Office 365 Authentication and Authorization with ASP.NET Core Web API and separate frontend

# Intro

In this article you will learn how to sign into an office 365 account on your frontend using an ASP.NET Core Web API. Should you wish to view my GitHub to see the full code, follow this link:

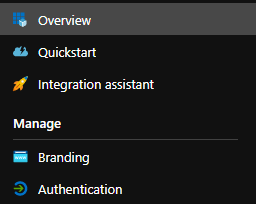
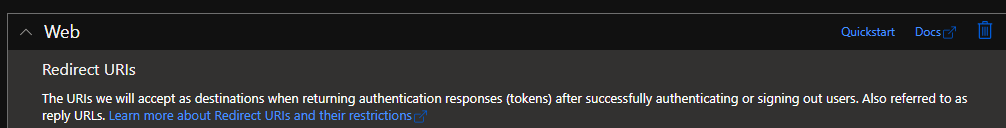
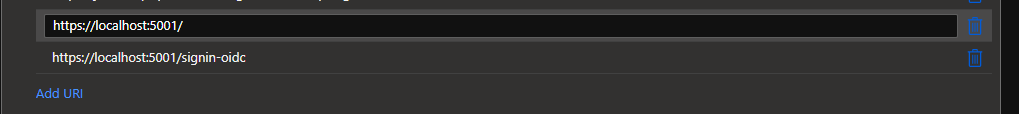
<https://github.com/johnmason27/basic-auth>

# Prerequisites

* Microsoft Azure Account <https://portal.azure.com/>
* Visual Studio and .NET Framework installed
* Basic Understanding of C#, JavaScript, ASP.NET Core Web API
* IIS (Internet Information Services) installed

# Contents

## Setting up Azure

* Log into your Azure portal <https://portal.azure.com/>
* Go to App registrations
* Create a new App Registration and fill in the details required
  + Enter application name
  + Select supported account types
  + **Click** register
* On the overview page you will see your ClientId and TenantId. You will need these later. When entering details into appsettings.json.
* Go to the Authentication tab on the left-hand side. You will need to add a new platform and add a web application.
* When you click on the web application you will need to give a redirect URL to your API.
  + If your API is running on <https://localhost:5501> your URL you add should be:
    - https://localhost:5501/signin-oidc
    - Or the {{ API URL }} + signin-oidc
  + Make sure to add 2 URLs into the new platform you created, one with the /signin—oidc and another without it just the localhost URL. Like in the picture.

## Setting up the API

### Intro

Create a new ASP.NET Core Web project called “SampleAuth” and when you get to choose what type of API you want select API.

### Nuget Packages

Once you have created the project you need to install the necessary Nuget packages. For this Authentication we only need the one package. So install “Microsoft.Identity.Web” before moving to the next step.

### Appsettings.json

To talk to the correct app registration on Azure we need to configure what the app will connect too. We do this using the “appsettings.json” file. Add the new object to the top of your file.

“AzureAd”: {

“Instance”: “<https://login.microsoftonline.com/>”,

“Domain”: “DOMAIN OF YOUR OFFICE ACCOUNT E.g. text after @ in your email outlook.com”,

“ClientId”: “Found on the Azure portal overview page for the App Registration”,

“TenantId”: “Found on the Azure portal overview page for the App Registration”,

“CallbackPath”: “/signin-oidc”

}

### Models

Create a new folder to put data models in called Models.

#### User.cs

In the “Models” folder create a new class called “User” and add 1 property inside.

public string Name { get; set; }

### Startup.cs

We will now start to configure the Startup.cs file. Firstly, go to the top of the file and enter the following using statements.

using Microsoft.Identity.Web;

using Microsoft.AspNetCore.Authentication.OpenIdConnect;

using Microsoft.AspNetCore.Authorization;

using System.Threading.Tasks;

Next go to the “ConfigureServices” method and remove all lines of code with the following.

services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)

.AddMicrosoftIdentityWebApp(options => {

this.Configuration.GetSection(“AzureAd”).Bind(options);

options.Events.OnRedirectToIdentityProvider = context => {

if (context.HttpContext.Items.ContainsKey(“allowRedirect”)) {

return Task.CompletedTask;

}

};

context.HandleResponse();

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

return Task.CompletedTask;

});

services.AddAuthorization(options => {

options.DefaultPolicy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();

});

services.AddControllers();

This will add authentication, authorization, and controllers to our API. The Authentication gets the configuration from the app settings as well as setting up the redirect configuration. If there is a key on the response allowing a redirect, it will allow the redirect to happen if there isn’t the redirect will replace by returning a status code 401. You can use this on the frontend side to check whether or not there is a 401 returned and if there is you can redirect the user to get logged in again. The authorize section allows us to add the tags above our controller methods in the next section. But will check to see whether we are authenticated before allowing us to use a certain controller.

