

# CompanyEmployee Manager Asp.Net Core MVC

## Deel 4 – Toevoegen van Authenticatie en Autorisatie via ASP.NET Core Identity

In deze les voegen we authenticatie en autorisatie toe. We gaan de ingebouwde ASP.NET Core Identity toevoegen aan het project.

Momenteel is er in het project nog geen security voorzien, we gaan in deze les gebruikers **authenticatie en autorisatie services** toevoegen aan het project. ASP.NET Core Identity ondersteunt de **creatie van user accounts, inloggen en outloggen**. De **user data wordt bijgehouden in een SQL Server Database**. **ASP.NET Core Identity ondersteunt eveneens externe login providers, zoals bv Facebook, Twitter en Microsoft account.**

De eerste stap is is het toevoegen **van ASP.NET Core Identity door installatie van een NuGet package in het MVC project**

### 1 Installatie van de NuGet Package

We gaan de volgende nieuwe pagina maken om een Employee te kunnen aanmaken:  
Installeer de volgende NuGet Package om **ASP.NET Core Identity** te kunnen gebruiken:

```
Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

Controleer of de volgende Packages geïnstalleerd zijn en installeer deze indien nodig:

```
Microsoft.EntityFrameworkCore.SqlServer  
Microsoft.EntityFrameworkCore.Design  
Microsoft.EntityFrameworkCore.Tools
```

# Identity model classes toevoegen

Voeg onder de folder Models van de MVC project een class Register toe:

```
public class Register
{
    [Required]
    [Display(Name = "User Name")]
    public string UserName { get; set; }

    [Required]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [Required]
    [Compare("Password")]
    [Display(Name = "Confirm Password")]
    public string ConfirmPassword { get; set; }

    [Required]
    [Display(Name = "Email")]
    [EmailAddress]
    public string Email { get; set; }
}
```

Voeg onder dezelfde folder Models eveneens een class SignIn toe:

```
public class SignIn
{
    [Required]
    [Display(Name = "User Name")]
    public string UserName { get; set; }

    [Required]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [Required]
    [Display(Name = "Remember Me")]
    public bool RememberMe { get; set; }
}
```

**Bij authenticatie en autorisatie moet de applicatie users en roles kunnen beheren. We gaan een aantal classes toevoegen die hiervoor zullen worden gebruikt**

We gaan **code-first** aanpak gebruiken van EF Core om een database met de users met hun rollen aan te maken.

We hebben dus een nieuwe context class voor de migraties te kunnen opzetten en om de ASP.NET Identity Core datatables te laten creëren via EF Core

**Voeg de volgende class toe onder een nieuwe folder Security**

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
```

```
namespace CompanyEmployees.MVC.Security
{
    public class SecurityDbContext : IdentityDbContext
    {
        public SecurityDbContext(DbContextOptions<SecurityDbContext> options)
        : base(options)
        {
        }
    }
}
```

De class `SecurityDbContext` is een afgeleide class van `IdentityDbContext` uit de namespace `Microsoft.AspNetCore.Identity.EntityFrameworkCore`.

## 2 Connectionstring toevoegen aan appSettings.json

Open `AppSettings.json` van het MVC project. Voeg een Connectionstring voor de database waarin de security-tabellen in zullen worden aangemaakt. We nemen bv dezelfde database, dus dezelfde connectionstring, met de naam `securityConnection`:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "sqlConnection": "server=(localDb)\\mssqlLocalDb;database=CompanyEmployee3; Integrated Security=true",
    "securityConnection": "server=(localDb)\\mssqlLocalDb;database=CompanyEmployee3; Integrated Security=true"
  },
  "AllowedHosts": "*"
}
```

### 3 ASP.NET Core Identity Configuratie toevoegen aan Startup

1. Open de startup.cs class en voeg de volgende lijnen toe aan configureServices()

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<RepositoryContext>(opt =>
    opt.UseSqlServer(Configuration.GetConnectionString("sqlConnection")));
    services.AddScoped<IRepositoryManager, RepositoryManager>();
    services.AddDbContext<SecurityDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("securityConnection")));
    services.AddIdentity<IdentityUser, IdentityRole>()
        .AddEntityFrameworkStores<SecurityDbContext>();
    services.ConfigureApplicationCookie(opt =>
    {
        opt.LoginPath = "/Security/SignIn";
        opt.AccessDeniedPath = "/Security/AccessDenied";
    });

    services.AddControllersWithViews();
}
```

