

Laboratorul 3

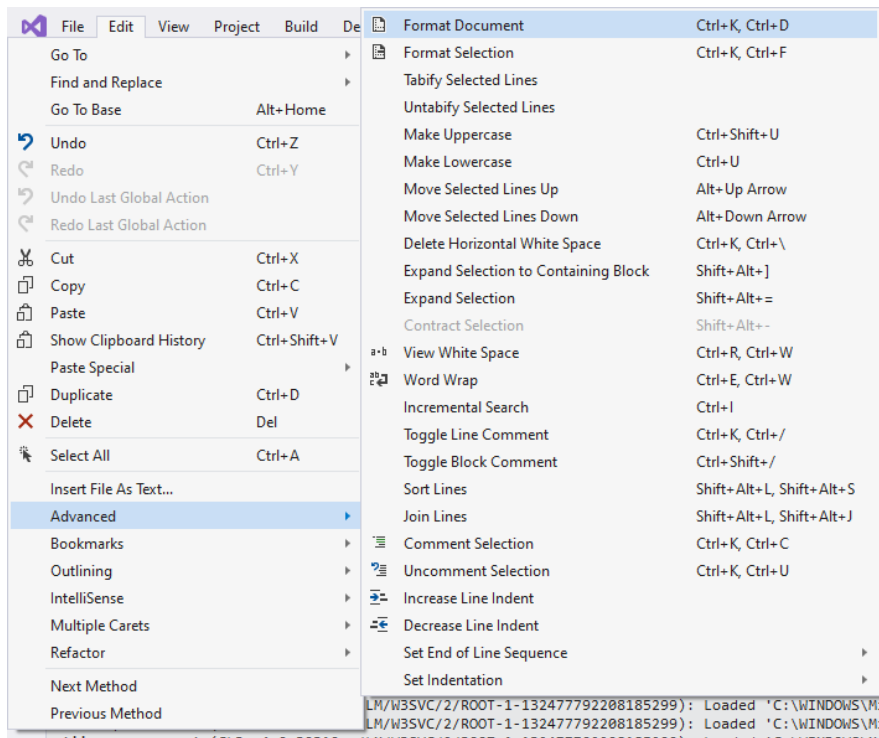
Mici detalii

Cum ar trebui sa **comentati** codul:

- Comentare cod **html**: `<!-- comentariu -->`
- Comentare cod **razor**: `@* comentariu *@`
- Comentare cod **C#**:
 - `// comentariu`
 - `/* comentariu */`

Formatarea codului se poate realiza in mai multe moduri:

- Ctrl + K, Ctrl + D
- Ctrl + K, Ctrl + F

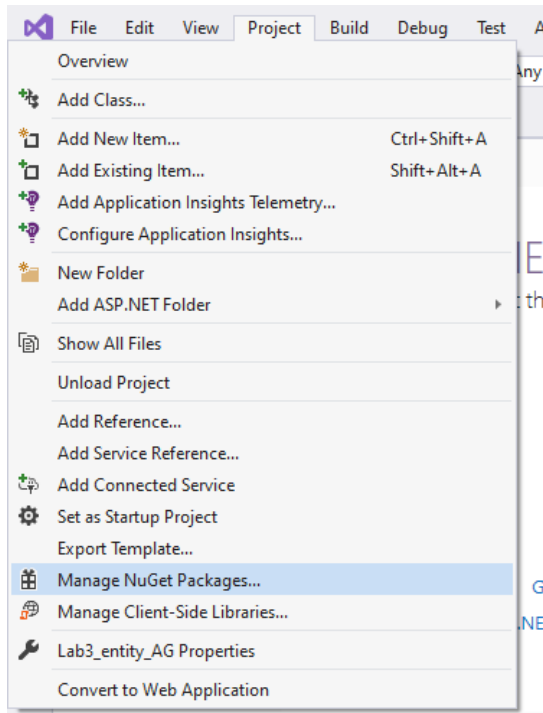


Exercitiu 1

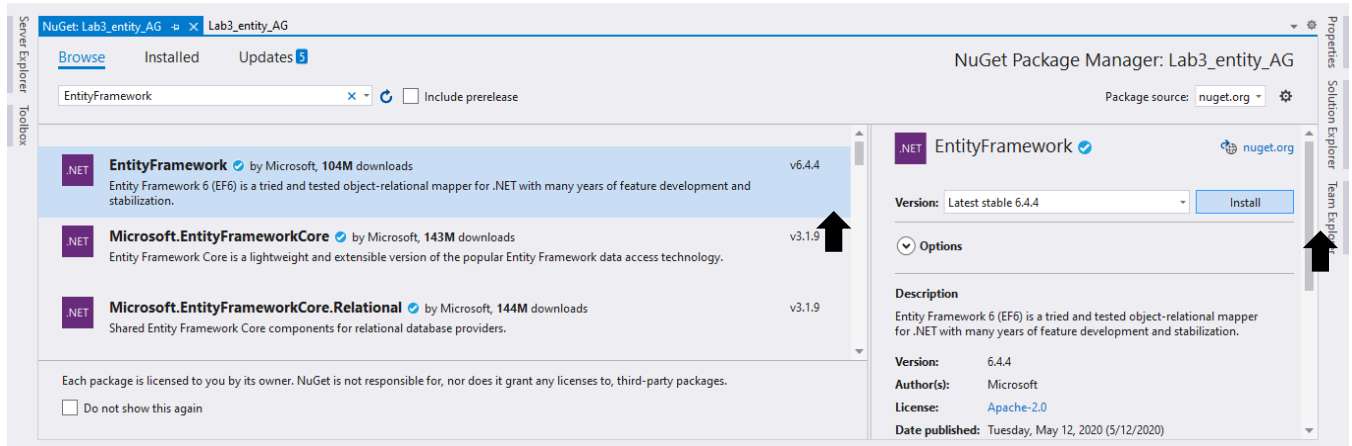
Implementati exemplul din curs. Observati comportamentul diferitelor strategii de initializare. (Creati in initializatorul personalizat o linie de tabel cu data si timpul curente pentru a putea determina momentul recrearii bazei de date). Examinati structura bazei de date in Server Explorer.

