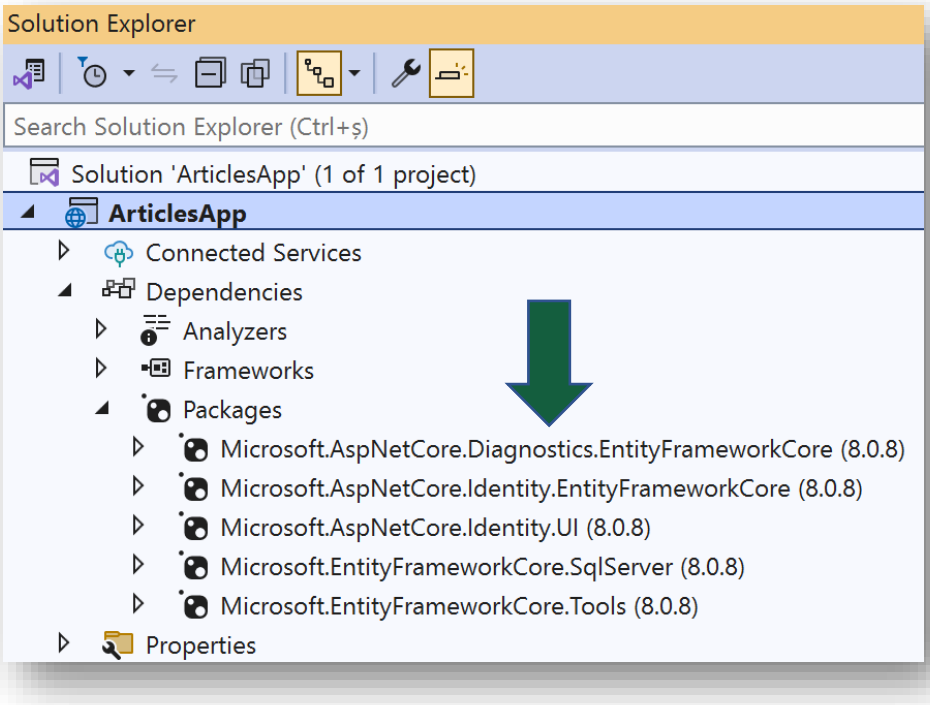
Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Laborator 6

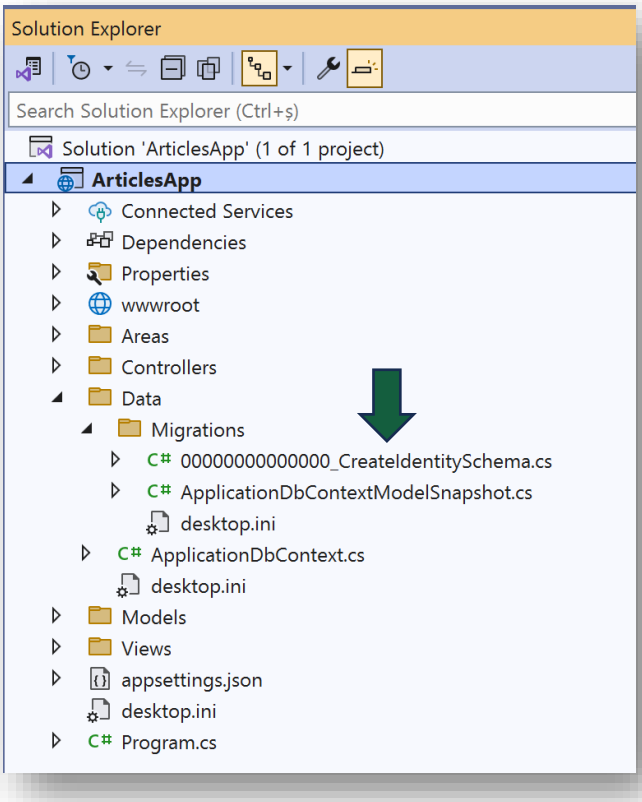
EXERCIȚII:

1. Să se creeze un nou proiect numit **ArticlesApp** de tipul ASP.NET Core Web App (Model-View-Controller). Proiectul o să conțină sistem de autentificare (**VEZI Curs 6 – Secțiunea Adăugarea Sistemului de Autentificare**).

2. Se verifică existența următoarelor pachete:



3. Se rulează migrația ***Update-Database*** pentru realizarea update-ului în baza de date. După acest pas, se pot vedea tabelele în SQL Server Object Explorer. Se rulează doar comanda Update-Database deoarece migrația inițială există (VEZI imaginea de mai jos).



