Udemy Course - Dotnet With Angular

source code : https://github.com/TryCatchLearn/skinet-2024

.NET is the free, open-source, cross-platform framework for building modern apps and powerful cloud services. source

Extension for VC

- C# Dev Kit (Microsfot)
- Nuget Gallery (pcislo)
- SQl Server (Microsoft)
- Material Icons Theme (Philipp Kief)

Creating a .net project

$ dotnet --info
$ dotnet -h
$ dotnet new list

See: sln, webapi,

$ dotnet new sln
Create a file with the same name of the current folder

$ dotnet new webapi -o API -controllers
$ dotnet new classlib -o Core
$ dotnet new classlib -o Infrastructure

$ dotnet sln add API
$ dotnet sln add Core
$ dotnet sln add Infrastructure

Solution:
API --> Infrastructure --> Core

$ cd API/
$ dotnet add reference ../Infrastructure

$ cd Infrastructure

$ cd ..
$ dotnet restore
$ dotnet build

$ cd API
$ dotnet run

launch a server http://localhost:5167/WeatherForecast

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase {
    ...
}
```

launchSettings.json

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "http://localhost:5167",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "https://localhost:7013;http://localhost:5167",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

In this development missing swagger

appsettings.Development.json, change Microsoft.AspNetCode from warning to information

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
```

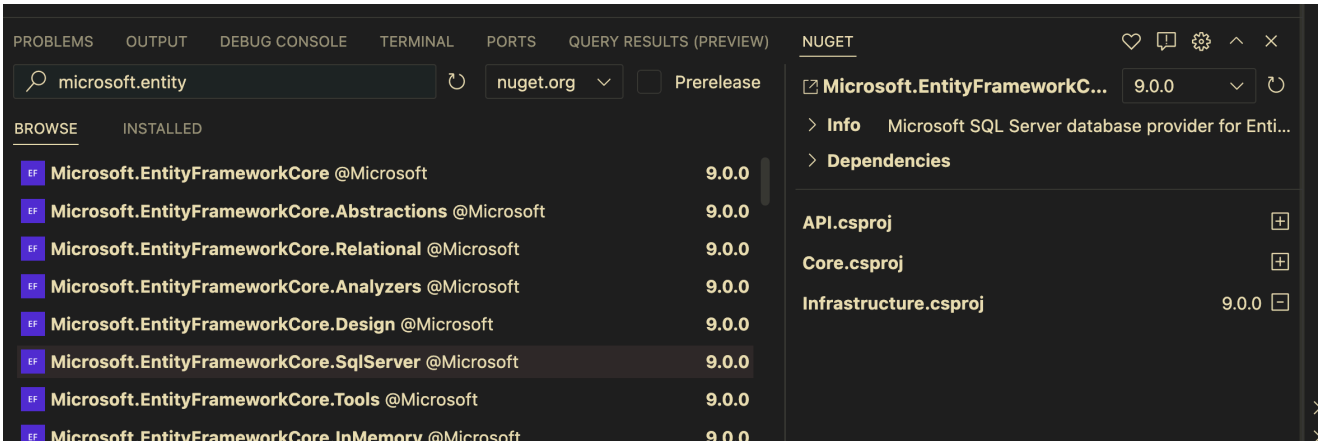
```
        "Microsoft.AspNetCore": "Information"
      }
    }
  }
```

$ dotnet run watch

using nutget

SqlServer (infraestructure)

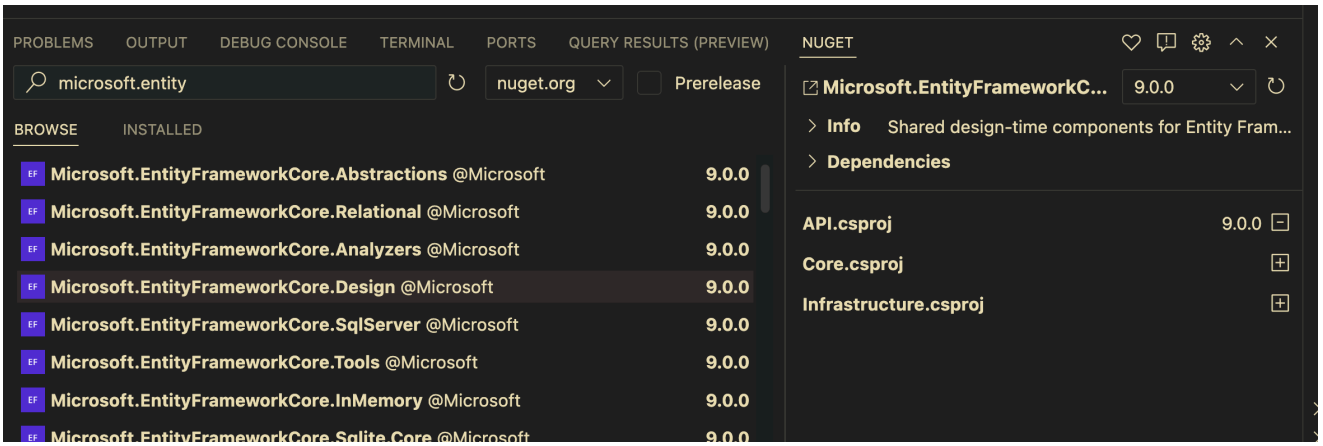
"Entity Framework Core (EF Core) is a modern object-database mapper that lets you build a clean, portable, and high-level data access layer with .NET (C#) across a variety of databases, including SQL Server (on-premises and Azure), SQLite, MySQL, PostgreSQL, Oracle, and Azure Cosmos DB. It supports LINQ queries, change tracking, updates, and schema migrations."



For migrations, need design (API)

"The Entity Framework Core tools help with design-time development tasks. They're primarily used to manage Migrations and to scaffold a `DbContext` and entity types by reverse engineering the schema of a database."

"The `Microsoft.EntityFrameworkCore.Design` package is required for either command-line or Package Manager Console-based tooling, and is a dependency of dotnet-ef and Microsoft.EntityFrameworkCore.Tools."



Note: A DbContext instance represents a session with the database and can be used to query and save instances of its entities. DbContext is a combination of the Unit of Work and Repository patterns.

Read C# Class with primary constructor

https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/primary-constructors

to use docker with microsoft sql server under mac, we will use a docker composer

```
docker compose up -d
```

We need to install a tool called dotnet-ef

```
$ dotnet tool install --global dotnet-ef --version 9.0.0

...
$ dotnet ef
$ dotnet ef migrations add InitialCreate -s API -p Infrastructure

# to remove last migrations
$ dotnet ef migrations remove -s API -p Infrastructure

# to update the database ( in this we use docker with microsoft sql and
there is a connection string)

$ dotnet ef database update
No project was found. Change the current working directory or use the --
project option.

$ dotnet ef database update -s API -p Infrastructure
```

execute api

```
$ cd API
```

```
$ dotnet watch
```

Instead of using postman for testing, uses rest client (vc extension)

Add Repository

```
$ dotnet new list
$ dotnet new gitignore
```

Architecture

API

- Controllers
- Startup
- Middleware
  Infrastructure
- Repository
- DbContext
- Services
  Core
- Entities
- Interfaces

Repository Pattern

Goals

- Decouple business code from data access
- Separation of concerns (avoid direct use data methods)
- Minimise duplicate query logic
- Testability

Why use it

- Avoid messy controllers
- Simplified testing
- Increased abstraction
- Maintainability
- Reduced duplicate code

Downsides

- Abstraction of an abstraction
- Optimisation challenges

The sequence is the following:
Controller
IRepository
GetProducts()
GetProduct(int id)
Note: We inject the instance of a repository in the controller

