Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC
Laborator 4

EXERCITII:

Exercitiul 1:

Sa se creeze un nou proiect si sa se adauge Controller-ul **ArticlesController** in care se vor implementa urmatoarele (**VEZI** Curs 4 – capitolul Controller):

1. Sa se adauge un Model numit **Article** care sa contina **Id**, **Title**, **Content** si **Date** astfel: Models -> click dreapta -> Add -> Class -> Adaugam o clasa Article.cs

```
public class Article
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }
        public DateTime Date { get; set; }
    }
```

2. Sa se adauge in Controller o metoda **NonAction**, numita `GetArticles()` (**VEZI** Curs 4 – in sectiunea **Selectori**) care va returna un array de obiecte de tip Article, array pe care o sa il folosim pentru afisare (procedam astfel deoarece in acest laborator nu o sa folosim baza de date, iar in acest mod ne cream articole pe care le putem prelucra).

(*) Cum se poate implementa o metoda NonAction? Ce reprezinta o metoda NonAction?

```
public Article[] GetArticles()
 {
     // Se instantiaza un array de articole
     Article[] articles = new Article[3];

     // Se creeaza articolele
     for (int i = 0; i < 3; i++)
     {
         Article article = new Article();
         article.Id = i;

         article.Title = "Articol " + (i + 1).ToString();
         article.Content = "Continut articol " + (i + 1).ToString();
         article.Date = DateTime.Now;

         // Se adauga articolul in array
         articles[i] = article;
     }
     return articles;
 }
```

3. Sa se adauge toate metodele pentru operatiile de tip C.R.U.D.

   - **Index** – pentru listarea tuturor articolelor
   - **Show** – pentru vizualizarea unui articol in functie de Id
   - **New** – pentru crearea unui nou articol
   - **Edit** – pentru editarea unui articol existent
   - **Delete** – pentru stergerea unui articol

De asemenea, se vor adauga verbele HTTP potrivite pentru fiecare metoda in parte (**VEZI** Curs 4 – sectiunea Selectori).

4. **Index**

- Sa se creeze metoda Index in Controller-ul ArticlesController

- Sa se creeze un **View** numit **Index**

In folderul View → se adauga un folder nou corespunzator Controller-ului Articles astfel → Click dreapta pe folderul Views → Add → New Folder → Folderul o sa se numeasca Articles. In acest folder se creeaza un fisier .cshtml asociat metodei Index → Click dreapta pe folderul Articles → Add → View → Se selecteaza optiunea Razor View Empty → Add → Se selecteaza ASP.NET Core si Razor View – Empty → Se modifica denumirea in Index → Add

- In continuare se utilizeaza array-ul de articole ucces, astfel incat sa preluam in metoda Index toate articolele, dupa care sa le trimitem catre View-ul asociat pentru afisare catre utilizatorul final

Metoda Index din ArticlesController:

```
public IactionResult Index()
   {
       Article[] articles = GetArticles();

       // Se adauga array-ul de articole in View
       ViewBag.Articles = articles;

       return View();

   }
```

Index.cshtml – in View-ul Articles

Sa se implementeze afisarea articolelor. De asemenea, pentru fiecare articol trebuie sa existe posibilitatea afisarii si editarii acestuia.

5. Modificati ruta **Default** din Program.cs, astfel incat la rularea proiectului sa ne redirectioneze de fiecare data catre pagina de listare a tuturor articolelor.

6. **Show**

   - In metoda Show din ArticlesController afisati detaliile unui articol, in functie de id-ul acestuia. In cazul in care articolul nu este gasit, afisati un View de eroare cu explicatiile aferente (mesajul de eroare aruncat de exceptie si un mesaj – “Articolul cautat nu poate fi gasit”).

   - Sa se creeze un View numit Show, in care sa se implementeze afisarea unui singur articol. View-ul va contine si un link catre pagina de afisare a tuturor articolelor.

   - Configurati o ruta pentru ruta existenta **/Articles/Show/{id}**, care dupa cum se oberva se poate accesa prin intermediul rutei Default, astfel incat sa se acceseze prin ruta **/articole/show/{id}**. Ruta o sa se numeasca **ArticlesShow**

ArticlesController → Show

```
// Afisarea unui singur articol
public IactionResult Show(int? id)
{
    Article[] articles = GetArticles();

    try
    {
        ViewBag.Article = articles[(int)id];
        return View();
    }

    catch (Exception e)
    {
        ViewBag.ErrorMessage = e.Message;
        return View(“Error”);
    }
```

```
    }
```

View → Show.cshtml

```
@{
    ViewBag.PageName = “Show”;
}

<h2>@ViewBag.PageName</h2>

<hr />

<h1>@ViewBag.Article.Title</h1>
<p>@ViewBag.Article.Content</p>
<small>@ViewBag.Article.Date</small>

<hr />
<br />

<a href=”/Articles/Index”>Afisare articole</a>
```

View → Error.cshtml

```
@{
    ViewBag.Msg = “Articolul cautat nu poate fi gasit!”;
}

<h2>@ViewBag.Msg</h2>
<br />
<p>Error message: @ViewBag.ErrorMessage</p>
```

Index.cshtml → ruta configurata

```
<a href=”/articole/show/@article.Id”>Afisare articol</a>
// Ruta configurata
```

7. **New**

- In Controller o sa existe doua metode numite **New**.
  Prima metoda este de tip [HttpGet] si se utilizeaza pentru afisarea formularului prin intermediul caruia se completeaza datele unui articol.
  A doua metoda este utilizata pentru trimiterea datelor corespunzatoare articolului catre server, pentru adaugarea noului articol in baza de date. Pentru trimiterea datelor intotdeauna se foloseste verbul [HttpPost].