EntityFramework

Mai intai trebuie sa adaugam EntityFramework proiectului nostru. Mergeti la **Project -> Manage NuGet Packages**

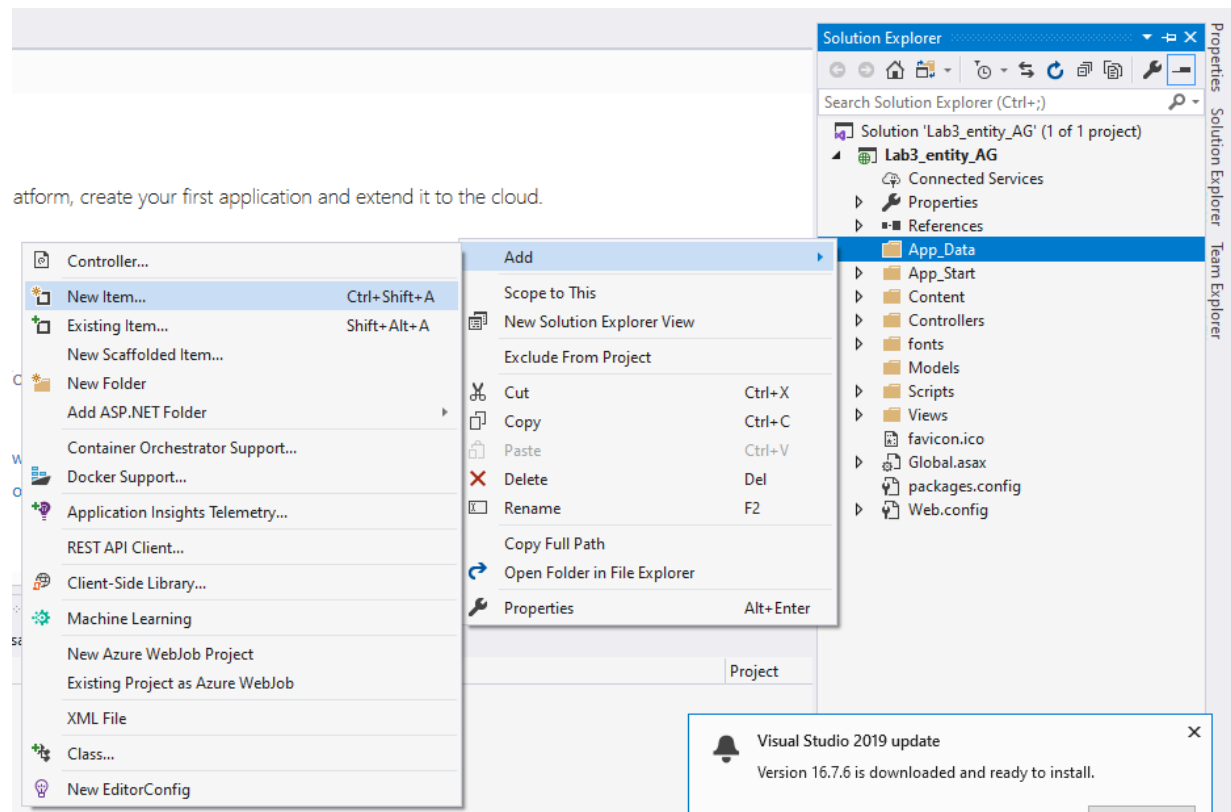


In Browse scrieti "Entity Framework" si apasati pe Install.

