Contents

[CAPA DE INFRAESTRUCTURA DE PERSISTENCIA DE DATOS 4](#_Toc159188168)

[Creación Interfaz, carpetas y Proyecto Transversal 4](#_Toc159188169)

[Crear proyecto en Carpeta Infraestructura 4](#_Toc159188170)

[Agregar referencia de Proyecto transversal en infraestructura 5](#_Toc159188171)

[Implementar clase ConnectionFactory 5](#_Toc159188172)

[Crear Proyecto (Interfaz) en Carpeta Infraestructura 6](#_Toc159188173)

[Crear proyecto en Carpeta Dominio 6](#_Toc159188174)

[Agregar referencia a Interfaz de Entidad 6](#_Toc159188175)

[Crear Interfaz Métodos síncronos y asíncronos 7](#_Toc159188176)

[Crear proyecto (Repositorio) en Carpeta de Infraestructura 7](#_Toc159188177)

[Agregar librería Napper 7](#_Toc159188178)

[Implementar clase CustomersRepository 7](#_Toc159188179)

[CAPA DE MODELO DE DOMINIO 8](#_Toc159188180)

[Crear proyecto Interfaz en Carpeta Dominio 8](#_Toc159188181)

[Crear proyecto Core en Carpeta Dominio 8](#_Toc159188182)

[Implementar clase CustomerDomain 9](#_Toc159188183)

[CAPA DE APLICACIÓN 9](#_Toc159188184)

[Crear proyecto DTO en Carpeta Aplicación 10](#_Toc159188185)

[Crear proyecto Mapper en Carpeta Transversal 10](#_Toc159188186)

[Crear proyecto Interface en Carpeta Aplicación 11](#_Toc159188187)

[Crear proyecto Main en Carpeta Aplicación 11](#_Toc159188188)

[CAPA DE SERVICIOS 12](#_Toc159188189)

[Crear proyecto WebApi en Capa de Servicios 12](#_Toc159188190)

[Configurar Archivo Startup 14](#_Toc159188191)

[Integración WebApi con swagger 14](#_Toc159188192)

[Swagger 15](#_Toc159188193)

[Configuración Startup 15](#_Toc159188194)

[Configuración Propiedades del Proyecto WebApi 16](#_Toc159188195)

[POLITICA CORS 16](#_Toc159188196)

[JSON WEB TOKEN JWT 17](#_Toc159188197)

[JWT HABILITAR SEGURIDAD EN WEB API WEB TOKEN 18](#_Toc159188198)

[JWT Implementar Carpeta Infraestructura 18](#_Toc159188199)

[JWT Implementar Carpeta Dominio 19](#_Toc159188200)

[JWT Implementar Carpeta Aplicación 19](#_Toc159188201)

[JWT Implementar Proyecto Webapi 20](#_Toc159188202)

[Mapear Atributos de Appseting con Helpers 20](#_Toc159188203)

[JWT Crear Controlador Users 20](#_Toc159188204)

[JWT Configurar Clase Startup 21](#_Toc159188205)

[JWT VALIDACION DEL TOKEN 22](#_Toc159188206)

[Ejecución de la validación de Token 23](#_Toc159188207)

[CAPA TRANSVERSAL 25](#_Toc159188208)

[Crear Proyecto Logging en Capa Transversal 25](#_Toc159188209)

[Inyectar Dependencias de Logger y realizar Traza en Controlador Customer 25](#_Toc159188210)

[Implementar Startup con Logger 26](#_Toc159188211)

[ACTUALIZACIÓN A NETCORE 3.0 26](#_Toc159188212)

[Actualizar la versión de la plataforma del proyecto WebApi a Net Core 3.0 27](#_Toc159188213)

[Descargar y/o actualizar librerías desde nuget 27](#_Toc159188214)

[Actualizar el código para la inyección de dependencias de la librería AutoMapper 27](#_Toc159188215)

[Actualizar la clase Program con las nuevas interfaces de .NET Core 3.0 28](#_Toc159188216)

[Actualizar clase Startup 29](#_Toc159188217)

[Actualizar Swagger al estándar OPEN API 29](#_Toc159188218)

[Actualizar Librerias de swagger 29](#_Toc159188219)

[METRICAS DE CODIGO 30](#_Toc159188220)

[Índice mantenibilidad del código 30](#_Toc159188221)

[Complejidad ciclomática 30](#_Toc159188222)

[Profundidad de herencia 30](#_Toc159188223)

[Acoplamiento de clases 31](#_Toc159188224)

[Líneas de código 31](#_Toc159188225)

[ACTUALIZACION A NET CORE 3.1 31](#_Toc159188226)

[REFACTORIZACION 33](#_Toc159188227)

[ACTUALIZACION A NET CORE 5.0 34](#_Toc159188228)

[VALIDACION DE PARAMETROS DE ENTRADA 35](#_Toc159188229)

[Crear proyecto Validator 35](#_Toc159188230)

[VALIDAR QUE USUARIO Y PASSWORD NO SEAN VACÍOS 35](#_Toc159188231)

[Adicionar validaciones en proyecto **transversal** 35](#_Toc159188232)

[Adicionar validaciones en proyecto **Aplicación** 35](#_Toc159188233)

[Adicionar Extensión de validaciones en proyecto **WebAPI** 36](#_Toc159188234)

[Actualizar clase STARTUP 36](#_Toc159188235)

[PRUEBAS UNITARIAS Y COBERTURA DE CODIGO 37](#_Toc159188236)

[ESTRUCTURA DE LAS PU 37](#_Toc159188237)

[COBERTURA DE CODIGO 37](#_Toc159188238)

[ACTUALIZACION A NET CORE 6.0 37](#_Toc159188239)

[VERSIONAR API 37](#_Toc159188240)

[Control de versiones usando parámetros de Cadena (Query String) 38](#_Toc159188241)

[Encabezados personalizados 38](#_Toc159188242)

[Parámetros en la URI (Path) 38](#_Toc159188243)

[Versionando API usando parámetros de consulta 38](#_Toc159188244)

[Versionando Controladores 40](#_Toc159188245)

[Se prueba la Versión 2 41](#_Toc159188246)

[CONTROL DE VERIONES 44](#_Toc159188247)

[ENCABEZADOS PERSONALISADOS CONTROL DE VERSIONES 44](#_Toc159188248)

[PARAMETROS DE SEGMENTO CONTROL DE VERSIONES 45](#_Toc159188249)

[Se actualiza encabezado de Controladores 45](#_Toc159188250)

[DEPRECANDO CONTROL DE VERSIONES 47](#_Toc159188251)

[PATRON HEALTH CHECK 49](#_Toc159188252)

[Librerías 49](#_Toc159188253)

[Configurar Health Check 49](#_Toc159188254)

[PATRON REPOSITORY 51](#_Toc159188255)

[Remover la Conexión Antigua y Reemplazar por Nueva 51](#_Toc159188256)

[**Implementar Interfaz Repositorio Genérica** 52](#_Toc159188257)

[Heredar de la Interfaz Genérica 52](#_Toc159188258)

[PATRON UNIT OF WORK 52](#_Toc159188259)

[ARQUITECTURA LIMPIA 53](#_Toc159188260)

[Capa dominio 54](#_Toc159188261)

[aplicación 54](#_Toc159188262)

[Infraestructura 54](#_Toc159188263)

[Reglas de la dependencia 54](#_Toc159188264)

# **CAPA DE INFRAESTRUCTURA DE PERSISTENCIA DE DATOS**

1. Construir la capa de Infraestructura de Datos.

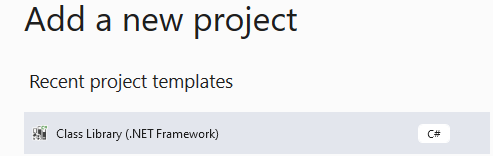
**Data, Interface y Repositorio.**

1. Construir la capa de Infraestructura de Datos.

**Common**.

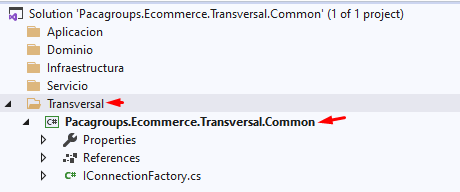
1. Construir la capa de Dominio.

**Entity.**

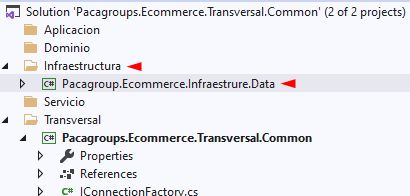


## Creación Interfaz, carpetas y Proyecto Transversal

El proyecto se arrastra a la carpeta transversal para mayor orden.



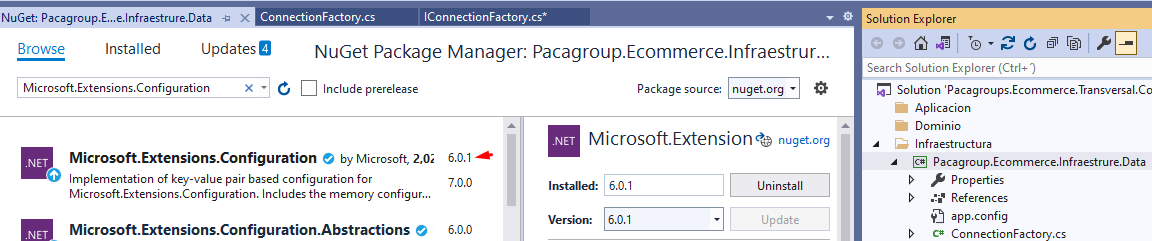
## Crear proyecto en Carpeta Infraestructura

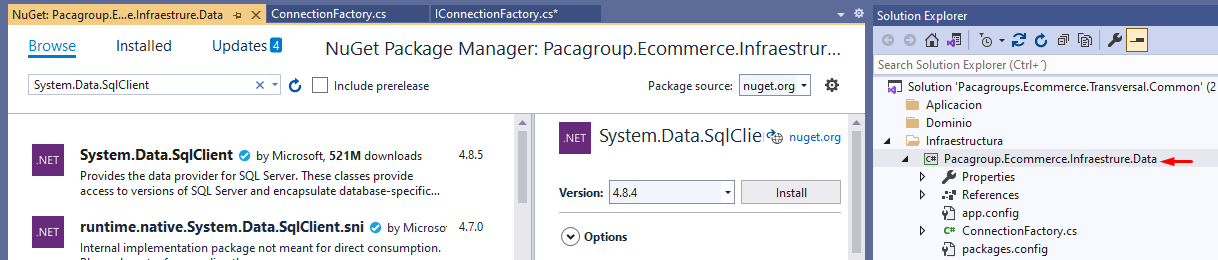


Instalar paquete nuget

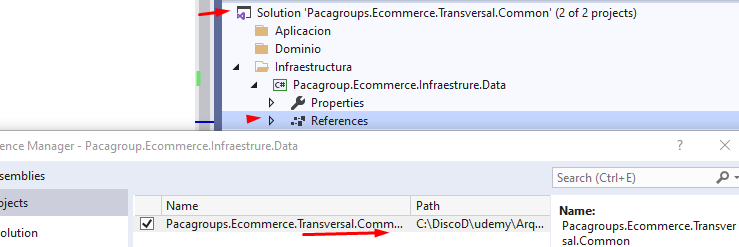
**Microsoft.Extensions.Configuration**

**System.Data.SqlClient**





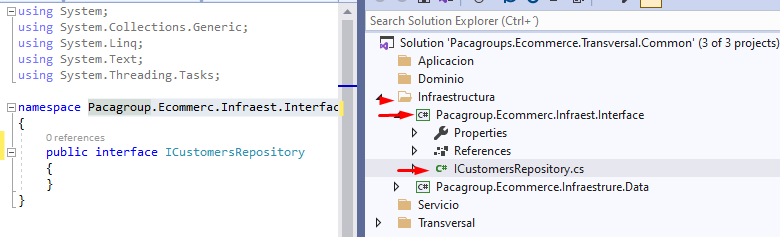
### Agregar referencia de Proyecto transversal en infraestructura



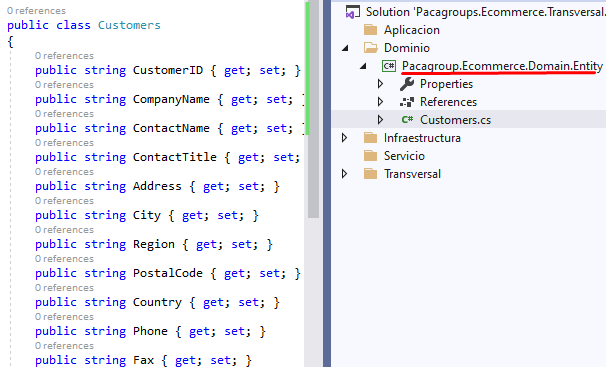
### Implementar clase ConnectionFactory



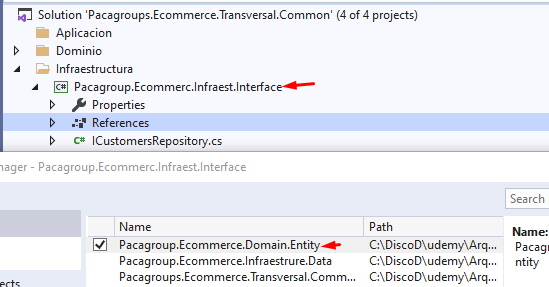
## Crear Proyecto (Interfaz) en Carpeta Infraestructura



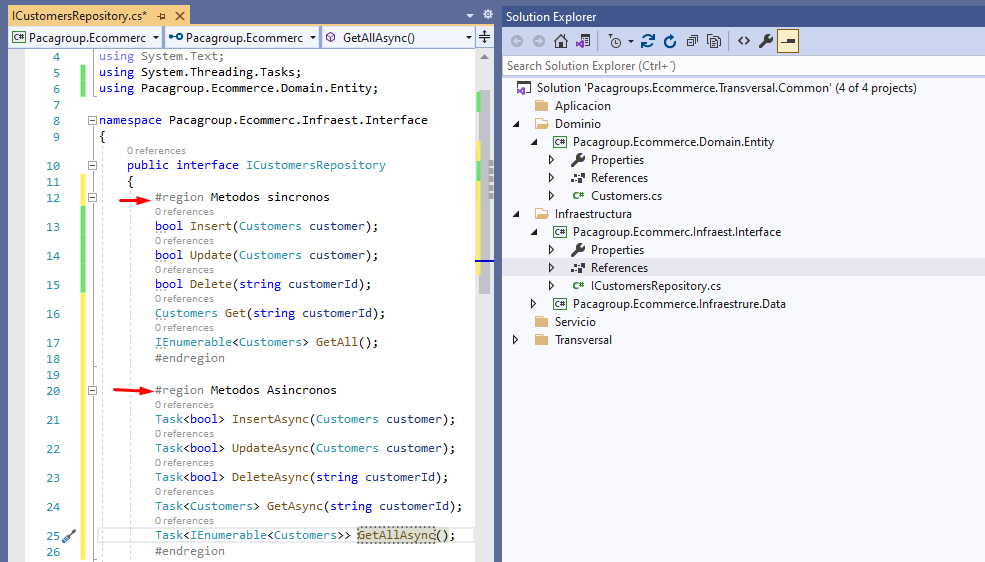
## Crear proyecto en Carpeta Dominio



### Agregar referencia a Interfaz de Entidad

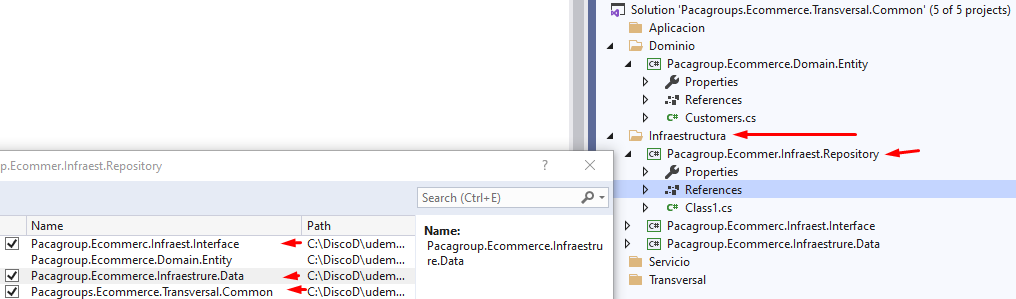


### Crear Interfaz Métodos síncronos y asíncronos

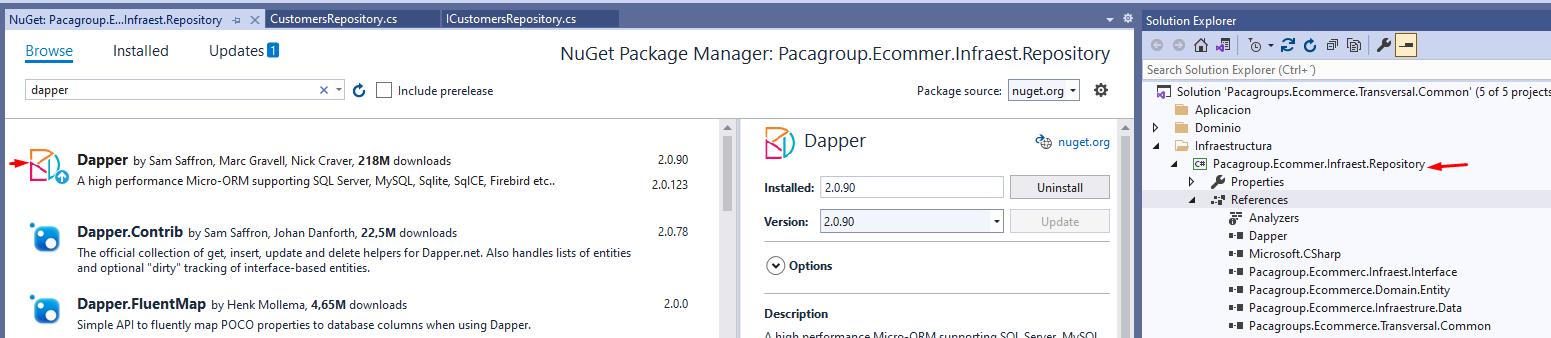


## Crear proyecto (Repositorio) en Carpeta de Infraestructura

Se agregan referencias al proyecto.



