

Configuración del entorno de desarrollo.....	3
Configuración en Linux.....	4
Configuración en Windows.....	6
1. Programación Basica	7
1.1. Creación de Web Apis Con .Net	7
1.1.1. Entity Framework.....	7
1.1.2. Protocolo HTTP	8
1.1.2.1. Verbos HTTP	9
1.1.3. Que es una Web Api.....	10
1.1.3.1. Códigos de status	11
2. Programación Intermedia	11
2.1. Usando Docker Con Mysql	11
2.1.1. Instalación	12
2.1.2. Configuración de Imagen Mysql en docker	14
3. Programación Avanzada	16

3.1. Proyecto Base (Arquitectura de la aplicación).....	16
3.2. Proyecto Base (Entidades)	21
3.3. Relaciones entre Entidades.....	29
3.4. Proyecto Base (Habilitando Migraciones).....	53
3.5. Proyecto Base (Fluent Api).....	58
3.5.1. Entity Mappings	61
3.5.1.1. Mapeando entidades a Tablas	62
3.5.1.2. Mapeando multiples Tablas.....	64
3.5.1.3 Mapeo de propiedades usando Fluent API.....	65
3.5.2. Relaciones	67
3.5.2.1. One to One	67
3.6. Repositorios, Unidad de trabajo y métodos de extensión	68
3.6.1. Repositorios	68
3.6.2. Unidades de trabajo.....	68
3.6.3 Reutilización de Código.....	69
3.6.4. Métodos de extensión	70
3.7 Mapeo de clases.....	71
3.7.1. DTO.....	72
3.7.1.1. Configuración del entorno y librerías	73
3.7. Buenas prácticas Desarrollo Backend	75
3.7.1. Rate Limiting.	75
3.7.1.1. Implementación Rate Limiting.....	76
3.7.2. Versionado Apis con Query String o Encabezado	80
3.7.3. Versionado desde el Header	86
3.7.4. Paginación en Net Core.....	89
3.8 JWT	96
3.8.1 Estructura JWT	97
Header.....	97
Signature	98
3.8.1.1. Autenticación en NetCore.....	98
3.9. Bitácoras o Logs	112
3.9.1. Personalización de Mensajes de Excepciones	115

Configuración del entorno de desarrollo

Para desarrollar en .NET Core, necesitarás cumplir con los siguientes requisitos:

Sistema operativo compatible: .NET Core es compatible con Windows, macOS y Linux. Asegúrate de tener un sistema operativo compatible instalado en tu máquina.

SDK de .NET Core: Debes descargar e instalar el SDK (Software Development Kit) de .NET Core correspondiente a tu sistema operativo desde el sitio web oficial de .NET Core. El SDK incluye las herramientas necesarias para desarrollar aplicaciones con .NET Core.

Entorno de desarrollo integrado (IDE): Aunque no es estrictamente necesario, se recomienda utilizar un IDE para facilitar el desarrollo en .NET Core. Microsoft Visual Studio es el IDE principal para .NET Core y ofrece características avanzadas para la programación en C# y otros lenguajes de .NET. También puedes utilizar Visual Studio Code, un editor de código ligero y altamente personalizable, que también es compatible con .NET Core.

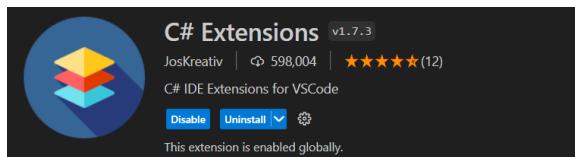
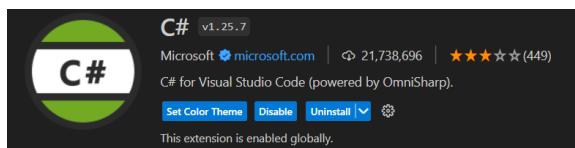
Conocimientos de programación: Para desarrollar en .NET Core, es necesario tener conocimientos de programación en C# u otros lenguajes compatibles con .NET Core, como F# o Visual Basic. Familiarizarte con los conceptos de programación orientada a objetos y los principios básicos de .NET Framework también es útil.

Control de versiones: Es recomendable utilizar un sistema de control de versiones, como Git, para mantener un registro de los cambios realizados en tu proyecto y facilitar la colaboración con otros desarrolladores.

Estos son los requisitos básicos para comenzar a desarrollar en .NET Core. A medida que te familiarices con el entorno y el desarrollo en .NET Core, también podrías necesitar aprender sobre las bibliotecas y frameworks adicionales que se utilizan comúnmente en el desarrollo de aplicaciones, como ASP.NET Core para el desarrollo web o Entity Framework Core para el acceso a bases de datos.

Software necesario

- Visual Studio Code
- Extensiones de visual Studio Code :

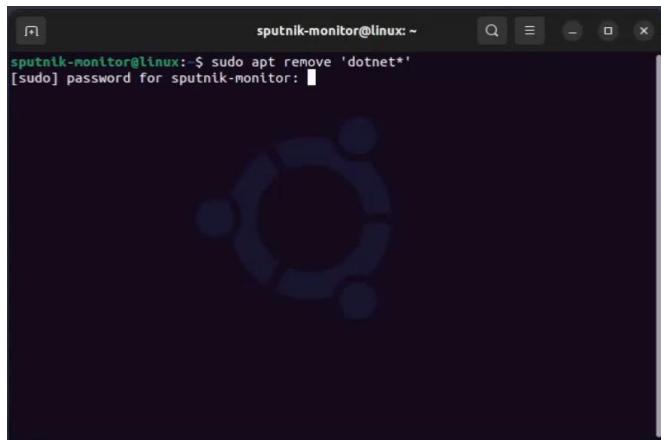


- SDK Net Core V 6.0 o 7.0
- En sistemas operativos se puede usar visual studio community edition.

Configuración en Linux



- Abrir la terminal de linux **Ctrl+Alt+T**
- Ingresar el comando **sudo apt remove 'dotnet*' para eliminar los paquetes de net core que se encuentren instalados.**

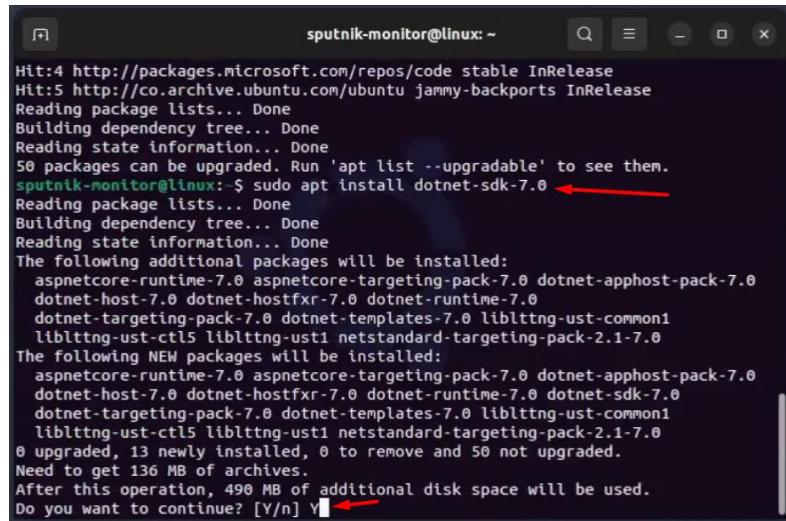


En caso de no contar con paquetes instalados se podrá visualizar el siguiente resultado:

```
Note, selecting 'dotnet-apphost-pack-7.0' for glob 'dotnet*'
Note, selecting 'dotnet-sdk-7.0-source-built-artifacts' for glob 'dotnet*'
Note, selecting 'dotnet-runtime-6.0' for glob 'dotnet*'
Note, selecting 'dotnet-runtime-7.0' for glob 'dotnet*'
Package 'dotnet-apphost-pack-6.0' is not installed, so not removed
Package 'dotnet-apphost-pack-7.0' is not installed, so not removed
Package 'dotnet-host' is not installed, so not removed
Package 'dotnet-host-7.0' is not installed, so not removed
Package 'dotnet-hostfxr-6.0' is not installed, so not removed
Package 'dotnet-hostfxr-7.0' is not installed, so not removed
Package 'dotnet-runtime-6.0' is not installed, so not removed
Package 'dotnet-runtime-7.0' is not installed, so not removed
Package 'dotnet-sdk-6.0' is not installed, so not removed
Package 'dotnet-sdk-6.0-source-built-artifacts' is not installed, so not removed
Package 'dotnet-sdk-7.0' is not installed, so not removed
Package 'dotnet-sdk-7.0-source-built-artifacts' is not installed, so not removed
Package 'dotnet-targeting-pack-6.0' is not installed, so not removed
Package 'dotnet-targeting-pack-7.0' is not installed, so not removed
Package 'dotnet-templates-6.0' is not installed, so not removed
Package 'dotnet-templates-7.0' is not installed, so not removed
Package 'dotnet' is not installed, so not removed
Package 'dotnet7' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 50 not upgraded.
sputnik-monitor@linux: ~
```

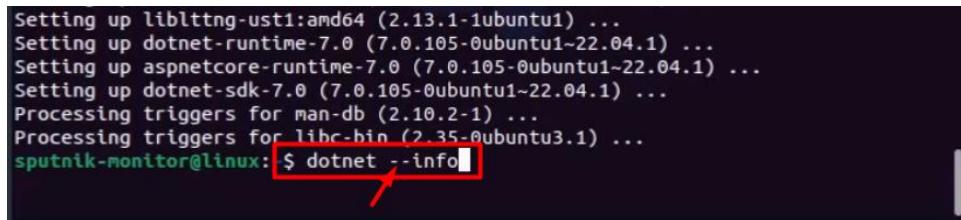
- Ingresar el comando **apt remove 'aspnetcore*' para remover paquetes y servicios runtime de aspnetcore.**
- Ejecutar el comando **sudo rm /etc/apt/sources.list.d/microsoft-prod.list** para eliminar repositorios en caso de haber instalado otras versiones de Net Core.
- Ejecutar el comando **sudo apt update** para realizar actualización de paquetes en linux.

- Ejecutamos el comando **sudo apt install dotnet-sdk-6.0** si deseamos instalar la versión 6 de .Net Core o el comando **sudo apt install dotnet-sdk-7.0** si se desea instalar la versión 7.0



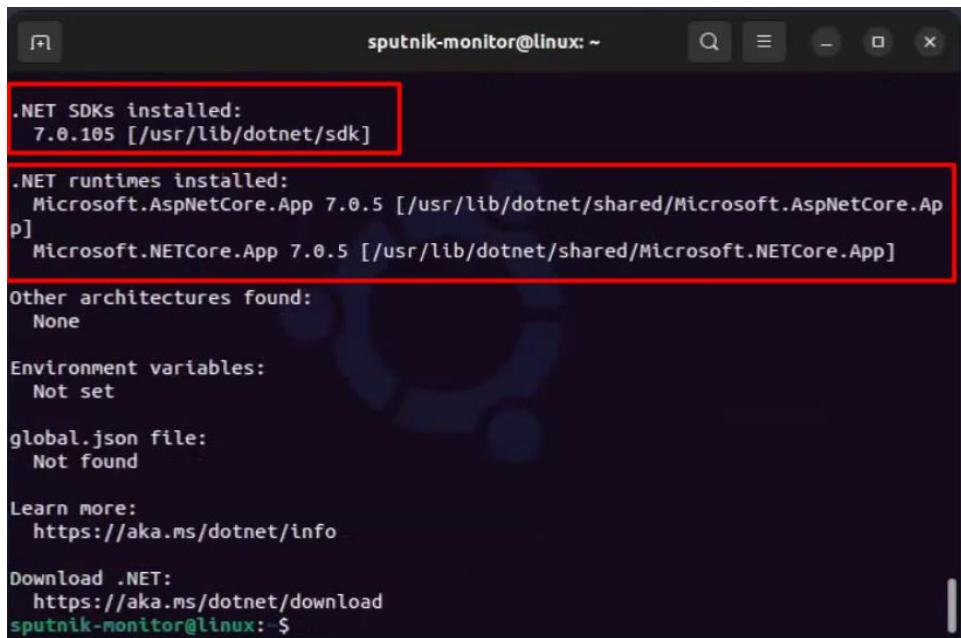
```
sputnik-monitor@linux: ~
Hit:4 http://packages.microsoft.com/repos/code stable InRelease
Hit:5 http://co.archive.ubuntu.com/ubuntu jammy-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
50 packages can be upgraded. Run 'apt list --upgradable' to see them.
sputnik-monitor@linux: $ sudo apt install dotnet-sdk-7.0
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  aspnetcore-runtime-7.0 aspnetcore-targeting-pack-7.0 dotnet-apphost-pack-7.0
  dotnet-host-7.0 dotnet-hostfxr-7.0 dotnet-runtime-7.0
  dotnet-targeting-pack-7.0 dotnet-templates-7.0 libliltng-ust-common1
  libliltng-ust-ctls libliltng-ust1 netstandard-targeting-pack-2.1-7.0
The following NEW packages will be installed:
  aspnetcore-runtime-7.0 aspnetcore-targeting-pack-7.0 dotnet-apphost-pack-7.0
  dotnet-host-7.0 dotnet-hostfxr-7.0 dotnet-runtime-7.0 dotnet-sdk-7.0
  dotnet-targeting-pack-7.0 dotnet-templates-7.0 libliltng-ust-common1
  libliltng-ust-ctls libliltng-ust1 netstandard-targeting-pack-2.1-7.0
0 upgraded, 13 newly installed, 0 to remove and 50 not upgraded.
Need to get 136 MB of archives.
After this operation, 490 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

- Como último paso tendremos que verificar la versión del SDK instalada, para esto se ingresa el comando **dotnet --info**



```
Setting up libliltng-ust1:amd64 (2.13.1-1ubuntu1) ...
Setting up dotnet-runtime-7.0 (7.0.105-0ubuntu1~22.04.1) ...
Setting up aspnetcore-runtime-7.0 (7.0.105-0ubuntu1~22.04.1) ...
Setting up dotnet-sdk-7.0 (7.0.105-0ubuntu1~22.04.1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
sputnik-monitor@linux: $ dotnet --info
```

Como resultado final obtenemos la siguiente información:



```
.NET SDKs installed:
  7.0.105 [/usr/lib/dotnet/sdk]

.NET runtimes installed:
  Microsoft.AspNetCore.App 7.0.5 [/usr/lib/dotnet/shared/Microsoft.AspNetCore.Ap
p]
  Microsoft.NETCore.App 7.0.5 [/usr/lib/dotnet/shared/Microsoft.NETCore.App]