Repository
_context.Products.ToList()
Note: We inject an instance of a DbContext into the repository

DbContext
Select * from Products

Migrations

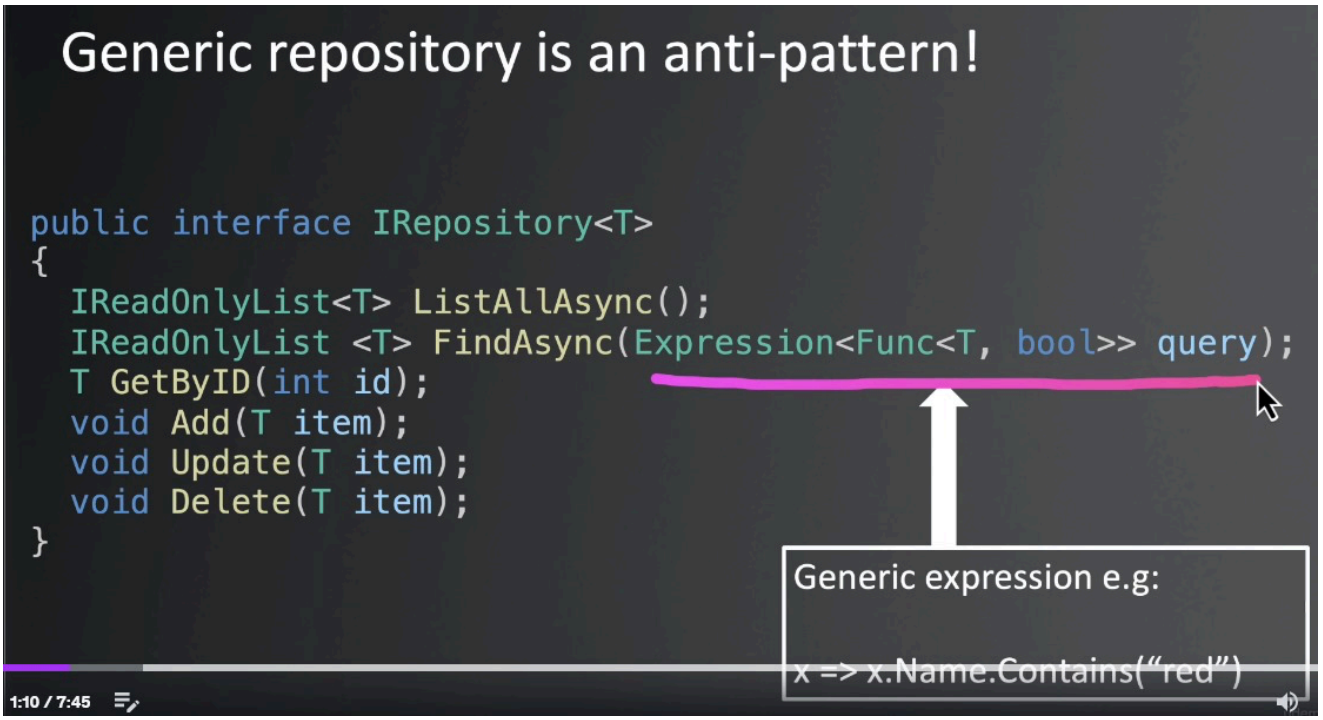
read https://medium.com/@zaynt.dev/understanding-serviceprovider-createscope-in-net-a-deep-dive-a0d4549cc4b1

to remove a database

```
$ dotnet ef database drop -p Infrastructure -s API
```

Generic Repository

Generic repository is an anti-pattern, it is a type that comprises of a set of a generic methods for performing CRUD operations. However, It's just another anti pattern and is used frequently with entity framework to abstract calls to the data access layer.



The specification pattern to the rescue!

- Describes a query in an object
- Returns an IQueryable<T>
- Generic List method takes specification as parameter
- Specification can have meaningful name
    - OrdersWithItemsAndSortingSpecification

Note: Need more information about how to use correctly the specification pattern

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-9.0
DataAnnotations

Installing Node.js with NVM

```
$ npm install -g @angular/cli@18
$ ng version
```

Optional, configure angular cli to use BUN
https://blog.khophi.co/switch-to-using-bun-in-angular/

How to install Bun

Install local certificate for angular to use https

https://github.com/FiloSottile/mkcert

```
$ mkcert -install
$ mkdir ssl
$ cd ssl
$ mkcert localhost
```

and add those files into angular.json

angular.json 1

```
  "projects": {
    "client": {
      "architect": {
        "build": {
          "configurations": {
          },
          "defaultConfiguration": "production"
        },
        "serve": {
          "builder": "@angular-devkit/build-angular:dev-server",
          "options": {
            "ssl": true,
            "sslCert": "ssl/localhost.pem",
            "sslKey": "ssl/localhost-key.pem"
          },
          "configurations": {
            "production": {
```

To download Angular Material

```
$ ng add @angular/material
```

see No for Angular Material typography styles

install Tailwind for Angular

```
$ npm install -D tailwindcss postcss autoprefixer
$ npx tailwindcss init
```

VC Extensions

- Angular Language Service (Angular.io)
- Tailwind CSS IntelliSense (tailwindcss.com
- Auto Rename Tag (Jun Han) - perhaps it's not neccessary

Adding Redis via docker

```
$ docker compose down
```

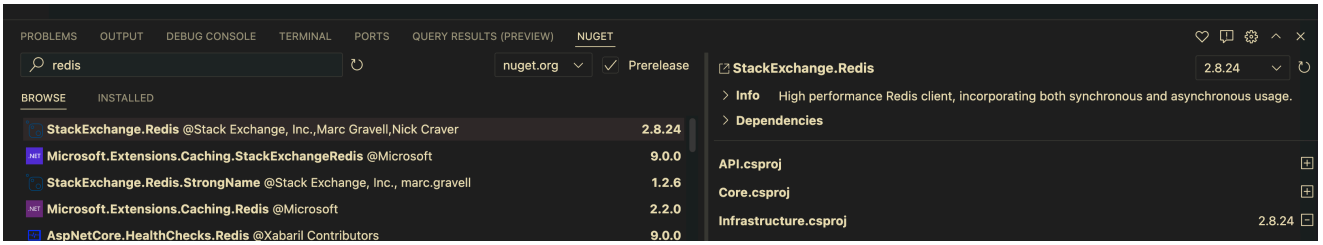
```
services:
  sql:
    image: mcr.microsoft.com/azure-sql-edge
    environment:
      ACCEPT_EULA: "1"
      MSSQL_SA_PASSWORD: "Password@1"
    ports:
      - "1433:1433"
    volumes:
      - sql-data:/var/opt/mssql
  redis:
    image: redis:latest
    ports:
      - "6379:6379"
    volumes:
      - redis-data:/data

volumes:
  redis-data:
  sql-data:
```

```
$ docker compose up -d
```

to remove the volumes too, use the following command:

```
$ docker compose down -v
```

install redis package via nuget in infrastructure




```
builder.Services.AddSingleton<IConnectionMultiplexer>(config =>
{
    var connString = builder.Configuration.GetConnectionString("Redis")
        ?? throw new Exception("Cannot get redis connection string");

    var configuration = ConfigurationOptions.Parse(connString, true);
```

```
    return ConnectionMultiplexer.Connect(configuration);
});
```

