

By Pranavsai Gandikota

Audience Analysis:

My intended audience is students who are trying to learn full-stack web development, those who have just started to learn Angular, and people looking to make a website who may need to make a website for their personal needs. Some people may have learned frontend and backend and may want a quick start to guide them in setting up their project to continue where they left off, which this guide helps in as well. It is more suited for those starting with angular framework development.

Coming to what the audience knows, it can be expected that they have a little prior knowledge on HTML basics, JavaScript and C#. If they are a developer, they may have used Visual Studio or Visual Studio Code sometime before as well. Regardless of their educational background, they would be able to set up a project and understand keywords. It is important to have a clear set of instructions, straight to the point, along with multiple screenshots of every step as some may have no clue what they may be clicking. In terms of distractions, if any, it would be having to go back and forth between tabs in certain steps. Any confusion will be overcome with the screenshots included to help guide through steps. There will also be a lot of spaced text and clean code so that it is simple to paste into the terminal and understand.

Full-Stack Web App Setup Guide Using Angular 2+, .NET Core API, SQL Server Developer, and Bootstrap

Introduction


This setup guide aims to help create a full stack web development environment with the help of:

- Angular 2+ (For front end)
- .NET Core Web API (For backend)
- SQL Server Developer (Database)
- Bootstrap (Optional for styling)

Who this guide is best for:


This guide is aimed at those who are beginning web development or are switching to Angular and want a simple quick guide to setting up all the required tools and connecting the back and front ends. Although this guide was made for the Windows operating system, most of the steps such as using Angular, Node.js and SQL Server can be applied for Mac or Linux.

Any terminology used can be found in the following site:

<https://v17.angular.io/guide/glossary> and <https://angular.dev/overview> for more information on Angular and <https://www.c-sharpcorner.com/article/what-is-dot-net-core/> for more information regarding .NET Core. 

At the end, you will have a working site which displays product data we add manually along with required elements installed and ready to use for building your own full-stack application.

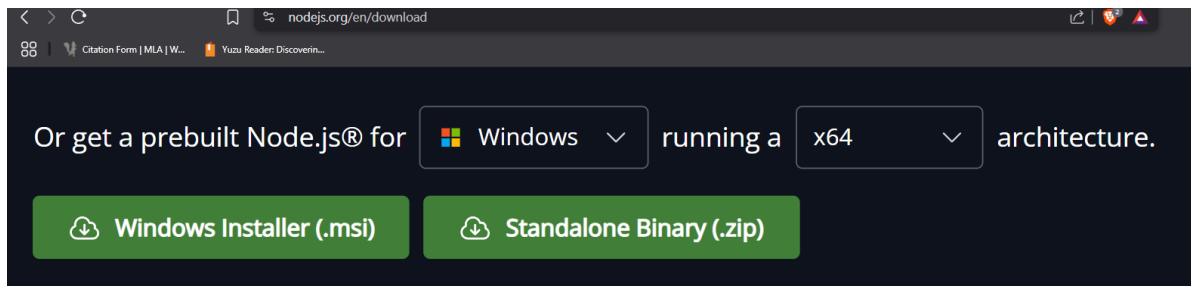
What we will use:

- Node.js and Angular CLI
- Visual Studio 2022 community edition (with .NET)
- SQL Server Developer Edition and SSMS (SQL Server Management Studio) 
- Visual Studio Code (optional)
- Bootstrap and Entity Framework

Note: that there is no need to configure authentication or advanced security, all tools are free, and default settings will be used for most of the steps.

Step 1: Creating the Environment

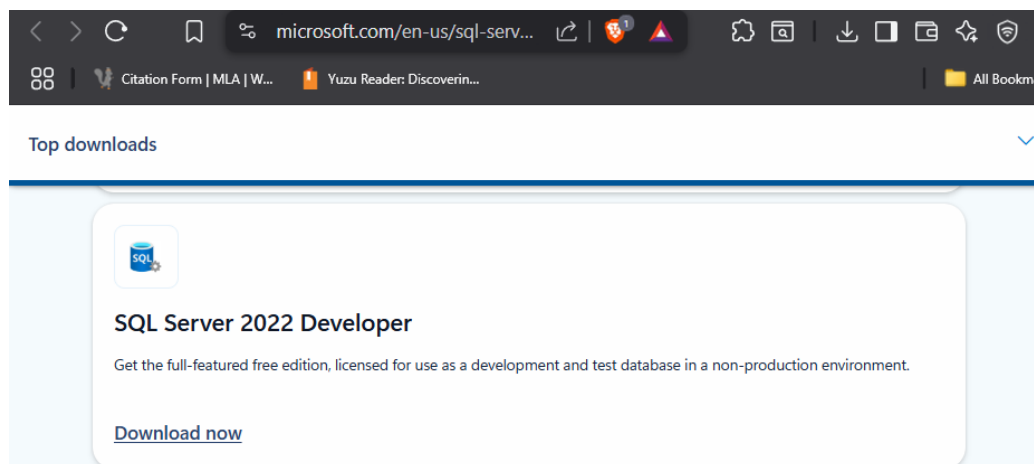
- 1) Begin by **Installing Node.js** for running Angular and installing dependencies. System respective install can be found here: (<https://nodejs.org/en/download>)



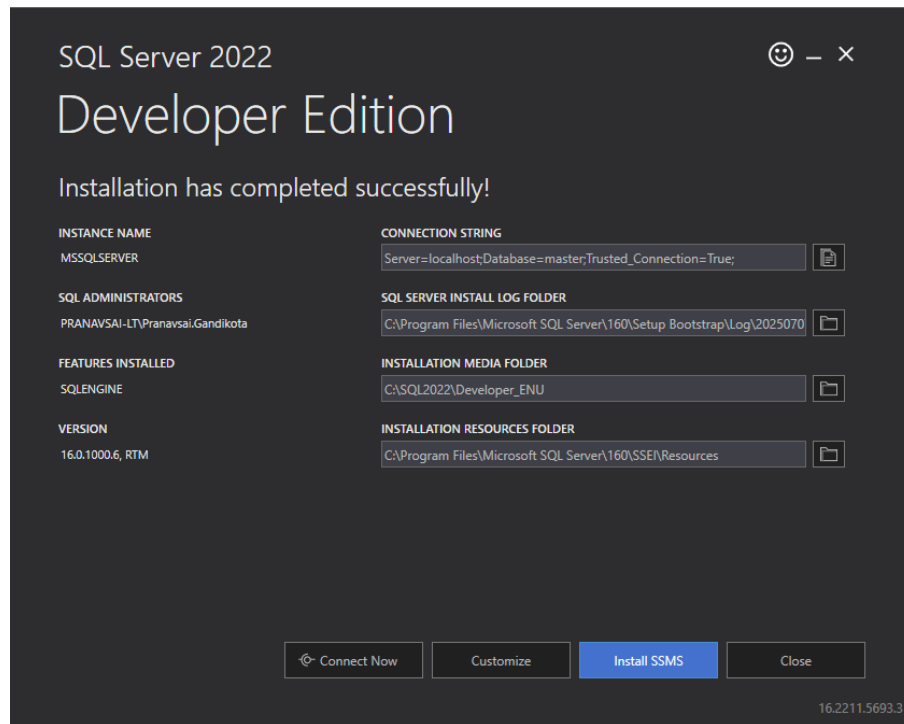
- 2) **Install Angular CLI (Command Line Interface)** by running the command below in the terminal:

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

- 3) **Install Visual Studio 2022** (Community Edition) from (<https://visualstudio.microsoft.com/downloads/>) and select **ASP.NET and web development** during setup. It is the Integrated Development Environment for .NET applications.
- 4) **Install SQL Server Developer and SSMS** (from <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>) Select **“developer”** sql server and then select **“Basic”** on the next setup screen.



