# Creating an Angular app with ASP.NET security.

1. Prerequisites
   1. Visual Studio Professional 2017 (Version 15.3.3 or newer)
   2. Dot net core 2.0 framework, available from <https://www.microsoft.com/net/download/core>
2. File🡪New Project, Select ASP.NET Core Web Application. Name it (I called mine AngularWithSecurity) and put it in the location that you want. Click OK.
3. In the New ASP.NET Core Web Application screen, select .NET Framework from the first dropdown at the top, and select ASP.NET Core 2.0 in the second drop down. (If ASP.NET Core 2.0 is missing, there should be a yellow bar at the top, which you can click on to install it.)
4. Select Angular as the template. It doesn’t provide security out of the box, so just click Ok to continue.
5. It creates the set up and installs npm packages. If npm can’t get through the firewall, go to the path C:\Users\[your username] and add a file called .npmrc that contains:

proxy=http://proxy.nab.com.au:10091

strict-ssl=false

https-proxy=http://proxy.nab.com.au:10091

registry=http://registry.npmjs.org

If npm is not installed on your system, go the web site nodejs.org and install the node 64 bit v6.11.3 LTS version. Npm (node package manager) comes with it.

1. Build the application and make sure it runs. Note that it could take some time in the NAB environment to load all the packages the first time, perhaps even 5 to 10 minutes. It’s almost instantaneous outside of this environment.

It should show a generic Angular app that provides a counter and shows how to fetch data via a web api call, but no database interaction. Now stop the application.

1. Create another application, File🡪New Project and call it DefaultMVCApplication. We will generate an MVC application with security so that we can easily copy the bits we need to the angular application. Click OK to create the application.
2. Select .NET Framework and ASP.NET Core 2.0 in the dropdowns and then choose the Web Application (Model-View-Controller) template. Click the Change Authentication button and choose Individual User Accounts. Make sure “Store user accounts in-app” is selected. Then click OK to close that dialog box. Then click OK to create the MVC application.
3. Build that application to make sure everything is installed properly.
4. In the AngularWithSecurity application, right click on the project and click on Manage Nuget Packages. Add the following packages to the application:
   1. EntityFramework (Latest stable 6.1.3)
   2. Microsoft.AspNetCore.Authentication.Cookies (2.0)
   3. Microsoft.AspNetCore.Identity.EntityFrameworkCore (2.0)
   4. Microsoft.EntityFrameworkCore.SqlServer (2.0)
   5. Microsoft.EntityFrameworkCore.Tools (2.0)
5. Add a new folder at the top level (the same level as ClientApp) called Models. Copy the entire Models folder from the DefaultMVCApplication into the Models folder of the AngularWithSecurity application.
6. Do the same with the Data folder:

Add a new folder at the top level (the same level as ClientApp) called Data. Copy the entire Data folder from the DefaultMVCApplication into the Data folder of the AngularWithSecurity application.

1. Do a global replace to replace the word DefaultMVCApplication with the word AngularWithSecurity.(Edit🡪Find and Replace🡪Replace in all files, Find “DefaultMVCApplication” Replace with “AngularWithSecurity”, Entire Solution, then click Replace All. Click Yes, you are ok with changing it blindly. ) Check one of the files just imported to make sure the change happened.
2. We have a slight change to the LoginViewModel to support usernames instead of email addresses for logging in. Go to the Models/AccountViewModels/LoginViewModel.cs file and add a UserName property. Make sure you add a Required attribute to the UserName and remove the Required attribute from the EmailAddress. Mine looks like this:

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace AngularWithSecurity.Models.AccountViewModels

{

public class LoginViewModel

{

[Required]

public string UserName { get; set; }

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[Display(Name = "Remember me?")]

