

## Laborator 6 – Laborant Guster Andreea

### Activarea autentificarii – Exercițiul 1

Vom crea un proiect nou. La creare selectati:

The screenshot shows the 'Create a new ASP.NET Web Application' dialog in Visual Studio. The 'MVC' template is selected, and the 'Change Authentication' dialog is open, showing 'Individual User Accounts' as the selected authentication method. Blue arrows indicate the flow of the process.

### Create a new ASP.NET Web Application

**Empty**  
An empty project template for creating ASP.NET applications. This template does not have any content in it.

**Web Forms**  
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

**MVC**  
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

**Web API**  
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

**Single Page Application**  
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

**Authentication**  
No Authentication  
[Change](#)

**Add folders & core references**  
☐ Web Forms  
☒ MVC  
☐ Web API

**Advanced**  
☒ Configure for HTTPS  
☐ Docker support  
(Requires [Docker Desktop](#))  
☐ Also create a project for unit tests  
lab6.Tests

[Back](#) [Create](#)

### Change Authentication

For applications that store user profiles in a SQL Server database. Users can register, or sign in using their existing account for Facebook, Twitter, Google, Microsoft, or another provider. [Learn more](#)

☐ No Authentication  
☒ Individual User Accounts  
☐ Work or School Accounts  
☐ Windows Authentication

[OK](#) [Cancel](#)

## Modalitati de a restrictiona accesul

Putem pune atributul **[Authorize]**:

1. Pe antetul unei singure **actiuni** din controller si permite accesarea acesteia DOAR de catre utilizatorii autentificati.
2. Inaintea **controller-ului** si permite accesarea tuturor actiunilor din controller-ul respectiv DOAR de catre utilizatorii autentificati.
3. In metoda **RegisterGlobalFilters** din clasa **FilterConfig**, din fisierul **App\_Start/FilterConfig.cs**, si permite accesarea DOAR de catre utilizatorii autentificati a tuturor actiunilor din **intreaga aplicatie**.

```
public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new AuthorizeAttribute());
        filters.Add(new HandleErrorAttribute());
    }
}
```

Atributul **[AllowAnonymous]** suprascrie, pentru o **actiune** sau un **controller**, o restrictie aplicata la nivel superior (vezi al 3-lea punct din lista precedenta). Acesta le permite tuturor tipurilor de utilizatori (inclusiv cei neautentificati) sa acceseze actiunea sau controller-ul pentru care este aplicata/aplicat.

## Roluri

Pentru a aloca la inregistrare (crearea unui utilizator nou) un anumit rol trebuie sa scriem urmatoarea secventa de cod in controller-ul **Account**, in actiunea **Register (HttpPost)**:

```
// 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);

            // To every new user, the role user will be binded to it
            var roleStore = new RoleStore<IdentityRole>(new ApplicationDbContext());
            var roleManager = new RoleManager<IdentityRole>(roleStore);

            if (!roleManager.RoleExists("User"))
                roleManager.Create(new IdentityRole("User"));
            UserManager.AddToRole(user.Id, "User");

            return RedirectToAction("Index", "Home");
        }
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form
```

```

        return View(model);
    }

```

## Accesarea in functie de roluri

Putem adauga parametrul **Roles** la atributul **Authorize** pentru a restrange mai mult aria de restrictie. De exemplu in functie de rolurile utilizatorilor (separate prin virgula):

```

[Authorize(Roles = "Super")]
[Authorize(Roles = "User,Admin")]

```

Intr-o actiune oarecare putem verifica care este rolul utilizatorului curent:

```

if (User.IsInRole("Admin"))

```

Aceasta functionalitate este utila atunci cand dorim ca unele functionlitati sa fie accesate doar de un anumit tip de utilizatori.

- De exemplu, utilizatorul de tipul **Admin** are dreptul sa stearga/modifice datele utilizatorilor inregistrati in aplicatie.
- Pentru o aplicatie de tipul unui blog ce contine articole putem avea urmatoarea impartire:
  - Utilizatorii cu rolul **Editor** isi pot modifica/sterge doar articolele create de ei, pot crea articole noi si pot vizualiza toate articolele din cadrul aplicatiei
  - Utilizatorii cu rolul **Admin** pot modifica/sterge/vizualiza toate articolele din cadrul aplicatiei
  - Utilizatorii **neautentificati** (atributul **AllowAnonymous**) pot doar sa vizualizeze articolele din cadrul aplicatiei. Acestia NU pot modifica/crea/sterge niciun articol

## Accesarea Bazei de Date

Clasa-context pentru a accesa baza de date a utilizatorilor este **ApplicationDbContext** (aflata in fisierul **Models/IdentityModels.cs**).