install a visual code extension called Redis Dunn

docker-compose.yaml

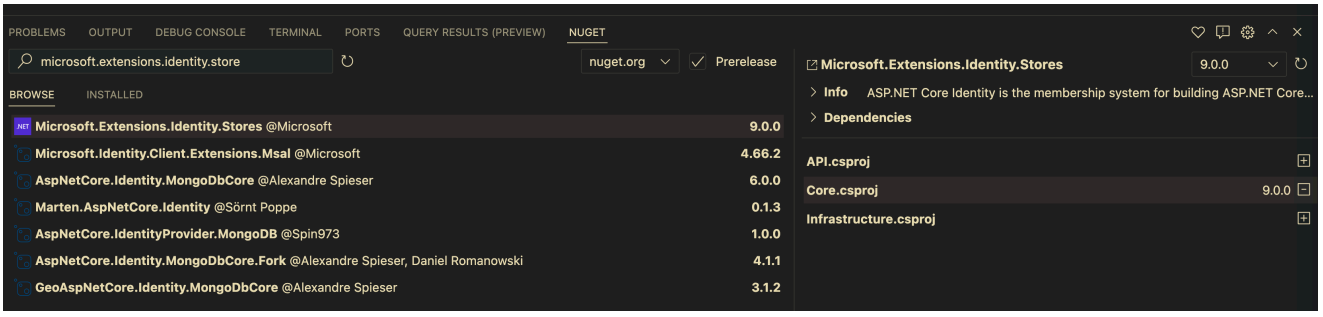
```yaml
services:
  sql:
    image: mcr.microsoft.com/azure-sql-edge
    environment:
      ACCEPT_EULA: "1"
      MSSQL_SA_PASSWORD: "Password@1"
    ports:
      - "1433:1433"
    volumes:
      - sql-data:/var/opt/mssql
  redis:
    image: redis:latest
    ports:
      - "6379:6379"
    volumes:
      - redis-data:/data

volumes:
  redis-data:
  sql-data:
```

read [How to use Identity to secure a Web API - aspnetcore 9]

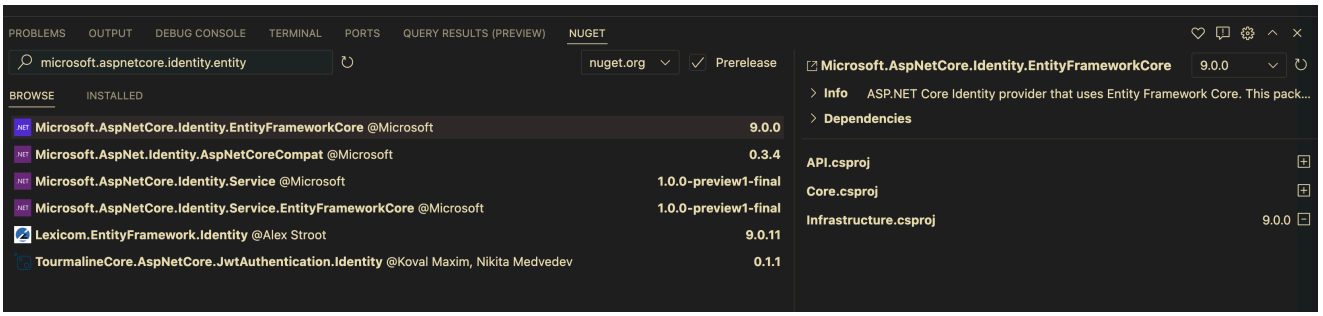
see under "The MapIdentityAp<TUser> endpoints"

installing nuget packages:

- microsoft.extensions.identity.store



- microsoft.aspnetcore.identity.entityFrameworkcore



in StoreContext.cs

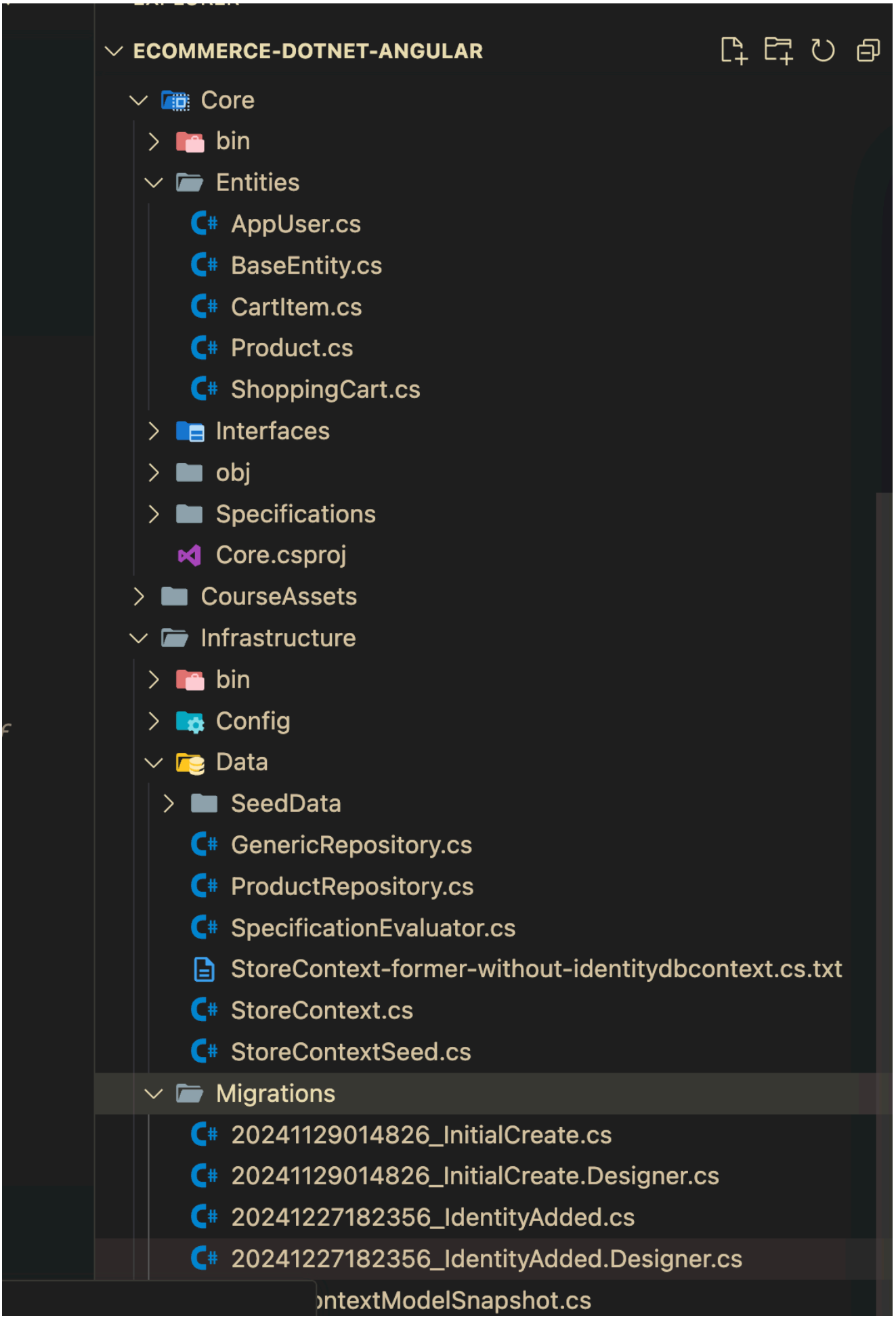
```
using Core.Entities;
using Infrastructure.Config;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace Infrastructure.Data;
public class StoreContext(DbContextOptions options) :
IdentityDbContext<AppUser>(options)
{
  public DbSet<Product> Products { get; set; }

  protected override void OnModelCreating(ModelBuilder modelBuilder)
  {
    base.OnModelCreating(modelBuilder);


modelBuilder.ApplyConfigurationsFromAssembly(typeof(ProductConfiguration).As
sembly);
  }

}
```

in program.cs

```
// identity - aspnet authorization

builder.Services.AddAuthorization();
builder.Services.AddIdentityApiEndpoints<AppUser>()
    .AddEntityFrameworkStores<StoreContext>();

//...

// add microsoft identity
app.MapIdentityApi<AppUser>();
```