- 5) Once done **install SSMS** from the screen here (or from online). This gives a GUI to create or view databases and run SQL queries. (Install page image shown below)



Step 2 - Determine which version of SQL Server Management Studio to install

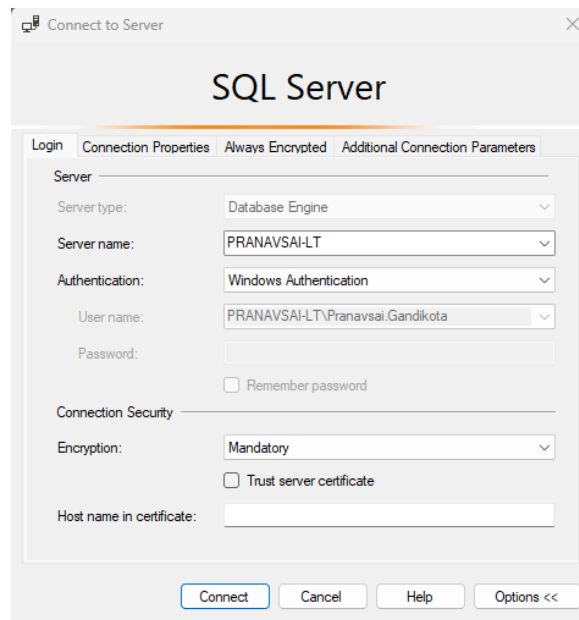
Decide which version of SSMS to install. The most common options are:

- The latest release of SQL Server Management Studio 21 that is hosted on Microsoft servers. To install this version, select the following link. The installer downloads a small *bootstrapper* to your *Downloads* folder.

[Download SSMS 21](#)

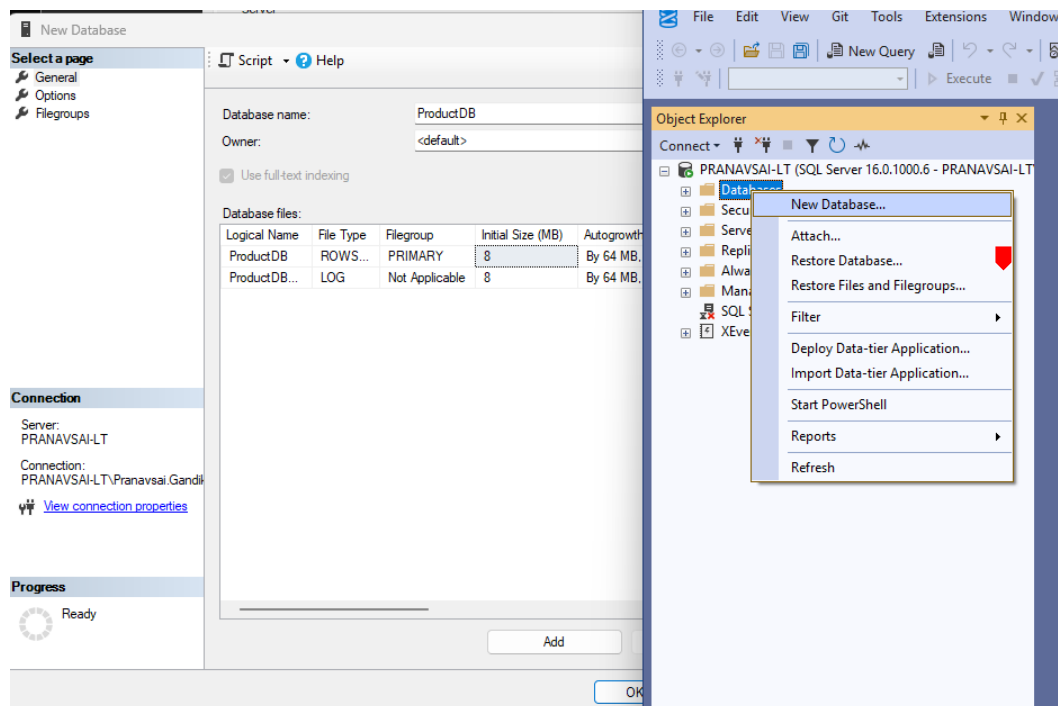
- If you already have SQL Server Management Studio 21 installed, you can install another version alongside it.

6) Once done, **open** the SQL Server Management Studio (SSMS), you should see this:



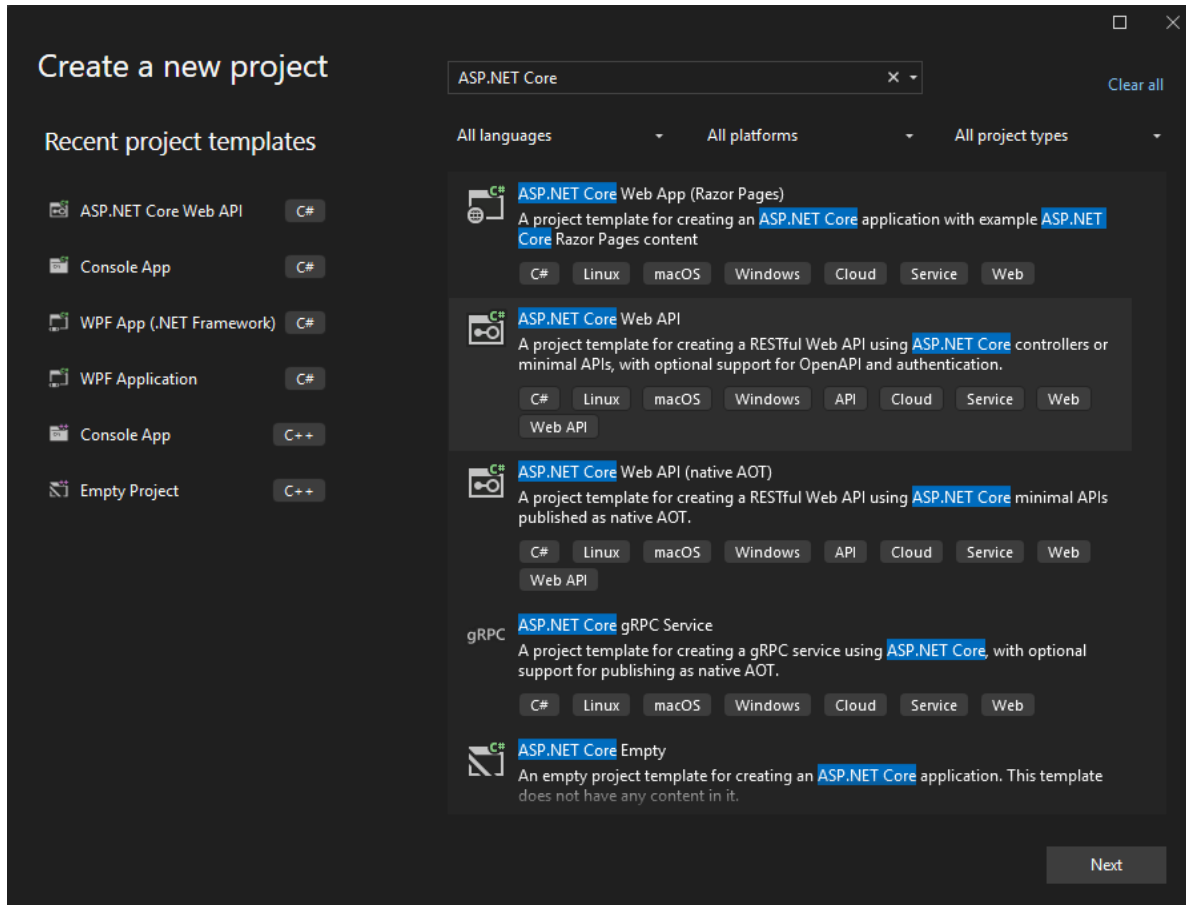
7) **Click Connect**, and “yes” if any message is prompted.

