuery(string sql, params object[]
        parameters);
```

In urmatoarea parte a cursului vom modifica Controller-ul Students pentru a face operatiile de tip C.R.U.D. prin intermediul Entity Framework. Controller-ul se va modifica astfel:

```
public class StudentsController : Controller
```

```
{  
    private StudentDbContext db = new StudentDbContext();
```

```
    public ActionResult Index()  
    {
```

```
        var students = from student in db.Students  
                        orderby student.Name  
                        select student;  
        ViewBag.Students = students;  
        return View();  
    }
```

Preluam toti studentii din baza de date, ordonati dupa nume prin intermediul db.Students

```
    public ActionResult Show(int id)  
    {
```

```
        Student student = db.Students.Find(id);  
        ViewBag.Student = student;  
        return View();  
    }
```

Metoda Find() primeste ca parametru o valoare pentru coloana care este cheia primara

```
    public ActionResult New()  
    {
```

```
        return View();  
    }
```

```
    [HttpPost]
```

```
    public ActionResult New(Student student)  
    {
```

```
        try  
        {  
            db.Students.Add(student);  
            db.SaveChanges();  
            return RedirectToAction("Index");  
        } catch (Exception e)
```

```
        {  
            return View();  
        }  
    }
```

Students.Add primeste ca parametru un obiect de tip Student iar SaveChanges va face commit in baza de date

```

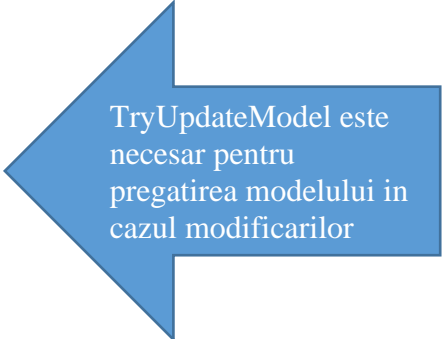
public ActionResult Edit(int id)
{
    Student student = db.Students.Find(id);
    ViewBag.Student = student;
    return View();
}

```

```

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

```

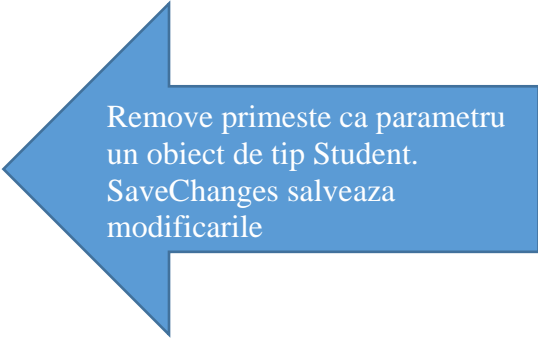


TryUpdateModel este necesar pentru pregatirea modelului in cazul modificarilor

```

[HttpDelete]
public ActionResult Delete(int id)
{
    Student student = db.Students.Find(id);
    db.Students.Remove(student);
    db.SaveChanges();
    return RedirectToAction("Index");
}
}

```



Remove primeste ca parametru un obiect de tip Student. SaveChanges salveaza modificarile

Pentru adaugarea constrangerilor si a cheii primare in cadrul unui model, acestea pot fi declarate inline in definitia sa, dupa cum urmeaza:

```
public class Student
{
    [Key]
    public int StudentId { get; set; }
    public string Name { get; set; }
    [Required]
    public string Email { get; set; }
    [MinLength(13)][MaxLength(13)]
    public string CNP { get; set; }
}

public class StudentDBContext : DbContext
{
    public StudentDBContext() : base("DBConnectionString") { }
    public DbSet<Student> Students { get; set; }
}
```

- [Key] reprezinta constrangerea de cheie primara
- [Required] face campul obligatoriu (acesta nu accepta valori null)
- [MinLength] si [MaxLength] adauga o constrangere pentru lungimea valorii

## View-ul pentru metoda Index (afisarea tuturor studentilor)

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

<h2>@ViewBag.Title</h2>

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

<a href="/Students/New">Adauga alt student</a>
<br />
```

### Lista studenti

Student 1

user@test.com

1121123123123

[Afisare student](#)

---

Student 2

user@test.ro

1121123123123

[Afisare student](#)

---

[Adauga alt student](#)

## View-ul pentru metoda Show (afisarea datelor unui student)

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

<h1>@ViewBag.Student.Name</h1>
<p>@ViewBag.Student.Email</p>
<p>@ViewBag.Student.CNP</p>
<br />

<a href="/Students/Edit/@ViewBag.Student.StudentId">Modifica
student</a>

<form method="post"
action="/Students/Delete/@ViewBag.Student.StudentId">

    @Html.HttpMethodOverride(HttpVerbs.Delete)

    <button type="submit">Sterge studentul</button>

</form>

<hr />
<a href="/Students/Index">Inapoi la lista studentilor</a>
<br />
<a href="/Students/New">Adauga alt student</a>
<br />
```

## Student 1

user@test.com

1121123123123

[Modifica student](#)

---

[Inapoi la lista studentilor](#)

[Adauga alt student](#)

---

## View-ul pentru metoda New (adaugarea unui student)

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

<p>Afisare formular de adaugare student</p>

<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>
```