### Agregar librería Napper



### Implementar clase CustomersRepository



# **CAPA DE MODELO DE DOMINIO**

1. Construir la capa de dominio.

**Interface** 🡺 Definición de los métodos de aplicación.

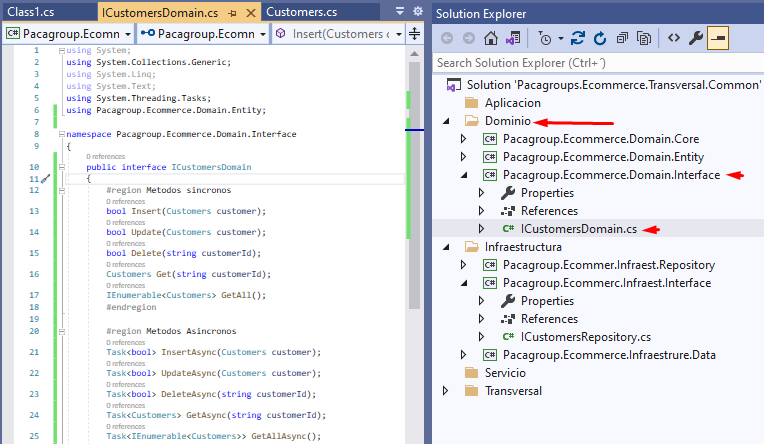
**Core** 🡺 Definición de las interfaces y lógica del negocio.

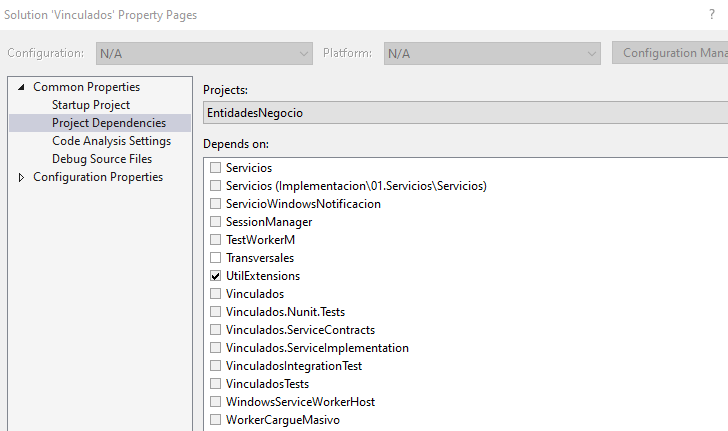
**Entity** 🡺 Entidades del negocio.

Límite de contexto (Microservicio).

## Crear proyecto Interfaz en Carpeta Dominio

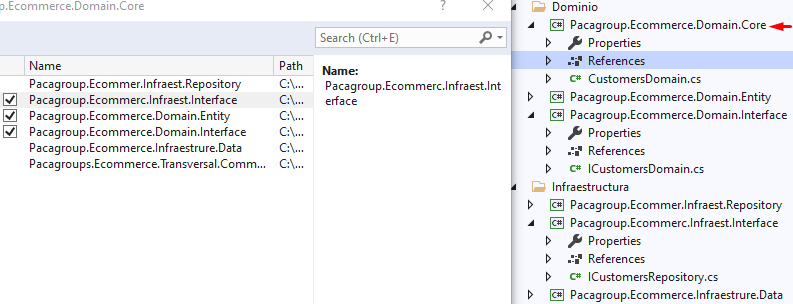
Se crea la Interfaz.



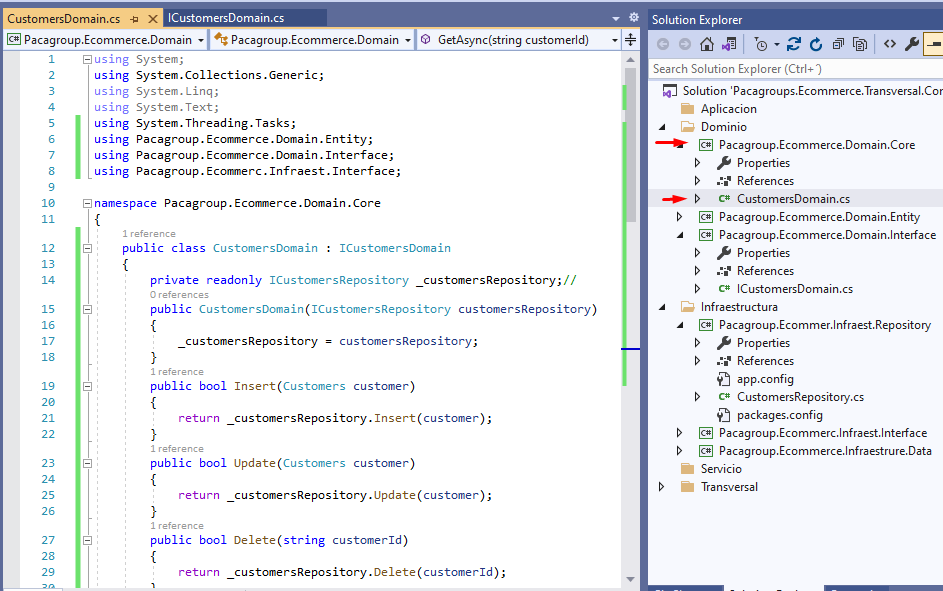


## 

## Crear proyecto Core en Carpeta Dominio



### Implementar clase CustomerDomain



# **CAPA DE APLICACIÓN**

1. Construir capa de aplicación.

**DTO** 🡺 Objetos de transferencia de datos.

**Interface** 🡺 Definición de los métodos de aplicación.

**Main** 🡺 Implementación de las interfaces y gestión de excepciones, transformación de objetos y manejo de transacciones.

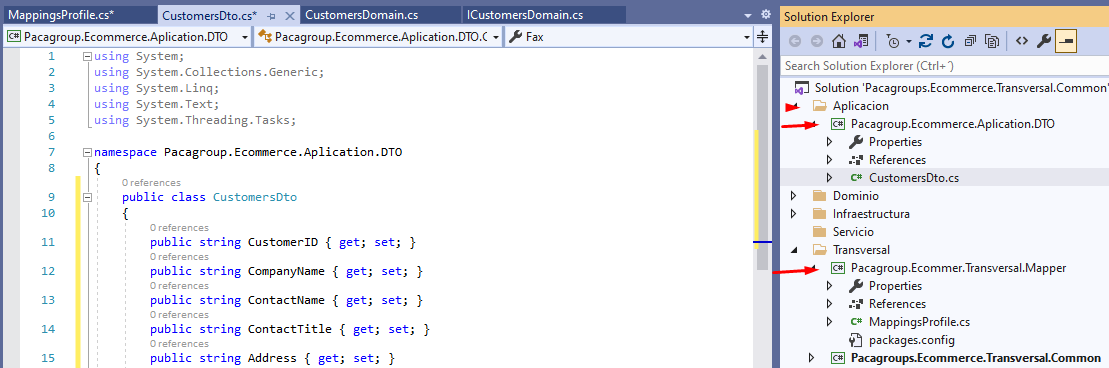
1. Construir Capa Transversal.

**Common** 🡺 crea base, interfaces y usuarios comunes.

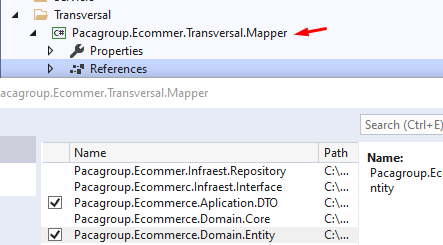
**Mapper** 🡺 Mapeo de objetos (DTO / Entidades y viceversa).

**Logging** 🡺 Trazabilidad (stack trace).

## Crear proyecto DTO en Carpeta Aplicación

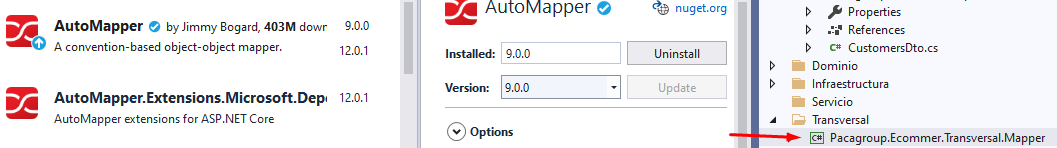


## Crear proyecto Mapper en Carpeta Transversal

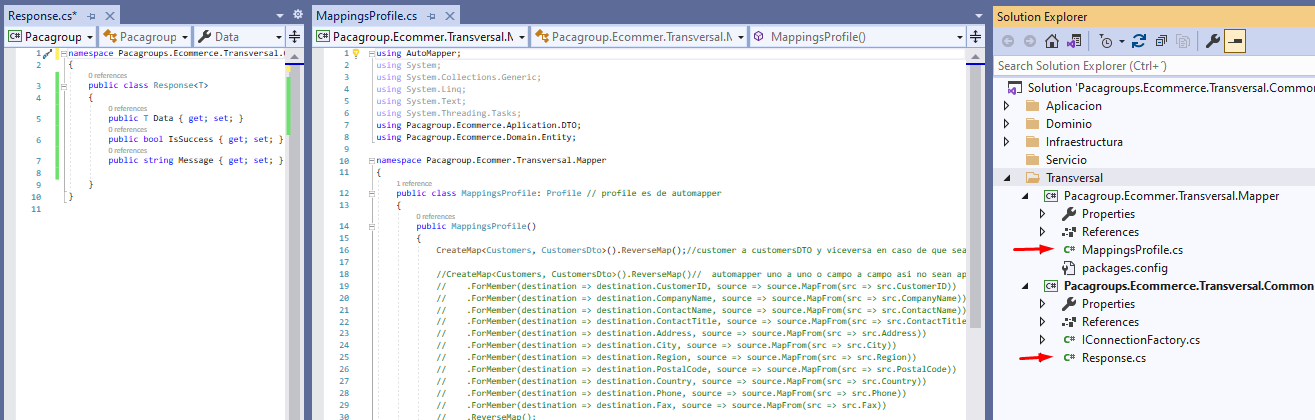


Instalar en proyecto Mapper

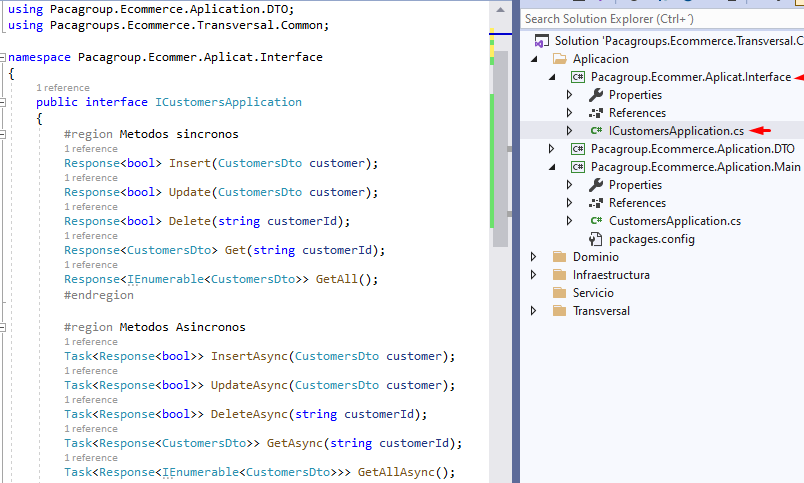
Librería 🡺 Automapper



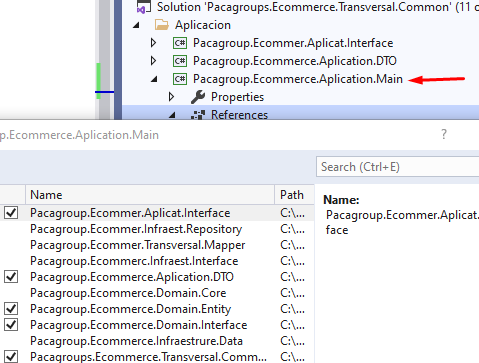
Se crea clase de response y de MappingsProfile.



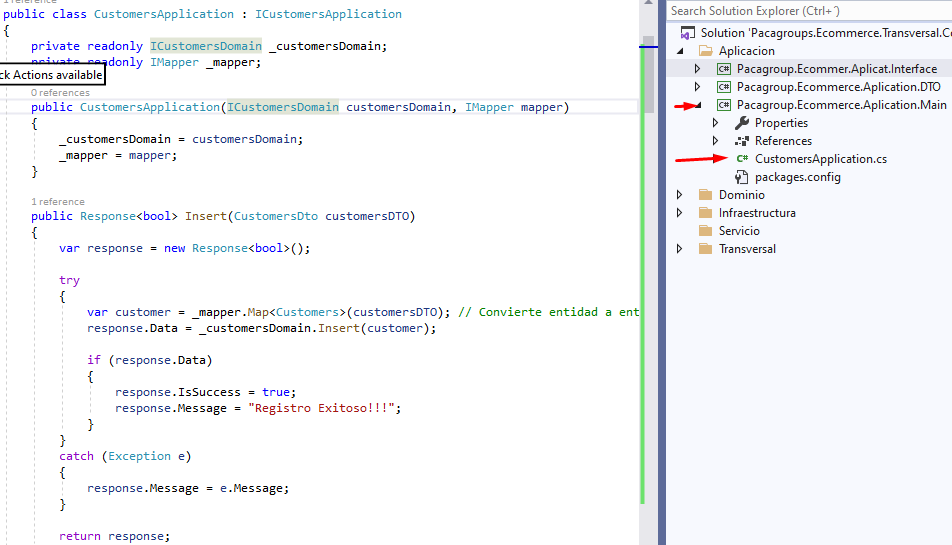
## Crear proyecto Interface en Carpeta Aplicación



## Crear proyecto Main en Carpeta Aplicación



A continuación, se implementan los métodos de la capa de interfaz de la aplicación.



# [**CAPA DE SERVICIOS**](#_Toc133416509)

**WEB API**

* Crear recursos.
* Crear las rutas de acceso (URI).
* Inyección de dependencias.
* Configurar appsettings.json.

1. Construir la capa de servicios.

Principios de Diseño

* Las API REST se diseñan en función de los recursos (cualquier tipo de objeto, datos o servicios a las que el cliente pueda acceder).
* Un recurso tiene un identificador, que es una URI que identifica de forma única al recurso (<http://pacagroup.com/api/Customers/GetAll>).
* Los clientes interactúan con el servicio intercambiando representaciones de recursos (JSON como formato de intercambio).
* Las API REST utilizan un modelo de solicitud sin estado. Las solicitudes HTTP deben ser independientes y pueden construir en cualquier orden, por lo que no es factible mantener información de estado transitoria entre solicitudes.

Tipos de Retorno de Acciones del Controlador.

* Specific Type 🡺 public IEnumerable<Product> Get(){

Return \_repository.GetProducts(); }

* IActionResult 🡺 public IActionResult GetById(int id){

If (!\_repository.TryGetProduct(id, out var producto)){

Return NotFound(); }

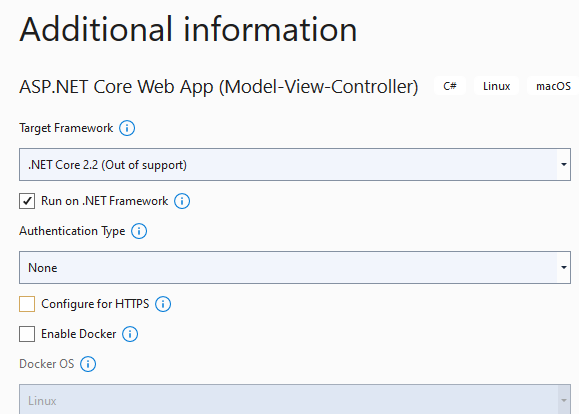
Return Ok (Products(); }

* ActionResult<T> 🡺 public ActionResult<IEnumerable<producto>> Get(){

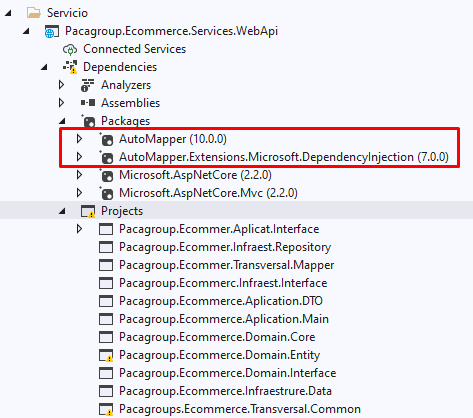
Return \_repository.GetProduct(); }

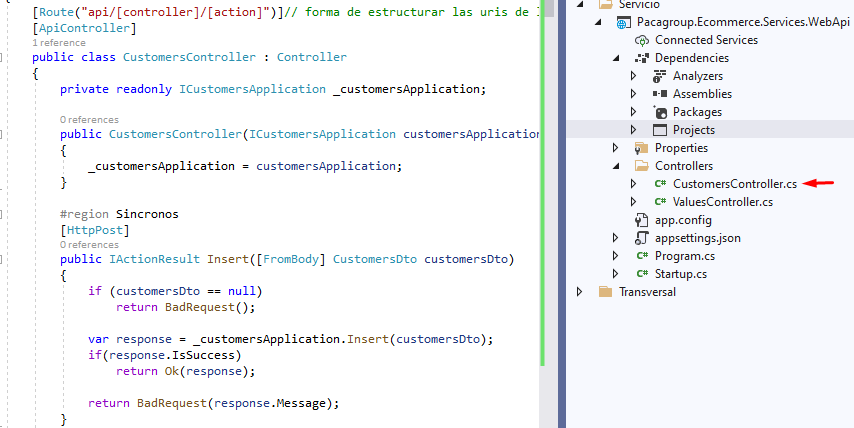
## Crear proyecto WebApi en Capa de Servicios





Crear Controlador y referencias





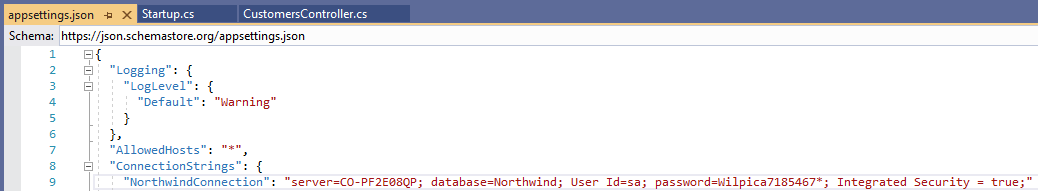
### 

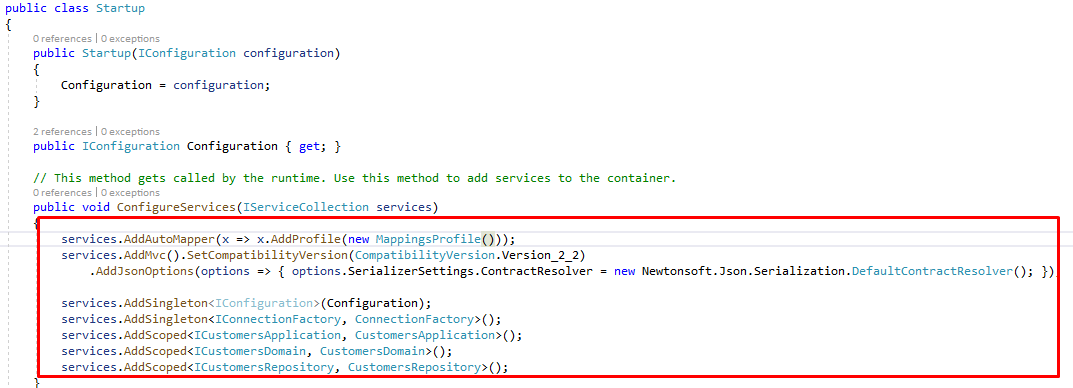
### Configurar Archivo Startup

Tipo de servicio

* Transitorio 🡺 (se usan cada vez que se necesitan),
* Alcance🡺 se instancia una vez por solicitud. SCOPED
* Singleton🡺 se crean una vez.

