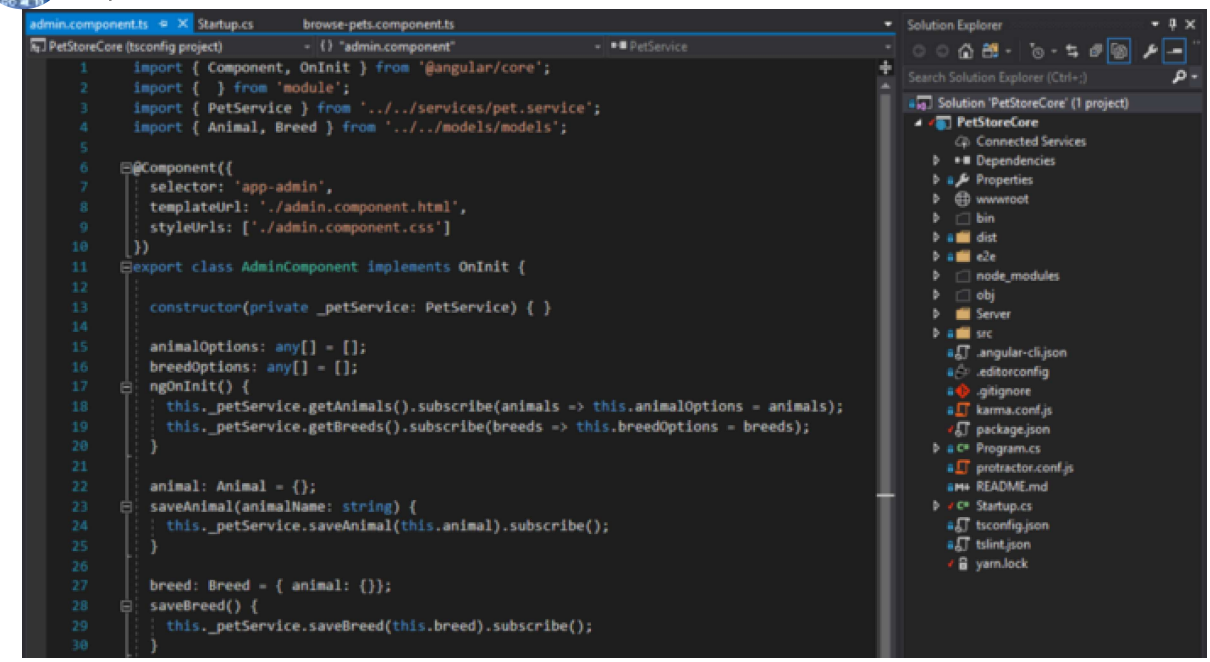
How to build an Angular Application with ASP.NET Core in Visual Studio 2017, visualized

Levi Fuller Follow
Apr 5, 2017 · 7 min read



With an arguably gratuitous number of visuals and code, we'll learn how to build a ready-to-deploy web application built on ASP.NET Core and Angular using the Angular-CLI.
SuperCoolApp on Github

Prerequisites

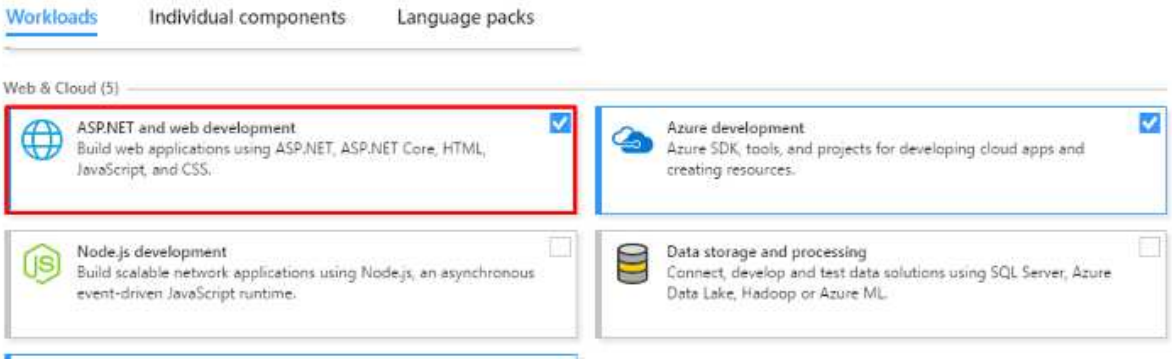
- **Node.js Installed** NOTE: May require a restart
- **.NET Core Installed**

NOTE: If you have a Mac, this excellent article should get you past the Environment Setup section by creating an app using the yeoman generator.

Environment Setup

1. Download and install Visual Studio

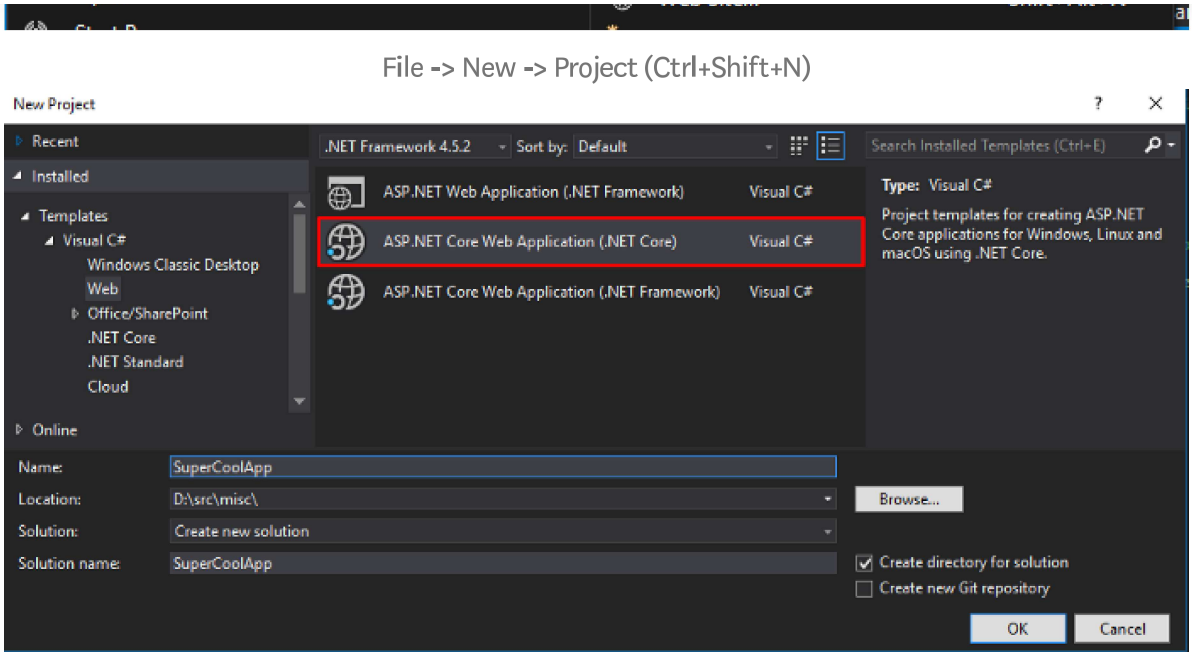
Download Visual Studio 2017 Community — a free, open source IDE — then install it



