

## Laboratorul 4 – Laborant Andreea Guster

Acest laborator este in continuarea laboratorului 3. Pentru a rezolva cerintele acestui laborator va trebui sa preluati codul scris in laboratorul precedent. **NU uitati ca de fiecare data cand creati un proiect nou in care vreti sa va conectati la baza de date, va trebui sa instalati EntityFramework (vezi lab 3)!!**


### CRUD (Create, Read, Update, Delete) – Exercitiul 1

#### Despre Create

Adaugati in BookController actiunea NEW de tipul GET.

```
[HttpGet] // nu este necesar, by default toate actiunile din controller sunt GET
public ActionResult New()
{
    Book book = new Book();
    book.Genres = new List<Genre>();
    return View(book);
}
```

Adaugati in BookController actiunea NEW de tipul POST.

 Aceasta actiune nu are asociat un view. Proprietatile obiectului **bookRequest** vor fi instantiate cu continutul campurilor corespunzatoare din form-ul ce a apelat actiunea cu HttpGet. Acest proces se numeste **Model Binding**.

**RedirectToAction:** In acesta secventa de cod, daca modelul este valid, utilizatorul va fi redirectionat catre pagina de afisare a tuturor cartilor, si anume, cea cu URL-ul Books/Index. In cazul in care apare o eroare la completarea formularului, utilizatorul nu va putea fi redirectionat catre alta pagina.

- **RedirectToAction("Index")** poate fi folosit asa cand vrem sa fim redirectinatti catre o actiune din cadrul aceluasi controller.
- **RedirectToAction("Index", "Home")** poate fi folosit asa cand vrem sa fim redirectionati catre o actiune din cadrul unui alt controller, in acest caz, controller-ul **Home**.

```
[HttpPost]
public ActionResult New(Book bookRequest)
{
    try
    {
        if (ModelState.IsValid)
        {
            bookRequest.Publisher = db.Publishers
                .FirstOrDefault(p => p.PublisherId.Equals(1));

            db.Books.Add(bookRequest);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        return View(bookRequest);
    }
    catch (Exception e)
    {
        return View(bookRequest);
    }
}
```

**ModelState.IsValid** indica daca s-a realizat corect fenomenul de *model binding* care leaga valorile primite de la cerere la model, si daca nu s-au incalcat in timpul procesului de *binding* **reguli de validare** specificate explicit.

**Validarile** pot fi specificate explicit prin attribute aplicate pe proprietatile unui model.

Liniile cu verde **vor fi sterse** cand vom implementa si alte functionalitati in laboratoarele viitoare. Daca le comentati o sa vedeti ca nu veti putea crea o carte noua din cauza erorii generate de BD pt relatia one-to-many dintre Book si Publisher. Eroare va fi prinsa de blocul **catch**.

Creare view-ul New.cshtml in Views/Book.

```
@model lab3_miercuri.Models.Book
@{
    ViewBag.Title = "Create";
}

<h2>@ViewBag.Title</h2>

@using (Html.BeginForm(actionName: "New", controllerName: "Book", method: FormMethod.Post))
{
    @Html.Label("Title", "Book Title:")
    <br />
    @Html.TextBoxFor(b => b.Title, null, new { placeholder = "Type in the book's title", @class = "form-control" })
    @Html.ValidationMessageFor(b => b.Title, "", new { @class = "text-danger" })
    <br />

    @Html.Label("Author", "Author:")
    <br />
    @Html.TextBoxFor(b => b.Author, null, new { placeholder = "Type in the book's author", @class = "form-control" })
    @Html.ValidationMessageFor(b => b.Author, "", new { @class = "text-danger" })
    <br />

    @Html.Label("Summary", "Summary:")
    <br />
    @Html.TextArea("Summary", null, new { placeholder = "Type in the book's summary", @class = "form-control", @rows = "10" })
    @Html.ValidationMessageFor(b => b.Summary, "", new { @class = "text-danger" })
    <br />

    <button class="btn btn-primary" type="submit">Create</button>
}

