Administracija pomoću rola

Role

- Radi lakše administracije korisnici se grupišu u role
- Umesto autorizacije pojedinačnih korisnika radi se autorizacija role
- Biblioteka Microsoft.AspNetCore.Identity sadrži klasu IdentityRole kojom se predstavlja rola
- Svaka rola ima svoj Id i svoje ime koje se predstavlja posredstvom svojstva Name
- Potrebno je registrovati servis za rad sa rolama čime će biti registrovan i servis RoleManager
- Klasa RoleManager omogući će rad sa rolama, kreiranje, editovanje i brisanje role

Klasa RoleManager

- Metoda **RoleExistsAsync** proverava da li je rola sačuvana u tabeli AspNetRoles i vraća true ako je tačno
- Svojstvo **Roles** objekta klase RoleManager vraća listu IdentityRole objekata koji odgovaraju rolama sačuvanim u bazi podataka
- Metoda **CreateAsync** služi za kreiranje nove role (upis reda u tabelu AspMetRoles) i vraća rezultat koji je referenca tipa IdentityResult
- Metoda **FindByIdAsync** proverava da li u bazi postoji rola sa datim Id-om i ako postoji vraća objekat klase IdentityRole na osnovu podataka iz baze
- Metoda **FindByNameAsync** proverava da li u bazi postoji rola sa datim imenom(svojstvo Name) i ako postoji vraća objekat klase IdentityRole na osnovu podataka iz baze
- Metoda **UpdateAsync** kao ulazni parametar očekuje objekat klase IdentityRole na osnovu koga će biti promenjena rola u bazi sa istim Id atributom kao i prosleđeni objekat
- Metoda **DeleteAsync** kao ulazni parametar očekuje objekat klase IdentityRole na osnovu koga će biti obrisana rola u bazi sa istim Id atributom kao i prosleđeni objekat

Klasa UserManager

- Klasa UserManager osim što služi za kreiranje korisnika aplikacije ima i metode za rad sa rolama
- Klasa **UserManager** ima metodu **IsInRoleAsync** kojom se proverava da li se neki korisnik nalazi u zahtevanoj roli
- Metoda **GetRolesAsync** kao ulazni parametar zahteva objekat klase ApplicationUser i vraća listu rola za datog korisnika

Polazne osnove

- Pretpostavlja se da je realizovana kastomizacija sistema za logovanje kao u prethodnom predavanju
- Korisnik se predstavlja model klasom ApplicationUser
- Kreirane su model klase za logovanje i registraciju: LoginModel i RegisterModel
- Kreirana je baza koja će sadržati tabele Identity sistema (pomoću code-first migracija)
- Kreirana je klasa AccountController sa metodama:
  - Login (GET i POST)
  - Register (GET i POST)
  - SignOut za uništavanje autentifikacionog kukija

Konfiguracija identity sistema

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationDbContext>(options =>
                options.UseSqlServer(

Configuration.GetConnectionString("DefaultConnection")));

            services.AddIdentity<ApplicationUser,IdentityRole>()
                .AddEntityFrameworkStores<ApplicationDbContext>();

            services.AddControllersWithViews();
```

Registracija Identity sistema se vrši korišćenjem metode AddIdentity

Podešavanje opcija za lozinku i korisnika

```
services.Configure<IdentityOptions>(opcije => {

    // Podesavanje lozinke
    opcije.Password.RequireDigit = false;
    opcije.Password.RequiredLength = 3;
    opcije.Password.RequireNonAlphanumeric = false;
    opcije.Password.RequireUppercase = false;
    opcije.Password.RequireLowercase = false;
    opcije.Password.RequiredUniqueChars = 1;

    // Podesavanje korisnika
    opcije.User.RequireUniqueEmail = false;

});
```

Podešavanje autentifikacionog kolačića

```
services.ConfigureApplicationCookie(opcije =>
{
    // Cookie settings
    opcije.Cookie.HttpOnly = true;
    opcije.ExpireTimeSpan = TimeSpan.FromMinutes(5);

    opcije.LoginPath = "/Account/Login";
    opcije.AccessDeniedPath = "/Account/AccessDenied";
    opcije.SlidingExpiration = true;
});
```

Metoda Configure

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
    //endpoints.MapRazorPages();
});
```

Pošto se koristi kastomizovana varijanta implementacije Identity sistema a
ne web razor biblioteka potrebno je zakomentarisati liniju koda kao što je pokazano

Klasa DbInitializer (folder Data)

- Ova klasa služi će za inicijalizaciju baze podataka
- Kreiraćemo asinhronu metodu metodu ove klase **CreateAdmin** koja će služiti za kreiranje korisnika aplikacije po imenu admin ukoliko takav korisnik ne postoji
- Kreiraćemo asinhronu metodu **CreateRoleAdmin** koja će služiti za kreiranje role **admin** ukoliko takva rola ne postoji
- Kreiraćemo statičku asinhronu metodu Seed koja će pozivati prethodne dve metode i koja će korisnika admin ubaciti u rolu admin
- Metoda Seed kao ulazne parametre imaće instance servisa RoleManager i UserManager
- Pošto se asinhrona metoda kod ASP.NET Core 3.1 MVC apliakcije ne može pozvati direktno pri startovanju aplikacije kasnije ćemo kreirati servis u tu svrhu

```
private static async Task<ApplicationUser> CreateAdmin(UserManager<ApplicationUser> um)
{
    ApplicationUser admin = await um.FindByNameAsync("admin");

    if (admin == null)
    {
        //Novi korisnik
        admin = new ApplicationUser
        {
            UserName = "admin",
            FirstName = "Marko",
            LastName = "Markovic"
        };

        string lozinka = "123";

        var rezultat = await um.CreateAsync(admin, lozinka);
        if (rezultat.Succeeded)
        {
            return admin;
        }
        else
        {
            return null;
        }
    }
    else
    {
        //admin vec postoji
        return admin;
    }
}
```

Metoda proverava da li u bazi postoji korisnik pod korisničkim imenom admin.
Ako postoji vraća referencu na objekat tipa ApplicationUser.
Ako ne postoji kreira ga ukoliko je moguće ili vraća null referencu ukoliko to nije moguće.

```
private static async Task<int> CreateRoleAdmin(RoleManager<IdentityRole> rm)
{
    bool roleExist = await rm.RoleExistsAsync("admin");

    if (roleExist)
    {
        return 0;
    }
    else
    {
        IdentityRole rolaAdmin = new IdentityRole("admin");
        var rezultat = await rm.CreateAsync(rolaAdmin);

        if (rezultat.Succeeded)
        {
            return 1;
        }
        else
        {
            return -1;
        }
    }
}
```

Metoda vraća 0 ako rola admin postoji, vraća 1 ukoliko je upravo kreiran i vraća -1 ukoliko je došlo do greške pri pokušaju kreiranja role.

```
public  static async Task Seed(RoleManager<IdentityRole> rm, UserManager<ApplicationUser> um)
{

    int roleCreated = await CreateRoleAdmin(rm);
    ApplicationUser admin = await CreateAdmin(um);

    if (admin != null && roleCreated != -1)
    {
        bool rezultat1 = await um.IsInRoleAsync(admin, "admin");

        if (!rezultat1)
        {
            await um.AddToRoleAsync(admin, "admin");

        }
    }
}
```

Kreiranje servisa za poziv asinhrone metode pri pokretanju aplikacije

```
public class MigratorHostedService : IHostedService
    {
        private readonly IServiceProvider _serviceProvider;
        public MigratorHostedService(IServiceProvider serviceProvider)
        {
            _serviceProvider = serviceProvider;
        }

        public async Task StartAsync(CancellationToken cancellationToken)
        {
            using (var scope = _serviceProvider.CreateScope())
            {
                var services = scope.ServiceProvider;

                var rm = services.GetRequiredService<RoleManager<IdentityRole>>();
                var um = services.GetRequiredService<UserManager<ApplicationUser>>();

                 await DbInitializer.Seed(rm, um);
            }
        }

        public Task StopAsync(CancellationToken cancellationToken)
        {
           return Task.CompletedTask;
        }
    }
```

Kreiranje servisa za poziv asinhrone metode pri pokretanju aplikacije

- Klasa mora da implementira interfejs IHostedService
- Unutar klase definiše se polje tipa IServiceProvider koje se inicijalizuje posredstvom DI kontejnera i koje omogućava pristup registrovanim servisima aplikacije
- Klasa u našem primeru MigratorHostedService mora se registrovati u servisima aplikacije