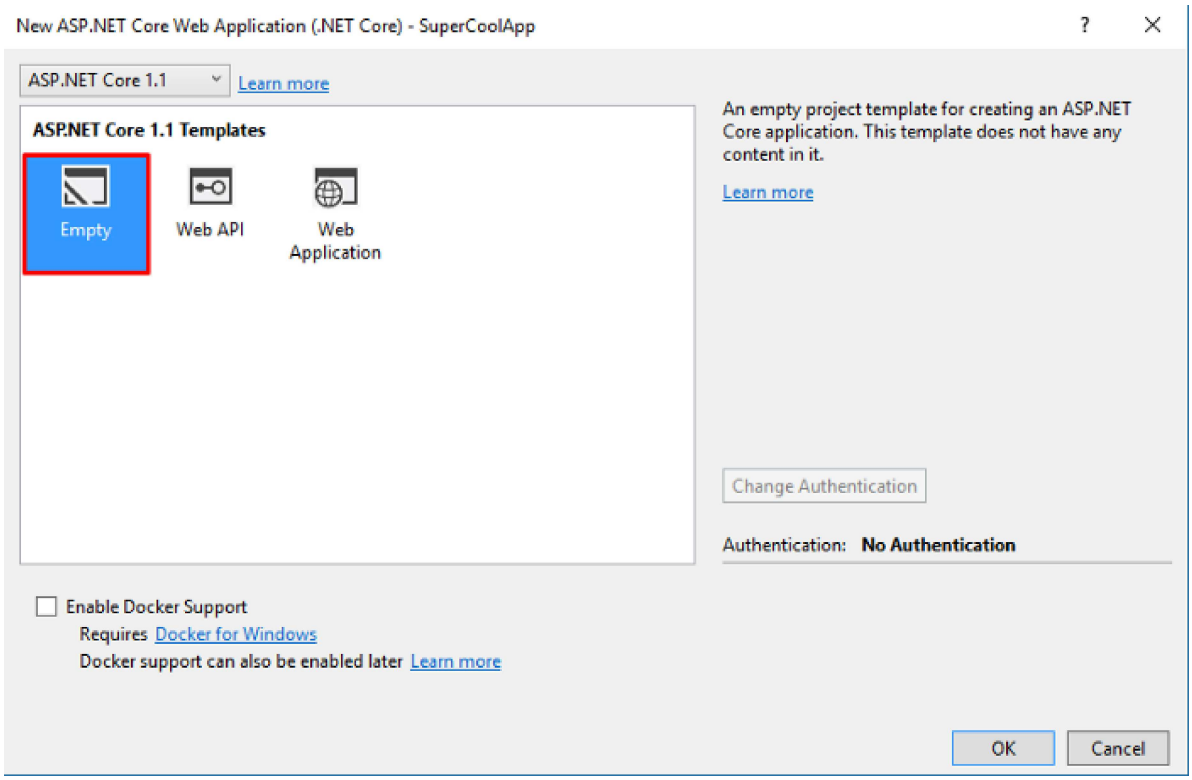
Make sure to install the ASP.NET and web development workload

2. Create a new project

Open Visual Studio 2017 and let the fun begin!