other architectures found:
  None

Environment variables:
  Not set

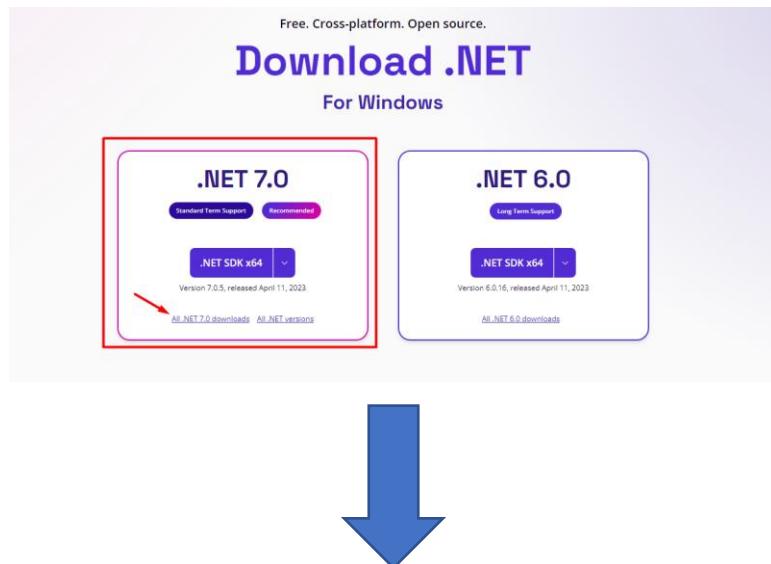
global.json file:
  Not found

Learn more:
  https://aka.ms/dotnet/info

Download .NET:
  https://aka.ms/dotnet/download
sputnik-monitor@linux: $
```

Configuración en Windows

- Ingresar a la Url <https://dotnet.microsoft.com/en-us/download>



This screenshot shows the 'Download .NET 7.0' page. It includes a header note about .NET 8 Preview and a 'Get .NET 8 Preview' button. The main content is the 'SDK 7.0.302' section, which provides download links for different OSes and architectures. The Windows row is highlighted with a red box and a red arrow pointing to the 'Arm64 | x64 | x86 | winget instructions' link.

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 Arm32 Alpine Arm64 Arm64 Alpine x64 x64 Alpine
macOS	Arm64 x64	Arm64 x64
Windows	Arm64 x64 x86 winget instructions	Arm64 x64 x86
All	dotnet-install scripts	

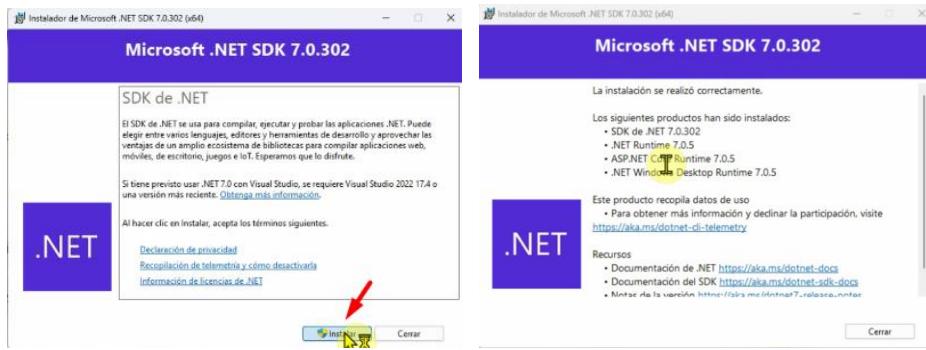
OS	Installers	Binaries
Linux	Package manager instructions	Arm32 Arm32 Alpine Arm64 Arm64 Alpine x64 x64 Alpine
macOS		Arm64 x64
Windows	Hosting Bundle x64 x86 winget instructions	Arm64 x64 x86

- Descargar el SDK 7.0 teniendo en cuenta la arquitectura del sistema operativo ya sea 32 (x86) o 64(x64) bits.

SDK 7.0.302

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 Arm32 Alpine Arm64 Arm64 Alpine x64 x64 Alpine
macOS	Arm64 x64	Arm64 x64
Windows	Arm64 x64 x86 winget instructions	Arm64 x64 x86
All	dotnet-install scripts	

- Ejecutar el asistente de instalación y seguir los pasos indicados por el asistente.



1. Programación Basica

1.1. Creación de Web Apis Con .Net

1.1.1. Entity Framework

Entity Framework (EF) es un framework de mapeo objeto-relacional (ORM) desarrollado por Microsoft. Proporciona una forma conveniente de trabajar con datos en una base de datos utilizando objetos y consultas en lugar de escribir sentencias SQL directamente.

Aquí hay algunos conceptos clave para entender Entity Framework:

- **Modelado de datos:** En Entity Framework, el modelado de datos se realiza mediante clases que representan entidades en tu base de datos. Estas clases se conocen como "entidades" y mapean a tablas en la base de datos. Cada propiedad de una entidad representa una columna en la tabla.
- **Contexto de base de datos:** El DbContext es una clase central en Entity Framework. Representa la sesión de trabajo con la base de datos y se utiliza para consultar, insertar, actualizar y eliminar entidades. El DbContext también se encarga de establecer la conexión con la base de datos y realizar cambios en la misma.
- **Migraciones:** Entity Framework proporciona migraciones para administrar los cambios en el esquema de la base de datos a lo largo del tiempo. Puedes crear migraciones para agregar, modificar o eliminar tablas, columnas y restricciones en tu base de datos sin perder los datos existentes.
- **Consultas LINQ:** Entity Framework permite realizar consultas utilizando Language Integrated Query (LINQ). LINQ proporciona una forma elegante y legible de escribir consultas y manipular datos utilizando sintaxis similar a SQL dentro de tu código C# o Visual Basic.
- **Carga diferida y explícita:** Entity Framework utiliza la técnica de carga diferida para cargar los datos de la base de datos cuando sea necesario. Esto significa que los datos no se cargarán automáticamente cuando se consulte una entidad, sino que se cargarán bajo demanda. También puedes utilizar la carga explícita para especificar qué propiedades de una entidad deseas cargar de inmediato.
- **Relaciones y navegación:** Entity Framework admite relaciones entre entidades, como uno a uno, uno a muchos y muchos a muchos. Puedes establecer estas relaciones en tu modelo de datos y luego navegar entre las entidades utilizando propiedades de navegación.

- Control de transacciones: Entity Framework permite trabajar con transacciones para asegurar que las operaciones de base de datos se realicen de manera segura y consistente. Puedes iniciar y confirmar transacciones utilizando el DbContext.

Entity Framework tiene varias versiones, incluyendo Entity Framework Core, que es una versión más ligera y multiplataforma que se utiliza comúnmente en aplicaciones modernas.

Para comenzar a utilizar Entity Framework, debes agregar el paquete NuGet correspondiente a tu proyecto y configurar el DbContext con la cadena de conexión de tu base de datos. Luego, puedes definir tus entidades, realizar consultas LINQ y realizar operaciones CRUD (crear, leer, actualizar, eliminar) en la base de datos.

1.1.2. Protocolo HTTP

El Protocolo de Transferencia de Hipertexto (HTTP, por sus siglas en inglés) es un protocolo de comunicación utilizado en la World Wide Web para la transferencia de datos. HTTP define la forma en que los clientes y servidores se comunican entre sí, permitiendo la solicitud y respuesta de recursos, como páginas web, imágenes, archivos, etc.

Aquí podrá encontrar algunas características clave sobre el protocolo HTTP:

- **Cliente y servidor:** En el contexto de HTTP, el cliente es el software que envía solicitudes HTTP, generalmente un navegador web, y el servidor es el software que recibe y procesa esas solicitudes, generalmente un servidor web.
- **Métodos de solicitud:** HTTP define varios métodos o verbos que indican la acción que se debe realizar en el recurso solicitado. Algunos de los métodos más comunes son:
 - **GET:** Sigue la recuperación de un recurso.
 - **POST:** Envía datos al servidor para ser procesados, como enviar un formulario HTML.
 - **PUT:** Crea o actualiza un recurso específico con los datos proporcionados.
 - **DELETE:** Elimina un recurso específico.
- **URL:** Uniform Resource Locator (URL) es una cadena que identifica de manera única un recurso en la web. Una URL típica tiene el siguiente formato: **protocolo://dominio/ruta**. Por ejemplo, <http://www.ejemplo.com/pagina.html> es una URL que apunta a un archivo llamado "pagina.html" en el dominio "www.ejemplo.com" utilizando el protocolo HTTP.
- **Cabeceras HTTP:** Las solicitudes y respuestas HTTP incluyen cabeceras que proporcionan información adicional sobre la comunicación. Las cabeceras pueden contener detalles como el tipo de contenido, las cookies, la codificación, las fechas, entre otros.
- **Códigos de estado:** Las respuestas HTTP incluyen un código de estado que indica el resultado de la solicitud. Algunos códigos de estado comunes son:

200 OK: La solicitud ha sido exitosa.

404 Not Found: El recurso solicitado no ha sido encontrado en el servidor.

500 Internal Server Error: Se produjo un error interno en el servidor.

- **Seguridad:** HTTP es un protocolo no seguro, lo que significa que los datos transmitidos no están encriptados. Para una comunicación segura, se utiliza HTTPS (HTTP Seguro), que utiliza el protocolo SSL/TLS para cifrar los datos.

HTTP es un protocolo ampliamente utilizado en la web y es fundamental para el intercambio de información entre clientes y servidores. Además de las características básicas mencionadas anteriormente, HTTP también admite la negociación de contenido, la compresión, la autenticación y otras funcionalidades más avanzadas.

1.1.2.1. Verbos HTTP

HTTP define varios métodos o verbos que indican la acción que se debe realizar en el recurso solicitado. A continuación, se muestran los verbos HTTP más comunes:

- **GET:** Se utiliza para recuperar un recurso del servidor. Al enviar una solicitud GET, el cliente solicita al servidor que devuelva el contenido de un recurso específico. Por ejemplo, al abrir una página web en un navegador, se realiza una solicitud GET para obtener el contenido HTML de esa página.
- **POST:** Se utiliza para enviar datos al servidor para ser procesados. Por lo general, se utiliza para enviar formularios HTML o enviar datos al servidor para crear un nuevo recurso. Los datos enviados en una solicitud POST se incluyen en el cuerpo de la solicitud. Por ejemplo, al enviar un formulario de registro en un sitio web, los datos del formulario se envían al servidor mediante una solicitud POST.
- **PUT:** Se utiliza para crear o actualizar un recurso específico con los datos proporcionados. Al enviar una solicitud PUT, el cliente envía los datos al servidor para crear un nuevo recurso o reemplazar completamente un recurso existente. El URI de la solicitud PUT suele apuntar al recurso específico que se desea crear o actualizar.
- **DELETE:** Se utiliza para eliminar un recurso específico del servidor. Al enviar una solicitud DELETE, el cliente solicita al servidor que elimine el recurso indicado por el URI de la solicitud. Después de una solicitud DELETE exitosa, el recurso se elimina del servidor.
- **PATCH:** Se utiliza para realizar modificaciones parciales en un recurso. A diferencia de la solicitud PUT que reemplaza completamente un recurso, la solicitud PATCH se utiliza para enviar cambios específicos a un recurso existente. Solo los campos proporcionados en la solicitud PATCH se actualizan en el recurso.
- **HEAD:** Es similar a una solicitud GET, pero en lugar de recuperar el contenido del recurso, la solicitud HEAD solicita solo los metadatos del recurso, como las cabeceras HTTP y la información de estado. Es útil para verificar la existencia y obtener información sobre un recurso sin descargar todo su contenido.

Estos son solo algunos de los verbos HTTP más comunes. HTTP también admite otros verbos menos utilizados, como OPTIONS para obtener información sobre las capacidades del servidor, TRACE para rastrear una solicitud a través de los servidores y CONNECT para establecer una conexión de túnel a través de un proxy.

1.1.3. Que es una Web Api

Una Web API (Application Programming Interface) es una interfaz de programación de aplicaciones que permite a diferentes aplicaciones y sistemas interactuar entre sí a través del protocolo HTTP. Proporciona un conjunto de puntos finales (endpoints) y métodos que permiten a los clientes realizar operaciones y acceder a recursos en un servidor.

A diferencia de una interfaz de usuario (UI) que permite la interacción entre humanos y computadoras, una Web API permite la comunicación entre aplicaciones y sistemas informáticos. Una Web API utiliza el protocolo HTTP para recibir solicitudes y enviar respuestas en un formato común, como JSON o XML.

Algunas características de las Web APIs son:

- **Comunicación a través del protocolo HTTP:** Las Web APIs utilizan el protocolo HTTP como el medio de comunicación estándar. Las solicitudes y respuestas se realizan mediante los métodos y códigos de estado definidos por HTTP.
- **Recursos y URIs:** Las Web APIs se basan en el concepto de recursos que se acceden mediante identificadores únicos llamados URIs (Uniform Resource Identifiers). Los clientes pueden acceder a estos recursos utilizando los métodos HTTP correspondientes.
- **Formatos de datos:** Las Web APIs suelen utilizar formatos de datos como JSON (JavaScript Object Notation) o XML (eXtensible Markup Language) para estructurar la información enviada y recibida. JSON es ampliamente utilizado debido a su simplicidad y fácil lectura por parte de las aplicaciones.
- **Métodos HTTP:** Las Web APIs utilizan los métodos o verbos HTTP, como GET, POST, PUT, DELETE, para indicar la acción que se debe realizar en un recurso. Cada método tiene un propósito específico, como obtener datos, enviar datos o eliminar recursos.
- **Autenticación y autorización:** Las Web APIs suelen utilizar mecanismos de autenticación y autorización para garantizar que solo los clientes autorizados puedan acceder a los recursos. Esto puede implicar el uso de tokens de acceso, claves API u otros métodos de autenticación.

Las Web APIs son ampliamente utilizadas para construir servicios web que permiten la integración de sistemas, el intercambio de datos y la construcción de aplicaciones distribuidas. Al proporcionar una interfaz estandarizada y basada en HTTP, las Web APIs facilitan la interoperabilidad y la comunicación entre diferentes aplicaciones y plataformas.

Las principales empresas y servicios en línea ofrecen Web APIs para permitir que los desarrolladores construyan aplicaciones y servicios que utilicen sus datos o funcionalidades. Estas APIs abiertas proporcionan a los desarrolladores acceso a recursos y datos específicos de manera controlada y segura, lo que fomenta la creación de aplicaciones innovadoras y la integración de servicios.

1.1.3.1. Códigos de status

Los códigos de estado HTTP son números de tres dígitos que se utilizan en las respuestas del servidor para indicar el resultado de una solicitud HTTP. Estos códigos proporcionan información sobre el estado de la solicitud y cómo se ha procesado. Aquí tienes algunos ejemplos de códigos de estado HTTP comunes:

- Códigos de estado informativos (1xx):
 - 100 Continue: El servidor ha recibido la solicitud inicial y el cliente debe continuar con la solicitud.
 - 101 Switching Protocols: El servidor acepta cambiar el protocolo solicitado por el cliente.
- Códigos de estado de éxito (2xx):
 - 200 OK: La solicitud ha sido exitosa.
 - 201 Created: La solicitud ha sido exitosa y se ha creado un nuevo recurso.
 - 204 No Content: La solicitud ha sido exitosa, pero no hay contenido para devolver.
- Códigos de estado de redirección (3xx):
 - 301 Moved Permanently: El recurso solicitado se ha movido permanentemente a una nueva ubicación.
 - 302 Found: El recurso solicitado se ha movido temporalmente a una nueva ubicación.
 - 304 Not Modified: El recurso solicitado no ha sido modificado desde la última vez que se accedió a él.
- Códigos de estado de error del cliente (4xx):
 - 400 Bad Request: La solicitud enviada por el cliente es incorrecta o no se puede procesar.
 - 401 Unauthorized: El cliente no está autenticado y no tiene permiso para acceder al recurso.
 - 403 Forbidden: El cliente no tiene permiso para acceder al recurso solicitado.
- Códigos de estado de error del servidor (5xx):
 - 500 Internal Server Error: Se ha producido un error interno en el servidor.
 - 503 Service Unavailable: El servidor no está disponible actualmente para manejar la solicitud debido a mantenimiento o sobrecarga.

2. Programación Intermedia

2.1. Usando Docker Con Mysql

Docker es una plataforma de código abierto que permite a los desarrolladores y administradores de sistemas crear, empaquetar y distribuir aplicaciones en contenedores. Los contenedores son entornos de ejecución aislados y livianos que encapsulan una aplicación y todas sus dependencias, incluyendo el sistema operativo, bibliotecas y herramientas necesarias para que la aplicación funcione correctamente.

Algunas características clave de Docker son:

- Contenedores: Docker utiliza la tecnología de contenedores para encapsular aplicaciones y todas sus dependencias en un entorno aislado. Cada contenedor se ejecuta como una instancia independiente y portátil, lo que significa que la aplicación funcionará de la misma manera en cualquier entorno compatible con Docker.
- Portabilidad: Los contenedores de Docker son portátiles, lo que facilita la ejecución de aplicaciones en diferentes sistemas y entornos. Puedes desarrollar una aplicación en tu entorno local, empaquetarla en un contenedor de Docker y luego ejecutarla en otros entornos sin preocuparte por las diferencias en la configuración del sistema operativo o las bibliotecas.
- Eficiencia y rendimiento: Los contenedores de Docker son ligeros y tienen un tiempo de inicio rápido, lo que permite una mayor eficiencia en términos de recursos y rendimiento. Esto los hace ideales para el despliegue y la escalabilidad de aplicaciones en entornos de producción.
- Administración de recursos: Docker proporciona herramientas y comandos para administrar contenedores, como iniciar, detener, reiniciar y eliminar contenedores. También ofrece capacidades de orquestación de contenedores, como Docker Swarm y Kubernetes, para administrar y escalar aplicaciones en un clúster de servidores.
- Registro de imágenes: Docker cuenta con un registro de imágenes centralizado llamado Docker Hub, donde los desarrolladores pueden compartir y descargar imágenes de contenedor preconstruidas. También es posible crear y administrar registros privados para almacenar imágenes personalizadas y compartirlas dentro de una organización.

Docker se ha convertido en una tecnología popular en el desarrollo de aplicaciones y el despliegue en la nube debido a su facilidad de uso, portabilidad y eficiencia. Permite a los desarrolladores crear entornos de desarrollo consistentes y reproducibles, facilita la colaboración entre equipos y simplifica el proceso de implementación y escalabilidad de aplicaciones.

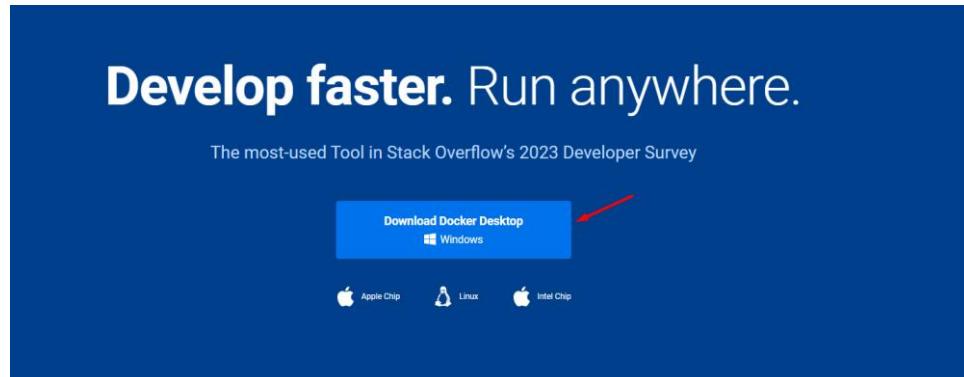
2.1.1. Instalación

Linux : <https://docs.docker.com/desktop/install/ubuntu/>

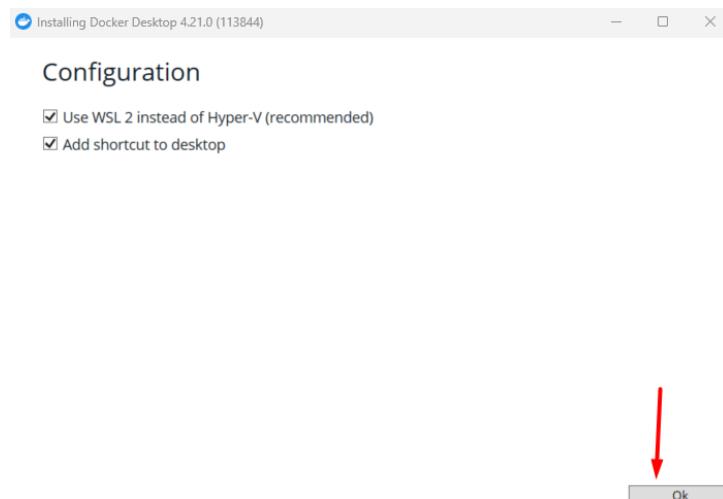
- Instalación gnome
 - sudo apt install gnome-terminal
- Eliminar instalaciones previas
 - sudo apt remove docker-desktop
- Para finalizar ejecute
 - `rm -r $HOME/.docker/desktop`

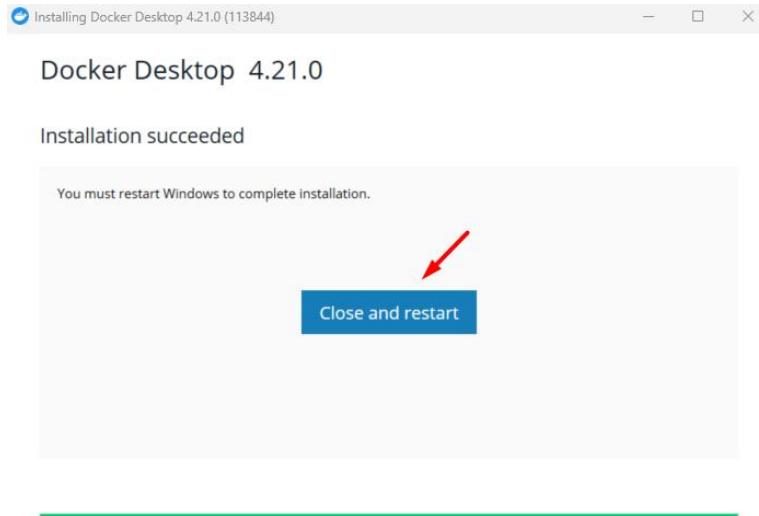
- `sudo rm /usr/local/bin/com.docker.cli`
 - `udo apt purge docker-desktop`
- Instalacion docker Desk
 - Actualizacion repository :
<https://docs.docker.com/engine/install/ubuntu/#set-up-the-repository>
 - `sudo apt-get update`
 - `sudo apt-get install ca-certificates curl gnupg`
 - Add Docker's official GPG ke
 - `sudo install -m 0755 -d /etc/apt/keyrings`
 - `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg`
 - `sudo chmod a+r /etc/apt/keyrings/docker.gpg`
 - Use the following command to set up the repository:
 - `echo \`
 - `"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]`
 - <https://download.docker.com/linux/ubuntu> \
 - `"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable"`
 - `| \`
 - `sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`

Windows



- Ejecute el Asistente de instalación





2.1.2. Configuración de Imagen Mysql en docker

- Ingrese a la pagina oficial de docker e inicie sesion
- En el buscador escriba Mysql
- Seleccione la opción de imagen oficial de mysql

Filters

Products

- Images
- Extensions
- Plugins

Trusted Content

- Docker Official Image
- Verified Publisher
- Sponsored OSS

Operating Systems

- Linux
- Windows

Architectures

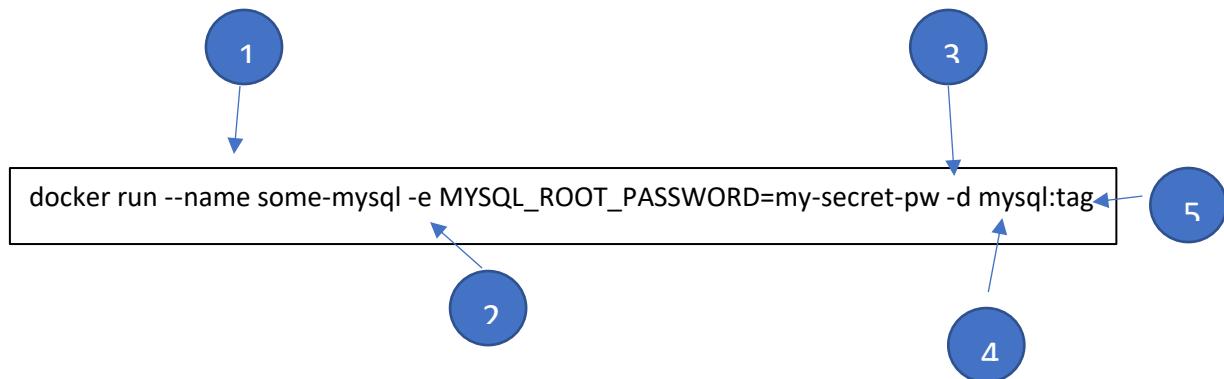
- ARM
- ARM 64
- IBM POWER
- IBM Z
- PowerPC 64 LE
- x86
- x86-64

1 - 25 of 10.000 results for mysql.

Best Match

Image	Name	Description	Updated	Pulls	Last week
	mysql	Docker OFFICIAL IMAGE · 1B+ · 10K+	Updated 16 days ago	5,160,930	Last week
	mariadb	Docker OFFICIAL IMAGE · 1B+ · 5.5K	Updated 5 days ago	1,209,493	Last week
	Percona	Docker OFFICIAL IMAGE · 100M+ · 617	Updated 9 days ago	495,046	Last week
	phpmyadmin	Docker OFFICIAL IMAGE · 50M+ · 831	Updated 16 days ago	116,485	Last week

- Comando de configuración de Instancia Mysql en docker



La instrucción **docker run** se utiliza para crear y ejecutar un contenedor de Docker a partir de una imagen.

1. **--name some-mysql:** Esto establece el nombre del contenedor como "some-mysql". Puedes elegir cualquier nombre que deseas para identificar el contenedor de manera única.
2. **-e MYSQL_ROOT_PASSWORD=my-secret-pw:** Esta opción establece una variable de entorno dentro del contenedor. En este caso, se está configurando la variable de entorno **MYSQL_ROOT_PASSWORD** con el valor "my-secret-pw". Esta variable define la contraseña de root para el servidor MySQL que se ejecutará dentro del contenedor.
3. **-d:** Esta opción indica que deseas que el contenedor se ejecute en segundo plano, es decir, en modo "detached".
4. **mysql:tag:** Esto especifica la imagen que se utilizará para crear el contenedor. En este caso, se está utilizando la imagen oficial de MySQL, seguida de la etiqueta o versión específica (**tag**) que deseas utilizar. Por ejemplo, podrías utilizar **mysql:latest** para obtener la última versión disponible.
5. **Tag :** Etiqueta o versión específica (**tag**) que deseas utilizar

En resumen, la instrucción **docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag** crea un contenedor llamado "some-mysql" utilizando la imagen de MySQL especificada. Configura la variable de entorno **MYSQL_ROOT_PASSWORD** con el valor "my-secret-pw" y ejecuta el contenedor en segundo plano.

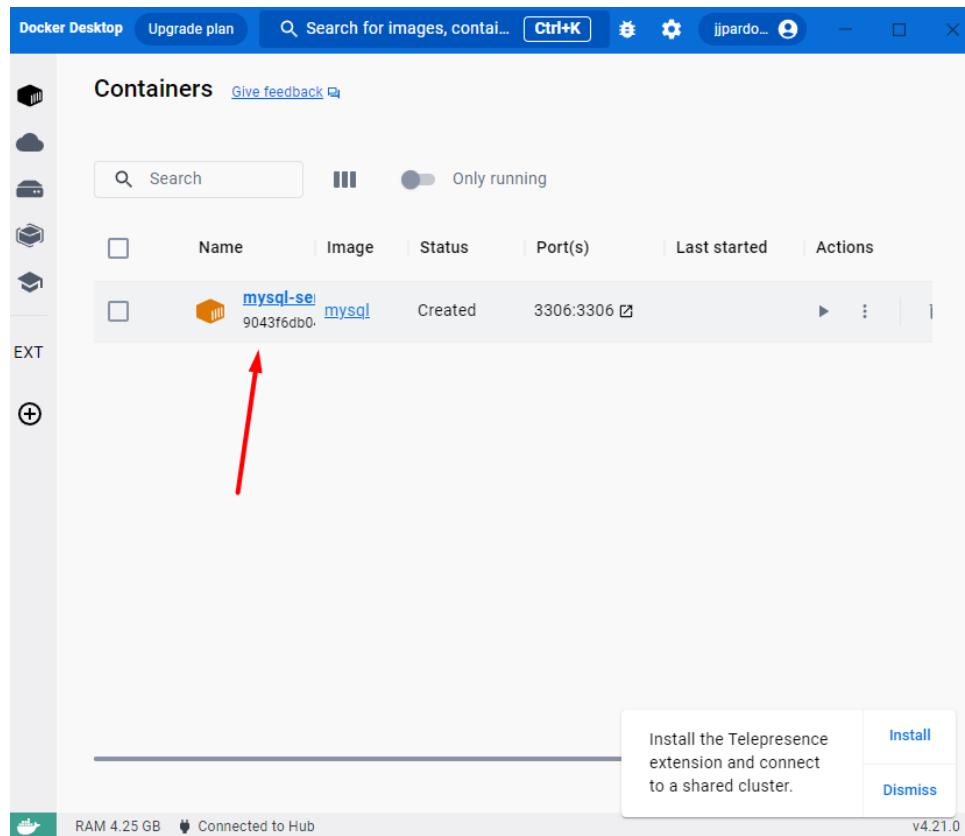
- Desde power shell de windows ejecute el comando

