Dezvoltarea Aplicatiilor Web-Anul 2
Laborator 7

Exercitiul 1

Se considera baza de date, cu cele trei modele Article.cs si Category.cs, Comment.cs, din laboratorul anterior.

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 atat ArticleController (acolo unde este necesar), cat si View-urile asociate, astfel incat sa se adauge validarile necesare.

Sugestii de implementare:

ArticleController

Index – sa se modifice View-ul Index, astfel incat sa existe posibilitatea utilizarii helper-ului **@model.** Pentru realizarea acestui lucru se poate include un Partial View (**Vezi** Curs 7, sectiunea Partial View). De asemenea, acest View poate fi folosit si pentru afisarea unui singur articol (Show). Partialele se folosesc atunci cand dorim sa reutilizam codul. Etapele adaugarii unui astfel de View se gasesc in Cursul 7. Partialul se va numi "`ArticleInfo`" si va contine detaliile despre: titlul articolului, continutul acestuia, data si ora postarii si numele categoriei. In View-ul Index o sa fie o legatura catre pagina de afisare a unui singur articol, catre pagina de afisare a tuturor categoriilor si catre pagina de adaugare a unui nou articol.

Show – View-ul Show va utiliza acelasi Partial View folosit si in cazul afisarii tuturor articolelor. De asemenea, in View-ul Show o sa existe o legatura (link) catre pagina de editare a respectivului articol, un buton pentru stergerea articolului, o legatura cu pagina de afisare a tuturor articolelor si cu pagina de adaugare a unui nou articol.

New – in metoda New din Controller, cea cu HttpGet, se instantiaza un obiect de tipul modelului Article si se procedeaza la fel ca in laboratorul trecut pentru preluarea tuturor categoriilor din baza de date si folosirea lor in View-ul New, intr-un dropdown. In momentul adaugarii unui nou articol, cu ajutorul dropdown-ului, se poate selecta si o categorie.

New.cshtml – pentru adaugarea validarilor trebuie sa modificam modelele in baza de date (**Vezi** Curs 7, sectiunea Validare). Se vor adauga urmatoarele validari in **modelul Article**: **Required** pentru titlu, continut si categorie, folosind mesajele -> "Titlul este obligatoriu", "Continutul articolului este obligatoriu", "Categoria este obligatorie". Campul de tip **DateTime** nu necesita o astfel de validare deoarece data este automat inserata in baza de date in momentul postarii articolului in platforma. De asemenea, campul fiind de tipul DateTime, o sa accepte doar intrari de tipul DateTime. Pentru titlu sa se adauge si o validare de tipul `StringLength` care sa primeasca ca parametrii lungimea 100 si mesajul "Titlul nu poate avea mai mult de 100 de caractere".

Pentru **modelul Category** sa existe un validator de tipul **Required** cu mesajul "Numele categoriei este obligatoriu".

Pentru afisarea mesajelor de validare nu este suficienta scrierea acestora la nivel de Model. Mesajele trebuie preluate in View-ul corespunzator (pentru adaugare si editare) si afisate in interfata (**Vezi** Curs 7, cautand dupa **@Html.ValidationMessageFor**).

In View-ul **New** sa se utilizeze pentru formular (form) helper-ul

```
@using (Html.BeginForm(actionName:"NumeActiune",controllerName:"NumeController"))
```

Pentru afisarea tuturor mesajelor de validare (un sumar care sa contina toate erorile intervenite in timpul validarii) se foloseste helper-ul **Html.ValidationSummary** astfel:

```
@Html.ValidationSummary(false, "", new { @class = "text-danger" })
```

(**Vezi** Curs 7, cautand dupa **Html.ValidationSummary**).

Exemplu:

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

Al doilea parametru este un string care nu are valoare asociata deoarece acolo se preia valoarea (mesajul de validare) din Model.

/!\ OBSERVATIE:

Daca mesajele de validare apar de la inceput in pagina, inainte ca utilizatorul sa incerce sa completeze campurile existente, atunci se include urmatoarele secvente de cod in **Site.css**, din folderul **Content** (**Vezi** Curs 7, cautand dupa Site.css):

```
.field-validation-valid
{
	display: none;
}
.validation-summary-valid
{
	display: none;
}
```