**!!! ATENTIE !!!** pentru cand o sa lucrati la proiecte. Acum contextul bazei de date se muta in modelul **IdentityModels.cs** daca doriti ca tabelele celorlalte modele din aplicatie (ex Book, Publisher, etc) sa fie incluse in aceeasi baza de date cu tabelele utilizatorilor. Exemplu:

```

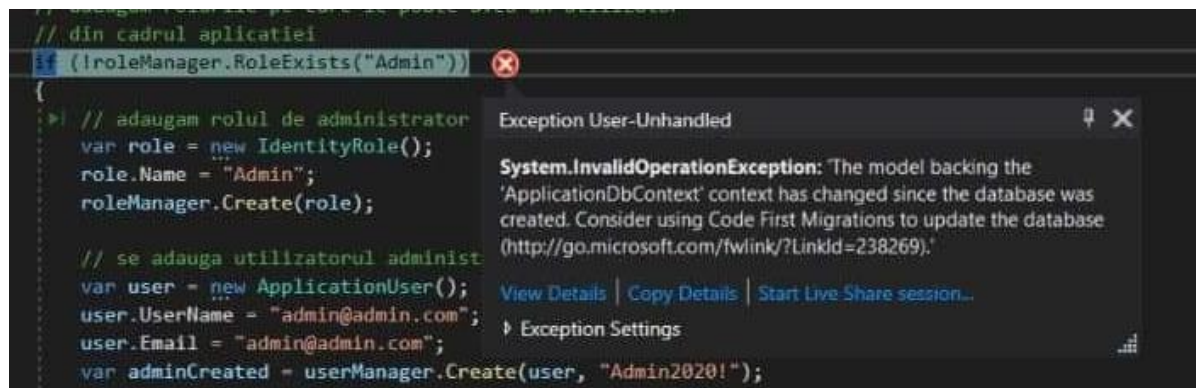
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext()
        : base("DefaultConnection", throwIfV1Schema: false)
    {
    }

    public DbSet<Book> Books { get; set; }
    public DbSet<Publisher> Publishers { get; set; }

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

```

**!!! ATENTIE !!!** Daca o sa modificati modelele **Book** sau **Publisher** o sa va apara urmatoarea eroare cand rulati aplicatia:



Pentru a rezolva aceasta problema este necesar sa ii mentionam contextului sa faca sterga baza de date si sa o creeze din nou atunci cand modificam modelele. Deci, in fisierul **Global.asax**, in interiorul metodei **Application\_Start** vom specifica care este modul de initializare al bazei de date:

```

protected void Application_Start()
{
    Database.SetInitializer<ApplicationDbContext>(new
DropCreateDatabaseIfModelChanges<ApplicationDbContext>());
    AreaRegistration.RegisterAllAreas();
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}

```

**!!! ATENTIE !!!** Inainte de a rula asigurati-va ca inchideti conexiunea la baza de date din **Server Explorer**  
-> **Data Connections** -> **Default Connection** -> click dreapta -> **Close Connection**.

### Crearea rolului de Administrator

Pentru a adauga rolul si user-ul **admin** in baza de date trebuie sa adaugati in fisierul **Startup.cs** in metoda **Configuration** urmatoarea metoda.

```

public partial class Startup
{
    public void Configuration(IApplicationBuilder app)
    {
        ConfigureAuth(app);
        CreateAdminAndUserRoles();
    }

    private void CreateAdminAndUserRoles()
    {
        var ctx = new ApplicationDbContext();
        var roleManager = new RoleManager<IdentityRole>(
            new RoleStore<IdentityRole>(ctx));
        var userManager = new UserManager<ApplicationUser>(
            new UserStore<ApplicationUser>(ctx));

        // adaugam rolurile pe care le poate avea un utilizator
        // din cadrul aplicatiei
        if (!roleManager.RoleExists("Admin"))
        {
            // adaugam rolul de administrator
            var role = new IdentityRole();
            role.Name = "Admin";
            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, "Admin2020!");
        if (adminCreated.Succeeded)
        {
            userManager.AddToRole(user.Id, "Admin");
        }
    }

    // ATENTIE !!! Pentru proiecte, pentru a adauga un rol nou trebuie sa adaugati secventa:
    /*if (!roleManager.RoleExists("your_role_name"))
    {
        // adaugati rolul specific aplicatiei voastre
        var role = new IdentityRole();
        role.Name = "your_role_name";
        roleManager.Create(role);

        // se adauga utilizatorul
        var user = new ApplicationUser();
        user.UserName = "your_user_email";
        user.Email = "your_user_email";
    }*/
}
}

```

## Exercitiul 2

[Cerinta] Creati un controller accesibil doar de catre administratori, ce contine actiuni ce afiseaza lista utilizatorilor si permit modificarea rolurilor lor.