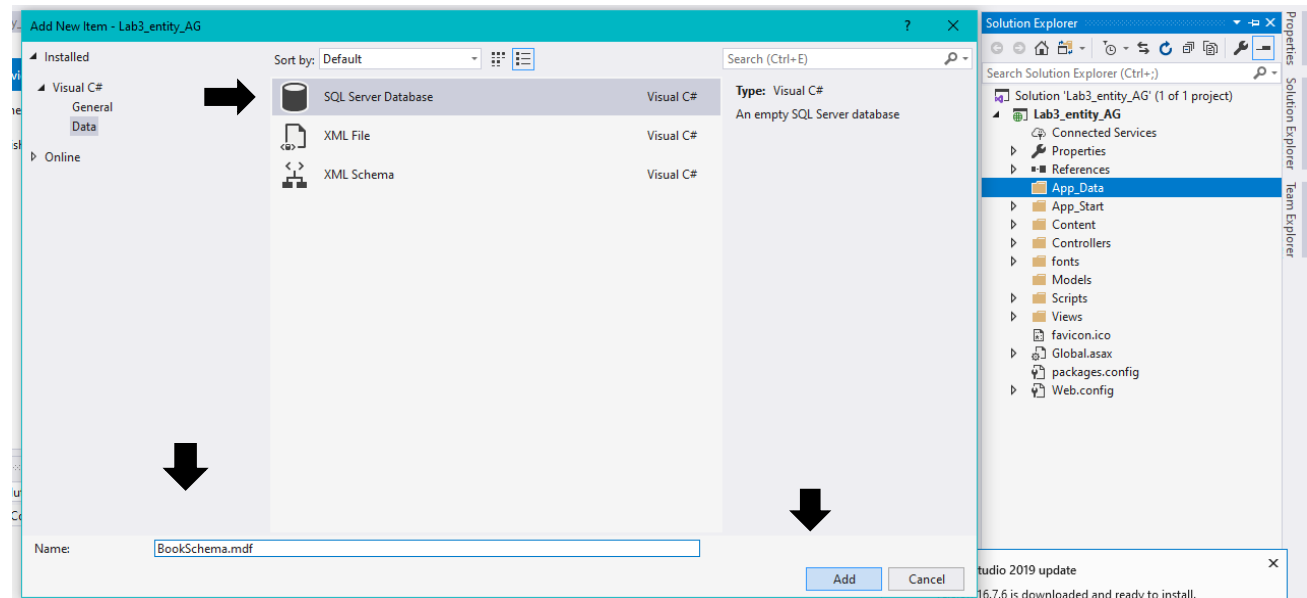


Crearea Bazei de Date

Click dreapta pe directorul App_Data -> Add -> New Item.



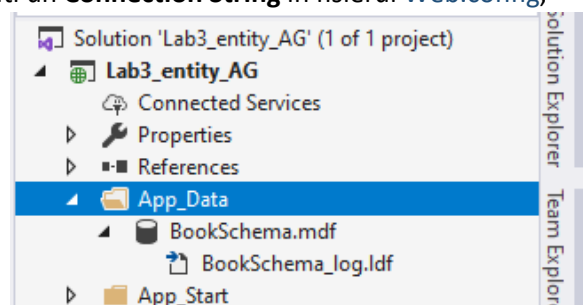
Selectati SQL Server Database, introduceti un nume bazei de date si apoi apasati pe Add.



Observati ca se va crea un fisier cu extensia .mdf.

Pentru a accesa baza de date nou-creata trebuie definiti un **Connection String** in fisierul **Web.config**, inainte de `</configuration>` astfel:

```
<connectionStrings>
```



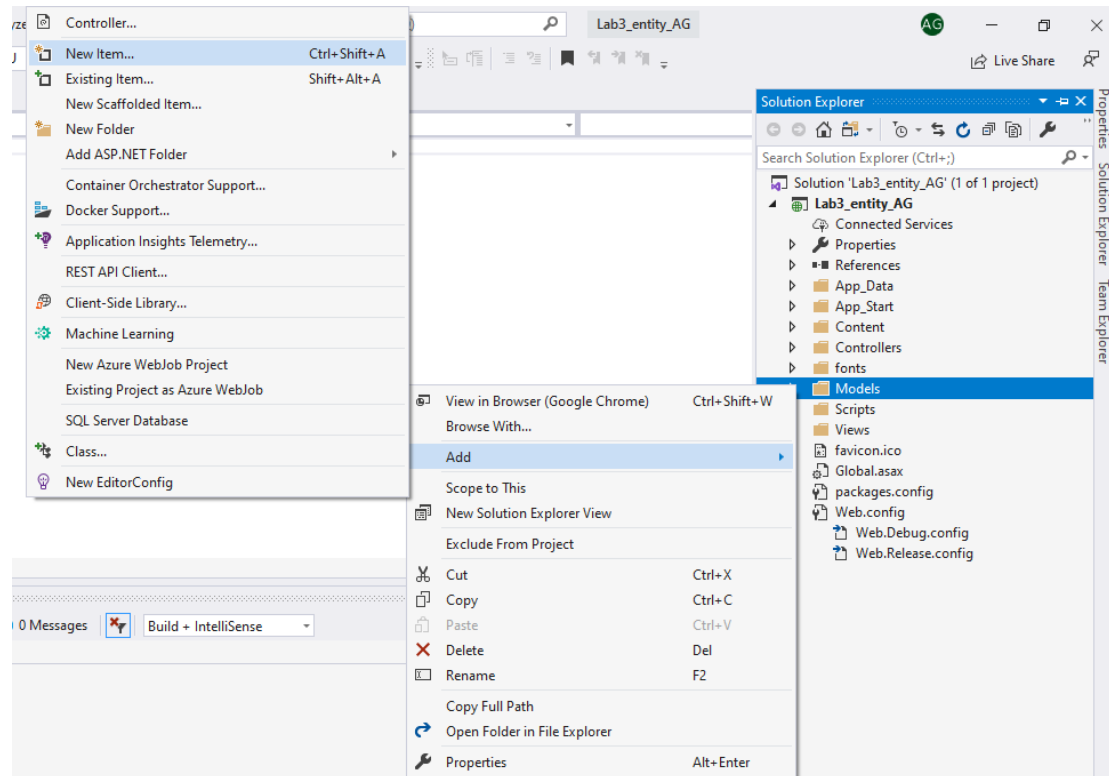
```
<add name="DbConnectionString" providerName="System.Data.SqlClient" connectionString="Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='C:\...\App_Data\BookSchema.mdf';Integrated
Security=True"/>
</connectionStrings>
```