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser,IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>();

    services.AddControllersWithViews();

    services.AddHostedService<MigratorHostedService>();

    ....
```

Ako se u ovom momentu pokrene aplikacija u bazi će biti kreiran korisnik admin i on će biti ubačen u rolu admin. Odnosno vrši se inicijalizacija baze podataka.

Tabela AspNet.Roles

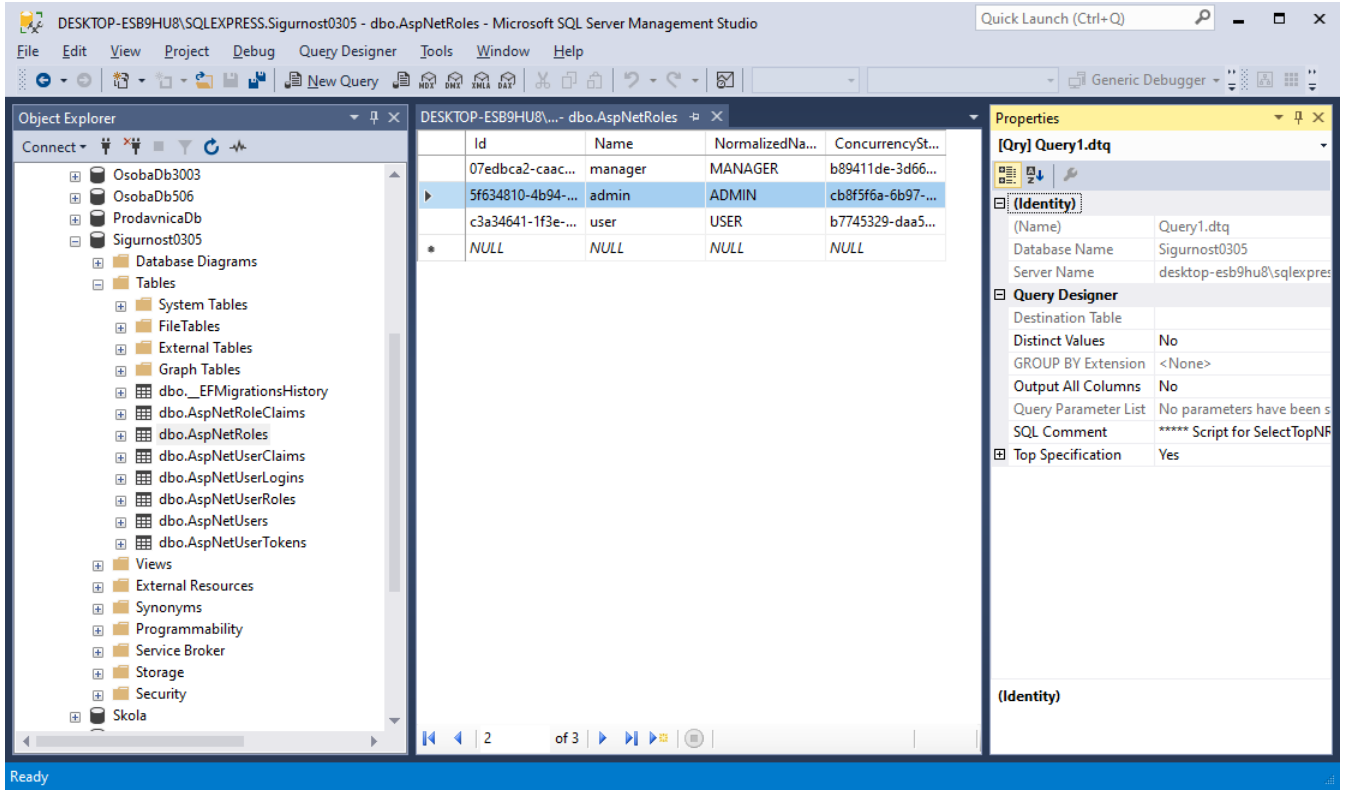
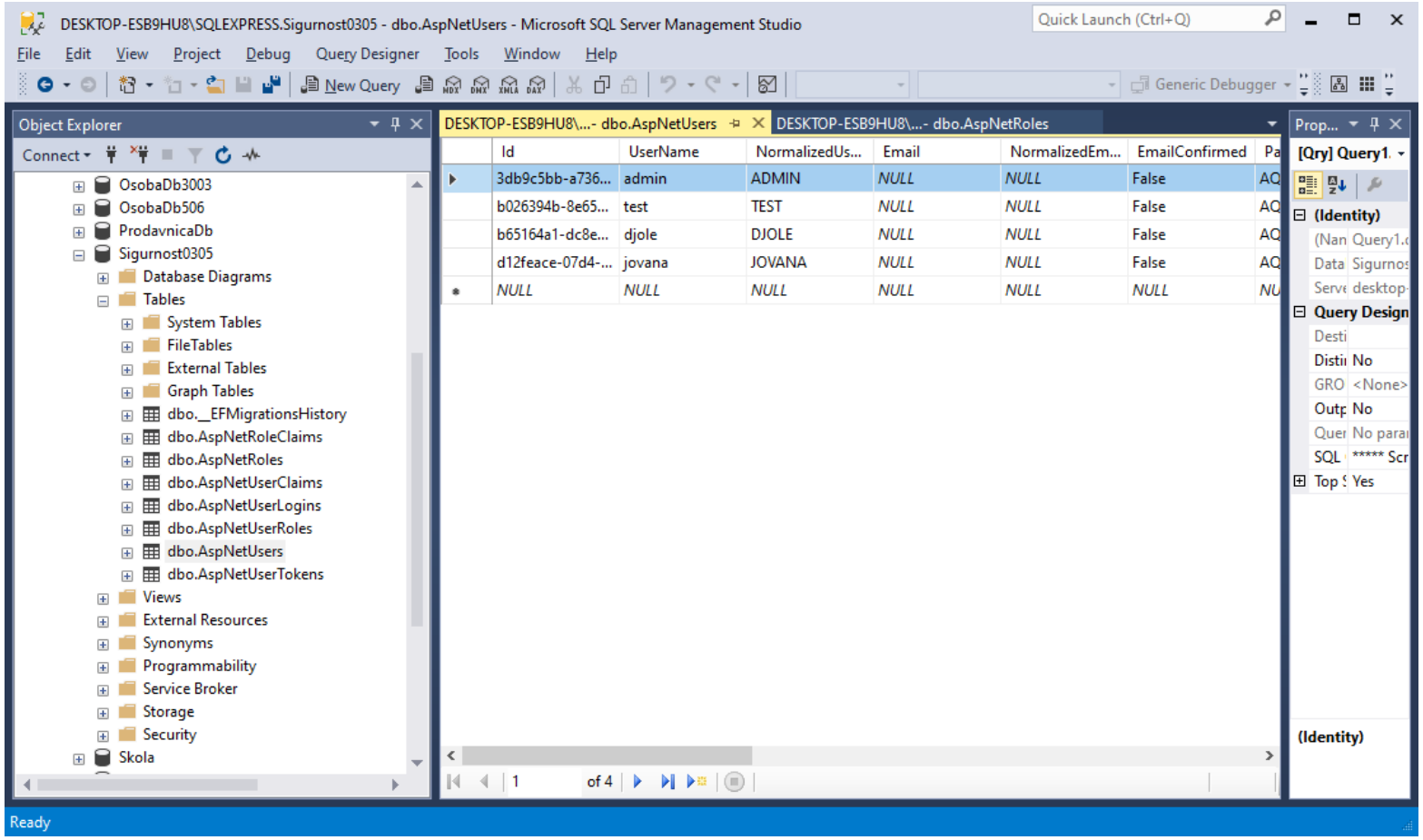
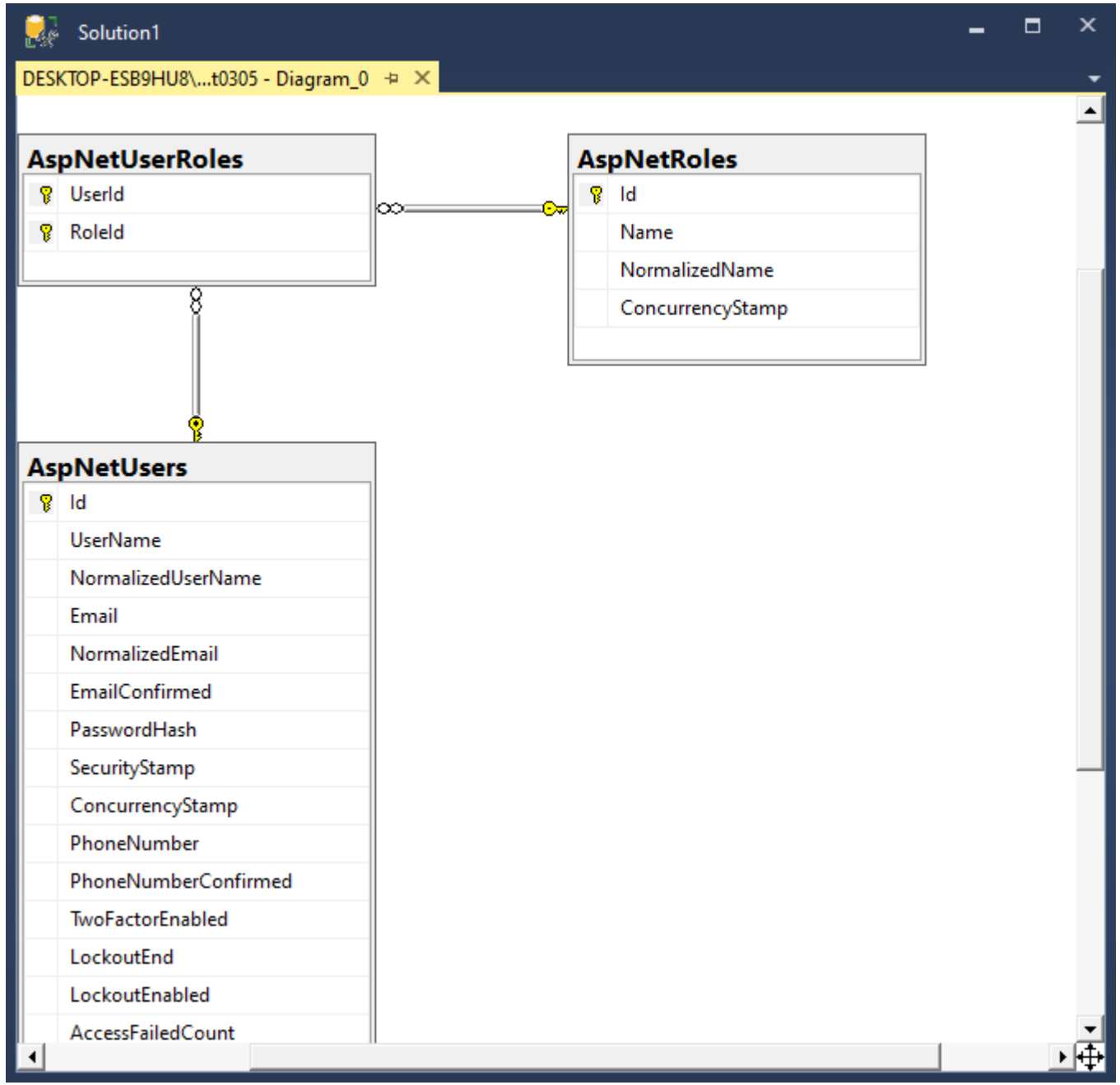


Tabela koja čuva role koje koristi aplikacija

Tabela ASpNetUsers





Veza tabela AspNetRoles i AspNetUsers

Kreiranje kontrolera za rad sa rolama