We need to update the schemas

```
$ dotnet ef migrations add IdentityAdded -s API -p Infrastructure
```



When you execute the dotnet watch in API, you will see the following warning

```
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key {ee185875-e50f-4c38-9a59-
148586f48a38} may be persisted to storage in unencrypted form.
```

We added Adressess for AppUser and StoreContext, we need to update the database

```
$ donet ef migrations add AddressAdded -s API -p Infrastructure
```

Add deliverymethod table

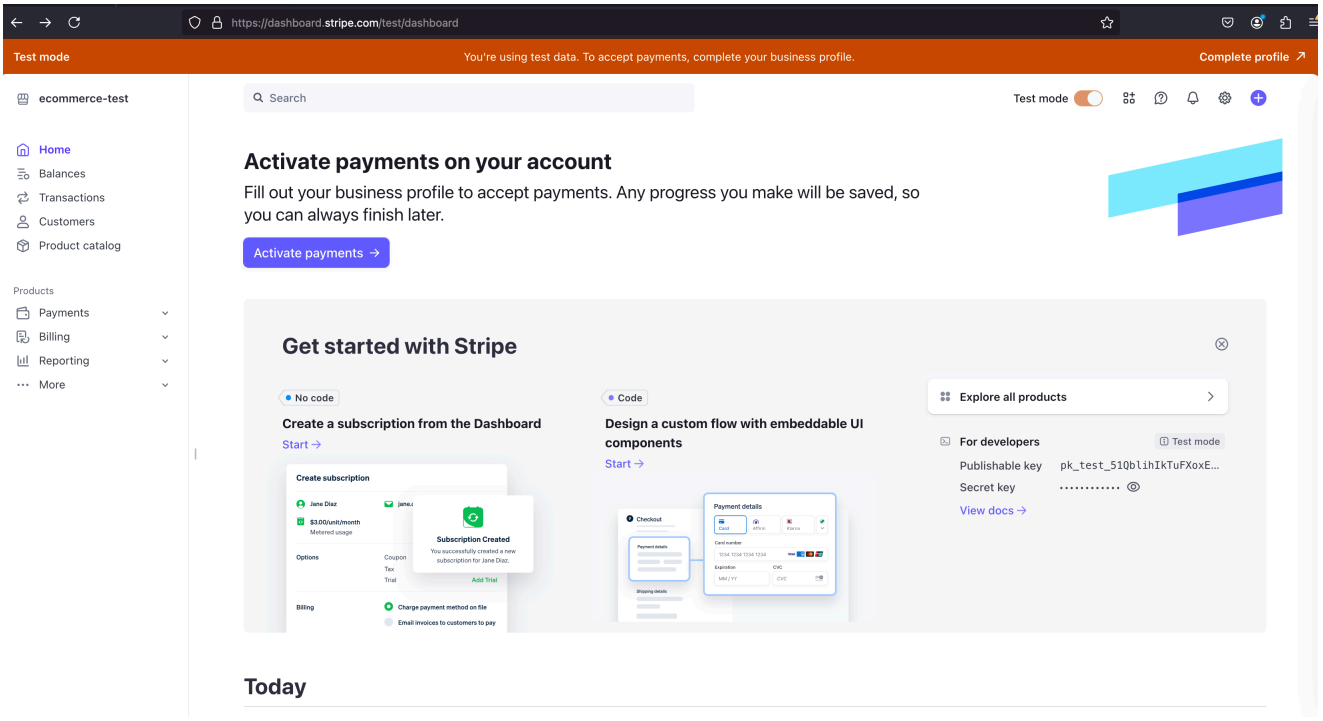
```
$ dotnet ef migrations add DeliveryMethodAdded -p Infrastructure -s API
```

to update your donet tools

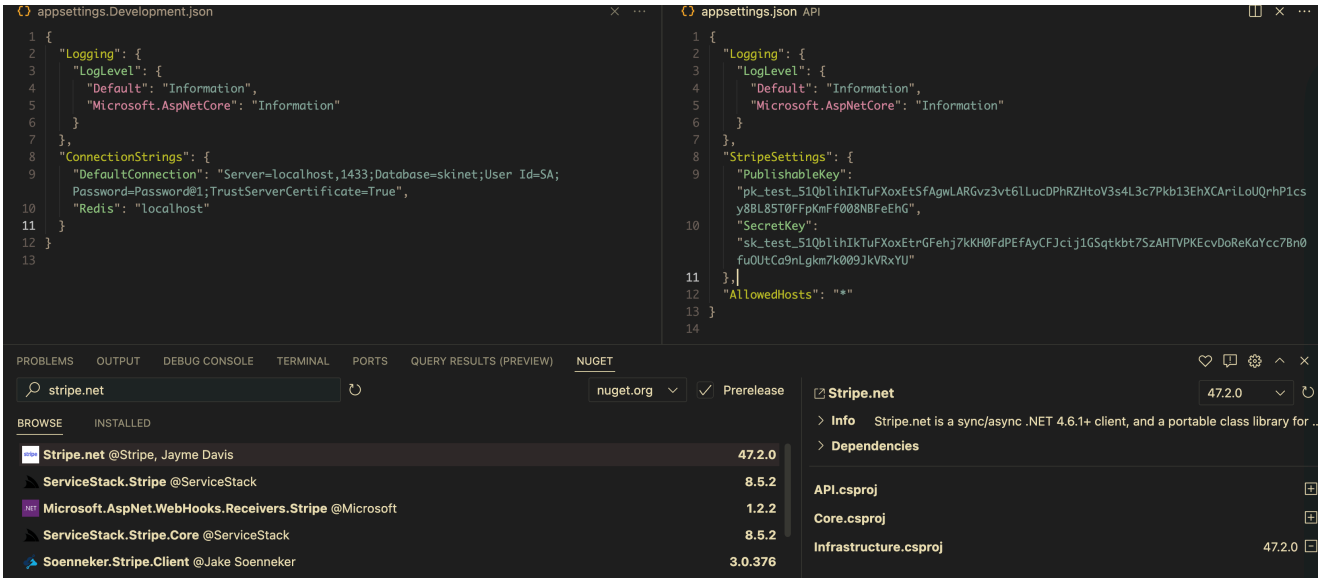
```
$ dotnet tool update dotnet-ef -g
```

create an account on stripe

pk_test_51Qbl3dfEEihIkTuFXoxEtSfAgwLAEEDFE343RGvz3vt6lLucDPhRZHtoV3s4L3c7Pkb13EhXCAriLoUQrhP1csy8BL85T0FFpKmFf008NBFeEh34G
sk_test_51QbfdEE33lihIkTuFXoxEtrGFehj7kKH0FdPEfdfwere53FEEAyCFJcij1GSqtkbt7SzAHTVPKEcvDoReKaYcc7Bn0fuOUtCa9nLgkm7k009JkVR34xYU



using nuget, add stripe.net



The `builder.OwnsOne` method in .NET is part of Entity Framework Core, which is Microsoft's object-relational mapper (ORM) for .NET. This method is used to configure a one-to-one relationship between two entities where one entity "owns" the other.

```
public class Order
{
    public int OrderId { get; set; }
    public ShippingAddress ShippingAddress { get; set; }
}

public class ShippingAddress
{
    public string Street { get; set; }
    public string City { get; set; }
    public string Country { get; set; }
}

public class AppDbContext : DbContext
{
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Order>()
            .OwnsOne(o => o.ShippingAddress);
    }

    public DbSet<Order> Orders { get; set; }
}
```

In this example, the `Order` entity owns a `ShippingAddress` entity. When `OwnsOne` is used, the `ShippingAddress` is treated as a part of the `Order` entity and is stored in the same table in the database.