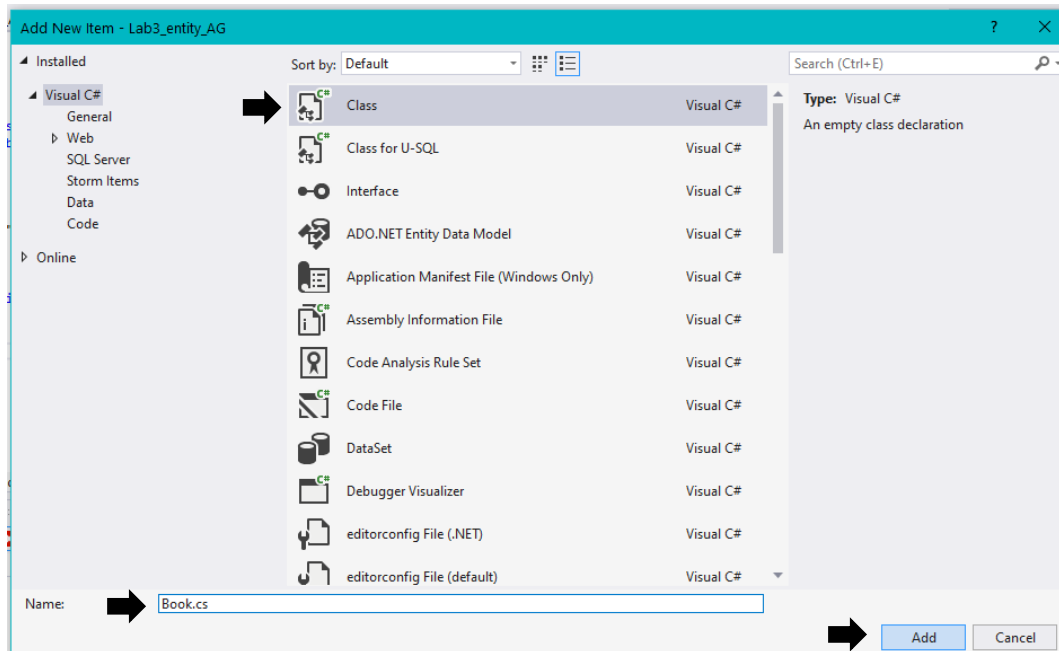
Puteti prelua connectionString-ul vostru din **Server Explorer** -> **click dreapta pe baza de date** -> **Properties** -> **copiati ce scrie la ConnectionString**.

Crearea Modelelor

Acum vom scrie clasa-model ce reprezinta tipurile de date corespunzatoare tabelor. In directorul **Models** -> **Add** -> **New Item**



Selectati **Class** si **introduceti un nume sugestiv** pentru clasa voastra (ex: Book).



Introduceti urmatoarea secventa de cod:

```
public class Book
{
    public int BookId { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
}
```

Ce sunt proprietatile?



Observam ca am declarat campurile clasei Book cu modificatorul de acces **public**.

De ce am pus **public** si nu **private** din moment ce avem declarate metodele specifice **getter** si **setter** ?

Raspuns: **Title**, **Author** si **BookId** NU sunt variabile publice, ci sunt proprietati intr-o forma public accesibila. In Laboratorul 2 am mentionat ca exista anumite conventii de scrierea codului si anume:

- PascalCase (ex: BookId) pentru nume de **metode** si **clase, proprietati**
- camelCase (ex: bookId) pentru nume de **variabile**


Observati ca aici **BookId**, **Title**, **Author** sunt scrise in PascalCase => Sunt niste proprietati auto-implementate, NU sunt variabile.


Daca aveam cazul:

```
public class Book
{
    public int BookId { get; set; }

    private string title;
    public string GetTitle()
    {
        return title;
    }
    public string Title
    {
```

```
        get { return title; }  
        set { title = value; }  
    }  
    public string Author { get; set; }  
}
```

 Aici, **title** este o variabila **private** (a se remarca scrierea in *camelCase*). Ca membru privat, acesta nu va putea fi accesat decat de catre metodele de membru. Metodele publice **GetTitle()** si **Title** au fost adaugate pentru a putea accesa membrul privat **title**. Membrul title este accesat printr-o metoda publica cu ajutorul lui **GetTitle()** si este accesat de o proprietate publica read-write numita **Title**.

 **Proprietatile** din C# sunt niste membri ai claselor care ofera un mecanism flexibil ce au scopul de a expune campurile private. Aceste proprietati sunt, de fapt, metode speciale numite *accesori*. O proprietate C# are 2 accesori:

- *get* = accesori de afisare a unei proprietati. Acesta **returneaza** valoarea proprietatii.
- *set* = accesori ce seteaza o proprietate. Acesta **atribuie** proprietatii o noua valoare.

 Cuvantul cheie **value** reprezinta valoarea unei proprietati.