```
[Authorize(Roles = "admin")]
public class RoleController : Controller
{
    private UserManager<ApplicationUser> um;
    private RoleManager<IdentityRole> rm;


    public RoleController(UserManager<ApplicationUser> _um, RoleManager<IdentityRole> _rm)
    {
        um = _um;
        rm = _rm;
    }

 }
```

Kontroler može pozivati samo autentifikovani korisnik koji je u roli admin

Index akcija kontrolera Role

```
public IActionResult Index()
{
    List<IdentityRole> uloge = rm.Roles.ToList();
    return View(uloge);
}
```

Index pogled kontrolera Role

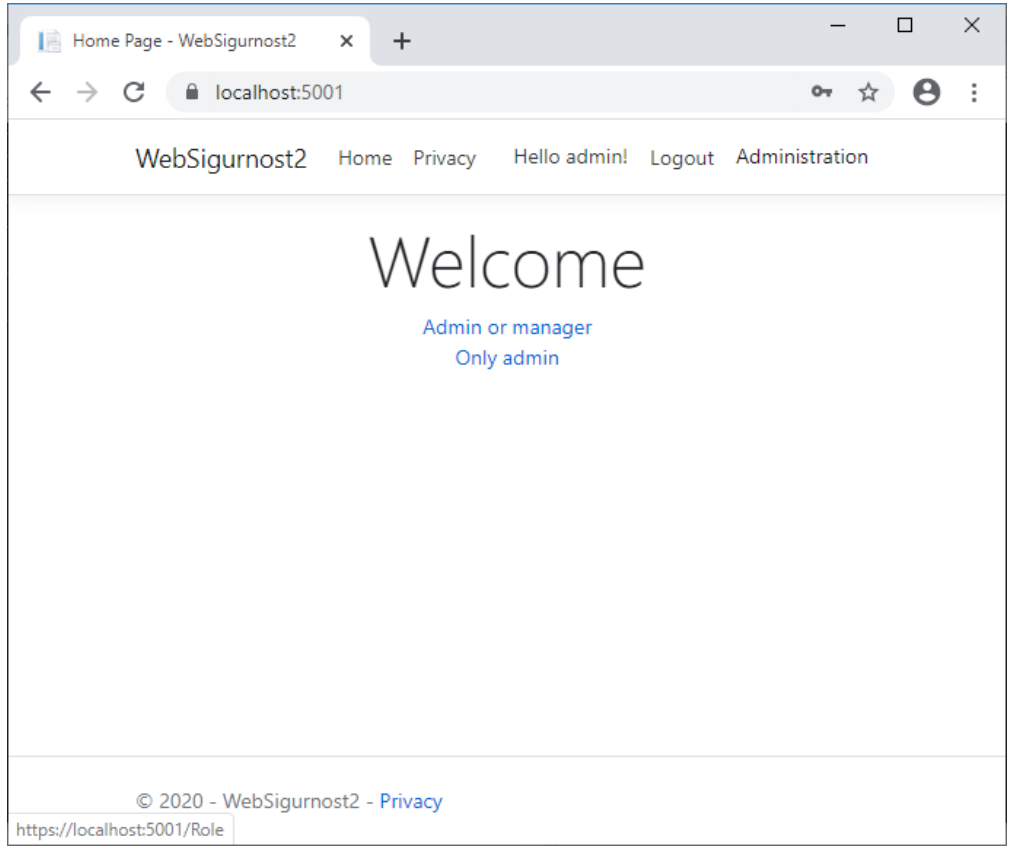
```
@using Microsoft.AspNetCore.Identity
@model IEnumerable<IdentityRole>
@{
    ViewData["Title"] = "Index";
}
<a asp-action="Create">New role</a>
<h2>Index</h2>
<div class="row">
    <table class="table table-bordered table-striped">
        <tr>
            <th>Role name</th>
            <th></th>
        </tr>

        @foreach (var r in Model)
        {
            <tr>
                <td>@r.Name</td>
                <td>
                    <a asp-action="Delete" asp-route-roleName="@r.Name">Delete</a> |
                    <a asp-action="Edit" asp-route-roleName="@r.Name">Edit</a>
                </td>
            </tr>
        }
    </table>
</div>