Conexión string





## Integración WebApi con swagger

**Que es Open API**🡺 Es un formato de descripción de API para REST. Un archivo Open Api permite describir una API de manera completa incluyendo.

* Endpoint disponibles y operaciones en cada uno.
* Parámetros de entrada y salida por cada operación.
* Métodos de autenticación.

**Que es Swagger** 🡺 conjunto de herramientas para especificar OPEN API que ayuda a diseñar, contruir documentar y consumir el Api Rest.

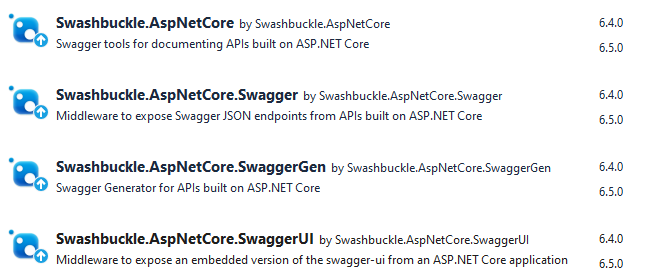
## Swagger

<https://swagger.io/tools/>

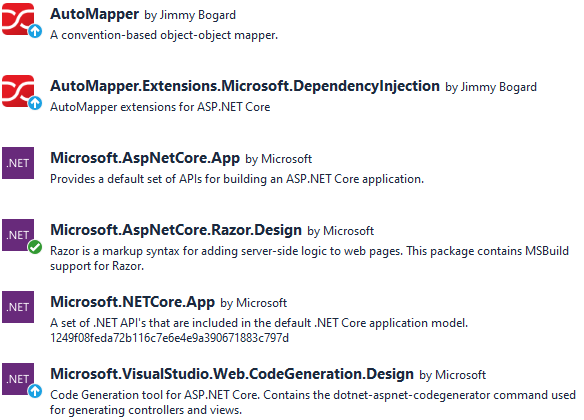
Instalar librerías

* Swashbuckle.AspNetCore.
* Swashbuckle.AspNetCore.Swagger.
* Swashbuckle.AspNetCore.SwaggerGen.
* Swashbuckle.AspNetCore.SwaggerUI.

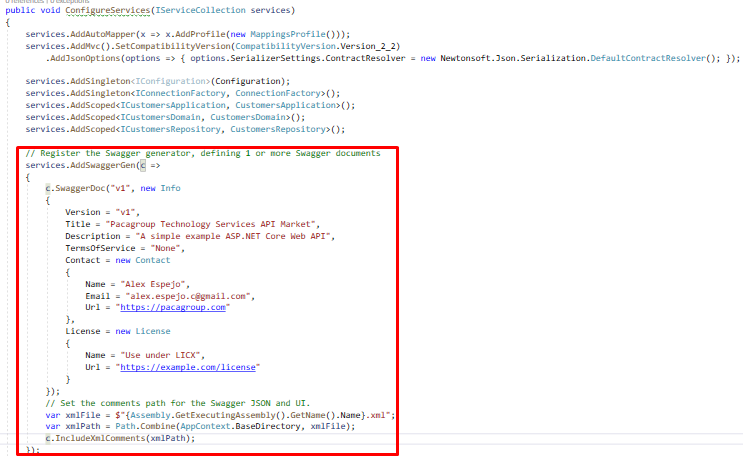
Principales

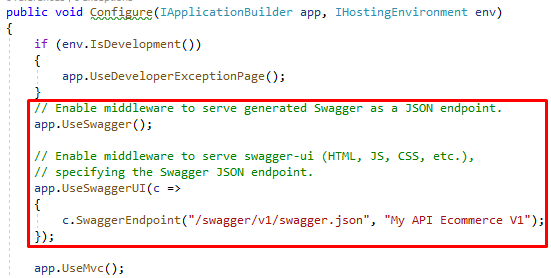


Anteriores



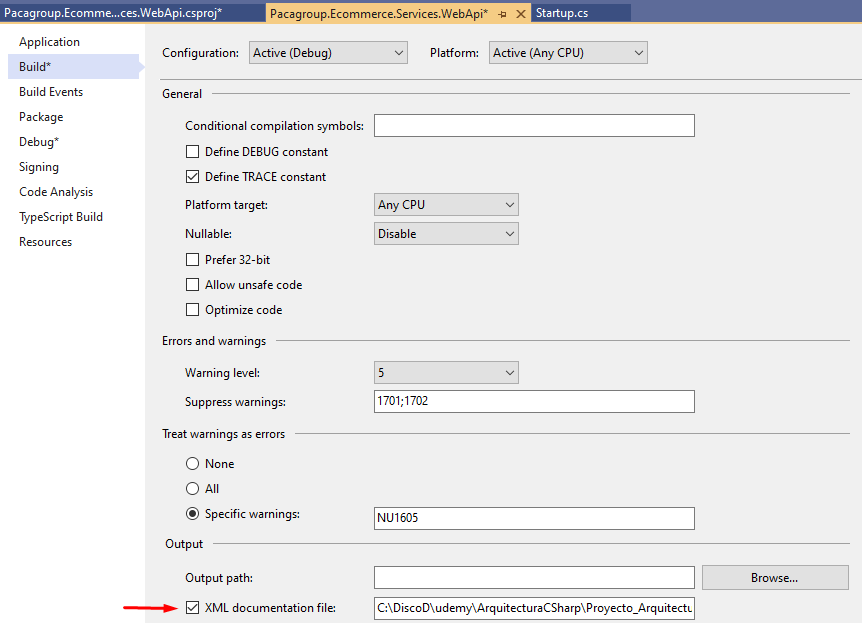
## Configuración Startup

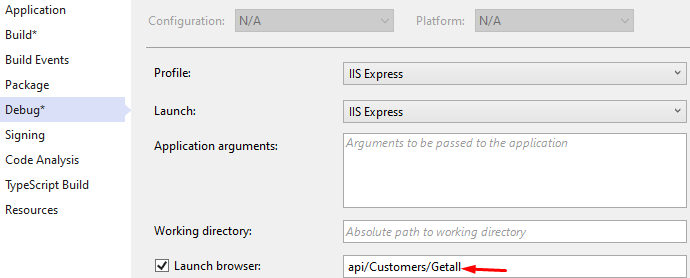




## Configuración Propiedades del Proyecto WebApi

Se habilita la documentación de XML.





<http://localhost:49327/swagger/index.html>

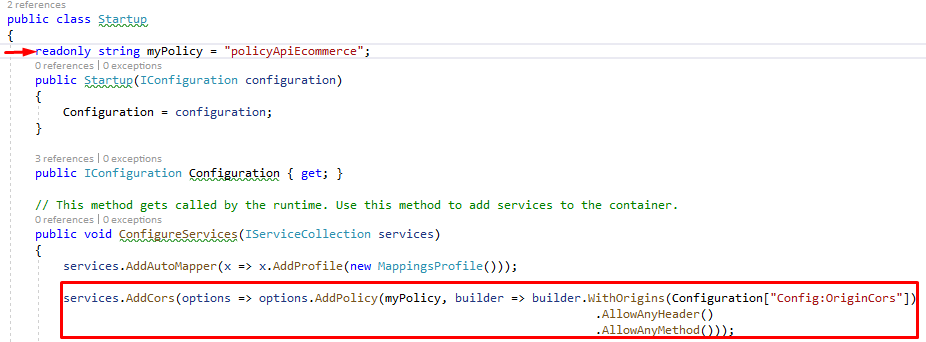
<https://editor.swagger.io/>

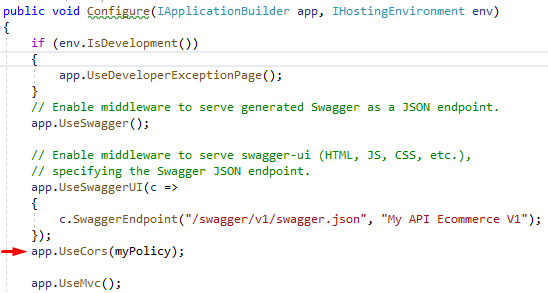
## POLITICA CORS

Es un estándar W3C que permite a un servidor flexibilizar la política del mismo origen. Flexibiliza la seguridad.



Parámetro en común con appseting es **ORIGINCORS.**





## JSON WEB TOKEN JWT

**Token** 🡺 Es una cadena alfanumérica con caracteres aleatorios.

**JWT** 🡺 Es un tipo de token el cual engloba una estructura, la cual puede ser desencriptada por el servidor y de esta forma, autenticarnos como usuario de la aplicación este tiene 3 sesiones separadas por punto.

* **HEADER** 🡺 Contiene tipo de token y algoritmo de encriptamiento.
* **PAYLOAD** 🡺 Contiene datos que identifican al usuario (id, nombre, etc).
* **FIRMA** 🡺 Se genera con las secciones anteriores está en **base64** y después encriptado.

## JWT HABILITAR SEGURIDAD EN WEB API WEB TOKEN

CREATE TABLE [dbo].[Users](

    [UserId] [int] NOT NULL,

    [FirstName] [varchar](50) NOT NULL,

    [LastName] [varchar](50) NOT NULL,

    [UserName] [varchar](50) NOT NULL,

    [Password] [varchar](50) NOT NULL

) ON [PRIMARY]

ALTER TABLE [dbo].[Users] ADD  CONSTRAINT [PK\_Users] PRIMARY KEY NONCLUSTERED

(

    [UserId] ASC

)

CREATE PROCEDURE UsersGetByUserAndPassword

(

    @UserName varchar(50),

    @Password varchar(50)

)

AS

BEGIN

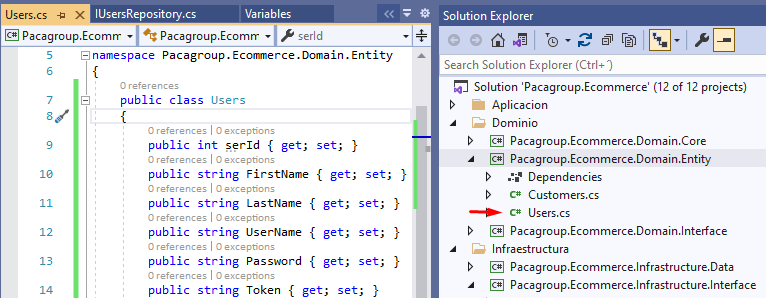
    SELECT UserId, FirstName, LastName, UserName, NULL as Password

    FROM Users

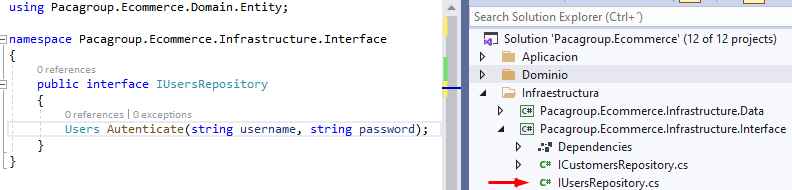
    WHERE UserName = @UserName and Password = @Password

END

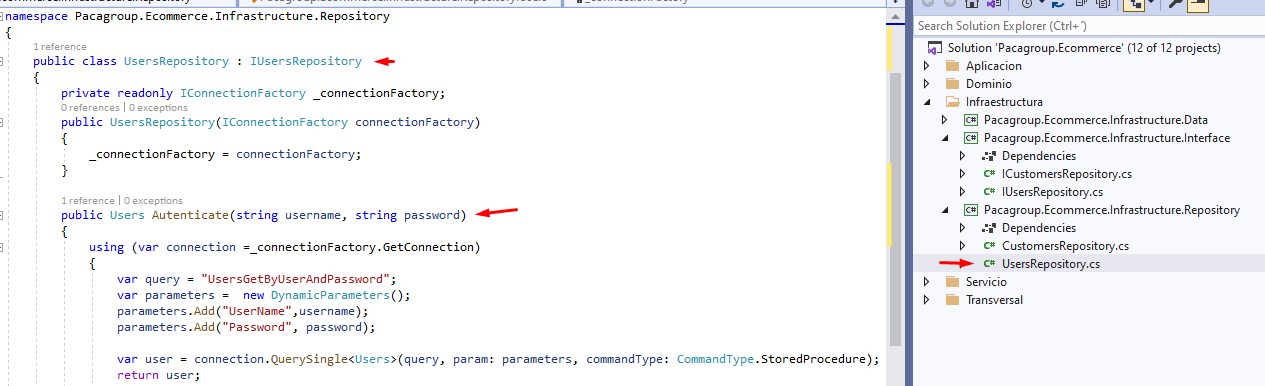
Se crea clase entidad usuario en Carpeta Dominio Entidad.



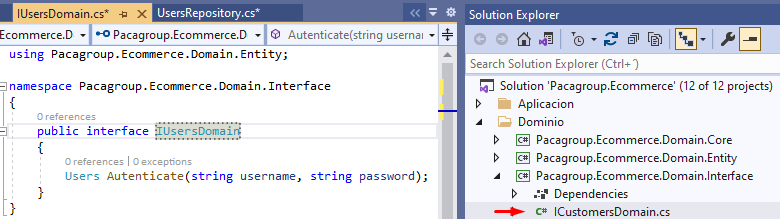
## JWT Implementar Carpeta Infraestructura



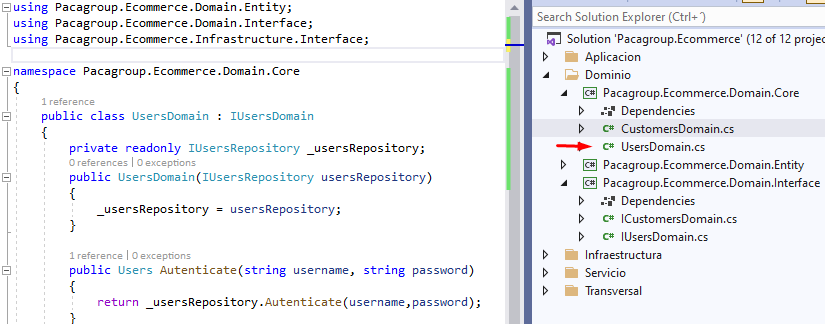
Se Implementa la interfaz en Carpeta infraestructura proyecto de Repositorio



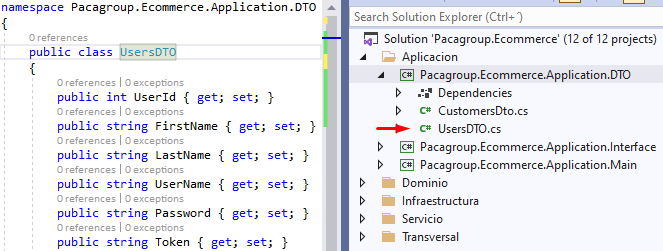
## JWT Implementar Carpeta Dominio



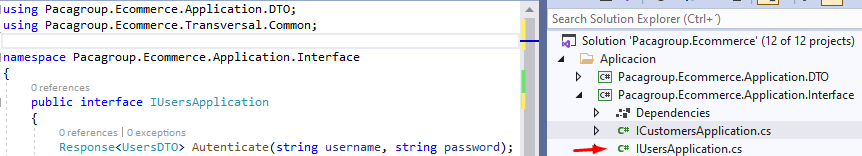
Se Implementa la interfaz en Carpeta Dominio proyecto de Core



## JWT Implementar Carpeta Aplicación



Se crea interfaz en Carpeta de Aplicación proyecto Interfaz

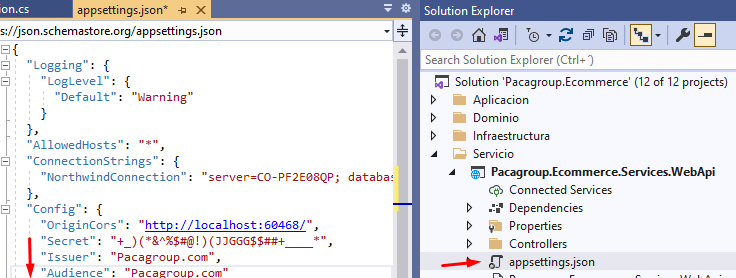


Se Implementa la interfaz **IUsersDomain** en Carpeta Aplicación proyecto Main



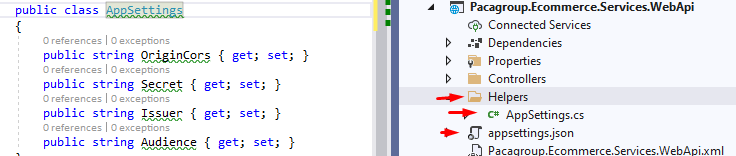
## JWT Implementar Proyecto Webapi

Se ajusta appSeting para generar Token

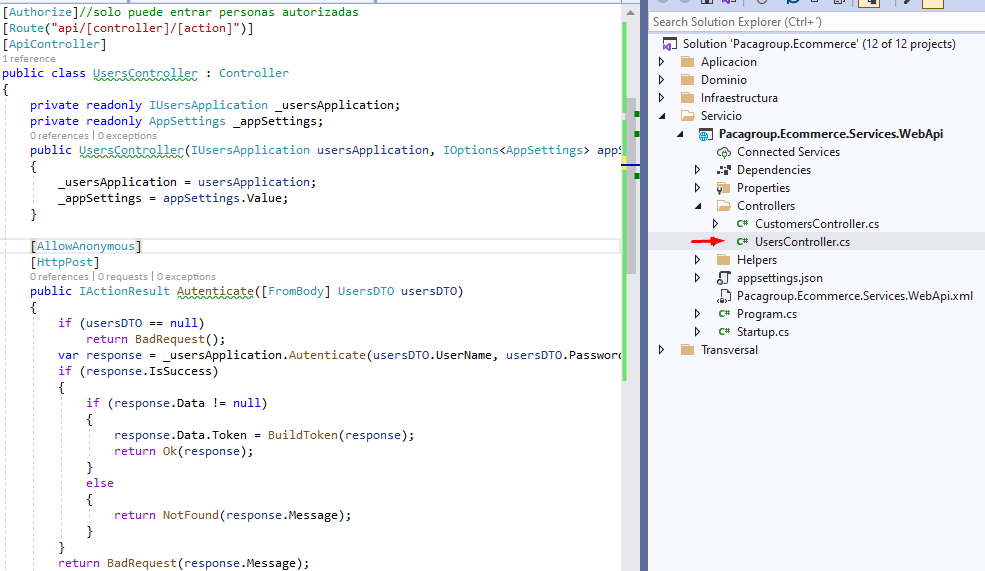


### Mapear Atributos de Appseting con Helpers

Los atributos del archivo appseting deben coincidir con la clase.

