Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC
Curs 9

Cuprins

Sistemul de autentificare 2
Roluri. Asocierea dintre roluri si utilizatori 3
Atributul [Authorize] 15
Implementare New, Edit si Delete utilizand roluri 15

Sistemul de autentificare

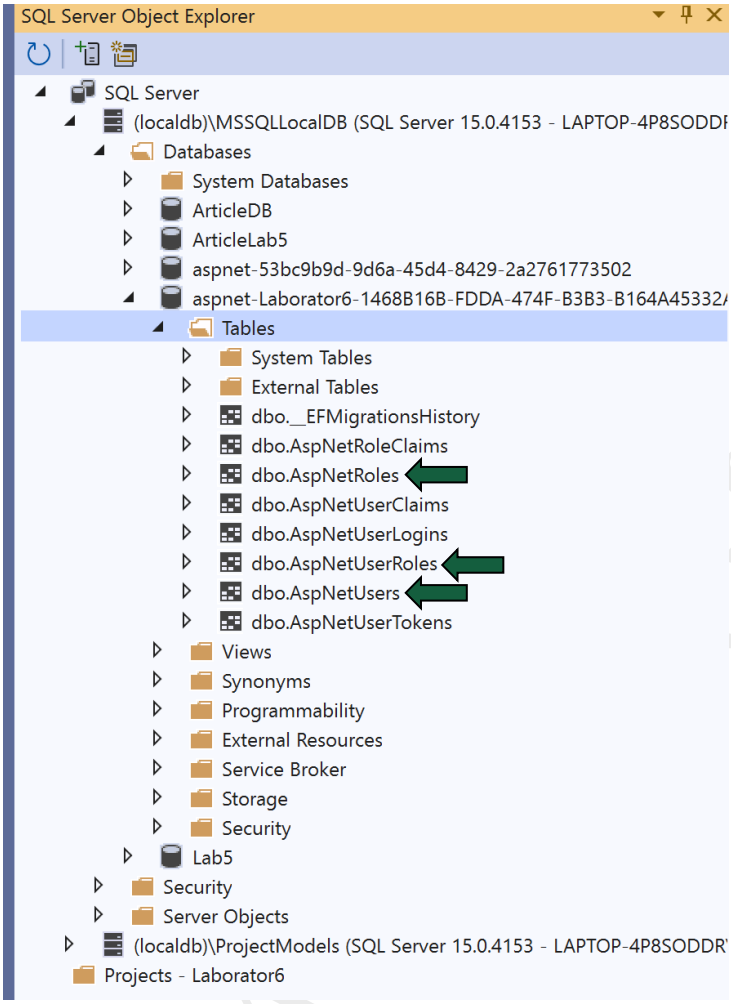
Framework-ul ASP.NET Core ofera posibilitatea integrarii unui sistem de autentificare folosind **Identity**.

Identity este compus dintr-o suita de clase si secvente de cod care faciliteaza implementarea rapida a unui sistem de autentificare complex. Acest sistem ofera posibilitatea autentificarii folosind user si parola, alocarea de roluri pentru utilizatori, autentificare folosind conturi 3rd party (autentificare prin retele de socializare – Google, Facebook, Twitter, etc). De asemenea, Identity include si posibilitatea crearii si manupularii rolurilor pe care le pot avea utilizatorii. ASP.NET Core Identity utilizeaza baza de date SQL Server pentru stocarea utilizatorilor, a rolurilor, dar si pentru asocierea dintre useri si roluri.

Pentru a genera un proiect care include componenta **Identity** pentru autentificare, trebuie sa alegem la crearea proiectului forma de autentificare: **Individual Accounts**.

