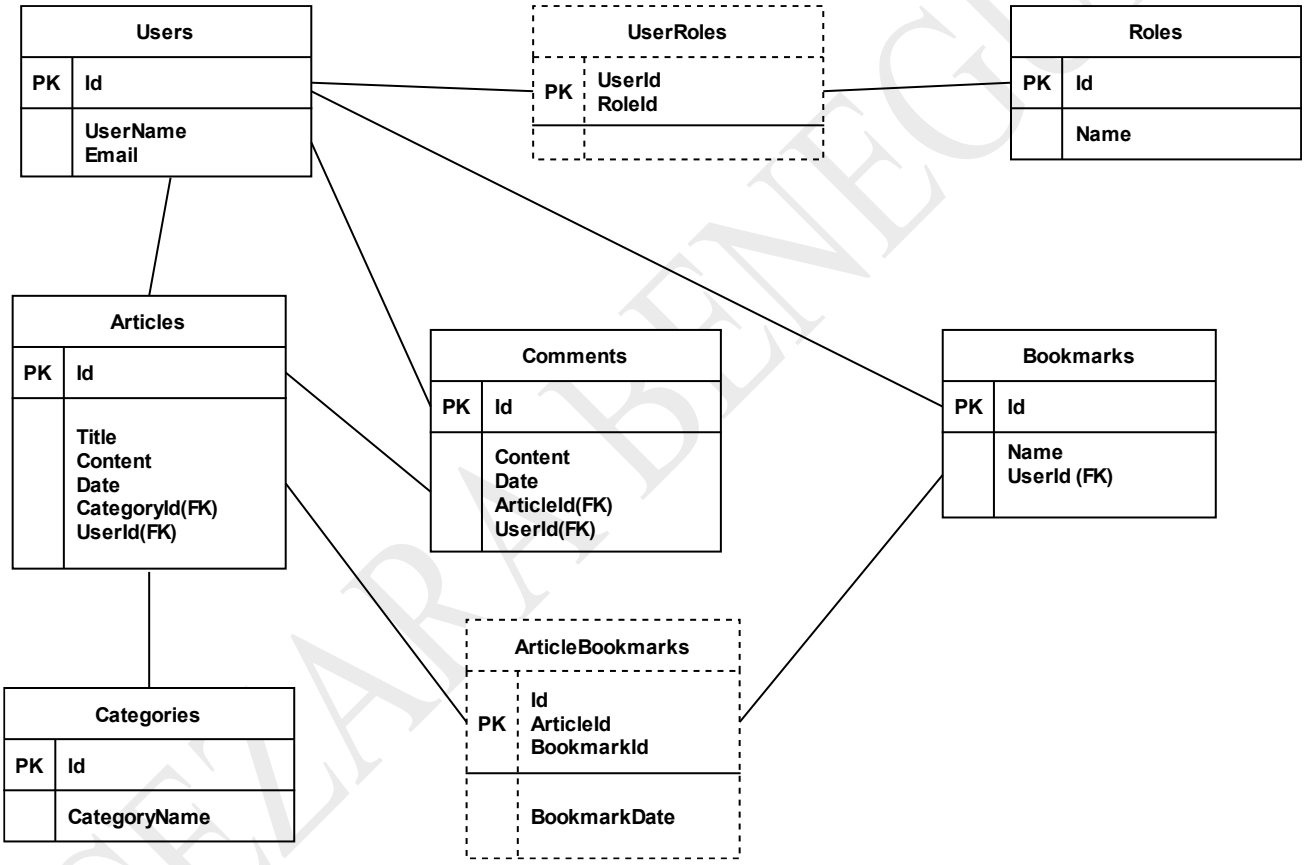
Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC
Curs 10

Cuprins

Implementarea relatiei many-to-many

Descrierea diagramei

Pentru implementarea si exemplificarea relatiei many-to-many vom utiliza urmatoarea diagrama conceptuala. Se va extinde diagrama proiectata in cursurile anterioare, prin adaugarea unui nou tabel → **Bookmark**



Relatia many-to-many se afla intre **Article** si **Bookmark**, avand urmatoarele specificatii:

- Un utilizator isi poate crea propriile colectii (Bookmarks);
- In momentul in care un utilizator isi adauga colectii, acesta o sa aiba acces doar la colectiile pe care le-a creat, existand posibilitatea editarii si a stergerii lor;
- Atat utilizatorii cu rolurile User sau Editor, cat si utilizatorii cu rolul Admin, pot crea propriile colectii. Utilizatorii cu rolul User sau Editor o sa aiba acces doar la colectiile create de ei, iar utilizatorii cu rolul Admin o sa aiba acces la toate colectiile existente in platforma;
- Dupa crearea colectiilor, utilizatorii isi pot adauga articole in colectii. Articolele pe care le pot adauga trebuie sa existe in platforma. Ei doar selecteaza un articol si il adauga intr-o colectie. Utilizatorii nu pot vedea colectiile altor utilizatori. Fiecare utilizator are acces doar la colectiile sale;
- Un articol poate face parte din mai multe colectii, iar o colectie poate contine mai multe articole;

Se considera urmatoarele clase: **Bookmark**, **ArticleBookmark (tabelul asociativ)** cu urmatoarele proprietati:

Bookmark:

- **Id** – int → id-ul colectiei (cheie primara)
- **Name** – string → denumirea colectiei, fiind o proprietate obligatorie (Required
- **UserId** – string → cheie externa – reprezinta utilizatorul care a creat colectia

ArticleBookmark:

- **Id**, **ArticleId**, **BookmarkId** – int → cheia primara compusa
- **Id** – valoare unica, avand auto-increment
- **BookmarkDate** – DateTime → Data si ora la care a fost adaugat un articol in cadrul unei colectii

Implementarea claselor

Pentru implementarea claselor se procedeaza astfel:

Bookmark.cs

```
public class Bookmark

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

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

   public string? UserId { get; set; }

   public virtual ApplicationUser? User { get; set; }

   public virtual ICollection<ArticleBookmark>? ArticleBookmarks
{ get; set; }

}
```

Article.cs

```
public class Article

{
   …

   public virtual ICollection<ArticleBookmark>? ArticleBookmarks
{ get; set; }

}
```

ArticleBookmark.cs

```
public class ArticleBookmark

{
   [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public int Id { get; set; }

   public int? ArticleId { get; set; }

   public int? BookmarkId { get; set; }

   public virtual Article? Article { get; set; }

   public virtual Bookmark? Bookmark { get; set; }

   public DateTime BookmarkDate { get; set; }

}
```

ApplicationUser.cs

```
public class ApplicationUser : IdentityUser

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

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

   public virtual ICollection<Bookmark>? Bookmarks { get; set; }

}
```

ApplicationDbContext.cs

```
      public class ApplicationDbContext :
IdentityDbContext<ApplicationUser>

      {
        public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }



        public DbSet<ApplicationUser> ApplicationUsers { get; set; }

        public DbSet<Article> Articles { get; set; }

        public DbSet<Category> Categories { get; set; }

        public DbSet<Comment> Comments { get; set; }

        public DbSet<Bookmark> Bookmarks { get; set; }

        public DbSet<ArticleBookmark> ArticleBookmarks { get; set; }



        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // definire primary key compus

            modelBuilder.Entity<ArticleBookmark>()
                .HasKey(ab => new { ab.Id, ab.ArticleId, ab.BookmarkId
});

            // definire relatii cu modelele Bookmark si Article (FK)

            modelBuilder.Entity<ArticleBookmark>()
                .HasOne(ab => ab.Article)
                .WithMany (ab => ab.ArticleBookmarks)
                .HasForeignKey(ab => ab.ArticleId);

            modelBuilder.Entity<ArticleBookmark>()
                .HasOne(ab => ab.Bookmark)
                .WithMany(ab => ab.ArticleBookmarks)
                .HasForeignKey(ab => ab.BookmarkId);
        }
      }
```

Se pot adauga si proprietati suplimentare in clasa ApplicationUser, extinzand astfel clasa. In exemplul urmator se adauga doua atribute → **FirstName** si **LastName**.

ApplicationUser.cs

```
public class ApplicationUser : IdentityUser
{
   public virtual ICollection<Comment>? Comments { get; set; }

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

   public virtual ICollection<Bookmark>? Bookmarks { get; set; }


   public string? FirstName { get; set; }

   public string? LastName { get; set; }

   [NotMapped]
   public IEnumerable<SelectListItem>? AllRoles { get; set; }

}
```

/!\ OBSERVATIE

Dupa modificarea claselor se executa migratiile.

Add-Migration NumeMigratie
Update-Database

Stocarea fisierelor in baza de date

```
public class Article
{

    [Key]
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime Date { get; set; }

    // Adaugam un string unde vom salva calea imaginii pentru articol
    public string Image { get; set; }
}
```

View-ul asociat:

```
<form enctype="multipart/form-data" asp-controller="Articles" asp-
action="UploadImage">

    <input class="form-control" placeholder="Article Name" type="text"
name="Title" />

    <br />

    <textarea class="form-control" placeholder="Article Content"
style="width: 100%; height: 100px;" name="Content"></textarea>

    <br />

    <input class="form-control" placeholder="Date" type="text"
name="Date" />

    <br />

    <input class="form-control" type="file" name="ArticleImage" />

    <br />

    <input class="btn btn-success" type="submit" value="Upload" />

</form>
```

Controller:

```
// Variabila locala de tip AppDBContext

private AppDBContext _context;
private IWebHostEnvironment _env;


// In constructor, se face dependency injection
public ArticlesController(AppDBContext context, IWebHostEnvironment
env)
{
    // Alocam conexiunea (injectata) cu baza de date unei proprietati
locale pentru a fi refolosita in metodele controller-ului
    _context = context;
    _env = env;
}


// Afisam view-ul cu form-ul
public IActionResult UploadImage()
{
    return View();
}


// Facem upload la fisier si salvam modelul in baza de date

[HttpPost]

public async Task<IActionResult> UploadImage(Article article,
IFormFile ArticleImage)

{
    // Verificam daca exista imaginea in request (daca a fost
incarcata o imagine)

    if (ArticleImage.Length > 0)

    {

        // Generam calea de stocare a fisierului
        var storagePath = Path.Combine(
                _env.WebRootPath, // Preluam calea folderului wwwroot
                "images", // Adaugam calea folderului images
                ArticleImage.FileName // Numele fisierului
            );
```

```
        // Generam calea de afisare a fisierului care va fi stocata in
baza de date
        var databaseFileName = "/images/" + ArticleImage.FileName;

        // Uploadam fisierul la calea de storage
        using (var fileStream = new FileStream(storagePath,
FileMode.Create))
        {
            await ArticleImage.CopyToAsync(fileStream);
        }
    }

    // Salvam storagePath-ul in baza de date
    article.Image = databaseFileName;
    _context.Articles.Add(article);
    _context.SaveChanges();


    return View();
}
```