Key Points to Remember:

- The owned entity (ShippingAddress) doesn't have an identity of its own.
- It shares the same primary key as the owning entity (Order).
- The owned entity's lifecycle is tied to the owning entity. If the owning entity is deleted, the owned entity is also deleted.

This feature is particularly useful when you have complex types or value objects that are inherently part of another entity and do not need their own database table.

```
builder.OwnsOne(x => x.ShippingAddress, o => o.WithOwner());
```

The `WithOwner()` method in the code `builder.OwnsOne(x => x.ShippingAddress, o => o.WithOwner())` is used to configure the relationship between the owned entity (`ShippingAddress`) and the owning entity (`Order`). Specifically, `WithOwner()` indicates that the `ShippingAddress` entity is dependent on the `Order` entity and shares the same ownership.

To break it down:

- `builder.OwnsOne(x => x.ShippingAddress, o => o.WithOwner())` : This specifies that the `Order` entity owns the `ShippingAddress` entity.
- `WithOwner()` : This method is used to configure the navigation property in the owned entity that points back to the owner. In this case, it tells Entity Framework that the `ShippingAddress` belongs to an `Order` .

In simpler terms, `WithOwner()` reinforces the relationship and dependency between the `ShippingAddress` and `Order` , ensuring that the `ShippingAddress` is always associated with a specific `Order` and cannot exist independently.

adding OrderAggregate

```
$ dotnet ef migrations add OrderAggregateAdded -p Infrastructure -s API
```

Unit of Work

The "Unit of Work" pattern is a design pattern used in software development to manage changes to data within a single transaction. The main idea behind this pattern is to ensure that a series of operations either all succeed or all fail, maintaining data consistency and integrity.

Here's a breakdown of the Unit of Work pattern:

Key Concepts:

1. **Transactional Operations**: The Unit of Work keeps track of all the operations (inserts, updates, deletes) on data during a business transaction.
2. **Commit and Rollback**: At the end of the transaction, you can either commit all the changes, making them permanent, or rollback, undoing all the changes if something goes wrong.
3. **Consistency and Integrity**: This pattern helps maintain data consistency and integrity by ensuring that all changes within a transaction are applied together.

How It Works:

1. **Start a Unit of Work**: When a business operation begins, a new Unit of Work is created.
2. **Perform Operations**: All changes to the data (inserts, updates, deletes) are tracked by the Unit of Work.
3. **Commit or Rollback**: At the end of the operation, the Unit of Work decides whether to commit all changes to the database or rollback if an error occurs.

Example in .NET (Entity Framework Core):

Here's a simple example to illustrate the Unit of Work pattern using Entity Framework Core:

```
public interface IUnitOfWork
{
    Task SaveChangesAsync();
    void Rollback();
}

public class UnitOfWork : IUnitOfWork, IDisposable
{
    private readonly AppDbContext _context;

    public UnitOfWork(AppDbContext context)
    {
        _context = context;
    }

    public async Task SaveChangesAsync()
    {
        await _context.SaveChangesAsync();
    }

    public void Rollback()
    {
        // In EF Core, rollback is typically handled through transaction
management
        // Here, you would implement logic to revert changes if needed
    }
```

```
    public void Dispose()
    {
        _context.Dispose();
    }
}
```

Benefits:

- **Transaction Management**: Simplifies transaction management by grouping related operations.
- **Data Integrity**: Ensures data integrity by committing all changes in a single transaction.
- **Decoupling**: Decouples business logic from data access logic, making the code easier to maintain and test.

In summary, the Unit of Work pattern is a powerful tool for managing data changes within a transaction, ensuring consistency, and simplifying transaction management.

```
$ dotnet ef migrations add PaymentSummaryCorrection -p Infrastructure -s API
```

Stripe has cli tools you can install in your terminal to interact with your localhost.

```
$ brew install stripe/stripe-cli/stripe

$ stripe login

$ stripe listen --forward-to https://localhost:5001/api/payments/webhook -e
payment_intent.succeeded
```

Adding SignalIR

SignalR is a free, open-source library that allows developers to add real-time communication to their ASP.NET applications. It enables servers to push content to clients instantly, creating a two-way communication channel.

What does SignalR do?

- **Simplifies real-time functionality**: SignalR provides an API that handles the complexity of managing connections and implementing real-time communication
- **Supports multiple transport protocols**: SignalR supports HTTP, WebSockets, Server-Sent Events, and Long Polling

- **Enables automatic reconnection**: SignalR automatically reconnects clients when they drop off
- **Scales horizontally**: SignalR can scale to thousands of clients using integrated and third-party providers

What can SignalR be used for?

- **Chat applications**

  SignalR can be used to create chat applications that allow users to communicate in real-time
- **Collaboration tools**

  SignalR can be used to create collaboration tools that allow users to work together in real-time
- **IoT device control**

  SignalR can be used to control IoT devices in real-time
- **Real-time dashboards**

  SignalR can be used to create real-time dashboards that provide instant updates

SignalR is supported on Linux, Windows, and macOS.

SignalR for client

```
$ npm install @microsoft/signalr
```

Publishing your project

preparing the client app for publishing

modify your angular.json to build in the folder /API/wwwroot

the original is "outputPath": "dist/client"
if we change to outputPath: "../API/base/browser" (where the browser is added by angular)
to avoid this, we use the following form.

```
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:application",
          "options": {
            "outputPath": {
              "base": "../API/wwwroot",
```

```
              "browser": ""
            },
```

see this original angular.json

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "client": {
      "projectType": "application",
      "schematics": {
        "@schematics/angular:component": {
          "style": "scss",
          "standalone": true,
          "changeDetection": "OnPush",
          "skipTests": true
        }
      },
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:application",
          "options": {
            "outputPath": {
              "base": "../API/wwwroot",
              "browser": ""
            },
            "index": "src/index.html",
            "browser": "src/main.ts",
            "polyfills": ["zone.js"],
            "tsConfig": "tsconfig.app.json",
            "inlineStyleLanguage": "scss",
            "assets": [
              {
                "glob": "**/*",
                "input": "public"
              }
            ],
            "styles": [
              "@angular/material/prebuilt-themes/azure-blue.css",
              "src/styles.scss"
            ],
            "scripts": []
          },
          "configurations": {
```

```
            "production": {
              "budgets": [
                {
                  "type": "initial",
                  "maximumWarning": "1MB",
                  "maximumError": "2MB"
                },
                {
                  "type": "anyComponentStyle",
                  "maximumWarning": "2kB",
                  "maximumError": "4kB"
                }
              ],
              "outputHashing": "all"
            },
            "development": {
              "optimization": false,
              "extractLicenses": false,
              "sourceMap": true,
              "fileReplacements": [
                {
                  "replace": "src/environments/environment.ts",
                  "with": "src/environments/environment.development.ts"
                }
              ]
            }
          },
          "defaultConfiguration": "production"
        },
        "serve": {
          "builder": "@angular-devkit/build-angular:dev-server",
          "options": {
            "ssl": true,
            "sslCert": "ssl/localhost.pem",
            "sslKey": "ssl/localhost-key.pem"
          },
          "configurations": {
            "production": {
              "buildTarget": "client:build:production"
            },
            "development": {
              "buildTarget": "client:build:development"
            }
          },
          "defaultConfiguration": "development"
        },
        "extract-i18n": {
          "builder": "@angular-devkit/build-angular:extract-i18n"
        },
        "test": {
```