In concluzie, secventa de cod

```
private string title;  
public string Title  
{  
    get { return title; }  
    set { title = value; }  
}
```

este **echivalenta** cu *proprietatile auto-implementate*, si anume secventa urmatoare:

```
public string Title { get; set; }
```

Atribute ale bazei de date

Exista mai multe tipuri de atribute pentru baza de date:

- Atribute care se pun deasupra campurilor unor clase, care se aplica unei proprietati (**COLOANE**)
 - `[Column("book_id")]` in mod implicit proprietatile entitatii sunt mapate la coloane din tabel cu acelasi nume ca proprietatea. Putem schimba numele coloanei din tabel cu ajutorul acestui atribut.
 - `[Column(TableName = "decimal(5,2)")]` SQL Server mapeaza in mod implicit tipul *DateTime* la coloane de tipul *datetime2(7)*, iar tipul *string* la coloane de tipul *NVARCHAR(max)*. Dezvoltatorul poate specifica tipul exact de date pe care il doreste pentru coloana respectiva.
 - `[Key]` marcheaza coloana pe care trebuie sa o mapeze proprietatea ca fiind o cheie primara (PK).
 - `[NotMapped]` in mod implicit, toate proprietatile publice cu getteri si setteri vor fi incluse tabel. Pentru a specifica ca un camp nu trebuie sa fie inclus in tabel ne vom folosi de acest atribut.
 - `[MinLength(10)]` specifica numarul minim de caractere sau octeti pentru coloana pe care trebuie sa o mapeze proprietatea.

- `[MaxLength(200)]` este echivalentul lui `NVARCHAR(200)` si specifica maximum de caractere sau octeti pentru coloana pe care trebuie sa o mapeze.
- `[Required]` aceasta conventie obliga proprietatea avizata sa NU fie null (adica marcheaza o coloana obligatorie). **!!!! Este obligatoriu sa o folositi pentru relatia one-to-one, altfel veti avea eroare la compilare !!!!**
- Atribute care se pun deasupra numelui clasei (**TABELE**)
 - `[Table("NumeTabel")]` permite modificarea numelui tabelului.

Se pot *aplica mai multe constrangeri* pe o singura proprietate:

```
[MinLength(10), MaxLength(200)]  
public string Title { get; set; }
```

SAU:

```
[Key]  
[Column("book_id")]  
public int BookId { get; set; }
```

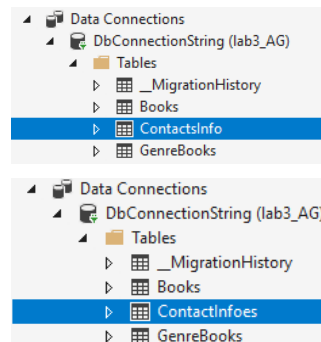
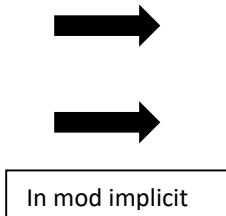
Se poate trimite si un *mesaj de validare*:

```
[MinLength(10, ErrorMessage = "Title cannot be less than 10")]  
public string Title { get; set; }
```

Exemplu pentru modificarea numelui tabelului:

```
[Table("ContactsInfo")]  
public class ContactInfo
```

```
public class ContactInfo
```



Continuarea rezolvarii laboratorului:


Pentru a descrie structura bazei de date, vom crea o noua clasa (in acelasi fisier Book.cs), numita clasa context, ce va mosteni `DbContext`. `DbContext` se afla in **System.Data.Entity**, asadar vom adauga o directiva corespunzatoare prin cuvantul cheie **using**.


```
namespace Lab3_entity_AG.Models  
{  
    public class Book  
    {  
        public int BookId { get; set; }  
        public string Title { get; set; }  
        public string Author { get; set; }  
    }  
  
    public class DbCtx : DbContext  
    {  
        public DbCtx() : base("DbConnectionString")  
        {  
            // set the initializer here  
            Database.SetInitializer<DbCtx>(new Initp());  
            //Database.SetInitializer<DbCtx>(new CreateDatabaseIfNotExists<DbCtx>());  
            //Database.SetInitializer<DbCtx>(new DropCreateDatabaseIfModelChanges<DbCtx>());  
            //Database.SetInitializer<DbCtx>(new DropCreateDatabaseAlways<DbCtx>());  
        }  
    }  
}
```


```
        public DbSet<Book> Books { get; set; }  
    }  
}
```

Constructorul clasei DbCtx apeleaza **constructorul clasei de baza**, si anume DbContext, cu un singur parametru. Acest parametru este un **string** ce **reprezinta numele connectionString-ului** si are rolul de a conecta clasa la baza de date corecta !!!

Clasa DbCtx contine si proprietatea **DbSet<T>** ce va reprezenta tabelul bazei de date ce se va numi "Books". T reprezinta tipul de obiect stocat in tabelul respectiv.

 Deci, in baza de date se va crea o tabelul Books corespunzator clasei Book, iar coloanele tabelului vor fi reprezentate de campurile clasei Book.

 Nu putem avea mai mult de un tabel de un anumit tip de date.

 Daca nu adaugati DropCreateDatabaseIfModelChanges si modificati o clasa-model, programul ne va da eroare deoarece trebuie modificata si structura bazei de date.