public bool RememberMe { get; set; }

}

}

1. Do the same with the RegisterViewModel class. Mine looks like this:

public class RegisterViewModel

{

[Required]

public string UserName { get; set; }

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

1. Go to the Controllers folder and add a controller called AccountController. My AccountController contains the following code. You should be able to add just the properties and methods into the main class. This is a stripped down version of the AccountController class found in the DefaultMVCApplication project.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using PoolMaster.Models;

using Microsoft.AspNetCore.Authentication;

using AngularWithSecurity.Models.AccountViewModels;

// For more information on enabling MVC for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace AngularWithSecurity.Controllers

{

[Authorize]

public class AccountController : Controller

{

private readonly UserManager<ApplicationUser> \_userManager;

private readonly SignInManager<ApplicationUser> \_signInManager;

public AccountController(

UserManager<ApplicationUser> userManager,

SignInManager<ApplicationUser> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

//

// GET: /Account/Login

[HttpGet]

[AllowAnonymous]

public async Task<IActionResult> Login(string returnUrl = null)

{

// Clear the existing external cookie to ensure a clean login process

await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

ViewData["ReturnUrl"] = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(model.UserName, model.Password, model.RememberMe, lockoutOnFailure: false);

if (result.Succeeded)

{

return RedirectToLocal(returnUrl);

}

ModelState.AddModelError(string.Empty, "Invalid login attempt.");

return View(model);

}

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

return View(model);

}

//

// GET: /Account/Register

[HttpGet]

[AllowAnonymous]

public IActionResult Register(string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterViewModel model, string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.UserName, Email = model.Email };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

// For more information on how to enable account confirmation and password reset please visit https://go.microsoft.com/fwlink/?LinkID=532713

// Send an email with this link

//var code = await \_userManager.GenerateEmailConfirmationTokenAsync(user);

//var callbackUrl = Url.Action(nameof(ConfirmEmail), "Account", new { userId = user.Id, code = code }, protocol: HttpContext.Request.Scheme);

//await \_emailSender.SendEmailAsync(model.Email, "Confirm your account",

// $"Please confirm your account by clicking this link: <a href='{callbackUrl}'>link</a>");

await \_signInManager.SignInAsync(user, isPersistent: false);

return RedirectToLocal(returnUrl);

}

AddErrors(result);

}

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

return View(model);

}

//

// POST: /Account/Logout

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction(nameof(HomeController.Index), "Home");

}

#region Helpers

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

private IActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

else

{

return RedirectToAction(nameof(HomeController.Index), "Home");

}

}

#endregion

}

}

1. If any of the fields in the AccountController class have red squiggly lines under them, it means that they need to have references added. Go over each one, click on Ctrl-. (Ctrl dot) and then select the first item on the tooltip helper for each one (I usually just click Enter when the tooltip appears).
2. At this stage, everything is compiling on my machine. Build the application to make sure everything compiles so far for you. If not, you need to go back and work through the steps to see what you’ve missed.
3. Go to the HomeController class in Controllers folder and add an Authorize attribute to the controller. This will make it require authorisation when attempting to access the main page. Mine looks like this:

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

namespace AngularWithSecurity.Controllers

{

public class HomeController : Controller

{

[Authorize]

public IActionResult Index()

{

return View();

}

public IActionResult Error()

{

ViewData["RequestId"] = Activity.Current?.Id ?? HttpContext.TraceIdentifier;

return View();

}

}

}

1. Add a default database connection string into the appsettings.json file found in the root folder. Make sure you update the user id and password with valid values for you. This will give you a different sql profiler id which will help with debugging. The ASP.NET security database is found in the pool\_master database. Mine looks like this:

{

"ConnectionStrings": {

"DefaultConnection": "Server=10.58.193.121\\SQL1,51565;Database=pool\_master;User Id=twright;Password=xxxxx;MultipleActiveResultSets=true"

},

"Logging": {

"IncludeScopes": false,

"Debug": {

"LogLevel": {

"Default": "Warning"

}

},

"Console": {

"LogLevel": {

"Default": "Warning"

