Employee Manager Asp.Net Core MVC

Deel 3 – Toevoegen van Authenticatie en Autorisatie via ASP.NET Core Identity

Start-project: <https://github.com/SyntraWestKortrijk/EmployeeManager.Mvc-DEEL2>

Source code: <https://github.com/SyntraWestKortrijk/EmployeeManager.Mvc--DEEL3>

Youtube videos :

<https://youtu.be/iBCsF6IBgiw>   
(Installeren ASP.NET Core Identity, opzetten migration via ef core)

<https://youtu.be/kaCWzV94XR4> (Toevoegen van Register, SignIn, SignOut, AccessDenied functionaliteit en razor views)

In deze video voegen we authenticatie en authorisatie 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 video gebruikers **authenticatie en authorisatie 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 de NuGet package

# 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

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Design

Microsoft.EntityFrameworkCore.Tools

Microsoft.VisualStudio.Web.CodeGeneration.Design

Identity model classes toevoegen

**Voeg onder models 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; }

[Required]

[Display(Name = "Full Name")]

public string FullName { get; set; }

[Required]

[Display(Name = "Birth Date")]

public DateTime BirthDate { get; set; }

}

**Voeg onder 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 authorisatie moet de applicatie users en roles kunnen beheren. We gaan een aantal classes toevoegen die hiervoor zullen worden gebruikt**

# Toevoegen van classes voor User en Roles

**Maak een nieuwe folder Security aan in de root van het project en maak hierin de volgende classes voor het bijhouden van de details van een User en een Role:**

public class AppIdentityUser : IdentityUser

{

public string FullName { get; set; }

public DateTime BirthDate { get; set; }

}

public class AppIdentityRole : IdentityRole

{

public string Description { get; set; }

}

De AppIdentityUser en AppIdentityRole informatie zal worden bijgehouden in de Northwind database.

De class AppIdentityUser zal de gegevens bijhouden van de ingelogde gebruiker en de AppIdentityRole zal de rol (bv Administrator of Manager) bijhouden van de ingelogde gebruiker. Het is mogelijk om meer informatie bij te houden, indien nodig, zoals bv FirstName, LastName,…

We gaan **code-first** aanpak gebruiken van EF Core om de nieuwe toe te voegen aan Northwind

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 de folder Security**

public class AppIdentityDbContext : IdentityDbContext<AppIdentityUser, AppIdentityRole, string>

{

public AppIdentityDbContext

(DbContextOptions<AppIdentityDbContext> options)

: base(options)

{

}

}

**De class** AppIdentityDbContext **is gelijkaardig aan de AppDbContext, maar deze is een afgeleide class van IdentityDbContext<TUser,TRole,Tkey> uit de namespace** Microsoft.AspNetCore.Identity.EntityFrameworkCore.

**TUser** parameter: duidt het type aan dat zal worden gebruikt voor de Users: AppIdentityUser

**TRole**: duidt het type aan voor de Roles: AppIdentityRole

**TKey**: duidt het type aan van de PK voor de users en roles (string in ons geval)

# 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.AddControllersWithViews();

services.AddDbContext<AppDbContext>(options => options.UseSqlServer(this.config.GetConnectionString("AppDb")));

**services.AddDbContext<AppIdentityDbContext>(options => options.UseSqlServer(this.config.GetConnectionString("AppDb")));**

**services.AddIdentity<AppIdentityUser, AppIdentityRole>()**

**.AddEntityFrameworkStores<AppIdentityDbContext>();**

**services.ConfigureApplicationCookie(opt =>**

**{**

**opt.LoginPath = "/Security/SignIn";**

**opt.AccessDeniedPath = "/Security/AccessDenied";**

**});**

**}**

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

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

**Services.AppIdentity<TUser,TRole>()** 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()**

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles();

app.UseRouting();

**app.UseAuthentication();**

**app.UseAuthorization();**

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=EmployeeManager}/{action=List}/{id?}");

});

}

}

# Migrations opzetten voor Identity Context db en update database met Identity tabellen

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

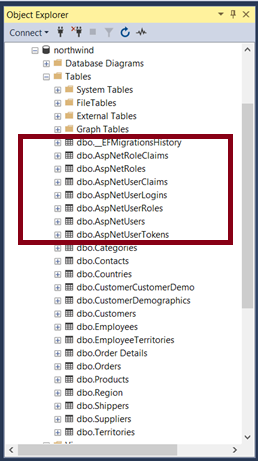
**Add-Migration IdentityMigration -context AppIdentityDbContext**