4. Se rulează proiectul și se înregistrează un cont (**VEZI Curs 6**), după care se poate vizualiza noul user în baza de date, tabelul **dbo.AspNetUsers**.

5. Se consideră entitățile **Article**, **Category** și **Comment** cu proprietățile de mai jos. De asemenea, se consideră cerințele din cadrul **Cursului 6 – secțiunea Exemplu practic pentru proiectarea Diagramei E/R.**

Article

- Id (int – primary key)
- Title (string – titlul este obligatoriu)
- Content (string – conținutul este obligatoriu)
- Date (DateTime)
- CategoryId (int – cheie externă – categoria din care face parte articolul)

Category:

- Id (int – primary key)
- CategoryName (string – numele este obligatoriu)

Comment:

- Id (int – primary key)
- Content (string – conținutul comentariului este obligatoriu)
- Date (DateTime – data la care a fost postat comentariul)
- ArticleId (int – cheie externă – articolul căruia ii aparține comentariul)

Cerințe aplicație:

- Să existe cel putin 4 tipuri de utilizatori: vizitator neînregistrat, utilizator înregistrat, editor și administrator;
- Orice utilizator poate vizualiza știrile apărute pe site. Pe pagina principală vor aparea știrile cele mai recente, în funcție de data la care au post postate. În acest caz, se alege un număr de x știri pentru afișare;
- Știrile vor fi împărțite pe categorii (create dinamic de către administrator): știință, tehnologie, sport, etc, existând posibilitatea de adăugare a noi categorii (administratorul poate face CRUD pe categorii);
- Știrile o să se afișeze paginat, alegându-se un număr x de știri pe o pagină;
- O să se includă și un editor de text, astfel încât să se editeze textul în momentul în care un editor publică o nouă știre. Știrea se poate scrie folosind și elemente de markup;
- Editorii se ocupă de publicarea știrilor noi și pot vizualiza, edita, șterge propriile știri;
- Utilizatorii care au cont pot adăuga comentarii la știrile apărute, își pot șterge și edita propriile comentarii;
- Utilizatorii care nu au cont pot să vadă doar pagina principală a aplicației. Dacă doresc să citească știri și să interacționeze cu alți utilizatori, aceștia vor fi redirecționați către pagina de înregistrare/autentificare;

- Știrile pot fi căutate prin intermediul unui motor de căutare propriu, în funcție de titlu, conținut, sau chiar conținutul comentariilor;
- Administratorii se ocupă de buna funcționare a întregii aplicații (ex: pot face CRUD pe știri, pe categorii, pe utilizatori etc.) și pot activa sau revoca drepturile utilizatorilor și editorilor;

Să se implementeze cele trei clase în Models, după care să se ruleze migrațiile → în acest caz ***Add-Migration NumeMigratie*** și ***Update-Database*** (**VEZI Pasul următor – exercițiul 7**).

Implementarea claselor:

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

   [Required(ErrorMessage = "Titlul este obligatoriu")]
   public string Title { get; set; }

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

   public DateTime Date { get; set; }

   [Required(ErrorMessage = "Categoria este obligatorie")]
   public int CategoryId { get; set; }

   public virtual Category Category { get; set; }

   public virtual ICollection<Comment> Comments { get; set; }
}
```

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

   [Required(ErrorMessage = "Numele categoriei este
obligatoriu")]
   public string CategoryName { get; set; }

   public virtual ICollection<Article> Articles { get; set; }
}



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

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

   public DateTime Date { get; set; }

   public int ArticleId { get; set; }

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

6. Se adaugă proprietățile în contextul bazei de date, pentru realizarea ulterioară a migrațiilor.

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

7. Să se ruleze sistemul de migrații, după care să se insereze manual, în fiecare tabel, 2-3 intrări (în acest caz se rulează ambele comenzi).

8. Să se adauge câte un Controller pentru fiecare clasă → **ArticlesController**, **CategoriesController** și **CommentsController**, în care se vor implementa operațiile CRUD asupra entităților.

9. Să se adauge câte un folder în ***Views*** pentru fiecare Controller.

10. Implementați operațiile CRUD asupra entităților, urmând pașii următori:

- Să existe posibilitatea realizării operațiilor **C.R.U.D. pentru entitatea Article** astfel:

    - **Index** – afișarea tuturor articolelor, împreună cu denumirea categoriei din care fac parte – se poate utiliza din Bootstrap ->

      Cards: https://getbootstrap.com/docs/5.2/components/card/#about

      Bootstrap Icons: https://blog.getbootstrap.com/2021/01/07/bootstrap-icons-1-3-0/

Exemplu pentru View:

```
<h2 class="text-center">Afisare articole</h2>
<br />

<div class="d-flex justify-content-center">

      <a class="btn btn-outline-success" href=" Configurare link ">Afisare
categorii</a>
      <br />
      <a class="btn btn-outline-success" href=" Configurare link ">Adauga
articol</a>
      <br />

</div>

<br />

@foreach (???)