<a asp-action="AddUserToRole">Add user to role</a>

<h3>@ViewBag.Message</h3>
```

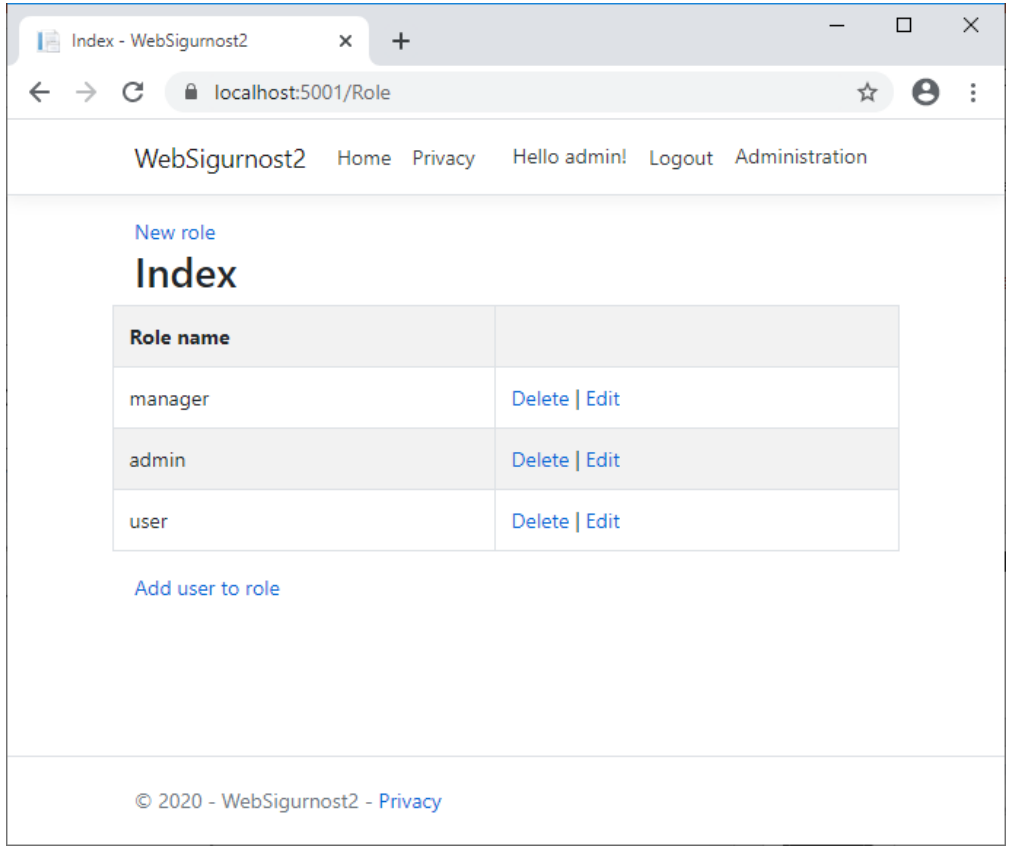
Modifikacija parcijalnog pogleda za logovanje

```
@using Microsoft.AspNetCore.Identity
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager



    <ul class="navbar-nav">
        @if (SignInManager.IsSignedIn(User))
        {
            ApplicationUser au = await UserManager.FindByNameAsync(User.Identity.Name);
            bool isAdmin = await UserManager.IsInRoleAsync(au, "admin");
            <li class="nav-item">
                <a class="nav-link text-dark">Hello @au.UserName!</a>
            </li>
            <li class="nav-item">
                <form class="form-inline" asp-controller="Account" asp-action="Logout"
asp-route-returnUrl="@Url.Action("Index", "Home")">
                    <button type="submit" class="nav-link btn btn-link text-
dark">Logout</button>
                </form>
            </li>
            @if (isAdmin)
            {
                <li class="nav-item">
                    <a class="nav-link text-dark" asp-controller="Role" asp-
action="Index">Administration</a>
                </li>
            }



        }
```

Dodaje se nova stavka menija ukoliko je korisnik koji poziva pogled u roli admin

Administratorski meni i prikaz rola





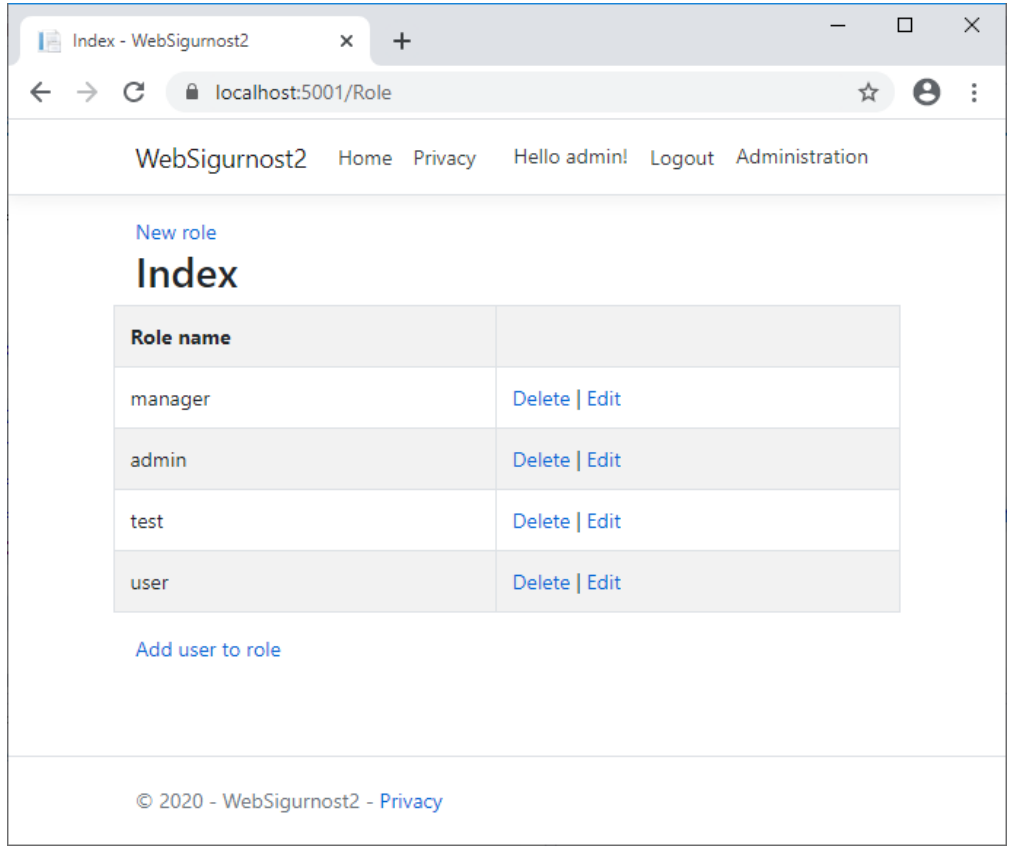
GET i POST metoda CREATE

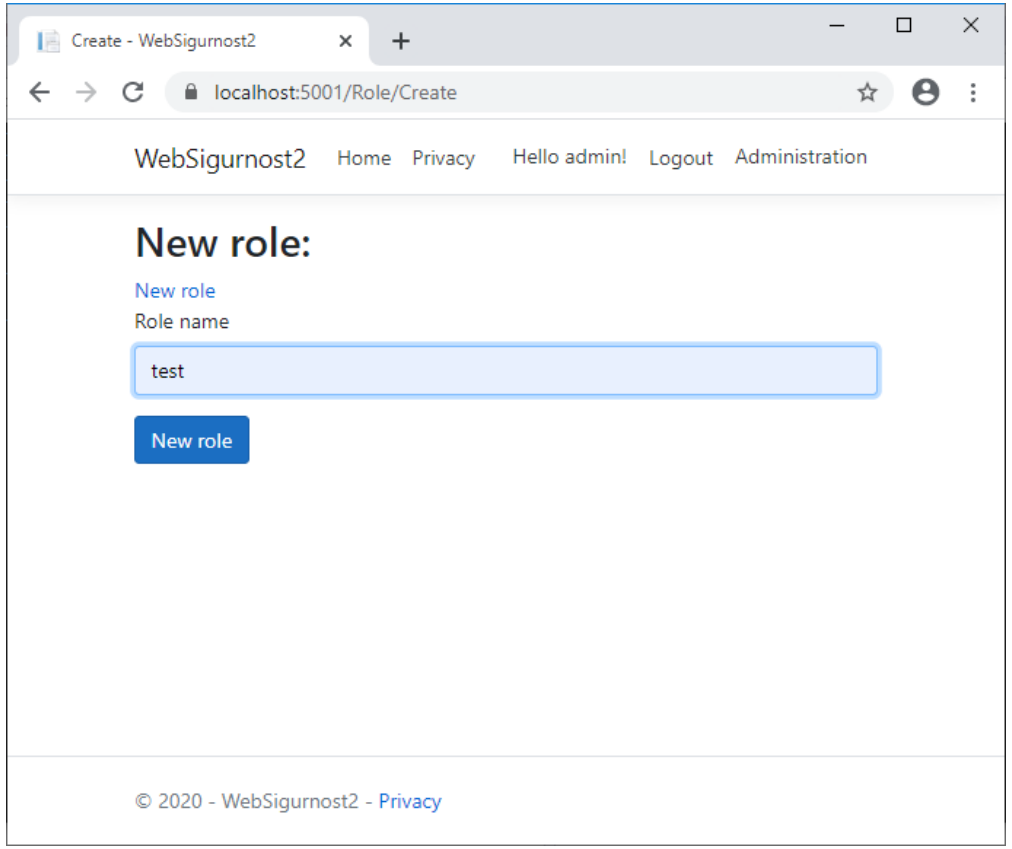
```
public ActionResult Create()
    {
        return View();
    }