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

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

**update-database -context AppIdentityDbContext**

Controleer in de northwind database dat de tabellen zijn aangemaakt:

****

# 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:**

public class SecurityController : Controller

{

private readonly UserManager<AppIdentityUser> userManager;

private readonly RoleManager<AppIdentityRole> roleManager;

private readonly SignInManager<AppIdentityUser> signinManager;

public SecurityController(UserManager<AppIdentityUser> userManager, RoleManager<AppIdentityRole> roleManager,

SignInManager<AppIdentityUser> 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()

{

}

\*/

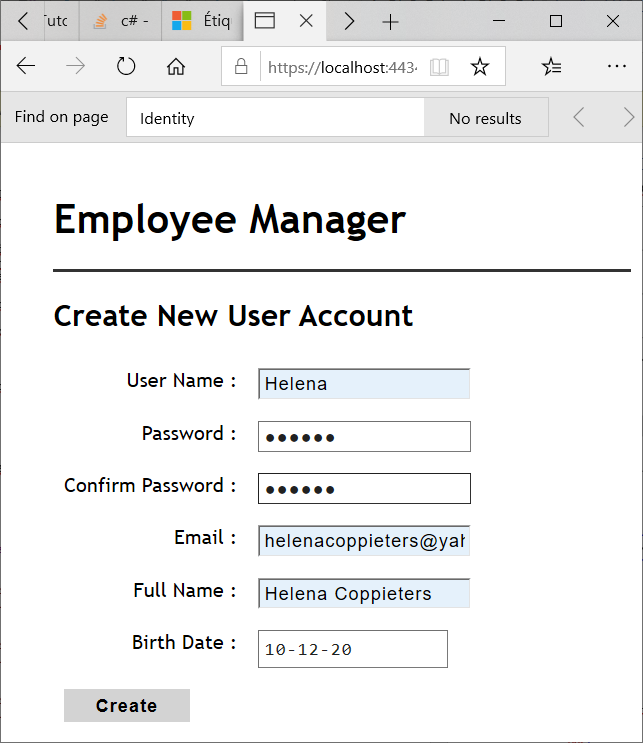
}

}

# Creatie van de User Registration Pagina

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

Zoals in de vorige video’s, gaan we hiervoor een Register class onder models, action methods in de Controller class en een Razor View voorzien



1. **Voeg de class Register toe onder de folder models:**

namespace EmployeeManager.Mvc.Models

{

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; }

[Required]

[Display(Name = "Full Name")]

public string FullName { get; set; }

[Required]

[Display(Name = "Birth Date")]

public DateTime BirthDate { get; set; }

}

}

1. **Register Action methods toevoegen:**

public IActionResult Register()

{

return View();

}

[HttpPost]

public IActionResult Register(Register obj)

{

if (ModelState.IsValid)

{

**if (!roleManager.RoleExistsAsync("Manager").Result)**

**{**

**AppIdentityRole role = new AppIdentityRole();**

**role.Name = "Manager";**

**role.Description = "Can perform CRUD operations.";**

**IdentityResult roleResult = roleManager.**

**CreateAsync(role).Result;**

**}**

AppIdentityUser user = new AppIdentityUser();

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.**

1. **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 class="right"><label **asp-for**="FullName"></label> :</td>