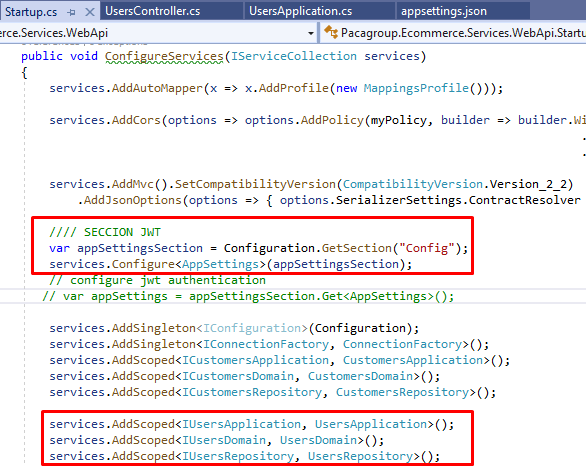


### JWT Crear Controlador Users

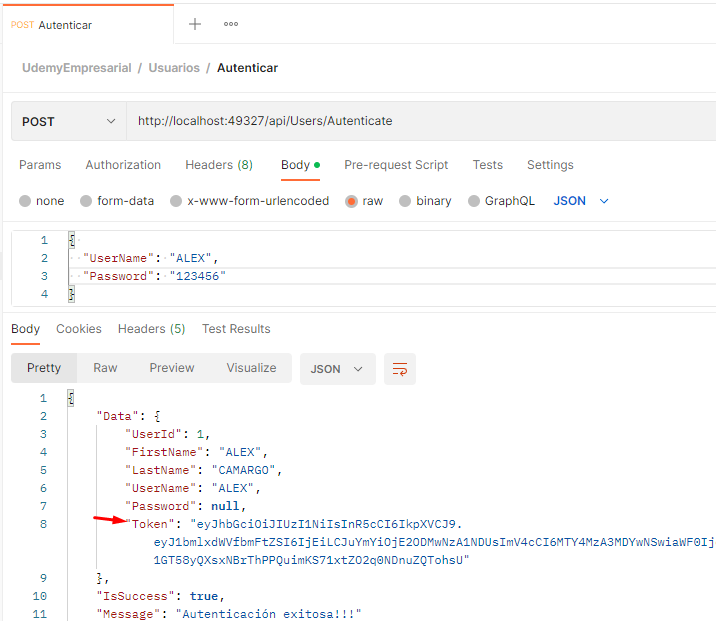


### JWT Configurar Clase Startup



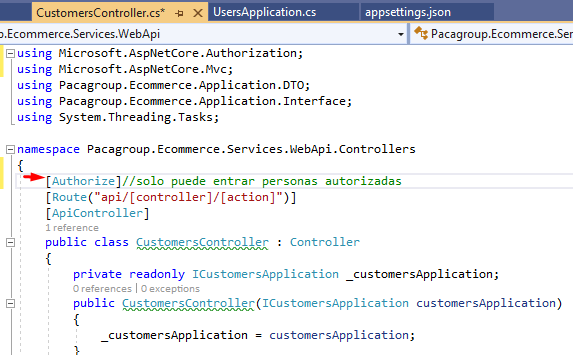


**Se genera el token correctamente**



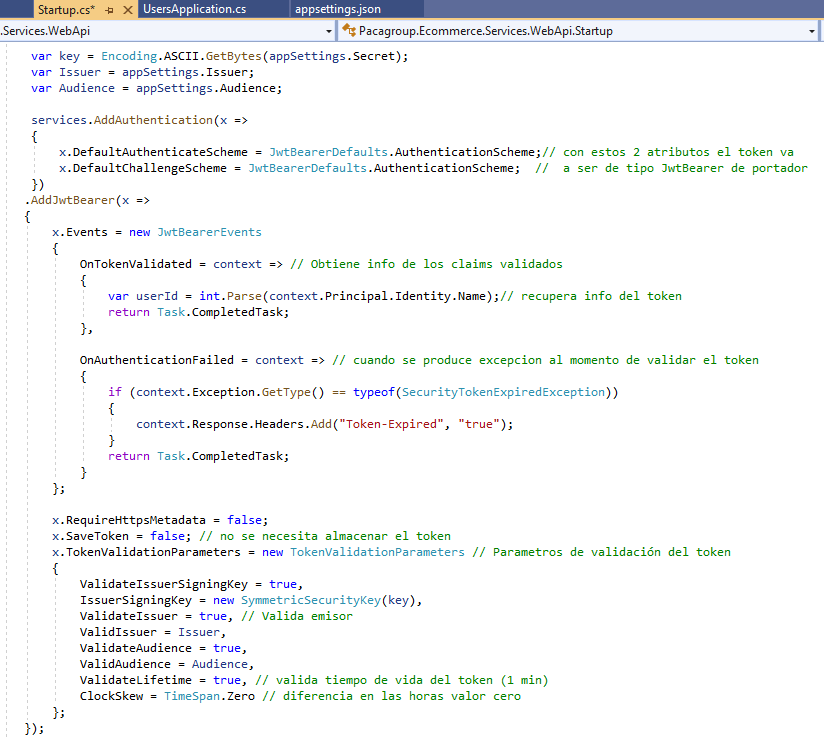
## JWT VALIDACION DEL TOKEN

Agregar seguridad en Customer

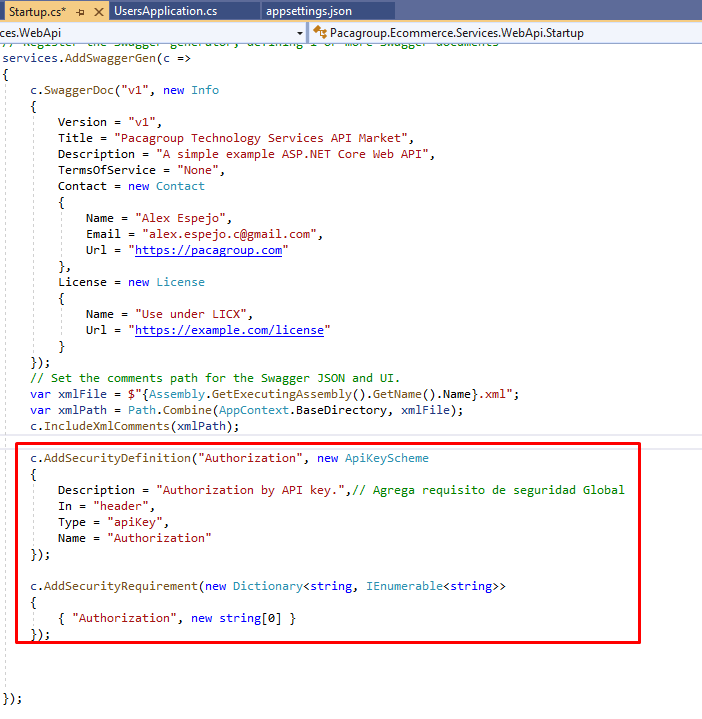


En **Startup** se llama Appseting.

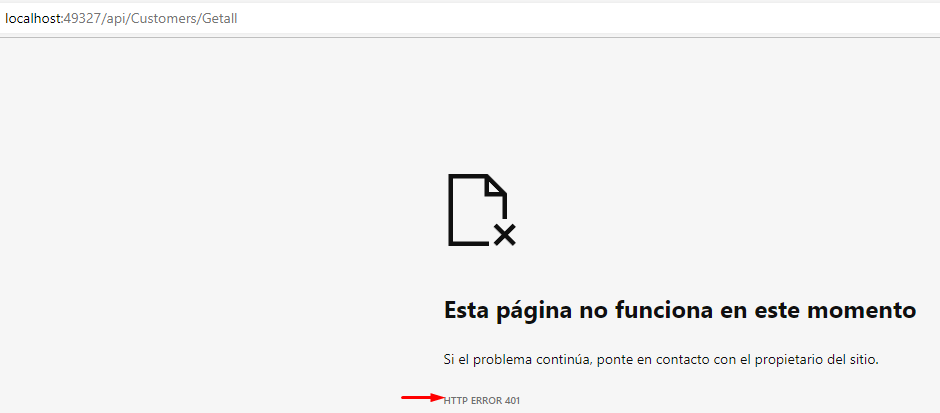
Se agrega todo este codigo

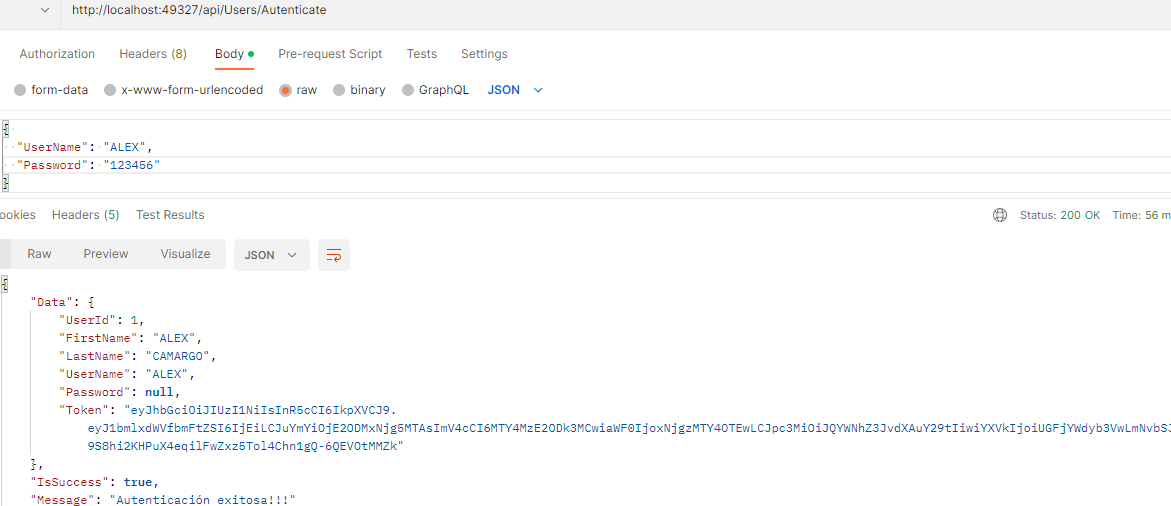


En el swagger también se valida la seguridad

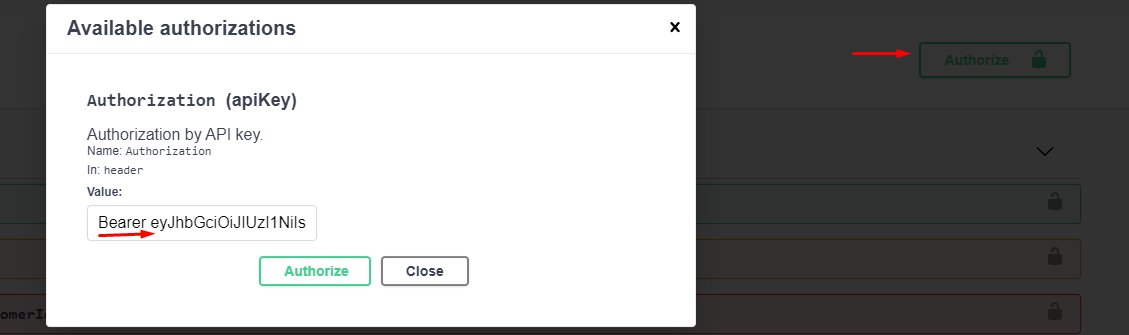


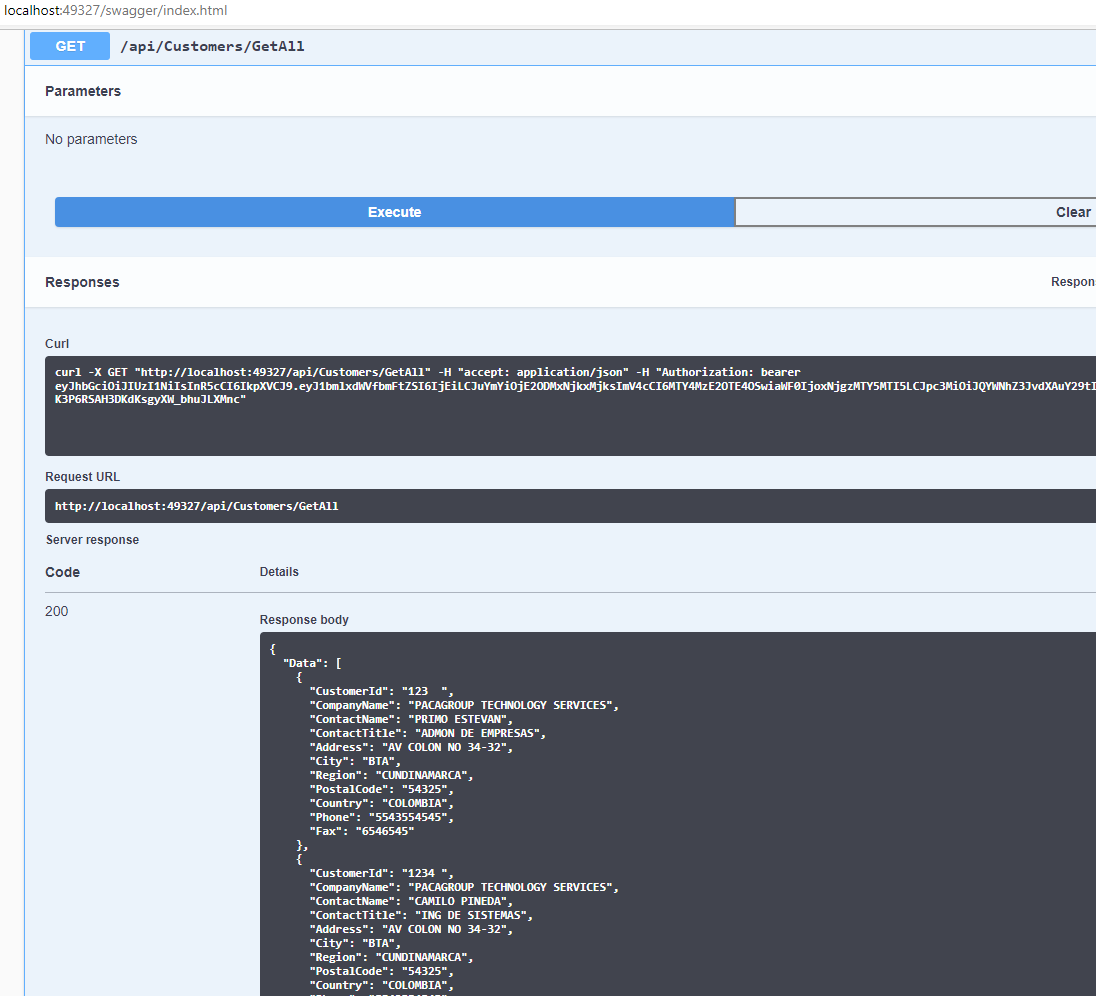
### Ejecución de la validación de Token

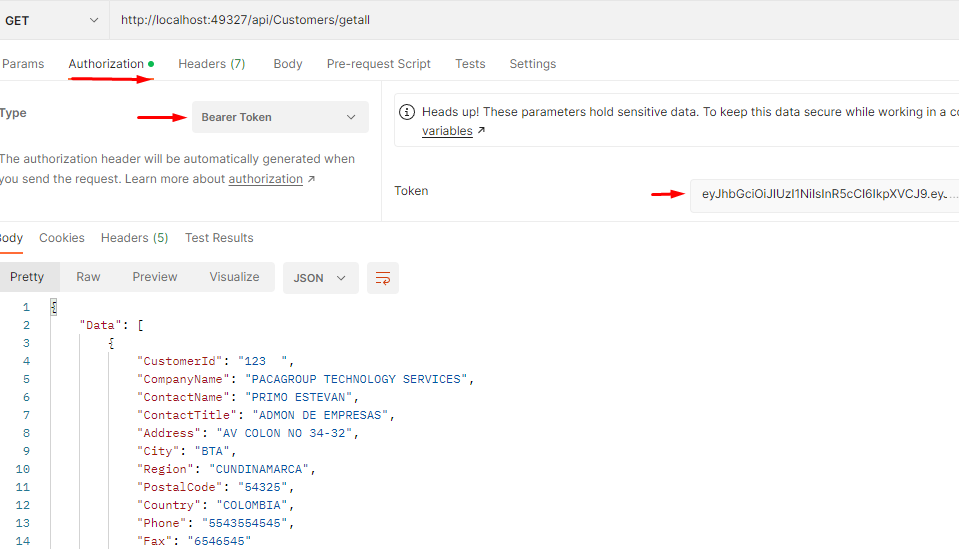




Se ingresa la palabra Bearer y el token







# **CAPA TRANSVERSAL**

* **Common**

Clases base, interfaces y funciones comunes.