    [HttpPost]
    [ValidateAntiForgeryToken]

    public async Task<IActionResult> Create(IdentityRole role)
    {
        var rezultat = await rm.CreateAsync(role);
        if (rezultat.Succeeded)
        {
            return RedirectToAction("Index");
        }
        else
        {
            return View(role);
        }
    }
```

Pogled Create

```
@model IdentityRole
@{
    ViewData["Title"] = "Create";
}

<h2>New role:</h2>
<a asp-action="Create">New role</a>

<form asp-action="Create">
    <div class="form-group">
        <label asp-for="Name">Role name</label>
        <input asp-for="Name" class="form-control" data-val="true" data-val-required="Enter role name" />
        <span asp-validation-for="Name"></span>
    </div>
    <button class="btn btn-primary" type="submit">New role</button>
</form>

@section Scripts{
    <script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>
```

Kreiranje nove role





Metode za editovanje role

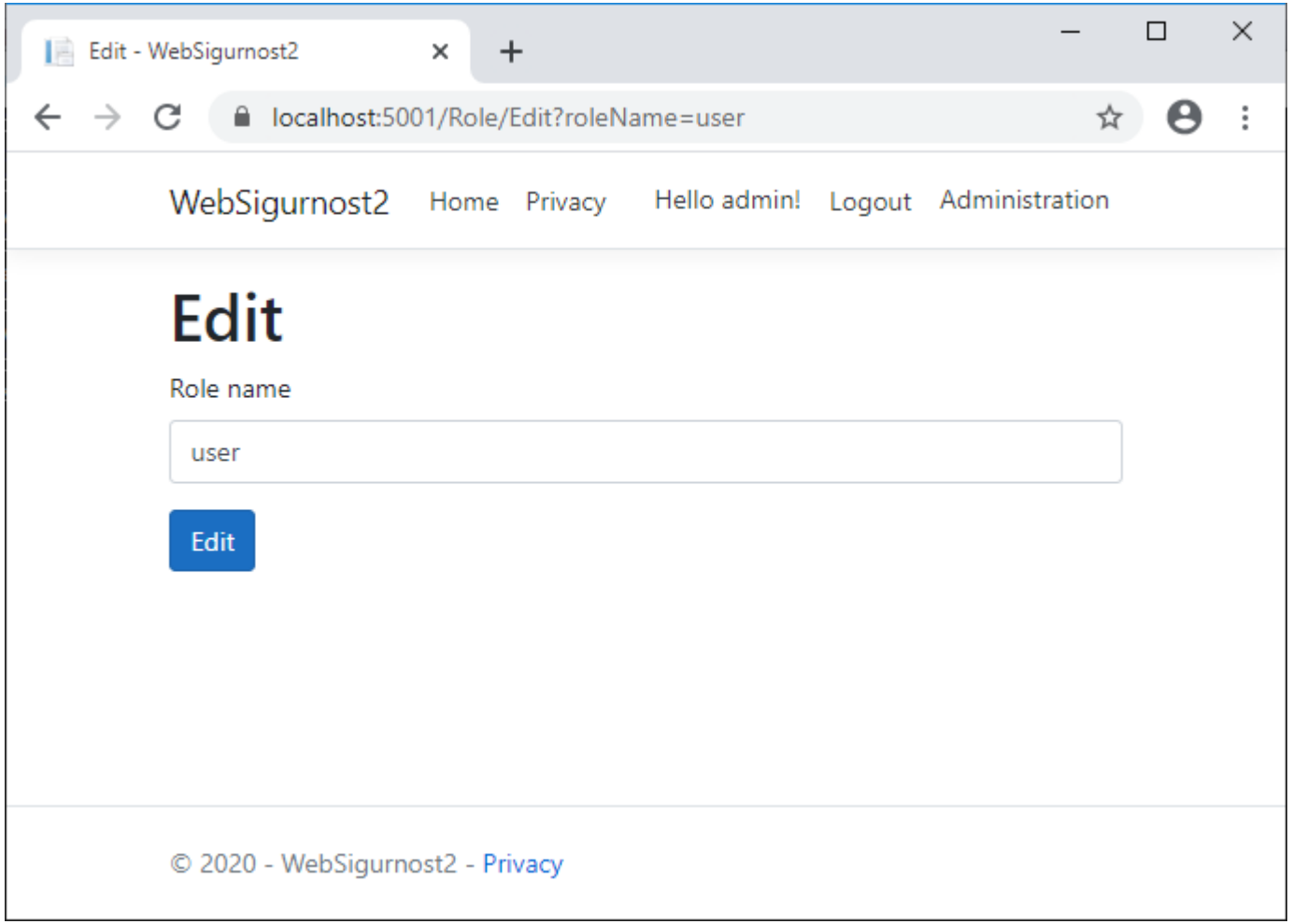
```
public async Task<ActionResult> Edit(string roleName)
{
    IdentityRole role = await rm.FindByNameAsync(roleName);

    return  View(role);
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Edit(IdentityRole role)
{
    IdentityRole r = await rm.FindByIdAsync(role.Id);
    try
    {

        r.Name = role.Name;
        await rm.UpdateAsync(r);
        return RedirectToAction("Index");
    }
    catch (Exception)
    {

        return View(r);
    }
}
```

Pogled za editovanje role

```
@model IdentityRole
@{
    ViewData["Title"] = "Edit";
}

<h1>Edit</h1>


<form asp-action="Edit">
    <div class="form-group">
        <input asp-for="Id" type="hidden" />
        <label asp-for="Name">Role name</label>
        <input asp-for="Name" class="form-control" data-val="true" data-val-required="Enter role name" />
        <span asp-validation-for="Name"></span>
    </div>
    <button class="btn btn-primary" type="submit">Edit</button>
</form>

@section Scripts{
    <script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>
}
```

Editovanje role



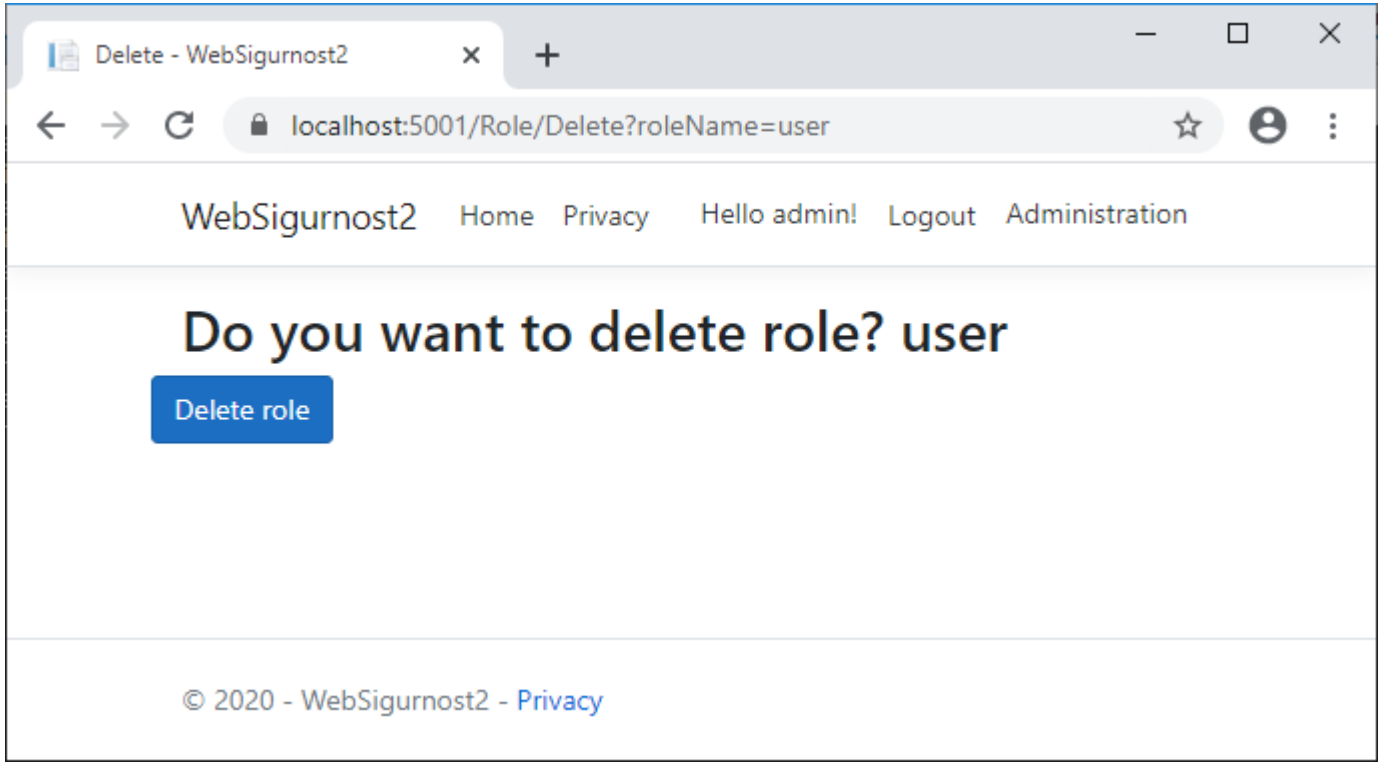
Metode za brisanje role