<td><input **type**="text" **asp-for**="FullName" /></td>

</tr>

<tr>

<td class="right"><label **asp-for**="BirthDate"></label> :</td>

<td class="left"><input **type**="date" **asp-for**="BirthDate" /></td>

</tr>

<tr>

<td colspan="2">

<button type="submit">Create</button>

</td>

</tr>

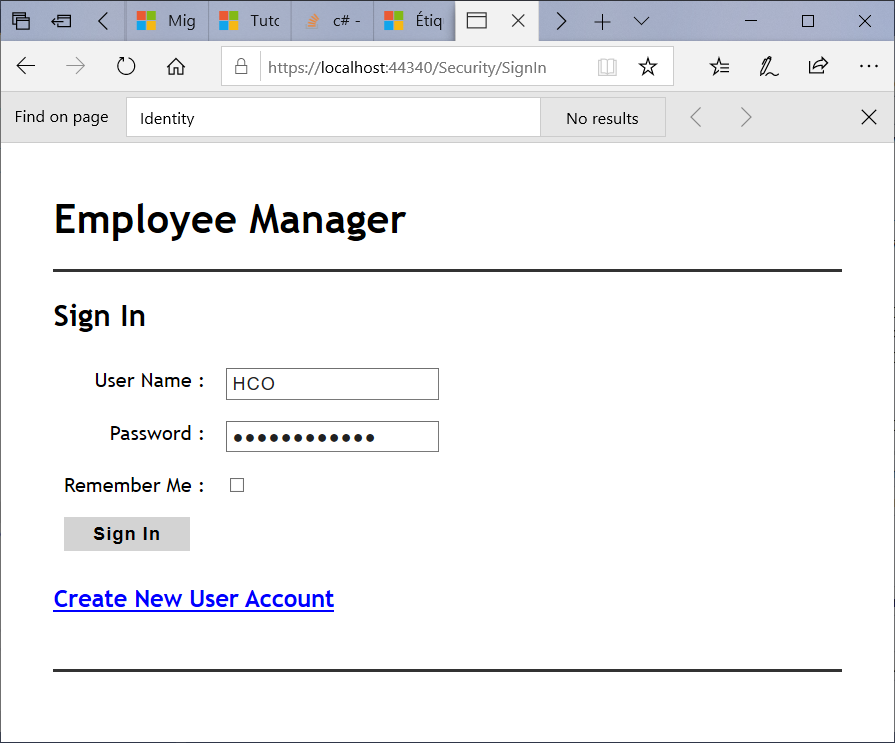
</table>

<div **asp-validation-summary**="All" class="message"></div>

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

</form>

# Creatie van Sign-In Pagina



1. **SignIn class toevoegen onder folder models**

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; }

}

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("List", "EmployeeManager");

}

else

{

ModelState.AddModelError("", "Invalid user details");

}

}

return View(obj);

}

1. **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>

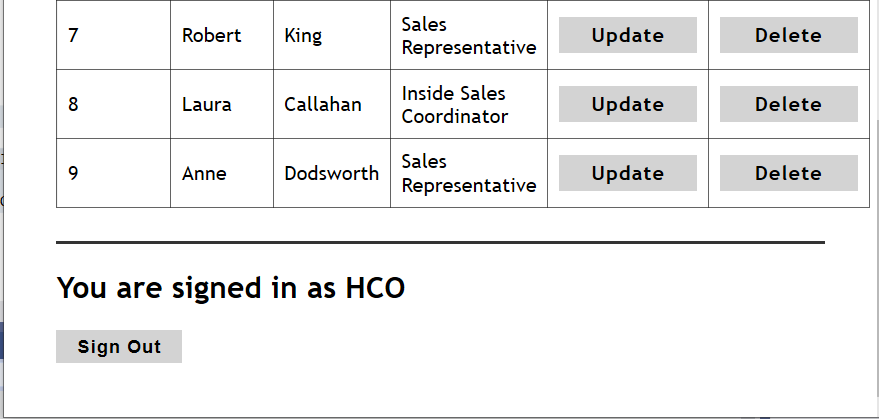
<h3><a **asp-controller**="security" **asp-action**="register">Create New User Account</a></h3>

</form>

# 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>

<head>

<meta name="viewport" content="width=device-width" />

<title>Employee Manager</title>

<link href="~/Styles/site.css" rel="stylesheet" />

<script src="~/Scripts/jquery.js"></script>

<script src="~/Scripts/jquery.validate.js"></script>

<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>

</head>

<body>

<h1>

Employee Manager

</h1>

<hr />

<div>

@RenderBody()

</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>

}

</body>

</html>

1. **SignOut action method toevoegen aan Security Controller**

[HttpPost]

public IActionResult SignOut()

{

signinManager.SignOutAsync().Wait();

return RedirectToAction("SignIn", "Security");

}

# 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();

}

1. **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>

# Authenticate en authorise users

**We gaan nu authorisaties 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 List()

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

# 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 kwaardaardige 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]