```

*Despre Update*

Adăugăm în BookController acțiunea EDIT de tipul GET.

```
[HttpGet]
public ActionResult Edit(int? id)
{
    if (id.HasValue)
    {
        Book book = db.Books.Find(id);

        if (book == null)
        {
            return HttpNotFound("Couldn't find the book with id " + id.ToString());
        }
        return View(book);
    }
    return HttpNotFound("Missing book id parameter!");
}

```

Adăugăm în BookController acțiunea EDIT de tipul PUT.

```
[HttpPut]
public ActionResult Edit(int id, Book bookRequest)
{
    try
    {
        if (ModelState.IsValid)
        {
            Book book = db.Books
                .Include("Publisher")
                .SingleOrDefault(b => b.BookId.Equals(id));

            if (TryUpdateModel(book))
            {
                db.SaveChanges();
            }
        }
    }
    catch { }
}

```

```

        {
            book.Title = bookRequest.Title;
            book.Author = bookRequest.Author;
            book.Summary = bookRequest.Summary;
            db.SaveChanges();
        }
        return RedirectToAction("Index");
    }
    return View(bookRequest);
}
catch (Exception e)
{
    return View(bookRequest);
}
}

```

Creata view-ul Edit.cshtml in Views/Book.

```

@model lab3_miercuri.Models.Book

@{
    ViewBag.Title = "Update";
}

<h2>@ViewBag.Title</h2>

@using (Html.BeginForm(actionName: "Edit", controllerName: "Book"))
{
    @Html.HttpMethodOverride(HttpVerbs.Put)

    <!--camp ascuns pentru a retine id-ul obiectului-->
    @Html.HiddenFor(b => b.BookId)
    <br />

    @Html.Label("Title", "Book Title:")
    <br />
    @Html.EditorFor(b => b.Title, new { htmlAttributes = new { @class = "form-control" } })
    @Html.ValidationMessageFor(b => b.Title, "", new { @class = "text-danger" })
    <br />
    <br />

    @Html.Label("Author", "Author:")
    <br />
    @Html.Editor("Author", new { htmlAttributes = new { @class = "form-control" } })
    @Html.ValidationMessageFor(b => b.Author, "", new { @class = "text-danger" })
    <br />
    <br />

    @Html.Label("Summary", "Summary:")
    <br />
    @Html.TextAreaFor(b => b.Summary, new { @class = "form-control", @rows = "10" })
    @Html.ValidationMessageFor(b => b.Summary, "", new { @class = "text-danger" })
    <br />
    <br />

    <button class="btn btn-primary" type="submit">Update</button>
}

```

### Despre Delete

Creata in BookController actiunea Delete de tipul DELETE.

```

[HttpDelete]
public ActionResult Delete(int id)
{
    Book book = db.Books.Find(id);
    if (book != null)
    {

```

```

        db.Books.Remove(book);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return HttpNotFound("Couldn't find the book with id " + id.ToString());
}

```

Creati view-ul Delete.cshtml in Views/Book. NU trebuie sa adaugati nimic in el.

#### Modificari in Index.cshtml

Modificati view-ul Views/Book/Index.cshtml astfel incat dupa ce se afiseaza lista de carti sa se afiseze pe ecran un link catre pagina de adaugare a unei carti noi in baza de date.

```

@{
    ViewBag.Title = "Books";
}

<h2>@ViewBag.Title</h2>

@foreach (var book in ViewBag.Books)
{
    <div class="panel panel-default">
        <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 style="margin-right:5px" class="btn btn-primary col-lg-1"
type="submit">Summary</button>
            }
        </div>
    </div>
}

@Html.ActionLink("Add book", "New")