{

      <div class="card">


            <div class="card-body">


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

                  <div class="card-text">Se afiseaza continutul articolului
      </div>

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

```
                    <div><i class="bi bi-globe"></i> Se afiseaza denumirea
categoriei </div>

                    <a class="btn btn-success" href=" Link-ul catre
afisarea unui singur articol">Afisare articol</a>

                    <span class="alert-success px-1 align-content-center">
Se afiseaza data la care a fost postat articolul articolului </span>


                </div>

            </div>

        </div>

        <br />
        <br />

}
```

- **Show** – afișarea unui singur articol (într-o pagină separată) împreună cu denumirea categoriei din care face parte articolul respectiv

- **New** – posibilitatea adăugării unui nou articol. În momentul în care se adaugă articolul, categoria se va selecta dintr-o lista existentă de categorii, folosind un element de tipul dropdown

Exemplu dropdown în View:

```
<select name="id-ul categoriei din baza de date - model
binding">

    @foreach (var // se parcurg categoriile)
    {
        <option value="cheie">valoare</option>
    }

</select>
```

- **Edit** – posibilitatea editării unui articol. În momentul în care se editează articolul, categoria se va selecta dintr-o lista existentă de categorii (element de tipul dropdown)

Exemplu dropdown în View:

```
<select name="">

    @foreach (var ... in ...)
    {
        if (//daca este categoria curenta a articolului)
        {
            <option selected="selected" value="id-ul
categoriei">denumirea categoriei</option>
        }
        else
        {
            <option value="id-ul categoriei">
                denumirea categoriei
            </option>
        }
    }

</select>
```

- **Delete** – posibilitatea ștergerii unui articol

- Să existe posibilitatea realizării operațiilor **C.R.U.D. pentru entitatea Category** (Atenție! În momentul în care se dorește ștergerea unei categorii, automat se vor șterge și toate articolele care fac parte din categoria respectivă).

Aveți exemple de implementare în cursul și laboratorul 5.

11. Prompt Engineering în estimarea duratelor și împărțirea taskurilor Agile

Formulați un prompt clar, specific și bine structurat pentru un agent AI (ex: ChatGPT, Cursor, Claude, Gemini, Copilor, etc.) care să vă ajute **să împarți eficient taskurile într-o echipă Agile** și **să estimați durata fiecărui sprint.**

Imaginează-ți că ești Scrum Master sau Product Owner într-o echipă care lucrează la dezvoltarea unei aplicații web. Trebuie să pregătești planul pentru următorul sprint, dar vrei să folosești inteligența artificială ca **asistent de planificare**.

AI-ul te poate ajuta să estimezi durata fiecărui task, să sugereze împărțirea lor în funcție de competențele echipei și să identifice potențiale blocaje.

Temă:

- Să existe posibilitatea realizării operațiilor **C.R.U.D.** pentru entitatea **Comment**, astfel:
    - **Afișarea tuturor comentariilor** corespunzătoare unui articol. Fiecare articol o să aibă un buton “Afisare articol” (Show) care redirecționează către o pagină în care vom avea articolul împreună cu toate comentariile corespunzătoare articolului respectiv;
    - **New** – posibilitatea adăugării unui nou comentariu;
    - **Edit** – posibilitatea editării unui comentariu existent;
    - **Delete** – posibilitatea ștergerii unui comentariu;

Surse utile:

Bootstrap – v5.2 → https://getbootstrap.com/docs/5.2/getting-started/introduction/

Bootstrap – icons → https://icons.getbootstrap.com/

Bootstrap – buttons → https://getbootstrap.com/docs/5.2/components/buttons/

Bootstrap – spacing → https://getbootstrap.com/docs/5.0/utilities/spacing/

Flexbox → https://css-tricks.com/snippets/css/a-guide-to-flexbox/