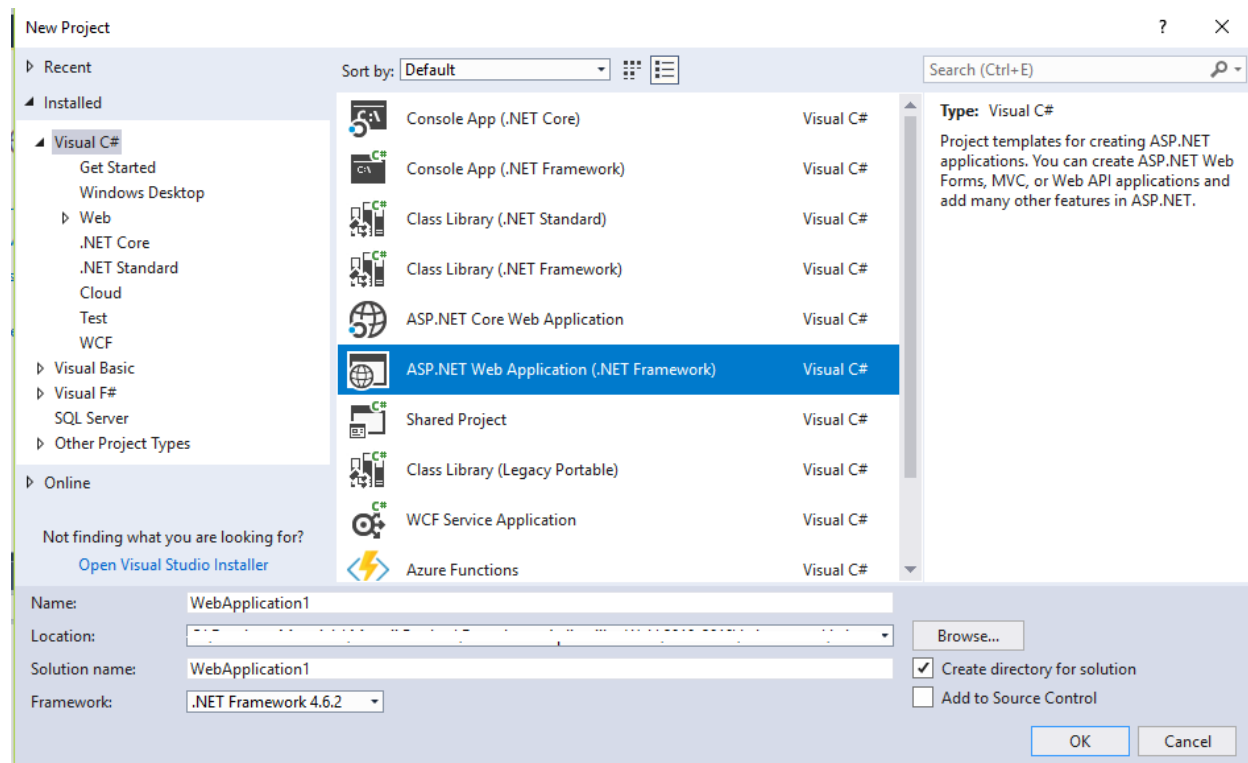


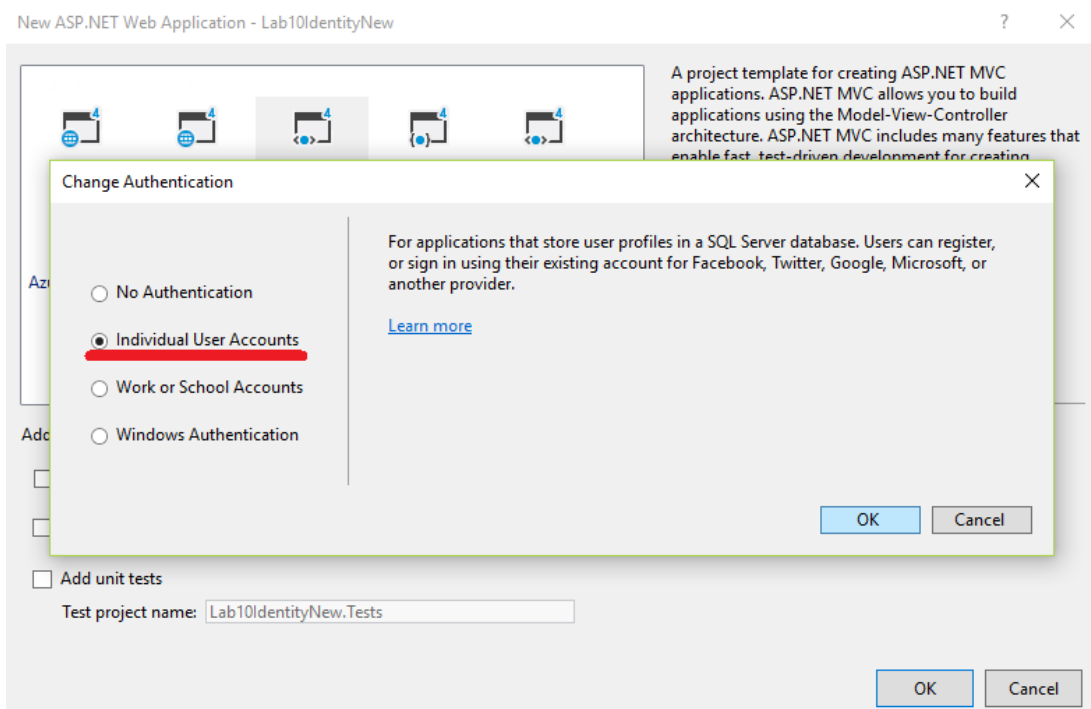