**Services.AddDbContext<SecurityDbContext>** registreert SecurityDbContext class met de DI (Dependency Injection) container van ASP.NET Core

Aangezien we de User en Roles informatie gaan bijhouden in de database, gebruiken we hier eveneens UseSqlServer() met als parameter de connectionstring die we uitlezen uit de AppSettings.json file voor de database

**Services.AppIdentity<IdentityUser, IdentityRole>()** wordt gebruikt om ASP.NET Core Identity services te registreren met de DI container

**AddEntityFrameworkStores:** Zal de nodige implementatie van de Identity data stores toevoegen

**Services.ConfigureApplicationCookie():**

Bij default plaatst ASP.NET Core Identity een cookie. Deze zal bijhouden of de gebruiker geauthenticeerd is. Wanneer een gebruiker (User) niet geauthenticeerd zijn, worden deze doorgestuurd naar de **SignIn()** action method die een SignIn view zal tonen

Er is eveneens een **AccessDenied()** action method die zal worden aangeroepen indien toegang niet kan worden verleend aan de User

2. voeg de volgende lijnen toe aan configure () Volgorde is belangrijk in de request pipeline, dus na `app.UseRouting()`

```
// This method gets called by the runtime. Use this method to configure the
// HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for
        // production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();
    app.UseAuthentication();
    app.UseAuthorization();

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

## 4 Migraties opzetten voor Identity Context db en update database met Identity tabellen

1. Open NuGetPackage Manager Console en voer het volgende commando uit:

```
Add-Migration SecurityInit -context SecurityDbContext
```

Deze zal de Migraties opzetten voor de Identity tabellen (folder Migrations wordt aangemaakt)

2. Voer het volgende commando uit om de ASP.NET Core Identity-tabellen in de database

