**Setup an Aurelia App with TypeScript and ASP.NET Core**

[matte](http://edgamat.com/author/matte/" \o "Posts by matte) February 4, 2017 [0](http://edgamat.com/2017/02/setup-aurelia/" \l "respond)

I have been experimenting with Aurelia as a means of building web applications. During that time I have created a number of projects using ASP.NET Core to host my application.

The following are the steps I use to create the ‘base’ project structure. My preferred development environment is:

* Windows 10
* Visual Studio 2015
* Aurelia CLI
* TypeScript
* SASS

Ensure the Aurelia CLI is installed on your PC:

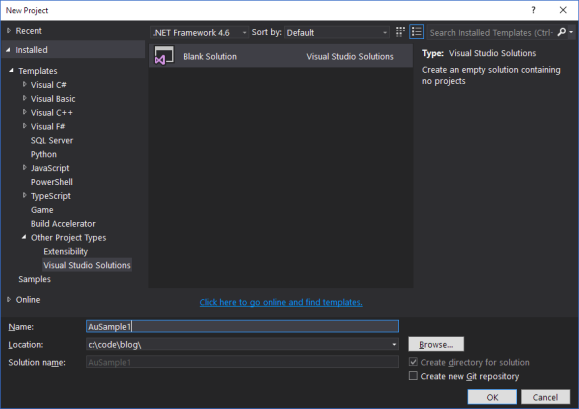
[http://aurelia.io/hub.html#/doc/article/aurelia/framework/latest/the-aurelia-cli/1](http://aurelia.io/hub.html" \l "/doc/article/aurelia/framework/latest/the-aurelia-cli/1" \t "_blank)

**Create an Empty ASP.NET Core Web Application**

Using Visual Studio 2015, create an Empty ASP.NET Core Web Application. I do this in two steps. First, I create a blank solution. I do this for historical reasons. This is how I learned to correctly lay out the folder structures using Visual Studio 2005 and Visual SourceSafe… memories…  However, it is still practical as it allows me to more easily add the projects under a root folder.

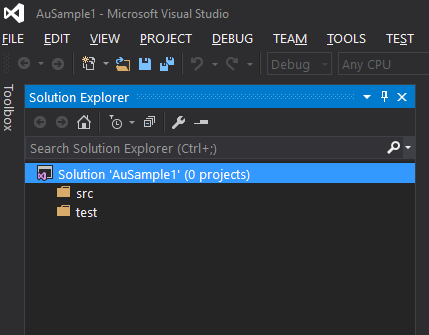
In Visual Studio, select File… New Project…

You should find under Templates, “Other Project Types”, and within there, “Visual Studio Solutions”. Select “Blank Solution”, and then give it a name and folder location.

[](http://edgamat.com/wp-content/uploads/2017/02/new-au-01-sln.png)

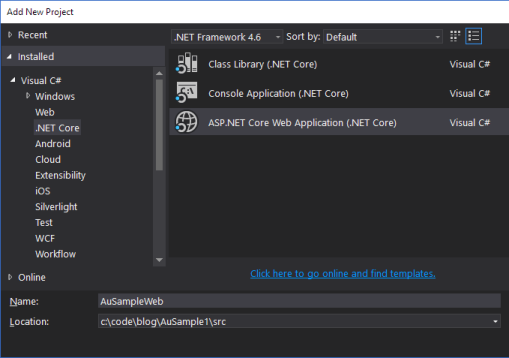
In Visual Studio, the newly created solution should be visible in the Solution Explorer. I prefer to include two sub-folders under the solution when developing ASP.NET Core applications. One to contain the source code (src) and another to contain the xUnit test projects (test). This approach matches the ‘default’ approach that Visual Studio uses if you allow it to. It makes it easier to manage the projects in my experience.

To manually create these sub-folders, right-mouse click on the solution name in the Solution Explorer and select “Add… New Solution Folder”. Enter the name of src. Repeat this for a second folder called test. You should see this when you are done:

[](http://edgamat.com/wp-content/uploads/2017/02/new-au-02-sln-folders.png)

Now, we can add the web project. Right-mouse click on the ‘src’ folder name and select “Add… New Project…”. You should see a menu of choices. Under “Installed”, expand the “Visual C#” group, and select the “.NET Core” sub-group. Within here, should should find “ASP.NET Core Web Application (.NET Core)”. This is the one we want. Once you select it, give it a name and a folder location. The folder name should be the same as you used when creating the solution, with the ‘src’ sub folder added to the end:

|  |  |
| --- | --- |
| 1  2 | Solution Location: C:\code\blog\AuSample1  Project Location C:\code\blog\AuSample1\src |

[](http://edgamat.com/wp-content/uploads/2017/02/new-au-03-web-app.png)

Next, the wizard will ask you to select a project type. Select “Empty Web”, since we are relying on Aurelia for all the fun bits, we don’t require anything else \*at this time\*. Obviously, we will add more later as requirements dictate, but for the initial setup, this is all we require.

