Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Laborator 8

EXERCIȚII:

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

Să se modifice implementarea, acolo unde este necesar, astfel încât să fie posibilă integrarea validărilor necesare. **Citiți cu atenție noțiunile din cadrul Cursului 8 și implementați exercițiile următoare.**

Sugestii de implementare:

1. Să se modifice Modelele (Article, Category, Comment) adăugându-se validările necesare la nivel de Model (**VEZI Curs 8 – Secțiunea Atribute de validare la nivel de Model**), astfel:

- Se adaugă asupra modelului **Article** următoarele validări:
  - **Titlul** articolului este obligatoriu (Required), poate avea o lungime maximă de 100 de caractere (StringLength) și nu poate avea mai puțin de 5 caractere (MinLength);
  - **Conținutul** articolului este obligatoriu (Required);
  - **Categoria** din care face parte articolul este obligatorie (Required);
- Se adaugă asupra modelului **Category** următoarea validare:
  - **Numele** categoriei este obligatoriu (Required);

- Se adaugă asupra modelului **Comment** următoarea validare:

  - **Conținutul** comentariului este obligatoriu (Required);

După adăugarea tuturor validărilor la nivelul atributelor modelelor se rulează aplicația. Se observă, atunci când se încearcă adăugarea unui articol necompletat, că mesajele de validare adăugate anterior nu se afișează în browser. Acestea se vor afișa doar în momentul în care vor fi preluate în View.

2. Preluați validările în View-urile asociate (**VEZI Curs 8 – Secțiunea Preluarea validărilor în View**), urmând pașii următori. De asemenea, după adăugarea validărilor în View studiați cu atenție comportamentul, analizând rezultatele din output (browser).

   Citiți cu atenție explicațiile din CURS 8!

   - Se preiau validările pentru entitatea Article, View-urile ***New*** și ***Edit*** (adăugarea unui nou articol și editarea unui articol existent)

   - Validările se includ în View cu ajutorul atributului `asp-validation-for`

   - După preluarea mesajelor de validare în View, afișați un sumar cu toate erorile apărute în timpul validării (**VEZI Curs 8 – Secțiunea Atributul asp-validation-summary**). Sumarul se va implementa atât în View-ul New, cât și în Edit

3. Implementați validările la nivel de Controller, astfel încât acestea să se afișeze corect, în funcție de acțiunea pe care o face utilizatorul final (**VEZI Curs 8 – Secțiunea Implementarea validărilor la nivel de Controller**). De asemenea, parcurgeți și explicațiile de mai jos.

Explicații privind adăugarea validărilor la nivel de Controller:

În metoda **New** cu **HttpPost** trebuie inclusă verificarea stării modelului. Prin intermediul variabilei **ModelState** se verifică dacă toate validările au trecut cu succes și se pot salva modificările în baza de date.

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

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

În momentul în care validările nu trec cu succes, ramura de execuție va fi cea din **else**. Astfel, în momentul în care returnăm View-ul este necesar să trimitem din nou modelul împreună cu toate atributele sale.

- **La fel se procedeaza si pentru editare**

```
[HttpPost]
public IActionResult Edit(int id, Article requestArticle)
{
      Article article = db.Articles.Find(id);

      if(ModelState.IsValid)
      {
            article.Title = requestArticle.Title;
            article.Content = requestArticle.Content;
            article.CategoryId = requestArticle.CategoryId;
            TempData["message"] = "Articolul a fost modificat";
            db.SaveChanges();
            return RedirectToAction("Index");
      }
      else
      {
            requestArticle.Categ = GetAllCategories();

             return View(requestArticle);
      }
}
```

4. Pentru o mentenanță mai buna a codului și pentru a elimina redundanța, utilizați un Partial View pentru afișarea informațiilor asociate unui articol.

   În cadrul aplicației, afișarea unui articol conține același cod, atât în cazul afișării tuturor articolelor din baza de date → View-ul Index, cât și în cazul afișării unui singur articol → View-ul Show. În acest caz, pentru a elimina redundanța, și anume scrierea repetitivă a aceluiași cod în cadrul ambelor View-uri, să se utilizeze un View Partial, numit ***ArticleInfo*** (**VEZI Curs 8 – Secțiunea View-uri partajate -> Partial View**).

5. Să se modifice Layout-ul existent, astfel încât să conțină link-uri către pagina de afișare a tuturor articolelor, către pagina de afișare a tuturor categoriilor și către pagina de adăugare a unui nou articol. O să se modifice Layout-ul existent, numit ***_Layout.cshtml***, aflat în folderul Views → folderul Shared. În cadrul acestui Layout o să se adauge cele trei link-uri. De asemenea, o să se modifice link-ul existent în Layout, cel care la apăsarea logo-ului duce la ***/Home/Index***. După modificare, acesta trebuie să redirecționeze către ***/Articles/Index***.

Pentru stilizare se pot modifica clasele existente de css, adăugând următoarele secvențe de cod în **wwwroot → css → site.css**

Sugestii de stilizare (pentru paleta de culori se poate utiliza: https://flatuicolors.com/)

```
.navbar {
    background-color: #dfe6e9 !important;
    border: none !important;
    box-shadow: 0 3px 12px #636e72 !important;
}

a.nav-link {
    color: #2c2c2c !important;
}
a.nav-link:hover {
    color: #00b894 !important;
}
```

```
.logoutbtn {
    color: #2c2c2c !important;
}

.logoutbtn:hover {
    color: #00b894 !important;
}
```

6. Să se adauge validările și pentru entitățile **Category** și **Comment**.

- **În Model** – Denumirea categoriei este obligatorie și Conținutul comentariului este obligatoriu;
- **În View** – să se utilizeze atributulele de validare specifice;
- **În Controller** – să se verifice starea modelului (se verifică dacă validările au trecut cu succes);

/!\ OBSERVAȚIE

În cazul adăugării unui comentariu, formularul de adăugare se afla în **View-ul Show** asociat **Controller-ului Articles**. Se procedează în acest mod deoarece un comentariu este asociat unui articol și este necesară afișarea acestuia împreună cu articolul de care depinde.