```
update-database -context SecurityDbContext
```

Controleer in de database dat er een reeks AspNet tabellen zijn aangemaakt:

- [-] CompanyEmployee
  - [+] Database Diagrams
  - [-] Tables
    - [+] System Tables
    - [+] FileTables
    - [+] External Tables
    - [+] ~~dbo. EF.MigrationsHistory~~
    - [+] ~~dbo.AspNetRoleClaims~~
    - [+] ~~dbo.AspNetRoles~~
    - [+] ~~dbo.AspNetUserClaims~~
    - [+] ~~dbo.AspNetUserLogins~~
    - [+] ~~dbo.AspNetUserRoles~~
    - [+] ~~dbo.AspNetUsers~~
    - [+] ~~dbo.AspNetUserTokens~~
    - [+] ~~dbo.Company~~
    - [+] ~~dbo.Employees~~

## 5 Security Controller class toevoegen

We gaan nu nieuwe pagina's maken om een gebruiker te kunnen registreren, in te loggen en een button om uit te loggen.

Hiervoor gaan we eerst een nieuwe controller class toevoegen en de nodige action methoden om te kunnen registreren, inloggen en uitloggen

1. Voeg onder de folder Controllers een SecurityController class toe:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace CompanyEmployees.MVC.Controllers
{
    public class SecurityController : Controller
    {
        private readonly UserManager<IdentityUser> userManager;
        private readonly RoleManager<IdentityRole> roleManager;
        private readonly SignInManager<IdentityUser> signInManager;

        public SecurityController(UserManager<IdentityUser> userManager,
            RoleManager<IdentityRole> roleManager,
            SignInManager<IdentityUser> signInManager)
        {
            this.userManager = userManager;
            this.roleManager = roleManager;
            this.signInManager = signInManager;
        }

        /*      public IActionResult Register()
        {
        }

        [HttpPost]
        public IActionResult Register(Register obj)
        {
        }

        public IActionResult SignIn()
        {
        }

        [HttpPost]
        public IActionResult SignIn(SignIn obj)
        {
        }

        [HttpPost]
        public IActionResult SignOut()
        {
        }

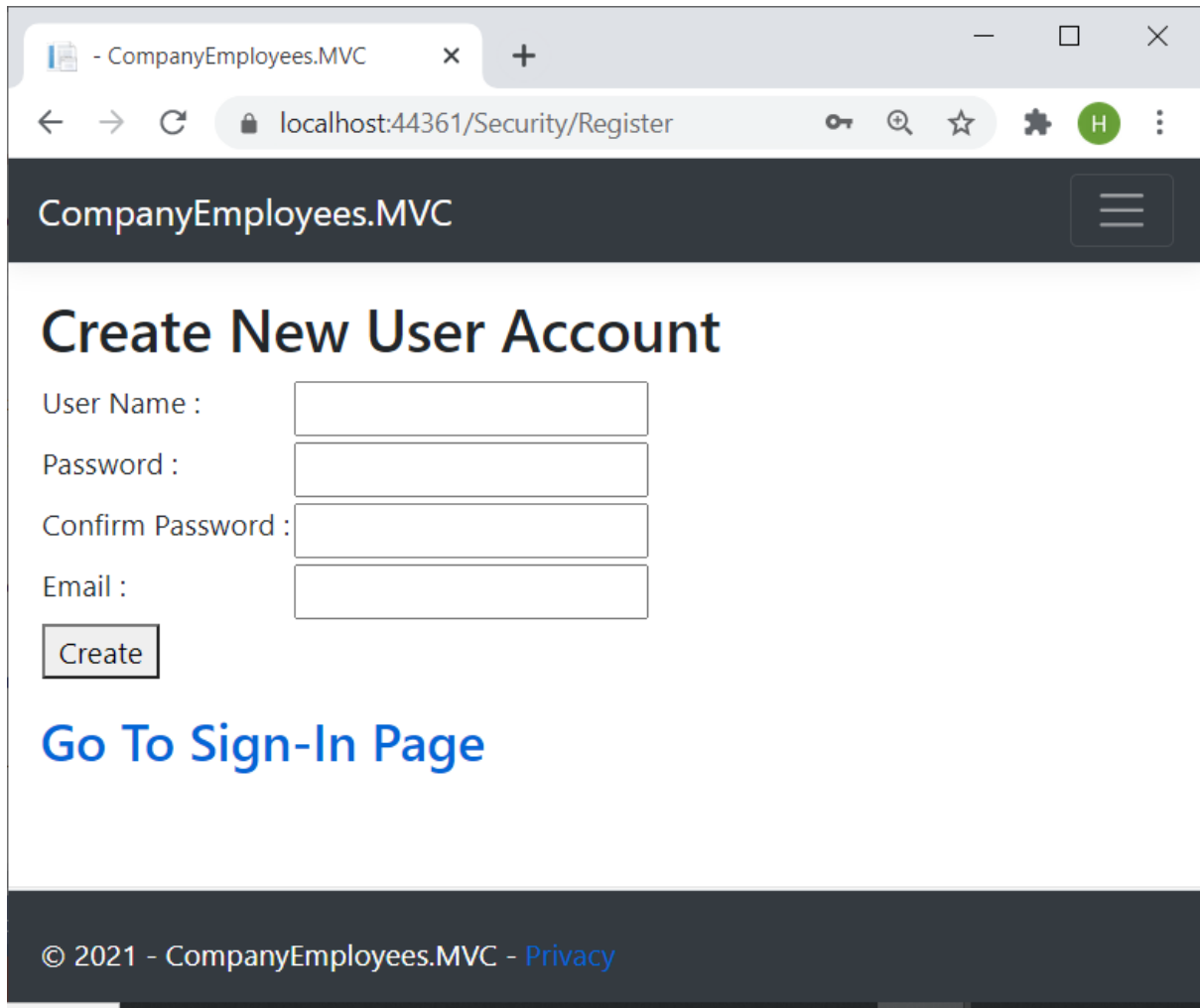
        public IActionResult AccessDenied()
        {
        }

        */
    }
}
```

## 6 Creatie van de User Registration Pagina

We gaan deze Register pagina tonen om een nieuwe user te kunnen registreren.

