

Dezvoltarea Aplicatiilor Web-Anul 3

Laborator 5

Solutiile laboratoarelor se gasesc aici:

<https://drive.google.com/drive/folders/1IGMximha8QfpEO-CGU9cyIZ4liMLXa3E?usp=sharing>

Exemplu pentru many-to-many:

https://www.entityframeworktutorial.net/code-first/configure-many-to-many-relationship-in-code-first.aspx?fbclid=IwAR19T-PgcfvenuE2KHBms1mI_AWW6Y1E6OcJZVHWRgK-wXNjxvDpIJ69KeM

Exercitiul 1

Se considera baza de date, cu cele doua modele Article.cs si Category.cs, din Laborator 4. Modificati atat View-urile, cat si metodele din Controllere (atunci cand este necesar) astfel incat sa se utilizeze helpere pentru View. Pentru exemple (**Vezi** Curs 5).

Daca se face o copie a proiectului realizat in laboratorul anterior, atentie la stringul de conexiune pentru baza de date (connection string). Acesta trebuie inlocuit in **Web.config** cu noul connection string (pentru a prelua noul connection string -> in Solution Explorer -> dublu click pe numele bazei de date -> se va deschide Server Explorer -> aici click dreapta pe baza de date -> proprietati -> din fereastra aparuta se copiaza connection string-ul).

Sa se modifice **ArticleController**, folosind helpere pentru View (**Vezi** Curs 5, sectiunea Helpere pentru View). De asemenea, pentru adaugare, editare si stergere, sa se utilizeze mesaje corespunzatoare folosind variabile de tipul **TempData** (**Vezi** Curs 5, sectiunea Trimiterea datelor catre View -> TempData). Exemplu de mesaje: "Articolul a fost adaugat cu success!", "Articolul a fost modificat!", "Articolul a fost sters!".

Pentru trimiterea datelor din baza de date, se va utiliza helper-ul **@model** (**Vezi** Curs 5, sectiunea Trimiterea datelor catre View). Se foloseste acest helper doar in cazul in care se transmite catre View o singura instanta de model. Pentru trimiterea array-urilor de obiecte, se folosesc in continuare variabile de tip ViewBag.

Sugestii de implementare:

ArticleController

Index – afisarea tuturor articolelor, afisand si categoria corespunzatoare fiecarui articol. Fiind array de obiecte de tipul Article, pentru transferul de date intre Controller si View, se foloseste in continuare variabila ViewBag. De asemenea, pe pagina Index o sa existe legatura (link/ancora) cu pagina de afisare a unui singur articol (Show), cu pagina de adaugare a unui nou articol (New) si cu pagina de afisare a tuturor categoriilor (/Category/Index).

Sa se modifice ruta Default din App_Start -> RouteConfig.cs, astfel incat in momentul rularii aplicatiei sa redirectioneze de fiecare data catre pagina de afisare a tuturor articolelor.

```
// GET
public ActionResult Index()
{
    var articles = db.Articles.Include("Category");
    ViewBag.Articles = articles;

    return View();
}
```

Index.cshtml

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

<h2>@ViewBag.Title</h2>
<hr />
@foreach (Laborator4.Models.Article article in ViewBag.Articles)
{
    <div class="panel panel-default">
        <div class="panel-heading">@article.Title</div>
        <div class="panel-body">
            Continut articol: <strong>@article.Content</strong>
            <br />
            <span class="label label-success">@article.Date</span>
            <br />
            <i class="glyphicon glyphicon-globe"></i> @article.Category.CategoryName
        </div>
    </div>
}
```

```

        <div class="panel-footer">
            <a class="btn btn-sm btn-success"
href="/Article/Show/@article.Id">Afisare articol</a>
        </div>
    </div>
    <br />