Finally, go to the “Configure” method and remove all the lines of code and replace them with the following.

if (env.IsDevelopment()) {

     app.UseDeveloperExceptionPage();

}

app.UseCors(policyBuilder => policyBuilder.AllowCredentials().SetIsOriginAllowed(origin => true).AllowAnyHeader())

.UseHttpsRedirection()

.UseRouting()

.UseAuthentication()

.UseAuthorization()

.UseEndpoints(endpoints => endpoints.MapControllers());

### LoginController.cs

Create a new empty controller called “LoginController” and start by adding the following attributes above the class.

[Route(“api/[controller]”)]

[ApiController]

This will make sure the controller is recognised. Next add the following using statements to the top of the file.

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authorization;

using System.Threading.Tasks;

Now we need to write the method to log a user in and check whether they are logged in or not. We will start by checking whether or not a user is logged in. Add the following method to the class.

[AllowAnonymous]

[HttpGet]

public ActionResult IsUserLoggedIn() {

if (!this.HttpContext.User.Identity.IsAuthenticated) {

return this.Unauthorized();

}

return this.Accepted();

}

You will now be able to call that method and check whether the user is logged in or not. Next we will write the method to login a user in. Add the following method to the class.

[AllowAnonymous]

[HttpGet(“Authenticate”)]

public async Task Login() {

if (!this.HttpContext.User.Identity.IsAuthenticated) {

this.HttpContext.Items.Add(“allowRedirect”, true);

await this.HttpContext.ChallengeAsync();

return;

}

this.HttpContext.Response.Redirect(“URL to your webpage from IIS”);

}

What we do here in this method is use the IsAuthenticated() method to check whether or not a user is authenticated and if they’re not we redirect them to the login page before redirecting them back to the frontend webpage.

### UserController.cs

Create a new empty controller called “UserController” and start by adding the following attributes above the class.

[Route(“api/[controller]”)]

[ApiController]

This will make sure the controller is recognised. Next add the using statements to the top of the file.

using Microsoft.AspNetCore.Authorization;

Next add a new method to the file inside the class.

[Authorize]

[HttpGet]

public ActionResult<User> GetUser() {

var user = new User();

user.Name = this.HttpContext.User.Identity.Name.ToString();

return this.Ok(user);

}

You will now be able to get the current user who is logged in when calling this method. However if nobody is logged in it will return a status code 401 for unauthorized. Telling you they are not logged in.

### Outro

You have now written all of the code needed for the API so next we will write the basic frontend up!

## Setting up the Frontend

### Intro

The frontend of this is going to be super simple so if you are new to JavaScript it should be easy to follow along.

### Getting Started

Firstly create a new folder called “sample-auth” and open it in a text editor of your choice.

Next create a file inside the folder called “index.html” and add the following lines of code.

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Basic Auth</title>

<script type="module" src="./Main.js"></script>

</head>

<body>

<h1>Welcome to my basic authentication sample</h1>

<p id="welcome-message">Hello Guest</p>

</body>

</html>

### JavaScript

Now that we have the HTML setup it’s time to setup the JavaScript side of things.

Create a new file called “Main.js” and enter the following code inside.

function getUser() {

fetch("https://localhost:5001/api/User", {

method: "GET",

headers: {

"Accept": "application/json",

"Content-Type": "application/json"

},

credentials: "include"

}).then(response => {

return response.json();

})

.then(response => {

document.querySelector("#welcome-message").innerHTML = "Hello, " + response.name;

}).catch((err) => {

document.body.innerHTML = err;

document.body.style.color = "red";

});

}

window.addEventListener("load", () => {

fetch("https://localhost:5001/api/Login", {

method: "GET",

headers: {

"Accept": "application/json",

"Content-Type": "application/json"

},

credentials: "include"

}).then(response => {

if (response.status !== 202) {

window.location = "https://localhost:5001/api/Login/authenticate";

} else {

getUser();

}

}).catch((err) => {

document.body.innerHTML = err;

document.body.style.color = "red";

});

});

What we do in the JavaScript is add a onload event to the window that will check whether the user is logged in or not. If they’re not logged in we set the window location to <https://localhost:5001/api/Login/authenticate> to send a get request to the API which will then redirect us to the office 365 login page before redirecting us back to the website. We have to send the get request this way to avoid CORS policy issue when trying to redirect with fetch requests. When you reload onto the page it won’t redirect you back to office 365 this time because you will be logged in, so we call the method getUser() and get the details of the logged in user and print them to the webpage.