```

#### Modificari in Details.cshtml

Modificati view-ul Views/Book/Details.cshtml astfel incat dupa ce se afiseaza toate detaliile unei carti sa afiseze pe ecran si doua butoane: UPDATE, DELETE cu un link fiecare catre pagina corespunzatoare actiunii din controller.

```

@model lab3_miercuri.Models.Book

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

<h2>Details</h2>

<div class="panel panel-default">
    <div class="panel-body">
        @Html.Label("Title", "Title:")
        <br />
        @Html.DisplayFor(b => b.Title)
    
```

```

        <br />
        <br />

        @Html.Label("Author", "Author:")
        <br />
        @Html.DisplayFor(b => b.Author)
        <br />
        <br />

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

            @Html.Label("Publisher", "Contact Info:")
            <br />
            @Html.DisplayFor(b => b.Publisher.ContactInfo.PhoneNumber)
            <br />
            <br />
        }

        @Html.Label("Summary", "Summary:")
        <br />
        <div class="panel-body">
            @if (Model.Summary.IsEmpty())
            {
                <p>@emptySummaryMsgVar</p>
            }
            else
            {
                @Html.Display("Summary")
                <br /><br />
            }
        </div>

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

@using (Html.BeginForm(actionName: "Edit", controllerName: "Book", method: FormMethod.Get, routeValues:
new { id = Model.BookId }))
{
    <button style="margin-right:5px" class="btn btn-primary col-lg-1" type="submit">Update</button>
}

@using (Html.BeginForm(actionName: "Delete", controllerName: "Book", method: FormMethod.Post, routeValues:
new { id = Model.BookId }))
{
    @Html.HttpMethodOverride(HttpVerbs.Delete)
    <button class="btn btn-primary col-lg-1" type="submit">Delete</button>
}
<br />

```

## CRUD pentru relatie one-to-many – Exercițiul 2

### Despre Chei Primare (Primary Key - PK)

[Cerinta] Creati o clasa ce va reprezenta un tip de carte (e.g. Hardcover, Paperback) si va avea id si nume.

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

namespace lab4.Models
{
    public class BookType
    {
        [Key] // nu este necesar
        public int BookTypeId { get; set; }
        public string Name { get; set; }
    }
}
```

### Despre Chei Externe (Foreign Key - FK)

[Cerinta] In clasa Book creati o legatura many-to-one catre acea clasa avand proprietatea de cheie externa scrisa explicit; proprietatea de referinta trebuie sa fie virtuala (de ce?).

In clasa **BookType** scrieti utmatotrul cod:

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

namespace lab4.Models
{
    public class BookType
    {
        public int BookTypeId { get; set; }
        public string Name { get; set; }

        // many to one
        public virtual ICollection<Book> Books { get; set; }
    }
}
```

Atributul **[Key]** este folosit pentru a specifica in mod explicit ca proprietatea pentru care este aplicata se mapeaza in baza de date ca fiind o cheie primara.

### !!!! ATENTIE !!!

Daca o proprietate poarta numele de ID sau <numele entitatii>ID (nu se tine cont de majuscule sau litere mici), aceasta va fi configurata ca o **cheie primara**. In concluzie, daca numiti proprietatea ca mai sus, nu mai este necesar sa folositi atributul **[Key]**.

Deci, in mod implicit, EF considera ca fiind chei primare DOAR proprietatile de forma: Id, BookId, ID, BookID.

```
1. public class Book
2. {
3.     public int Id { get; set; }
4.     public string Title { get; set; }
5. }

1. public class Book
2. {
3.     public int BookId { get; set; }
4.     public string Title { get; set; }
5. }
```

EF (Entity Framework) va specifica ca valorile coloanei cheii primare sunt generate automat de BD.