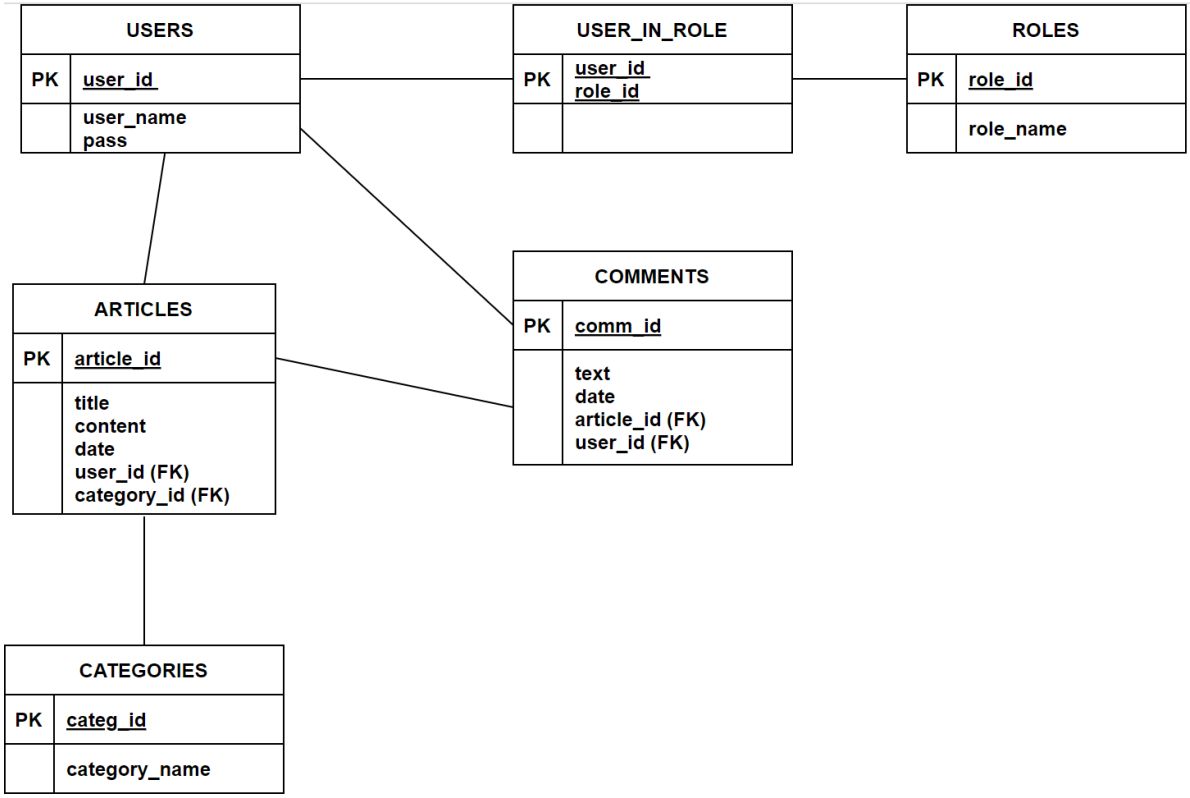
Adaugarea sistemului de autentificare a fost studiata in cadrul **Cursului 6 – Sectiunea Adaugarea sitemului de autentificare**.

Dupa crearea proiectului impreuna cu sistemul de autentificare, se pot vizualiza tabelele:



Roluri. Asocierea dintre roluri si utilizatori

Exemplele urmatoare sunt realizate in cadrul aplicatiei implementata in laborator (Engine de stiri), conform urmatoarei diagrame:



Se considera entitatile **Article**, **Category** si **Comment** cu urmatoarele proprietati:

Article:

- **Id** – int → id-ul articolului (cheie primara)
- **Title** – string → titlul articolului este obligatoriu (Required), poate avea o lungime maxima de 100 de caractere (StringLength) si nu poate avea mai putin de 5 caractere (MinLength)
- **Content** – string → continutul articolului este obligatoriu (Required)
- **Date** – DateTime → data si ora la care este postat articolul
- **Categoria** din care face parte articolul este obligatorie (Required)
- **CategoryId** – int → cheie externa – categoria din care face parte articolul
- **UserId** – string → cheie externa – reprezinta utilizatorul care a postat articolul

Category:

- **Id** – int → id-ul categoriei (cheie primara)
- **CategoryName** – string → numele categoriei este obligatoriu (Required)

Comment:

- **Id** – int → id-ul comentariului (cheie primara)
- **Content** – string → continutul comentariului este obligatoriu (Required)
- **Date** – DateTime → data la care a fost postat comentariul
- **ArticleId** – int (cheie externa) → articolul caruia ii apartine comentariul
- **UserId** – string (cheie externa) → utilizatorul care a postat comentariul

In continuare vom modifica anumite configuratii existente si generate de ASP.NET Core Identity, astfel incat sistemul de autentificare sa functioneze si sa putem configura si rolurile din cadrul aplicatiei.

Pentru adaugarea rolurilor si pentru realizarea asocierii dintre utilizatori si roluri trebuie parcursi urmatorii pasi:

PASUL 1:

Pentru definirea utilizatorilor din aplicatie, Identity include in baza de date un tabel Users, impreuna cu toate atributele necesare (Id, UserName, Email, Password, PhoneNumber, etc).

In cadrul **Pasului 1**, se creeaza o noua clasa in folderul Models, clasa care o sa mosteneasca clasa de baza IdentityUser din pachetul Microsoft.AspNetCore.Identity. In cazul in care dorim sa extindem clasa User, adaugand atribute, putem realiza acest lucru in clasa pe care urmeaza sa o implementam.

Se adauga o clasa pe care o numim **ApplicationUser**

```
namespace ArticlesApp.Models
{
	public class ApplicationUser : IdentityUser
	{

	}
}
```

Clasa pe care o mosteneste, **IdentityUser**, este clasa care descrie Userul in baza de date (cea care contine toate atributele unui utilizator).

PASUL 2:

In cadrul **Pasului 2** se adauga serviciile necesare in fisierul **Program.cs**

```
builder.Services.AddDefaultIdentity<ApplicationUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
                .AddRoles<IdentityRole>()
                .AddEntityFrameworkStores<ApplicationDbContext>(
                );
```

In fisier exista deja configuratia pentru **AddDefaultIdentity** (serviciul de User Interface, cookies si toate mecanismele necesare pentru functionarea autentificarii) si **AddEntityFrameworkStores** (realizeaza conexiunea cu baza de date, conectandu-se la baza de date a carui string de conexiune se afla in appsettings.json), fiind necesara adaugarea serviciului de management al rolurilor → **AddRoles**.

PASUL 3:

In folderul Data exista contextul bazei de date, fisierul **ApplicationDbContext.cs**, care contine conexiunea cu baza de date si configurarea provider-ului de baze de date (in cazul nostru SQL Server) folosind dependency injection.

In secventa urmatoare de cod, clasa ApplicationDbContext mosteneste clasa de baza IdentityDbContext – clasa care se ocupa cu managementul userilor si rolurilor (contine proprietati si metode cu ajutorul carora se pot prelucra userii si rolurile din aplicatie).

Clasa este de forma: `IdentityDbContext<TUser>` ceea ce inseamna ca primeste tipul clasei creata pentru prelucrarea userilor din baza de date.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
            : base(options)
        {
        }
      …
}
```

PASUL 4:

Se creeaza o noua clasa in folderul Models, cu ajutorul careia se vor adauga in baza de date Rolurile necesare. In acelasi timp se pot asocia si utilizatori cu fiecare rol. Acest lucru se face o singura data, urmand ca utilizatorii care creeaza cont sa primeasca un rol in momentul inregistrarii (logica la nivel de Controller).

Se creeaza o clasa, numita sugestiv **SeedData**, in cadrul careia vom implementa o metoda numita **Initialize** prin intermediul careia vom popula cele trei tabele: Users, Roles si UserRoles.

Un Service Provider este utilizat pentru a injecta dependentele in aplicatie (baza de date, sesiuni, pachete, autentificare, etc). In cazul nostru, cerem serviciul care realizeaza conexiunea cu baza de date.