Așadar, în pagina Show a articolului, se afișează articolul împreună cu toate comentariile sale, dar și un formular în care utilizatorul poate adăuga un nou comentariu articolului respectiv. Implementarea formularului este următoarea:

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

    <div class="card-body">

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

        <label>Continut comentariu</label>
        <br />

        <textarea class="form-control" name="Content"></textarea>

        <span asp-validation-for="Content" class="text-danger"></span>
        <br /><br />

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

    </div>

</form>
```

În cadrul formularului a fost adăugată validarea cu ajutorul atributului de validare specific.

De asemenea, se poate observa și inputul de tip ***hidden***, cu ajutorul căruia reținem id-ul articolului căruia îi corespunde comentariul, conform implementării modelului (se observă cheia externă ArticleId):

```
public class Comment
    {
        [Key]
        public int Id { get; set; }

        [Required(ErrorMessage = "Continutul comentariului este
obligatoriu")]
        public string Content { get; set; }

        public DateTime Date { get; set; }

        public int? ArticleId { get; set; }

        public virtual Article? Article { get; set; }
    }
```

Având în vedere că un View nu poate avea mai mult de un Model inclus, nu se pot utiliza Tag Helpers pentru restul componentelor de HTML (input, label, textarea). În acest caz, se pot utiliza în continuare tag-urile de HTML.

Deoarece se dorește ca, în momentul în care se încearcă adăugarea unui comentariu fără conținut, utilizatorul să primească mesajul de validare "Comentariul este obligatoriu", dar este necesar și ca pe ramura else să se retrimită întregul articol împreună cu toate comentariile pe care le are, se va proceda astfel:

- Se mută metoda New cu HttpPost din CommentsController în ArticlesController

- În ArticlesController, metoda se va numi Show, deoarece scopul său este să redirecționeze către această metodă atunci când adăugarea unui comentariu nu este posibilă. Acest lucru va permite afișarea corectă a articolului împreună cu toate comentariile sale. Această abordare este folosită pentru a păstra obiectul articolului, **Article**, care trebuie transmis ca argument în View pentru metoda Show. Astfel, se va afișa atât articolul cu toate comentariile existente, cât și un formular pentru adăugarea unui nou comentariu. Deoarece în ArticlesController există deja o metodă cu numele Show, dedicată afișării articolelor, va fi necesar să utilizăm un verb HTTP diferit pentru a distinge între afișarea și adăugarea comentariilor. Adăugarea unui comentariu este o acțiune de scriere și, prin urmare, va folosi verbul HTTP HttpPost. Aceasta abordare permite utilizarea aceluiași nume de metodă, Show, pentru operațiuni diferite în cadrul aceluiași Controller.

- Pentru metoda Show din ArticlesController, modelul de bază este Article. Când se încearcă adăugarea unui comentariu, metoda extrage și id-ul articolului, care reprezintă id-ul modelului principal afișat în View. Pentru a colecta eficient doar datele specifice comentariului și pentru a le insera corect în baza de date, în metoda Show se folosește atributul **[FromForm]**. Acest atribut asigură că datele sunt preluate exclusiv din formular, facilitând astfel gestionarea corectă și separată a datelor comentariului.

```
public IActionResult Show([FromForm] Comment comment)

        [HttpPost]
        public IActionResult Show([FromForm] Comment comment)
        {
            comment.Date = DateTime.Now;

            if (ModelState.IsValid)
            {
                db.Comments.Add(comment);
                db.SaveChanges();
                return Redirect("/Articles/Show/" +
comment.ArticleId);
            }

            else
            {
               Article art =
     db.Articles.Include("Category").Include("Comments")
                                .Where(art => art.Id ==
comment.ArticleId)
                                .First();

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

                return View(art);
            }
        }
```

Dacă sursa implicită care trimite datele nu trece corect argumentele la nivel de Controller într-o aplicație ASP.NET Core, se pot utiliza următoarele atribute pentru a specifica explicit sursa de unde trebuie preluate valorile:

- **[FromBody]** – se utilizează acest atribut pentru a arăta că valorile parametrului sunt extrase din corpul cererii HTTP. Este frecvent utilizat pentru a prelua obiecte JSON din cererile POST.

- **[FromForm]** – se utilizează pentru a indica faptul că valorile parametrului sunt preluate din datele formularului, transmise printr-o cerere POST. Este ideal pentru preluarea datelor trimise prin formulare HTML.

- **[FromQuery]** – se utilizează pentru a specifica faptul că valorile ar trebui să fie preluate din parametrii query string ai URL-ului cererii. Acesta este util pentru cereri GET în care datele sunt incluse în URL.

- **[FromRoute]** - acest atribut specifică că valorile parametrului trebuie extrase din parametrii de rută ai URL-ului cererii, care sunt specificați în modelul de rutare al URL-ului.

- **[FromHeader]** – se utilizează pentru a specifica faptul că valorile ar trebui să fie preluate din antetul cererii HTTP. Acesta este util pentru preluarea datelor precum tokenuri de autentificare sau alte valori specifice setate în anteturi.

- **[FromServices]** - acest atribut este utilizat pentru a injecta servicii direct în metodele acțiunii Controller-ului. Permite accesarea serviciilor configurate în containerul de dependențe.