8) In SSMS under the Object Explorer, right-click Databases, **select New Database**, and **name it ProductDB** (or anything) and click ok.



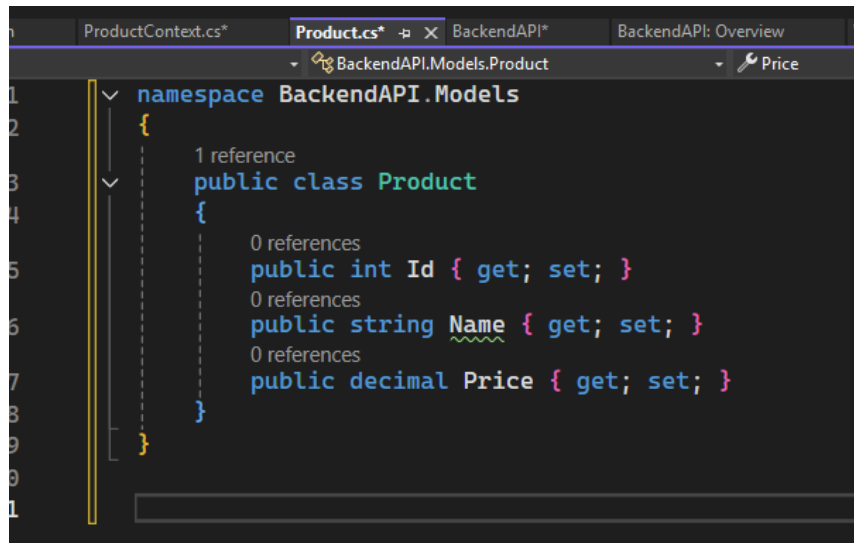
Step 2: Creating the .NET Core Backend

- 1) Open Visual Studio and **create** a project by choosing **ASP.NET Core Web API** with project name: **BackendAPI**



- 2) Once made, **open the solution explorer** on the right and right click in the empty area and under “Add”, click on “New Folder”, and name it “**Models**”
- 3) Right-click Models folder, Add -> Class, and name it **Product.cs**.

4) Add the following code in Product.cs:

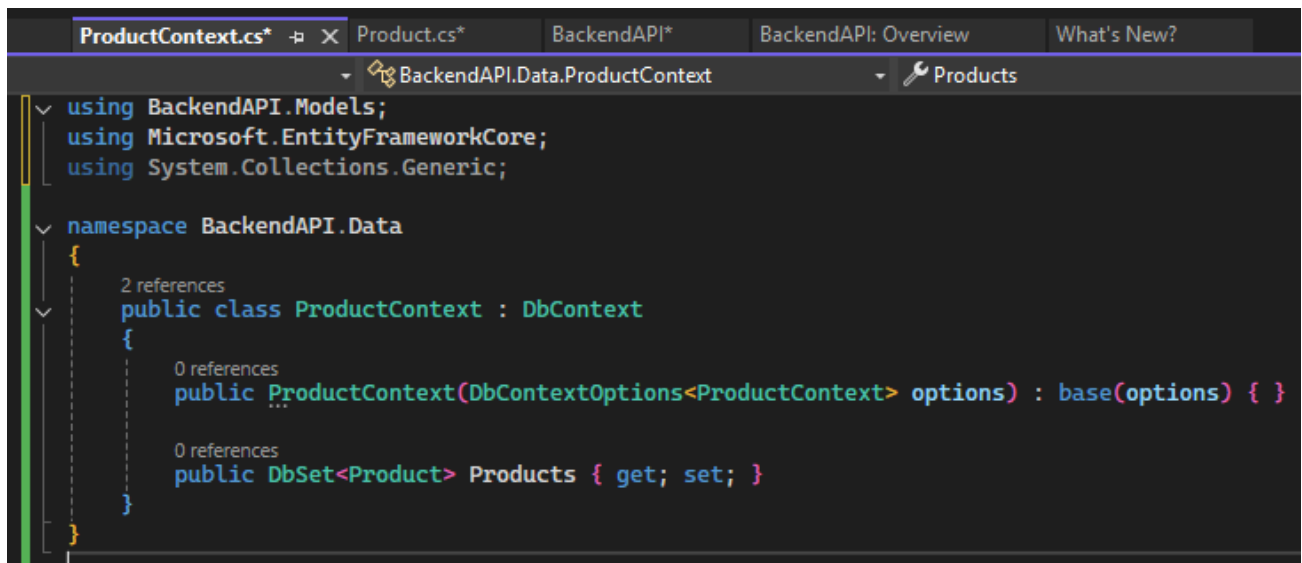


```
1 namespace BackendAPI.Models
2 {
3     1 reference
4     public class Product
5     {
6         0 references
7         public int Id { get; set; }
8         0 references
9         public string Name { get; set; }
10        0 references
11        public decimal Price { get; set; }
12    }
13 }
```

(Ignore any squiggly lines and errors for now)

5) Similarly, click on the project and make another new folder called: Data

6) Right click the “Data” folder and add a class called **ProductContext.cs** with the code below:



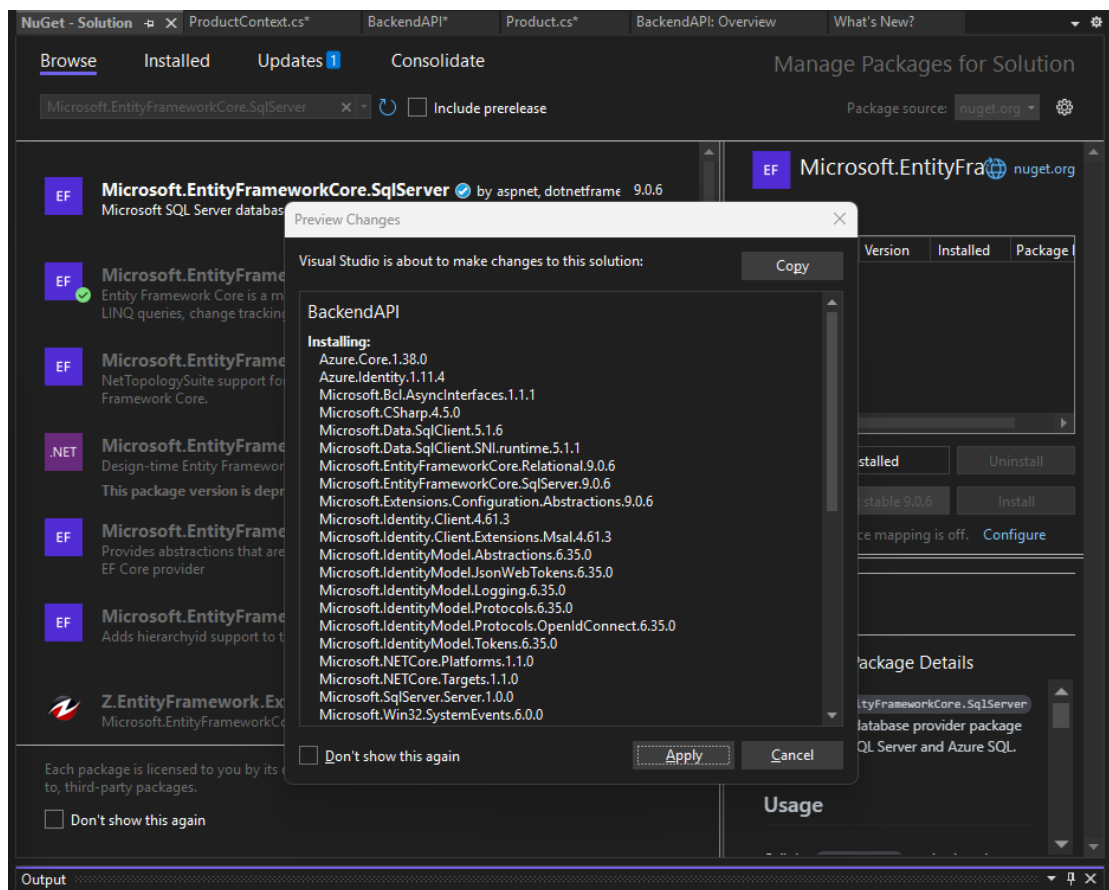
```
1 using BackendAPI.Models;
2 using Microsoft.EntityFrameworkCore;
3 using System.Collections.Generic;
4
5 namespace BackendAPI.Data
6 {
7     2 references
8     public class ProductContext : DbContext
9     {
10         0 references
11         public ProductContext(DbContextOptions<ProductContext> options) : base(options) { }
12
13         0 references
14         public DbSet<Product> Products { get; set; }
15     }
16 }
```

Now to install the packages to help solve the errors that you may be getting.

7) Under tools on the top bar, open the **NuGet Package Manager** and then click on **Manage NuGet Packages for this Solution**. Look up the names below, install and apply them (accept any agreements and prompts you may get).

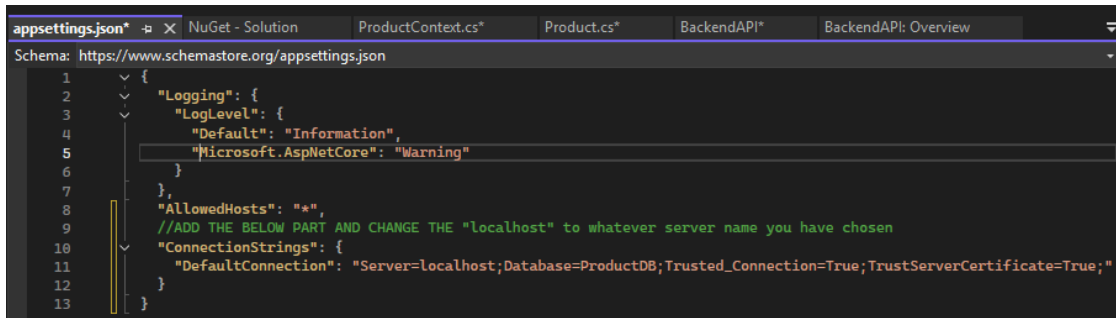
- i. Microsoft.EntityFrameworkCore
- ii. Microsoft.EntityFrameworkCore.SqlServer
- iii. Microsoft.EntityFrameworkCore.Tools

Entity Framework helps use C# classes instead of SQL queries to help on CRUD (Create, Read, Update, Delete) operations.



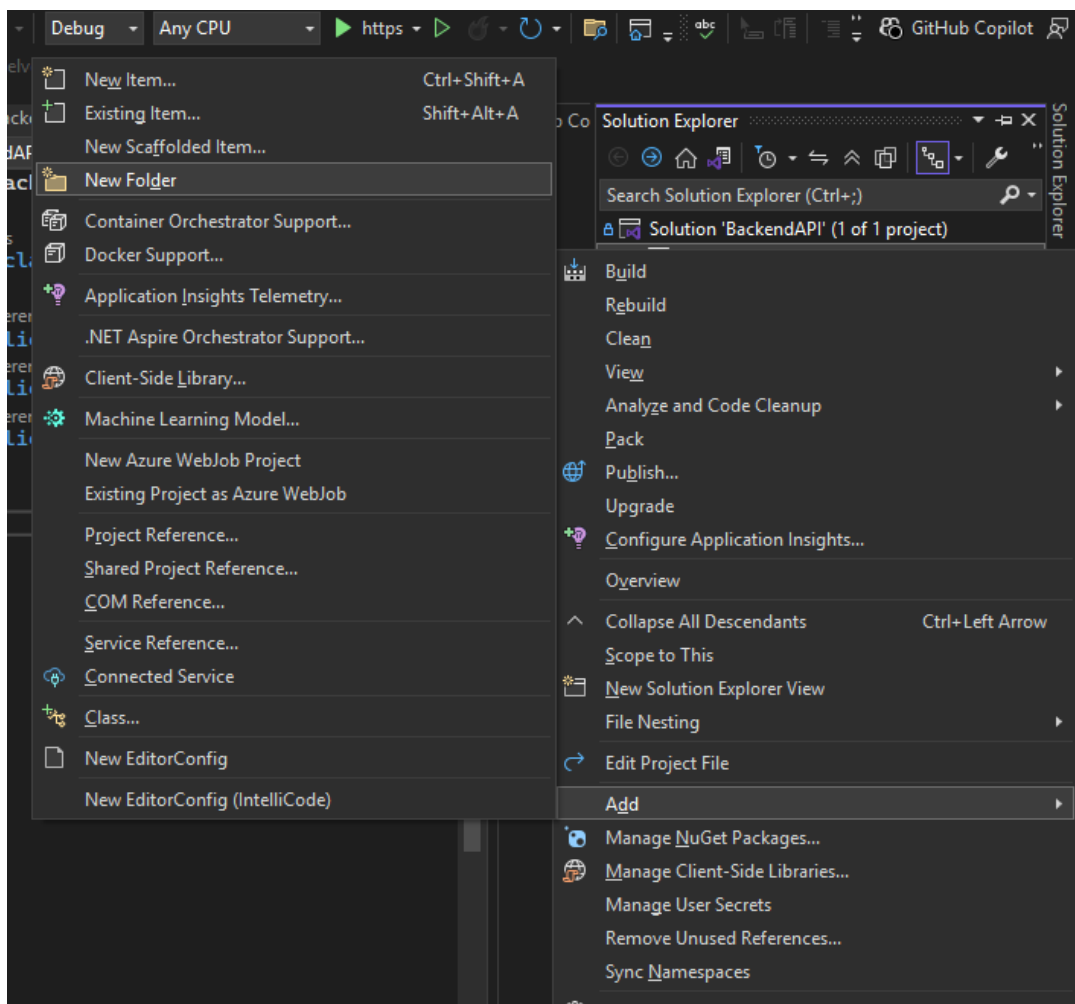
All errors should now go away.