# Run en test applicatie

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

# Beschermen tegen CSRF Attacks

## CSRF attack

**Voorbeeld van CSRF attack van ‘kwaadaardige web app’ (bv www.bad-crook-site.com) met deze form:**

<h1>Congratulations! You're a Winner!</h1>

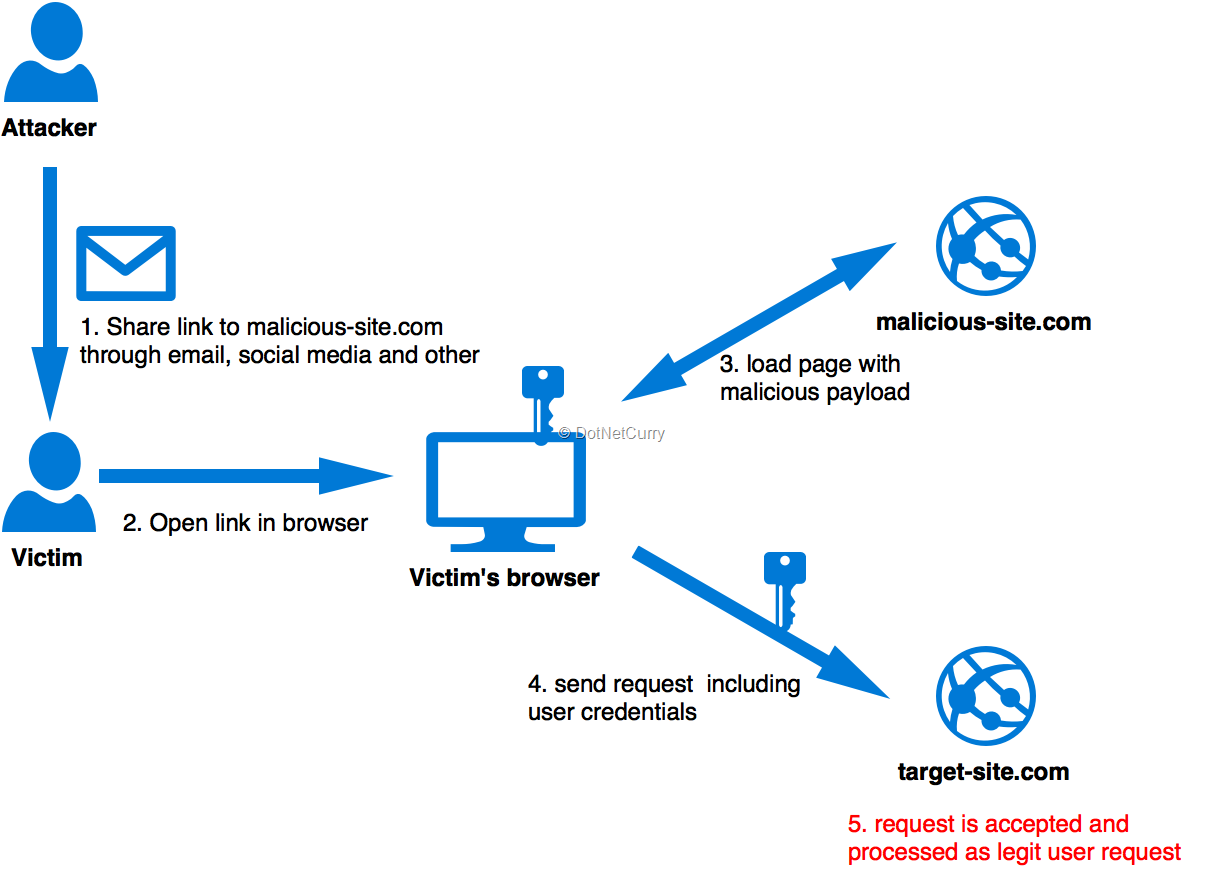
<form action="http://good-banking-site.com/api/account" method="post">

<input type="hidden" name="Transaction" value="withdraw">

<input type="hidden" name="Amount" value="1000000">

<input type="submit" value="Click to collect your prize!">

</form>



Figuur 1: CSRF attack

## ASP.NET Core bescherming tegen CSRF attack requests

In de Razor view staat een form tag helper

Standaard wordt er een anti-forgery key gegenereerd die kan worden gecontroleerd bij de HttpPost action methods in de controller die de request afhandelt

<form asp-action="Process"

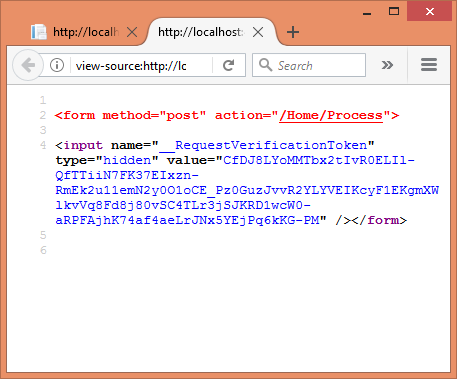
asp-controller="Home"

asp-antiforgery="true"