### Outro

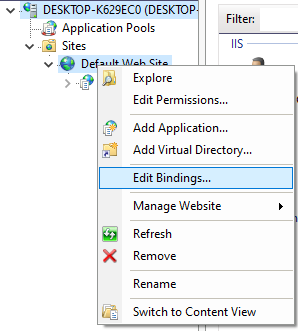
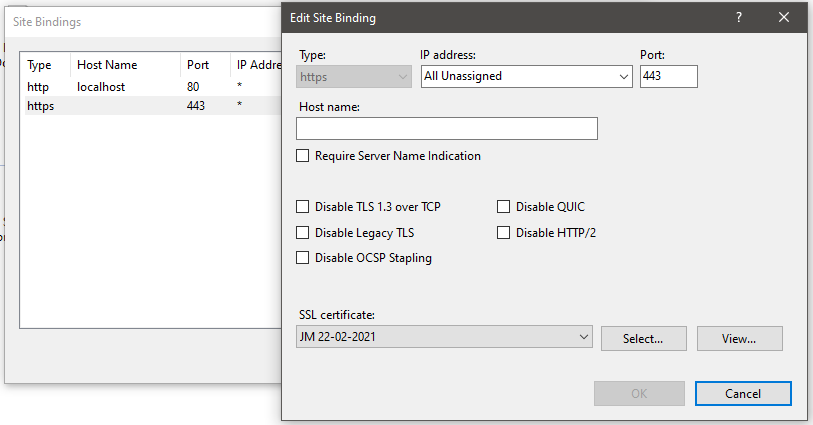
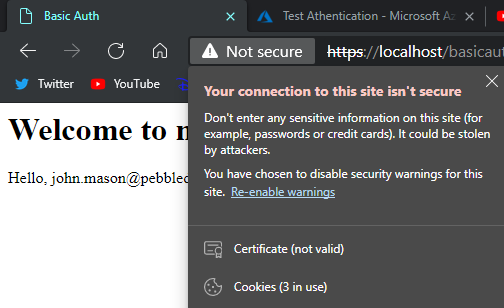
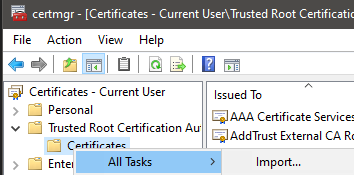
That is it now for the frontend. So it’s time to link all this up together to talk to each other!

# IIS

### Intro

We now need to setup our website in IIS so we can have a static URL to redirect too.

### Adding our site

* Open IIS
* Go into Sites > Right-click on “Default Web Sites” and then click on add an application.
* Call the alias, “basicauth”
* Add the path to your application. I recommend adding your files at the top level of your hard drive to get around any permission issues. (e.g. C:\\)
* Click ok
* Before this next step we need to create an SSL certificate.
  + To make a self-signed certificate click on the computer at the top of the menu on the left inside IIS.
  + Then click on SSL certificates.
  + Create self-signed certificate.
  + Enter a good name such as John’s Certificate 22-02-2021 and click ok.
* Now right click on “Default Web sites” and then click on edit bindings.
* Adding a new binding
  + Click add
  + Change the type to HTTPS
  + Then change the SSL certificate to the one you just made and click ok.
* If you navigate now to your site it will still be saying it’s not secure.
* Since we made a self-signed signature we need to trust it ourselves.
* To do this click on the not secure part of the URL and then on the certificate. Then go to details, copy to file, next, select Cryptographic Message Syntax Standard and check Include all certificates in the certification path if possible, save to your computer and press finish.
* Using a terminal type certmgr.msc and press enter.
* Click on Trusted Root Certification Authorities, right click on certificates, all tasks and then import, import your saved certificate, press next, next and finish.
* Your site is now fully setup. Navigate to <https://localhost/basicauth> to get to your site.

### Updating our API redirect

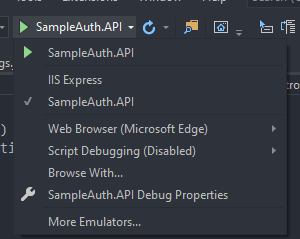
Go back to the API and into the LoginController and replace the URL part of the Redirect with the URL of your site from IIS.

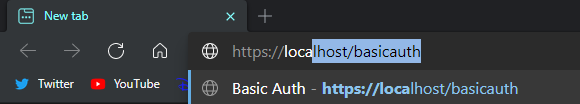
this.HttpContext.Response.Redirect("URL to your webpage from IIS");

E.g. this.HttpContext.Response.Redirect("https://localhost/basicauth");

Your site should now be fully configured and ready to test which is in the next section.

# Testing

If you turn on your API making sure to sure it on <https://localhost:5501/> to do this make sure before pressing start to select the SampleAuth.API option.

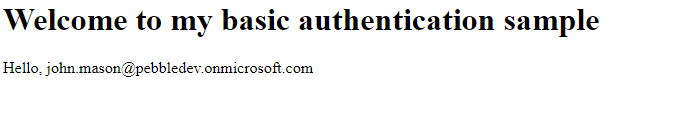
Next open the frontend in a web browser of your choice, mine will be Edge Chromium and navigate to your IIS site. Mine being <https://localhost/basicauth>!

You should see what I see or similar.

When the page has loaded the fetch requests should’ve taken place and you should be redirected to the Office sign in page.

Pick your account, and login.

You should now be redirected back to your website and see the message Hello, followed by your office accounts email.



# Outro

You should now be fully setup with a fully working site and API. Hope you found this helpful, checkout the full code on my GitHub using this link:

<https://github.com/johnmason27/basic-auth>

Link to my LinkedIn:

<https://www.linkedin.com/in/john--mason/>