```
PS C:\Users\desarrollo> docker run --name mysql-server -p 3306:3306 -e MYSQL_ROOT_PASSWORD=admin -d mysql
```

```
docker run --name mysql-server -p 3306:3306 -e MYSQL_ROOT_PASSWORD=admin -d mysql
```

Si el contenedor de Mysql no existe lo descarga de forma automatica.

```
Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
PS C:\Users\desarrollo> docker run --name mysql-server -p 3306:3306 -e MYSQL_ROOT_PASSWORD=admin -d mysql
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
46ef68baacb7: Pull complete
94c1114b2e9c: Pull complete
ff05e3f38802: Pull complete
41cc3fcdf9912: Pull complete
97bbc8bdff52a: Pull complete
6d88f83726a0: Download complete
cfc5c7d5d33f7: Downloading [=====] 6.454MB/58.6MB
9db3175a2a66: Download complete
feaedeb27fa9: Downloading [>] 1.071MB/56.57MB
cf91e784414: Waiting
b1770db1c329: Waiting
|
```



3. Programación Avanzada

3.1. Proyecto Base (Arquitectura de la aplicación)

El proyecto estará compuesta de varias carpetas, la carpeta principal que contendrá toda la solución y 3 carpetas las cuales comenzaremos a continuación.

- Core : Esta carpeta contendrá todas las clases entidades e interfaces del project.
- Infrastructure : esa carpeta va a contener todo lo relacionado con el acceso de datos, los repositorios y las unidades de trabajo.
- Api: esta carpeta guardará el proyecto donde crearemos las diferentes apis o Endpoints para ser consumidos por las aplicaciones externas.

Durante el proceso de creación de este proyecto base se crearán migraciones de datos haciendo uso de entity framework; también tendremos la posibilidad de crear archivos que nos permitirán inicializar la base de datos con información externa.

- creación de la carpeta o contenedora:

```
PS D:\projectsNetCore> mkdir tienda ←  
  
Directorio: D:\projectsNetCore  
  
Mode          LastWriteTime        Length Name  
----          -----          ---- -  
d----  3/07/2023  12:44 p. m.           tienda
```

- A continuación, nos ubicamos en la carpeta que creamos llamada tienda y verificamos la versión que se encuentra instalada del dotnet

```
PS D:\projectsNetCore> cd tienda ↗
PS D:\projectsNetCore\tienda> dotnet --version ↗
7.0.302 ↙
PS D:\projectsNetCore\tienda>
```

- El siguiente paso es crear una solución dentro del directorio tienda para esto sigamos los siguientes pasos:

Ejecutar el comando dotnet –h : ese comando permite visualizar las diferentes opciones de SDK de .Net

```
Ejecute un comando del SDK de .NET.

sdk-options:
  -d|--diagnostics  Habilita la salida de diagnóstico.
  -h|--help          Muestra ayuda de la línea de comandos.
  --info             Muestra la información de .NET.
  --list-runtimes   Muestra los runtimes instalados.
  --list-sdks        Muestra los SDK instalados.
  --version          Muestra la versión del SDK de .NET en uso.

Comandos de SDK:
  add              Agrega un paquete o una referencia a un proyecto de .NET.
  build            Compila un proyecto de .NET.
  build-server     Interactúa con los servidores que inicia una compilación.
  clean            Limpia los resultados de compilación de un proyecto .NET.
  format           Aplicar preferencias de estilo a un proyecto o solución.
  help             Muestra ayuda de la línea de comandos.
  list              Enumera las referencias de proyecto de un proyecto de .NET.
  msbuild          Ejecuta comandos de Microsoft Build Engine (MSBuild).
  new               Crea un nuevo archivo o proyecto de .NET.
  nuget            Proporciona comandos NuGet adicionales.
  pack              Crea un paquete de NuGet.
  publish           Publica un proyecto de .NET para implementación.
  remove            Quita un paquete o una referencia de un proyecto de .NET.
  restore           Restaura dependencias especificadas en un proyecto de .NET.
  run               Compila y ejecuta la salida de un proyecto de .NET.
```

Ejecutar el comando dotnet new –h : Para visualizar la ayuda del comando.

```
PS D:\projectsNetCore\tienda> dotnet new -h
Description:
  Comandos de creación de instancias de la plantilla para CLI de .NET.

Usage:
  dotnet new [<template-short-name> [<template-args>...]] [options]
  dotnet new [command] [options]

Arguments:
  <template-short-name>  Nombre corto de la plantilla que se va a crear.
  <template-args>        Opciones específicas de la plantilla que se van a usar.

Options:
  -o, --output <output>    Ubicación en la que se colocará el resultado generado.
  -n, --name <name>        Nombre de la salida que se va a crear. Si no se especifica ningún nombre, se usa el nombre del directorio de salida.
  --dry-run                Muestra un resumen de lo que sucede si se ejecuta la línea de comandos dada si se crea una plantilla.
  --force                  Fuerza la generación de contenido aunque cambie a los archivos existentes.
  --no-update-check        Deshabilita la comprobación de las actualizaciones del paquete de plantillas al crear una plantilla de forma instantánea.
  --project <project>      Proyecto que se debe usar para la evaluación de contexto.
  -v, --verbosity <LEVEL>  Permite establecer el nivel de detalle. Los valores permitidos son q[uiet], m[inimal], n[ormal] y diag[nostic]. [default: normal]
  -d, --diagnostics        Permite habilitar la salida de diagnóstico.
  -?, -h, --help            Muestra ayuda de la línea de comandos.
```

Ejecutar el comando dotnet new –l : Permite ver las plantillas disponibles para la creación de proyectos. **Si Aparece una advertencia de desuso de –l use dotnet new list**

```
PS D:\projectsNetCore\tienda> dotnet new list ↗
Estas plantillas coinciden con su entrada:

Nombre de la plantilla           Nombre corto  Idioma  Etiquetas
-----                         -----
Aplicación Blazor para WebAssembly  blazorwasm  [C#]   Web/Blazor/WebAssembly/PWA
Aplicación Blazor Server          blazorserver  [C#]   Web/Blazor
Aplicación Blazor Server vacía    blazorserver-empty  [C#]   Web/Blazor/Empty
Aplicación Blazor WebAssembly vacía blazorwasm-empty  [C#]   Web/Blazor/WebAssembly/PWA/Empty
Aplicación de consola            console      [C#,F#,VB] Common/Console
Aplicación de Windows Forms       winforms     [C#,VB]  Common/WinForms
Aplicación web de ASP.NET Core    webapp,razor  [C#]   Web/MVC/Razor Pages
Aplicación web de ASP.NET Core (Modelo-Vista-Controlador) mvc      [C#,F#]  Web/MVC
Aplicación WPF                   wpf         [C#,VB]  Common/WPF
Archivo de bufer de protocolo    proto       [C#]   Web/gRPC
Archivo de la solución            sln,solution  [C#]   Solution
Archivo de manifiesto de la herramienta local de dotnet tool-manifest  Config
Archivo Directory.Build.props de MSBuild   buildprops  MSBuild/props
```

A continuación, debemos ejecutar el comando para crear la solución en la carpeta activa es decir la carpeta tienda. el comando es el siguiente: dotnet new sln

```
PS D:\projectsNetCore\tienda> dotnet new sln ↗  
La plantilla "Archivo de la solución" se creó correctamente.  
PS D:\projectsNetCore\tienda>
```

El archivo de la solución tendrá el mismo nombre de la carpeta es decir tienda

```
PS D:\projectsNetCore\tienda> dir  
  
Directorio: D:\projectsNetCore\tienda  
  
Mode LastWriteTime Length Name  
---- ----- ---- -  
-a--- 3/07/2023 1:07 p. m. 441 tienda.sln ↗
```

A continuación, vamos a crear el proyecto webApi, para esto ejecutamos el comando

dotnet new webapi -o API.

```
PS D:\projectsNetCore\tienda> dotnet new webapi -o API ↗  
La plantilla "ASP.NET Core Web API" se creó correctamente.  
  
Procesando acciones posteriores a la creación...  
Restaurando D:\projectsNetCore\tienda\API\API.csproj:  
    Determinando los proyectos que se van a restaurar...  
    Se ha restaurado D:\projectsNetCore\tienda\API\API.csproj (en 4,85 sec).  
Restauración realizada correctamente.  
  
PS D:\projectsNetCore\tienda>
```

A continuación debemos asociar el proyecto api con la solución global para esto ejecuta el siguiente comando: **dotnet sln add .\API**

```
PS D:\projectsNetCore\tienda> dotnet sln add .\API\ ↗  
Se ha agregado el proyecto "API\API.csproj" a la solución. ↗  
PS D:\projectsNetCore\tienda>
```

Para continuar con la arquitectura del proyecto vamos a crear los siguientes proyectos el de core y el de infraestructura, Para esto siga los siguientes comandos:

dotnet new classlib -o core

```
PS D:\projectsNetCore\tienda> dotnet new classlib -o Core ←
La plantilla "Biblioteca de clases" se creó correctamente.

Procesando acciones posteriores a la creación...
Restaurando D:\projectsNetCore\tienda\Core\Core.csproj:
  Determinando los proyectos que se van a restaurar...
    Se ha restaurado D:\projectsNetCore\tienda\Core\Core.csproj (en 45 ms).
  Restauración realizada correctamente.
```

Se agrega Core a la solución.

```
PS D:\projectsNetCore\tienda> dotnet sln add .\Core\
Se ha agregado el proyecto "Core\Core.csproj" a la solución.
PS D:\projectsNetCore\tienda> |
```

Crear biblioteca de clases de Infraestructure:

```
PS D:\projectsNetCore\tienda> dotnet new classlib -o Infrastructure ←
La plantilla "Biblioteca de clases" se creó correctamente.

Procesando acciones posteriores a la creación...
Restaurando D:\projectsNetCore\tienda\Infrastructure\Infrastructure.csproj:
  Determinando los proyectos que se van a restaurar...
    Se ha restaurado D:\projectsNetCore\tienda\Infrastructure\Infrastructure.csproj (en 47 ms).
  Restauración realizada correctamente.

PS D:\projectsNetCore\tienda> dotnet sln add .\Infrastructure\←
Se ha agregado el proyecto "Infrastructure\Infrastructure.csproj" a la solución.
PS D:\projectsNetCore\tienda> |
```

A continuación, se verificará la cantidad de proyectos que se encuentran asociados a la solución. Para esto ingrese el siguiente comando: **dotnet sln list**

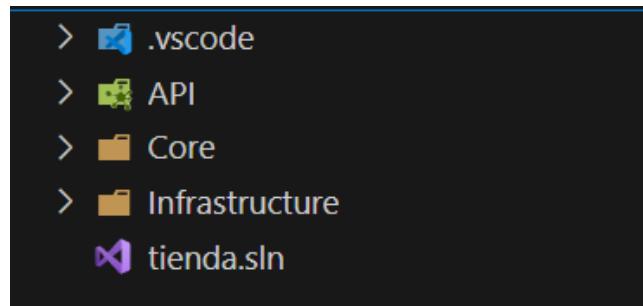
```
PS D:\projectsNetCore\tienda> dotnet sln list ←
Proyectos
-----
API\API.csproj
Core\Core.csproj
Infrastructure\Infrastructure.csproj
PS D:\projectsNetCore\tienda> |
```

Como se puede visualizar en la gráfica anterior el comando me lista todos los proyectos que se encuentran asociados a la solución tienda.

Teniendo en cuenta los requerimientos de la arquitectura que hemos definido tenemos que establecer una relación entre cada uno de los proyectos en este caso infraestructura dependerá de uno de los proyectos anteriores y el proyecto api también tendrá referencia con otro de los proyectos que hemos definido a continuación siga los pasos para establecer la relación entre cada uno de estos proyectos mencionados anteriormente. Es importante tener en cuenta que para establecer relación entre los proyectos se debe que estar ubicado en la carpeta del proyecto que se va a relacionar: Por ejemplo, si infraestructura depende no se relaciona con core nos debemos ubicar en la carpeta de infraestructura. Siga los pasos teniendo en cuenta las siguientes imágenes:

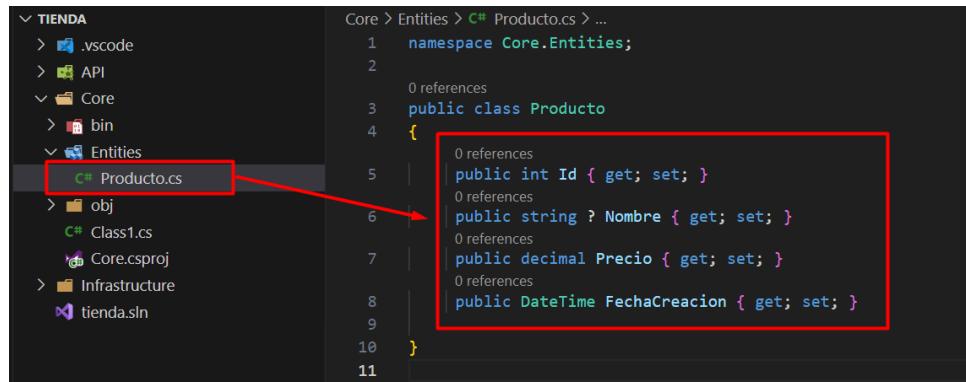
```
PS D:\projectsNetCore\tienda> cd .\Infrastructure\
PS D:\projectsNetCore\tienda\Infrastructure> dotnet add reference ..\Core\
Se ha agregado la referencia "..\Core\Core.csproj" al proyecto.
PS D:\projectsNetCore\tienda\Infrastructure> |
```