method="post">

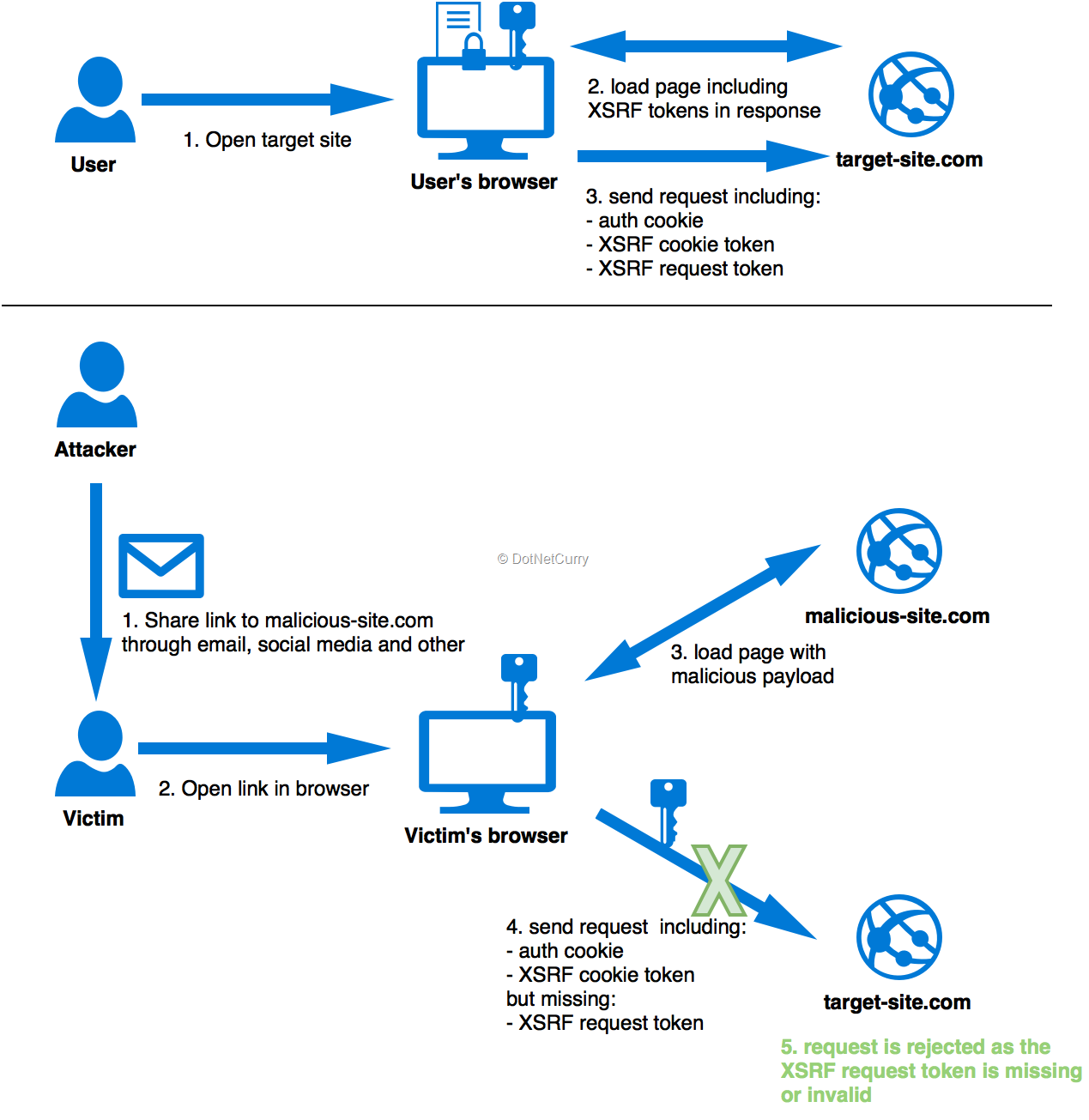
</form>

Deze bovenstaande razor code genereert de volgende HTML



Er wordt bij de form eveneens in een hidden field (onzichtbaar voor de gebruiker) een RequestVerificationToken bijgehouden. Bij de form submit wordt deze key meegestuurd in de Http post request

De Action methods in de Controller class die HttpPost requests afhandelen kunnen deze verificatie key controleren via het attribuut [ValidateAntiForgeryToken] boven de action methods te plaatsen.



Figuur 2: Antiforgery bij CSRF attack requests

<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>