```
          "builder": "@angular-devkit/build-angular:karma",
          "options": {
            "polyfills": ["zone.js", "zone.js/testing"],
            "tsConfig": "tsconfig.spec.json",
            "inlineStyleLanguage": "scss",
            "assets": [
              {
                "glob": "**/*",
                "input": "public"
              }
            ],
            "styles": [
              "@angular/material/prebuilt-themes/azure-blue.css",
              "src/styles.scss"
            ],
            "scripts": []
          }
        }
      }
    }
  }
}
```

the project added a delay in dev environment, we change that

notice that we use identity rxjs

loading.interceptor.ts

```
import { HttpInterceptorFn } from '@angular/common/http';
import { inject } from '@angular/core';
import { delay, finalize, identity } from 'rxjs';
import { BusyService } from '../services/busy.service';
import { environment } from '../../../environments/environment';

export const loadingInterceptor: HttpInterceptorFn = (req, next) => {
  const busyService = inject(BusyService);

  busyService.busy();

  return next(req).pipe(
    (environment.production ? identity : delay(500)),
    finalize(() => busyService.idle())
  )
};
```

remember that in angular.json there are some configuration to the size of the bundle

```
          "configurations": {
            "production": {
              "budgets": [
                {
                  "type": "initial",
                  "maximumWarning": "1MB",
                  "maximumError": "2MB"
                },
                {
                  "type": "anyComponentStyle",
                  "maximumWarning": "2kB",
                  "maximumError": "4kB"
                }
              ],
              "outputHashing": "all"
            },
```

```
~/code/udemy/ecommerce-dotnet-angular
❯ cd client
❯ ng build
Initial chunk files   | Names           |  Raw size | Estimated transfer size
chunk-ZMGNDDHD.js     | -               | 191.52 kB |                42.81 kB
chunk-D3HH3I3Z.js     | -               | 188.44 kB |                54.97 kB
chunk-HUJZUZTO.js     | -               | 169.33 kB |                29.05 kB
main-MZW6ORRL.js      | main            | 147.36 kB |                24.86 kB
styles-6EURB4QU.css   | styles          | 135.46 kB |                11.84 kB
chunk-JT6ZLRXC.js     | -               |  76.08 kB |                17.24 kB
chunk-HWZZGXA3.js     | -               |  57.92 kB |                13.21 kB
polyfills-6EAL64PA.js | polyfills       |  34.23 kB |                11.13 kB
chunk-NQYQCVGC.js     | -               |  20.80 kB |                 5.67 kB
chunk-AANWVVNP.js     | -               |  12.51 kB |                 3.85 kB
chunk-LQ4XQAP4.js     | -               |  11.07 kB |                 3.69 kB
chunk-JV74U5KP.js     | -               |   4.99 kB |                 1.23 kB
chunk-JGYM447R.js     | -               |   4.27 kB |                 1.06 kB
chunk-6D2NRMAR.js     | -               | 331 bytes |              331 bytes

                      | Initial total   |   1.05 MB |              220.95 kB

Lazy chunk files      | Names           |  Raw size | Estimated transfer size
chunk-2TVC5JH3.js     | routes          |  93.68 kB |                17.57 kB
chunk-6YZ5ETOM.js     | admin-component |  84.69 kB |                17.59 kB
chunk-XHKVSB7S.js     | browser         |  63.61 kB |                16.87 kB
chunk-FCPM7THH.js     | routes          |   5.93 kB |                 2.04 kB
chunk-BK6JAUUT.js     | routes          |   4.98 kB |                 1.59 kB
chunk-LNAAKBMU.js     | -               |   1.56 kB |              553 bytes
chunk-V5JIL5KR.js     | -               | 687 bytes |              687 bytes

Output location: /Users/dthm/Code/udemy/ecommerce-dotnet-angular/API/wwwroot

Application bundle generation complete. [3.879 seconds]

▲ WARNING bundle initial exceeded maximum budget. Budget 1.05 MB was not met by 5.73
 kB with a total of 1.05 MB.
```

those files will be hosted under the API project, and we need to add a controller to server them and modify the program.cs, adding a middleware to allow serving static files

Program.cs

```
77 app.UseAuthentication();
78 app.UseAuthorization();
79 // End SignalR
80
81
82 // Adding a middleware to allow server static files ( angular client )
83 app.UseDefaultFiles();
84 app.UseStaticFiles();
85
86
87
88 app.MapControllers();
89 // add microsoft identity
90 // app.MapIdentityApi<AppUser>(); // /login
91 // Note: SignalR doesn't use app.MapGroup("api").MapIdentityApi<AppUser>(); to
   authenticate,
92 app.MapGroup("api").MapIdentityApi<AppUser>(); // api/login
93
94 // Add SignalR Hub
95 app.MapHub<NotificationHub>("/hub/notification");
96
97
98 // for Angular client, see FallbackController
99 app.MapFallbackToController("Index", "Fallback");
100
101
```

FallbackController.cs API/Controllers

```
1  using Microsoft.AspNetCore.Mvc;
2
3  namespace API.Controllers;
4
   0 references
5  public class FallbackController : Controller
6  {
       0 references
7      public IActionResult Index()
8      {
9          return PhysicalFile(
10             Path.Combine(
11                 Directory.GetCurrentDirectory(), "wwwroot", "index.html"),
12             "text/HTML");
13     }
14 }
```

We need to include the route of the data json

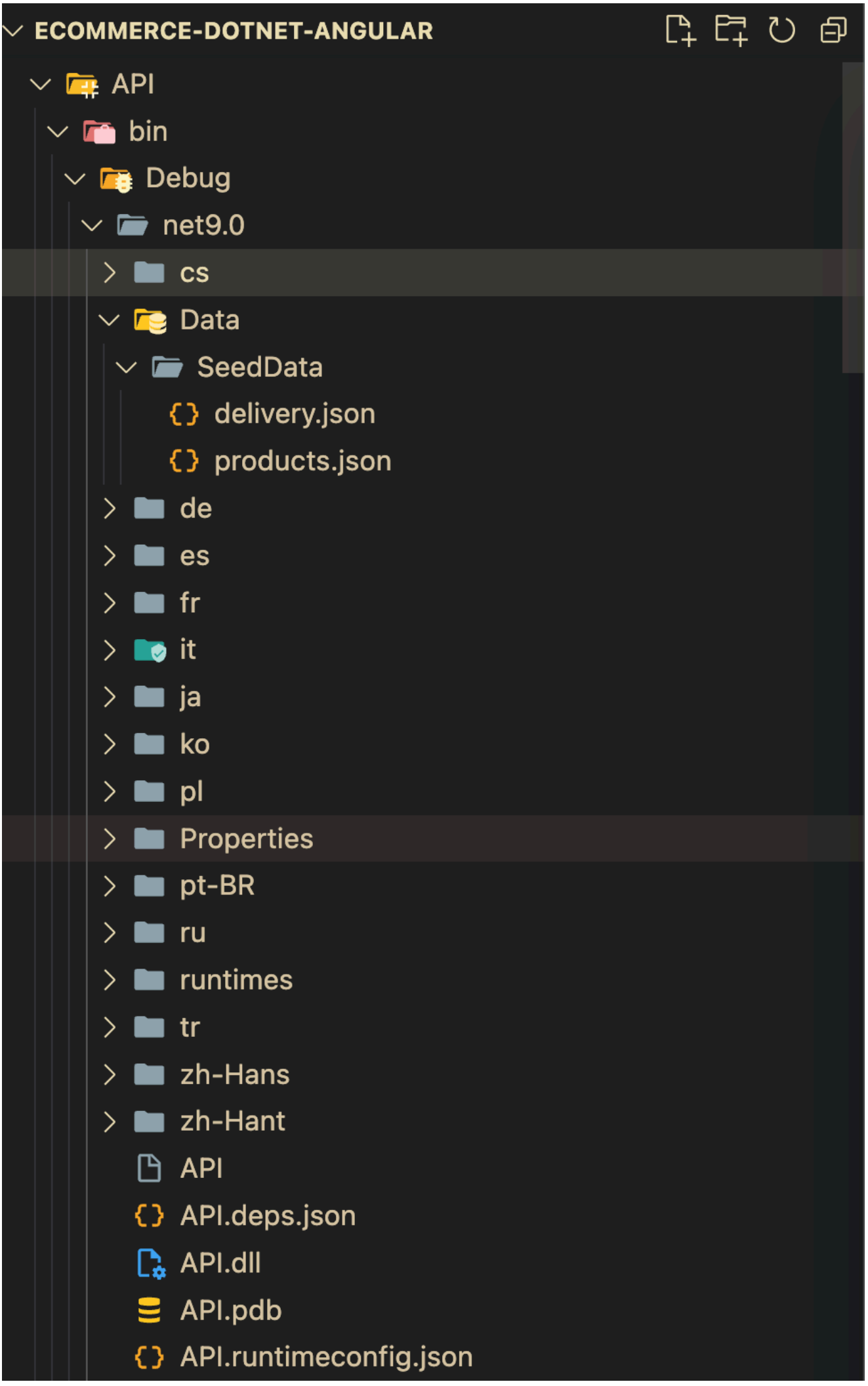
```
var path = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);