We gaan hiervoor een Register class onder models, action methods in de Controller class en een Razor View voorzien



The screenshot shows a web browser window with the following elements:

- Browser Tab:** - CompanyEmployees.MVC
- Address Bar:** localhost:44361/Security/Register
- Page Header:** CompanyEmployees.MVC with a hamburger menu icon on the right.
- Main Content:**
  - Title:** Create New User Account
  - Form Fields:**
    - User Name :
    - Password :
    - Confirm Password :
    - Email :
  - Submit Button:** Create
  - Link:** [Go To Sign-In Page](#)
- Page Footer:** © 2021 - CompanyEmployees.MVC - [Privacy](#)



## 1. Register Action methods toevoegen:

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

[HttpPost]
public IActionResult Register(Register obj)
{
    if (ModelState.IsValid)
    {
        if (!roleManager.RoleExistsAsync("Manager").Result)
        {
            IdentityRole role = new IdentityRole();
            role.Name = "Manager";
            IdentityResult roleResult = roleManager.
                CreateAsync(role).Result;
        }

        IdentityUser user = new IdentityUser();
        user.UserName = obj.UserName;
        user.Email = obj.Email;
        user.FullName = obj.FullName;
        user.BirthDate = obj.BirthDate;

        IdentityResult result = userManager.CreateAsync
            (user, obj.Password).Result;

        if (result.Succeeded)
        {
            userManager.AddToRoleAsync(user, "Manager").Wait();
            return RedirectToAction("SignIn", "Security");
        }
        else
        {
            ModelState.AddModelError("", "Invalid user details: " +
result.Errors.ToString());
        }
    }
    return View(obj);
}
```

Wanneer de SecurityController een Register PostHttp request opvangt, zal deze eerst nagaan of er reeds een Manager role in de database bestaat, indien deze niet bestaat, zal deze geregistreerde user de manager role toegekend krijgen.

## 2. Maak een subfolder Security onder Views folder en maak onder Views/Security een Razor View Register.cshtml aan

@model Register

```
<h2>Create New User Account</h2>
<form asp-controller="Security" asp-action="Register" method="post">
    <table>
        <tr>
            <td class="right"><label asp-for="UserName"></label> :</td>
            <td class="left"><input type="text" asp-for="UserName" /></td>
        </tr>
        <tr>
            <td class="right"><label asp-for="Password"></label> :</td>
            <td class="left"><input type="password" asp-for="Password" /></td>
        </tr>
        <tr>
            <td class="right"><label asp-for="ConfirmPassword"></label> :</td>
            <td class="left"><input type="password" asp-for="ConfirmPassword" /></td>
        </tr>
        <tr>
            <td class="right"><label asp-for="Email"></label> :</td>
            <td class="left"><input type="text" asp-for="Email" /></td>
        </tr>
        <tr>
            <td colspan="2">
                <button type="submit">Create</button>
            </td>
        </tr>
    </table>
    <div asp-validation-summary="All" class="message"></div>

    <h3><a asp-controller="Security" asp-action="SignIn">Go To Sign-In Page</a></h3>
</form>
```

## 7 Creatie van Sign-In Pagina

### 1. SignIn class action methods toevoegen aan SecurityController

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

[HttpPost]
public IActionResult SignIn(SignIn obj)
{
    if (ModelState.IsValid)
    {
        var result = signinManager.PasswordSignInAsync
            (obj.UserName, obj.Password,
            obj.RememberMe, false).Result;

        if (result.Succeeded)
        {
            return RedirectToAction("Index", "EmployeeManager");
        }
        else
        {
            ModelState.AddModelError("", "Invalid user details");
        }
    }
    return View(obj);
}
```

## 2. SignIn razor view SignIn.cshtml toevoegen onder folder Views/Security

@model SignIn

```
<h2>Sign In</h2>
<form asp-controller="Security" asp-action="SignIn" method="post">
    <table>
        <tr>
            <td class="right"><label asp-for="UserName"></label> :</td>
            <td class="left"><input type="text" asp-for="UserName" /></td>
        </tr>
        <tr>
            <td class="right"><label asp-for="Password"></label> :</td>
            <td class="left"><input type="password" asp-for="Password" /></td>
        </tr>
        <tr>
            <td class="right"><label asp-for="RememberMe"></label> :</td>
            <td class="left"><input type="checkbox" asp-for="RememberMe" /></td>
        </tr>
        <tr>
            <td colspan="2">
                <button type="submit">Sign In</button>
            </td>
        </tr>
    </table>
    <div asp-validation-summary="All" class="message"></div>

    <h3><a asp-controller="security" asp-action="register">Create New User
Account</a></h3>

</form>
```