Pentru metoda **New** cu **HttpPost** trebuie sa adaugam in aceasta metoda si verificarea starii modelului (**Vezi** Curs 7, cautand dupa **ModelState**). Prin intermediul variabilei **ModelState** putem verifica daca toate validarile au trecut cu succes si putem salva modificarile.

```
[HttpPost]
public ActionResult New(Article article)
{
    article.Categ = GetAllCategories();
    article.Date = DateTime.Now;

    try
    {
        if (ModelState.IsValid)
        {
            db.Articles.Add(article);
            db.SaveChanges();
            TempData["message"] = "Articolul a fost adaugat!";
            return RedirectToAction("Index");
        }
        else
        {
            return View(article);
        }
    }
    catch (Exception e)
    {
        return View(article);
    }
}
```

In momentul in care validarile nu trec cu succes, ramura de executie va fi cea din **else**. Astfel, in momentul in care returnam View-ul este necesar sa trimitem din nou modelul impreuna cu toate atributele sale.

In momentul in care validarile nu trec, se afiseaza acelasi View, in care utilizatorul trebuie sa completeze din nou datele articolului. Astfel, prin pasarea modelului **article** vom avea acces la toate proprietatile pe care utilizatorul le-a completat.

Metoda **Edit** cu **HttpGet**, se implementeaza la fel cum am procedat pana acum.

View-ul pentru **Edit**, trebuie sa cuprinda si in acest caz toate validarile necesare, la fel cum am procedat pentru View-ul New.

In metoda **Edit** cu **HttpPut** o sa implementam aceleasi modificari realizate si in cazul adaugarii unui nou articol. Trebuie sa utilizam **ModelState.IsValid** pentru a ne asigura ca toate validarile au trecut cu succes, **TryUpdateModel()** pentru pregatirea modelului in cazul modificarii si nu in ultimul rand obligatoriu trebuie sa incarcam toate categoriile si sa pastram de fiecare data toate modificarile realizate de utilizator.

```
[HttpPut]
public ActionResult Edit(int id, Article requestArticle)
{
    requestArticle.Categ = GetAllCategories();

    try
    {
        if (ModelState.IsValid)
        {
            Article article = db.Articles.Find(id);
            if (TryUpdateModel(article))
            {
                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");
        }
        else
        {
            return View(requestArticle);
        }

    }
    catch (Exception e)
    {
        return View(requestArticle);
    }
}
```

In metoda **Delete** din Controller procedam la fel, iar in View-ul Show adaugam si butonul de stergere (nefiind nevoie de un View special pentru Delete). Modificati formularul astfel incat sa utilizati helper-ul: `@using (Html.BeginForm()).` In momentul in care dorim sa preluam si id-ul (pentru a retine id-ul entitatii pe care dorim sa o stergem) putem utiliza doua metode:

- Prima metoda retine id-ul ca parametru in **BeginForm** astfel:

```
@using (Html.BeginForm(actionName: "Delete", controllerName: "Article",
method: FormMethod.Post, routeValues: new { id = @Model.Id })),
```

parametrul Method are valoarea default POST, deci putem omite scrierea lui.

- A doua metoda retine id-ul in pagina, iar in BeginForm nu o sa mai fie inclus:

```
@using (Html.BeginForm(actionName: "Delete", controllerName: "Article"))

{

      @Html.HiddenFor(m => m.Id)

}
```

Exemplu pentru **View-ul New:**

```
@model Laborator5App.Models.Article

<h2>Adaugare articol</h2>

@using (Html.BeginForm(actionName: "New", controllerName: "Articles"))
{
    @Html.ValidationSummary(false, "", new { @class = "text-danger" })

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

    @Html.ValidationMessageFor(m => m.Title, null, new { @class = "text-danger" })
    <br /><br />

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

    <label>Selectati categoria</label>
    @Html.DropDownListFor(m => m.CategoryId, new SelectList(Model.Categ, "Value",
"Text"),
    "Selectati categoria", new { @class = "form-control" })
    @Html.ValidationMessageFor(m => m.CategoryId, "", new { @class = "text-danger" })
    <br />
    <br />
    <button class="btn btn-sm btn-success" type="submit">Adauga articol</button>

}
```

Vezi Curs 7, sectiunea Layout View – in continuare sa se modifice layout-ul existent, astfel incat sa contina link-uri atat catre pagina de afisarea a tuturor articolelor, catre pagina de afisare a tuturor categoriilor si catre pagina de adaugare a unui nou articol.

De asemenea, se pot modifica clasele existente de css, adaugand stilizare in fisierul **Site.css.** De exemplu:

```
.navbar {
    background-color: #dadada;
    border: none;
    box-shadow: 0 6px 12px #dadada;
}


.navbar a {
    color: #2c2c2c;
}


.navbar a:hover {
    color: #5cb85c;
}
```