Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Laborator 7

EXERCIȚII:

Se consideră baza de date, cu cele trei modele Article.cs, Category.cs și Comment.cs din Laborator 6.

Descărcați proiectul, numit ***ArticlesAppLab7Template***, și modificați atât View-urile, cât și metodele din Controllere (atunci când este necesar), astfel încât să se utilizeze helpere pentru View. Se va utiliza varianta cu Tag Helpers Pentru exemple – **VEZI Curs 7 Secțiunea Helpere pentru View.**

După descărcarea proiectului, se creează o nouă bază de date și se modifică stringul de conexiune din appsettings.json.

După acest pas, se execută o migrație, numită ***AddUserArticleCategoryCommentModels***

Nu uitați de Update-Database, imediat după adăugarea migrației!

În continuare, se vor implementa următoarele funcționalități:

ArticlesController și View-urile asociate:

1. Studiați secțiunea **Trimiterea datelor către View → Model** din cadrul Cursului 7, după care implementați următoarea funcționalitate:

    a. Să se modifice acțiunea (metoda) **Show** din Controller-ul Articles. Se implementează corect query-ul, realizând operația ***join***. Se adaugă condiția corectă în clauza ***where***. La final, se trimite Modelul de tip ***Article*** către View-ul asociat. Pasul următor este modificarea View-ului, astfel încât să includă modelul și să afișeze proprietățile acestuia către utilizatorul final. Elementele care necesită modificări sunt cele scrise cu negru si bold.

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

                <h3 class="card-title alert-success py-3 px-3 rounded-2">Titlul
articolului</h3>

                <div class="card-text">Continutul</div>

                <div class="d-flex justify-content-between flex-row mt-5">

                        <div><i class="bi bi-globe"></i> Numele categoriei din
care face parte articolul</div>

                        <span class="alert-success">Data la care a fost publicat
articolul</span>

                </div>

        </div>


        …

        <div class="d-flex flex-row justify-content-between">

                <a asp-controller="Articles" asp-action="Edit" asp-route-
id="articolul pe care dorim sa-l editam" class="btn btn-success">Editează
articol</a>

                <form method="post" asp-controller="Articles" asp-
action="Delete" asp-route-id="articolul pe care dorim sa-l stergem">

                        <button class="btn btn-danger" type="submit">Sterge
articol</button>

                </form>

        </div>
```

Analizați modificările realizate asupra link-urilor, în momentul în care se utilizează Tag Helpers.

Link-ul inițial:

```
<a class="btn btn-success" href="/Articles/Edit/@Model.Id">Editeaza
articol</a>
```

Link-ul utilizând Tag Helpers:

```
<a asp-controller="Articles" asp-action="Edit" asp-route-id="@Model.Id"
class="btn btn-success">Editează articol</a>
```

Analizați implementările realizate asupra modelului ***Comment***. Modificați link-ul următor, folosind Tag Helpers:

```
<div>

    <a class="btn btn-outline-primary" href="/Comments/Edit/@comm.Id"><i
class="bi bi-pencil-square"></i>Editeaza</a>

</div>
```

Analizați formularul de ștergere a unui comentariu, împreună cu acțiunea specifică din Controller.

Analizați formularul în care se poate adăuga un comentariu. De ce este nevoie de acest input?

```
<input type="hidden" name="ArticleId" value="@Model.Id" />
```

Analizați metoda pentru adăugarea unui comentariu în baza de date. De ce este nevoie de redirect către metoda ***Show*** din ***ArticlesController***?

```
...
return Redirect("/Articles/Show/" + comm.ArticleId);

...
```

2. Modificați funcționalitatea de **adăugare a unui nou articol** în baza de date, împreună cu selectarea categoriei din care face parte, astfel:

    a. **Metoda New cu Get** o să apeleze o metodă, cu ajutorul căreia se vor prelua categoriile din baza de date, în vederea trimiterii acestora în View-ul asociat, pentru a popula un element de tip dropdown. Este necesar un dropdown deoarece în momentul în care utilizatorul final dorește să adăuge un nou articol în baza de date, acesta trebuie să aleagă și o categorie din care o să facă parte articolul respectiv. Alegerea categoriei o să se realizeze cu

ajutorul un dropdown. **Ce fel de metoda trebuie să fie metoda prin care se preiau toate categoriile din baza de date?**

```
public IEnumerable<SelectListItem> GetAllCategories()
   {
        // generam o lista de tipul SelectListItem fara elemente
        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
            // id-ul categoriei si denumirea acesteia
            selectList.Add(new SelectListItem
            {
                Value = category.Id.ToString(),
                Text = category.CategoryName.ToString()
            });
        }