```
PS D:\projectsNetCore\tienda\Infrastructure> cd ..  
PS D:\projectsNetCore\tienda> cd .\API\  
PS D:\projectsNetCore\tienda\API> dotnet add reference ..\Infrastructure\  
Se ha agregado la referencia "..\Infrastructure\Infrastructure.csproj" al proyecto.  
PS D:\projectsNetCore\tienda\API> |
```



3.2. Proyecto Base (Entidades)

Para la generación de las entidades del proyecto se debe crear una carpeta llamada entidades o entities en la carpeta core. a continuación, siga los pasos qué se muestran en las imágenes siguientes:



```
Core > Entities > C# Producto.cs > ...
1  namespace Core.Entities;
2
3  public class Producto
4  {
5      public int Id { get; set; }
6      public string? Nombre { get; set; }
7      public decimal Precio { get; set; }
8      public DateTime FechaCreacion { get; set; }
9
10 }
11
```

namespace Core.Entities;

```
public class Producto
{
    public int Id { get; set; }

    public string? Nombre { get; set; }

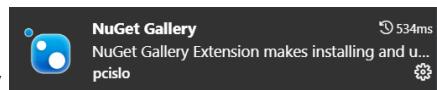
    public decimal Precio { get; set; }

    public DateTime FechaCreacion { get; set; }
```

}

A continuación, se deben agregar los paquetes de entity framework para poder trabajar con todo el esquema de punto net para esto podemos hacerlo de diferentes maneras desde la línea de comandos o haciendo uso de visual studio code. En esta ocasión vamos a realizar la instalación de estos paquetes usando visual studio code siga los siguientes pasos:

- ingrese en la paleta de comandos utilizando el menú view > command palette. Escribe Nuget. Para que este comando funcione correctamente se debe contar



con la extensión de Nuget gallery

>nuget

NuGet: Open NuGet Gallery

recently used

The screenshot shows the NuGet Gallery interface with the search bar containing 'C# Producto.cs'. The results page is for 'Newtonsoft.Json' by James Newton-King, version v13.0.3. Below it, several related packages are listed:

- Microsoft.Extensions.DependencyInjection** by Microsoft, v7.0.0: Default implementation of dependency injection for Microsoft.Extensions.DependencyInjection.
- Microsoft.Extensions.Logging** by Microsoft, v7.0.0: Logging infrastructure default implementation for Microsoft.Extensions.Logging.
- System.Text.Json** by Microsoft, v7.0.3: Provides high-performance and low-allocating types that serialize objects to JavaScript Object Notation (JSON) text and deserialize
- Microsoft.Bcl.AsyncInterfaces** by Microsoft, v7.0.0: Provides the IAsyncEnumerable<T> and IAsyncDisposable interfaces and helper types for .NET Standard 2.0. This package is

The screenshot shows the NuGet Gallery interface with the search bar containing 'Microsoft.Ent'. The results page is for 'Microsoft.EntityFrameworkCore' by Microsoft, version v7.0.8. Below it, other Entity Framework Core packages are listed:

- Microsoft.EntityFrameworkCore.Relational** by Microsoft, v7.0.8: Shared Entity Framework Core components for relational database providers.
- Microsoft.EntityFrameworkCore.Abstractions** by Microsoft, v7.0.8: Provides abstractions and attributes that are used to configure Entity Framework Core.

The screenshot shows the details page for 'Microsoft.EntityFrameworkCore' version v7.0.8. A red box highlights the package entry. To the right, a modal window titled 'Microsoft.EntityFrameworkCore ...' shows three checkboxes: 'API.csproj Install', 'Core.csproj Install', and 'Infrastructure.csproj Install'. The 'Infrastructure.csproj Install' checkbox is checked. Red arrows point from the highlighted package to the checked checkbox and the 'Install' button in the modal.

Instalación proveedor Mysql : para realizar la instalación del proveedor de Mysql debemos realizarlo a través de la línea de comandos, para eso debemos ingresar a la página web de los nugets. <https://www.nuget.org/>. quién es buscador escribimos la palabra mysql. Cuando se realiza la búsqueda aparecen diferentes conectores se recomienda utilizar el conector que se encuentra encerrado en la imagen inferior.

Frameworks

- .NET
- .NET Core
- .NET Standard
- .NET Framework

Package type

- All types
- Dependency
- .NET tool
- Template

Options

- Include prerelease

Sort by Relevance

2.093 packages returned for mysql

Package	Version	Downloads	Last updated	Latest version	Tags
MySql.Data	8.0.33	63.642.259	3 months ago	8.0.33	MySQL .NET Connector Connector/.NET core Connector/.Net coreclr C/.NET More tags
Pomelo.EntityFrameworkCore.MySQL	7.0.0	43.304.929	6 months ago	7.0.0	pomelo mysql mariadb Entity Framework Core entity-framework-core ef efcore ef More tags
MySqlConnector	2.3.0-beta.2	70.919.276	3 months ago	2.3.0-beta.2	mysql mysqlconnector async ado.net database netcore

Apply Reset

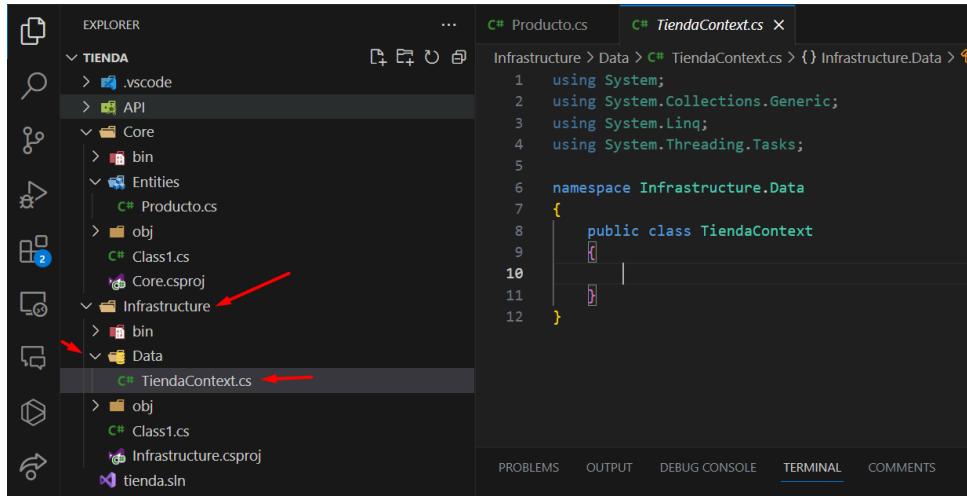
Y por último ejecutamos el comando `dotnet add package Pomelo.EntityFrameworkCore.MySQL --version 7.0.0` en la terminal de visual studio code. Es importante tener en cuenta que la instalación la debemos realizar dentro de la carpeta infraestructura.

```
desarrollo@DESKTOP-L4V647N MINGW64 /d/projectsNetCore/tienda/Infrastructure
$ dotnet add package Pomelo.EntityFrameworkCore.MySQL --version 7.0.0
Determinando los proyectos que se van a restaurar...
Writing C:\Users\desarrollo\AppData\Local\Temp\tmpF3E7.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for
de signing.
```

Otros conectores:

SQL Server or Azure SQL	.UseSqlServer(connectionString)	Microsoft.EntityFrameworkCore.SqlServer
Azure Cosmos DB	.UseCosmos(connectionString, databaseName)	Microsoft.EntityFrameworkCore.Cosmos
SQLite	.UseSqlite(connectionString)	Microsoft.EntityFrameworkCore.SQLite
EF Core in-memory database	.UseInMemoryDatabase(databaseName)	Microsoft.EntityFrameworkCore.InMemory
PostgreSQL*	.UseNpgsql(connectionString)	Npgsql.EntityFrameworkCore.PostgreSQL
MySQL/Maria DB*	.UseMySql(connectionString)	Pomelo.EntityFrameworkCore.MySql
Oracle*	.UseOracle(connectionString)	Oracle.EntityFrameworkCore

El siguiente paso que vamos a realizar es la creación de la clase de contexto la cual se utilizará para la conexión a la base de datos. Para una organización efectiva del proyecto la clase de contexto se crea dentro de la carpeta infraestructura y una carpeta interna llamada data. Como buena práctica la clase de contexto le vamos a colocar el nombre del proyecto seguido de la palabra context, teniendo en cuenta la estructura CamelCase.



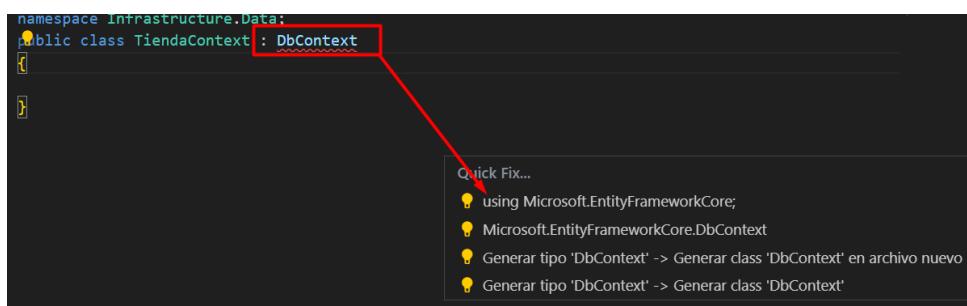
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Infrastructure.Data
{
    public class TiendaContext
    {
    }
}

```

La clase tiendaContext heredará de la clase DbContext la cual pertenece al entity framework core.



A continuación, se debe crear el constructor de la clase para esto haga clic derecho sobre un área o línea de texto vacía dentro de la clase y seleccione la opción

Refactor...

Ctrl+Shift+R

y a continuación seleccione la y a continuación

seleccione la opción  Generar el constructor 'TiendaContext(options)' la cual aparece en la ventana en la ventana emergente.

```
1  using Microsoft.EntityFrameworkCore;
2  namespace Infrastructure.Data;
3  public class TiendaContext : DbContext
4  {
5      public TiendaContext(DbContextOptions<TiendaContext> options) : base(options)
6      {
7      }
8  }
```

Añadir esta directiva

```
using Microsoft.EntityFrameworkCore;
```

```
namespace Infrastructure.Data;
```

```
public class TiendaContext : DbContext
{
    public TiendaContext(DbContextOptions<TiendaContext> options) : base(options)
    {
    }
}
```

A continuación, se debe establecer referencia a cada una de las entidades que hemos creado en la carpeta core. para el ejemplo práctico solamente hemos creado una sola entidad.

```
1  using Core.Entities; ←
2  using Microsoft.EntityFrameworkCore; ←
3  namespace Infrastructure.Data;
4  public class TiendaContext : DbContext
5  {
6      public TiendaContext(DbContextOptions<TiendaContext> options) : base(options)
7      {
8      }
9      public DbSet<Producto> Productos { get; set; } ←
10 }
11
```

```
using Core.Entities;
```

```
using Microsoft.EntityFrameworkCore;
```

```
namespace Infrastructure.Data;
```

```
public class TiendaContext : DbContext
```

```
{
```

```
public TiendaContext(DbContextOptions<TiendaContext> options) : base(options)
{
}

public DbSet<Producto> Productos { get; set; }

}
```

Muy bien ahora que ya hemos creado nuestra clase de contexto la debemos agregar Al contenedor de inyección de dependencia si deseamos poder utilizar la clase de contexto en todos los componentes del proyecto.

¿Qué es el contenedor de inyección de dependencias?

El contenedor de inyección de dependencias es un patrón de diseño y una característica común en muchos marcos de trabajo y contenedores de inversión de control (IoC, por sus siglas en inglés), incluido el ecosistema de desarrollo de .NET. Proporciona una forma de gestionar y resolver las dependencias entre los componentes de una aplicación.

En términos simples, la inyección de dependencias es un enfoque para administrar las dependencias entre objetos. En lugar de que un objeto cree y mantenga sus propias dependencias, se delega la responsabilidad de proporcionar las dependencias externamente. El contenedor de inyección de dependencias es el encargado de resolver estas dependencias y proporcionar las instancias necesarias.

El contenedor de inyección de dependencias mantiene un registro de las dependencias y sus configuraciones correspondientes. Cuando se solicita una instancia de un objeto que tiene dependencias, el contenedor se encarga de crear y proporcionar esas dependencias automáticamente.

El contenedor de inyección de dependencias también se encarga de administrar el ciclo de vida de las instancias creadas. Puede mantener una única instancia de una dependencia durante toda la duración de la aplicación (singleton), crear una nueva instancia para cada solicitud (transient) o mantener una instancia por solicitud en un determinado ámbito (scoped).

La inyección de dependencias tiene varios beneficios, incluyendo la facilidad de mantenimiento, la mejora de la modularidad y la facilitación de las pruebas unitarias, ya que se pueden reemplazar las dependencias reales por implementaciones simuladas o de prueba.

En el ecosistema de desarrollo de .NET, el contenedor de inyección de dependencias más comúnmente utilizado es el contenedor de servicios incorporado en ASP.NET Core, que proporciona una implementación de inyección de dependencias robusta y flexible para desarrollar aplicaciones web.

La configuración del contenedor de inyección de dependencias las debemos realizar generar archivo program.cs de la carpeta API.

```

API > C# Program.cs
1 var builder = WebApplication.CreateBuilder(args);
2
3 // Add services to the container.
4
5 builder.Services.AddControllers();
6 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
7 builder.Services.AddEndpointsApiExplorer();
8 builder.Services.AddSwaggerGen();
9
10 var app = builder.Build();
11
12 // Configure the HTTP request pipeline.
13 if (app.Environment.IsDevelopment())
14 {
15     app.UseSwagger();
16     app.UseSwaggerUI();
17 }

```

```
var builder = WebApplication.CreateBuilder(args);
```

```
// Add services to the container
```

```
builder.Services.AddControllers();
```

```
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
```

```
builder.Services.AddEndpointsApiExplorer();
```

```
builder.Services.AddSwaggerGen();
```

```
var app = builder.Build();
```

```
// Configure the HTTP request pipeline.
```

```
if (app.Environment.IsDevelopment())
```

```
{
```

```
    app.UseSwagger();
```

```
    app.UseSwaggerUI();
```

```
}
```

```

API > C# Program.cs
1 using Infrastructure.Data;
2 using Microsoft.EntityFrameworkCore;
3
4 var builder = WebApplication.CreateBuilder(args);
5
6 // Add services to the container.
7
8 builder.Services.AddControllers();
9 builder.Services.AddDbContext<TiendaContext>(optionsBuilder =>
10 {
11     string? connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
12     optionsBuilder.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString));
13 });
14 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
15 builder.Services.AddEndpointsApiExplorer();
16 builder.Services.AddSwaggerGen();
17
18 var app = builder.Build();
19
20 // Configure the HTTP request pipeline.

```

using Infrastructure.Data;

```
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container

builder.Services.AddControllers();
builder.Services.AddDbContext<TiendaContext>(optionsBuilder =>
{
    string ? connectionString =
    builder.Configuration.GetConnectionString("DefaultConnection");
    optionsBuilder.UseMySql(connectionString,
    ServerVersion.AutoDetect(connectionString));
});

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore-api-configuration
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

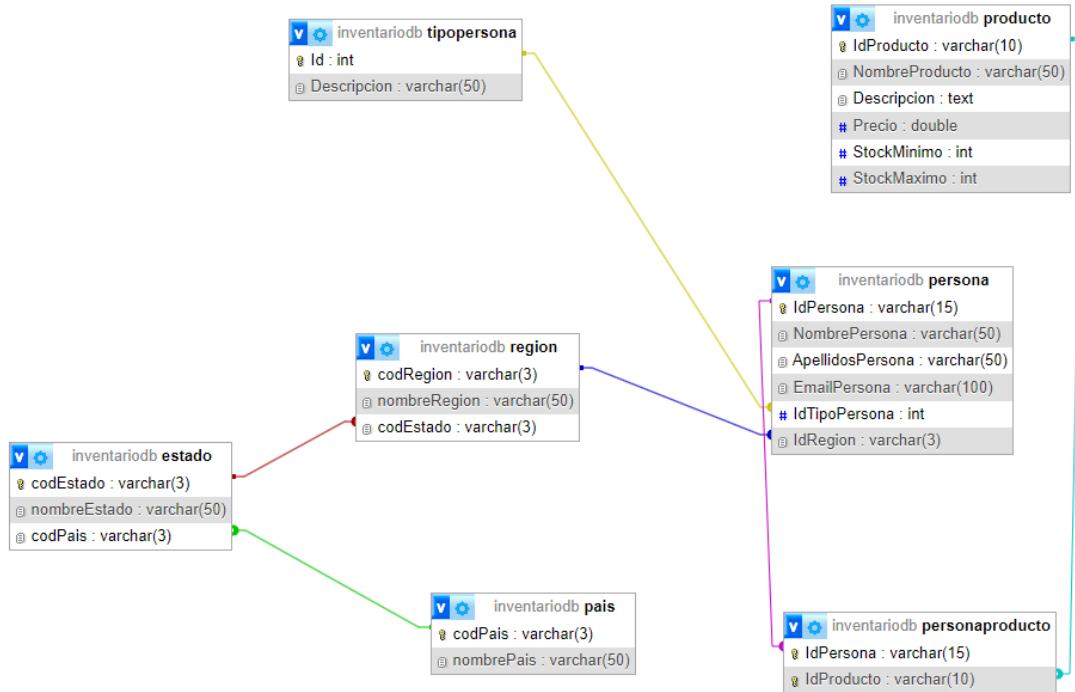
var app = builder.Build();
```

Nota. En caso de salir un error de referencia en DbContext ejecutar la siguiente instrucción en la carpeta API **dotnet add package Microsoft.EntityFrameworkCore**

Para finalizar la configuración de la conexión a la base de datos vamos a crear una variable de conexión en el archivo **{ } appsettings.Development.json**.

3.3. Relaciones entre Entidades

Entity Framework admite tres tipos de relaciones, al igual que una base de datos: 1) Uno a uno, 2) Uno a muchos y 3) Muchos a muchos.



Relación Uno a Muchos

La cardinalidad uno a muchos se refiere a una relación entre dos entidades en la cual una entidad tiene una relación única con una o varias entidades de la otra entidad, pero la entidad relacionada solo tiene una relación con la entidad original.

En términos más simples, esto significa que una instancia de una entidad A puede estar relacionada con múltiples instancias de una entidad B, pero una instancia de la entidad B solo puede estar relacionada con una instancia de la entidad A.

Para crear una referencia entre entidades siga las siguientes instrucciones:

- Genere la entidad principal y secundaria. Ver DER. Para el ejemplo práctico se crearán las relaciones entre país y estado.

```
1  using System.ComponentModel.DataAnnotations;
2  namespace Core.Entities;
3
4  public class Pais
5  {
6      [Key]
7      public string? codPais { get; set; }
8      public string? nombrePais { get; set; }
9      public ICollection<Estado>? Estados { get; set; }
10 }
```

using System.ComponentModel.DataAnnotations;

namespace Core.Entities;

public class Pais

{

[Key]

public string? codPais { get; set; }

public string? nombrePais { get; set; }

public ICollection<Estado> Estados { get; set; }

}

using System.ComponentModel.DataAnnotations;

Esta línea importa el espacio de nombres **System.ComponentModel.DataAnnotations**, que proporciona atributos para realizar validaciones y especificar metadatos en las entidades.

namespace Core.Entities

Esta línea indica que la clase "Pais" pertenece al espacio de nombres "Core.Entities".