Exista mai multe strategii de initializare a bazei de date:

- CreateDatabaseIfNotExists
- DropCreateDatabaseIfModelChanges
- DropCreateDatabaseAlways
- Initializare personalizata

Vom incepe prin a crea o initializare personalizata:

```
public class Initp : DropCreateDatabaseAlways<DbCtx>  
{  
    // custom initializer  
    protected override void Seed (DbCtx ctx)  
    {  
        ctx.Books.Add(new Book { Title = "The Atomic Times", Author = "Michael Harris"});  
        ctx.Books.Add(new Book { Title = "In Defense of Elitism", Author = "Joel Stein"});  
        ctx.Books.Add(new Book { Title = "Data curenta", Author = DateTime.Now.ToString() });  
        ctx.SaveChanges();  
        base.Seed(ctx);  
    }  
}
```

Pentru crearea unei initializari a bazei de date personalizate trebuie mostenit un tip de initializator deja existent, in cazul de fata am ales **DropCreateDatabaseAlways**. Initializatorul bazei de date poate fi setat si din fisierul de configurare **app.config**. Pentru mai multe detalii consultati documentatia:

<https://www.entityframeworktutorial.net/code-first/database-initialization-strategy-in-code-first.aspx>

Exercitiul 2

Creati o actiune noua cu un parametru id unde view-ul afiseaza detaliile cartii care are acel numar drept BookId. In view-ul corespunzator actiunii ce enumera toate cartile, creati cu Razor pentru fiecare carte un link catre pagina cu detalii.

Crearea Controller-ului

Creati controller-ul BookController.cs si introduceti urmatoarea secventa de cod

```
namespace Lab3_entity_AG.Controllers  
{
```



```
public class BookController : Controller
{
    private DbCtx db = new DbCtx();

    // GET: Book
    public ActionResult Index()
    {
        List<Book> books = db.Books.ToList();
        ViewBag.Books = books;

        return View();
    }
}
```

Crearea view-ului

In Views -> Book -> Index.cshtml

```
@{
    ViewBag.Title = "Books";
}
<h2>@ViewBag.Title</h2>

@foreach (var book in ViewBag.Books)
{
    <div class="panel-body">
        @Html.Label("Title", "Title:")
        <br />
        <p>@book.Title</p>

        @Html.Label("Author", "Author:")
        <br />
        <p>@book.Author</p>
    </div>
}
```

Exercitiul 3

In clasa Book, creati un nou camp Summary ce va reprezenta rezumatul unei carti. Pe pagina cu detalii despre o carte, afisati, pe langa informatiile existente:

- Summary: urmat de continutul campului, in cazul in care este nevid
- un mesaj de informare, in cazul in care este vid

Modificare Book.cs

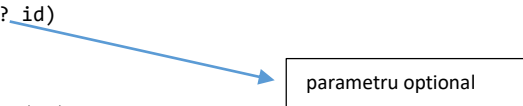
Adaugati in clasa Book urmatorul camp

```
public string Summary { get; set; }
```

Modificare BookController.cs

Adaugati in controller actiunea **Details**

```
public ActionResult Details(int? id)
{
    if (id.HasValue)
    {
        Book book = db.Books.Find(id);
        if (book != null)
        {
            return View(book);
        }
        return HttpNotFound("Couldn't find the book with id " + id.ToString() + "!");
    }
}
```



```
}  
    return HttpNotFound("Missing book id parameter!");  
}
```

Creare Details.cshtml

In **Views -> Book** creati view-ul **Details.cshtml**.

```
@model Lab3_entity_AG.Models.Book
```

```
@{  
    ViewBag.Title = "Details";  
    var emptySummaryMsgVar = "This book has no summary";  
    string emptySummaryMsg = "This book has no summary";  
}
```

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

```
@Html.Label("Author", "Author:")  
<br />
```

```
<p>@Model.Author</p>
```

```
@Html.Label("Summary", "Summary:")  
<br />
```

```
<div class="panel-body">
```

```
    @if (Model.Summary.IsEmpty())  
    {
```

```
        <!--Modalitati de afisare a unui mesaj-->
```

```
        <p>@emptySummaryMsg</p>
```

```
        <p>@emptySummaryMsgVar</p>
```

```
        <p>This book has no summary</p>
```

```
    }
```

```
    else
```

```
    {
```

```
        @*@Html.Display("Summary")*@
```

```
        @Html.DisplayFor(b => b.Summary)
```

```
    }
```

```
</div>
```

Aici trebuie sa puneti numele proiectului vostru

Bloc de cod in cadrul caruia se pot defini si initializa variabile, structuri if, while, etc.

Cod razor pentru tag-ul html <label> care specifica si textul label-ului, ex "Author:". Este echivalentul lui <label for="Author">Author:</label> in HTML.

Afiseaza valoarea campului Author, al modelului Book.

Pentru a putea **prelua** valoarea unei variabile se foloseste @.