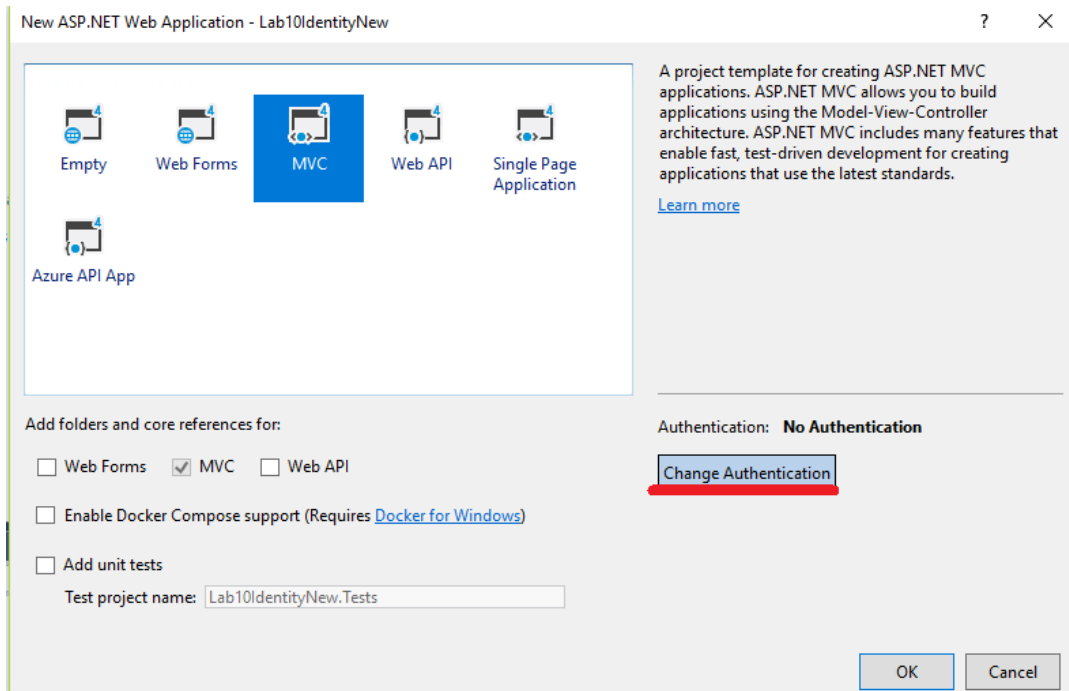
Autentificare. Roluri. Asocierea dintre roluri si utilizatori.