### Creare student

Afisare formular de adaugare student

Nume

Adresa e-mail

CNP

Adauga student

## View-ul pentru metoda Edit (modificarea unui student)

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

<h2>Edit</h2>

<p>Afisare formular de editare student - cu datele vechi ale
studentului</p>

<form method="post" action="/Students/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>CNP</label>
    <br />
    <input type="text" name="CNP" value="@ViewBag.Student.CNP" />
    <br />
    <button type="submit">Modifica student</button>

</form>
```

### Edit

Afisare formular de editare student - cu datele vechi ale studentului

**Nume**

**Adresa e-mail**

**CNP**



## Modificarea modelelor

În momentul în care structura unui model se modifică (adică se șterg sau se adaugă proprietăți) rularea proiectului nu mai este posibilă.

```
public class Student
{
    [Key]
    public int StudentId { get; set; }
    public string Name { get; set; }
    [Required]
    public string Email { get; set; }
    [MinLength(13)][MaxLength(13)]
    public string CNP { get; set; }
    public string Address { get; set; }
}
```

s-a adăugat o nouă proprietate

```
public class StudentDbContext : DbContext
{
    public StudentDbContext() : base("DBConnectionString") { }
    public DbSet<Student> Students { get; set; }
}
```

Framework-ul ne atenționează asupra modificărilor efectuate:

```
public ActionResult Index()
{
    var students = from student in db.Students
        orderby student.Name
        select student;
    ViewBag.Students = students;
    return View();
}
```

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

Exception User-Unhandled

**System.InvalidOperationException:** 'The model backing the 'StudentDbContext' context has changed since the database was created. Consider using Code First Migrations to update the database (<http://go.microsoft.com/fwlink/?LinkId=238269>).'

[View Details](#) | [Copy Details](#)

▶ [Exception Settings](#)

Este necesara adaugarea unei secvente de cod in **Global.asax** pentru reinitializarea bazei de date si a modelelor in cazul modificarii acestora.

```
Database.SetInitializer<StudentDBContext>(new  
DropCreateDatabaseIfModelChanges<StudentDBContext>());
```


Acest initializer primeste ca parametru contextul care necesita reinitializare si seteaza functionalitatea de “drop” si “create” a tabelelor in cazul in care modelele aferente se modifica.

## Relatiile dintre modele (Join)

Un student poate avea mai multe note, iar o nota apartine unui singur student. Pentru a efectua acest lucru trebuie sa adaugam modelul care reprezinta notele si sa definim relatia dintre acestea.

Se adauga modelul Mark astfel:

```
namespace Curs4.Models  
{  
    public class Mark  
    {  
        [Key]  
        public int MarkId { get; set; }  
        [Required]  
        public int Value { get; set; }  
        public int StudentId { get; set; }  
  
        public virtual Student Student; }  
}
```



Acest model are o proprietate publica numita “Student” de tipul Student (modelului Student). Reprezinta relatia dintre modelul Mark si un Student, adica, o nota este asociata unui singur student.

Modelul Student se va modifica astfel:

```
namespace Curs .Models
{
    public class Student
    {
        [Key]
        public int StudentId { get; set; }
        public string Name { get; set; }
        [Required]
        public string Email { get; set; }
        [MinLength(13)][MaxLength(13)]
        public string CNP { get; set; }
        public string Address { get; set; }

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

    public class StudentDbContext : DbContext
    {
        public StudentDbContext() : base("DBConnectionString") { }
        public DbSet<Student> Students { get; set; }
        public DbSet<Mark> Marks { get; set; }
    }
}
```

Pentru relatia one – to – many (un Student are mai multe note) se adauga metoda virtuala care returneaza o colectie de note (Marks) conform definitiei de mai sus.

De asemenea, pentru noul model Mark va trebui sa adaugam in contextul bazei de date DbSet-ul asociat acestuia.

Preluarea datelor asociate (din relatie) unui student se face apeland colectia Marks dintr-un obiect de tipul Student:

```
public ActionResult Show(int id)
{
    Student student = db.Students.Find(id);
    ViewBag.Student = student;
    var note = from nota in student.Marks
               select nota;
    ViewBag.Marks = note;

    return View();
}
```

Adaugarea unei note pentru un student, se face astfel:

```
// Adaugare nota student
public ActionResult AddMark(int id)
{
    ViewBag.StudentId = id;
    return View();
}
```

Preluam StudentId din URL pentru a putea adauga unui anumit student nota

```
[HttpPost]
public ActionResult AddMark(Mark mark)
{
    try
    {
        db.Marks.Add(mark);
        db.SaveChanges();
        return Redirect("/Students/Show/" + mark.StudentId);
    } catch (Exception e){
        ViewBag.StudentId = mark.StudentId;
        return View();
    }
}
```

Folosim db.Marks (DbSet-ul pentru modelul Mark)

View-ul asociat acestei metode:

```
<form method="post" action="/Students/AddMark">
    <input type="hidden" name="StudentId" value="@ViewBag.StudentId" />
    <br />
    <label>Nota:</label>
    <br />
    <input type="text" name="Value" />
    <button type="submit">Adauga nota</button>
</form>
```