8) In the appsettings.json put the code given below



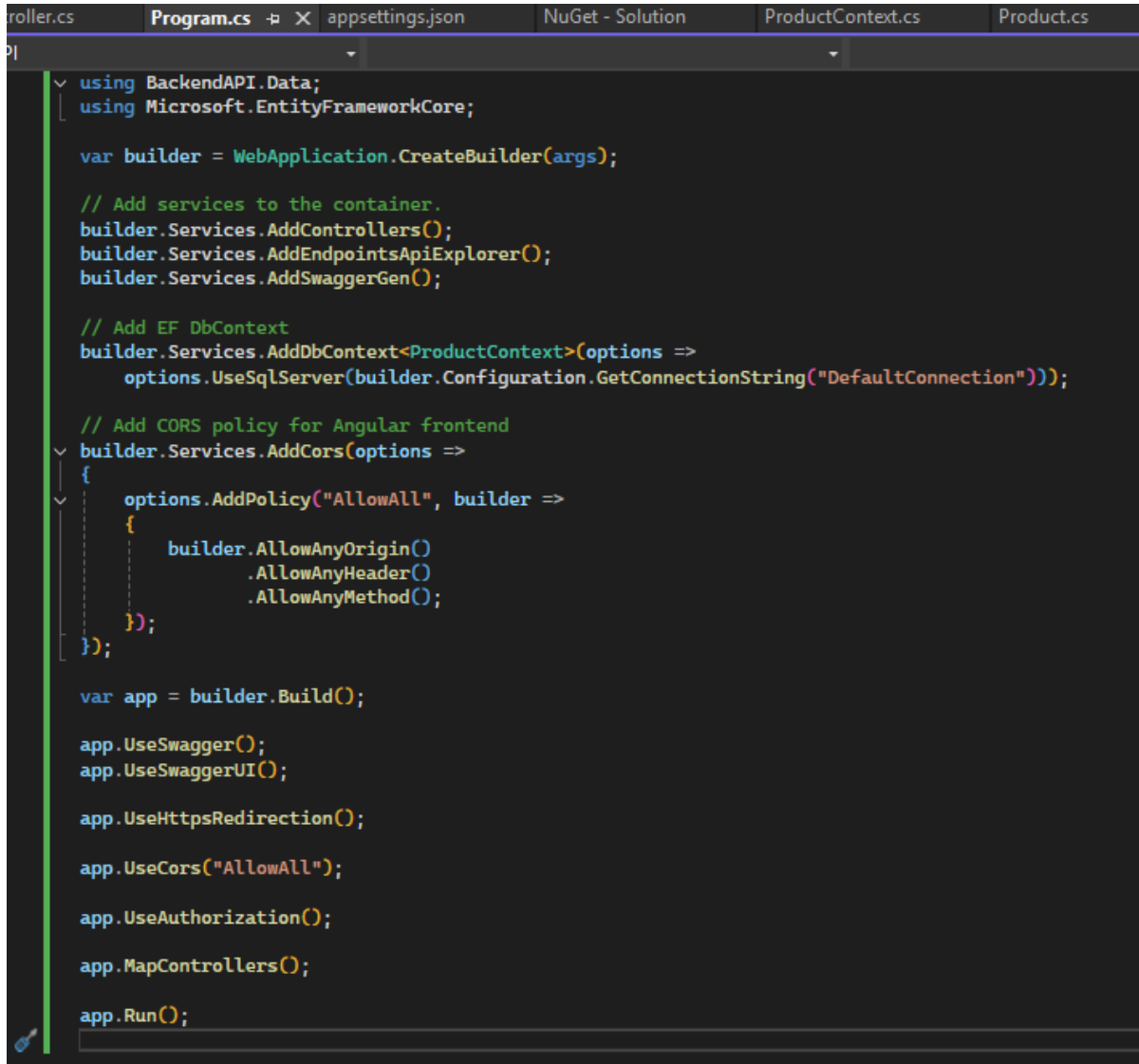
The screenshot shows the 'appsettings.json' file in a Visual Studio editor. The file is open in a tab labeled 'appsettings.json'. The schema is 'https://www.schemastore.org/appsettings.json'. The code is as follows:

```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "AllowedHosts": "*",
9   //ADD THE BELOW PART AND CHANGE THE "localhost" to whatever server name you have chosen
10  "ConnectionStrings": {
11    "DefaultConnection": "Server=localhost;Database=ProductDB;Trusted_Connection=True;TrustServerCertificate=True;"
12  }
13 }
```



- 9) Update the program.cs to configure services (shown below)