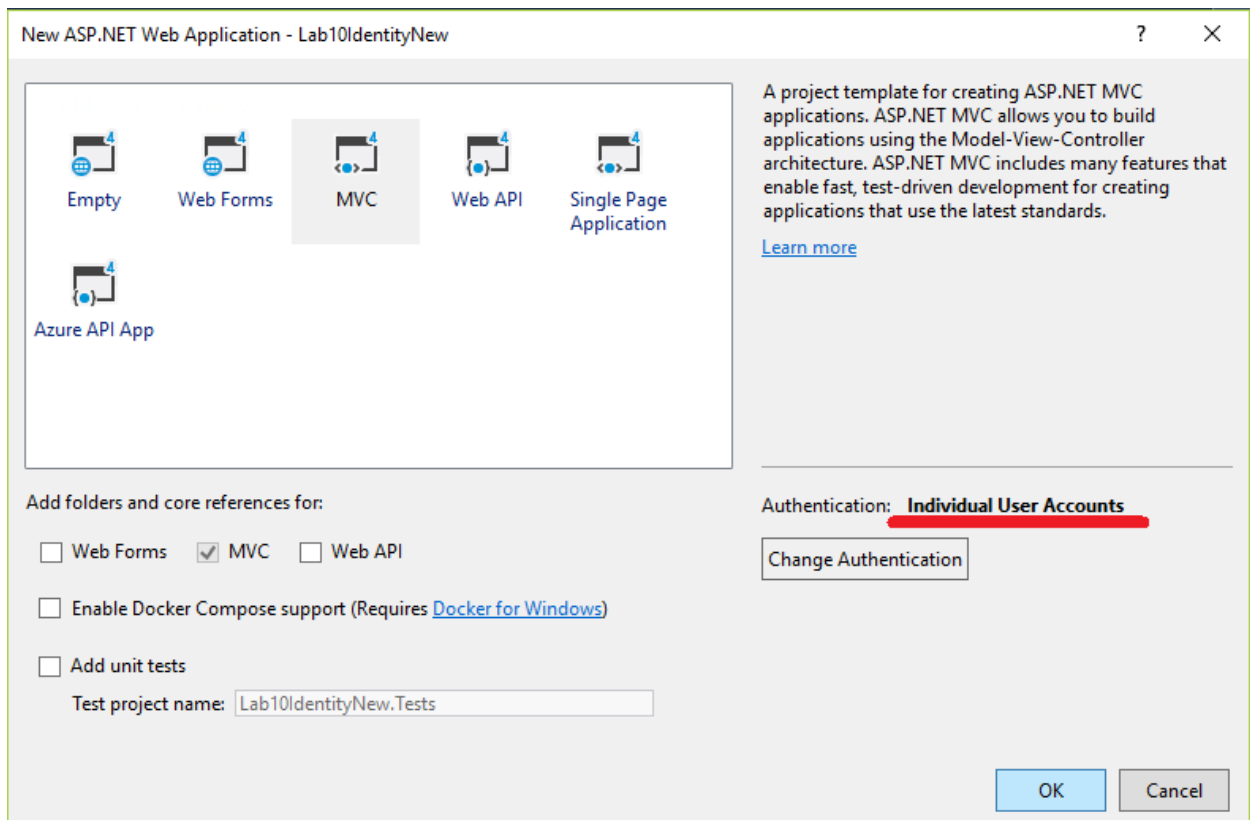
Autentificare

Framework-ul ASP.NET MVC 5 ofera posibilitatea generarii 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, Microsoft).