    <br />
    <a class="btn btn-success" href="/Category/Index">Afisare lista categorii</a>
    <br /><br />
    <a class="btn btn-info" href="/Article/New">Adauga articol</a>
    <br />
}

```

New – pagina de adaugare a unui articol nou. In acest caz, pe langa formularul in care se completeaza datele noului articol, este necesara si selectarea unei categorii. Astfel, avem nevoie de o metoda (ex: GetAllCategories) cu ajutorul careia sa preluam toate categoriile din baza de date si sa le stocam intr-o lista de tipul SelectListItem. O astfel de lista poate fi incarcata ulterior intr-un dropdown, cu ajutorul helper-ului

@Html.DropDownListFor (Vezi Curs 5, cautand DropDownListFor).

ATENTIE! tot in aceasta sectiune a cursului se observa necesitatea adaugarii unei noi proprietati in Model, cu ajutorul careia putem prelua si transfera toate categoriile din baza de date in helper-ul pentru DropDownList. In Controller, vom apela metoda GetAllCategories si vom trimite lista tuturor categoriilor catre View.

In View folosim urmatoarele helpere: @Html.Label – pentru texte, @Html.TextBox – pentru inputuri, @Html.TextArea – pentru continutul articolului, @Html.Hidden – pentru data, deoarece data sa va genera automat folosind pentru al doilea parametru DateTime.Now si @Html.DropDownListFor – pentru incarcarea listei de categorii in dropdown (**Vezi Curs 5, sectiunea Helpere pentru View -> adaugarea unui student in baza de date**).

In metoda New cu HttpPost, intr-o variabila de tipul TempData, sa se stocheze un mesaj “Articolul a fost adaugat”. Mesajul se va afisa in pagina Index, dupa ce vom fi redirectionati acolo, cu ajutorul metodei RedirectToAction. Pentru adaugarea de mesaje in variabile de tip TempData (**Vezi Curs 5, sectiunea Trimiterea datelor catre View – si modificati atat metoda Index, cat si View-ul asociat acesteia**).

public IEnumerable<SelectListItem> Categories { get; set; } -> atributul pentru preluarea categoriilor

```
public ActionResult New()
{
    Article article = new Article();

    // preluam lista de categorii din metoda GetAllCategories()

    article.Categories = GetAllCategories();

    return View(article);
}

[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;
}
```

View-ul New

```
@model Laborator4.Models.Article

@{
    ViewBag.Titlu = "Adaugare articol";
}

<h2>@ViewBag.Titlu</h2>
<br/>
<form method="post" action="/Article/New">

    @Html.Label("Title", "Titlu Articol")
    <br />
    @Html.TextBox("Title", null, new { @class = "form-control" })
    <br /><br />

    @Html.Label("Content", "Continut Articol")
    <br />
    @Html.TextArea("Content", null, new { @class = "form-control" })
    <br /><br />

    @Html.Hidden("Date", DateTime.Now, new { @class = "form-control" })

    <label>Selectati categoria</label>
    @Html.DropDownListFor(m => m.CategoryId, new SelectList(Model.Categories,
"Value", "Text"),
    "Selectati categoria", new { @class = "form-control" })
    <br />

    <button class="btn btn-sm btn-success" type="submit">Adauga articol</button>

</form>
```

Metoda New cu HttpPost

```
[HttpPost]
public ActionResult New(Article article)
{
    try
    {
        db.Articles.Add(article);
        db.SaveChanges();
        TempData["message"] = "Articolul a fost adaugat!";
        return RedirectToAction("Index");
    }
    catch (Exception e)
    {
        return View();
    }
}
```

Metoda Show – metoda de afisare a unui singur articol, impreuna cu categoria din care face parte. In acest caz se poate transmite obiectul de tipul modelului catre View, ca parametru in return -> return View(numeObiect).

```
public ActionResult Show(int id)
{
    Article article = db.Articles.Find(id);
    return View(article);
}
```

View-ul Show – in View trebuie inclus modelul pentru a putea utiliza helper-ul @model (**Vezi** Curs 5, cautand dupa @model). Se include astfel: @model Laborator4.Models.Article, unde Laborator4 este numele aplicatiei, care devine namespace, iar Article este numele clasei (numele modelului). Acum pentru afisarea unui atribut din baza de date se foloseste de exemplu, @Model.Title. La fel se procedeaza si pentru afisarea categoriei, doar ca de aceasta data folosim proprietatea Category existenta in modelul Article, adaugata in momentul in care am definit relatia one-to-many: @Model.Category.CategoryName. De asemenea, atunci cand afisam datele articolului trebuie sa existe si o legatura (link) catre pagina de editare a acestui articol, cat si alte doua link-uri catre lista tuturor articolelor si catre pagina de adaugare a unui nou articol.

```
@model Laborator4.Models.Article

@{
    ViewBag.Title = "Show";
}

<h1>Afisare articol</h1>

<div class="panel panel-default">
    <div class="panel-heading">@Model.Title</div>
    <div class="panel-body">
        Continut articol: <strong>@Model.Content</strong>
        <br />
        <span class="label label-success">@Model.Date</span>
        <br />
        <i class="glyphicon glyphicon-globe">@Model.Category.CategoryName</i>
    </div>
    <div class="panel-footer">
        <a class="btn btn-success" href="/Article/Edit/@Model.Id">Modifica
articol</a>
        <br />
    </div>
</div>
```

```

<form method="post" action="/Article/Delete/@Model.Id">

    @Html.HttpMethodOverride(HttpVerbs.Delete)
    <br />
    <button class="btn btn-success" type="submit">Sterge articol</button>

</form>

<br />
<hr />
<a class="btn btn-success" href="/Article/Index">Inapoi la articole</a>
<br /><br />
<a class="btn btn-info" href="/Article/New">Adauga articol</a>
<br />

```

Metoda Edit – pentru editarea unui singur articol trebuie sa afisam in metoda Edit cu HttpGet un formular in care sa preluam datele articolului respectiv, existente in baza de date. De asemenea, trebuie sa afisam si categoria, intr-un dropdown, oferind astfel posibilitatea utilizatorului de a selecta o noua categorie in momentul editarii. In dropdown o sa fie selectata categoria existenta, asociata articolului. Si in acest caz trimitem obiectul de tipul modelului catre View pentru a putea utiliza helper-ul **@model**.