!!! ATENTIE !!! @Html.Display primeste ca parametru un string, dar NU il afiseaza pe ecran. El primeste ca parametru numele unei proprietati/camp din model si afiseaza pe ecran valoarea sa. @Html.DisplayFor afiseaza acelasi lucru, dar foloseste lambda expresii

Modificare Index.cshtml

Adaugati in interiorul tag-ului <div></div> urmatoarea secventa:

```
@using (Html.BeginForm(actionName: "Details", controllerName: "Book", method: RequestMethod.Get,  
routeValues: new { id = book.BookId }))  
{  
    <button class="btn btn-primary">Details</button>  
}
```

Scrie un tag <form> </form>

La apasarea butonului Details, utilizatorul este redirectionat catre ruta **Book/Details/id**

Mai multe detalii despre HTML Helpers gasiti aici <https://www.tutorialsteacher.com/mvc/html-helpers>.

Exercitiul 4

Creati o clasa noua numita Publisher. Implementati, pe rand, relatii one-to-one, many-to-many, one-to-many intre ea si Book. Examinati structura bazei de date in Server Explorer.

Exemplu One to Many: <https://www.entityframeworktutorial.net/code-first/configure-one-to-many-relationship-in-code-first.aspx>

Exemplu Many to Many: <https://www.entityframeworktutorial.net/code-first/configure-many-to-many-relationship-in-code-first.aspx>

Exemplu One to One: <https://www.entityframeworktutorial.net/code-first/configure-one-to-one-relationship-in-code-first.aspx>

Pentru restul codului mergeti la sectiunea "Cod".

Cod final

Models

Book.cs

```
using Microsoft.SqlServer.Server;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace lab3_AG.Models
{
    public class Book
    {
        //[Key]
        //[Column("Book_id")]
        [Key, Column("Book_id")]
        public int BookId { get; set; }

        //[MinLength(10), MaxLength(200)]
        //[MinLength(10, ErrorMessage = "Title cannot be less than 10")]
        [MinLength(10, ErrorMessage = "Title cannot be less than 10"),
        MaxLength(200, ErrorMessage = "Title cannot be more than 200")]
        public string Title { get; set; }
        public string Author { get; set; }

        [NotMapped]
        public DateTime LoadedFromDatabase { get; set; }

        public string Summary { get; set; }

        /*private string summary;

        public string Summary
        {
            get { return summary; }
            set { summary = value; }
        }*/


        // one to many
        [Column("Publisher_id")]
        public int PublisherId { get; set; }
        public virtual Publisher Publisher { get; set; }

        // many to many
        public virtual ICollection<Genre> Genres { get; set; }
    }
    public class DbCtx : DbContext
    {
        public DbCtx() : base("DbConnectionString")
        {
            Database.SetInitializer<DbCtx>(new Initp());
            //Database.SetInitializer<DbCtx>(new CreateDatabaseIfNotExists<DbCtx>());
            //Database.SetInitializer<DbCtx>(new DropCreateDatabaseIfModelChanges<DbCtx>());
            //Database.SetInitializer<DbCtx>(new DropCreateDatabaseAlways<DbCtx>());
        }
        public DbSet<Book> Books { get; set; }
    }
}
```

```
public DbSet<Publisher> Publishers { get; set; }
public DbSet<Genre> Genres { get; set; }
public DbSet<ContactInfo> ContactsInfo { get; set; }
}

public class Initp : DropCreateDatabaseAlways<DbContext>
{
    protected override void Seed(DbContext ctx)
    {
        ctx.Books.Add(new Book
        {
            Title = "The Atomic Times",
            Author = "Michael Harris",
            Publisher = new Publisher { Name = "HarperCollins", ContactInfo = new ContactInfo {
PhoneNumber = "07123456789" } },
            Genres = new List<Genre> {
                new Genre { Name = "Horror"}
            }
        });
        ctx.Books.Add(new Book
        {
            Title = "In Defense of Elitism",
            Author = "Joel Stein",
            Publisher = new Publisher { Name = "Macmillan Publishers", ContactInfo = new ContactInfo {
PhoneNumber = "07123458789" } },
            Genres = new List<Genre> {
                new Genre { Name = "Humor"}
            }
        });
        ctx.Books.Add(new Book
        {
            Title = "The Canterbury Tales",
            Author = "Geoffrey Chaucer",
            Summary = "At the Tabard Inn, a tavern in Southwark, near London, the narrator joins a
company of twenty-nine pilgrims. The pilgrims, like the narrator, are traveling to the shrine of the
martyr Saint Thomas Becket in Canterbury. The narrator gives a descriptive account of twenty-seven of
these pilgrims, including a Knight, Squire, Yeoman, Prioress, Monk, Friar, Merchant, Clerk, Man of Law,
Franklin, Haberdasher, Carpenter, Weaver, Dyer, Tapestry-Weaver, Cook, Shipman, Physician, Wife, Parson,
Plowman, Miller, Manciple, Reeve, Summoner, Pardoner, and Host. (He does not describe the Second Nun or
the Nun's Priest, although both characters appear later in the book.) The Host, whose name, we find out in
the Prologue to the Cook's Tale, is Harry Bailey, suggests that the group ride together and entertain one
another with stories. He decides that each pilgrim will tell two stories on the way to Canterbury and two
on the way back. Whomever he judges to be the best storyteller will receive a meal at Bailey's tavern,
courtesy of the other pilgrims. The pilgrims draw lots and determine that the Knight will tell the first
tale.",
            Publisher = new Publisher { Name = "Scholastic", ContactInfo = new ContactInfo {
PhoneNumber = "07113456789" } },
            Genres = new List<Genre> {
                new Genre { Name = "Satire"},
                new Genre { Name = "Fabilau"},
                new Genre { Name = "Allegory"},
                new Genre { Name = "Burlesque"}
            }
        });
        ctx.Books.Add(new Book
        {
            Title = "Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to
Programming",
            Author = "Eric Matthers",
            Publisher = new Publisher { Name = "Schol", ContactInfo = new ContactInfo { PhoneNumber =
"07126656789" } },
            Genres = new List<Genre> {
                new Genre { Name = "Programming"}
            }
        });

        //ctx.Books.Add(new Book { Title = "Data curenta", Author = DateTime.Now.ToString() });
        ctx.SaveChanges();
    }
}
```



```
        base.Seed(ctx);
    }
}
```