            /* Sau se poate implementa astfel:
             *
            foreach (var category in categories)
            {
                var listItem = new SelectListItem();
                listItem.Value = category.Id.ToString();
                listItem.Text = category.CategoryName.ToString();

                selectList.Add(listItem);
             }*/


        // returnam lista de categorii
        return selectList;
    }
```

Implementare dropdown:

După implementarea metodei anterioare, prin care sunt preluate toate categoriile din baza de date, urmează apelarea acesteia în metoda New din Controller-ul Articles și stocarea categoriilor cu ajutorul unei proprietăți, după cum urmează:

```
public IActionResult New()
   {

       Article article = new Article();

       article.Categ = GetAllCategories();

       return View(article);
    }
```

- Se instanțiază clasa Article deoarece în cadrul formularului o să se creeze un nou obiect în baza de date, de tipul modelului Article;

- Se apelează metoda `GetAllCategories()` prin care sunt preluate toate categoriile din baza de date și se stochează aceste categorii pentru a fi trimise către View-ul New;

- Stocarea listei care conține toate categoriile din baza de date (de forma id_categorie – denumire categorie) se realizează prin adăugarea unei noi proprietăți în Modelul Article:

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

Cu ajutorul acestui atribut se pot prelua categoriile în cadrul Helper-ului `select`

```
<select asp-for="CategoryId" asp-items="Model.Categ"
class="form-control">
    <option value="">Selectati categoria</option>
</select>
```

- Se trimite un obiect de tipul modelului Article către View-ul asociat, astfel încât View-ul să primească acest model prin intermediul Helperului @model

View-ul New:

Atenție la modul în care se include modelul!

Atenție la modul în care se implementează formularul, utilizând Tag Helpers!

```
@model ArticlesApp.Models.Article

<h2 class="text-center mt-5">Adaugare articol</h2>

<br />

<div class="container mt-5">
    <div class="row">
        <div class="col-6 offset-3">

            <form method="post" asp-action="New" asp-controller="Articles">

                <div class="form-group">
                    <label asp-for="Title" class="form-label">Titlu
Articol</label>
                    <br />
                    <input asp-for="Title" class="form-control" />
                    <br /><br />
                </div>

                <div class="form-group">
                    @* Aici se adauga codul pentru continutul articolului*@
                </div>

                <div class="form-group">
                    @* Aici se adauga codul pentru dropdown*@
                </div>

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

            </form>
        </div>
    </div>
</div>
```

În View există și varianta formularului în care se utilizează HTML Helpers (această variantă este comentată).

b. **Metoda New cu POST** – să se adauge articolul în baza de date și să se afișeze un mesaj sugestiv "Articolul a fost adăugat". Pentru afișarea mesajului o să se utilizeze o variabilă de tip TempData conform **Curs 7 – Secțiunea Trimiterea Datelor către View -> TempData**.

   Ce trebuie să returnăm în cazul în care adăugarea articolului în baza de date nu s-a putut realiza cu succes (pe ramura catch())?

3. Modificați funcționalitatea de **editare a unui articol existent** în baza de date, împreună cu selectarea categoriei din care face parte, astfel:
   a. Actiunea Edit cu GET să folosească metoda `GetAllCategories()` pentru preluarea categoriilor din baza de date și popularea unui element de tip `select`
   b. Acțiunea Edit cu GET o să trimită către View-ul asociat un model de tip Article
   c. View-ul Edit trebuie să conțină Helpere pentru View

`Se vor completa campurile scrise cu negru si bold.`

Trebuie inclus modelul!

```
<h2 class="text-center mt-5">Editare articol</h2>

<br />

<div class="container mt-5">
    <div class="row">
        <div class="col-md-6 offset-3">
            <form>
                <div class="form-group">
                    <label asp-for="Title" class="form-label">Titlu
Articol</label>
                    <br />
                    <input asp-for="Title" class="form-control" />
                    <br /><br />
                </div>

                <div class="form-group">
                    <label asp-for="Content" class="form-label">Continut
Articol</label>
                    <br />
                    <textarea asp-for="Content" class="form-control"></textarea>
                    <br /><br />
                </div>
```

Se vor utiliza Tag Helpers pentru toti parametrii din formular!