}

}

}

}

1. Create a new folder at the root level (same level as ClientApp) and call it Filters. Add a classe called AngularAntiForgeryCookieResultFilter. Mine looks like this:

using Microsoft.AspNetCore.Antiforgery;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace AngularWithSecurity.Filters

{

public class AngularAntiforgeryCookieResultFilter : ResultFilterAttribute

{

private IAntiforgery antiforgery;

public AngularAntiforgeryCookieResultFilter(IAntiforgery antiforgery)

{

this.antiforgery = antiforgery;

}

public override void OnResultExecuting(ResultExecutingContext context)

{

if (context.Result is ViewResult)

{

var tokens = antiforgery.GetAndStoreTokens(context.HttpContext);

context.HttpContext.Response.Cookies.Append("XSRF-TOKEN", tokens.RequestToken, new CookieOptions() { HttpOnly = false });

}

}

}

}

1. Go to the Startup.cs file and make modifications to the ConfigureServices method, as follows. Make sure you get rid of any red squiggly lines by using the Ctrl-. trick.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

//services.AddOptions();

// services.Configure<ValidateOriginOptions>(Configuration);

// Add framework services.

services.AddAntiforgery(opts => opts.HeaderName = "X-XSRF-Token");

services.AddMvc(opts =>

{

opts.Filters.AddService(typeof(AngularAntiforgeryCookieResultFilter));

})

.AddJsonOptions(options=> {

options.SerializerSettings.DateTimeZoneHandling = DateTimeZoneHandling.Local;

});

services.AddTransient<AngularAntiforgeryCookieResultFilter>();

}

1. Still an Startup.cs, in another method, this tile called Configure, add a line to tell ASP.NET Core to use the Authentication framework.
2. Find the Views folder at the root level and open the \_ViewImports.cshtml file. Make sure it has code like this, replacing the Application name with your own:

@using AngularWithSecurity

@using AngularWithSecurity.Models

@using AngularWithSecurity.Models.AccountViewModels

@using Microsoft.AspNetCore.Identity

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@addTagHelper \*, Microsoft.AspNetCore.SpaServices

1. Still in the Views folder at the root level, add a new folder called Account. Inside this folder, add a new item, select the “MVC View Page” template, and call it Login.cshtml, then click Add. Then cut and paste the following code (remembering to replace the application name.)

@using System.Collections.Generic

@using System.Linq

@using Microsoft.AspNetCore.Http

@using Microsoft.AspNetCore.Http.Authentication

@model AngularWithSecurity.Models.AccountViewModels.LoginViewModel

<style>

.banner {

background: #32408f;

height: 64px;

width: 100%;

padding-left: 15px;

padding-top: 20px;

}

.banner h1 {

height: 30px;

display: table-cell;

vertical-align: middle;

color: #eee;

margin: 0;

float: left;

font-size: 24px;

}

.banner img {

margin-top: -6px;

float: left;

height: 36px;

}

.search-overlay {

background: rgba(68,138,255, 0.50);

border: 1px solid rgba(63,81,181, 0.9);

margin-top: 60px;

/\*height: 233px;\*/

height: 240px;

font-weight: bold;

width: 400px;

padding: 40px;

font-family: "Roboto";

margin-left: auto;

margin-right: auto;

display: block;

}

.search-overlay .row {

margin-top: 3px;

}

.search-overlay label {

margin-top: 2px !important;

font-size: 18px;

font-weight: normal;

color: #fff;

}

.search-overlay input {

color: #fff;

font-size: 18px;

font-weight: normal;

}

.search-overlay button {

color: #fff !important;

background-color: #448aff;

border: 1px solid #3077eb;

}

.main-login-div {

background-image: url(/images/main-banner.jpg);

background-color: #d9dfe3;

background-repeat: no-repeat;

background-attachment: fixed;

background-position: top left;

height: 100%;

min-height: 1000px;

min-width: 600px;

padding: 0;

margin: 0;