**!!!ATENTIE!!!** Intre **Users** si **Roles** este relatia **many-to-many**. Pentru a putea adauga un rol unui utilizator din pagina de editare ne vom folosi de un model auxiliar care va contine pe langa user si o variabila de tipul **string** care memoreaza numele rolului utilizatorului. Si vom trimite view-ului edit obiectul de tipul modelului auxiliar (UserViewModel).

In folder-ul **Models** creati clasa **UserViewModel**.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace lab6.Models
{
    public class UserViewModel
    {
        public ApplicationUser User { get; set; }
        public string RoleName { get; set; }
    }
}

```

Creati controller-ul **UsersController**.

```

using lab6.Models;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace lab6.Controllers

```

```

{ // poate fi accesat doar de catre Admin
  [Authorize(Roles="Admin")]
  public class UsersController : Controller
  {
    private ApplicationDbContext ctx = new ApplicationDbContext();
    public ActionResult Index()
    {
      ViewBag.UsersList = ctx.Users
        .OrderBy(u => u.UserName)
        .ToList();

      return View();
    }

    public ActionResult Details(string id)
    {
      if (String.IsNullOrEmpty(id))
      {
        return HttpNotFound("Missing the id parameter!");
      }
      ApplicationUser user = ctx.Users
        .Include("Roles")
        .FirstOrDefault(u => u.Id.Equals(id));

      if (user != null)
      {
        ViewBag.UserRole = ctx.Roles
          .Find(user.Roles.First().RoleId).Name;
        return View(user);
      }
      return HttpNotFound("Cloudn't find the user with given id!");
    }

    public ActionResult Edit(string id)
    {
      if (String.IsNullOrEmpty(id))
      {
        return HttpNotFound("Missing the id parameter!");
      }

      UserViewModel uvm = new UserViewModel();
      uvm.User = ctx.Users.Find(id);

      IdentityRole userRole = ctx.Roles
        .Find(uvm.User.Roles.First().RoleId);
      uvm.RoleName = userRole.Name;
      return View(uvm);
    }

    [HttpPut]
    public ActionResult Edit(string id, UserViewModel uvm)
    {
      ApplicationUser user = ctx.Users.Find(id);
      try
      {
        if (TryUpdateModel(user))
        {
          {
            var um = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(ctx));
            foreach (var r in ctx.Roles.ToList())
            {
              um.RemoveFromRole(user.Id, r.Name);
            }
            um.AddToRole(user.Id, uvm.RoleName);
            ctx.SaveChanges();
          }
          return RedirectToAction("Index");
        }
      }
      catch (Exception e)
      {
      }
    }
  }
}

```

```

        return View(uvm);
    }
}
}

```

In folder-ul **Views** creati view-urile **Details.cshtml**, **Edit.cshtml**, **Index.cshtml**.

#### Index.cshtml

```

@{
    ViewBag.Title = "Users";
}

<h2>@ViewBag.Title</h2>

<table class="table">
    <thead>
        <th>Username</th>
        <th>Email</th>
        <th>Update</th>
        <th>Details</th>
    </thead>
    <tbody>
        @foreach (var user in ViewBag.UsersList)
        {
            <tr>
                <td>@user.UserName</td>
                <td>@user.Email</td>
                <td>@Html.ActionLink("Update", "Edit", new { id = user.Id })</td>
                <td>@Html.ActionLink("View details", "Details", new { id = user.Id })</td>
            </tr>
        }
    </tbody>
</table>

```

#### Details.cshtml

```

@model lab6.Models.ApplicationUser
@{
    ViewBag.Title = "Details";
}

<h2>Details</h2>

@Html.LabelFor(u => u.Email, "Email:")
<br />
@Html.DisplayFor(u => u.Email)
<br />
@Html.LabelFor(u => u.UserName, "Username:")
<br />
@Html.DisplayFor(u => u.UserName)
<br />
<label>Role:</label>
<p>@ViewBag.UserRole</p>

```

#### Edit.cshtml

```

@model lab6.Models.UserViewModel
@{
    ViewBag.Title = "Edit role of user";
}

<h2>@ViewBag.Title</h2>

```

```
@using (Html.BeginForm(actionName: "Edit", controllerName: "Users", routeValues: new { id =
@Model.User.Id}))
{
    @Html.HttpMethodOverride(HttpVerbs.Put)
    @Html.HiddenFor(m => m.User.Id)
    <br />

    @Html.LabelFor(m => m.RoleName, "Role:")
    <br />
    @Html.EditorFor(m => m.RoleName, new { htmlAttributes = new { @class = "form-control" } })
    <br />
    <br />

    <button class="btn btn-sm btn-success" type="submit">Save changes</button>
}
```