if (!context.Products.Any())
{
  // var productsData = await File.ReadAllTextAsync("../Infrastructure/Data/SeedData/products.json");
  var productsData = await File.ReadAllTextAsync(path + @"/Data/SeedData/products.json");
  var products = JsonSerializer.Deserialize<List<Product>>(productsData);

  if (products == null) return;

  context.Products.AddRange(products);

  await context.SaveChangesAsync();
}

if (!context.DeliveryMethods.Any())
{
  var dmData = await File.ReadAllTextAsync(path + "/Data/SeedData/delivery.json");
  var methods = JsonSerializer.Deserialize<List<DeliveryMethod>>(dmData);
```

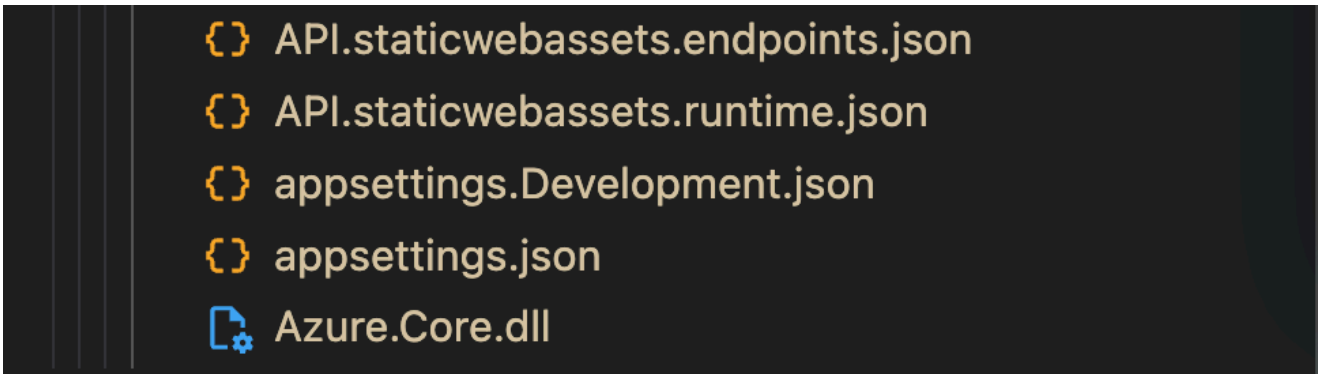
```
$ dotnet ef database drop -p Infrastructure -s API
```

```
❯ dotnet ef database drop -p Infrastructure -s API
MSBUILD : error MSB1009: Project file does not exist.
Switch: /Users/dthm/Code/udemy/ecommerce-dotnet-angular/API/Infrastructure
Unable to retrieve project metadata. Ensure it's an SDK-style project. If you're using
a custom BaseIntermediateOutputPath or MSBuildProjectExtensionsPath values, Use the --m
sbuildprojectextensionspath option.
❯ cd ..
❯ dotnet ef database drop -p Infrastructure -s API
Build started...
Build succeeded.
Are you sure you want to drop the database 'skinet' on server 'localhost,1433'? (y/N)
y
Dropping database 'skinet' on server 'localhost,1433'.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (8ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (23ms) [Parameters=[], CommandType='Text', CommandTimeout='60'
]
      IF SERVERPROPERTY('EngineEdition') <> 5
      BEGIN
          ALTER DATABASE [skinet] SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
      END;
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (54ms) [Parameters=[], CommandType='Text', CommandTimeout='60'
]
      DROP DATABASE [skinet];
Successfully dropped database 'skinet'.

~/Code/udemy/ecommerce-dotnet-angular  8s 3.12.2 system
❯
```

```
$ cd API
$ dotnet build
```


Successfully dropped database 'skinet'.
❯ cd API
❯ dotnet build
Restore complete (0.2s)
Core succeeded (0.1s) → /Users/dthm/Code/udemy/ecommerce-dotnet-angular/Core/bin/Debug/net9.0/Core.dll
Infrastructure succeeded (0.1s) → /Users/dthm/Code/udemy/ecommerce-dotnet-angular/Infrastructure/bin/Debug/net9.0/Infrastructure.dll
API succeeded (0.4s) → bin/Debug/net9.0/API.dll
Build succeeded in 1.0s
~/Code/udemy/ecommerce-dotnet-angular/API
3.12.2
system
❯


ECOMMERCE-DOTNET-ANGULAR
API
bin
Debug
net9.0
cs
Data
SeedData
delivery.json
products.json
de
es
fr
it
ja
ko
pl
Properties
pt-BR
ru
runtimes
tr
zh-Hans
zh-Hant
API
API.deps.json
API.dll
API.pdb
API.runtimeconfig.json


API.staticwebassets.endpoints.json
API.staticwebassets.runtime.json
appsettings.Development.json
appsettings.json
Azure.Core.dll

```
$ dotnet run
```

```
 dotnet run
sing launch settings from /Users/dthm/Code/udemy/ecommerce-dotnet-angular/API/Propert
s/launchSettings.json...
uilding...
nfo: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[62]
     User profile is available. Using '/Users/dthm/.aspnet/DataProtection-Keys' as ke
repository; keys will not be encrypted at rest.
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
     Executed DbCommand (368ms) [Parameters=[], CommandType='Text', CommandTimeout='6
]
     CREATE DATABASE [skinet];
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
     Executed DbCommand (106ms) [Parameters=[], CommandType='Text', CommandTimeout='6
]
     IF SERVERPROPERTY('EngineEdition') <> 5
     BEGIN
         ALTER DATABASE [skinet] SET READ_COMMITTED_SNAPSHOT ON;
     END;
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
     Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30'
     SELECT 1
nfo: Microsoft.EntityFrameworkCore.Migrations[20411]
     Acquiring an exclusive lock for migration application. See https://aka.ms/efcore
ocs-migrations-lock for more information if this takes too long.
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
     Executed DbCommand (13ms) [Parameters=[], CommandType='Text', CommandTimeout='30