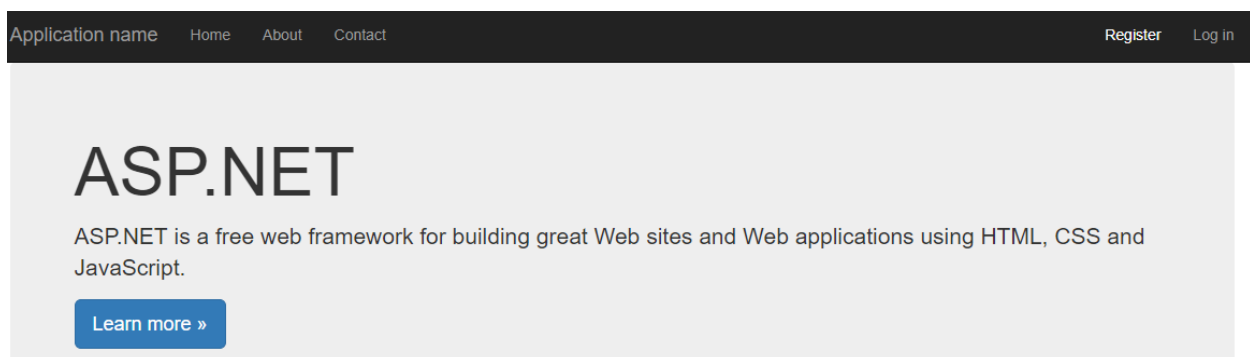
Pentru a genera un proiect care include componenta Identity pentru autentificare, trebuie sa alegem la crearea proiectului forma de autentificare: “Individual user accounts”.








Proiectul nou creat contine Identity framework si toate mecanismele aferente autentificarii. Putem, in acest moment, sa cream un cont si sa ne autentificam in acesta:



Register.

Create a new account.

Email	<input type="text" value="user@test.com"/>	
Password	<input type="password" value="....."/>	
Confirm password	<input type="password" value="....."/>	
<input type="button" value="Register"/>		

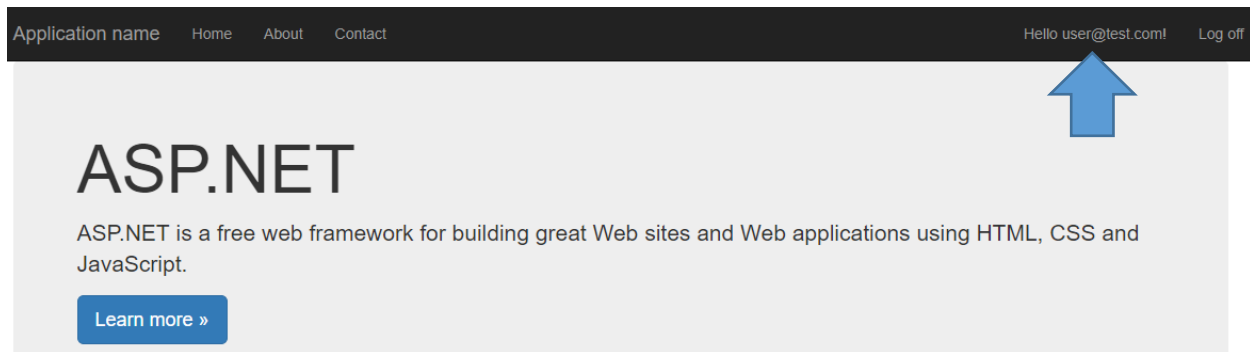
© 2018 - My ASP.NET Application

Register.