```
public async Task<IActionResult> Delete(string roleName)
{
    IdentityRole role = await rm.FindByNameAsync(roleName);

    return View(role);
}


[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(string roleName)
{
    IdentityRole role = await rm.FindByNameAsync(roleName);

    var rez = await rm.DeleteAsync(role);


    if (rez.Succeeded)
    {
        return RedirectToAction("Index");
    }

    return View(role);
}
```

Pogled za brisanje role

```
@model IdentityRole
@{
    ViewData["Title"] = "Delete";
}

<h2>Do you want to delete role? @Model?.Name</h2>

<div class="row">
    <form asp-action="Delete">
        <input type="hidden" name="roleName" value="@Model?.Name" />
        <button class="btn btn-primary">Delete role</button>
    </form>

</div>
```

Brisanje role



Metoda AddUserToRole - GET

```
public IActionResult AddUserToRole()
{
    ViewBag.Users = new SelectList(um.Users, "UserName", "UserName");
    ViewBag.Roles = new SelectList(rm.Roles, "Name", "Name");
    return View();
}
```

```
public SelectList (System.Collections.IEnumerable items, string
dataValueField, string dataTextField);
```

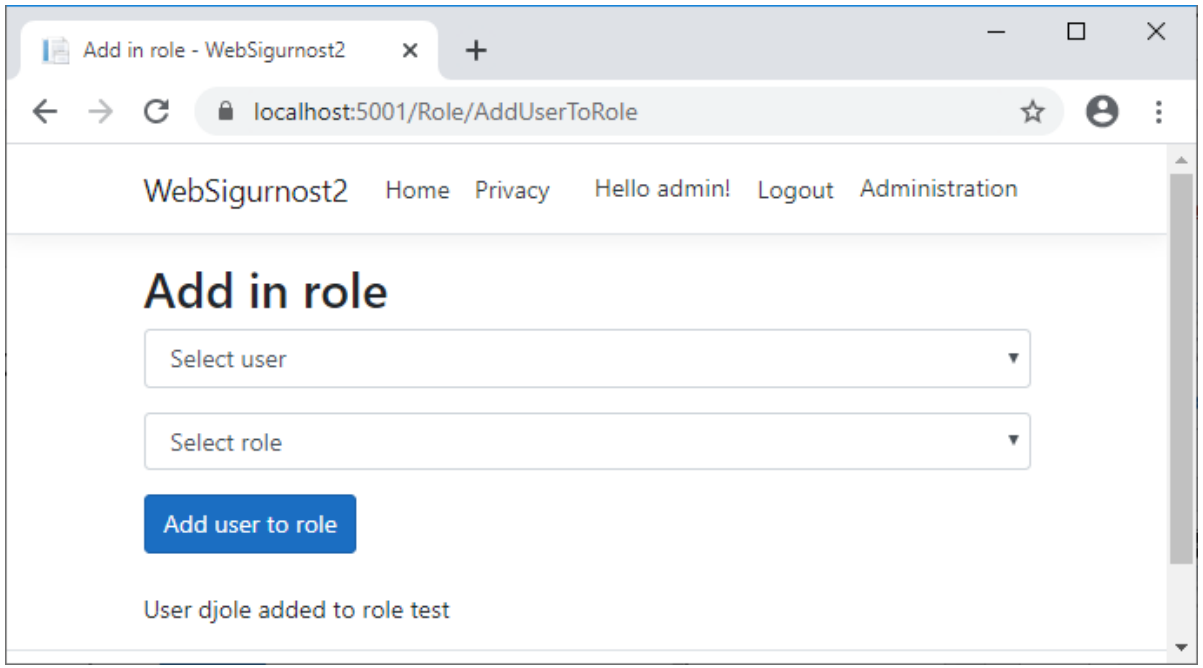
Klasa SelectList služi za popunjavanje select elementa automatski generisanim option elementima.
Instanca klase SelectList prosleđuje se pogledu posredstvom ViewBag polja.
SelectTagHelper pomoću atributa asp-items povezuje select element sa ViewBag poljem.

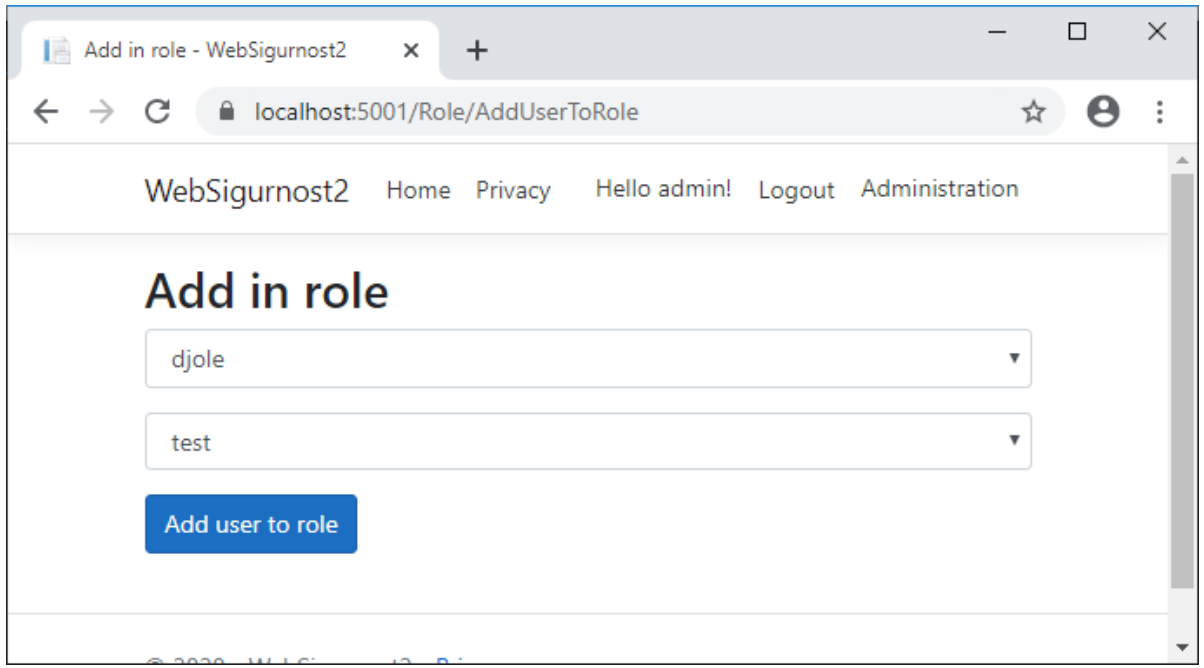
```
<select name="user" asp-items="ViewBag.Users"
class="form-control">
```

Metoda AddUserToRole - POST

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> AddUserToRole(string user, string role)
{
    ViewBag.Users = new SelectList(um.Users, "UserName", "UserName");
    ViewBag.Roles = new SelectList(rm.Roles, "Name", "Name");
    ApplicationUser au = await um.FindByNameAsync(user);

    bool rez1 = await um.IsInRoleAsync(au, role);

    if (rez1)
    {
        ViewBag.Message = $"User {user} is already in {role} role";
        return View();
    }

    var rez2 = await um.AddToRoleAsync(au, role);

    if (rez2.Succeeded)
    {
        ViewBag.Message = $"User {user} added to role {role}";
    }
    else
    {
        ViewBag.Poruka = "Error adding user in role";
    }

    return View();
}
```