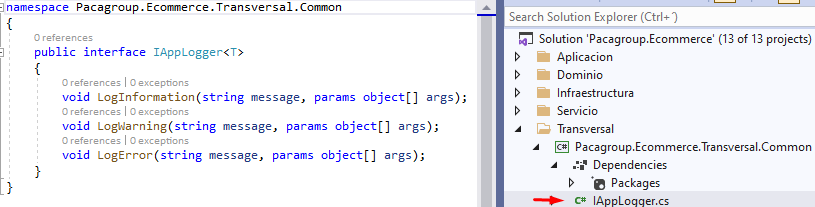
* **Mapper**

Mapeo de objetos DTO / entidades y viceversa.

* **Logging**

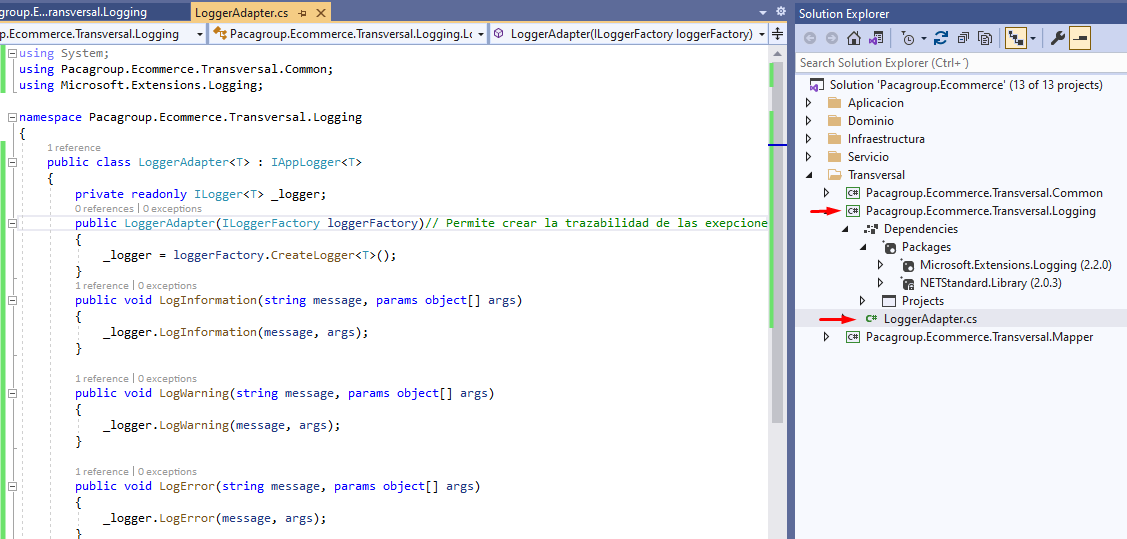
Trazabilidad (stack trace).

Se crea Interfaz en Proyecto Transversal **Common**

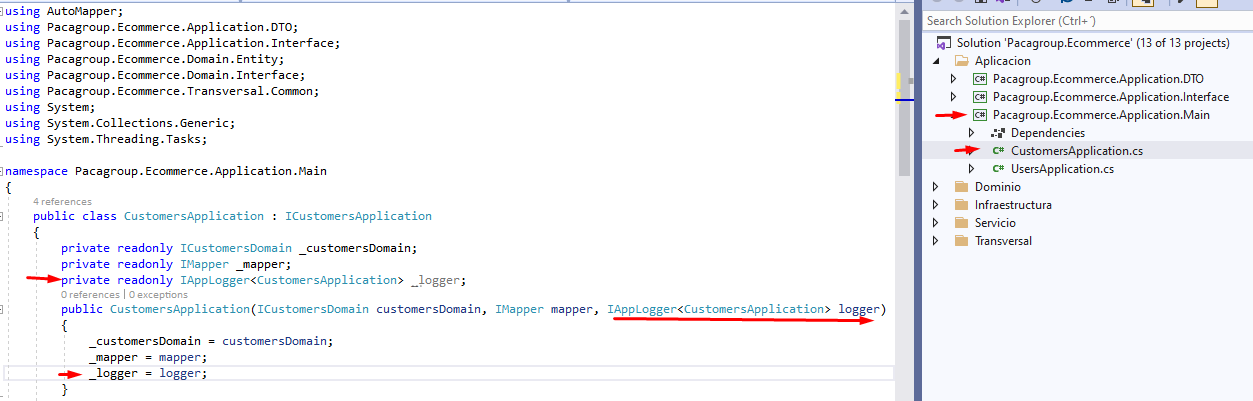


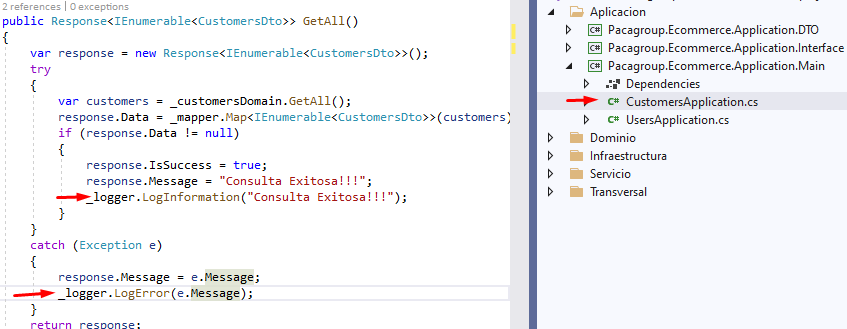
## Crear Proyecto Logging en Capa Transversal

* Instalar librería Microsoft.Extensions.Logging

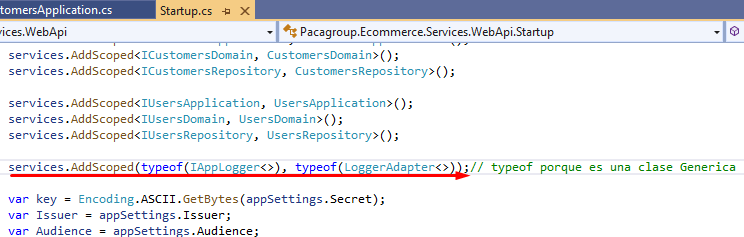


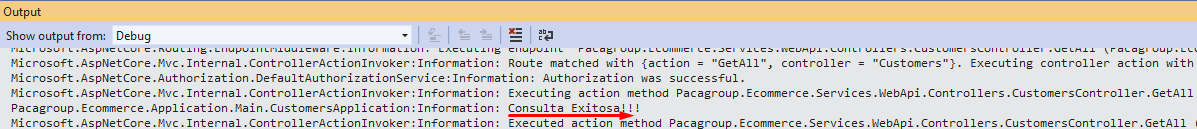
## Inyectar Dependencias de Logger y realizar Traza en Controlador Customer





## Implementar Startup con Logger





En la imagen anterior se evidencia la escritura correcta del log.

# **ACTUALIZACIÓN A NETCORE 3.0**

Prequisitos:

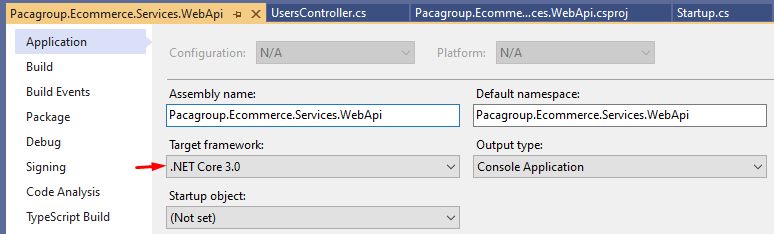
* .NET Core SDK
* .NET Core Runtime.

<https://dotnet.microsoft.com/en-us/download>

Se debe realizar lo siguiente:

* Actualizar la versión de la plataforma del proyecto webApi a Net Core 3.0.
* Descargar y/o actualizar librerías desde nuget.
* Actualizar el código para la inyección de dependencias de la librería AutoMapper.
* Actualizar la clase program con las nuevas interfaces de .NET Core 3.0.
* Actualizar clase Startup.

## Actualizar la versión de la plataforma del proyecto WebApi a Net Core 3.0



## Descargar y/o actualizar librerías desde nuget

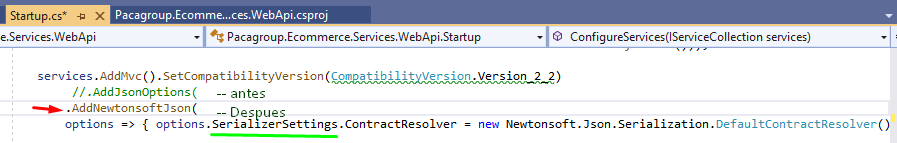
Microsoft.IdentityModel.Tokens

System.IdentityModel.Tokens.Jwt

Microsoft.AspNetCore.Authentication.JwtBearer

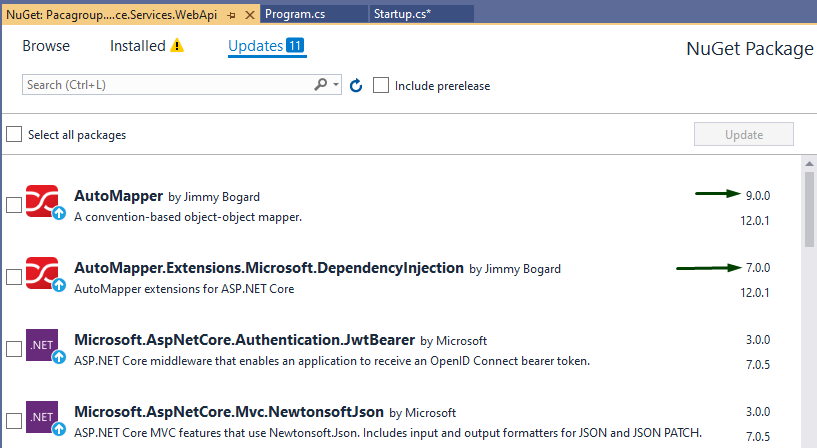
Microsoft.AspNetCore.Mvc.NewtonsoftJson

\*\* Modificar la clase startup

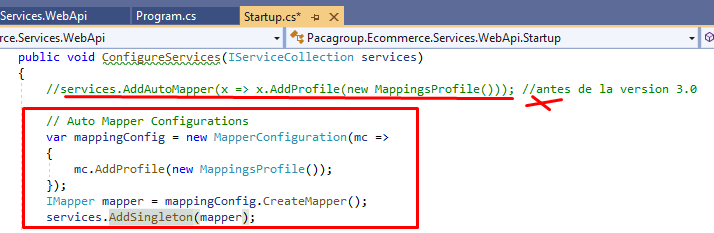


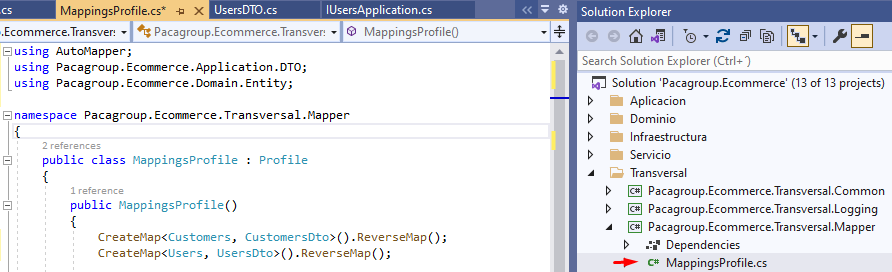
NOTA: Actualizar las librerias a los paquetes de cada Proyecto.

## Actualizar el código para la inyección de dependencias de la librería AutoMapper

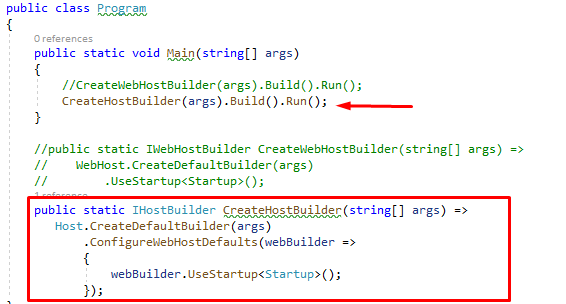


Se cambia la versión de autommapper y de inyección de dependencias.

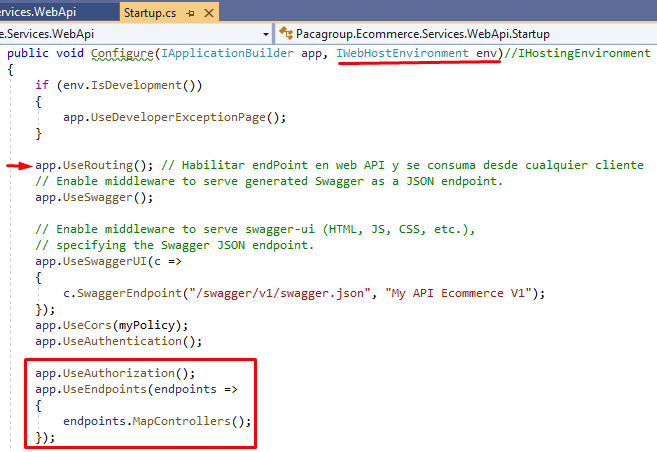




## Actualizar la clase Program con las nuevas interfaces de .NET Core 3.0



## Actualizar clase Startup



# Actualizar Swagger al estándar OPEN API

OPEN API 🡺 Es un estándar para definir contratos de API.

Actualizar los EndPoints de los Controladores

Migrar Componentes de Swagger al estándar Open API

## Actualizar Librerias de swagger

A continuación, se mencionan las librerias que se cambian a versión 5.0.

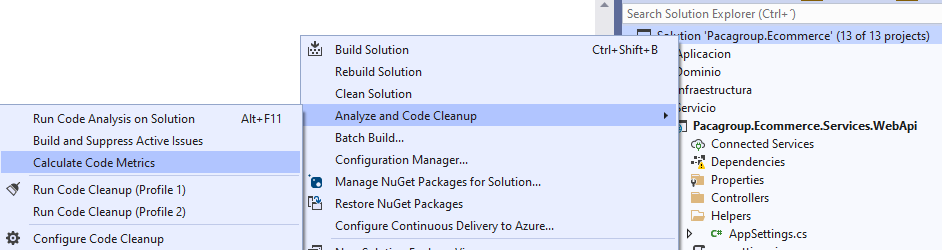
* Swashbuckle.AspNetCore.Swagger.
* Swashbuckle.AspNetCore.SwaggerUI.
* Swashbuckle.AspNetCore.SwaggerGen

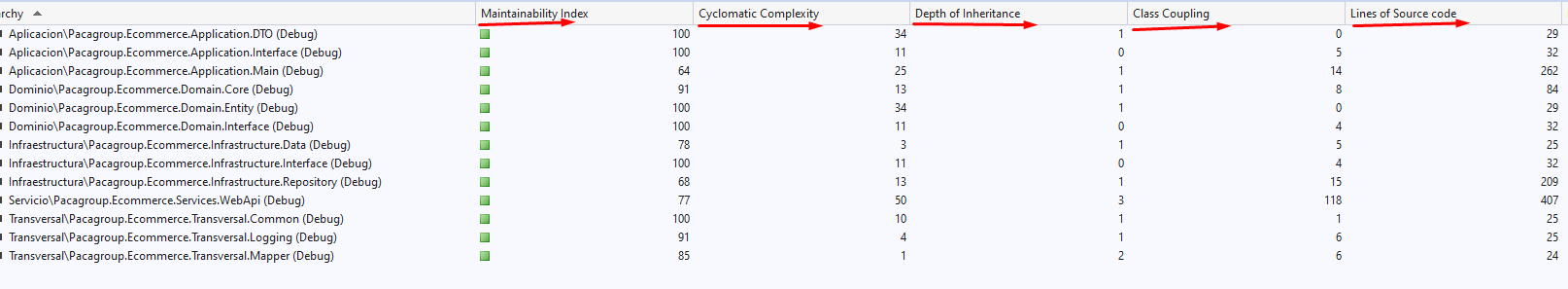
# METRICAS DE CODIGO

Se calculan las principales métricas:

* Índice mantenibilidad del código. ¨¨¨¨¨¨Principal centro
* Complejidad ciclomática.
* Acoplamiento de clases.
* Profundidad de herencia.
* Líneas de código.

Calcular métricas de Código





## Índice mantenibilidad del código

Calcula el valor entre 0 y 100.

**Umbrales**

0.9 =🡺 Rojo

10-19 🡺 Amarillo

20-100 🡺 Verde.

## Complejidad ciclomática

Mide la complejidad estructural del código. Se obtiene calculando el numero de diferentes rutas de código en el flujo del programa.

**Umbrales** 🡺 Valores bajos menores a 50.

## Profundidad de herencia

Indica la cantidad de clases que se heredan entre sí, hasta la clase base.

**Umbrales** 🡺 Valores bajos menores a 40.

## Acoplamiento de clases

Mide el acoplamiento a través de variables locales , tipos de retorno, llamadas a métodos. El alto acoplamiento indica un diseño que es difícil de reutilizar y mantener debido a sus muchas interdependencias con otros tipos.