- O sa existe, de asemenea, doua View-uri.
  Un View pentru afisarea formularului de creare a unui articol.
  Al doilea View (caruia ii dam un nume diferit – de exemplu: NewPostMethod) folosit in momentul in care datele pentru creare vor fi trimise catre server. In acest caz, al doilea View o sa afiseze doar un mesaj deoarece nu avem inca ucces la o baza de date.

- Link-ul catre pagina de creare a unui articol o sa fie adaugat in Index.

ArticlesController → New ([HttpGet])

```
// GET: Afisarea formularului de creare a unui articol

[HttpGet]
public IactionResult New()
{
    return View();
}
```

ArticlesController → New ([HttpPost])

```
        // POST: Trimiterea datelor despre articol catre server
pentru adaugare in baza de date

        [HttpPost]
        public IactionResult New(Article article)
        {
            // … cod creare articol …

            return View(“NewPostMethod”);
        }
```

View → New.cshtml

```
<h2>Afisare formular de adaugare articol</h2>

<form method=”post” action=”/Articles/New”>

    <button type=”submit”>Adauga articol</button>

</form>
```

View →NewPostMethod.cshtml

```
@{
    ViewBag.New = “Articolul a fost adaugat cu  uccess!”;
}

<h2>
    @ViewBag.New
</h2>
```

8. Edit

- In Controller o sa existe doua metode numite **Edit**.
  Prima metoda este de tip [HttpGet] si se utilizeaza pentru afisarea formularului care o sa contina datele unui articol existent in baza de date. Formularul se afiseaza cu scopul modificarii campurilor (o parte din ele sau chiar toate).
  A doua metoda este utilizata pentru trimiterea datelor corespunzatoare articolului catre server, pentru editarea articolului in baza de date. Pentru trimiterea datelor intotdeauna se foloseste verbul [HttpPost]. In cazul API-urilor se utilizeaza [HttpPut], dar in ASP.NET Core se poate utiliza [HttpPost] atat pentru New, cat si pentru Edit si Delete). A doua metoda poate returna un View, la fel ca in exemplul anterior sau se poate redirectiona catre pagina principala a aplicatiei, pagina in care se afiseaza lista tuturor articolelor.

- Si in acest caz o sa existe doua View-uri.
  Un View pentru afisarea articolului din baza de date, articol care urmeaza sa fie editat.
  Al doilea View (caruia ii dam un nume diferit – de exemplu: EditMethod) este folosit in momentul in care datele pentru editare vor fi trimise catre server. In acest caz, al doilea View o sa afiseze doar un mesaj deoarece nu avem inca ucces la o baza de date.

ArticlesController → Edit ([HttpGet])

```
// GET: Afisarea datelor unui articol pentru editare

[HttpGet]
public IactionResult Edit(int? id)
{
    ViewBag.Id = id;
    return View();
}
```

ArticlesController → Edit ([HttpPost])

```
        // POST: Trimiterea modificarilor facute catre server
pentru stocare in baza de date

        [HttpPost]
        public IactionResult Edit(Article article)
        {
            // … cod adaugare articol editat in baza de date

            //return Redirect(“/Articles/Index”);

            return View(“EditMethod”);
        }
```

View → Edit.cshtml

```
<br />

<p>Afisare formular de editare articol – in acest formular
de preiau datele curente ale articolului</p>

<form method=”post” action=”/Articles/Edit/@ViewBag.Id”>

    <button type=”submit”>Editeaza articol</button>

</form>
```

View → EditMethod.cshtml

```
@{
    ViewBag.New = “Articolul a fost editat cu  uccess!”;
 }

<h2> @ViewBag.New </h2>
```

9. **Delete**

- In pagina Show inserati formularul corespunzator stergerii intrarii prin **metoda DELETE**, folosind un buton la fel ca in exemplele anterioare. Verbul folosit este [HttpPost]. In cazul implementarii unui API, verbul HTTP o sa fie [HttpDelete]. In ASP.NET Core se poate utiliza [HttpPost] atat pentru New, cat si pentru Edit si Delete.

ArticlesController → Delete ([HttpPost])

```
// POST Stergere articol din baza de date

[HttpPost]
public IactionResult Delete(int? id)
{
    // … cod stergere articol din baza de date

    return Content("Articolul a fost sters din baza de date!");
}
```

View → Show.cshtml

```
<form method="post" action = "/Articles/Delete/@ViewBag.Article.Id">
    <button type="submit">Stergere articol</button>
</form>
```

10. Redenumiti metoda Index din Controller in ***listare***, folosind selectori (**VEZI** Curs 4 – sectiunea Selectori). **Ce observati dupa redenumire? Ce modificari trebuie realizate?**

TEMA:

/!\ Finalizati toate exercitiile din acest laborator.