Dodavanje korisnika u ulogu





Autorizacija metode bazirano na rolama

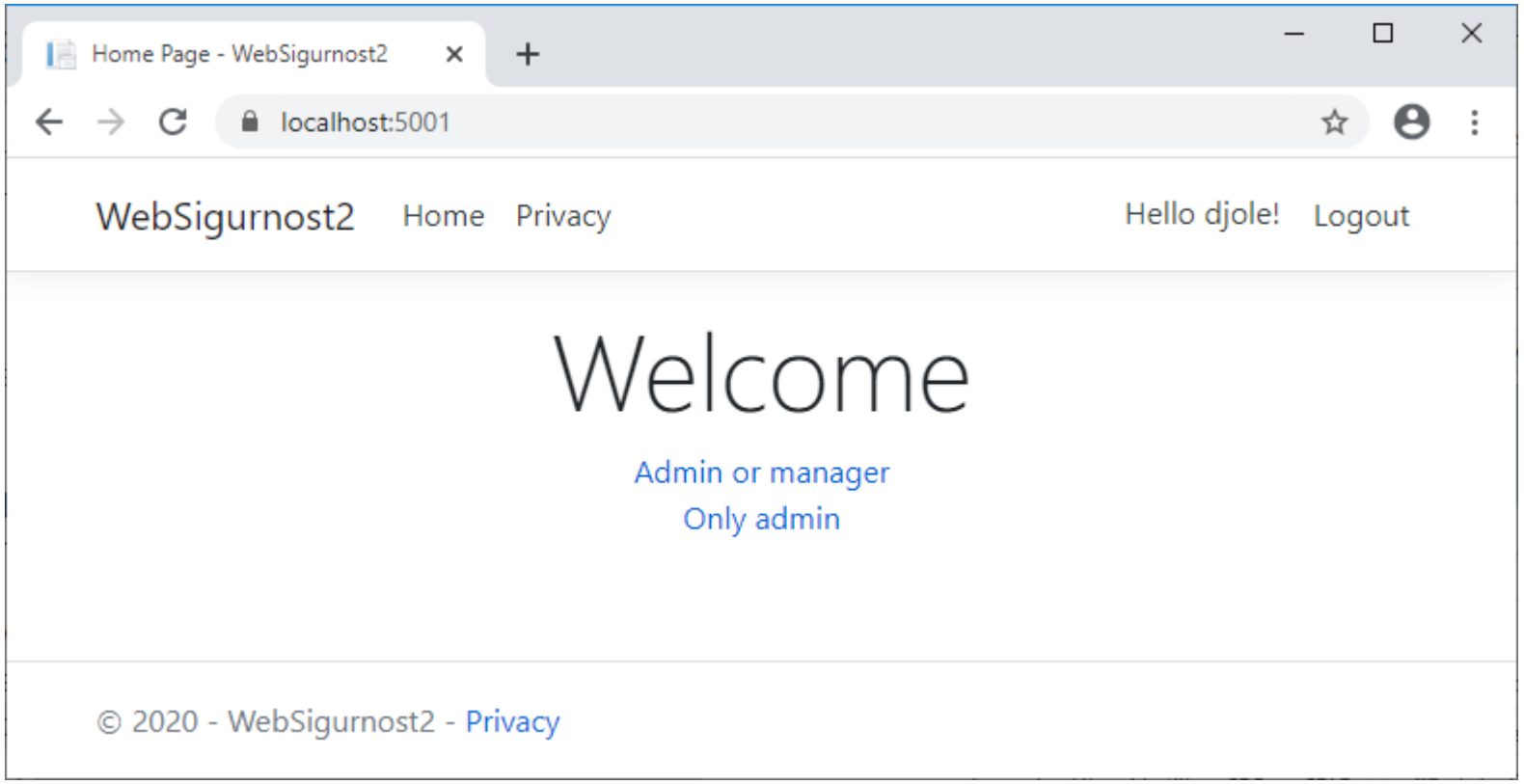
```
[Authorize(Roles = "admin")]
public ActionResult OnlyAdmin()
{
    return View("AdminOrManager");
}

[Authorize(Roles = "admin, manager")]
public IActionResult AdminOrManager()
{

    return View();
}
```

Metodu OnlyAdmin može da poziva samo korisnik u roli admin.
Metodu AdminOrManager može da poziva samo korisnik koji je u roli admin ili manager

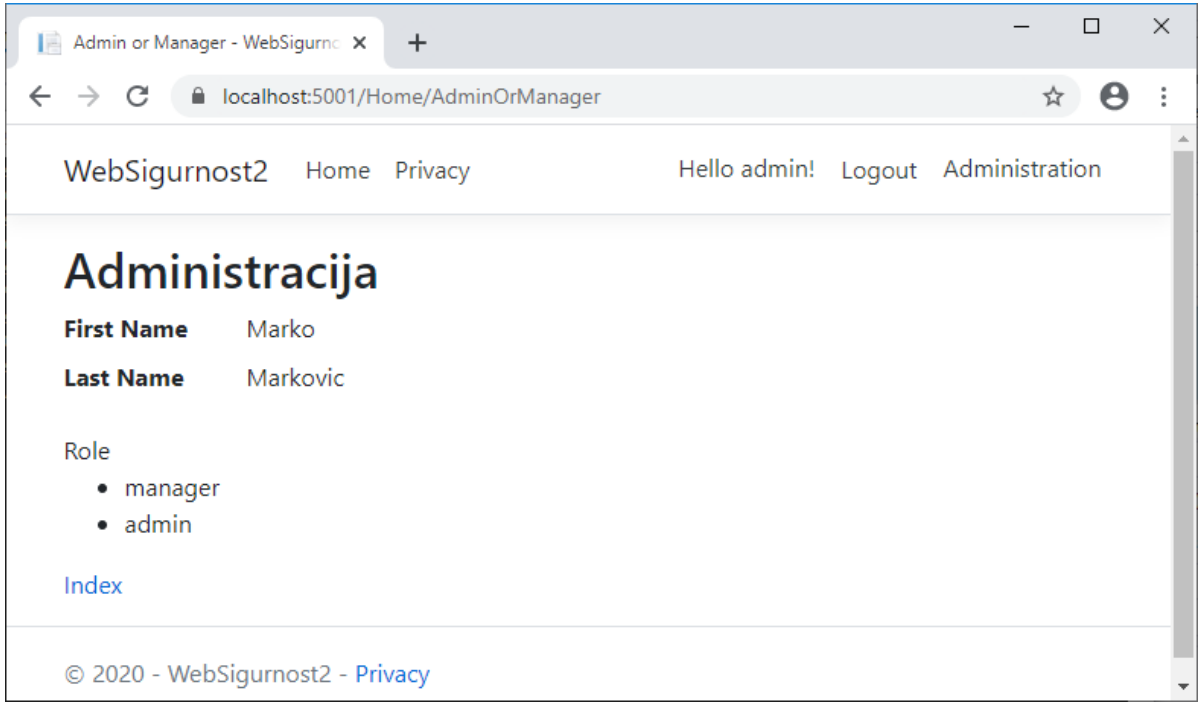
```
@inject UserManager<ApplicationUser> um

@{
    ViewData["Title"] = "Admin or Manager";

    ApplicationUser user = await um.GetUserAsync(User);

    IList<string> roles = await um.GetRolesAsync(user);
}

<h2>@ViewBag.Poruka</h2>

<h2>Administracija</h2>

<dl class="row">
    <dt class="col-sm-2">First Name</dt>
    <dd class="col-sm-10">@user.FirstName</dd>

    <dt class="col-sm-2">Last Name</dt>
    <dd class="col-sm-10">@user.LastName</dd>
</dl>
    Role
    <ul>
        @foreach (var role in roles)
        {
            <li>@role</li>
        }
    </ul>

<a asp-action="Index">Index</a>
```