Lasati aceasta linie comentata!!! Daca o decommentati veti avea o eroare de BD deoarece entitatea Book este conectata la entitatile Genre, Publisher, ContactInfo si este obligatoriu ca pentru fiecare carte sa inseram in tabel si informatiile legate de Genre, Publisher si ContactInfo.

Publisher.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Web;

namespace lab3_entity.Models
{
    public class Publisher
    {
        [Key]
        public int PublisherId { get; set; }
        public string Name { get; set; }

        // many-to-one relationship
        public virtual ICollection<Book> Books { get; set; }
        // one-to one-relationship
        [Required]
        public virtual ContactInfo ContactInfo { get; set; }
    }
}
```

Este foarte important ca in clasa Publisher pe campul ContactInfo sau in clasa ContactInfo pe campul Publisher sa puneti atributul **[Required]** !!!

Genre.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace lab3_entity.Models
{
    public class Genre
    {
        [Key]
        public int GenreId { get; set; }
        public string Name { get; set; }

        // many-to-many relationship
        public virtual ICollection<Book> books { get; set; }
    }
}
```

ContactInfo.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace lab3_entity.Models
{
    [Table("ContactsInfo")]
    public class ContactInfo
    {
        [Key]
        public int ContactInfoId { get; set; }
        public string PhoneNumber { get; set; }
    }
}
```

```
        // one-to one-relationship
        public virtual Publisher Publisher { get; set; }
    }
}
```

Controller

BookController

```
using lab3_entity.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace lab3_entity.Controllers
{
    public class BookController : Controller
    {
        private DbCtx db = new DbCtx();

        [HttpGet]
        public ActionResult Index()
        {
            List<Book> books = db.Books.Include("Publisher").ToList();
            ViewBag.Books = books;

            return View();
        }

        [HttpGet]
        public ActionResult Details(int? id)
        {
            if (id.HasValue)
            {
                Book book = db.Books.Find(id);
                if (book != null)
                {
                    return View(book);
                }
                return HttpNotFound("Couldn't find the book with id " + id.ToString() + "!");
            }
            return HttpNotFound("Missing book id parameter!");
        }
    }
}
```

Views

Detials.cshtml

```
@model lab3_AG.Models.Book
```

```
@{
    ViewBag.Title = "Details";
    var emptySummaryMsgVar = "This book has no summary";
    string emptySummaryMsg = "This book has no summary";
}
```

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

```
@Html.Label("Title", "Title:")
```

```
<br />
```

```
<p>@Model.Title</p>
```

```
@Html.Label("Author", "Author:")
```

```
<br />
<p>@Model.Author</p>

@if (Model.Publisher != null)
{
    @Html.Label("Publisher", "Publisher:")
    <br />
    <p>@Model.Publisher.Name</p>

    @Html.Label("Publisher", "Contact Info:")
    <br />
    <p>@Model.Publisher.ContactInfo.PhoneNumber</p>
}

@Html.Label("Summary", "Summary:")
<br />
<div class="panel-body">
    @if (Model.Summary.IsEmpty())
    {
        <!--Modalitati de afisare a unui mesaj-->
        <p>@emptySummaryMsg</p>
        <p>@emptySummaryMsgVar</p>
        <p>This book has no summary</p>

        @*Html.Display nu o sa afiseze nimic in acest caz*@
        @Html.Display("emptySummaryMsgVar")
    }
    else
    {
        @*@Html.Display("Summary")*@
        @Html.DisplayFor(b => b.Summary)
    }
}
</div>

@if (Model.Genres.Count > 0)
{
    @Html.Label("Genres", "Genres:")
    <br />
    <ul>
        @foreach (var genre in Model.Genres)
        {
            <li>@genre.Name</li>
        }
    </ul>
}
}
```

Index.cshtml

```
@{
    ViewBag.Title = "Books";
}
<h2>@ViewBag.Title</h2>

@foreach (var book in ViewBag.Books)
{
    <div class="panel-body">
        @Html.Label("Title", "Title:")
        <br />
        <p>@book.Title</p>

        @Html.Label("Author", "Author:")
        <br />
        <p>@book.Author</p>

        @using (Html.BeginForm(actionName: "Details", controllerName: "Book", method: FormMethod.Get,
routeValues: new { id = book.BookId }))
        {
            <button class="btn btn-primary">Details</button>
        }
    }
}
```

Laborant DAW: Guster Andreea

```
    </div>  
}
```