Cross-Origin Resource Sharing (CORS) is a browser security feature to prevent frontend apps (e.g. Angular) from making requests hosted on different backend APIs. Here our Angular connection relates to .NET Core API on different ports, and to let it do so we enable CORS.



```
Program.cs
using BackendAPI.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Add EF DbContext
builder.Services.AddDbContext<ProductContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Add CORS policy for Angular frontend
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", builder =>
    {
        builder.AllowAnyOrigin()
            .AllowAnyHeader()
            .AllowAnyMethod();
    });
});

var app = builder.Build();

app.UseSwagger();
app.UseSwaggerUI();

app.UseHttpsRedirection();

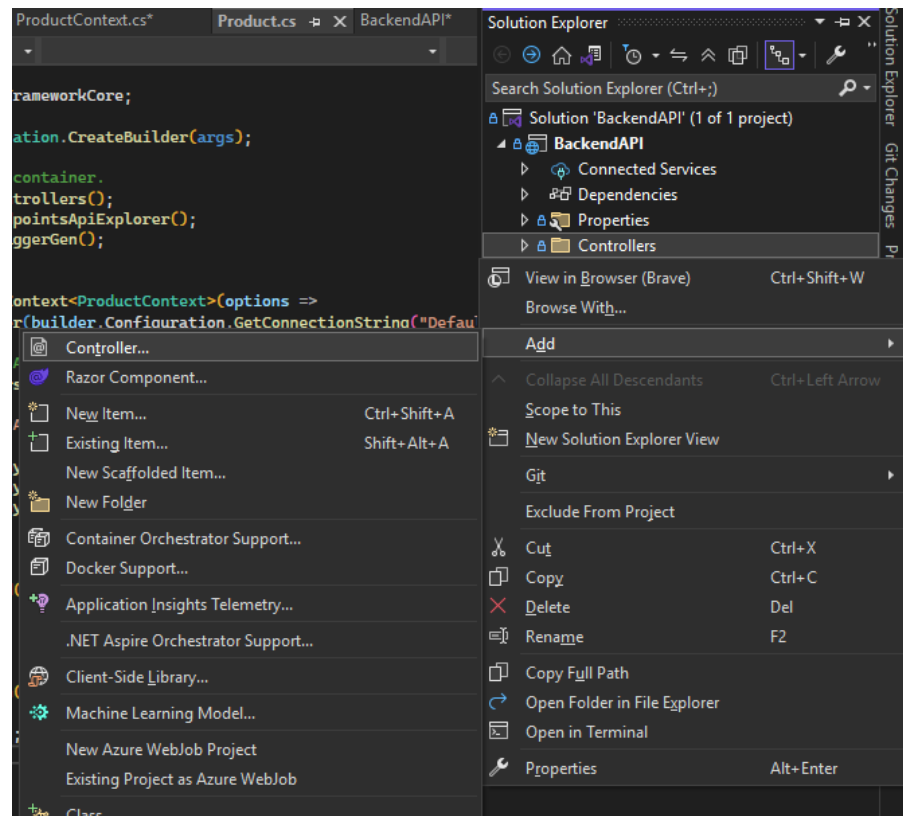
app.UseCors("AllowAll");

app.UseAuthorization();

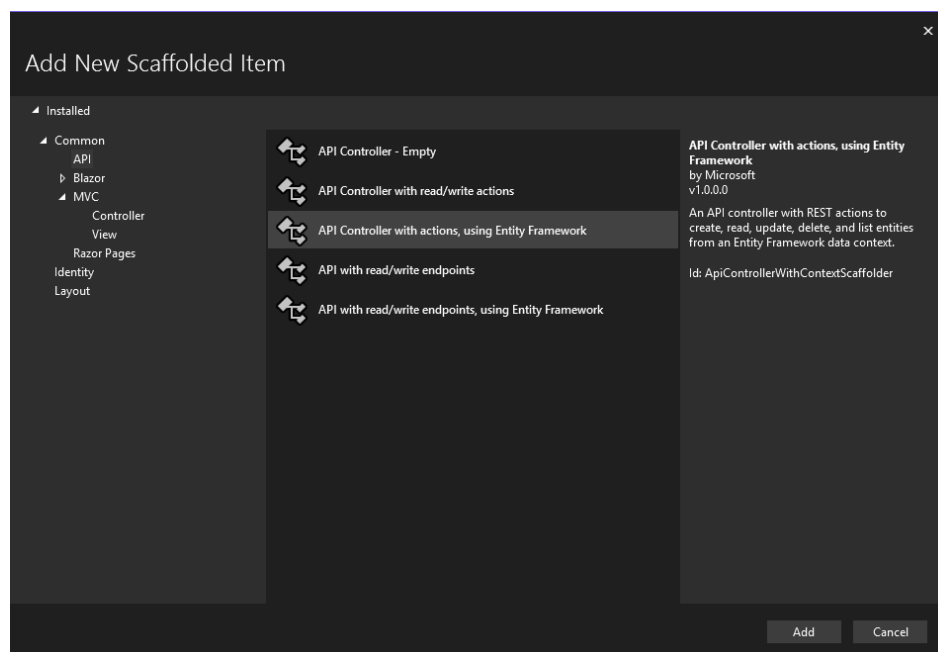
app.MapControllers();

app.Run();
```

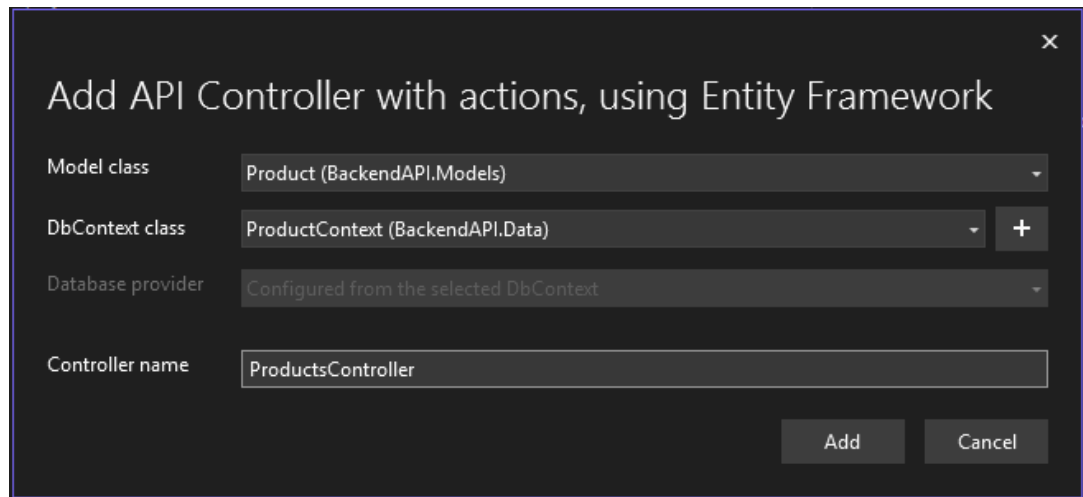
10) To create the API controller, right click the **Controllers** folder and add a controller.



11) Choose **API controller with actions using Entity Framework** under the API dropdown.



- 12) Select Product as model and ProductContext as DbContext to generate ProductsController.cs.



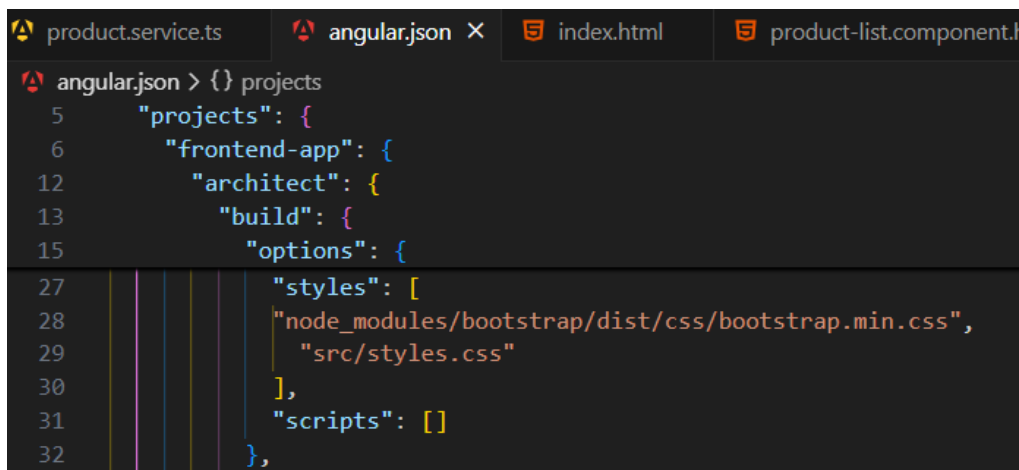
Step 3) Creating the Angular App

- 1) **Run** the following in the terminal (it can be opened from the view tab on top):

```
ng new frontend-app --routing --style=css
cd frontend-app
npm install
npm install bootstrap
```