```
                <div class="form-group">
                    <label for="se completeaza acest camp">Selectati
categoria</label>
                    <br />

                     @*Aici se include dropdown-ul, folosind helperul select*@

                    <br />
                </div>

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

            </form>

        </div>
    </div>
</div>
```

d. **Acțiunea Edit cu POST** să realizeze adăugarea modificărilor în baza de date și să se afișeze un mesaj corespunzător: "Articolul a fost modificat", utilizând o variabilă de tip TempData.

4. Modificați funcționalitatea de ștergere a unui articol din baza de date, astfel încât în metoda **Delete**, după realizarea operației de ștergere, să se afișeze mesajul: "Articolul a fost șters", utilizând o variabilă de tip TempData.
   Unde se află View-ul asociat acestei metode?
   Care sunt parametrii care se trimit în Controller?

5. Să se procedeze la fel ca în implementările anterioare și pentru entitatea **Category** (utilizarea helperelor pentru View și a mesajelor de tip **TempData** pentru afișarea mesajelor sugestive către utilizatorul final).

   În cazul ***CategoriesController*** si ***CommentsController***, analizați implementările existente.

Execuția proprietăților DbSet<>

Ce este DbSet?

În ApplicationDbContext.cs avem:

```
public DbSet<Article> Articles { get; set; }
public DbSet<Category> Categories { get; set; }
public DbSet<Comment> Comments { get; set; }
```

DbSet<T> este o proprietate de tip DbSet<TEntity>, care este o colecție de entități cu operații de query și modificare.

Implementarea internă:

1. Tipul real al DbSet<T>

DbSet<T> implementează IQueryable<T>, deci se poate utiliza LINQ. LINQ permite interogarea și manipularea datelor direct în C#, fără SQL. Existentă două stiluri: query syntax și method syntax.

Metodele LINQ nu execută imediat, ci se construiește un **expression tree** care se traduce în SQL la nevoie.

2. Cum funcționează în spate

Atunci când scriem:

```
var articles = db.Articles
                 .Include(a => a.Category)
                 .OrderByDescending(a => a.Date);
```

- db.Articles returnează un DbSet<Article>
- .Include() și .OrderByDescending() construiesc un expression tree
- Nu se execută nimic până când se materializeză (de ex: prin .ToList(), .First(), iterare în foreach, trimitere în View a datelor)

3. Ce se întâmplă la runtime

Entity Framework Core:

- Detectează proprietățile DbSet<T> din DbContext
- Le asociază cu tabele în baza de date
- Creează proxy-uri/metadate pentru query-uri și tracking

Un proxy este un obiect care acționează ca intermediar pentru alt obiect, interceptând accese și oferind funcționalități suplimentare (lazy loading, change tracking, etc.).

4. Operații utilizate des în practică

Query (citire):

- Aceasta NU execută query-ul imediat

```
IQueryable<Article> query = db.Articles.Where(a => a.Id
= 10);
```

- Query-ul se execută aici

```
List<Article> articles = query.ToList();
```

Add (adaugare):

```
db.Articles.Add(article);  // Adaugă în Change Tracker (nu în DB)
db.SaveChanges();          // Execută INSERT în baza de date
```

Find (căutare):

Article article = db.Articles.Find(id); // Execută imediat SELECT

Exemplu din codul nostru:

În ArticlesController.cs:

- Se construiește query, NU se execută

```
var articles = db.Articles
                 .Include(a => a.Category)
                 .OrderByDescending(a => a.Date);
```

- Se materializează - query-ul se execută în acest punct

```
ViewBag.Articles = articles;
```

Ce se întâmplă pas cu pas:

1. db.Articles - returnează DbSet<Article>
2. .Include(a => a.Category) - adaugă JOIN în expression tree
3. .OrderByDescending(a => a.Date) - adaugă ORDER BY
4. Query-ul se execută când este materializat
5. EF Core traduce expression tree-ul în SQL:

```
SELECT a.*, c.*
FROM Articles a
LEFT JOIN Categories c ON a.CategoryId = c.Id
ORDER BY a.Date DESC
```

Change Tracker

DbContext ține un Change Tracker cu statusurile entităților:

- Unchanged
- Added (db.Articles.Add())
- Modified (la modificarea proprietăților)
- Deleted (db.Articles.Remove())
- La SaveChanges(), se execută SQL-urile necesare.

Concluzie

- DbSet<T> este un wrapper peste IQueryable<T> cu operații de modificare
- Query-urile se construiesc ca expression trees și se execută la materializare
- Change Tracker gestionează modificările până la SaveChanges()
- EF Core traduce LINQ în SQL la nevoie