**Umbrales** 🡺 Valores bajos menores a 40.

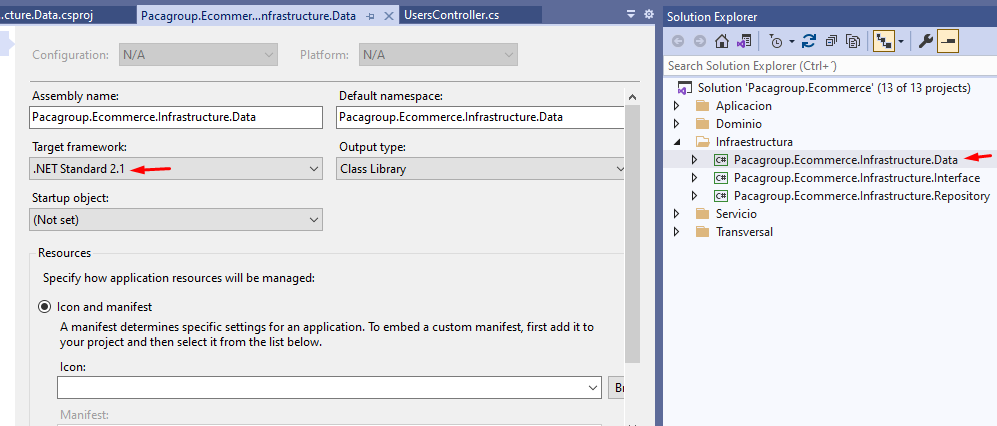
## Líneas de código

**Umbrales** 🡺 Valores bajos menores a 30 líneas de código en métodos.

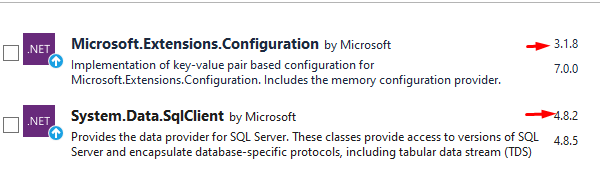
# [**ACTUALIZACION**](#_Toc134110840) **A NET CORE 3.1**

**Pacagroup.Ecommerce.Infrastructure.Data**

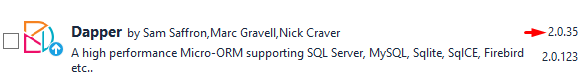
Antes estaba en versión 2.0



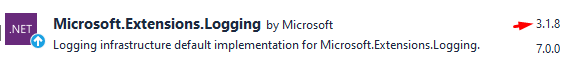
Se actualizan versiones NUget



**Pacagroup.Ecommerce.Infrastructure.Repository**



**Pacagroup.Ecommerce.Transversal.Logging**



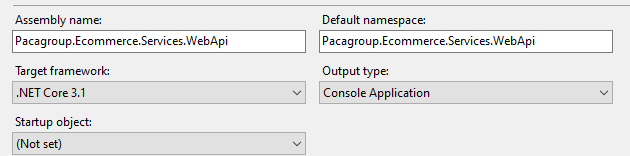
**Pacagroup.Ecommerce.Transversal.Mapper**



**Pacagroup.Ecommerce.Application.Main**

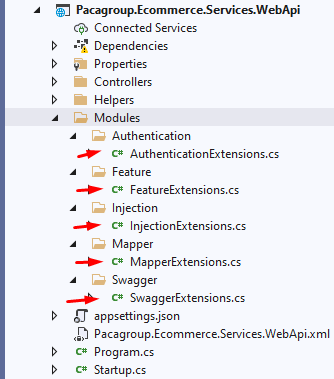


**Pacagroup.Ecommerce.Services.WebApi**



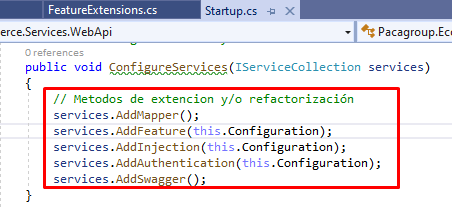
# REFACTORIZACION

Se crean las clases de extensión.



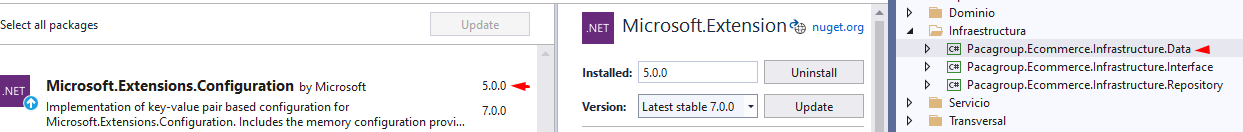
Se referencia el método de las clases de extensión en la clase startup.

Nota🡺 se traslada lo que había antes de startup y luego se invoca.

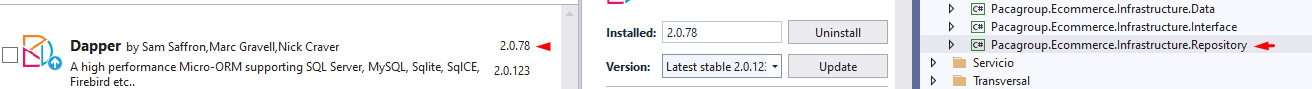


# [**ACTUALIZACION**](#_Toc134110840) **A NET CORE 5.0**

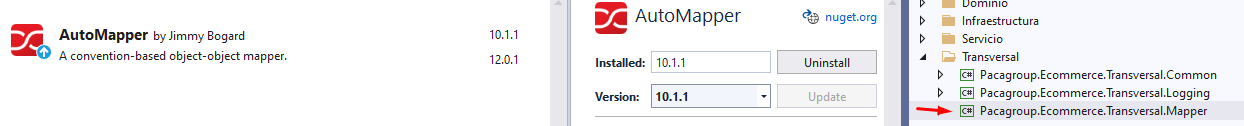
Pacagroup.Ecommerce.Infrastructure.Data



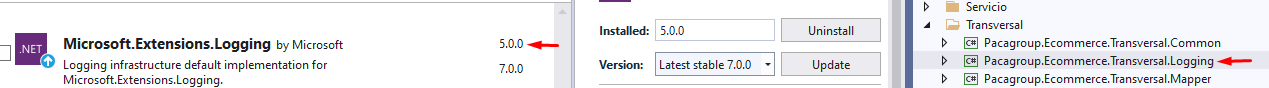
Pacagroup.Ecommerce.Infrastructure.Repository



Pacagroup.Ecommerce.Transversal.Mapper

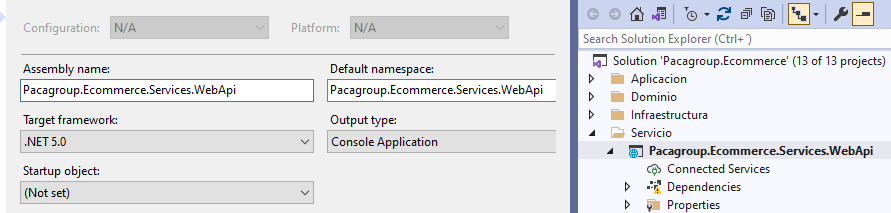


Pacagroup.Ecommerce.Transversal.Logging



Pacagroup.Ecommerce.Application.Main





# VALIDACION DE PARAMETROS DE ENTRADA

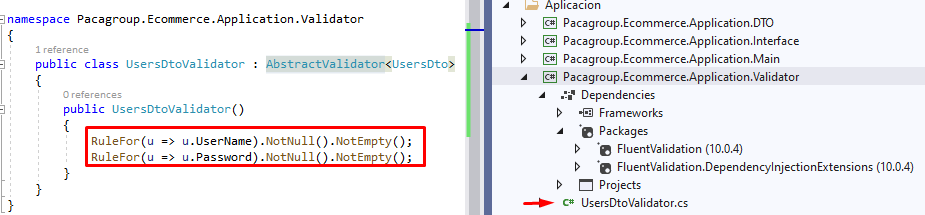
<https://docs.fluentvalidation.net/en/latest/>

## Crear proyecto Validator

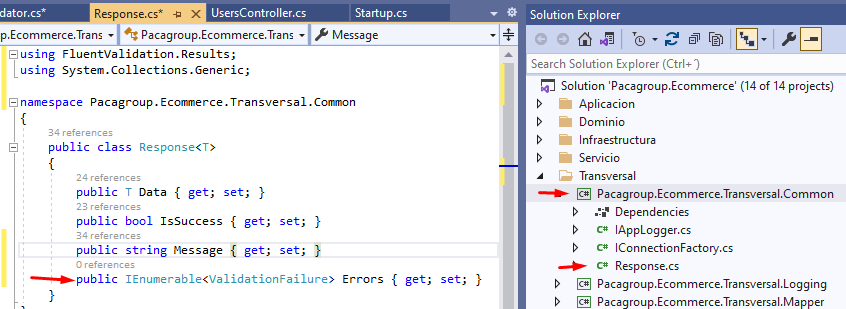
Pacagroup.Ecommerce.Application.Validator

* Instalar librería **FluentValidation**, **FluentValidation**.**DependencyInjection**.

## VALIDAR QUE USUARIO Y PASSWORD NO SEAN VACÍOS



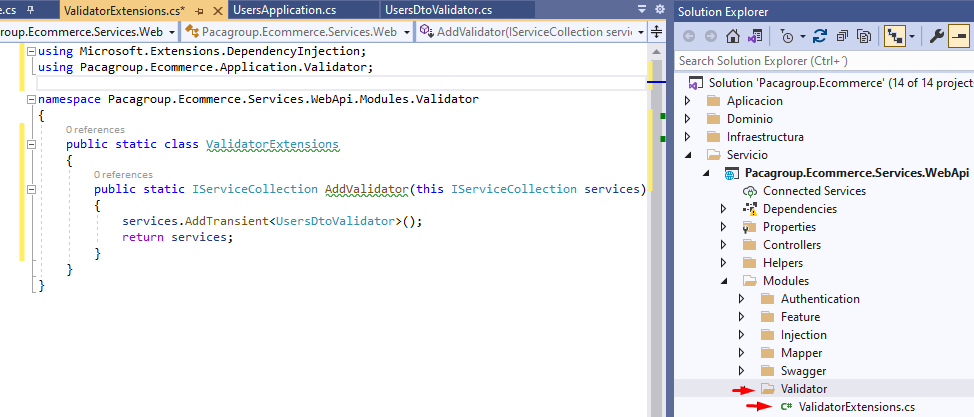
## Adicionar validaciones en proyecto **transversal**



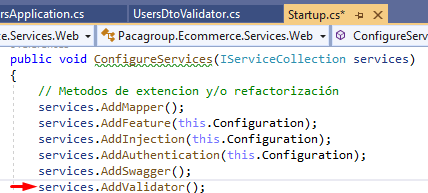
## Adicionar validaciones en proyecto **Aplicación**

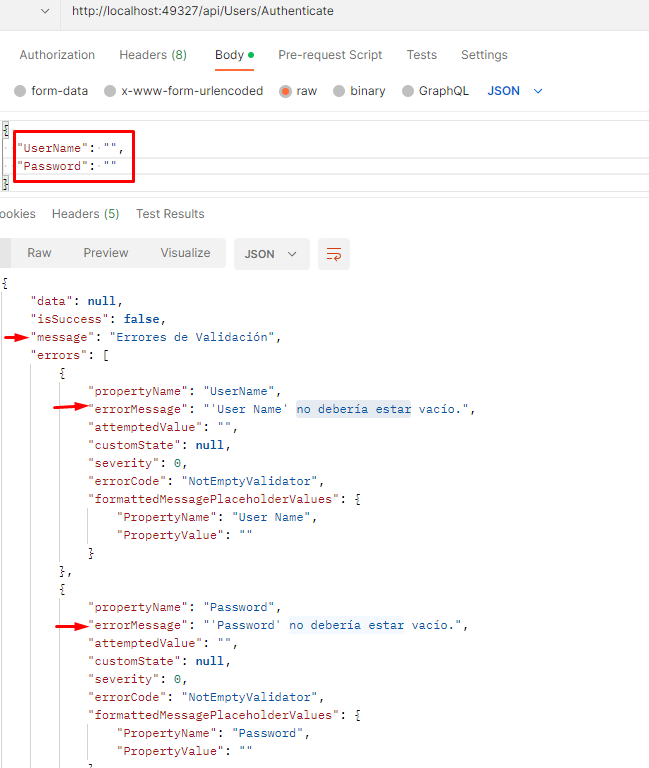


## Adicionar Extensión de validaciones en proyecto **WebAPI**



## Actualizar clase STARTUP





# **PRUEBAS UNITARIAS Y COBERTURA DE CODIGO**

Consiste en probar áreas de Código de forma aislada del resto y obtener el resultado que se espera.

El nombre de la prueba debe representar lo que hace la prueba sin tener que revisar la prueba en si.

## ESTRUCTURA DE LAS PU

**Arrange** 🡺 Donde se inicializan los objetos necesarios para la ejecución de Código.

**Act**🡺 Donde se ejecuta el método que se va a probar y se obtiene el resultado.

**Assert** 🡺 Donde se comprueba que el resultado obtenido es el esperado.

Instalar **Microsoft.Extensions.configuration.Abstractions** versión 5.0

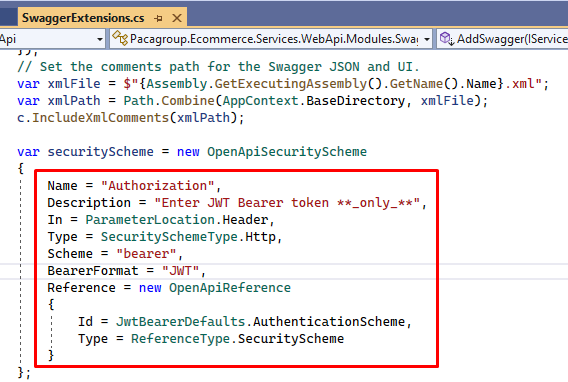
**Microsoft.Extensions.DependencyInjection.Abstractions** versión 5.0

## COBERTURA DE CODIGO

Se cuenta en bloques. Un bloque es un fragmento de Código con un punto de entrada y de salida exactamente.

# [**ACTUALIZACION**](#_Toc134110840) **A NET CORE 6.0**

Se debe actualizar swagger extensión.



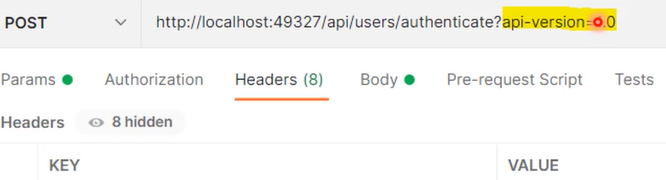
## VERSIONAR API

**Api** 🡺 Promesa de realizar los servicios cuando se les solicite de manera especifica.

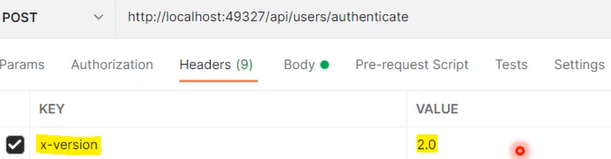


### Control de versiones usando parámetros de Cadena (Query String)

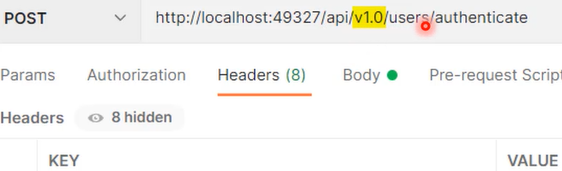
pac



### Encabezados personalizados

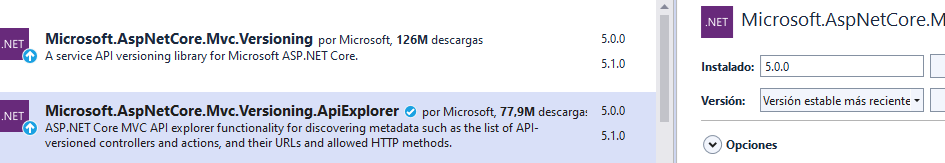


### Parámetros en la URI (Path)

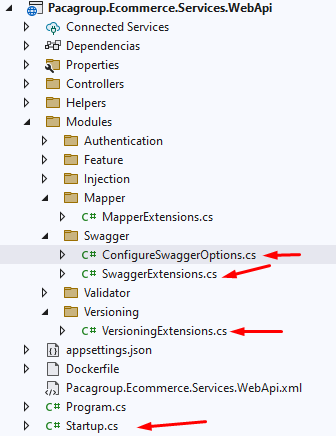


### Versionando API usando parámetros de consulta

Install-Package Microsoft.AspNetCore.Mvc.Versioning -Version 5.0.0

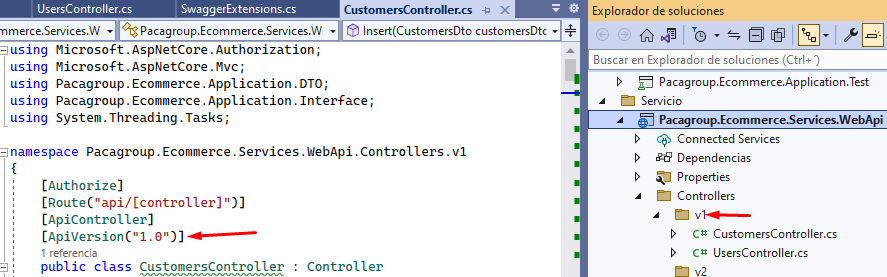


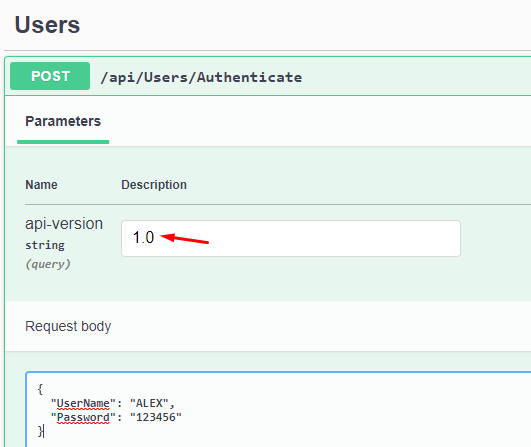


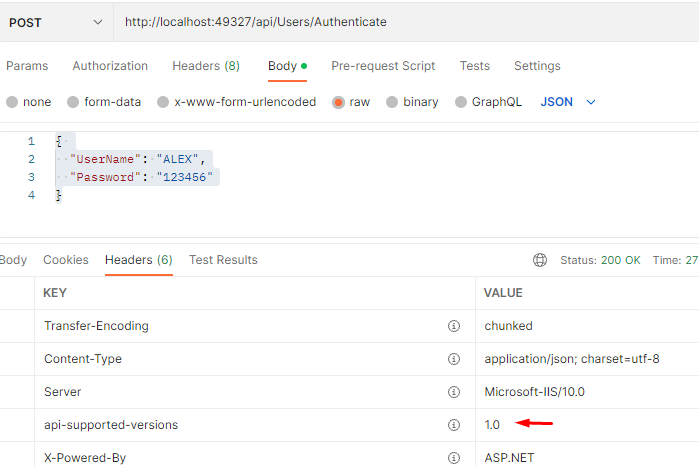


### Versionando Controladores

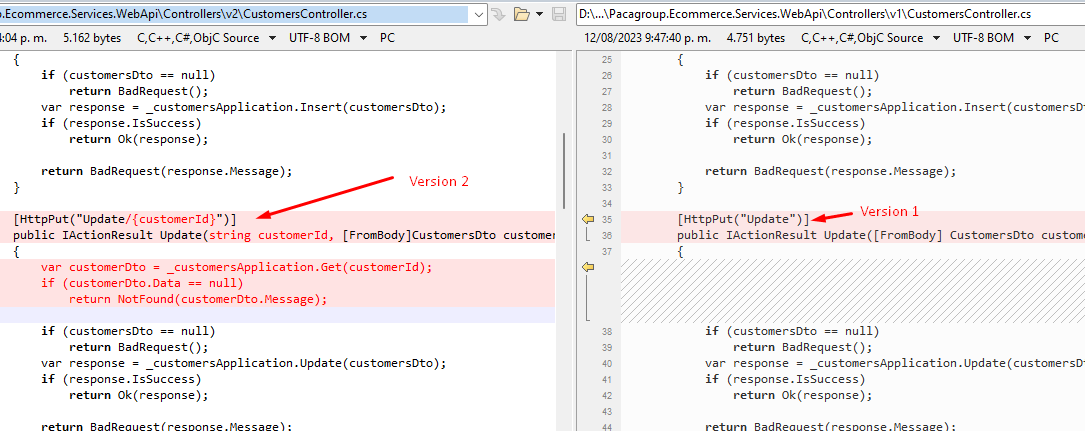
Se crea la carpeta v1 y se mueven los controladores, luego se coloca. v1 en los **namespaces** y se especifica [ApiVersion("1.0")].