Create an ASP.NET Core Web application



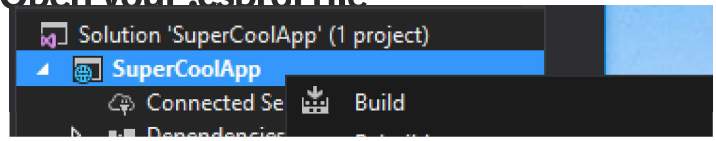
Use the empty template

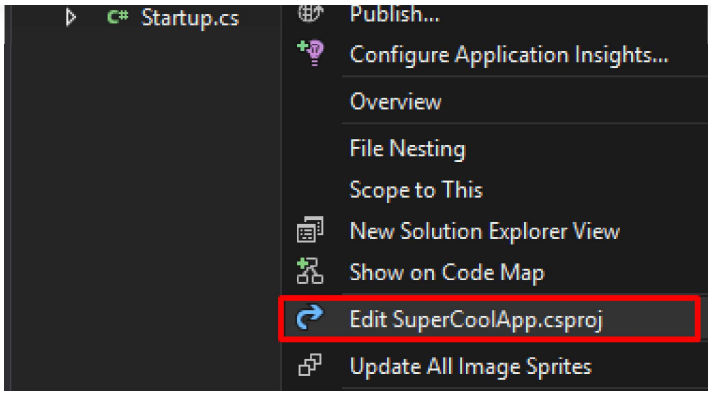
• • •

Configure ASP.NET Core

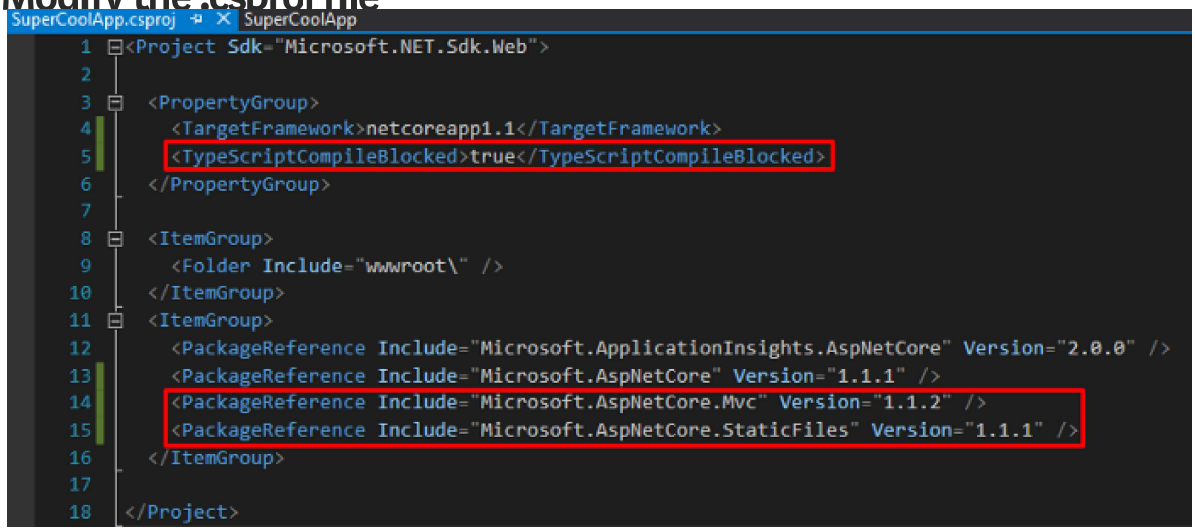
Next, we'll install the dependencies, ensure we don't get TypeScript compile errors, and configure our server.

1. Open your .csproj file





2. Modify the .csproj file



Modified .csproj file

Add the following packages:

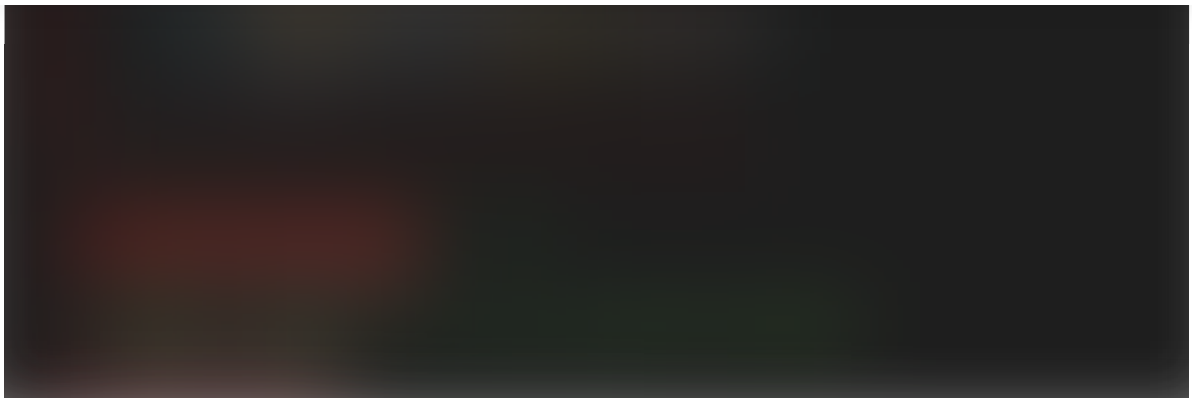
```
<PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2"
/>
<PackageReference Include="Microsoft.AspNetCore.StaticFiles"
Version="1.1.1" />
```

In our case, the **MVC** package enables us to add Controllers to build an API and the **StaticFiles** package enables us to configure our server to serve static files from a specific directory, `/wwwroot` by default.
Since we will eventually have TypeScript files in our project, we should also disable any TypeScript compilation errors.

```
<TypeScriptCompileBlocked>true</TypeScriptCompileBlocked>
```

After saving the file, your dependencies should download automatically. If you're using MacOS/Linux, run `dotnet restore` to install the dependencies.

3. Open the Startup.cs file



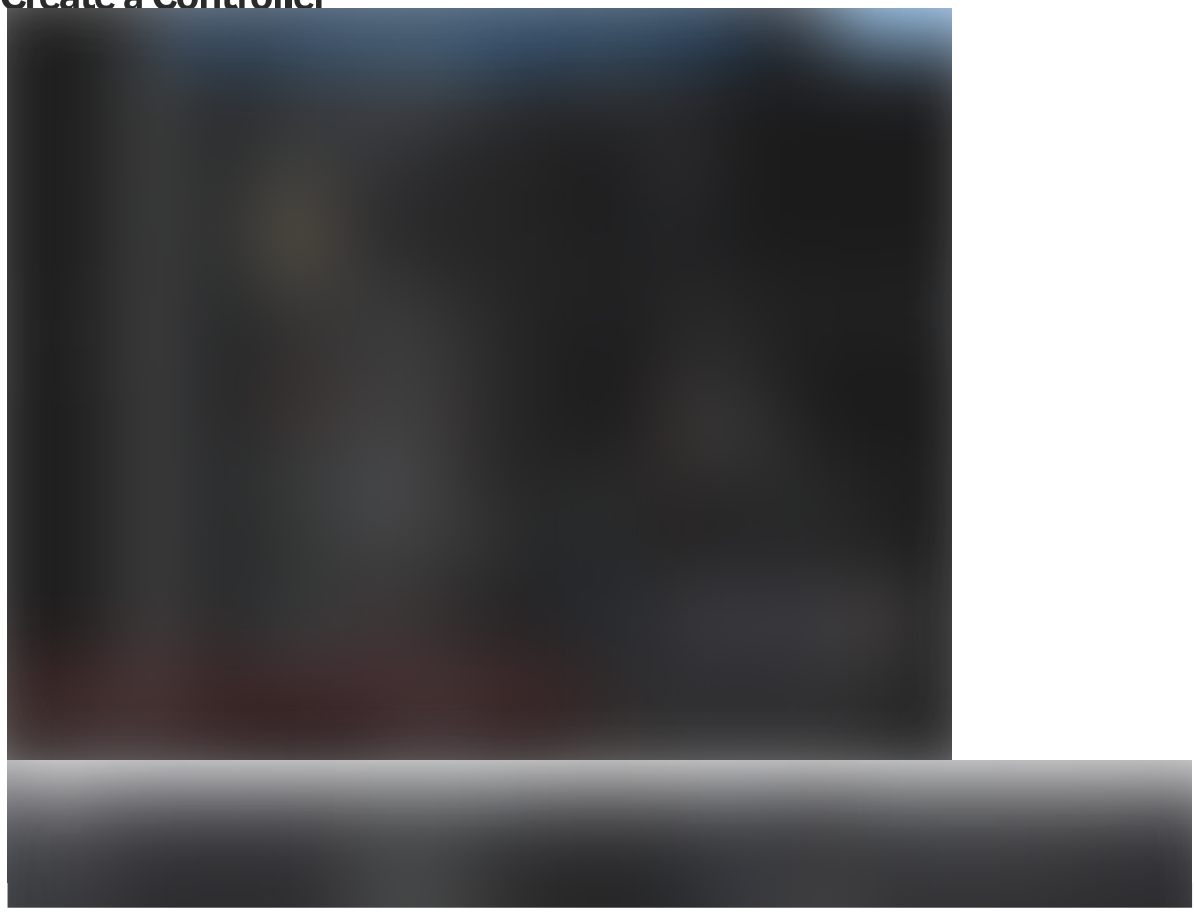
In the `ConfigureServices(...)` method, add:

```
services.AddMvc();
```