     DECLARE @result int;
     EXEC @result = sp_getapplock @Resource = '__EFMigrationsLock', @LockOwner = 'Ses
on', @LockMode = 'Exclusive';
     SELECT @result
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
     Executed DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='30'
     IF OBJECT_ID(N'[__EFMigrationsHistory]') IS NULL
     BEGIN
         CREATE TABLE [__EFMigrationsHistory] (
             [MigrationId] nvarchar(150) NOT NULL,
             [ProductVersion] nvarchar(32) NOT NULL,
             CONSTRAINT [PK___EFMigrationsHistory] PRIMARY KEY ([MigrationId])
         );
     END;
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
     Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30'
     SELECT 1
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
```

```
ert
nfo: Microsoft.Hosting.Lifetime[14]
     Now listening on: http://localhost:5000
nfo: Microsoft.Hosting.Lifetime[14]
     Now listening on: https://localhost:5001
nfo: Microsoft.Hosting.Lifetime[0]
     Application started. Press Ctrl+C to shut down.
nfo: Microsoft.Hosting.Lifetime[0]
     Hosting environment: Development
nfo: Microsoft.Hosting.Lifetime[0]
     Content root path: /Users/dthm/Code/udemy/ecommerce-dotnet-angular/API
[[15;2~
```

Notice that we use a cloud redis server instead of our docker compose redis

The server is https://upstash.com/pricing

To start from scratch, you can do the following to remove any changes in your local github

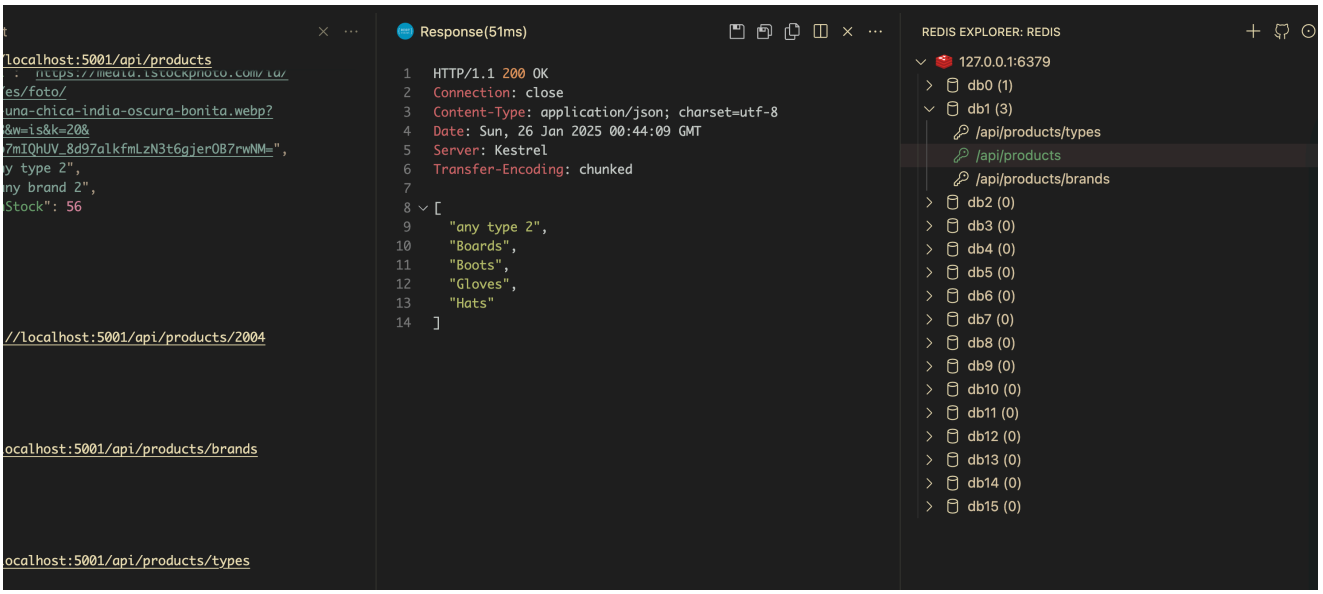
```
# Approach 1

$ git clean -df # remove untracked changes and folders
$ git checkout -- # erase changes in tracked files

# Approach 2

$ git add . # stages the changes
$ git commit "SadFaceEmoji"
$ git reset --hard HEAD # remove changes to commit
```

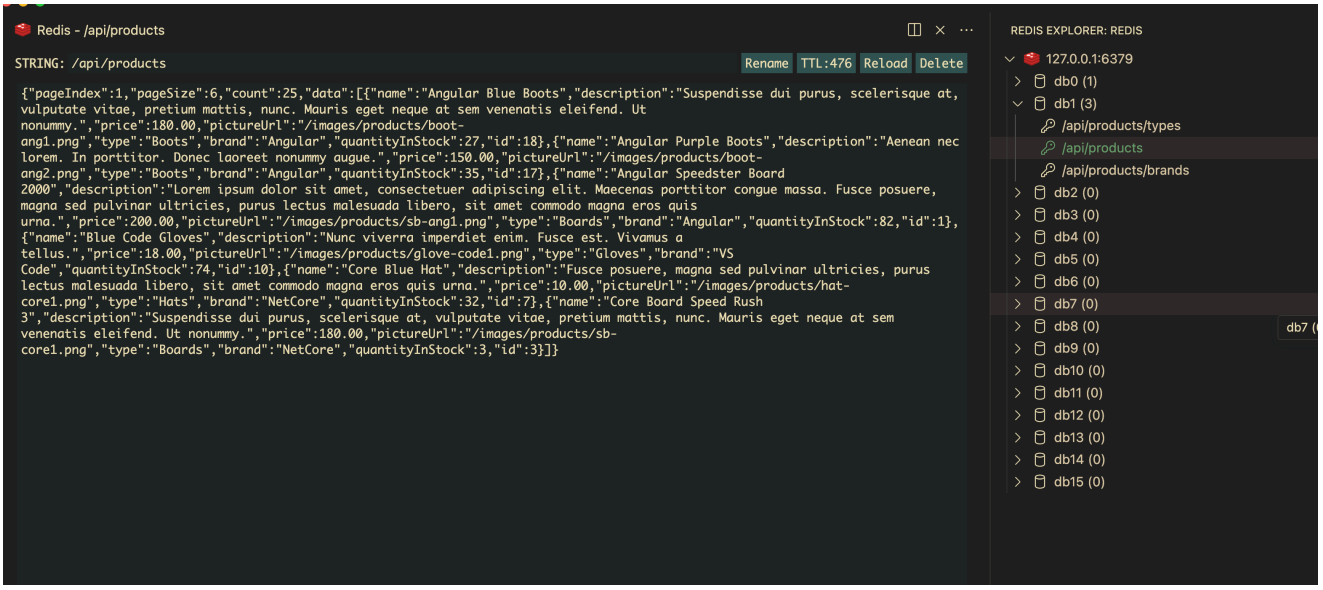
```
$ dotnet ef migrations add RolesAdded -s API -p Infrastructure

# remove migrations
# $ dotnet ef migrations remove -s API -p Infrastructure

# to update the database
# $ dotnet ef database update -s API -p Infrastructure



$ dotnet ef migrations add BasicDataRolesAdded -s API -p Infrastructure
```

Note, there is an issue with RoleConfiguration with donet 9. For that reason I down ground the donet to 8 and rebuild the dotnet.

To improve the API, we are setting up caching on the API

to do this, see IResponseCacheService and this will work with redis. See CacheAttribute.cs and productController.cs





you can extend this app adding:

Inventory
Email service (resend)
Product Review System

trycatchlearn.com/tutorials

How to run : https://www.trycatchlearn.com/tutorials/skinet-setup-tutorial/setting-up-the-infrastructure

Inventory: https://www.trycatchlearn.com/tutorials/skinet-inventory-tutorial/creating-a-re-usable-table-component

get ideas to add this product: flowbite.com