Click ‘OK’ and let the project get created.

**Tweak the Web Project for TypeScript.**

As the Aurelia online documentation points out, we don’t want Visual Studio worrying about compiling our TypeScript. So to disable this, we need to tweak the project file. In the Solution Explorer, right-mouse click on the Web Project and select “Unload Project” (it is near the bottom of the list of options). Now right-mouse click again on the project (which should appear as ‘Unavailable’) and select “Edit {ProjectName}.xproj. Visual Studio now opens the project file in a text editor so you can manually update it.

Add the following inside the first <PropertyGroup>:

<TypeScriptCompileBlocked>true</TypeScriptCompileBlocked>

Save the changes and close the file. Now in the Solution Explorer, right-mouse click on the project again, and this time select “Reload Project”.

**Add a new Aurelia project.**

Now we switch over to Aurelia on the command line. Open a command prompt, or PowerShell, or your shell of choice, and navigate to the web project root folder. Now enter the following command:

au new --here

This will start the wizard to create the new Aurelia project. It will step you through a series of choices in order to make the

The first choice is the platform. Select 2, the ASP.NET Core platform. Next, the type of transpiler. Select 2, TypeScript.

I prefer SASS, so select 3, SASS when choosing the CSS processor.

The next choice is for unit testing. You can skip this if you wish, but I always add it since I end up needing it more often than not.

New let the wizard create the project AND install the project dependencies. That last one takes a while. No, seriously, it does take like 5 minutes. Go for a walk, stretch, coffee, whatever… you’ll be waiting.

**Tweak the Aurelia Configuration for TypeScript**

Now that the Aurelia project template is created, we can return to Visual Studio from the command line. In the Solution Explorer you should see a bunch of new folders and files that the wizard created.

The CLI creates a aurelia\_project folder to contain the infrastructure of the project. It contains tasks for building, running and testing the code. I have found that it doesn’t quite get the TypeScript configuration correct. Open the aurelia.json file. In here you will need to make two minor updates. What I found was that the transpiled TypeScript code would be deployed into a ‘Scripts’ folder in the root of the project. However, since I am using the ASP.NET Core platform, the folder needs to exist in the ‘WWWROOT’ sub-folder.

Under the platform node, find the ‘index’ property:

change “index”: “index.html” to “index”: “wwwroot/index.html”

Under the ‘build’ node, make similar updates:

Change:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | "build": {    "targets": [      {        "id": "aspnetcore",        "displayName": "ASP.NET Core",        "output": "scripts",        "index": "index.html",        "baseUrl": "scripts"      }    ], |

to:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | "build": {    "targets": [      {        "id": "aspnetcore",        "displayName": "ASP.NET Core",        "output": "wwwroot/scripts",        "index": "wwwroot/index.html",        "baseUrl": "scripts"      }    ], |

**Add DOT NET Watcher**

ASP.NET Core includes a file watcher that provides a much nicer development environment. When running, it will detect changes on the ASP.NET core side of things automatically and then recompile the code for you. To install this tool, simply open the project.json file and add the following under the ‘tools’ node:

|  |  |
| --- | --- |
| 1 | "Microsoft.DotNet.Watcher.Tools": "1.0.0-preview2-final" |

Right-mouse click on the project in the Solution Explorer and select “Restore Packages”.

NOTE: If you run into issues, check out this great post from Scott Hanselman on how to fix things:

<http://www.hanselman.com/blog/TheMysteryOfDotnetWatchAndMicrosoftNETCoreAppVersion110preview100110000WasNotFound.aspx>

**Add Static Files**

The empty ASP.NET Core template is truly empty. It doesn’t have the ability to serve files. So we have to add that manually.

In the project.json file, add the following under “dependencies”:

|  |  |
| --- | --- |
| 1 | "Microsoft.AspNetCore.StaticFiles": "1.0.0-rc2-final" |

Now go to the Startup.cs and replace:

|  |  |
| --- | --- |
| 1  2  3  4 | app.Run(async (context) =>  {      await context.Response.WriteAsync("Hello World!");  }); |

with:

|  |  |
| --- | --- |
| 1  2 | app.UseDefaultFiles();  app.UseStaticFiles(); |

**Add Watch Command File**

In order to have the ASP.NET Core host running AND have the Aurelia CLI running at the same time, you need to open two command windows. It’s a pain, so I use a command file to do it for me. In the Web Project, add a new text file called Watch.cmd and copy the following into it:

|  |  |
| --- | --- |
| 1  2  3  4  5 | set ROOT\_DIR=%~dp0    start cmd.exe /K "cd %ROOT\_DIR% && dotnet watch run"    start cmd.exe /K "cd %ROOT\_DIR% && au run --watch" |

When you run this from the command line, it will spawn two new command windows that automatically start the two services.

Go ahead and give it a try.

If you browse to http://localhost:5000/ you should see the Hello World from the Aurelia default landing page.

**Done!**

I have written these down for my own purposes, so I don’t forget all these step. But I hope you find them useful as well.