Namespace Core.Entities

public class Pais

Aquí comienza la definición de la clase "Pais".

public class Pais;

```
[Key]
```

```
public string? codPais { get; set; }
```

El atributo [Key] indica que la propiedad "codPais" es la clave primaria de la entidad. Es de tipo **string** y permite valores nulos (**string?**).

```
[Key]
```

```
public string? codPais { get; set; }
```

```
public string? nombrePais { get; set; }
```

Esta propiedad "nombrePais" representa el nombre del país y también permite valores nulos.

```
public string? nombrePais { get; set; }
```

```
public ICollection<Estado>? Estados { get; set; }
```

La propiedad "Estados" es una colección (**ICollection**) de objetos de tipo **Estado** (otra entidad) y representa la relación entre el país y sus estados. El signo de interrogación (**ICollection<Estado>?**) indica que esta propiedad puede ser nula.

```
public ICollection<Estado> Estados { get; set; }
```

```
1  using System.ComponentModel.DataAnnotations;
2  namespace Core.Entities;
3  5 references
4  public class Estado
5  {
6      [Key]
7      1 reference
8      public string? codEstado { get; set; }
9      1 reference
10     public string? nombreEstado { get; set; }
11     1 reference
12     public string? codPais { get; set; }
13     1 reference
14     public País? País { get; set; }
15 }
```

```
using System.ComponentModel.DataAnnotations;
```

```
namespace Core.Entities;
```

```
public class Estado
{
    [Key]
    public string? codEstado { get; set; }
    public string? nombreEstado { get; set; }
    public string? codPais { get; set; }
    public Pais? Pais { get; set; }
}
```

```
using System.ComponentModel.DataAnnotations;
```

Esta línea importa el espacio de nombres **System.ComponentModel.DataAnnotations**, que proporciona atributos para realizar validaciones y especificar metadatos en las entidades.

```
using System.ComponentModel.DataAnnotations;
```

```
namespace Core.Entities
```

Esta línea indica que la clase "Estado" pertenece al espacio de nombres "Core.Entities".

```
namespace Core.Entities
```

```
public class Estado
```

Aquí comienza la definición de la clase "Estado".

```
public class Estado
```

```
[Key]
```

```
public string? codEstado { get; set; }
```

El atributo **[Key]** indica que la propiedad "codEstado" es la clave primaria de la entidad. Es de tipo **string** y permite valores nulos (**string?**).

```
[Key]
```

```
public string? codEstado { get; set; }
```

```
public string? nombreEstado { get; set; }
```

Esta propiedad "nombreEstado" representa el nombre del estado y también permite valores nulos.

En resumen, la clase "Estado" representa un estado dentro de un país, con propiedades para el código del estado, su nombre, el código del país al que pertenece, una referencia al objeto "Pais".

```
public string? nombreEstado { get; set; }
```

En la clase de contexto se deben agregar los DBSet correspondiente a las entidades estado y país.

```
1  using System.Reflection;
2  using Core.Entities;
3  using Microsoft.EntityFrameworkCore;
4  namespace Infrastructure.Data;
5  public class InventarioContext : DbContext
6  {
7      0 references
8      public InventarioContext(DbContextOptions<InventarioContext> options) : base(options)
9      {
10     }
11    2 references
12    public DbSet<Pais>? Paises { get; set; }
13    0 references
14    public DbSet<Estado>? Estados { get; set; }
15    0 references
16    public DbSet<Region>? Regiones { get; set; }
17    0 references
18    public DbSet<Persona>? Personas { get; set; }
19    0 references
20    public DbSet<TipoPersona>? TipoPersonas { get; set; }
21    0 references
22    public DbSet<Producto>? Productos { get; set; }
23    0 references
24    public DbSet<PersonaProducto>? PersonaProductos { get; set; }
25    1 reference
26    protected override void OnModelCreating(ModelBuilder modelBuilder)
27    {
28        modelBuilder.Entity<Persona>().HasIndex(idx => idx.EmailPersona).IsUnique();
29        modelBuilder.Entity<PersonaProducto>().HasKey(r => new {r.IdPersona,r.IdProducto});
30        base.OnModelCreating(modelBuilder);
31        modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
32    }
33}
```

```
using System.Reflection;
using Core.Entities;
using Microsoft.EntityFrameworkCore;
namespace Infrastructure.Data;
```

```
public class InventarioContext : DbContext
{
    public InventarioContext(DbContextOptions<InventarioContext> options) :
    base(options)

    {
    }

    public DbSet<Pais>? Paises { get; set; }
```

```

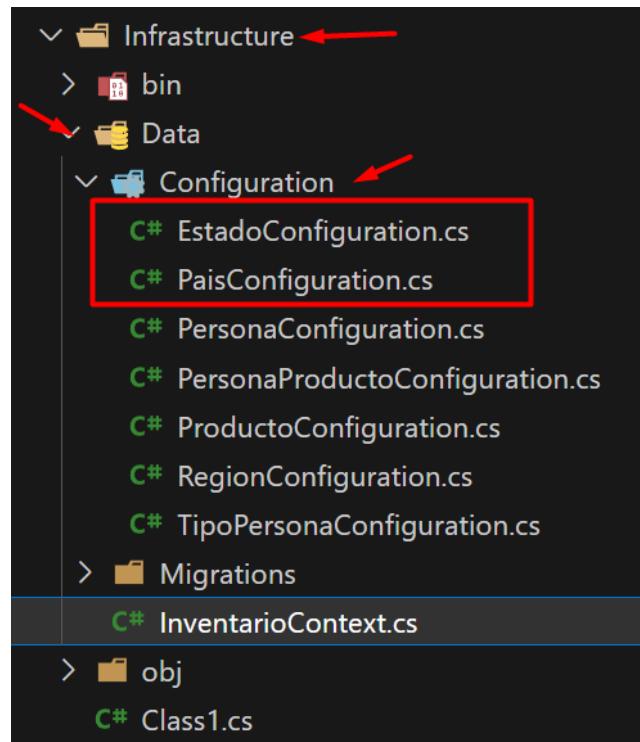
public DbSet<Estado>? Estados { get; set; }
public DbSet<Region>? Regiones { get; set; }
public DbSet<Persona>? Personas { get; set; }
public DbSet<TipoPersona>? TipoPersonas { get; set; }
public DbSet<Producto>? Productos { get; set; }
public DbSet<PersonaProducto>? PersonaProductos { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Persona>().HasIndex(idx => idx.EmailPersona).IsUnique();
    modelBuilder.Entity<PersonaProducto>().HasKey(r => new {r.IdPersona,
r.IdProducto});
    base.OnModelCreating(modelBuilder);

modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
}
}

```

Genere los archivos de configuración para modificar la estructura de las tablas en base de datos y definir las llaves principales y foráneas. En este paso se debe crear una carpeta llamada Configuration dentro de la carpeta Data en el proyecto Infrastructure .



PaisConfiguration

```
1  using Core.Entities;
2  using Microsoft.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore.Metadata;
4  using Microsoft.EntityFrameworkCore.Metadata.Builders;
5  namespace Infrastructure.Data.Configuration;
6  public class PaisConfiguration : IEntityTypeConfiguration<Pais>
7  {
8      public void Configure(EntityTypeBuilder<Pais> builder)
9      {
10         builder.ToTable("Pais");
11         builder.Property(p => p.codPais)
12             .HasAnnotation("MySQL:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn)
13             .HasMaxLength(3);
14
15         builder.Property(p => p.nombrePais)
16             .IsRequired()
17             .HasMaxLength(50);
18
19     }
20 }
21 }
```

```
using Core.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
namespace Infrastructure.Data.Configuration;

public class PaisConfiguration : IEntityTypeConfiguration<Pais>
{
    public void Configure(EntityTypeBuilder<Pais> builder)
    {
        builder.ToTable("Pais");

        builder.Property(p => p.codPais)
            .HasAnnotation("MySQL:ValueGenerationStrategy",
MySqlValueGenerationStrategy.IdentityColumn)
            .HasMaxLength(3);

        builder.Property(p => p.nombrePais)
            .IsRequired()
            .HasMaxLength(50);
    }
}
```

```
}
```

```
using Core.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

Estas líneas importan los espacios de nombres necesarios para trabajar con Entity Framework Core y configurar las entidades.

```
using Core.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

```
namespace Infrastructure.Data.Configuration
```

Esta línea indica que la clase "PaisConfiguration" pertenece al espacio de nombres "Infrastructure.Data.Configuration".

```
namespace Infrastructure.Data.Configuration;
```

```
public class PaisConfiguration : IEntityTypeConfiguration<Pais>
```

Aquí comienza la definición de la clase "PaisConfiguration". La clase implementa la interfaz **IEntityTypeConfiguration<Pais>**, que se utiliza para configurar el mapeo y las propiedades de la entidad "Pais".

```
public class PaisConfiguration : IEntityTypeConfiguration<Pais>
```

```
public void Configure(EntityTypeBuilder<Pais> builder)
```

Este método **Configure** es parte de la interfaz **IEntityTypeConfiguration** y se utiliza para configurar las propiedades y el mapeo de la entidad "Pais".

```
public void Configure(EntityTypeBuilder<Pais> builder)
```

```
builder.ToTable("Pais");
```

La línea **builder.ToTable("Pais")** indica que la tabla correspondiente a la entidad "Pais" se llamará "Pais" en la base de datos.

```
builder.ToTable("Pais");
```

```
builder.Property(p => p.codPais)
    .HasAnnotation(" MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn)
    .HasMaxLength(3);
```

La línea

builder.Property(p => p.codPais) se utiliza para configurar la propiedad "codPais" de la entidad "Pais". **.HasMaxLength(3)** establece la longitud máxima de caracteres permitidos para el código del país.

**.HasAnnotation(" MySql:ValueGenerationStrategy",
MySqlValueGenerationStrategy.IdentityColumn)** indica que el valor de la propiedad se generará automáticamente en la base de datos como una columna de identidad.

```
builder.Property(p => p.codPais)
```

```
    .HasAnnotation(" MySql:ValueGenerationStrategy",
    MySqlValueGenerationStrategy.IdentityColumn)
```

```
    .HasMaxLength(3);
```

```
builder.Property(p => p.nombrePais)
    .IsRequired()
    .HasMaxLength(50);
```

La línea **builder.Property(p => p.nombrePais)**

se utiliza para configurar la propiedad "nombrePais" de la entidad "Pais". **.IsRequired() indica que el valor de la propiedad es obligatorio y **.HasMaxLength(50)** establece la longitud máxima de caracteres permitidos para el nombre del país.**

```
builder.Property(p => p.nombrePais)
```

```
    .IsRequired()
```

```
    .HasMaxLength(50);
```

En resumen, el código representa la configuración de la entidad "Pais" utilizando Entity Framework Core. Define el nombre de la tabla, las propiedades y sus restricciones, y cómo se generará la clave primaria y las columnas en la base de datos.

EstadoConfiguration

```
1  using Core.Entities;
2  using Microsoft.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore.Metadata;
4  using Microsoft.EntityFrameworkCore.Metadata.Builders;
5  namespace Infrastructure.Data.Configuration;
6  public class EstadoConfiguration : IEntityTypeConfiguration<Estado>
7  {
8      0 references
9      public void Configure(EntityTypeBuilder<Estado> builder)
10     [
11         builder.ToTable("Estado");
12
13         builder.Property(p => p.codEstado)
14             .HasAnnotation(" MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn)
15             .HasMaxLength(3);
16
17         builder.Property(p => p.nombreEstado)
18             .IsRequired()
19             .HasMaxLength(50);
20
21         builder.HasOne(p => p.Pais)
22             .WithMany(p => p.Estados)
23             .HasForeignKey(p => p.codPais);
24     ]
25 }
```

```
using Core.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
namespace Infrastructure.Data.Configuration;
```

```
public class EstadoConfiguration : IEntityTypeConfiguration<Estado>
{
    public void Configure(EntityTypeBuilder<Estado> builder)
    {
        builder.ToTable("Estado");

        builder.Property(p => p.codEstado)
            .HasAnnotation(" MySql:ValueGenerationStrategy",
MySqlValueGenerationStrategy.IdentityColumn)
            .HasMaxLength(3);

        builder.Property(p => p.nombreEstado)
            .IsRequired()
            .HasMaxLength(50);
```

```
builder.HasOne(p => p.Pais)
    .WithMany(p => p.Estados)
    .HasForeignKey(p => p.codPais);
}

}
```

```
using Core.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

Estas líneas importan los espacios de nombres necesarios para trabajar con Entity Framework Core y configurar las entidades.

```
using Core.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

```
namespace Infrastructure.Data.Configuration
```

Esta línea indica que la clase "EstadoConfiguration" pertenece al espacio de nombres "Infrastructure.Data.Configuration".

```
namespace Infrastructure.Data.Configuration;
```

```
public class EstadoConfiguration : IEntityTypeConfiguration<Estado>
```

Aquí comienza la definición de la clase "EstadoConfiguration". La clase implementa la interfaz **IEntityTypeConfiguration<Estado>**, que se utiliza para configurar el mapeo y las propiedades de la entidad "Estado".

```
public class EstadoConfiguration : IEntityTypeConfiguration<Estado>
```

```
public void Configure(EntityTypeBuilder<Estado> builder)
```

Este método **Configure** es parte de la interfaz **IEntityTypeConfiguration** y se utiliza para configurar las propiedades y el mapeo de la entidad "Estado".

```
public void Configure(EntityTypeBuilder<Estado> builder)
```

```
    builder.ToTable("Estado");
```

La línea **builder.ToTable("Estado")** indica que la tabla correspondiente a la entidad "Estado" se llamará "Estado" en la base de datos.

```
builder.ToTable("Estado");
```

```
builder.Property(p => p.codEstado)
    .HasAnnotation(" MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn)
    .HasMaxLength(3);
```

La línea **builder.Property(p => p.codEstado)** se utiliza para configurar la propiedad "codEstado" de la entidad "Estado". **.HasMaxLength(3)** establece la longitud máxima de caracteres permitidos para el código del estado. **.HasAnnotation(" MySql:ValueGenerationStrategy", MySqlValueGenerationStrategy.IdentityColumn)** indica que el valor de la propiedad se generará automáticamente en la base de datos como una columna de identidad.

```
builder.Property(p => p.codEstado)
    .HasAnnotation(" MySql:ValueGenerationStrategy",
        MySqlValueGenerationStrategy.IdentityColumn)
    .HasMaxLength(3);
```

```
builder.Property(p => p.nombreEstado)
    .IsRequired()
    .HasMaxLength(50);
```

La línea **builder.Property(p => p.nombreEstado)** se utiliza para configurar la propiedad "nombreEstado" de la entidad "Estado". **.IsRequired()** indica que el valor de la propiedad es obligatorio y **.HasMaxLength(50)** establece la longitud máxima de caracteres permitidos para el nombre del estado.

```
builder.Property(p => p.nombreEstado)
    .IsRequired()
    .HasMaxLength(50);
```

```

builder.HasOne(p => p.Pais)
    .WithMany(p => p.Estados)
    .HasForeignKey(p => p.codPais);

```

La línea **builder.HasOne(p => p.Pais)**

establece una relación uno a muchos entre la entidad "Estado" y la entidad "Pais". Indica que un estado está asociado a un único país. **.WithMany(p => p.Estados)** especifica que la propiedad de navegación "Estados" en la entidad "Pais" representa la colección de estados relacionados. **.HasForeignKey(p => p.codPais)** establece la clave foránea en la entidad "Estado" que hace referencia al código del país.

```

builder.HasOne(p => p.Pais)
    .WithMany(p => p.Estados)
    .HasForeignKey(p => p.codPais);

```

En resumen, el código representa la configuración de la entidad "Estado" utilizando Entity Framework Core. Define el nombre de la tabla, las propiedades y sus restricciones, y establece la relación uno a muchos con la entidad "Pais".

Relación Muchos a Muchos

La relación muchos a muchos se refieren a una relación entre dos entidades donde múltiples instancias de una entidad están relacionadas con múltiples instancias de la otra entidad. En este tipo de relación, una entidad puede estar asociada con varias instancias de la otra entidad y viceversa.

Para implementar una relación muchos a muchos, generalmente se utiliza una tabla de unión que actúa como un enlace entre las dos entidades principales. Esta tabla de unión almacena las claves primarias de las dos entidades principales, estableciendo las conexiones entre ellas.



Como se puede observar en la gráfica anterior existe una relación de muchos a muchos entre la tabla persona y producto; y según las reglas de normalización de bases de datos este tipo de relaciones se deben romper con una table intermedia cuyo nombre se componen de los nombres de las dos tablas principales.

Clase Persona

```
1  using System.ComponentModel.DataAnnotations;
2  namespace Core.Entities;
3  public class Persona
4  {
5      [Key]
6      public string ? IdPersona { get; set; }
7      public string ? NombrePersona { get; set; }
8      public string ? ApellidosPersona { get; set; }
9      public string ? EmailPersona { get; set; }
10     public int IdTipoPersona { get; set; }
11     public TipoPersona ? TipoPersona{ get; set; }
12     public string ? IdRegion { get; set; }
13     public Region ? Region{ get; set; }
14     public ICollection<PersonaProducto>? PersonaProductos { get; set; }
15 }
```

using System.ComponentModel.DataAnnotations;

namespace Core.Entities;

public class Persona

{

[Key]

public string ? IdPersona { get; set; }

public string ? NombrePersona { get; set; }

public string ? ApellidosPersona { get; set; }

public string ? EmailPersona { get; set; }

public int IdTipoPersona { get; set; }

public TipoPersona ? TipoPersona { get; set; }

public string ? IdRegion { get; set; }

public Region ? Region{ get; set; }

public ICollection<PersonaProducto>? PersonaProductos { get; set; }

}

```
[Key]
```

```
public string? IdPersona { get; set; }
```

El atributo **[Key]** indica que la propiedad "IdPersona" es la clave primaria de la entidad. Es de tipo **string** y permite valores nulos (**string?**).

```
[Key]
```

```
public string ? IdPersona { get; set; }
```

```
public TipoPersona? TipoPersona{ get; set; }
```

La propiedad "TipoPersona" es de tipo **TipoPersona** (otra entidad) y representa el tipo de persona al que pertenece. El signo de interrogación (**TipoPersona?**) indica que esta propiedad puede ser nula.

```
public TipoPersona ? TipoPersona { get; set; }
```

```
public string? IdRegion { get; set; }
```

La propiedad "IdRegion" representa el identificador de la región a la que pertenece la persona y también permite valores nulos.

```
public string ? IdRegion { get; set; }
```

```
public Region? Region{ get; set; }
```

La propiedad "Region" es de tipo **Region** (otra entidad) y representa la región a la que pertenece la persona. El signo de interrogación (**Region?**) indica que esta propiedad puede ser nula.

```
public Region? Region{ get; set; }
```

```
public ICollection<PersonaProducto>? PersonaProductos { get; set; }
```

La propiedad "PersonaProductos" es una colección (**ICollection**) de objetos de tipo **PersonaProducto** y representa los productos asociados a la persona. El signo de interrogación (**ICollection<PersonaProducto>?**) indica que esta propiedad puede ser nula.

```
public ICollection<PersonaProducto>? PersonaProductos { get; set; }
```

En resumen, la clase "Persona" representa información de una persona, como su nombre, apellidos, correo electrónico, tipo de persona, región y los productos asociados a ella.