In cazul de mai jos este **obligatorie** folosirea atributului **[Key]** deoarece numele proprietatii **nu** respecta "naming convention":

```
1. public class Order
2. {
3.     [Key]
4.     public int OrderNumber { get; set; }
5.     public DateTime DateCreated { get; set; }
6. }
```

In clasa **Book** scrieti urmatorul cod:

```
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.Mvc;

namespace lab4.Models
{
    public class Book
    {
        [Key] // nu este necesar
        public int BookId { get; set; }

        [MinLength(5, ErrorMessage = "Title cannot be less than 5"),
        MaxLength(200, ErrorMessage = "Title cannot be more than 200")]
        public string Title { get; set; }
        ....

        // one to many
        public int BookTypeId { get; set; }

        [ForeignKey("BookType")]
        public virtual BookType BookType { get; set; }
        .....
    }
    .....
```

#### Explicatii cheie externa

Atributul **[ForeignKey(name)]** poate fi aplicat in 3 modalitati:

1. **[ForeignKey(NavigationPropertyName)]** pe proprietatea cheii externe din entitatea dependenta. Primeste ca parametru numele entitatii.

```
1. public class Student
2. {
3.     public int StudentId { get; set; }
4.     public string Name { get; set; }
5.
6.     [ForeignKey("School")]
7.     public int SchoolRefId { get; set; }
8.     public virtual School School { get; set; }
9. }
10.
11. public class School
12. {
13.     public int SchoolId { get; set; }
14.     public string Name { get; set; }
15.
16.     // many to one
17.     public virtual ICollection<Student> Students { get; set; }
18. }
```

In acest caz, EF va crea **SchoolRefId** ca fiind coloana cheii externe din tabelul **Students**. In tabel **nu** se va mai genera cheia externa implicita **SchoolId**.

Atributul **[ForeignKey(name)]** este folosit pentru a specifica in mod explicit ca proprietatea pentru care este aplicata se mapeaza in baza de date ca fiind o cheie externa.

#### !!! ATENTIE !!!

Acesta suprascrie conventia implicita pentru o cheie externa. Ne permite sa marcam proprietatea din entitatea dependenta al carei **nume nu se potriveste** cu proprietatea cheii primare a entitatii principale ca fiind o cheie externa.

Deci, in mod implicit, EF modifica o proprietate in proprietate cu cheie externa atunci cand numele sau este **identic** cu proprietatea cu cheie primara a unei entitati conexe. Nu este necesar aplicarea atributului **[ForeignKey(name)]** pt cazul:

```
1. public class Student
2. { // cheie primara implicita
3.     public int StudentId { get; set; }
4.     public string Name { get; set; }
5.
6.     // cheie externa implicita
7.     public int SchoolId { get; set; }
8.     public virtual School School { get; set; }
9. }
10.
11. public class School
12. { // cheie primara implicita
13.     public int SchoolId { get; set; }
14.     public string Name { get; set; }
15.
16.     // many to one
17.     public virtual ICollection<Student>
18.         Students { get; set; }
19. }
```

2. `[ForeignKey(ForeignKeyPropertyName)]` pe proprietatea de referinta a entitatii principale si primeste ca parametru numele cheii externe.

```
1. public class Student
2. {
3.     public int StudentID { get; set; }
4.     public string Name { get; set; }
5.
6.     public int SchoolRefId { get; set; }
7.     public virtual School School { get; set; }
8. }
9.
10. public class School
11. {
12.     public int SchoolId { get; set; }
13.     public string Name { get; set; }
14.
15.     // many to one
16.     [ForeignKey("SchoolRefId")]
17.     public virtual ICollection<Student> Students { get; set; }
18. }
```

In acest caz, atributul este aplicat pe proprietatea de referinta **Students** din entitatea principala **School**. EF va crea coloana cheii externe numita **SchoolRefId** in tabelul **Students** din BD. In tabel **nu** se va mai genera cheia externa implicita **SchoolId**.

3. `[ForeignKey(ForeignKeyPropertyName)]` pe proprietatea de referinta a entitatii dependente si primeste ca parametru numele cheii externe.

```
1. public class Student
2. {
3.     public int StudentID { get; set; }
4.     public string Name { get; set; }
5.
6.     public int SchoolRefId { get; set; }
7.
8.     [ForeignKey("SchoolRefId")]
9.     public virtual School School { get; set; }
10. }
11.
12. public class School
13. {
14.     public int SchoolId { get; set; }
15.     public string Name { get; set; }
16.
17.     // many to one
18.     public virtual ICollection<Student> Students { get; set; }
19. }
```

In acest caz, atributul este pus pe proprietatea de referinta **School** din **Student**.