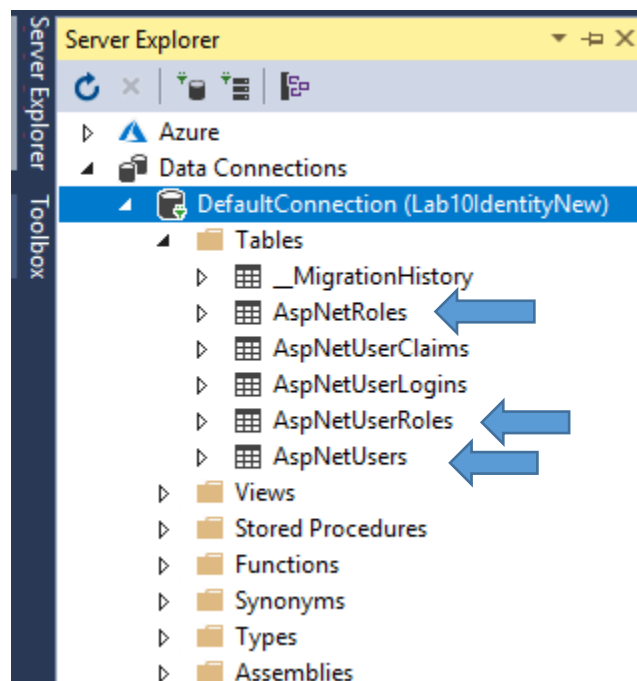
Create a new account.

- Passwords must have at least one non letter or digit character. Passwords must have at least one digit ('0'-'9'). Passwords must have at least one uppercase ('A'-'Z').

Email	<input type="text" value="user@test.com"/>
Password	<input type="password"/>
Confirm password	<input type="password"/>
<input type="button" value="Register"/>	



Sistemul genereaza baza de date corespunzatoare autentificarii (utilizatori, roluri, legatura dintre roluri si utilizatori):



Roluri. Asocierea dintre roluri si utilizatori

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

- In folderul **App_Start**, in fisierul **Startup.Auth.cs** se adauga urmatoarea linie de cod pentru managerul de roluri

```
app.CreatePerOwinContext<ApplicationRoleManager>(ApplicationRoleManager.Create);
```

- In folderul **App_Start**, in fisierul **IdentityConfig.cs** se adauga urmatoarele linii de cod pentru managerul de roluri

```
public class ApplicationRoleManager : RoleManager<IdentityRole>
{
    public ApplicationRoleManager(IRoleStore<IdentityRole, string> store) :
base(store)
    {
    }

    public static ApplicationRoleManager
Create(IdentityFactoryOptions<ApplicationRoleManager> options, IOwinContext context)
    {
        var roleStore = new
RoleStore<IdentityRole>(context.Get<ApplicationDbContext>());
        return new ApplicationRoleManager(roleStore);
    }
}
```

- In fisierul Startup.cs se defineste o metoda care se apeleaza in corpul functiei **Configuration**. Aceasta metoda va contine codul necesar pentru adaugarea rolurilor si a utilizatorilor impliciti in aplicatie

```
public void Configuration(IAppBuilder app)
{
    ConfigureAuth(app);
    // Se apeleaza o metoda in care se adauga contul de administrator
si rolurile aplicatiei
    createAdminUserAndApplicationRoles();
}
```

```

private void createAdminUserAndApplicationRoles()
{
    ApplicationDbContext context = new ApplicationDbContext();
    var roleManager = new RoleManager<IdentityRole>(new
RoleStore<IdentityRole>(context));
    var UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(context));

    // Se adauga rolurile aplicatiei
    if (!roleManager.RoleExists("Administrator"))
    {
        // Se adauga rolul de administrator
        var role = new IdentityRole();
        role.Name = "Administrator";
        roleManager.Create(role);

        // se adauga utilizatorul administrator
        var user = new ApplicationUser();
        user.UserName = "admin@admin.com";
        user.Email = "admin@admin.com";

        var adminCreated = UserManager.Create(user, "Administrator1!");
        if (adminCreated.Succeeded)
        {
            UserManager.AddToRole(user.Id, "Administrator");
        }
    }

    if (!roleManager.RoleExists("Editor"))
    {
        var role = new IdentityRole();
        role.Name = "Editor";
        roleManager.Create(role);
    }

    if (!roleManager.RoleExists("User"))
    {
        var role = new IdentityRole();
        role.Name = "User";
        roleManager.Create(role);
    }
}

```

Dupa adaugarea modificarilor, se ruleaza aplicatia. Astfel, codul va fi executat iar informatiile vor fi stocate in baza de date:

Server Explorer Toolbox

dbo.AspNetRoles [Data] X

Max Rows: 1000

	Id	Name
▶	7fa-0a2d13c9b100	Administrator
	3acf17d0-a5d0-...	Editor
	a71b8d7f-4428-...	User
★	NULL	NULL

Toolbox

dbo.AspNetUsers [Data] X

dbo.AspNetRoles [Data]

dbo.AspNetUserRoles [Data]

Max Rows: 1000

	Id	Email	EmailConfirmed	PasswordHash	SecurityStamp	Phor
	0fec526b-2530-45e4-a569-38615c38199b	user@test.com	False	ABYQ+ujyPH3l...	18ea4ff3-ddb2-...	NULL
▶	45199f83-227d-41d2-8ede-9aec62281d45	admin@admin....	False	AlvdguAKOuIW...	ba24420d-0210...	NULL
★	NULL	NULL	NULL	NULL	NULL	NULL

Toolbox

dbo.AspNetUsers [Data]

dbo.AspNetRoles [Data]

dbo.AspNetUserRoles [Data] X

Max Rows: 1000

	UserId	RoleId
	45199f83-227d-41d2-8ede-9aec62281d45	e0ed7d37-e081-40a2-a7fa-0a2d13c9b100
▶★	NULL	NULL

Exemplu

Consideram urmatorul exemplu: O aplicatie in care se pot adauga articole de catre utilizatori (editori). Fiecare articol are o singura categorie. Categoriile pot fi administrate doar de utilizatorii cu rolul "Administrator". Editorii pot sa adauge, listeze, modifice si stearga **doar** articolele *proprie*. Utilizatorii inregistrati (fara rol de editor – sau cu rolul "utilizator") pot doar sa vizualizeze articolele existente. Utilizatorii neinregistrati nu pot vedea articolele. Acestia sunt redirectionati catre pagina de inregistrare sau autentificare.

Rezolvare

- Pentru a restrictiona accesul utilizatorilor la metodele pentru C.R.U.D. pe categorii, vom folosi helper-ul Authorize la nivel de Controller pentru controller-ul Categories:

```
...
[Authorize(Roles = "Administrator")]
public class CategoriesController : Controller
{
...

```

Pentru rolul de editor trebuie facute mai multe modificari. In primul rand, se adauga pentru metodele **New**, **Edit** si **Delete** urmatoarele:

- Pentru metodele **Index**, **Show** vom folosi:
[Authorize(Roles = "User,Editor,Administrator")]
- Pentru metodele **New**, **Edit**, **Delete** vom folosi:
[Authorize(Roles = "Editor,Administrator")]

Dupa alocarea rolurilor este necesar sa modificam modelul **Article** (modelul corespunzator pentru articole) astfel incat sa includa si ID-ul utilizatorului care a creat respectivul articol:

```
...
public string UserId { get; set; }
public virtual ApplicationUser User { get; set; }
...

```

De asemenea, contextul bazei de date, se va muta in modelul **IdentityModel.cs** deoarece este necesar ca tabelele utilizatorilor, cat si tabelele celorlalte modele din aplicatie sa se regaseasca in aceeași baza de date.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext()
        : base("DefaultConnection", throwIfV1Schema: false)
    {
    }

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

    public static ApplicationDbContext Create()
    {
        return new ApplicationDbContext();
    }
}
```

Metodele Edit si New se modifica corespunzator astfel incat la inserarea in baza de date si la modificarea unui articol sa se tina cont de ID-ul utilizatorului care face aceasta actiune.

La inserarea in baza de date trebuie transmis ca parametru, prin intermediul formularului, si ID-ul utilizatorului. Astfel, el va fi pus in variabila ViewBag din Controller si va fi inclus sub forma unui camp ascuns in View:

```
...
[Authorize(Roles = "Editor,Administrator")]
public ActionResult New()
{
    Article article = new Article();
    // preluam lista de categorii din metoda GetAllCategories()
    article.Categories = GetAllCategories();
    // Preluam ID-ul utilizatorului curent
    article.UserId = User.Identity.GetUserId();
    return View(article);
}
...

