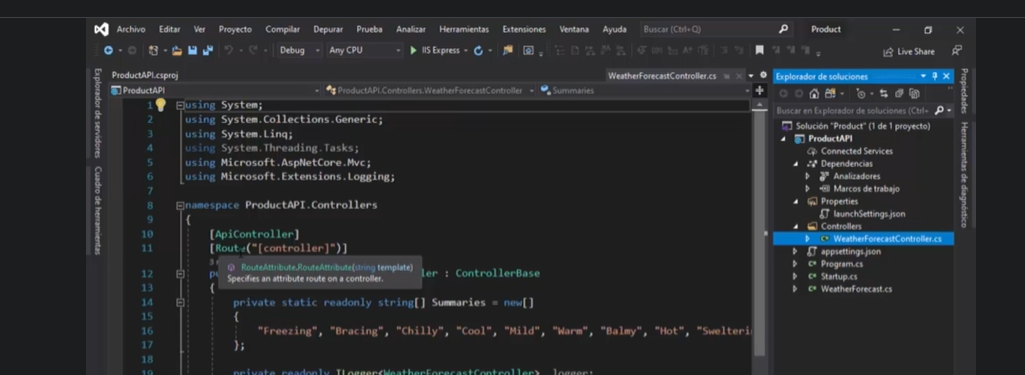
ESTRUCTURA DE PROYECTO WEB API

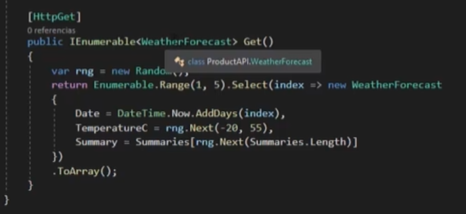
**.csproj –** Version Net.core

**Properties/launchsetting.json** - Perfiles de la aplicación (por defecto weatherforecast).

**Controllers** – Controladores, heredan de la ControllerBase, también tiene el atributo router



También se definen aquí los métodos de la API, por ejemplo, el GET:



**Appsetting.json.** Permite configurar, cadenas de conexión a bases de datos, errores.

**Program.cs.** Clase Program, crea el servidor web.

**Startup.** Permite configurar los servicios

Creamos una carpeta Models con las clases

using System.ComponentModel.DataAnnotations;

PAQUETES NUGET

Microsoft.EntityFrameworkCore (Conexión bbdd)

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.tool

Creamos una carpeta Context y dentro una clase ApplicationDbContext

namespace ProductAPI.Context

{

public class ApplicationDbContext: DbContex

{

ctor doble TAB

public ApplicationDbContext() CONSTRUCTOR DE LA CLASE

{

}

}

}

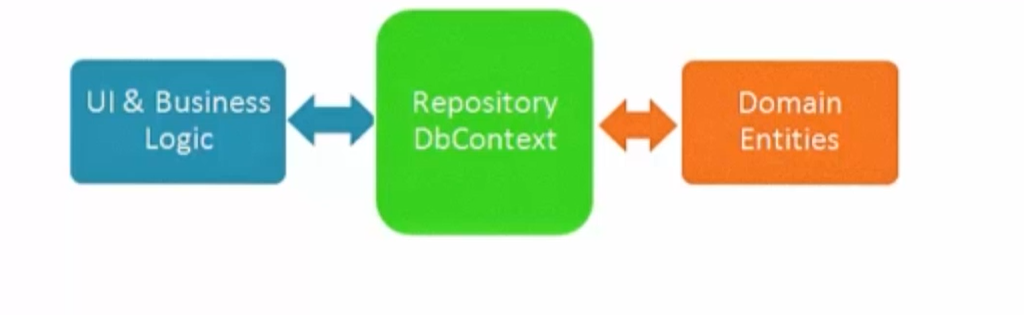
Los comandos de consola;

add-migration NOMBRE (Coge la estructura de Models y crea una carpeta en el proyecto llamada Migrations con la estructura de las tablas a crear.

update-database (crea la base de datos y las tablas)

LOS REPOSITORIOS

Son una capa de abstracción entre la capa de acceso a datos y la lógica del negocio.



No duplicamos código. Lo permite el uso de Entity-Framework.

Creamos una carpeta Repository, dentro otra IRepository y dentro una clase de tipo Interfaz llamada IProductRepository, la ponemos como pública.

Definimos dentro de ella todas las cabeceras de los métodos que va usar la clase Product (Update, delete, créate, retrieve)

Dentro de la carpeta Repository creamos la clase ProductRepository: IProductRepository.

Si le damos a CTRL + . se abre un menú con la opción Implementar interfaz y nos trae todas las cabeceras definidas.