Svojstvo User unutar cshtml fajla daje informacije o logovanom korisniku

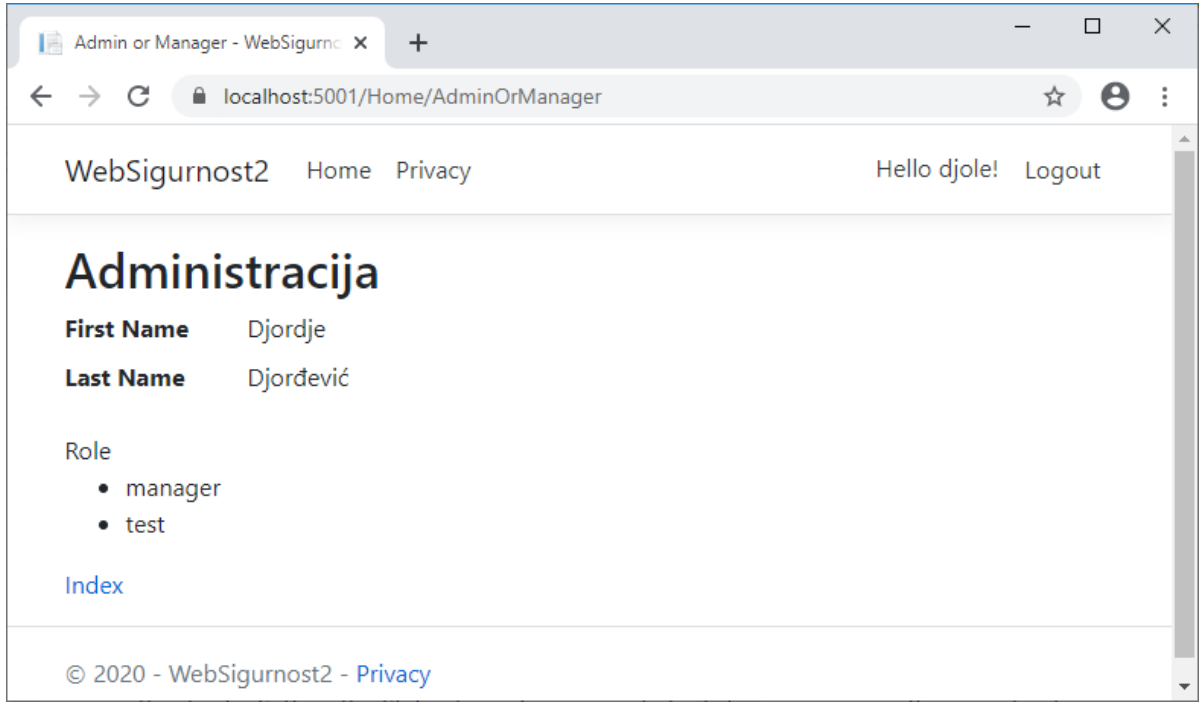
Početna strana aplikacije



Primer korisnika i njima dodeljenih uloga



Korisnik admin je u rolama admin i manager



Korisnik djole je u rolama manager i test

AccountController metoda AccessDenied

```
public IActionResult AccessDenied()
{
    return View();
}
```

Pogled AccessDenied.cshtml

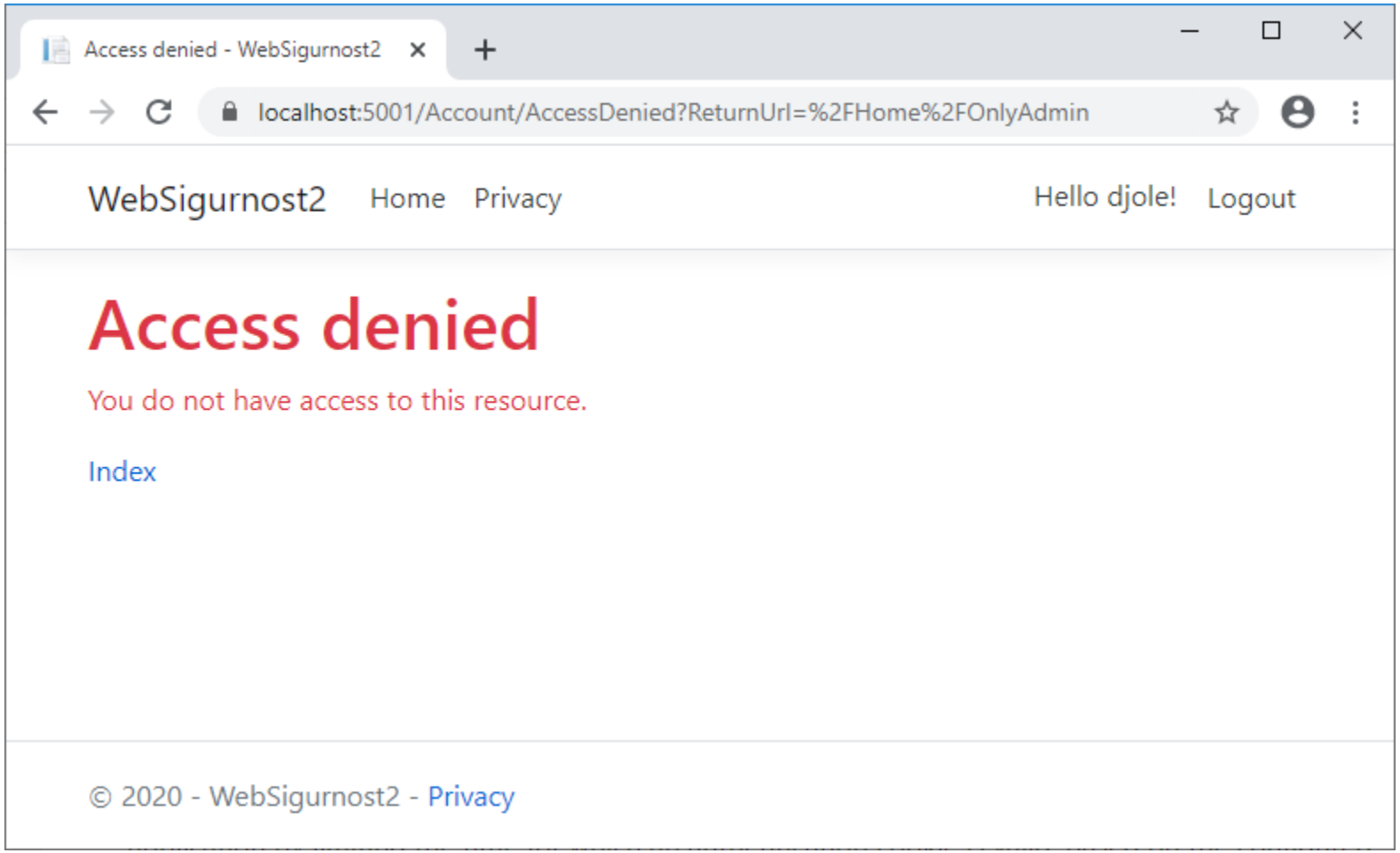
```
@{
    ViewData["Title"] = "Access denied";
}

<header>
    <h1 class="text-
danger">@ViewData["Title"]</h1>
    <p class="text-danger">You do not have
access to this resource.</p>
</header>
<a asp-controller="Home" asp-
action="Index">Index</a>
```

Putanja AccessDenied strane

```
services.ConfigureApplicationCookie(opcije =>
{
    // Cookie settings
    opcije.Cookie.HttpOnly = true;
    opcije.ExpireTimeSpan = TimeSpan.FromMinutes(5);

    opcije.LoginPath = "/Account/Login";
    opcije.AccessDeniedPath = "/Account/AccessDenied";
    opcije.SlidingExpiration = true;
});
```

Strana koja se prikazuje kada korisnik pokuša da pristupi zabranjenom resursu

Korisnik poziva metodu za koju nema dozvolu



Pitanje 1

Identity system kao model klasu za rad sa rolama koristi klasu baziranu na baznoj klasi?

a. UserManager
b. IdentityRole
c. ApplicationUser>

Odgovor: b

Pitanje 2

Neka je ApplicationUser klasa izvedena iz klase IdentityUser. Metoda AddIdentity<ApplicationUser, IdentityRole>() služi za:

a. Definisanje role ApplicationUser
b. Podešavanje middleware komponente za autorizaciju
c. Registraciju servisa koji omogućavaju korišćenje identity sistema za autentifikaciju uključujući i rad sa rolama

Odgovor: c

Pitanje 3

Biblioteka Microsoft.AspNetCore.Identity za rad sa rolama definiše klasu

a. Roles
b. RoleManager
c. ApplicationRole

Odgovor: b

Pitanje 4

Metoda **IsInRoleAsync** kojom se proverava da li se korisnik nalazi u roli, pripada sledećoj klasi iz biblioteke Microsoft.AspNetCore.Identity:

a. UserManager
b. RoleManager
c. ApplicationRole

Odgovor: a

Pitanje 5

Koje se svojstvo koristi pri definisanja autentifikacionog kolačića da bi se specificirala metoda koja se poziva kada korisnik pokuša da pozove zabranjenu metodu:

a. AccessDisabled
b. AccessDeniedPath
c. AccessPath

Odgovor: b

Pitanje 6

Potrebno je dozvoliti da akcionu metodu kontrolera mogu pozivati samo korisnici koju su u roli admin. To postižemo upotrebom sledećeg koda:

a. [Authorize(Roles (admin)]
b. [Authorize(Roles : admin)]
c. [Authorize(Roles = "admin")]

Odgovor: c