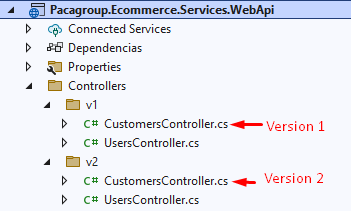


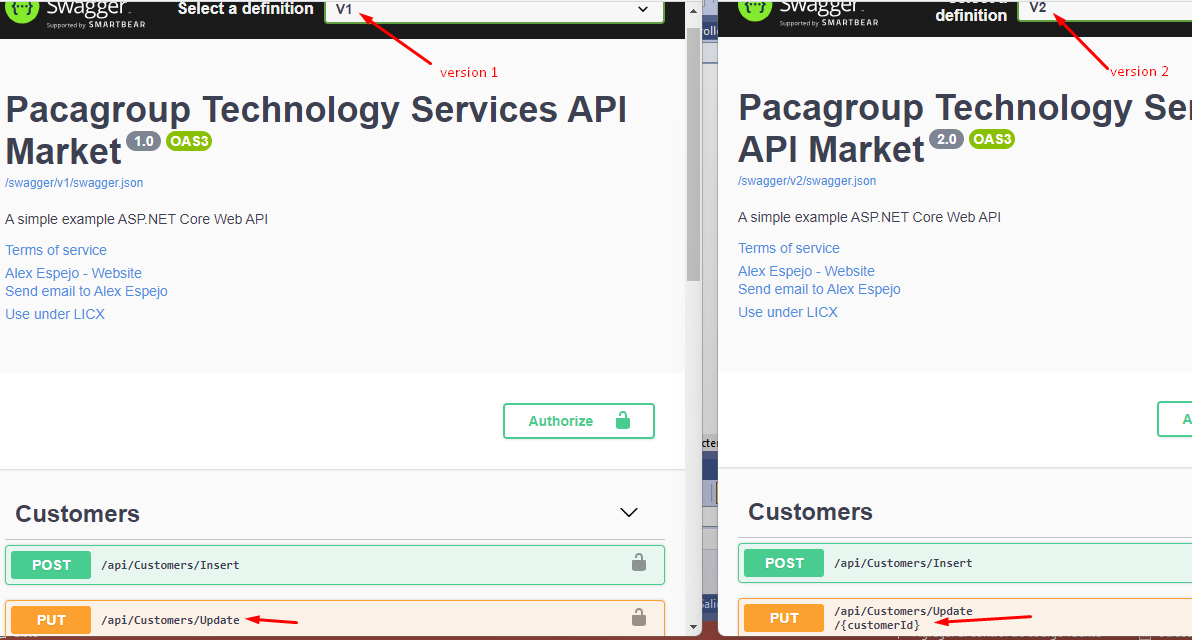




### Se prueba la Versión 2

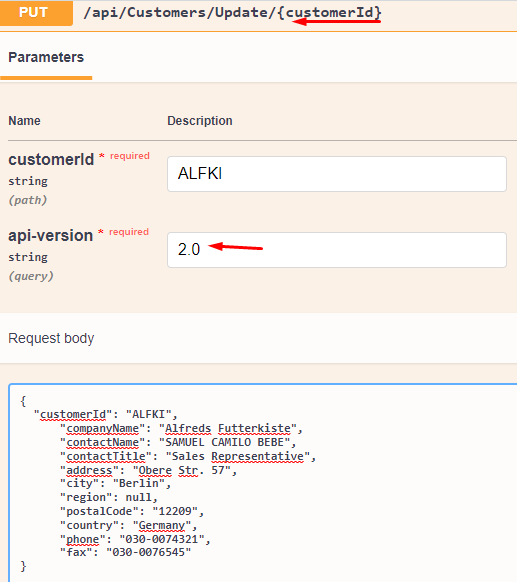






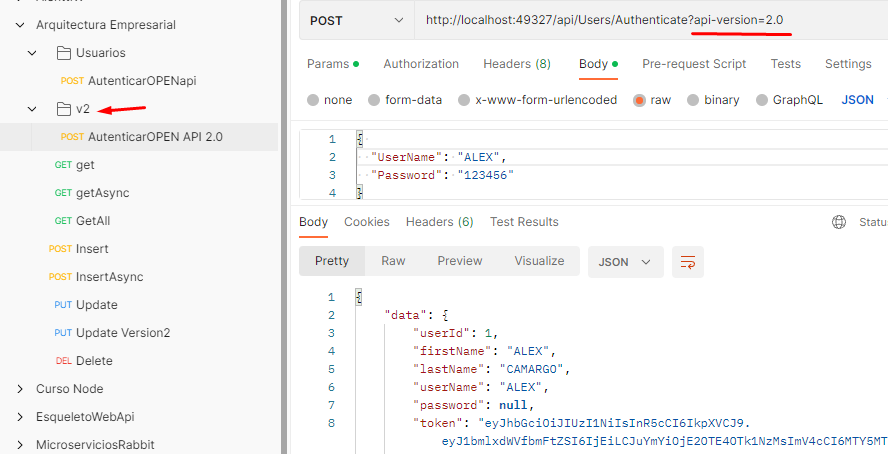
Me autentico con versión 2





En postman se debe adicionar ?api-version=2.0

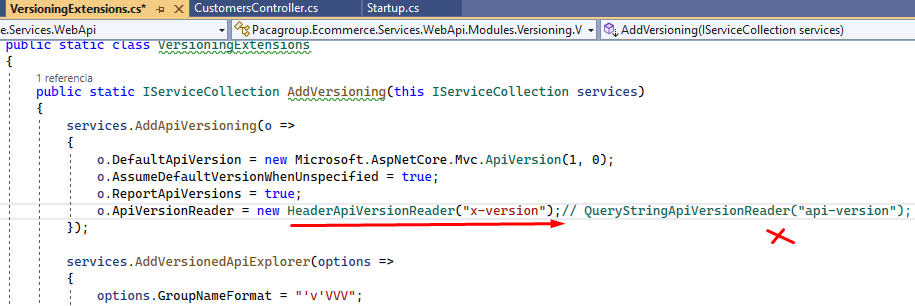
http://localhost:49327/api/Users/Authenticate?api-version=2.0



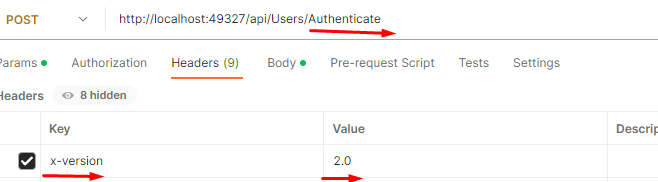
# CONTROL DE VERIONES

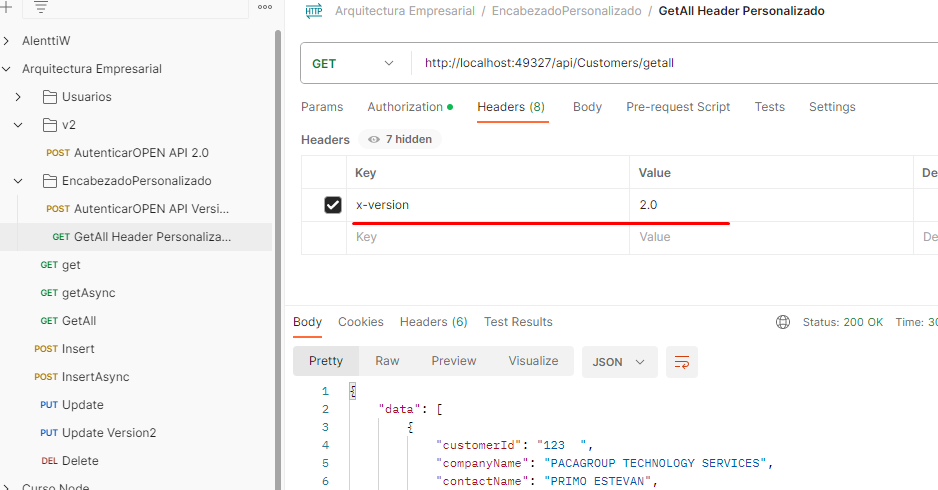
## ENCABEZADOS PERSONALISADOS CONTROL DE VERSIONES

Se cambia el parámetro en la clase VersioningExtensions

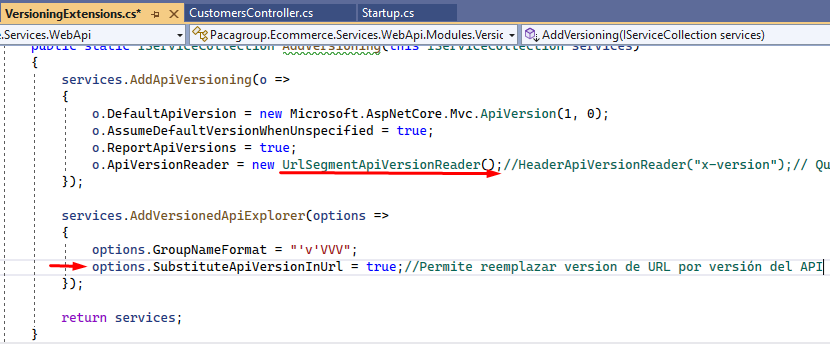


En header se cambia se especifica la versión

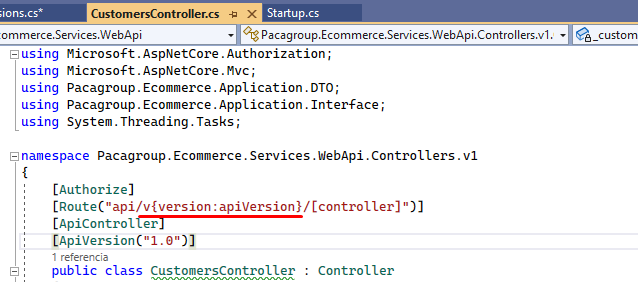


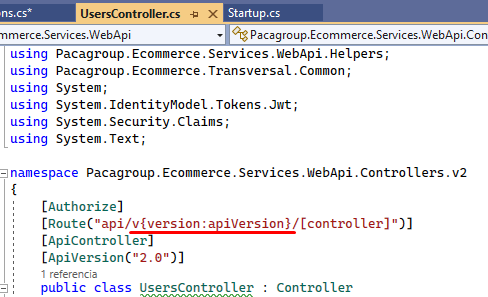


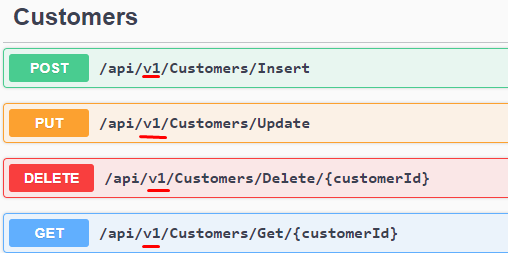
## PARAMETROS DE SEGMENTO CONTROL DE VERSIONES

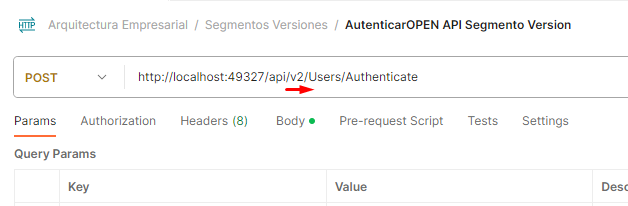


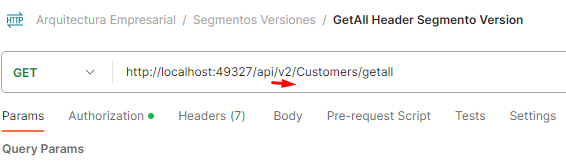
## Se actualiza encabezado de Controladores





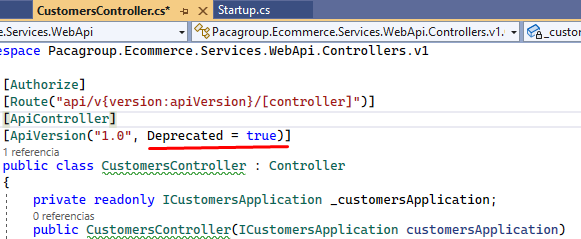






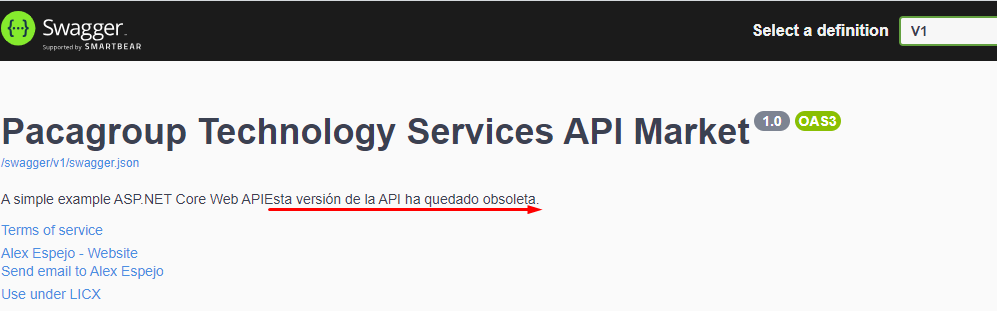
## DEPRECANDO CONTROL DE VERSIONES

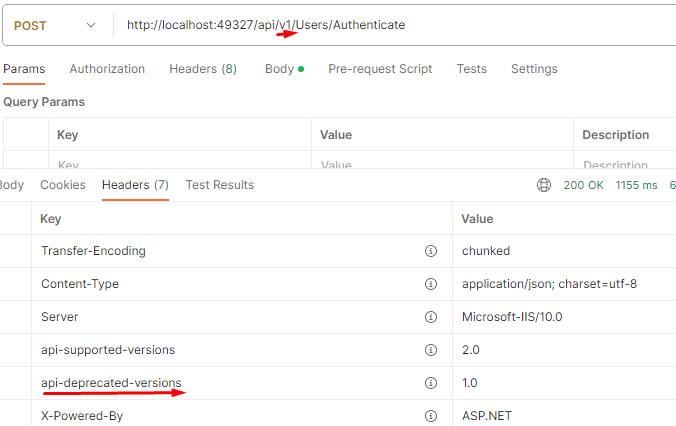
Se adiciona Deprecated = true)]

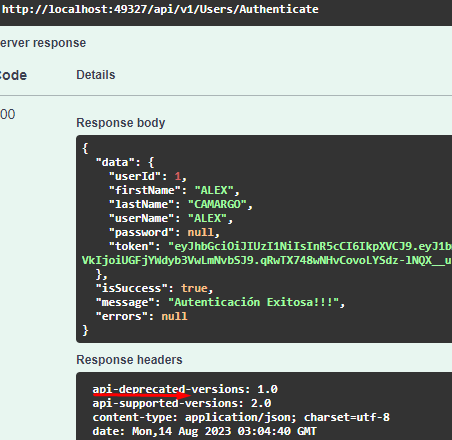


Se adiciona la condición siguiente para indicar que la versión esta deprecada, en la clase ConfigureSwaggerOptions









# **PATRON HEALTH CHECK**

Permite comprobar el estado general y disponibilidad de la infraestructura, se presentan como **endpoint** http y si la app puede comunicarse con el proveedor de la base de datos.

* Componente principal es la Base de Datos.

Proporciona 3 niveles de Health Check:

* **Healthy (Saludable)** 🡺 Esta en un estado de funcionamiento normal.
* **Degraded (Degradado)** 🡺 Esta no responde dentro de un tiempo esperado.
* **UnHealthy (En mal Estado)** 🡺 No está en buen estado y está fuera de línea o se lanzó una excepción no controlada.