Replace everything in the `Configure(...)` method with the following:

```
app.Use(async (context, next) => {
   await next();
   if (context.Response.StatusCode == 404 &&
      !Path.HasExtension(context.Request.Path.Value) &&
      !context.Request.Path.Value.StartsWith("/api/")) {
         context.Request.Path = "/index.html";
         await next();
      }
});
app.UseMvcWithDefaultRoute();
app.UseDefaultFiles();
app.UseStaticFiles();
```

4. Create a Controller



Remove everything in the class except for the `Get()` method

```
[Route("api/[controller]")]
public class ValuesController : Controller {
[HttpGet]
   public IEnumerable<string> Get() {
      return new string[] { "Hello", "World" };
   }
}
```

Create the Angular Application

Now that the web server is built, let's add a touch of front-end dazzle.

1. Open a Command prompt in the project location

Open a terminal/command prompt and navigate to your project's directory

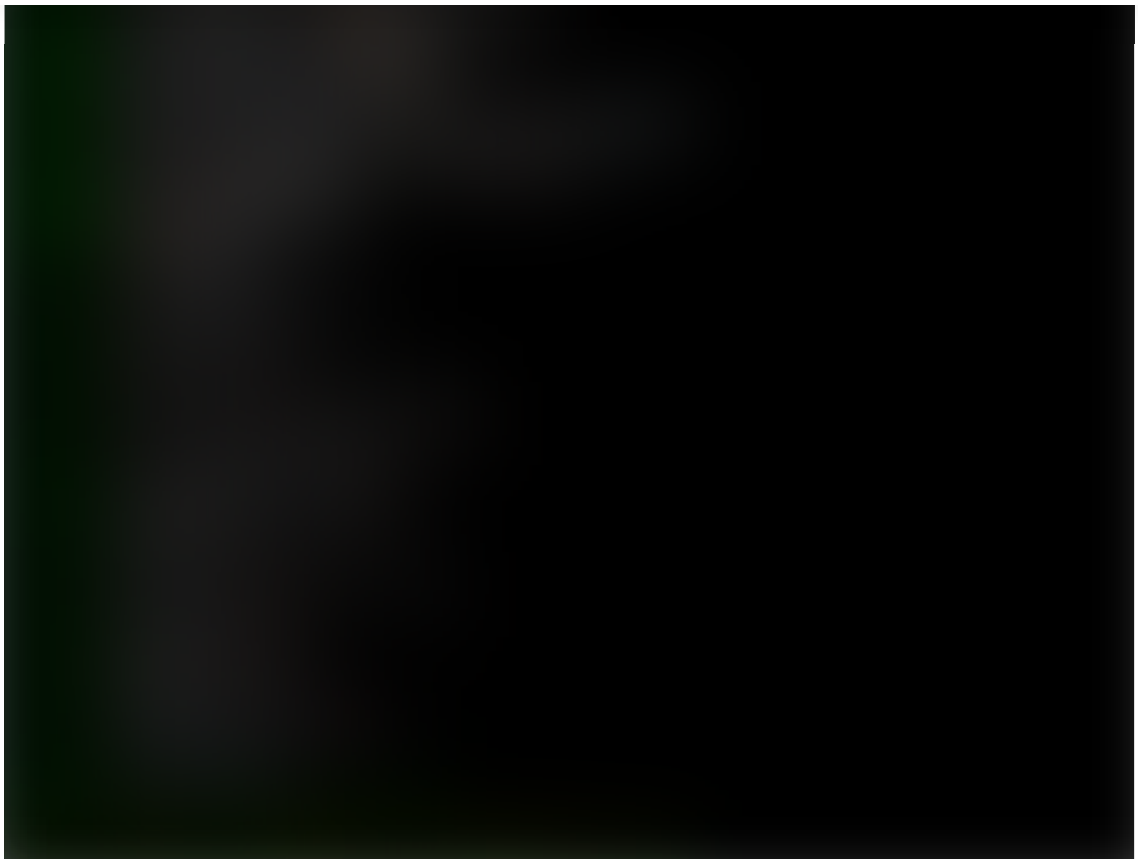
```
cd "D:\src\misc\SuperCoolApp\SuperCoolApp\"
```