```
public static class SeedData
    {
        public static void Initialize(IServiceProvider
serviceProvider)
        {
            using (var context = new ApplicationDbContext(
                serviceProvider.GetRequiredService
                <DbContextOptions<ApplicationDbContext>>()))
            {
                // Verificam daca in baza de date exista cel putin un
rol
                // insemnand ca a fost rulat codul
                // De aceea facem return pentru a nu insera rolurile
inca o data
                // Acesta metoda trebuie sa se execute o singura data
                if (context.Roles.Any())
                {
                    return;   // baza de date contine deja roluri
                }

                // CREAREA ROLURILOR IN BD
                // daca nu contine roluri, acestea se vor crea
                context.Roles.AddRange(
                    new IdentityRole { Id = "2c5e174e-3b0e-446f-86af-
483d56fd7210", Name = "Admin", NormalizedName = "Admin".ToUpper() },
                    new IdentityRole { Id = "2c5e174e-3b0e-446f-86af-
483d56fd7211", Name = "Editor", NormalizedName = "Editor".ToUpper() },
                    new IdentityRole { Id = "2c5e174e-3b0e-446f-86af-
483d56fd7212", Name = "User", NormalizedName = "User".ToUpper() }
                    );

                // o noua instanta pe care o vom utiliza pentru
crearea parolelor utilizatorilor
                // parolele sunt de tip hash
                var hasher = new PasswordHasher<ApplicationUser>();

                // CREAREA USERILOR IN BD
                // Se creeaza cate un user pentru fiecare rol
                context.Users.AddRange(
                    new ApplicationUser
                    {
                        Id = "8e445865-a24d-4543-a6c6-9443d048cdb0",
// primary key
                        UserName = "admin@test.com",
                        EmailConfirmed = true,
                        NormalizedEmail = "ADMIN@TEST.COM",
                        Email = "admin@test.com",
                        NormalizedUserName = "ADMIN@TEST.COM",
                        PasswordHash = hasher.HashPassword(null,
"Admin1!")
                    },
                    new ApplicationUser
                    {
```

```
                    Id = "8e445865-a24d-4543-a6c6-9443d048cdb1",
// primary key
                    UserName = "editor@test.com",
                    EmailConfirmed = true,
                    NormalizedEmail = "EDITOR@TEST.COM",
                    Email = "editor@test.com",
                    NormalizedUserName = "EDITOR@TEST.COM",
                    PasswordHash = hasher.HashPassword(null,
"Editor1!")
                },
                new ApplicationUser
                {
                    Id = "8e445865-a24d-4543-a6c6-9443d048cdb2",
// primary key
                    UserName = "user@test.com",
                    EmailConfirmed = true,
                    NormalizedEmail = "USER@TEST.COM",
                    Email = "user@test.com",
                    NormalizedUserName = "USER@TEST.COM",
                    PasswordHash = hasher.HashPassword(null,
"User1!")
                }
            );

            // ASOCIEREA USER-ROLE
            context.UserRoles.AddRange(
                new IdentityUserRole<string>
                {
                    RoleId = "2c5e174e-3b0e-446f-86af-
483d56fd7210",
                    UserId = "8e445865-a24d-4543-a6c6-
9443d048cdb0"
                },
                new IdentityUserRole<string>
                {
                    RoleId = "2c5e174e-3b0e-446f-86af-
483d56fd7211",
                    UserId = "8e445865-a24d-4543-a6c6-
9443d048cdb1"
                },
                new IdentityUserRole<string>
                {
                    RoleId = "2c5e174e-3b0e-446f-86af-
483d56fd7212",
                    UserId = "8e445865-a24d-4543-a6c6-
9443d048cdb2"
                }
            );

            context.SaveChanges();

        }
    }
```

```
}
```

Id-urile (Id, RoleId, UserId) au fost generate utilizand: https://guidgenerator.com/

PASUL 5:

Prin instanta curenta a aplicatiei se apeleaza din clasa SeedData, metoda Initialize, ducand la crearea rolurilor si a utilizatorilor in baza de date.

In **Program.cs** se adauga **Pasul 5**:

```
builder.Services.AddControllersWithViews();

var app = builder.Build();

// PASUL 5 - useri si roluri

using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    SeedData.Initialize(services);
}
```

PASUL 6:

Se adauga proprietatile in clasele din Model, astfel:

In clasa Article

```
// PASUL 6 - useri si roluri
   public virtual ApplicationUser User { get; set; } → un
articol apartine unui singur utilizator
```

In clasa Comment

```
// PASUL 6 - useri si roluri
  public virtual ApplicationUser User { get; set; } → un
comentariu apartine unui singur utilizator
```

PASUL 7:

In folderul Views → Shared → _LoginPartial.cshtml → se modifica astfel:

```
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager
```

```
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager
```

PASUL 8:

Se realizeaza o noua migratie in baza de date.

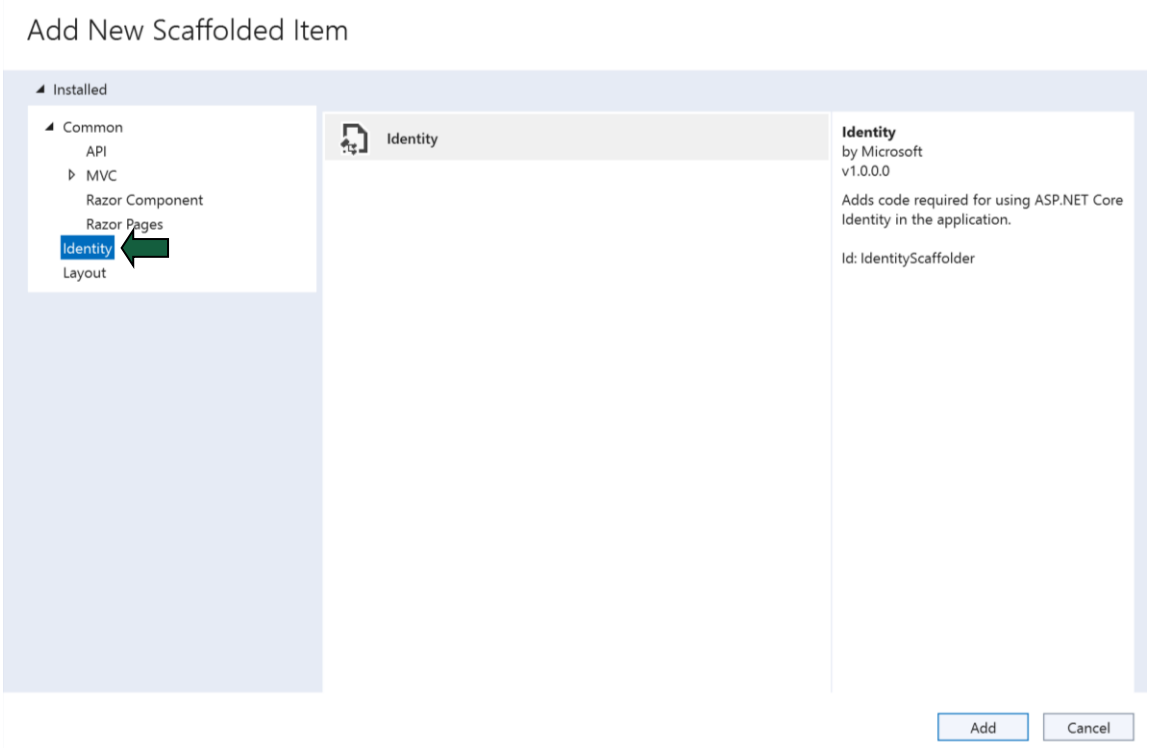
Se ruleaza pe rand:

1. Add-Migration DenumireMigratie
2. Update-Database

PASUL 9:

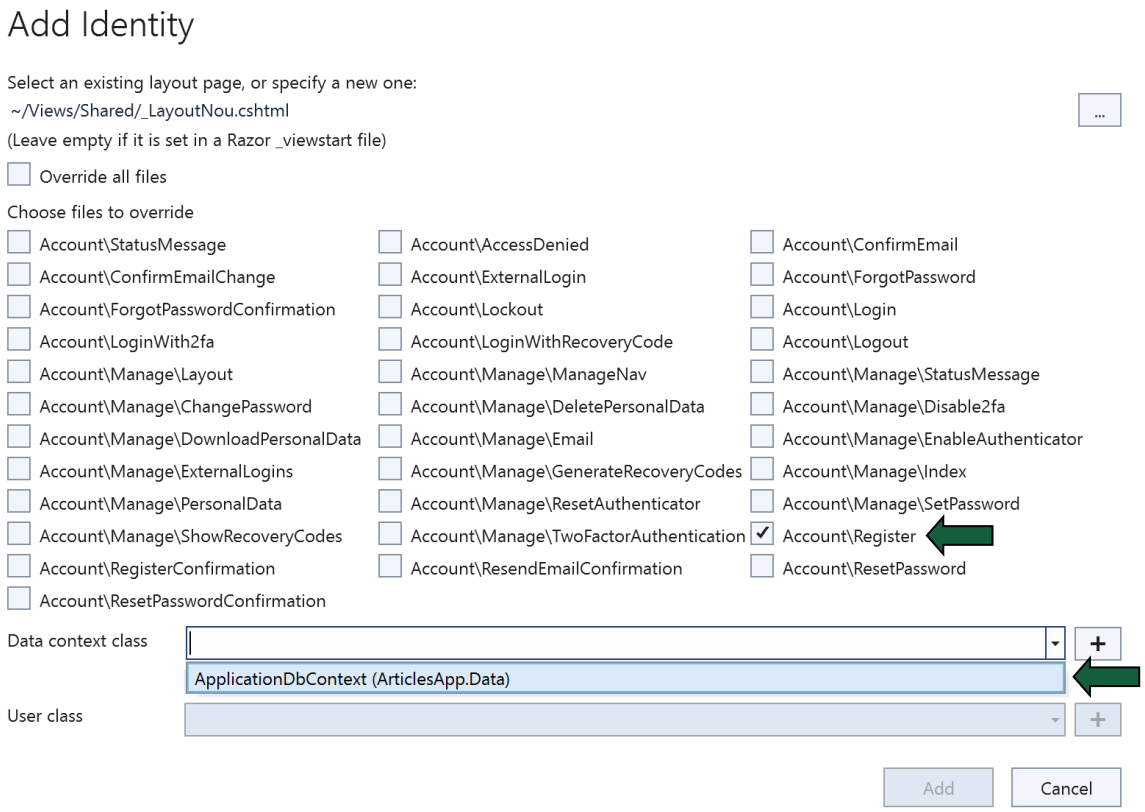
Pentru modificarea functionalitatilor implementate in framework, se poate utiliza optiunea **Add Scaffolded Item**, prin intermediul careia se poate aduce in folderele de lucru codul sursa.

Click dreapta pe numele proiectului din Solution Explorer → Add → New Scaffolded Item



Pentru exemplul din cadrul cursului, avem nevoie de functionalitatea de inregistrare. In momentul in care un utilizator se inregistreaza in aplicatie, acesta trebuie sa primeasca un rol. In cazul aplicatiei Engine de stiri, utilizatorii inregistrati o sa primeasca rolul User. Userii inregistrati pot vedea articolele, pot lasa comentarii, pot edita si sterge propriile comentarii. Utilizatorii devin editori numai in momentul in care Adminul schimba rolul.

Pentru adaugarea rolului de User in momentul inregistrarii, trebuie modificata metoda Register.



Codul inclus se poate accesa din Solution Explorer → Areas → Identity → Pages → Account → Register.cshtml → Register.cshtml.cs\