- 2) In angular.json, under the "styles" array, add:

```
"node_modules/bootstrap/dist/css/bootstrap.min.css"
```

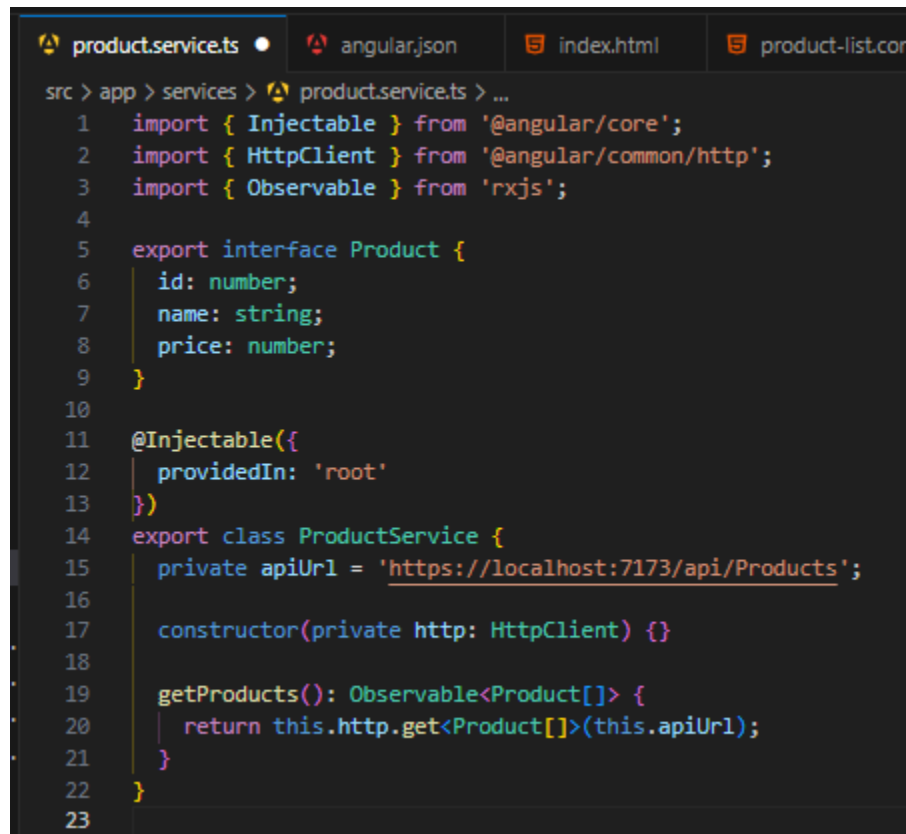


Step 4) Making a Product Service in Angular

1) Run in Terminal:

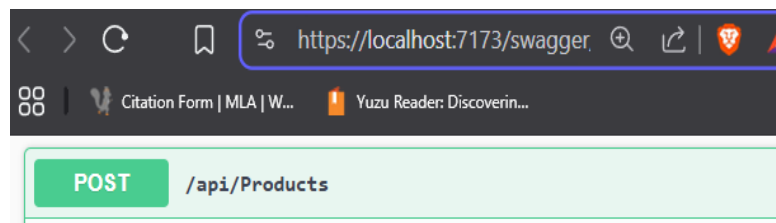
ng generate service services/product

2) Open **product.service.ts** and put the code given below in:



```
src > app > services > product.service.ts > ...
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3  import { Observable } from 'rxjs';
4
5  export interface Product {
6      id: number;
7      name: string;
8      price: number;
9  }
10
11  @Injectable({
12      providedIn: 'root'
13  })
14  export class ProductService {
15      private apiUrl = 'https://localhost:7173/api/Products';
16
17      constructor(private http: HttpClient) {}
18
19      getProducts(): Observable<Product[]> {
20          return this.http.get<Product[]>(this.apiUrl);
21      }
22  }
23
```

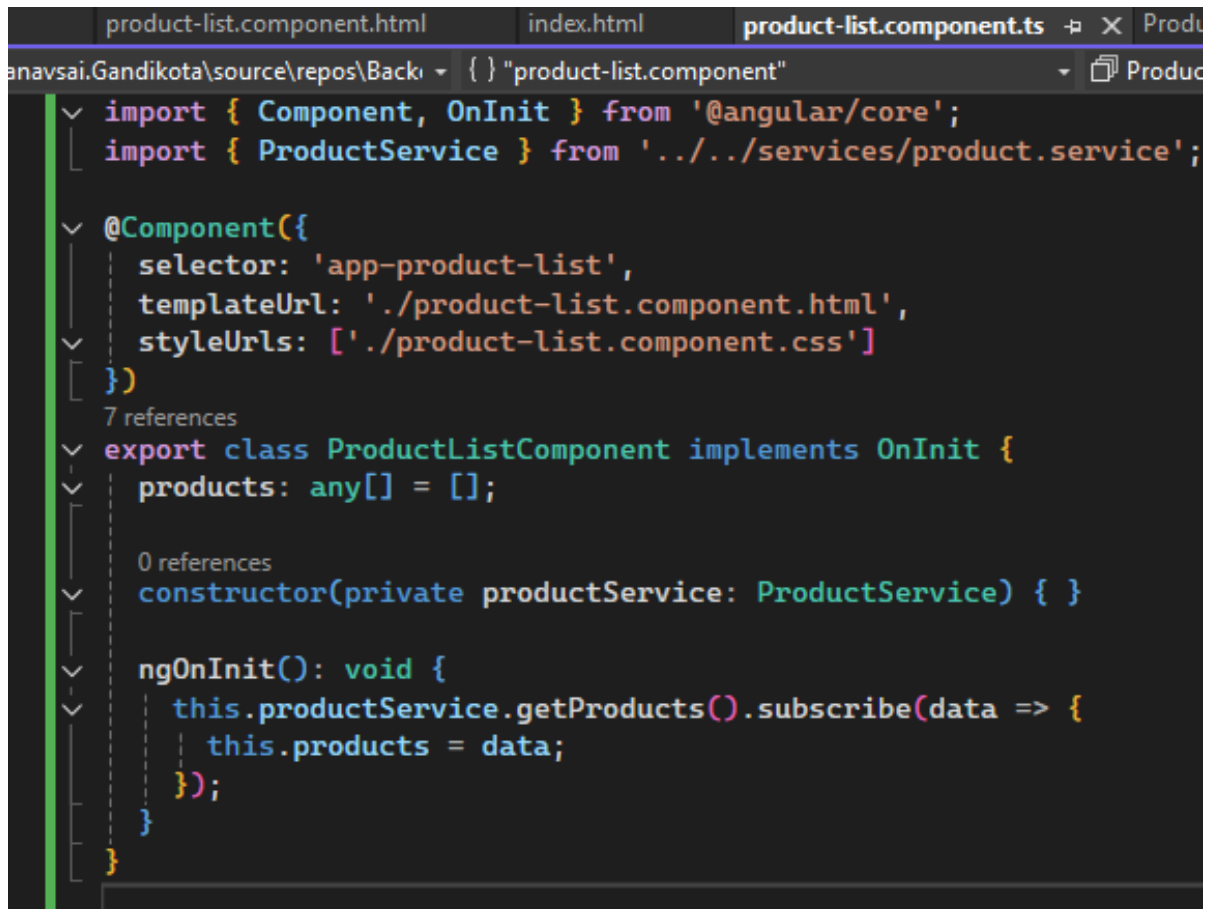
Make sure that the apiUrl inserted matches the url formed by Swagger.