2. Install the Angular-CLI

```
npm install @angular/cli --global
```

3. Scaffold a new Angular application

```
ng new {kebab-cased-app-name-here} --skip-install
```



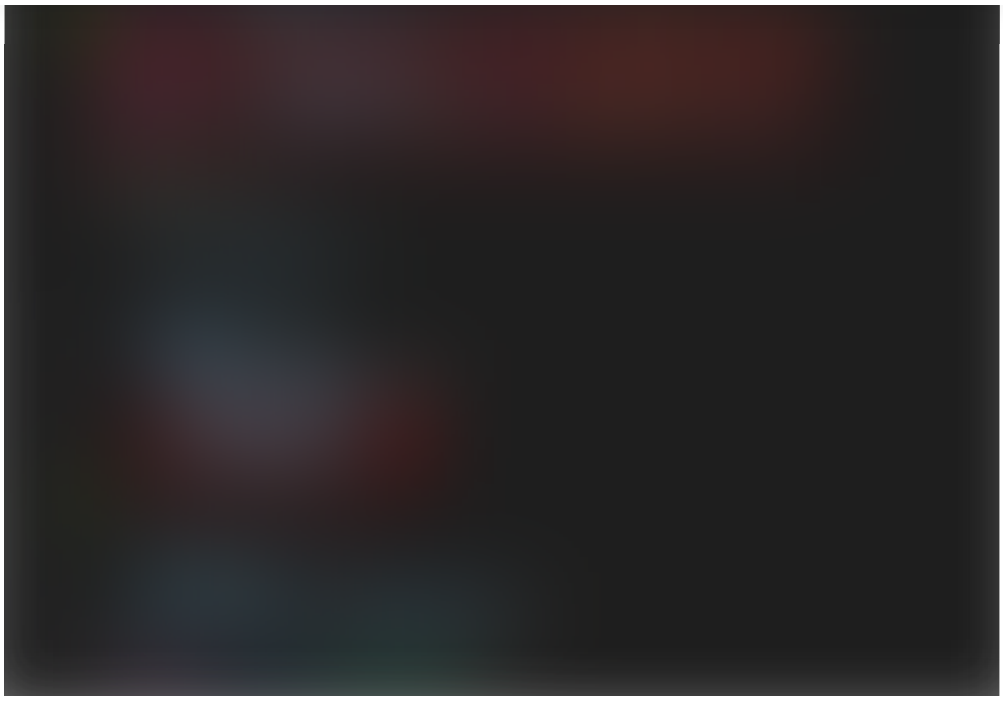
This will scaffold the Angular app without automatically installing the dependencies.

4. Move the files to the root of the project

Drag them to the project node — delete the old folder

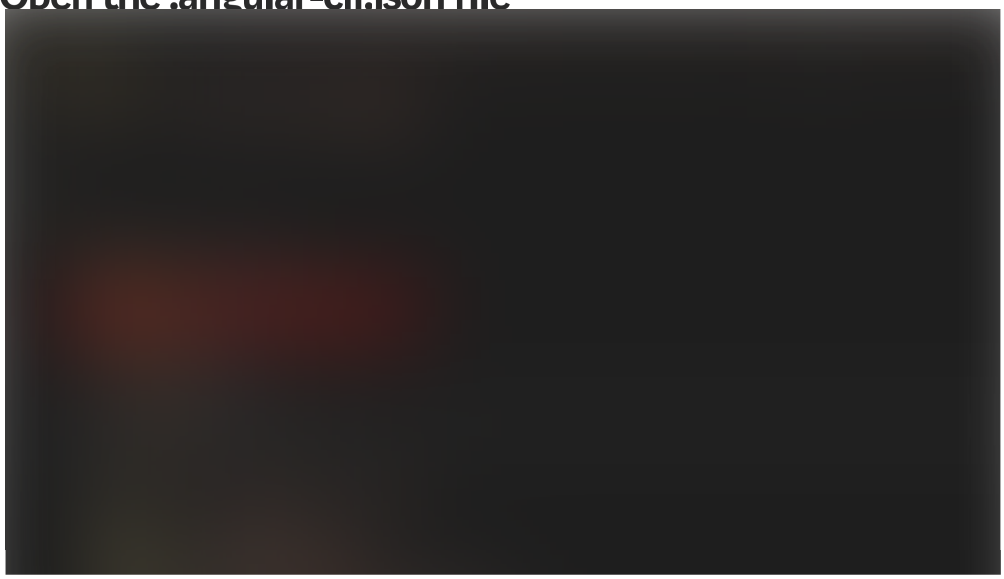
5. Enable HTTP and Form Binding

Open your `src/app/app.module.ts` file and import the `FormsModule` and `HttpModule`



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
@NgModule({
   declarations: [
      AppComponent
   ],
   imports: [
      BrowserModule,
      FormsModule,
      HttpModule
   ],
   providers: [],
   bootstrap: [AppComponent]
})
export class AppModule { }
```

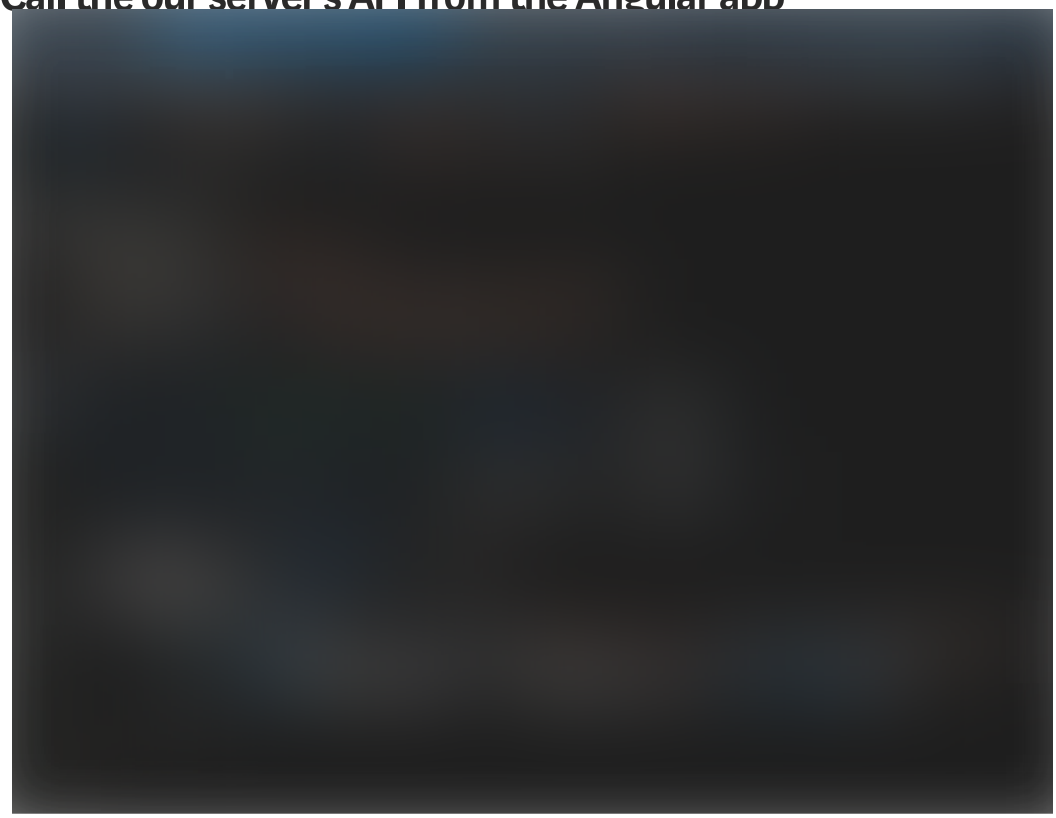
6. Open the .angular-cli.json file



Set "outDir" to "wwwroot"

When the Angular-CLI to builds the application, it will now output the assets to the `/wwwroot` directory — the same directory we configured ASP.NET Core to serve static files from.

7. Call the our server's API from the Angular app



Open the `src/app/app.component.ts` file and update it to:

```
import { Component, OnInit } from '@angular/core';
import { Http } from '@angular/http'
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
    constructor(private _httpService: Http) { }
    apiValues: string[] = [];
    ngOnInit() {
        this._httpService.get('/api/values').subscribe(values => {
            this.apiValues = values.json() as string[];
        });
    }
}
```

When Angular is running on the server, it will make a `GET` request to the `ValuesController` we created and return a string array.

app.component.html

```
    <li *ngFor="let value of apiValues">{{value}}</li>
</ul>
```

The `*ngFor` loop will iterate over each `value` of the `apiValues` array and output each one into a list item.

8. Install the Angular application's dependencies

```
npm install
```

. . .

Build and Run the Web Application

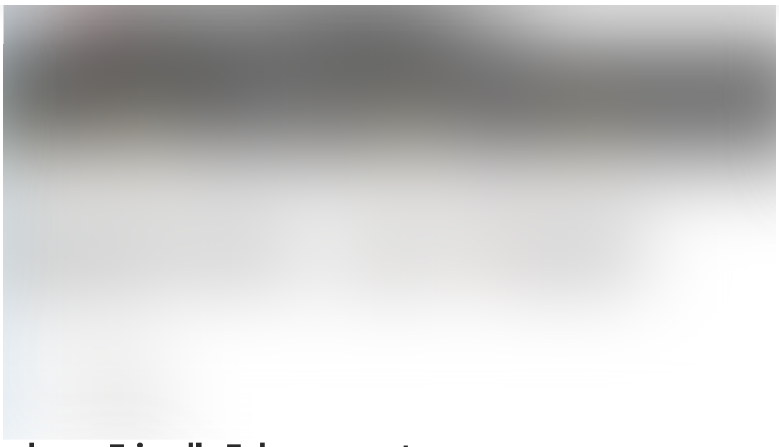
Finally we can build and run our application

1. Build the Angular application

```
ng build
```

2. Run the application

```
dotnet run
```

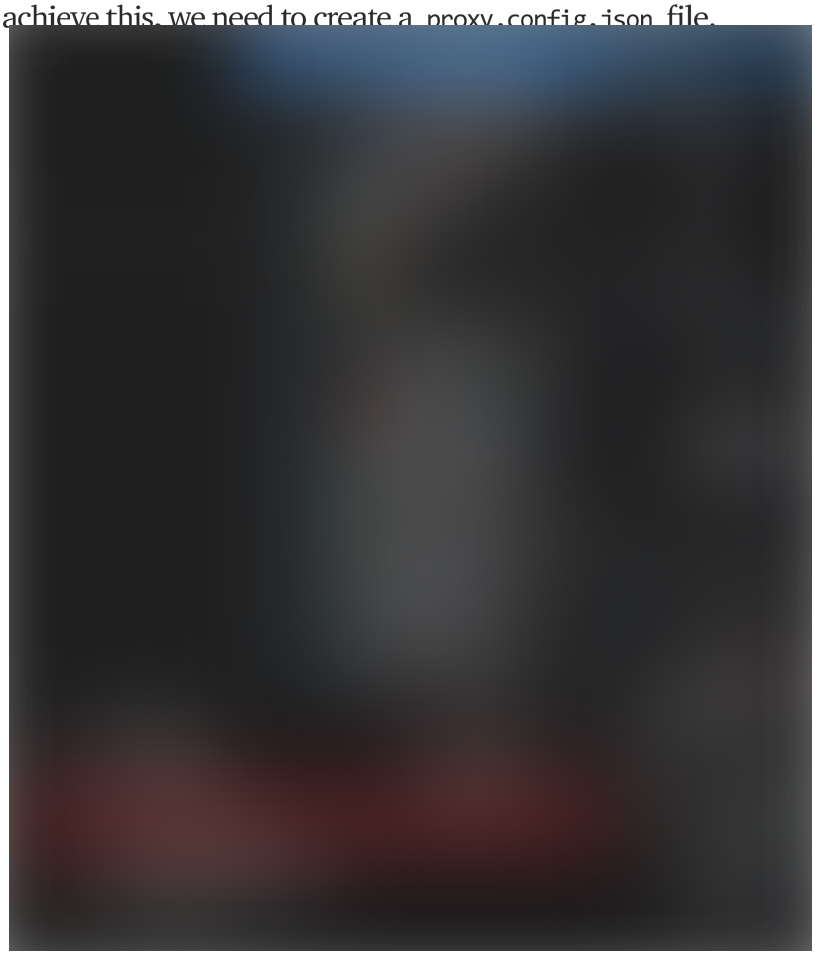


Developer-Friendly Enhancements

Next, let's simplify the development process by enabling both the Angular application and the ASP.NET Core application to rebuild whenever you make a change to its respective code.

1. Proxy your API calls to the ASP.NET Core Server

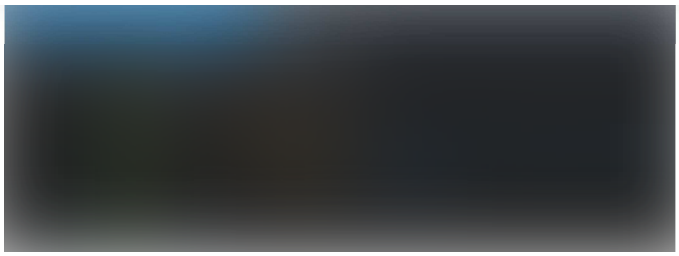
During development, we should use the `ng serve` command which watches for changes to your Angular code, transpiles the TypeScript code, and re-serves it to `localhost:4200`, by default.
Since your Angular application is being served on a different port than the API, it will send requests to `localhost:4200/api` instead of our API which is running on `localhost:5000`, by default.
To achieve this, we need to create a `proxy.config.json` file.



Create a new empty file called proxy.config.json

Add the following to your `proxy.config.json`:

```
{
   "/api": {
      "target": "http://localhost:5000",
      "secure": false
```



Set the target to your API's server address

2. Enable automatic re-compilation for ASP.NET Core

Wouldn't it be great if you were able to make any change to your server-side code and have your still-running Angular application utilize the latest changes to the API? I think so — and that's exactly what we're going to do.
Edit the `.csproj` file and add add the following:

```
<ItemGroup>
   <DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools"
Version="1.0.0" />
</ItemGroup>
```

Install Microsoft.DotNet.Watcher.Tools Version 1.0.0

Again, if you're VS 2017, it will auto-restore the dependencies once the file is saved, otherwise run `dotnet restore`

3. Run both applications in watch mode

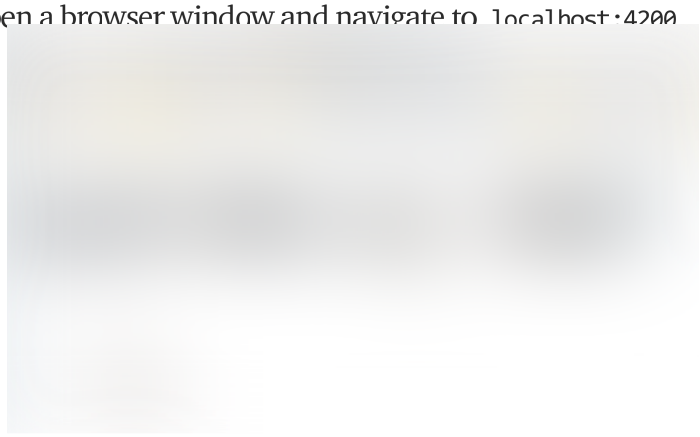
Open a terminal in the project directory and start the ASP.NET Core server

```
dotnet watch run
```

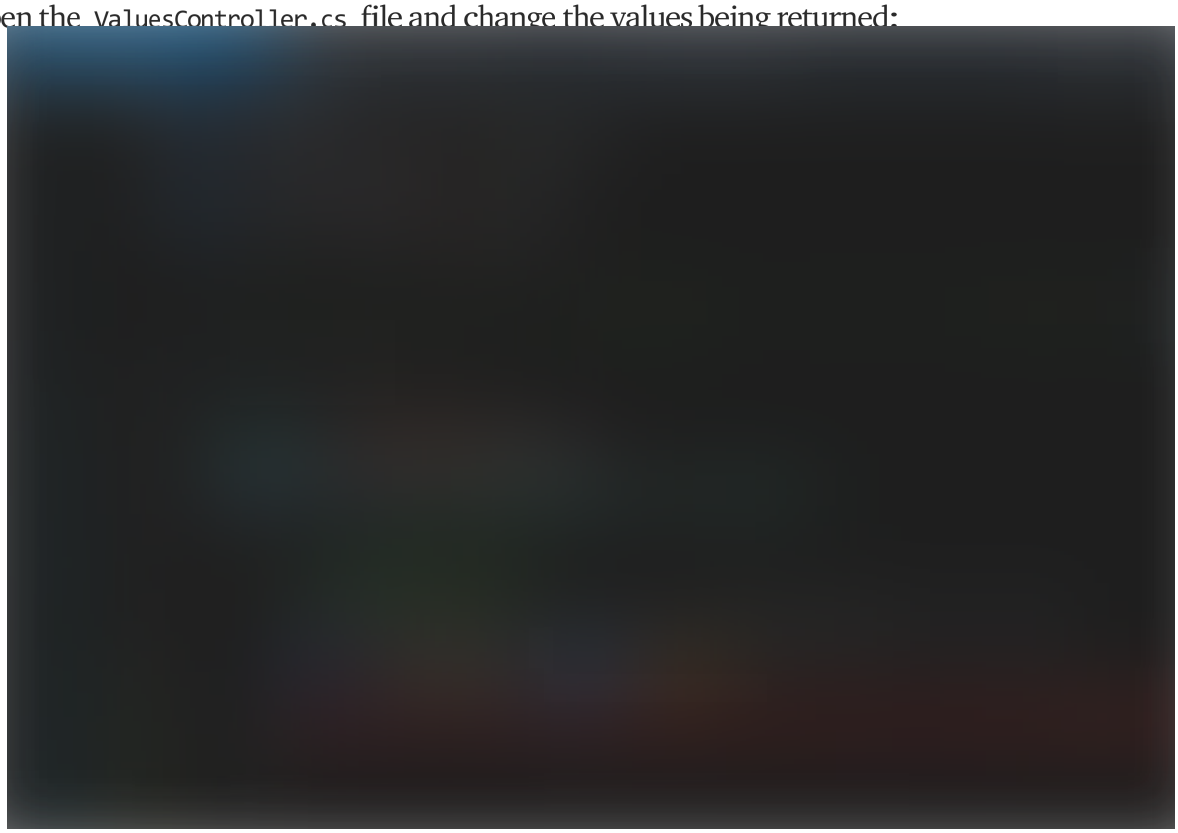
Open another terminal and start the Angular application