En cada uno de los métodos ponemos el código adecuado:

using ProductAPI.Models;

using ProductAPI.Repository.IRepository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ProductAPI.Context;

namespace ProductAPI.Repository

{

    public class ProductRepository : IProductRepository

    {

        private readonly ApplicationDbContext context;

        public ProductRepository(ApplicationDbContext context)

        {

            this.context = context;

        }

        public bool CreateProduct(Product product)

        {

            context.Product.Add(product);

            return Save();

        }

        public bool DeleteProduct(Product product)

        {

            context.Product.Remove(product);

            return Save();

        }

        public IEnumerable<Product> GetAllProducts()

        {

            return context.Product.ToList();

        }

        public Product GetProduct(int id)

        {

            return context.Product.FirstOrDefault(x => x.Id.Equals(id));

        }

        public bool ProductExist(string name)

        {

            return context.Product.Any(x => x.Name.ToLower().Trim ().Equals (name.ToLower().Trim()));

        }

        public bool ProductExist(int id)

        {

            return context.Product.Any(x => x.Id.Equals(id));

        }

        public bool Save()

        {

            return context.SaveChanges() > 0 ? true : false;

        }

        public bool UpdateProduct(Product product)

        {

            context.Product.Update(product);

            return Save();

        }

    }

}

Por ultimo hay que registrar la interfaz y su implementación, Startup.cs

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

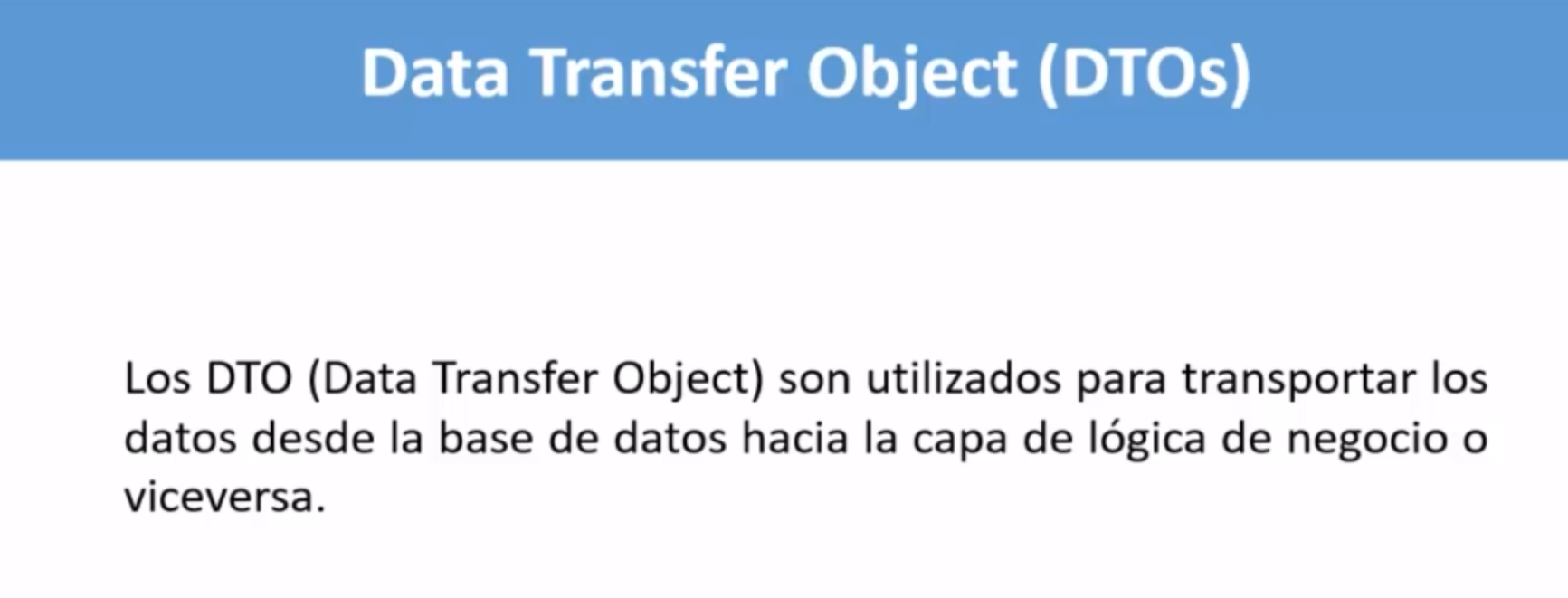
{

services.AddDbContext<ApplicationDbContext>(option=> option.UseSqlServer(Configuration.GetConnectionString("DefaultConnectionString")));

services.AddControllers();

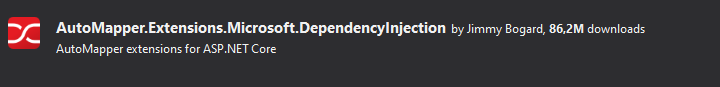
services.AddScoped<IProductRepository, ProductAPI.Repository.ProductRepository>();

}



Elimina de la capa de negocio datos que no se necesitan como id, fechacreacion.  
Creamos en Models una nueva carpeta dto y una clase product.dto.

Agregamos paquete Nugget automapper.



public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(option=> option.UseSqlServer(Configuration.GetConnectionString("DefaultConnectionString")));

services.AddControllers();

services.AddScoped<IProductRepository, ProductAPI.Repository.ProductRepository>();

services.AddAutoMapper(configuration => {

configuration.CreateMap<Product, ProductAPI.Models.dto.productDTO>();

}, typeof(Startup));

}

El siguiente paso es crear el Controlador de Products en la carpeta Controllers, seleccionando nueva Web API en blanco. Nombre ProductsController.cs