}

.nopadding {

padding: 0;

}

.validation-summary-errors {

background: #fff;

width: 318px;

color: red;

margin-top: 40px;

text-align: center;

}

.validation-summary-errors ul {

list-style: none;

padding: 10px;

}

</style>

<div class="container-fluid nopadding" style="overflow-x:hidden">

<div class="row">

<div class="col-md-12">

<div class="row">

<div class="col-xs-12">

<div class="banner">

<div class="pull-left">

<img src="data:image/png;base64,"><h1>PoolMaster</h1>

</div>

</div>

</div>

</div>

</div>

<div class="row">

<div class="col-xs-12" style="height:44px;background-color:#448aff">

</div>

</div>

</div>

<div class="row main-login-div">

<div class="col-xs-12">

<form asp-controller="Account" asp-action="Login" asp-route-returnurl="@ViewData["ReturnUrl"]" method="post" class="form-horizontal">

<div class="search-overlay">

<div>

<div class="row">

<div class="col-xs-12">

<label class="control-label pull-left" style="width:30%;text-align:left;">Login:</label>

<input asp-for="UserName" class="form-control pull-left" style="width:65%" />

<span asp-validation-for="UserName" class="text-danger"></span>

</div>

</div>

<div class="row">

<div class="col-xs-12">

<label class="control-label pull-left" style="width:30%;text-align:left;">Password:</label>

<input asp-for="Password" class="form-control pull-left" style="width:65%" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

</div>

<div class="row">

<div class="col-xs-offset-4 col-xs-8" style="margin-top:8px;">

<input type="checkbox" id="RememberMe" name="RememberMe" class="form-control pull-left" style="width:30%;text-align:left;height:16px;width:16px;" />

<label asp-for="RememberMe" style="width:65%;margin-top:0 !important"></label>

</div>

</div>

<div class="row">

<div class="col-xs-4"><a href="/Account/Register" style="color:#fff;text-decoration:underline;font-weight:normal">Register</a></div>

<div class="col-xs-8"><button class="btn pull-right" type="submit">Login <i class="fa fa-key"></i></button></div>

</div>

<div class="row">

<div class="col-xs-12">

<div asp-validation-summary="All"></div>

</div>

</div>

</div>

</div>

</form>

</div>

</div>

</div>

@section Scripts {

}

1. Add another .cshtml file, this time called Register.cshtml. Mine looks like this:

@model RegisterViewModel

@{

ViewData["Title"] = "Register";

}

<h2>@ViewData["Title"].</h2>

<**form** **asp-controller**="Account" **asp-action**="Register" **asp-route-returnurl**="@ViewData["ReturnUrl"]" method="post" class="form-horizontal">

<h4>Create a new account.</h4>

<hr />

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

<div class="form-group">

<**label** **asp-for**="Email" class="col-md-2 control-label"></**label**>

<div class="col-md-10">

<**input** **asp-for**="Email" class="form-control" />

<**span** **asp-validation-for**="Email" class="text-danger"></**span**>

</div>

</div>

<div class="form-group">

<**label** **asp-for**="Password" class="col-md-2 control-label"></**label**>

<div class="col-md-10">

<**input** **asp-for**="Password" class="form-control" />

<**span** **asp-validation-for**="Password" class="text-danger"></**span**>

</div>

</div>

<div class="form-group">

<**label** **asp-for**="ConfirmPassword" class="col-md-2 control-label"></**label**>

<div class="col-md-10">

<**input** **asp-for**="ConfirmPassword" class="form-control" />

<**span** **asp-validation-for**="ConfirmPassword" class="text-danger"></**span**>

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<button type="submit" class="btn btn-default">Register</button>

</div>

</div>

</**form**>

@section Scripts {

}

1. Next, I will add a bit of angular code so that we can see who the logged in user is. Go into the SampleDataController class, found in the Controllers folder. Add the following method, which returns the currently logged in user name:

[HttpGet("[action]")]

public string WhoAmI()

{

return this.User.Identity.Name;

}

1. Go to the fetchdata.component.ts file, found in the ClientApp/app/components/fetchdata folder. Add the following code, which I have highlighted, to the constructor method of the FetchDataComponent:

import { Component, Inject } from '@angular/core';

import { Http } from '@angular/http';

@Component({

selector: 'fetchdata',

templateUrl: './fetchdata.component.html'

})

export class FetchDataComponent {

public forecasts: WeatherForecast[];

public whoami: string = 'no-one';

constructor(http: Http, @Inject('BASE\_URL') baseUrl: string) {

http.get(baseUrl + 'api/SampleData/WeatherForecasts').subscribe(result => {

this.forecasts = result.json() as WeatherForecast[];

}, error => console.error(error));

http.get(baseUrl + 'api/SampleData/WhoAmI').subscribe(result => {

this.whoami = result.text();

}, error => console.error(error));

}

}

interface WeatherForecast {

dateFormatted: string;

temperatureC: number;

temperatureF: number;

summary: string;

}

1. Go to the fetchdata.component.html file. Add the following highlighted code to the fetchdata.component.html page:

<h1>Weather forecast</h1>

<p>This component demonstrates fetching data from the server.</p>

<p \*ngIf="!forecasts"><em>Loading...</em></p>

<p>Congratulations, you are: **{{**whoami**}}**</p>

<table class='table' \*ngIf="forecasts">

<thead>

<tr>

<th>Date</th>

<th>Temp. (C)</th>

<th>Temp. (F)</th>

<th>Summary</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let forecast of forecasts">

<td>**{{** forecast.dateFormatted **}}**</td>

<td>**{{** forecast.temperatureC **}}**</td>

<td>**{{** forecast.temperatureF **}}**</td>

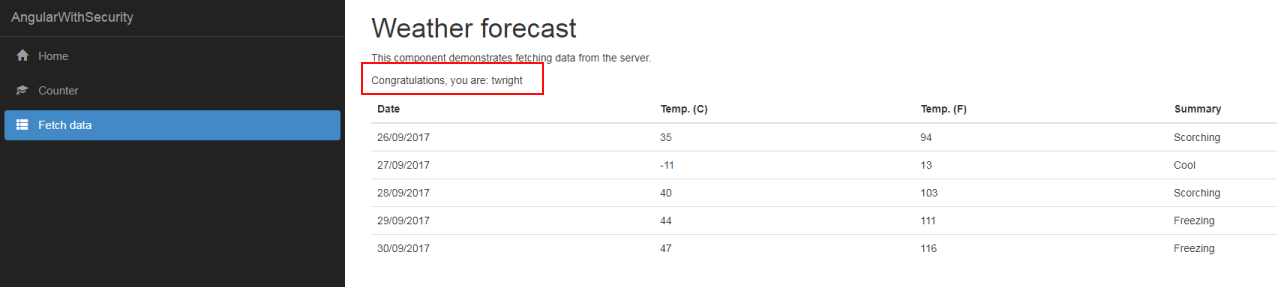
<td>**{{** forecast.summary **}}**</td>

</tr>

</tbody>

</table>

1. Right click on the project file and click on Properties. Go to the Debug tab and scroll down to Web Server Settings. Tick the Enable SSL box, then click on Copy to copy the SSL url to the clipboard. Select the App URL and right-click, paste, to paste the https url into the debug box. The application should now compile.
2. Run the application. It should automatically redirect you to the login page. Log in using twright, password abc123 (or another account you know of that works in poolmaster). The home page should appear. Click on Fetch data. It should fetch data from the Web Api, and also return who the logged in user is. My page looks like this:



And that’s it, you’ve added ASP.NET security to the default Angular application in Visual Studio.