```
// PASUL 9 - useri si roluri (adaugarea rolului la inregistrare)
await _userManager.AddToRoleAsync(user, "User");
```

PASUL 10:

Ultimul pas este reprezentat de configurarea managerului de useri si roluri. In Controller-ul **ArticlesController** se implementeaza urmatoarea secventa de cod:

```
public class ArticlesController : Controller

{
  private readonly ApplicationDbContext db;

  private readonly UserManager<ApplicationUser> _userManager;

  private readonly RoleManager<IdentityRole> _roleManager;

  public ArticlesController(
      ApplicationDbContext context,
      UserManager<ApplicationUser> userManager,
      RoleManager<IdentityRole> roleManager
  )
  {
      db = context;
      _userManager = userManager;
      _roleManager = roleManager;
  }


…
 }
```

Documentatie Microsoft – User Manager: https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.usermanager-1?view=aspnetcore-6.0

Documentatie Microsoft – Role Manager: https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.rolemanager-1?view=aspnetcore-6.0

Atributul [Authorize]

Pentru a restrictiona accesul utilizatorilor la metodele din cadrul unui Controller, se utilizeaza atributul [**Authorize]** la nivelul Controller-ului respectiv.

Pentru a restrictiona accesul in cadrui fiecarei metode, se utilizeaza acelasi atribut, astfel:

```
[Authorize(Roles = "User,Editor,Admin")]
public IActionResult Index()
{  …  }
```

Doar utilizatorii cu rolul User, Editor sau Admin, pot accesa metoda Index.

Implementare New, Edit si Delete utilizand roluri

In cazul metodelor de adaugare, editare si stergere trebuie sa se tina cont si de id-ul utilizatorului care adauga, editeaza sau sterge articolul. In cazul adaugarii o sa se preia id-ul utilizatorului pentru adaugarea lui in baza de date. In cazul editarii se verifica daca utilizatorul care doreste sa editeze articolul este utilizatorul caruia ii apartine articolul respectiv (adica utilizatorul care a postat articolul). Asemanator se procedeaza si in cazul stergerii unui articol. Un user poate sterge un articol doar daca respectivul articol ii apartine. Un utilizator nu poate sterge articole postate in platforma de alti utilizatori.

Exemplu implementare metoda New cu Post din ArticlesController:

```
[Authorize(Roles = "Editor,Admin")]

[HttpPost]

public IActionResult New(Article article)
{
    article.Date = DateTime.Now;

    article.UserId = _userManager.GetUserId(User);
```

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

In exemplul anterior se poate observa cum cu ajutorul managerului de useri este preluat id-ul userului curent, folosind metodei `GetUserId`. Astfel, in momentul inserarii articolului in baza de date, se insereaza si id-ul utilizatorului care a postat articolul respectiv.

```
_userManager.GetUserId(User);
```

In cazul editarii, se verifica daca utilizatorul care doreste sa modifice articolul este utilizatorul care a postat articolul. Preluarea id-ului userului curent se realizeaza tot cu ajutorul metodei `GetUserId`. De asemenea, Administratorul avand drepturi depline asupra aplicatiei, se verifica si rolul. Daca utilizatorul are rolul Admin, atunci el poate edita articolul. Pentru verificarea rolului se utilizeaza metoda `IsInRole("NumeRol")`

Edit - HttpGet

```
…
    if (article.UserId == _userManager.GetUserId(User) ||
User.IsInRole("Admin"))
    {
         return View(article);
    }

    else
    {
        TempData["message"] = "Nu aveti dreptul sa faceti
modificari asupra unui articol care nu va apartine";
        return RedirectToAction("Index");
    }
    …
```

La fel se procedeaza si in cazul editarii cu HttpPost, dar si in cazul stergerii. Se verifica daca utilizatorul care intentioneaza sa editeze sau sa stearga articolul este userul care a creat respectivul articol. In plus, se verifica si rolul utilizatorului care face actiunea. Daca acesta este Admin, atunci poate face C.R.U.D. asupra oricarei entitati din aplicatie.