Hay 2 tipos de **HEALTH CHECK**

## Librerías

AspNetCore.healthchecks.sqlserver

AspNetCore.healthchecks.UI

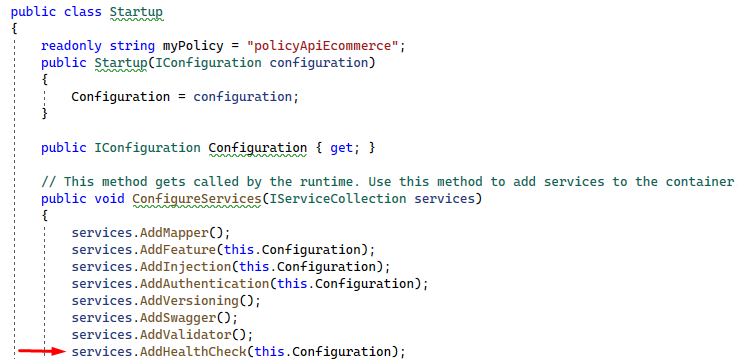
AspNetCore.healthchecks.UI.Client

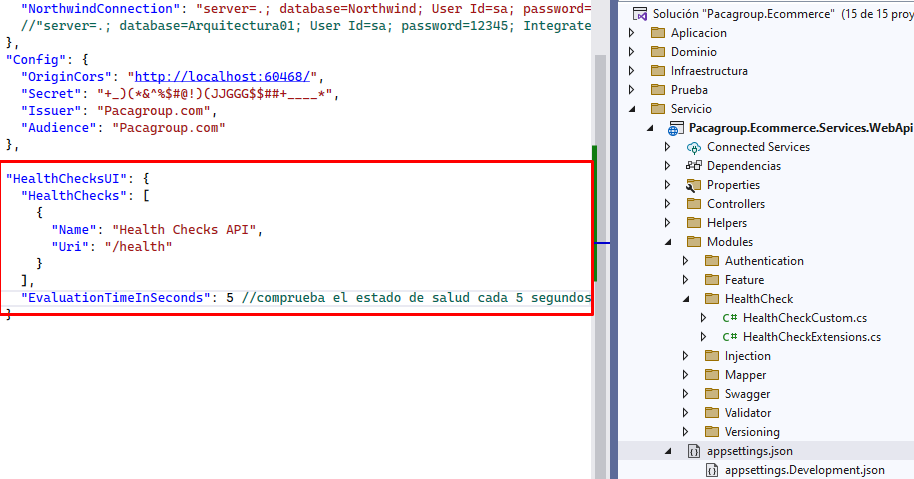
AspNetCore.healthchecks.UI.InMemory.Storage

## Configurar Health Check



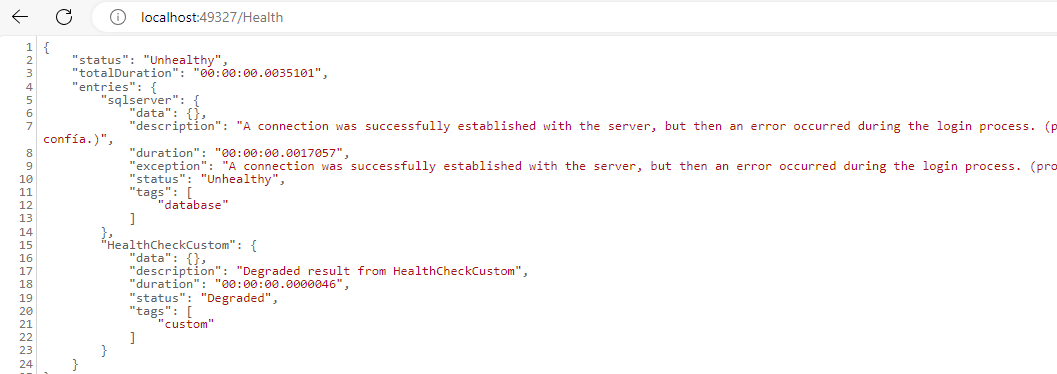






[localhost:49327/Health](http://localhost:49327/Health)

<http://localhost:49327/Healthchecks-ui#/healthchecks> 🡺 Modo Grafico



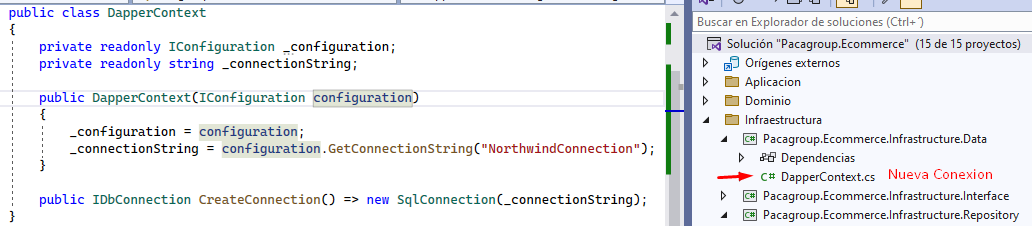
# PATRON REPOSITORY

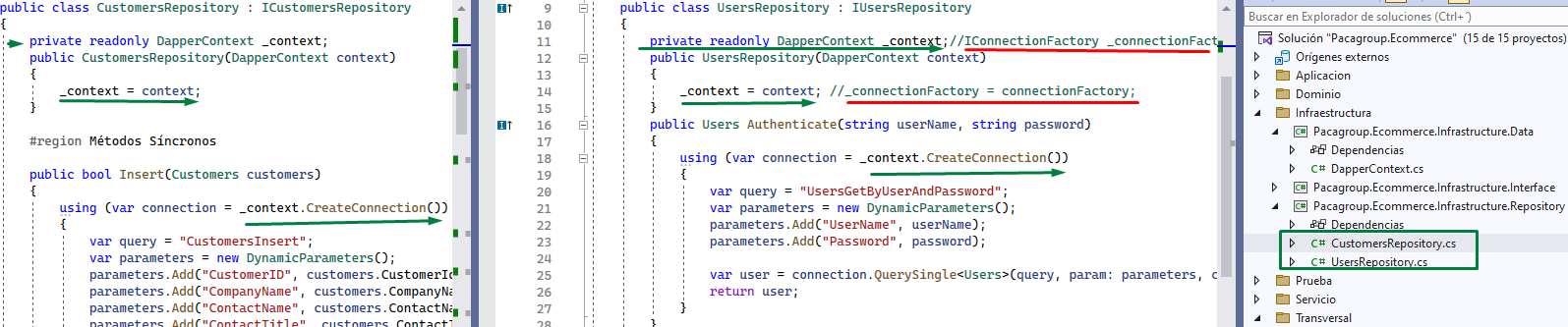
Se utiliza para separar la lógica empresarial de la lógica de acceso a datos.

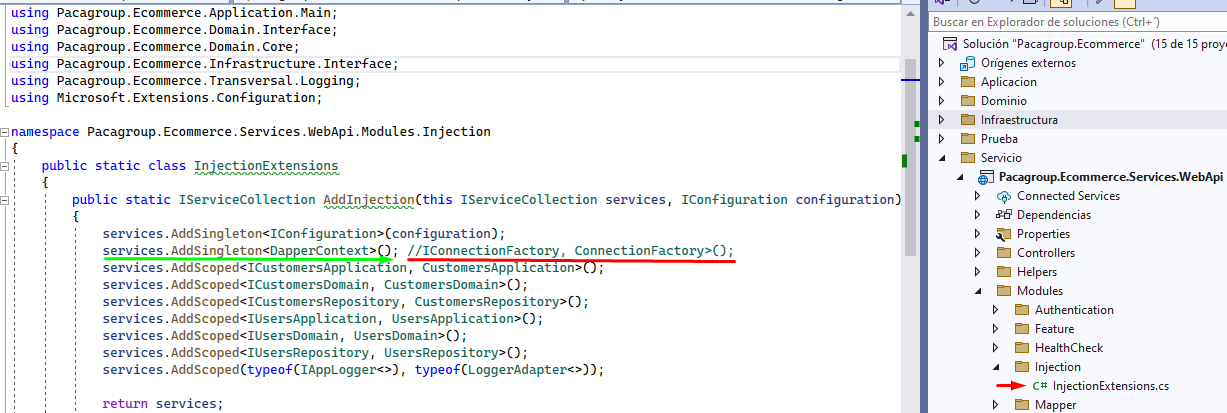
Permitiendo que la lógica empresarial sea independiente de la forma en que los datos están almacenados, consultados y manipulados.

Facilita las pruebas Unitarias automatizadas o el desarrollo basado en pruebas TDD(Test-Driven Development (Desarrollo basado en Pruebas)).

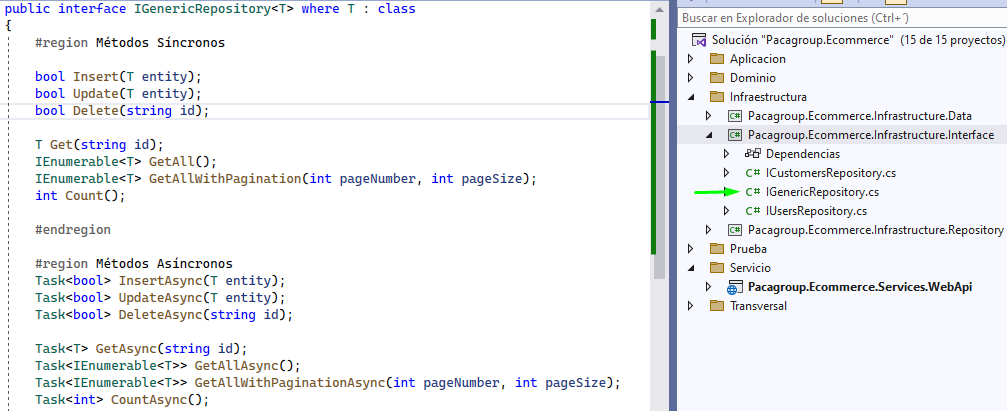
## Remover la Conexión Antigua y Reemplazar por Nueva



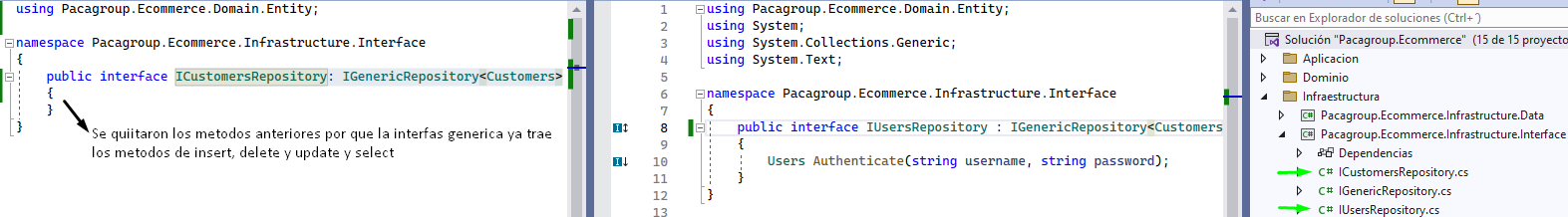




## **Implementar Interfaz Repositorio Genérica**



## Heredar de la Interfaz Genérica



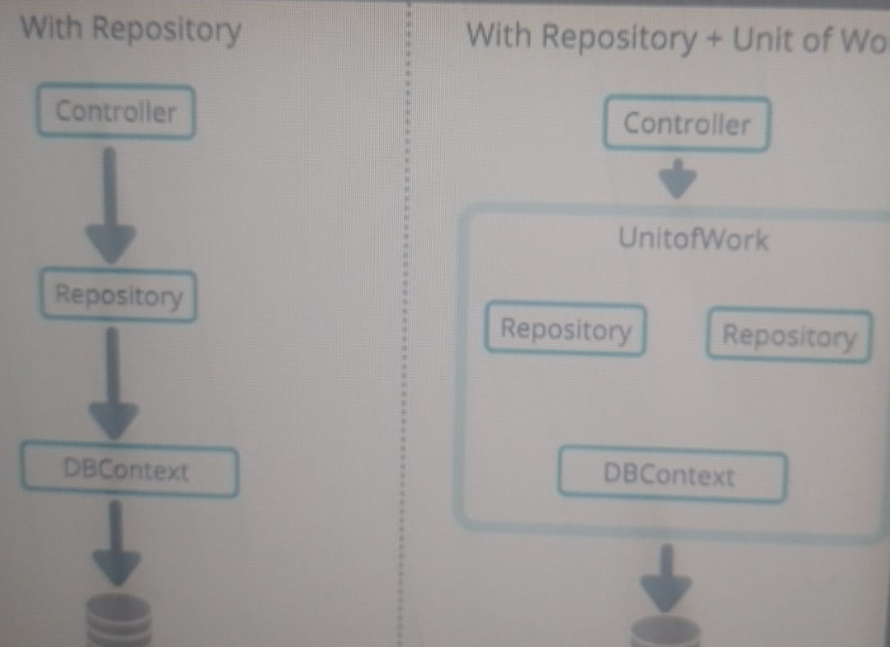
# **PATRON UNIT OF WORK**

Trata como Unidad los objetos nuevos y modificados con respecto a una fuente de Datos.

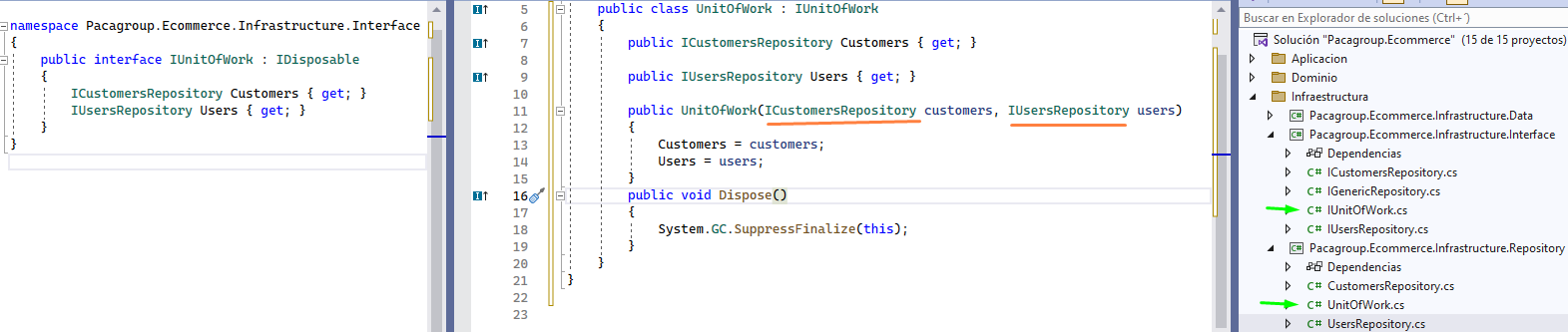
Permite centralizar las fuentes de datos.

Persiste conjunto de acciones evitando el exceso de conexiones en la misma.

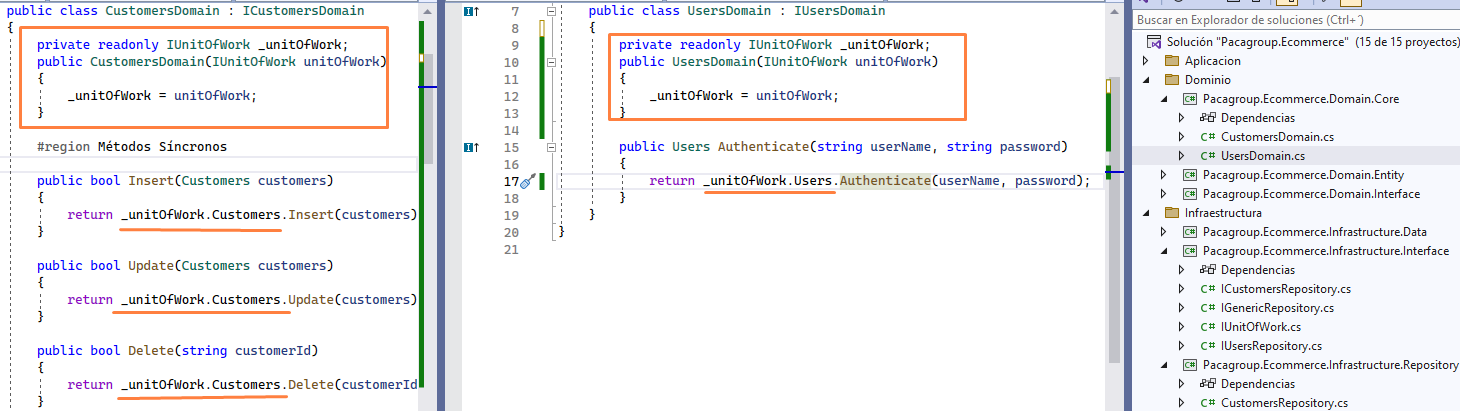
**Interfas Idisposable** 🡺 Permite librear Recursos en memoria.



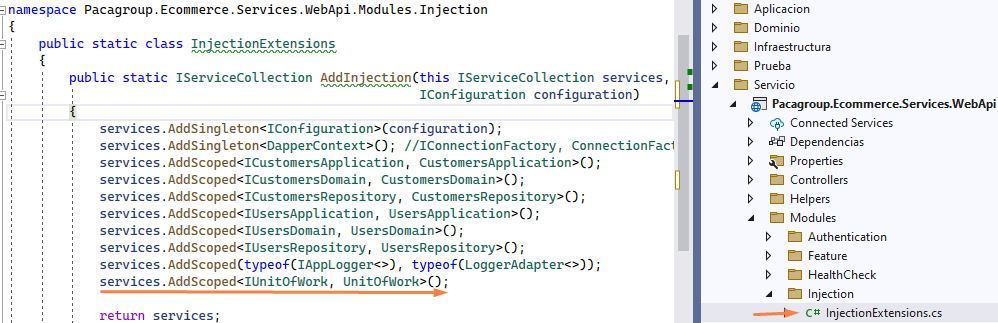
## Declarar interface e Implementación



## Instanciar IUnitOf Work y usar abstracción



## Inyectar dependencias



Voy Minuto 0

**Voy minuto 20**

# **ARQUITECTURA LIMPIA**

USUARIO

presentación

Infraestructura

persistencia

APLICACIÓN

BASE DE DATOS

Dominio

Capa dominio 🡺 Contiene todas las entidades, enumeraciones, excepciones, tipos y lógica específicos en la capa de dominio.

aplicación 🡺 Contiene toda la lógica e la aplicación, DTOs, mapeadores.

Depende de la capa de dominio, pero no depende de ninguna otra capa. Esta capa define interfaces que son implementadas por capas externas.

Infraestructura 🡺 Contiene clases para acceder a recursos externos como base de datos, sistema de archivos, servicios web, smtp , mensajería, colas etc. Estas clases deben basarse en interfaces definidas dentro de la capa de aplicación.

Reglas de la dependencia🡺 Solo pueden apuntar hacia adentro.