Clase Producto

```
1  using System.ComponentModel.DataAnnotations;
2  namespace Core.Entities;
3
4      4 references
5  public class Producto
6  {
7      [Key]
8          1 reference
9      public string? IdProducto { get; set; }
10         1 reference
11      public string? NombreProducto { get; set; }
12         1 reference
13      public string? Descripcion { get; set; }
14         1 reference
15      public double Precio { get; set; }
16         1 reference
17      public int StockMinimo { get; set; }
18         1 reference
19      public int StockMaximo { get; set; }
20         1 reference
21      public ICollection<PersonaProducto>? PersonaProductos { get; set; }
22
23 }
```

```
using System.ComponentModel.DataAnnotations;
namespace Core.Entities;

public class Producto
{
    [Key]
    public string? IdProducto { get; set; }

    public string? NombreProducto { get; set; }

    public string? Descripcion { get; set; }

    public double Precio { get; set; }

    public int StockMinimo { get; set; }

    public int StockMaximo { get; set; }

    public ICollection<PersonaProducto>? PersonaProductos { get; set; }
}
```

```
[Key]
public string? IdProducto { get; set; }
```

El atributo **[Key]** indica que la propiedad "IdProducto" es la clave primaria de la entidad. Es de tipo **string** y permite valores nulos (**string?**).

[Key]

```
public string ? IdProducto { get; set; }
```

```
public ICollection<PersonaProducto>? PersonaProductos { get; set; }
```

La propiedad "PersonaProductos" es una colección (**ICollection**) de objetos de tipo **PersonaProducto** y representa las personas asociadas al producto. El signo de interrogación (**ICollection<PersonaProducto>?**) indica que esta propiedad puede ser nula.

```
public ICollection<PersonaProducto>? PersonaProductos { get; set; }
```

En resumen, la clase "Producto" representa información de un producto, como su nombre, descripción, precio, stock mínimo y máximo, y las personas asociadas a él.

PersonaProducto (Tabla Intermedia)

9 }

namespace Core.Entities;

```
public class PersonaProducto
{
    public string? IdPersona { get; set; }
    public Persona? Personas { get; set; }
    public string? IdProducto { get; set; }
    public Producto? Productos { get; set; }
}
```

public string? IdPersona { get; set; } La propiedad "IdPersona" representa el identificador de la persona en la relación entre "Persona" y "Producto". Es de tipo **string** y permite valores nulos (**string?**).

```
public string? IdPersona { get; set; }
```

public Persona? Personas { get; set; } La propiedad "Personas" es de tipo **Persona** (otra entidad) y representa la referencia a la entidad "Persona" en la relación entre "Persona" y "Producto". El signo de interrogación (**Persona?**) indica que esta propiedad puede ser nula.

```
public Persona? Personas { get; set; }
```

public string? IdProducto { get; set; } La propiedad "IdProducto" representa el identificador del producto en la relación entre "Persona" y "Producto". Es de tipo **string** y permite valores nulos (**string?**).

```
public string? IdProducto { get; set; }
```

```
public Producto? Productos { get; set; }
```

La propiedad "Productos" es de tipo **Producto** (otra entidad) y representa la referencia a la entidad "Producto" en la relación entre "Persona" y "Producto". El signo de interrogación (**Producto?**) indica que esta propiedad puede ser nula.

```
public Producto? Productos { get; set; }
```

En resumen, la clase "PersonaProducto" representa la relación entre una entidad "Persona" y una entidad "Producto". Tiene propiedades que representan los identificadores de la persona y el producto, así como referencias a los objetos "Persona" y "Producto" en sí.

Se deben crear las clases de configuración de las clases persona, producto y personaproducto.

```
1  using Core.Entities;
2  using Microsoft.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore.Metadata.Builders;
4  namespace Infrastructure.Data.Configuration;
5  public class PersonaConfiguration : IEntityTypeConfiguration<Persona>
6  {
7      public void Configure(EntityTypeBuilder<Persona> builder)
8  {
9      builder.ToTable("Persona");
10     builder.Property(p => p.IdPersona)
11         .IsRequired()
12         .HasMaxLength(15);
13
14     builder.Property(p => p.NombrePersona)
15         .IsRequired()
16         .HasMaxLength(50);
17
18     builder.Property(p => p.ApellidosPersona)
19         .IsRequired()
20         .HasMaxLength(50);
21
22     builder.Property(p => p.EmailPersona)
23         .IsRequired()
24         .HasMaxLength(100);
25
26     builder.HasOne(p => p.TipoPersona)
27         .WithMany(e => e.Personas)
28         .HasForeignKey(f => f.IdTipoPersona);
29
30     builder.HasOne(p => p.Region)
31         .WithMany(r => r.Personas)
32         .HasForeignKey(f => f.IdRegion);
33
34 }
35 }
```

```
public class PersonaConfiguration : IEntityTypeConfiguration<Persona>
```

```
{
```

```
    public void Configure(EntityTypeBuilder<Persona> builder)
```

```
{
```

```
        builder.ToTable("Persona");
```

```
        builder.Property(p => p.IdPersona)
```

```
            .IsRequired()
```

```
            .HasMaxLength(15);
```

```
        builder.Property(p => p.NombrePersona)
```

```
            .IsRequired()
```

```
            .HasMaxLength(50);
```

```
        builder.Property(p => p.ApellidosPersona)
```

```
            .IsRequired()
```

```

.HasMaxLength(50);

builder.Property(p => p.EmailPersona)
    .IsRequired()
    .HasMaxLength(100);

builderhasOne(p => p.TipoPersona)
    .WithMany(e => e.Personas)
    .HasForeignKey(f => f.IdTipoPersona);

builderhasOne(p => p.Region)
    .WithMany(r => r.Personas)
    .HasForeignKey(f => f.IdRegion);

}
}

```

```

1  using Core.Entities;
2  using Microsoft.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore.Metadata.Builders;
4
5  namespace Infrastructure.Data.Configuration;
6
7  0 references
8  public class ProductoConfiguration : IEntityTypeConfiguration<Producto>
9  {
10     0 references
11     public void Configure(EntityTypeBuilder<Producto> builder)
12     {
13         builder.ToTable("Producto");
14
15         builder.Property(p => p.IdProducto)
16             .IsRequired()
17             .HasMaxLength(10);
18
19         builder.Property(p => p.NombreProducto)
20             .IsRequired()
21             .HasMaxLength(50);
22
23         builder.Property(p => p.Descripcion)
24             .HasColumnType("text");
25
26         builder.Property(p => p.Precio)
27             .HasColumnType("double");
28
29         builder.Property(p => p.StockMaximo)
30             .HasColumnType("int");
31
32         builder.Property(p => p.StockMinimo)
33             .HasColumnType("int");
34     }
35 }

```

using Core.Entities;

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Infrastructure.Data.Configuration;

public class ProductoConfiguration : IEntityTypeConfiguration<Producto>
{
    public void Configure(EntityTypeBuilder<Producto> builder)
    {
        builder.ToTable("Producto");

        builder.Property(p => p.IdProducto)
            .IsRequired()
            .HasMaxLength(10);

        builder.Property(p => p.NombreProducto)
            .IsRequired()
            .HasMaxLength(50);

        builder.Property(p => p.Descripcion)
            .HasColumnType("text");

        builder.Property(p => p.Precio)
            .HasColumnType("double");

        builder.Property(p => p.StockMaximo)
            .HasColumnType("int");

        builder.Property(p => p.StockMinimo)
            .HasColumnType("int");
    }
}
```

```
    }  
}  
  
}
```

```
1  using Core.Entities;  
2  using Microsoft.EntityFrameworkCore;  
3  using Microsoft.EntityFrameworkCore.Metadata.Builders;  
4  
5  namespace Infrastructure.Data.Configuration;  
6  
7  public class PersonaProductoConfiguration : IEntityTypeConfiguration<PersonaProducto>  
8  {  
9      public void Configure(EntityTypeBuilder<PersonaProducto> builder)  
10     {  
11         builder.ToTable("PersonaProducto");  
12         builder.HasOne(p => p.Personas)  
13             .WithMany(p => p.PersonaProductos)  
14             .HasForeignKey(f => f.IdPersona);  
15  
16         builder.HasOne(p => p.Productos)  
17             .WithMany(p => p.PersonaProductos)  
18             .HasForeignKey(f => f.IdProducto);  
19  
20     }  
21 }
```

```
using Core.Entities;  
using Microsoft.EntityFrameworkCore;  
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

```
namespace Infrastructure.Data.Configuration;
```

```
public class PersonaProductoConfiguration :  
IEntityTypeConfiguration<PersonaProducto>  
{  
    public void Configure(EntityTypeBuilder<PersonaProducto> builder)  
    {  
        builder.ToTable("PersonaProducto");  
  
        builder.HasOne(p => p.Personas)  
            .WithMany(p => p.PersonaProductos)  
            .HasForeignKey(f => f.IdPersona);  
  
        builder.HasOne(p => p.Productos)  
            .WithMany(p => p.PersonaProductos)  
            .HasForeignKey(f => f.IdProducto);  
    }  
}
```

```
    }  
}  
}
```

En las tablas intermedias las llaves foraneas forman la clave principal de la tabla. Para poder establecer la llave principal se debe realizar usando el modelBuilder. En la clase contexto agregue el siguiente código.

```
1 reference  
protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
    modelBuilder.Entity<Persona>().HasIndex(idx => idx.EmailPersona).IsUnique();  
    modelBuilder.Entity<PersonaProducto>().HasKey(r => new {r.IdPersona, r.IdProducto});  
    base.OnModelCreating(modelBuilder);  
    modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());  
}
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
    modelBuilder.Entity<Persona>().HasIndex(idx => idx.EmailPersona).IsUnique();  
    modelBuilder.Entity<PersonaProducto>().HasKey(r => new {r.IdPersona, r.IdProducto});  
    base.OnModelCreating(modelBuilder);  
  
    modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());  
}
```

3.4. Proyecto Base (Habilitando Migraciones)

Las migraciones en el contexto de desarrollo de bases de datos son una forma de llevar a cabo cambios estructurales en un esquema de base de datos de manera controlada y reproducible. Las migraciones son una parte esencial de los sistemas de control de versiones y permiten mantener un historial de los cambios realizados en la estructura de la base de datos a lo largo del tiempo.

Cuando trabajas en un proyecto que utiliza un ORM (Object-Relational Mapping) como Entity Framework Core en el ecosistema .NET, las migraciones se utilizan para crear, modificar o eliminar tablas, columnas, restricciones y otros objetos relacionados en la base de datos.

Las migraciones ofrecen varias ventajas:

- **Control de versiones:** Las migraciones proporcionan un mecanismo para mantener un historial de los cambios en el esquema de la base de datos. Cada

migración es una versión del esquema y se pueden aplicar o revertir para llevar la base de datos a un estado específico en cualquier momento.

- **Desarrollo colaborativo:** Las migraciones facilitan el trabajo en equipo, ya que cada desarrollador puede generar y aplicar sus propias migraciones en su entorno de desarrollo. Luego, las migraciones se pueden combinar y aplicar en otros entornos.
- **Reproducibilidad:** Al utilizar migraciones, es posible automatizar el proceso de implementación de cambios en la base de datos. Esto permite que cualquier persona pueda ejecutar las migraciones en su entorno y obtener la misma estructura de base de datos.
- **Pruebas y despliegue:** Las migraciones facilitan la gestión de la estructura de la base de datos en diferentes entornos, como entornos de pruebas y producción. Puedes aplicar las migraciones necesarias en cada entorno de manera controlada y realizar cambios sin tener que recrear la base de datos por completo.

Al generar una migración, se crea un archivo de código que representa los cambios realizados en el modelo de datos. Luego, puedes aplicar esa migración a la base de datos utilizando comandos como **dotnet ef database update**. Esto ejecutará el código de la migración y realizará los cambios necesarios en la base de datos.

Para habilitar las migraciones en el proyecto debemos verificar que se encuentren instaladas las herramientas de migración de entity framework para esto ejecutamos el comando: `dotnet tool list -g`

```
PS D:\projectsNetCore\tienda\Infrastructure> dotnet tool list -g
Id. de paquete      Versión      Comandos
-----
PS D:\projectsNetCore\tienda\Infrastructure> |
```

Como vemos en la gráfica anterior no contamos con ninguna herramienta de entity framework instalada en el proyecto, restaurar las herramientas debemos ejecutar el siguiente comando: `dotnet tool install --global dotnet-ef`.

```
PS D:\projectsNetCore\tienda\Infrastructure> dotnet tool install --global dotnet-ef
Puede invocar la herramienta con el comando siguiente: dotnet-ef
La herramienta "dotnet-ef" (versión '7.0.8') se instaló correctamente.
```

Si volvemos a ingresar el comando de verificación de paquetes podemos observar que ya aparecen los paquetes recién instalados:

```
PS D:\projectsNetCore\tienda\Infrastructure> dotnet tool list -g
Id. de paquete      Versión      Comandos
-----
dotnet-ef          7.0.8        dotnet-ef ←
PS D:\projectsNetCore\tienda\Infrastructure>
```

```

PS D:\projectsNetCore\tienda\Infrastructure> dotnet tool list -g
Id. de paquete      Versión     Comandos
-----
dotnet-ef           7.0.8       dotnet-ef
PS D:\projectsNetCore\tienda\Infrastructure> dotnet-ef -h
Entity Framework Core .NET Command-line Tools 7.0.8

Usage: dotnet ef [options] [command]

Options:
  --version          Show version information
  -h|--help          Show help information
  -v|--verbose       Show verbose output.
  --no-color         Don't colorize output.
  --prefix-output    Prefix output with level.

Commands:
  database   Commands to manage the database.
  dbcontext  Commands to manage DbContext types.
  migrations Commands to manage migrations. ←

Use "dotnet ef [command] --help" for more information about a command.
PS D:\projectsNetCore\tienda\Infrastructure>

```

```

PS D:\projectsNetCore\tienda\Infrastructure> dotnet-ef migrations -h
                                                               ←

Usage: dotnet ef migrations [options] [command]

Options:
  -h|--help          Show help information
  -v|--verbose       Show verbose output.
  --no-color         Don't colorize output.
  --prefix-output    Prefix output with level.

Commands:
  add    Adds a new migration.
  bundle Creates an executable to update the database.
  list   Lists available migrations.
  remove Removes the last migration.
  script Generates a SQL script from migrations.

Use "migrations [command] --help" for more information about a command.
PS D:\projectsNetCore\tienda\Infrastructure>

```

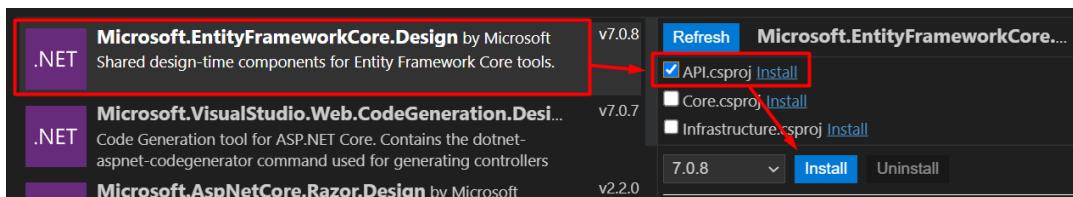
Para poder habilitar las migraciones debemos encontrarnos dentro de la carpeta principal de tienda y ejecutar el siguiente comando: dotnet ef migrations add InitialCreate --project ./Infrastructure/ --startup-project ./API/ --output-dir ./Data/Migrations

```

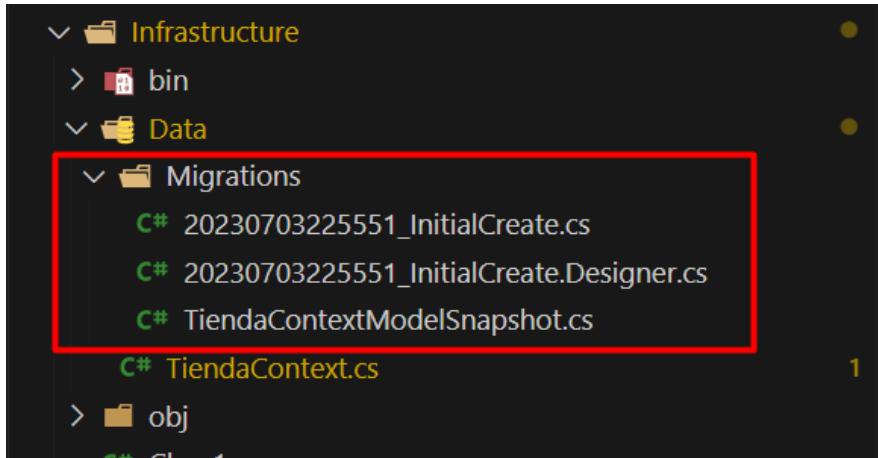
PS D:\projectsNetCore\tienda> dotnet ef migrations add InitialCreate --project ./Infrastructure/ --startup-project ./API/ --output-dir ./Data/Migrations
Build started...
Build succeeded.
Your startup project 'API' doesn't reference Microsoft.EntityFrameworkCore.Design. This package is required for the Entity Framework Core Tools to work. Ensure your startup project is correct, install the package, and try again.
PS D:\projectsNetCore\tienda>

```

Es importante tener en cuenta que no hay imagen anterior se genera un error ya que no existe referencia al entity framework. Design para corregir este error es necesario instalar este paquete. la instalación se debe hacer en la carpeta api que es donde se está indicando la falta de dicha referencia.



Después de haber instalado el paquete debemos ejecutar nuevamente el comando `dotnet ef migrations add InitialCreate --project ./Infrastructure/ --startup-project ./API/ --output-dir ./Data/Migrations` y obtendremos como salida lo siguiente:



`C# 20230703225551_InitialCreate.cs` este archivo contiene todas las sentencias que son necesarias para aplicar las migraciones a la base de datos.

`C# 20230703225551_InitialCreate.Designer.cs` este archivo contiene metadatos que será utilizado por el entity framework.

`C# TiendaContextModelSnapshot.cs` este archivo contiene una imagen o instantánea de la migración actual, este archivo es utilizado cuando se realizan nuevas migraciones para verificar cambios en la estructura del modelo de datos.