## 8 Sign-out button toevoegen

We gaan eveneens op alle views voor de ingelogde gebruikers een button "Sign out" voorzien om te kunnen uitloggen (Signout)

### 1. Open Shared/\_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - CompanyEmployees.MVC</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
    <!-- <link rel="stylesheet" href="~/css/TableStyles.css" />-->
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-dark
border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">CompanyEmployees.MVC</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
```

```

        aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
between">
        <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
                <a class="nav-link text-white" asp-area="" asp-
controller="Home" asp-action="Index">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-white" asp-area="" asp-
controller="Home" asp-action="Privacy">Privacy</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-white" asp-area="" asp-
controller="EmployeeManager" asp-action="Index">Employees</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-white" asp-area="" asp-
controller="CompanyManager" asp-action="Index">Companies</a>
            </li>
        </ul>
    </div>
</div>
</nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>
<br />
<hr />

@if (User.Identity.IsAuthenticated)
{
    <h2>You are signed in as @User.Identity.Name</h2>
    <form asp-controller="Security" asp-action="SignOut" method="post">
        <button type="submit">Sign Out</button>
    </form>
}

<footer class="border-top footer text-muted bg-dark">
    <div class="container text-white">
        &copy; 2021 - CompanyEmployees.MVC - <a asp-area="" asp-controller="Home"
asp-action="Privacy">Privacy</a>
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>

```

## 2. SignOut action method toevoegen aan Security Controller

```
[HttpPost]
public IActionResult SignOut()
{
    signInManager.SignOutAsync().Wait();
    return RedirectToAction("SignIn", "Security");
}
```

## 9 Access Denied Pagina

We voegen eveneens een AccessDenied() action methode en Razor View toe die zal worden getoond indien er geen toegang kan worden verleend aan een gebruiker:

### 1. Voeg de volgende action methode toe aan SecurityController class

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

### 2. Voeg onder Views/Security de Razor view AccessDenied.cshtml toe

```
<h2 class="message">
    There was an unexpected error while signing in to the system.
</h2>
```

## 10 Authenticate en authorize users

We gaan nu autorisaties toekennen in de EmployeeManagerController.cs

We willen dat alle action methoden, (behalve List()) enkel mogen worden aangeroepen door geauthenticeerde Users met de Role="Manager"

Plaats in de lijn boven EmployeeManagerController het attribuut

```
[Authorize(Roles = "Manager")]
public class EmployeeManagerController : Controller

    [AllowAnonymous]
    public IActionResult Index()
```

[AllowAnonymous] boven de action method Index() zorgt ervoor dat iedereen de Index view met de tabel van alle Employees zal kunnen bekijken.

## 11 Beveilig de applicatie tegen Cross-site Request Forgery attacks (CSRF attacks)

Cross-site request forgery attack gebeurt wanneer een 'kwaadaardige' web app die in dezelfde browser runt als een door de gebruiker 'trusted' web app. Deze stuurt dan CRUD request operaties (doet zich voor als de trusted web applicatie), zonder dat de gebruiker er zich van bewust is.

ASP.Net Core voorziet een beveiliging tegen CSRF attacks via een hidden form field die automatisch wordt aangemaakt via een Form Tag Helper. De form field wordt antiforgery token genoemd. Om zeker te zijn dat de een POST request niet van een kwaadaardige web app kan komen, wordt het attribuut voor elke HttpPost action method geplaatst:

`[ValidateAntiForgeryToken]`

**Voeg in EmployeeManagerController onder elke [HttpPost] attribuut deze attribuut toe**

`[ValidateAntiForgeryToken]`

## 12 Run en test applicatie

Bij de Registratiepagina, let erop dat je een strong paswoord invult in de Registratie pagina

<https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-3.1>

<https://www.dotnetcurry.com/aspnet/1343/aspnet-core-csrf-antiforgery-token>

<http://www.binaryintellect.net/articles/d5ede370-a1d6-4b1f-9afb-10422bccfa7c.aspx>