Explicatii pentru cuvantul cheie **virtual**

*De ce proprietatea de referinta trebuie sa fie virtuala?*

EF necesita ca proprietatile de referinta sa fie marcate ca fiind virtuale, astfel incat sa fie acceptat *'lazy loading'* si urmarirea eficienta a modificarilor. EF foloseste **mostenirea** pentru a sprijini aceasta functionalitate, motiv pentru care necesita ca anumite proprietati sa fie marcate virtuale in casele de



baza. Deci, EF, prin cuv. cheie virtual creeaza noi tipuri care deriva din tipurile implementate. Subclasele create dinamic de EF devin vizibile la **run time**, NU la **compile time**.

- **Lazy Loading:** Cand entitatea este citita pentru prima data, datele aferente nu sunt recuperate. Cu toate acestea, prima data cand incercati sa accesati o proprietate de referinta, datele necesare pentru acea proprietate de navigare sunt recuperate automat. Acest lucru are ca rezultat mai multe interogari trimise la BD – una pentru entitatea in sine si una de fiecare data cand trebuie recuperate datele aferente pentru entitate. Clasa DbContext primeste implicit *lazy loading*.

```
departments = context.Departments
foreach (Department d in departments)
{
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows

Query: Course rows related to Department d

- **Eager Loading:** Cand entitatea este citita, datele aferente sunt recuperate impreuna cu acestea. De obicei, rezulta o singura interogare care recupereaza toate datele necesare. Eager Loading se specifica utilizand metoda **Include**. => face join

```
departments = context.Departments.Include(x => x.Courses)
foreach (Department d in departments)
{
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows and related Course rows

[Cerinta] In clasa context adaugati un tabel nou pentru tipuri de carte.

```
public class DbCtx : DbContext
{
    public DbCtx() : base("DbConnectionString")
    {
        Database.SetInitializer<DbCtx>(new Initp());
    }
    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 DbSet<BookType> BookTypes { get; set; }
}
```

[Cerinta] In clasa initializator adaugati tipuri de carte in BD inainte de a adauga carti (scriind explicit cheia primara), iar fiecarei carti adaugati-i tipul sub forma cheii externe.

```
namespace lab4.Models
{
    ....
    public class Initp : DropCreateDatabaseIfModelChanges<DbCtx>
    {
        protected override void Seed(DbCtx ctx)
        {

```

```

BookType cover1 = new BookType { BookTypeId = 60, Name = "HardCover" };
BookType cover2 = new BookType { BookTypeId = 61, Name = "Paperback" };

ctx.BookTypes.Add(cover1);
ctx.BookTypes.Add(cover2);

ctx.Books.Add(new Book
{
    Title = "The Atomic Times",
    Author = "Michael Harris",
    Publisher = new Publisher { Name = "HarperCollins", ContactInfo = new ContactInfo {
PhoneNumber = "07123456789" } },
    BookTypeId = cover1.BookTypeId,
    Genres = new List<Genre> {
        new Genre { Name = "Horror" }
    }
});
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" } },
    BookTypeId = cover2.BookTypeId,
    Genres = new List<Genre> {
        new Genre { Name = "Satire"},
        new Genre { Name = "Fabilau"},
        new Genre { Name = "Allegory"},
        new Genre { Name = "Burlesque"}
    }
});
ctx.SaveChanges();
base.Seed(ctx);
}
}
}

```

[Cerinta] Creati un view model ce pe langa carte va contine si o lista de tipuri de carte.

```

public class Book
{
    ...

    [NotMapped]
    public IEnumerable<SelectListItem> PublishersList { get; set; }

    [NotMapped]
    public IEnumerable<SelectListItem> BookTypesList { get; set; }
}

```

[Cerinta] In actiunile New/Edit veti pasa view-ului un obiect din acest viewmodel in care ati extras din BD lista tipurilor de carte.