```

public ActionResult Edit(int id)
{
    Article article = db.Articles.Find(id);
    article.Categories = GetAllCategories();
    return View(article);
}

```

View-ul Edit – se vor utiliza helpere pentru View, dupa cum urmeaza: @Html.HiddenFor(m => m.Id), - pentru a mentine id-ul in pagina, incat sa se poata realiza modificarea cu succes, @Html.Label("Title", "Titlu Articol") - pentru adaugarea label-urilor, @Html.EditorFor(m => m.Title) - pentru campurile din baza de date; prin intermediul acestui helper valorile atributelor se vor selecta in formular, @Html.HiddenFor(m => m.Date) - pentru campul de tip Date, deoarece nu dorim ca data sa fie introdusa manual; aceasta se genereaza in momentul postarii articolului. Dropdown-ul se implementeaza la fel ca atunci cand adaugam un articol nou. Si in acest caz vom utiliza helper-ul **@model**.

```

@model Laborator4.Models.Article

@{
    ViewBag.Titlu = "Editare Articol";
}

<h2>@ViewBag.Titlu</h2>

<form method="post" action="/Article/Edit/@Model.Id">

    @Html.HttpMethodOverride(HttpVerbs.Put)

    @Html.HiddenFor(m => m.Id)
    <br />
    @Html.Label("Title", "Titlu Articol")
    <br />
    @Html.EditorFor(m => m.Title)
    <br /><br />

    @Html.Label("Content", "Continut Articol")
    <br />
    @Html.EditorFor(m => m.Content)
    <br /><br />

    @Html.HiddenFor(m => m.Date)
    <br />

    <label>Selectati categoria</label>
    @Html.DropDownListFor(m => m.CategoryId,
        new SelectList(Model.Categories, "Value", "Text"),
        "Selectati categoria", new { @class = "form-control" })
    <br />

    <button class="btn btn-sm btn-success" type="submit">Modifica articol</button>

</form>

```

Metoda Edit cu HttpPut – salvam modificarile in baza de date si folosim o variabila de tip TempData pentru a trimite catre urmatorul request (catre pagina spre care facem redirectionarea -> Index) mesajul “Articolul a fost modificat!”.

```

[HttpPut]
public ActionResult Edit(int id, Article requestArticle)
{
    try
    {
        Article article = db.Articles.Find(id);

        article.Title = requestArticle.Title;
    }
}

```



```

        article.Content = requestArticle.Content;
        article.Date = requestArticle.Date;
        article.CategoryId = requestArticle.CategoryId;
        db.SaveChanges();
        TempData["message"] = "Articolul a fost modificat!";
        return RedirectToAction("Index");
    }
    catch (Exception e)
    {
        return View();
    }
}

```

View-ul Delete – dupa cum am observat si in laboratoarele trecute, pentru Delete nu este necesar un View de afisare. In acest caz putem utiliza View-ul Show deoarece aici afisam datele unui singur articol. In cazul stergerii, nu trebuie sa stergem si categoria. Intr-o categorie fiind mai multe articole, in momentul in care stergem articolul, categoria obligatoriu trebuie sa raman in baza de date.

```

<form method="post" action="/Article/Delete/@Model.Id">
    @Html.HttpMethodOverride(HttpVerbs.Delete)
    <br />
    <button class="btn btn-success" type="submit">Sterge articol</button>
</form>


```

Metoda Delete – pentru metoda delete, gasim articolul cu id-ul corespunzator, stergem articolul si afisam prin intermediul unei variabile TempData mesajul “Articolul a fost sters!”.

```

[HttpDelete]
public ActionResult Delete(int id)
{
    Article article = db.Articles.Find(id);
    db.Articles.Remove(article);
    db.SaveChanges();
    TempData["message"] = "Articolul a fost sters!";
    return RedirectToAction("Index");
}

```

 **TEMA:** Implementati in aceeași manieră, folosind helpere corespunzătoare, (@model pentru model, diverse helpere pentru view) și variabile (TempData pentru afișarea mesajelor, ViewBag pentru transmiterea datelor către View) și pentru CategoryController. De asemenea, din pagina de afișare a tuturor categoriilor să existe o legătură către pagina de afișare a tuturor articolelor.