```
ng serve --proxy-config proxy.config.json
```

Open a browser window and navigate to `localhost:4200`

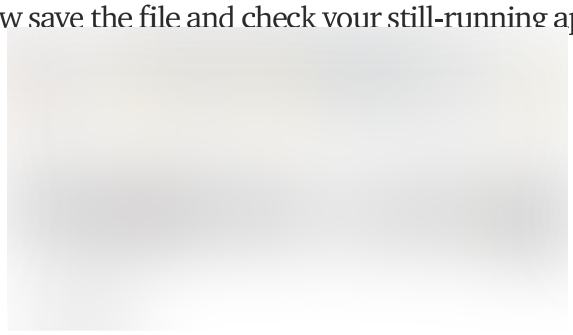


Open the `ValuesController.cs` file and change the values being returned:



Open the `src/app/app.component.html` file and change the header:

Now save the file and check your still-running application magically update!



All done with the App

From here, you can deploy the app to Azure or start building your Angular application.
Want to setup a Continuous Integration build pipeline to publish your application to Azure?

How to deploy an Angular CLI application built on ASP.NET

Let's learn how to deploy the app we just built as an Azure Web App using Continuous Integration. The end goal is to be...

medium.com

Angular-Tour_of_Heroes_App

Angular is a development platform for building mobile and desktop web applications

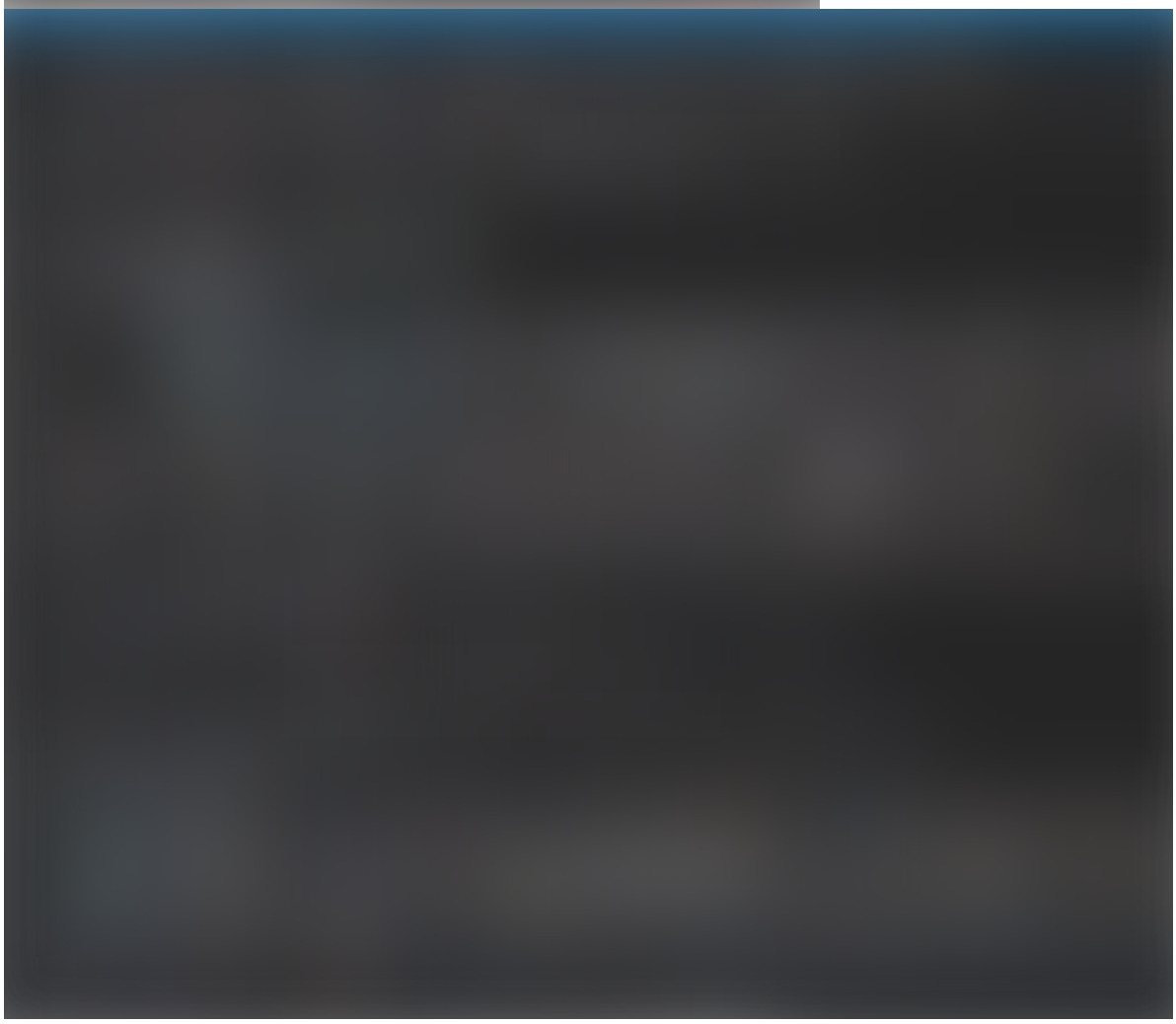
angular.io

Literally One-Click Publishing from Visual Studio

Wouldn't it be convenient for Visual Studio to automatically build the Angular app for the production environment whenever you click the publish button? I think so.
Open your `.csproj` and add the following:

```
<Target Name="Build Angular"
Condition="'$(Configuration)'=='Release'" BeforeTargets="Build">
   <Message Text="* * * * * * Building Angular App * * * * * *"
Importance="high" />
   <Exec Command="ng build -prod -aot" />
</Target>
```

Now publish the app:



If you notice that ng build seems to have executed twice then you're not alone

If your .NET Core app is configured to deploy to Azure then the latest code is now live on the interwebs for all the world to see. How neat is that?
Now go forth and build cool shit!

Angular2 Aspnetcore Angular C Sharp Visual Studio

Levi Fuller

Follow

Work hard. Be a good person. Enjoy life.

See responses (122)

More From Medium

Related reads

.Net Core Email Sender Library with Razor Templates (.cshtml) contained in it

Gavril... Nov 7, 201... 318

Related reads

An awesome guide on how to build RESTful APIs with ASP.NET Core

Evandro Gomes ... Feb 21 · 43 min... 3.7K

Related reads

How to build a single page application using server-side Blazor

Ankit Sharma in... Aug 17, 2018 · 15... 724