In **BookController** adaugati metoda:

```

[NonAction] // specificam faptul ca nu este o actiune
public IEnumerable<SelectListItem> GetAllBookTypes()
{
    // generam o lista goala
    var selectList = new List<SelectListItem>();

    foreach (var cover in db.BookTypes.ToList())
    {
        // adaugam in lista elementele necesare pt dropdown
        selectList.Add(new SelectListItem
        {
            Value = cover.BookTypeId.ToString(),
            Text = cover.Name
        });
    }
    // returnam lista pentru dropdown
    return selectList;
}

```

In **BookController** adaugati in actiunea **Edit** de tipul GET:

```

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

```

In **BookController** adaugati in actiunea **Edit** de tipul PUT:

```

[HttpPut]
public ActionResult Edit(int id, Book bookRequest)
{
    try
    {
        bookRequest.BookTypesList = GetAllBookTypes();
        if (ModelState.IsValid)
        {
            Book book = db.Books
                .Include("Publisher")
                .SingleOrDefault(b => b.BookId.Equals(id));

            if (TryUpdateModel(book))
            {
                book.Title = bookRequest.Title;
                book.Author = bookRequest.Author;
                book.Summary = bookRequest.Summary;
                db.SaveChanges();
            }
            return RedirectToAction("Index");
        }
        return View(bookRequest);
    }
    catch (Exception e)
    {
        return View(bookRequest);
    }
}

```

In **BookController** adaugati in actiunea **NEW** de tipul GET:

```
[HttpGet]
public ActionResult New()
{
    Book book = new Book();
    // preluam lista de booktypes necesara dropdown-ului
    book.BookTypesList = GetAllBookTypes();
    book.Genres = new List<Genre>();
    return View(book);
}
```

In **BookController** adaugati in actiunea **NEW** de tipul POST:

```
[HttpPost]
public ActionResult New(Book bookRequest)
{
    try
    {
        bookRequest.BookTypesList = GetAllBookTypes();
        if (ModelState.IsValid)
        {
            db.Books.Add(bookRequest);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        return View(bookRequest);
    }
    catch (Exception e)
    {
        return View(bookRequest);
    }
}
```

[Cerinta] In view-urile New/Edit modelul va fi acel view model. Adaugati o lista derulanta prin care utilizatorul va putea selecta tipul de carte.

In **Views/Book/Details.cshtml** adaugati:

```
@if (Model.BookType != null)
{
    @Html.Label("BookType", "Book Type:")
    <br />
    @Html.DisplayFor(b => b.BookType.Name)
    <br />
    <br />
}
```

In **Views/Book/Edit.cshtml** si in **Views/Book/New.cshtml** adaugati:

```
@Html.Label("BookType", "BookType:")
<br />
@Html.DropDownListFor(b => b.BookTypeId, new SelectList(Model.BookTypesList, "Value", "Text"), "Choose
a book type", new { @class = "form-control" })
@Html.ValidationMessageFor(b => b.BookTypeId, "", new { @class = "text-danger" })
<br />
<br />
```

**TEMA:** Implementati operatiile CRUD pentru entitatea Publisher. Intre Publisher si Book avem o relatie many-to-one. Operatia Read a fost deja implementata in laboratorul 3 (actiunile de Index si Details). Trebuie sa urmati aceiasi pasi ca in cazul lui BookType. Voi va trebui sa:

- In clasa Book adaugati o lista IEnumerable<SelectListItem> ce va contine toate firmele de publicatii.

- Adaugati in controller-ul Book o metoda care sa nu fie actiune si sa returneze o lista de tipul `IEnumerable<SelectListItem>` ce extrage din baza de date lista tuturor firmelor de publicatii.
- Adaugati in actiunile Edit de tipul Get si Put linia de cod care trimite view-ului lista necesara drop-down-ului ce contine toate firmele de publicatii din baza de date.
- Adaugati in Views/Book/New si Views/Book/Edit structura de cod necesara pentru a afisa pe ecran o lista derulabila.