<form method="post" action="/Article/New">
    @Html.HiddenFor(m => m.UserId)
...

```

Pentru afisarea tuturor informatiilor referitoare la un articol (de exemplu in metoda Index) este necesar sa facem join si cu modelul User (ApplicationUser din fisierul IdentityModels.cs) astfel:

```
var articles = db.Articles.Include("Category").Include("User");
```

View-ul se va modifica pentru a include si numele autorului articolului:


```
...  
<i class="glyphicon glyphicon-user"></i> <i>articol scris de</i> <strong>  
@Model.User.UserName  
</strong>  
...
```


Afisare articole

test

Continut articol: **test**

12/2/2018 1:13:42 AM

 test

 articol scris de **admin@admin.com**

Afisare articol

Adauga un articol

Ultimul pas necesar pentru finalizarea cerintelor este de a verifica daca ID-ul utilizatorului care doreste sa editeze, respectiv sa stearga un articol, corespunde cu ID-ul utilizatorului care a creat articolul respectiv. De asemenea, daca utilizatorul care incearca aceste actiuni are rolul "Administrator" atunci verificarea nu mai este necesara.

```
[Authorize(Roles = "Editor,Administrator")]
public ActionResult Edit(int id)
{
    Article article = db.Articles.Find(id);
    ViewBag.Article = article;
    article.Categories = GetAllCategories();

    if(article.UserId == User.Identity.GetUserId() ||
User.IsInRole("Administrator"))
    {
        return View(article);
    } else
    {
        TempData["message"] = "Nu aveti dreptul sa faceti modificari asupra
unui articol care nu va apartine!";
        return RedirectToAction("Index");
    }
}
. . .
[HttpPost]
[Authorize(Roles = "Editor,Administrator")]
public ActionResult Edit(int id, Article requestArticle)
{
    try
    {
        if (ModelState.IsValid)
        {
            Article article = db.Articles.Find(id);

            if (article.UserId == User.Identity.GetUserId() ||
User.IsInRole("Administrator"))
            {
                if (TryUpdateModel(article))
                {
                    article.Title = requestArticle.Title;
                    article.Content = requestArticle.Content;
                    article.Date = requestArticle.Date;
                    article.CategoryId = requestArticle.CategoryId;
                    db.SaveChanges();
                    TempData["message"] = "Articolul a fost modificat!";
                }
                return RedirectToAction("Index");
            }
        }
    }
}
```

```

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

    }
    else
    {
        return View();
    }

}
catch (Exception e)
{
    return View();
}
}

. . .

[HttpDelete]
[Authorize(Roles = "Editor,Administrator")]
public ActionResult Delete(int id)
{
    Article article = db.Articles.Find(id);

    if (article.UserId == User.Identity.GetUserId() ||
User.IsInRole("Administrator"))
    {
        db.Articles.Remove(article);
        db.SaveChanges();
        TempData["message"] = "Articolul a fost sters!";
        return RedirectToAction("Index");
    }
    else
    {
        TempData["message"] = "Nu aveti dreptul sa stergeti un articol care
nu va apartine!";
        return RedirectToAction("Index");
    }
}
}

```

Pentru restrictionarea accesului utilizatorilor neinregistrati in paginile de articole, folosim la nivel de Controller filtrul **[Authorize]**.

/!\ Atentie: Pentru a aloca in mod automat unui utilizator un rol la *inregistrare* este necesar sa se modifice fisierul **AccountController** din folderul Controllers. In metoda **Register ([HttpPost])** trebuie adaugata secventa de cod necesara pentru adaugarea utilizatorului la rolul dorit in momentul salvarii:

```
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email =
model.Email };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await SignInManager.SignInAsync(user, isPersistent:false,
rememberBrowser:false);

            UserManager.AddToRole(user.Id, "User");
            . . .
        }
    }
}
```