(If following along with this tutorial **add the “/api/Products”** after the localhost:7173)

Step 5) To Display Products in the Angular Component

- 1) **Run** the line in the terminal:
ng generate component components/product-list
- 2) Update the **product-list.component.ts** with the code given below.



```
import { Component, OnInit } from '@angular/core';
import { ProductService } from '../services/product.service';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {
  products: any[] = [];

  constructor(private productService: ProductService) { }

  ngOnInit(): void {
    this.productService.getProducts().subscribe(data => {
      this.products = data;
    });
  }
}
```

- 3) Also update the product-list.component.html to show the bootstrap table with the data:

```
app.module.ts  product-list.component.html  index.html  product-list.component.ts  ProductsController.cs
1  <div class="container mt-5">
2    <h2 class="mb-4 text-center">Product List</h2>
3
4    <div class="table-responsive">
5      <table class="table table-bordered table-hover align-middle text-start">
6        <thead class="table-secondary">
7          <tr>
8            <th scope="col">ID</th>
9            <th scope="col">Product Name</th>
10           <th scope="col">Price ($)</th>
11         </tr>
12       </thead>
13       <tbody>
14         <tr *ngFor="let product of products">
15           <td>{{ product.id }}</td>
16           <td>{{ product.name }}</td>
17           <td>{{ product.price }}</td>
18         </tr>
19       </tbody>
20     </table>
21   </div>
22
23   <div *ngIf="products.length === 0" class="alert alert-info text-center mt-4">
24     No products found.
25   </div>
26 </div>
27
```

Quick Checks:

Make sure to update the app.module.ts too to match the following to avoid any errors (you may need to import the HttpClientModule in your app.module.ts):

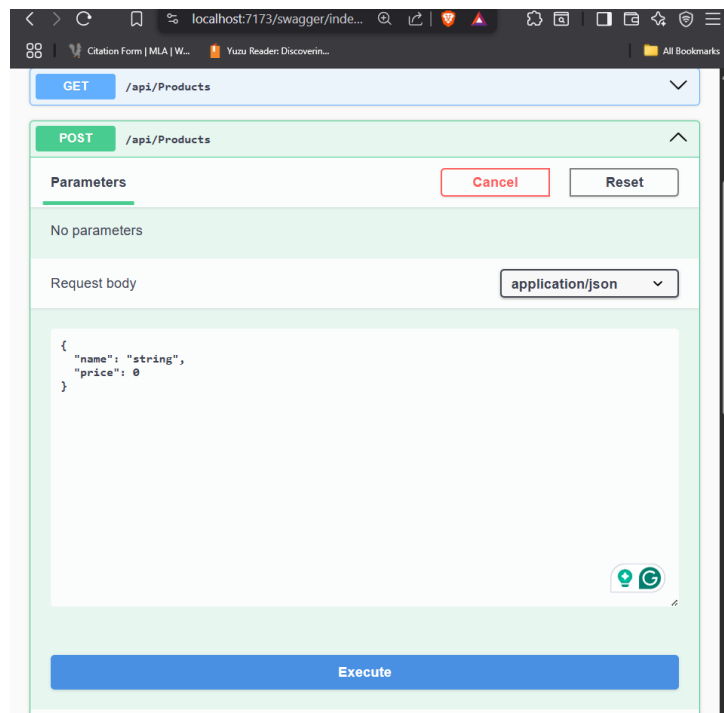
```
app.module.ts  product-list.component.html  index.html  product-list.component.ts  ProductsController.cs
C:\Users\Pranavsai.Gandikota\source\repos\Backi > { } "app.module"  ProductListComponent
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { ProductListComponent } from './components/product-list/product-list.component';
7  import { HttpClientModule } from '@angular/common/http';
8
9  @NgModule({
10   declarations: [
11     AppComponent,
12     ProductListComponent
13   ],
14   imports: [BrowserModule, AppRoutingModule, HttpClientModule],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
```

Step 6) Adding (POST) Data into the Table through Swagger

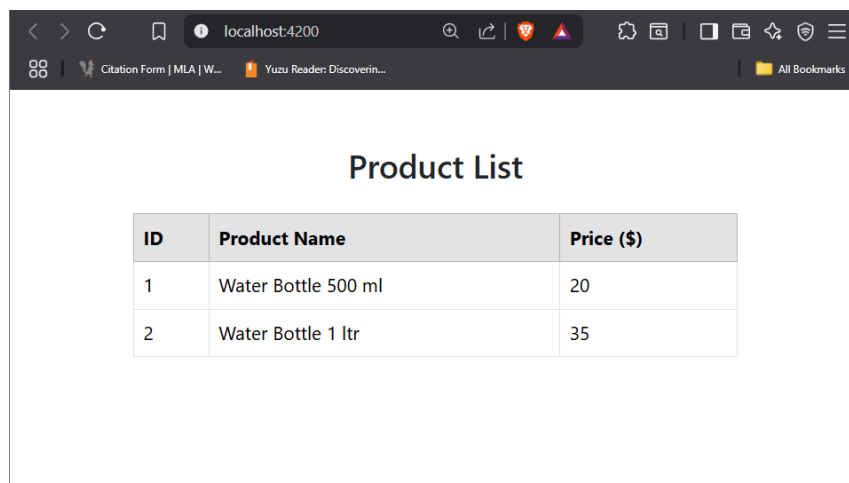
- 1) Make sure .NET Core backend API is open and running in Visual Studio and open the swagger link

(In our case): <https://localhost:7173/swagger>

- 2) To add products, go under the **POST** tab and click “**Try it Out**”, add data in with the format given below, and click **execute**. (Remove the “id” field if present)



- 3) Open the Angular app and refresh, you should see the page with the new product. You now have a completed web application with a working backend show on the frontend using bootstrap.



ID	Product Name	Price (\$)
1	Water Bottle 500 ml	20
2	Water Bottle 1 ltr	35

Troubleshooting:

Some common problems and solutions are listed below:

Bootstrap does not apply	make sure it installed properly and is added in the angular.json styles array and is present in the package.json "dependencies".
Angular shows empty screen	Make sure the app component has the <app-product-list></app-product-list> tag.
CORS error	Make sure CORS is enabled and try running again.
Data is empty	Make sure that it is added using POST in Swagger.

Conclusion

You have now completed setting up a full stack web development environment with Angular 2+, .NET Core Web API, SQL Server Developer and Bootstrap. You should now also be able to run it and add data to it with it fetched from the backed and updated in a bootstrap table. It is also ready to be extended with more features or for a different project to be built in the environment.