En este momento las migraciones no se han aplicado al servidor de base de datos, para poder aplicar las migraciones a la estructura de la base de datos siga los siguientes pasos:

- desde la línea de comandos en la carpeta principal tienda ejecuta el comando `dotnet build`

```
PS D:\projectsNetCore\tienda> dotnet build ←
MSBuild version 17.6.1+8ffc3fe3d for .NET
Determinando los proyectos que se van a restaurar...
Todos los proyectos están actualizados para la restauración.
Core -> D:\projectsNetCore\tienda\Core\bin\Debug\net7.0\Core.dll
Infrastructure -> D:\projectsNetCore\tienda\Infrastructure\bin\Debug\net7.0\Infrastructure.dll
API -> D:\projectsNetCore\tienda\API\bin\Debug\net7.0\API.dll

Compilación correcta.
0 Advertencia(s) ←
0 Errores

Tiempo transcurrido 00:00:01.07
PS D:\projectsNetCore\tienda>
```

- Para aplicar la migración a la base de datos ejecuta el comando: `dotnet ef database update --project ./Infrastructure/ --startup-project ./API/`

```
PS D:\projectsNetCore\tienda> dotnet ef database update --project ./Infrastructure/ --startup-project ./API/
Build started...
Build succeeded.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE DATABASE 'tiendadb';
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
```

Como resultado final podemos verificar en el que phpmyadmin los cambios registrados en el servidor de base de datos.

The screenshot shows the phpMyAdmin interface with the database 'tiendadb' selected. In the left sidebar, under the 'Base de datos: tiendadb' section, there is a tree view with several schemas and tables. One table, 'productos', is highlighted with a red border. A red arrow points from this highlighted table to the main content area where a list of tables is displayed. The list includes 'productos' and '_efmigrationshistory'. The '_efmigrationshistory' table has a single row with the value '1'. At the bottom of the interface, there is a 'Crear nueva tabla' (Create new table) button.

Cómo podemos observar en la imagen anterior se creó la base de datos tiendadb y la tabla productos.

En .net también podemos ejecutar migraciones pendientes cuando ejecutamos el proyecto ahora esto siga los siguientes pasos:

- En el archivo **C# Program.cs** que se encuentra en la carpeta api agregue las

```

26 // Configure the HTTP request pipeline.
27 if (app.Environment.IsDevelopment())
28 {
29     app.UseSwagger();
30     app.UseSwaggerUI();
31 }

```

siguientes líneas de código después de:

// Configure the HTTP request pipeline.

```
if(app.Environment.IsDevelopment())
```

```
{
```

```
    app.UseSwagger();
```

```
    app.UseSwaggerUI();
```

```
}
```

```

using(var scope= app.Services.CreateScope()){
    var services = scope.ServiceProvider;
    var loggerFactory = services.GetRequiredService<ILoggerFactory>();
    try{
        var context = services.GetRequiredService<TiendaContext>();
        await context.Database.MigrateAsync();
    }
    catch(Exception ex){
        var logger = loggerFactory.CreateLogger<Program>();
        logger.LogError(ex,"Ocurrió un error durante la migración");
    }
}

```

```

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

using (var scope = app.Services.CreateScope()){
    var services = scope.ServiceProvider;
    var loggerFactory = services.GetRequiredService<ILoggerFactory>();
    try{
        var context = services.GetRequiredService<TiendaContext>();
        await context.Database.MigrateAsync();
    }
    catch(Exception ex){
        var logger = loggerFactory.CreateLogger<Program>();
        logger.LogError(ex."Ocurrió un error durante la migración");
    }
}

app.UseHttpsRedirection();

```

Como ya se aplicó una migración anteriormente se recomienda eliminar la base de datos del servidor mysql para ejecutar el siguiente comando: dotnet run y así poder ejecutar la aplicación. La ejecución de este comando se debe realizar desde la carpeta api.

Como podemos observar en la imagen anterior al momento de ejecutar el comando automáticamente se ejecuta el comando de migraciones y hace actualizar la base de datos en el servidor teniendo en cuenta los cambios realizados en los modelos de datos.

3.5. Proyecto Base (Fluent Api)

La Fluent Api de Entity Framework se usa para configurar las clases de dominio y sobre escribir sus convenciones. Ef Fluent Api se basa en el patrón de diseño Fluent API (también conocido como Fluent Interface) donde el resultado es formulado mediante el encadenamiento de métodos.

En entity Framework Core, la clase modelBuilder actúa como una api fluida, ya que permite utilizar diferentes métodos y brindar diferentes opciones de configuración en sus atributos.

proporciona más opciones de configuración que los atributos de anotación de datos.

Entity Framework Core Fluent API configura los siguientes aspectos de un modelo:

1. Configuración del modelo: configura un modelo EF para asignaciones de base de datos. Configura el esquema predeterminado, las funciones de base de datos, los atributos de anotación de datos adicionales y las entidades que se excluirán del mapeo.
2. Configuración de la entidad: Configura la entidad a la tabla y el mapeo de relaciones, por ejemplo. PrimaryKey, AlternateKey, Index, nombre de tabla, relaciones uno a uno, uno a muchos, muchos a muchos, etc. nombre de columna, predeterminado
3. Configuración de propiedades: configura la propiedad para la asignación de columnas, p. nombre de columna, valor predeterminado, nulabilidad, clave externa, tipo de datos, columna de concurrencia, etc.

En la siguiente lista se pueden visualizar los métodos más importantes para cada tipo de configuración.

Configuraciones	Método Fluent API	Uso
Configuración de Modelo Model Configurations	HasDbFunction()	Configura una función de base de datos cuando se dirige a una base de datos relacional.
	HasDefaultSchema()	Especifica el esquema de la base de datos
	HasAnnotation()	Agrega y actualiza actualiza atributos de anotación de datos en la entidad
	HasSequence()	Configura una secuencia de base de datos cuando se dirige a una base de datos relacional.
Entity Configuration	HasAlternateKey()	Configura una clave alternativa en el modelo EF para la entidad.
	HasIndex()	Configura un índice de las propiedades especificadas.
	HasKey()	Configura la propiedad o enumera las propiedades como clave principal.
	HasMany()	Configura la parte Muchos de la relación, donde una entidad contiene la propiedad de colección de referencia de otro tipo para relaciones de uno a varios o de varios a varios.
	HasOne()	Configura la parte Uno de la relación, donde una entidad contiene la propiedad de referencia de otro tipo para relaciones uno a uno o uno a muchos.
	Ignore()	Configura que la clase o propiedad no se debe asignar a una tabla o columna.
	OwnsOne()	Configura una relación donde la entidad de destino es propiedad de esta entidad. El valor de la clave de la entidad de destino se propaga desde la entidad a la que pertenece.
Property Configuration	ToTable()	Configura la tabla de la base de datos a la que se asigna la entidad.
	HasColumnName()	Configura el nombre de la columna correspondiente en la base de datos para la propiedad.
	HasColumnType()	Configura el tipo de datos de la columna correspondiente en la base de datos para la propiedad.
	HasComputedColumnSql()	Configura la propiedad para que se asigne a la columna calculada en la base de datos

		cuando se dirige a una base de datos relacional.
	HasDefaultValue()	Configura el valor predeterminado para la columna a la que se asigna la propiedad cuando se dirige a una base de datos relacional.
	HasDefaultValueSql()	Configura la expresión de valor predeterminada para la columna a la que se asigna la propiedad cuando se dirige a una base de datos relacional.
	HasField()	Especifica el campo de respaldo que se utilizará con una propiedad.
	HasMaxLength()	Configura la longitud máxima de datos que se pueden almacenar en una propiedad.
	IsConcurrencyToken()	Configura la propiedad para que se use como token de simultaneidad optimista.
	IsRequired()	Configura si se requiere el valor válido de la propiedad o si nulo es un valor válido
	IsRowVersion()	Configura la propiedad que se utilizará en la detección de simultaneidad optimista.
	IsUnicode()	Configura la propiedad de cadena que puede contener caracteres Unicode o no.
	ValueGeneratedNever()	Configura una propiedad que no puede tener un valor generado cuando se guarda una entidad.
	ValueGeneratedOnAdd()	Configura que la propiedad ha generado valor al guardar una nueva entidad
	ValueGeneratedOnAddOrUpdate()	Configura que la propiedad ha generado valor al guardar nueva entidad existente.
	ValueGeneratedOnUpdate()	Configura que propiedad tiene un valor generado al guardar una entidad existente.

3.5.1. Entity Mappings

Fluent Api se puede utilizar para configurar una entidad para asignarla con tablas de bases de datos, esquemas predeterminados, etc.

Primero, configuremos un esquema predeterminado para las tablas de la base de datos. Sin embargo, puede cambiar el esquema mientras crea las tablas individuales. El siguiente ejemplo establece el esquema de administración como esquema predeterminado. Todos los objetos de la base de datos se crearán bajo el esquema de administración a menos que especifique un esquema diferente explícitamente.

```

public class SchoolContext: DbContext
{
    public SchoolDBContext(): base()
    {
    }

    public DbSet<Student> Students { get; set; }
    public DbSet<Standard> Standards { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        //Configure default schema
        modelBuilder.HasDefaultSchema("Admin");
    }
}

```

Fuente: <https://www.entityframeworktutorial.net/code-first/configure-entity-mappings-using-fluent-api.aspx>

```

public class SchoolContext: DbContext
{
    public SchoolDBContext(): base()
    {
    }

    public DbSet<Student> Students { get; set; }
    public DbSet<Standard> Standards { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        //Configure default schema
        modelBuilder.HasDefaultSchema("Admin");
    }
}

```

3.5.1.1. Mapeando entidades a Tablas

Code First creara las tablas de la base de datos con el nombre de las propiedades del DbSet que se encuentran en la clase de contexto. Se puede anular esta convencion y dar un nombre de tabla diferente al de las propiedades del DbSet, como podemos observar en el ejemplo de la siguiente imagen.

```

namespace CodeFirst_FluentAPI_Tutorials
{
    public class SchoolContext: DbContext
    {
        public SchoolDBContext(): base()
        {

        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Standard> Standards { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            //Configure default schema
            modelBuilder.HasDefaultSchema("Admin");

            //Map entity to table
            modelBuilder.Entity<Student>().ToTable("StudentInfo");
            modelBuilder.Entity<Standard>().ToTable("StandardInfo", "dbo");
        }
    }
}

```

Fuente : <https://www.entityframeworktutorial.net/code-first/configure-entity-mappings-using-fluent-api.aspx>

namespace CodeFirst_FluentAPI_Tutorials

```

{
    public class SchoolContext: DbContext
    {
        public SchoolDBContext(): base()
        {

        }
    }
}
```

```

public DbSet<Student> Students { get; set; }
public DbSet<Standard> Standards { get; set; }
```

protected override void OnModelCreating(DbModelBuilder modelBuilder)

```

{
    //Configure default schema
    modelBuilder.HasDefaultSchema("Admin");
```

```

    //Map entity to table
    modelBuilder.Entity<Student>().ToTable("StudentInfo");
```

```

        modelBuilder.Entity<Standard>().ToTable("StandardInfo","dbo");
    }
}

}

```

3.5.1.2. Mapeando multiples Tablas

```

namespace CodeFirst_FluentAPI_Tutorials
{
    public class SchoolContext: DbContext
    {
        public SchoolDBContext(): base()
        {

        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Standard> Standards { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Student>().Map(m =>
            {
                m.Properties(p => new { p.StudentId, p.StudentName });
                m.ToTable("StudentInfo");
            }).Map(m => {
                m.Properties(p => new { p.StudentId, p.Height, p.Weight, p.Photo, p.DateOfBirth });
                m.ToTable("StudentInfoDetail");
            });

            modelBuilder.Entity<Standard>().ToTable("StandardInfo");
        }
    }
}

```

Fuente: <https://www.entityframeworktutorial.net/code-first/configure-entity-mappings-using-fluent-api.aspx>

```

namespace CodeFirst_FluentAPI_Tutorials
{
    public class SchoolContext: DbContext
    {
        public SchoolDBContext(): base()
        {

        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Standard> Standards { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Student>().Map(m =>
            {
                m.Properties(p => new { p.StudentId, p.StudentName });
                m.ToTable("StudentInfo");
            }).Map(m => {
                m.Properties(p => new { p.StudentId, p.Height, p.Weight, p.Photo,
                p.DateOfBirth });
                m.ToTable("StudentInfoDetail");
            });

            modelBuilder.Entity<Standard>().ToTable("StandardInfo");
        }
    }
}

```

3.5.1.3 Mapeo de propiedades usando Fluent API

La API Fluent se puede utilizar para configurar las propiedades de una entidad para asignarla con una columna de base de datos. Con Fluent API, puede cambiar el nombre

de la columna correspondiente, el tipo, el tamaño, Null o NotNull, PrimaryKey, ForeignKey, columna de simultaneidad, etc.

```
public class Student
{
    public int StudentKey { get; set; }
    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public byte[] Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }

    public Standard Standard { get; set; }
}

public class Standard
{
    public int StandardKey { get; set; }
    public string StandardName { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

Configure Primary Key and Composite Primary Key

```
public class SchoolContext: DbContext
{
    public SchoolDBContext(): base()
    {

        public DbSet<Student> Students { get; set; }
        public DbSet<Standard> Standards { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            //Configure primary key
            modelBuilder.Entity<Student>().HasKey<int>(s => s.StudentKey);
            modelBuilder.Entity<Standard>().HasKey<int>(s => s.StandardKey);

            //Configure composite primary key
            modelBuilder.Entity<Student>().HasKey<int>(s => new { s.StudentKey, s.StudentName });
        }
}
```

Fuente : <https://www.entityframeworktutorial.net/code-first/configure-property-mappings-using-fluent-api.aspx>

```
public class SchoolContext: DbContext
{
    public SchoolDBContext(): base()
```

```

{
}

public DbSet<Student> Students { get; set; }

public DbSet<Standard> Standards { get; set; }

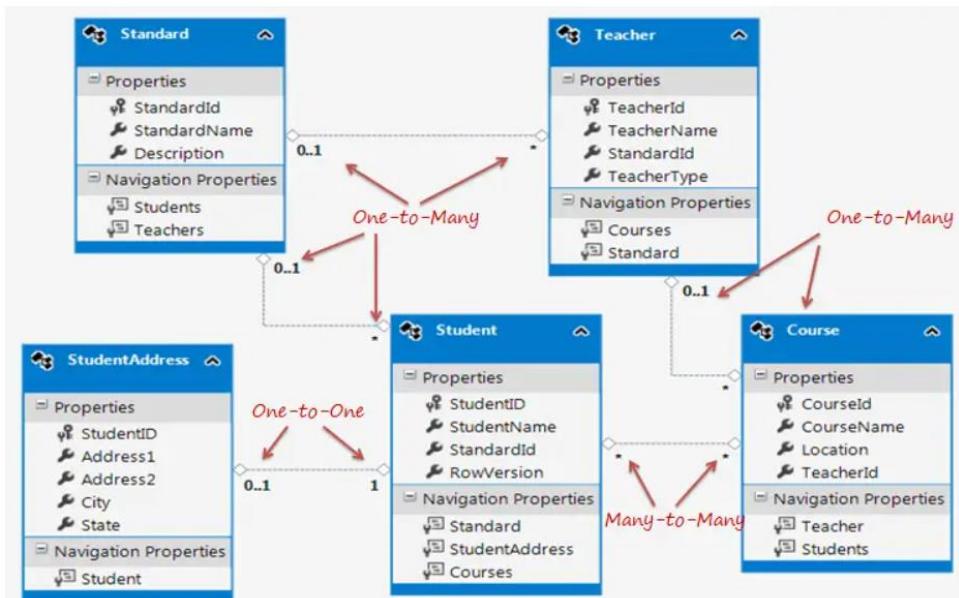
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    //Configure primary key
    modelBuilder.Entity<Student>().HasKey<int>(s => s.StudentKey);
    modelBuilder.Entity<Standard>().HasKey<int>(s => s.StandardKey);

    //Configure composite primary key
    modelBuilder.Entity<Student>().HasKey<int>(s => new { s.StudentKey,
s.StudentName });

}
}

```

3.5.2. Relaciones



3.5.2.1. One to One

Como puede ver en la figura anterior, Student y StudentAddress tienen una relación uno a uno (cero o uno). Un estudiante puede tener sólo una o ninguna dirección. El marco de la entidad agrega la propiedad de navegación de referencia de Estudiante a la entidad StudentAddress y la entidad de navegación StudentAddress a la entidad Student. Además, la entidad StudentAddress tiene la propiedad StudentId como PrimaryKey y ForeignKey, lo que la convierte en una relación uno a uno.

3.6. Repositorios, Unidad de trabajo y métodos de extensión

3.6.1. Repositorios

Los repositorios son componentes que se utilizan para acceder y manipular datos almacenados en una fuente de datos, como una base de datos. Los repositorios actúan como una capa de abstracción entre la lógica de negocio de la API y los detalles específicos de cómo se accede y se almacenan los datos.

Los repositorios se utilizan comúnmente para implementar el patrón de diseño Repository en el desarrollo de aplicaciones. Este patrón promueve la separación de responsabilidades al definir una interfaz común para el acceso a datos y proporcionar implementaciones concretas que interactúan con la fuente de datos subyacente.

En el contexto de una API web, los repositorios pueden proporcionar métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en los datos, como crear un nuevo registro, obtener un registro por su identificador, actualizar un registro existente o eliminar un registro de la base de datos. Estos métodos encapsulan la lógica necesaria para realizar las operaciones en la fuente de datos subyacente.

Además, los repositorios también pueden aplicar lógica adicional, como filtrar datos, realizar paginación, ordenar resultados, etc. También pueden manejar la conexión y el manejo de transacciones con la base de datos.

El uso de repositorios en una API web ayuda a mantener un código más limpio y modular, ya que separa las operaciones de acceso a datos de la lógica de negocio y facilita las pruebas unitarias, ya que se pueden simular los repositorios para probar la lógica sin acceder a una base de datos real.

Es importante destacar que los repositorios son una opción arquitectónica, pero no son una parte obligatoria de una API web. Su uso depende de los requisitos y la complejidad de tu aplicación.

3.6.2. Unidades de trabajo

Las unidades de trabajo (también conocidas como "UnitOfWork") son componentes que se utilizan para agrupar y coordinar operaciones relacionadas con la persistencia de datos. Proporcionan una abstracción de más alto nivel que encapsula las transacciones y las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos.

El patrón de Unidad de Trabajo se utiliza comúnmente junto con el patrón de Repositorio para manejar la interacción entre la lógica de negocio de la API y la capa de acceso a

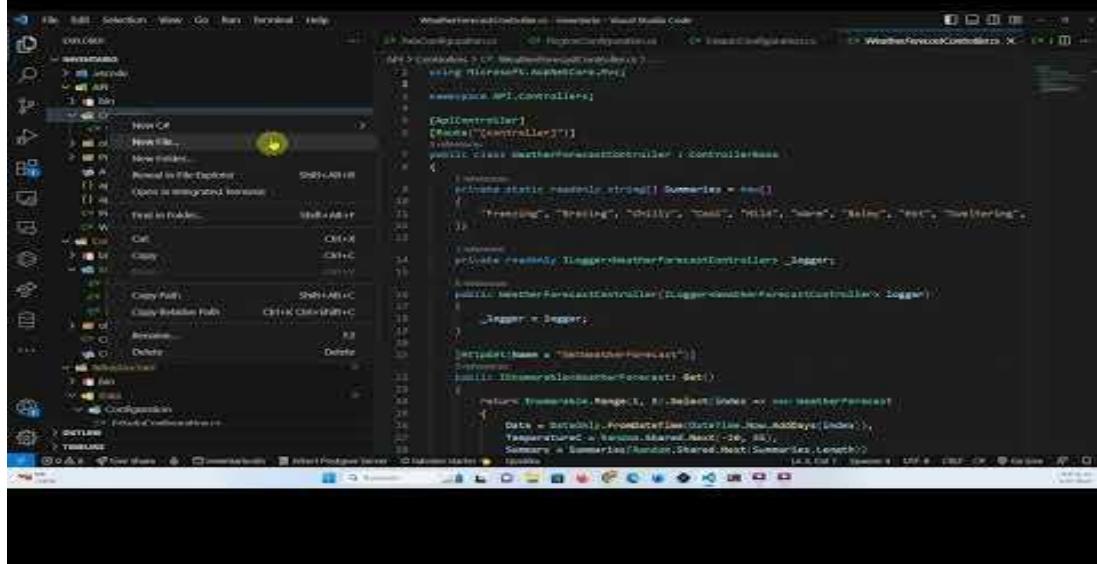
datos. La Unidad de Trabajo se encarga de orquestar las operaciones de los repositorios y asegura que se realicen en el contexto de una transacción única.

Las Unidades de Trabajo ofrecen las siguientes ventajas:

- Coherencia transaccional: Las operaciones dentro de una Unidad de Trabajo se realizan dentro de una única transacción. Esto significa que, si alguna operación falla, todas las operaciones realizadas hasta ese punto se pueden revertir, manteniendo la integridad de los datos.
- Simplificación del código: Las Unidades de Trabajo proporcionan una abstracción más alta que los repositorios individuales, lo que facilita la coordinación de operaciones y evita la duplicación de código para comenzar y finalizar transacciones.
- Control de contexto: La Unidad de Trabajo mantiene una única instancia del contexto de base de datos durante toda su duración. Esto ayuda a evitar problemas de concurrencia y asegura que todas las operaciones se realicen en el mismo contexto.

3.6.3 Reutilización de Código

Hasta el momento durante el desarrollo de este documento hemos trabajado una gran cantidad de elementos para el desarrollo de una web api, en este capítulo vamos a ver cómo podemos reutilizar el código creando clases base. Ver enlace [WebApi NetCore Part07](#)



3.6.4. Métodos de extensión

Los métodos de extensión en C# son una característica que permite agregar métodos a tipos existentes sin modificar directamente el código fuente de esos tipos. Los métodos de extensión se definen como métodos estáticos en una clase estática y se pueden invocar como si fueran métodos de instancia del tipo extendido. Esto proporciona una forma de agregar funcionalidad adicional a los tipos existentes sin heredar de ellos o modificar su implementación original.

Aquí hay un ejemplo de cómo se ve la sintaxis de un método de extensión:

```
public static class StringExtensions
{
    public static bool IsPalindrome(this string input)
    {
        // Implementación del método de extensión
    }
}
```

```
public static class StringExtensions
{
    public static bool IsPalindrome(this string input)
    {
        // Implementación del método de extensión
    }
}
```

En el ejemplo anterior, se define un método de extensión llamado **IsPalindrome** en la clase estática **StringExtensions**. Este método toma una cadena (**this string input**) como argumento y proporciona una implementación para verificar si la cadena es un palíndromo.

Para utilizar el método de extensión, simplemente se invoca como si fuera un método de instancia del tipo extendido:

```
string word = "radar";
bool isPalindrome = word.IsPalindrome();
```

```
string word = "radar";
bool isPalindrome = word.IsPalindrome();
```

En este caso, el método de extensión **IsPalindrome** se invoca en una instancia de tipo **string (word)** y devuelve el resultado de la verificación.

Es importante tener en cuenta algunas consideraciones al trabajar con métodos de extensión:

- Los métodos de extensión solo pueden agregar nuevos métodos, no nuevos miembros de datos.
- Los métodos de extensión deben estar definidos en una clase estática.
- Los métodos de extensión deben estar en el mismo espacio de nombres que el código que los utiliza o en un espacio de nombres importado con la instrucción **using**.
- Los métodos de extensión solo pueden acceder a los miembros públicos del tipo extendido.

Los métodos de extensión son una poderosa herramienta para extender la funcionalidad de los tipos existentes en C# de manera modular y sin modificar el código fuente original.

Para la implementación de los métodos de extensión lo realizaremos creando una clase que nos permita implementar el control de orígenes cruzados.

CORS (Cross-Origin Resource Sharing) es un mecanismo de seguridad implementado en los navegadores web que controla las solicitudes HTTP realizadas desde un origen (dominio, protocolo y puerto) a otro origen distinto. CORS impone restricciones sobre las solicitudes de recursos (como API) que se originan en un dominio y se dirigen a otro dominio diferente.

En Visual Studio Code y necesitas habilitar CORS para permitir solicitudes entre diferentes orígenes, puedes seguir estos pasos:

- Configuración del servidor:
- Identifica el servidor que utilizas en tu aplicación web (por ejemplo, Express.js o ASP.NET Core).
- Consulta la documentación del servidor para obtener información sobre cómo habilitar CORS. Los diferentes servidores tienen métodos específicos para habilitar CORS.
- Configura las opciones de CORS para permitir solicitudes desde los orígenes deseados. Esto puede implicar especificar dominios específicos o permitir cualquier origen mediante el uso del comodín '*'.

3.7 Mapeo de clases

El mapeo de clases en el contexto del desarrollo backend se refiere a la técnica de vincular o relacionar las clases de un lenguaje de programación con las tablas de una base de datos relacional. Esta técnica se utiliza para establecer una correspondencia entre las estructuras de datos en el código de la aplicación y las estructuras de datos almacenadas en la base de datos.

El mapeo de clases se utiliza para facilitar la interacción y la manipulación de los datos en la base de datos mediante el código de la aplicación. Proporciona una abstracción que permite a los desarrolladores trabajar con objetos en lugar de tener que lidiar directamente con las consultas y los comandos de la base de datos.

Existen diferentes enfoques y tecnologías para realizar el mapeo de clases en el backend. Algunos de los más comunes son:

- ORM (Mapeo Objeto-Relacional): Es una técnica que utiliza un framework o librería ORM para mapear las clases de la aplicación con las tablas de la base de datos de forma automática. Ejemplos populares son Entity Framework (para .NET), Hibernate (para Java) y SQLAlchemy (para Python).
- Consultas SQL personalizadas: En este enfoque, los desarrolladores escriben consultas SQL personalizadas para recuperar y manipular los datos de la base de datos. Utilizan herramientas o librerías que facilitan la ejecución y el mapeo de los resultados de estas consultas en objetos de la aplicación.

El mapeo de clases en el backend permite a los desarrolladores trabajar con un modelo de datos orientado a objetos en su código, lo que simplifica el desarrollo y el mantenimiento de las aplicaciones. Además, ayuda a mantener una separación entre la lógica de negocio y los detalles de la implementación de la base de datos.

3.7.1. DTO

DTO (Data Transfer Object) es un patrón de diseño utilizado en el desarrollo de software para transferir datos entre componentes de un sistema. Los DTOs son objetos simples que contienen datos, pero no tienen lógica ni comportamiento asociado.

La finalidad principal de los DTOs es proporcionar una estructura de datos optimizada para la transferencia de información entre diferentes capas o componentes de un sistema. Por ejemplo, pueden utilizarse para enviar datos desde el backend hacia el frontend, o para transmitir información entre diferentes servicios o microservicios en una arquitectura distribuida.

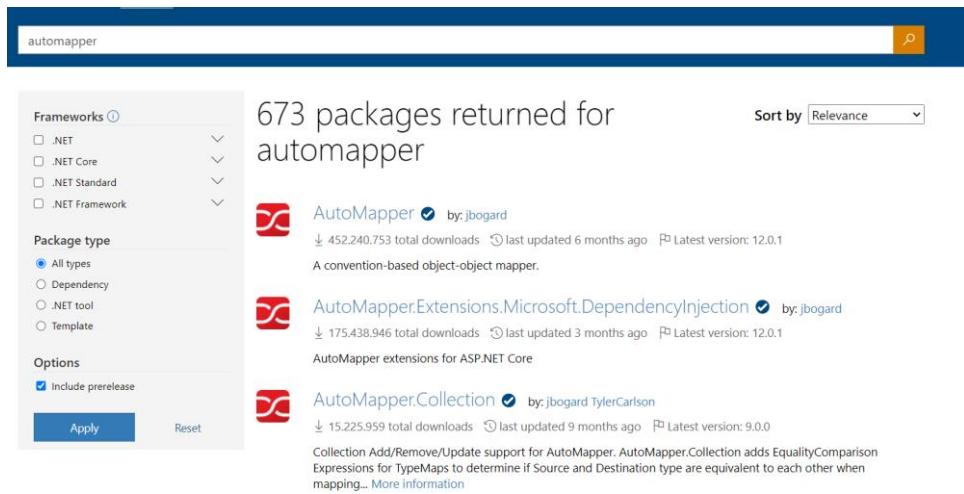
Algunas características de los DTOs son:

- Estructura simple: Los DTOs generalmente contienen solo propiedades o campos que representan los datos que se desean transferir. No deben contener lógica de negocio ni comportamiento adicional.
- Datos desacoplados: Los DTOs se utilizan para desacoplar las capas o componentes del sistema, ya que permiten transferir solo los datos necesarios, evitando el transporte de objetos complejos o dependencias innecesarias.
- Adaptación a la comunicación: Los DTOs se adaptan a los requerimientos de comunicación, como el formato de datos o la estructura necesaria para el intercambio de información entre componentes.
- Transferencia eficiente: Los DTOs están diseñados para ser eficientes en cuanto a la transferencia de datos, minimizando el tamaño y el costo de la comunicación entre componentes.

Los DTOs son objetos simples utilizados para transferir datos entre componentes de un sistema. Su objetivo es proporcionar una estructura de datos optimizada para la transferencia eficiente de información sin agregar lógica o comportamiento adicional.

3.7.1.1. Configuración del entorno y librerías

- Ir a la url <https://www.nuget.org/>
- En el buscador escribir AutoMapper



automapper

673 packages returned for automapper

Sort by Relevance

AutoMapper by jbogard
452,240,753 total downloads last updated 6 months ago Latest version: 12.0.1
A convention-based object-object mapper.

AutoMapper.Extensions.Microsoft.DependencyInjection by jbogard
175,438,946 total downloads last updated 3 months ago Latest version: 12.0.1
AutoMapper extensions for ASP.NET Core

AutoMapper.Collection by jbogard TylerCarlson
15,225,959 total downloads last updated 9 months ago Latest version: 9.0.0
Collection Add/Remove/Update support for AutoMapper. AutoMapper.Collection adds EqualityComparison Expressions for TypeMaps to determine if Source and Destination type are equivalent to each other when mapping... More information

Frameworks .NET .NET Core .NET Standard .NET Framework

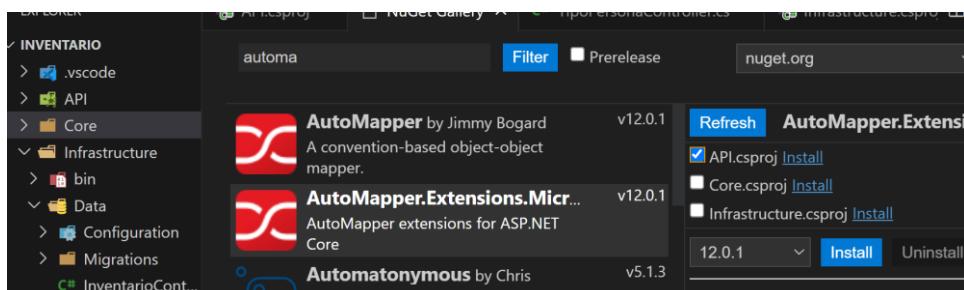
Package type All types Dependency .NET tool Template

Options Include prerelease

Apply Reset

Hacer clic en el enlace **AutoMapper.Extensions.Microsoft.DependencyInjection**

Copiar el comando `dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection --version 12.0.1` en la terminal del proyecto. Se debe instalar en el proyecto API



INVENTARIO

automa

AutoMapper by Jimmy Bogard v12.0.1
A convention-based object-object mapper.

AutoMapper.Extensions.Microsoft.DependencyInjection by Jimmy Bogard v12.0.1
AutoMapper extensions for ASP.NET Core

Automatonymous by Chris v5.1.3

Refresh Install

API.csproj Install

Core.csproj Install

Infrastructure.csproj Install

12.0.1 Install Uninstall

nuget.org

Se debe agregar el servicio en program.cs de API

```
using System.Reflection;
using API.Extensions;
using Infrastructure.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddAutoMapper(Assembly.GetEntryAssembly());
builder.Services.ConfigureCors();
builder.Services.AddControllers();
builder.Services.AddAplicacionServices();
```

using System.Reflection;

```

using API.Extensions;
using Infrastructure.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddAutoMapper(Assembly.GetEntryAssembly());
builder.Services.ConfigureCors();
builder.Services.AddControllers();
builder.Services.AddAplicacionServices();

```

Ajustar el repositorio de Países en el metodo GetByIdAsync.

```

2 references
public async Task<Pais> GetByIdAsync(string id){
    return await _context.Paises
        .Include(p => p.Estados)
        .FirstOrDefaultAsync(p => p.codPais == id);
}

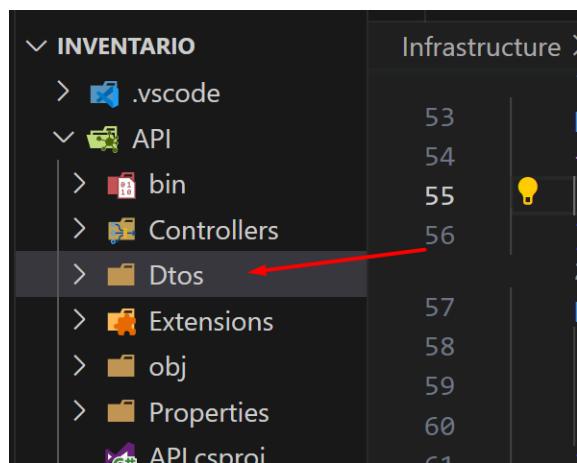
```

```

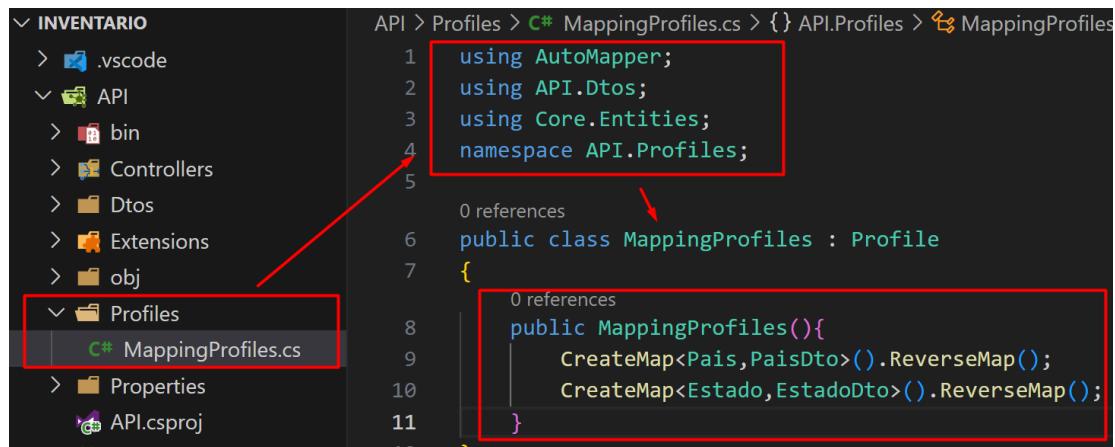
public async Task<Pais> GetByIdAsync(string id){
    return await _context.Paises
        .Include(p => p.Estados)
        .FirstOrDefaultAsync(p => p.codPais == id)
}

```

Agregar las clases dto en un folder Dtos en el proyecto API



Agregar la carpeta profiles, esta carpeta contendrá el mapeo de las clases. Se debe crear un archivo llamado MappingProfiles



```
1 using AutoMapper;
2 using API.Dtos;
3 using Core.Entities;
4 namespace API.Profiles;
5
6 public class MappingProfiles : Profile
7 {
8     public MappingProfiles()
9     {
10         CreateMap<Pais, PaisDto>().ReverseMap();
11         CreateMap<Estado, EstadoDto>().ReverseMap();
12     }
13 }
```

```
using AutoMapper;
using API.Dtos;
using Core.Entities;
namespace API.Profiles;
```

```
public class MappingProfiles : Profile
{
    public MappingProfiles()
    {
        CreateMap<Pais, PaisDto>().ReverseMap();
        CreateMap<Estado, EstadoDto>().ReverseMap();
    }
}
```

En el controlador de pais se debe agregar la dependencia de automapper.

3.7. Buenas prácticas Desarrollo Backend

3.7.1. Rate Limiting.

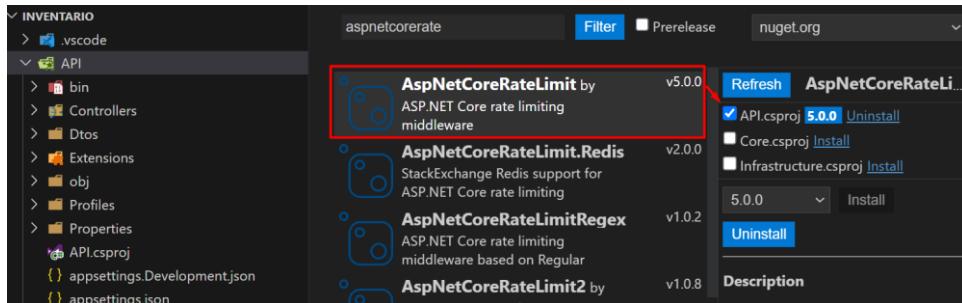
Rate limiting en .NET Core es una técnica utilizada para controlar la cantidad de solicitudes que una aplicación puede recibir de un cliente en un período de tiempo determinado. Esta funcionalidad es comúnmente empleada para prevenir abusos o ataques de denegación de servicio (DoS) y proteger los servidores de sobrecargas o comportamientos maliciosos.

3.7.1.1. Implementación Rate Limiting

Para la implementación de rate limiting siga el siguiente proceso:

1. Instalar AspNetCoreRateLimit :Git Autor :

<https://github.com/stefanprodan/AspNetCoreRateLimit>



En esta ocasión se realizará restricción de petición por IP. Agregue el siguiente código en el archivo de extensión que se encuentra en API > Extensions > ApplicationServiceExtension.cs

```
//services.AddScoped<IPersona, IPersonaRepository>(); //  
services.AddScoped<IUnitOfWork, UnitOfWork>();  
  
public static void ConfigureRateLimiting(this IServiceCollection services)  
{  
    services.AddMemoryCache();  
    services.AddSingleton<IRateLimitConfiguration, RateLimitConfiguration>();  
    services.AddInMemoryRateLimiting();  
    services.Configure<IpRateLimitOptions>(options =>  
    {  
        options.EnableEndpointRateLimiting = true;  
        options.StackBlockedRequests = false;  
        options.HttpStatusCode = 429;  
        options.RealIpHeader = "X-Real-IP";  
        options.GeneralRules = new List<RateLimitRule>()  
        {  
            new RateLimitRule  
            {  
                Endpoint = "*",  
                Period = "10s",  
                Limit = 2  
            };  
        };  
    });  
}
```

public static void ConfigureRateLimiting(this IServiceCollection services)

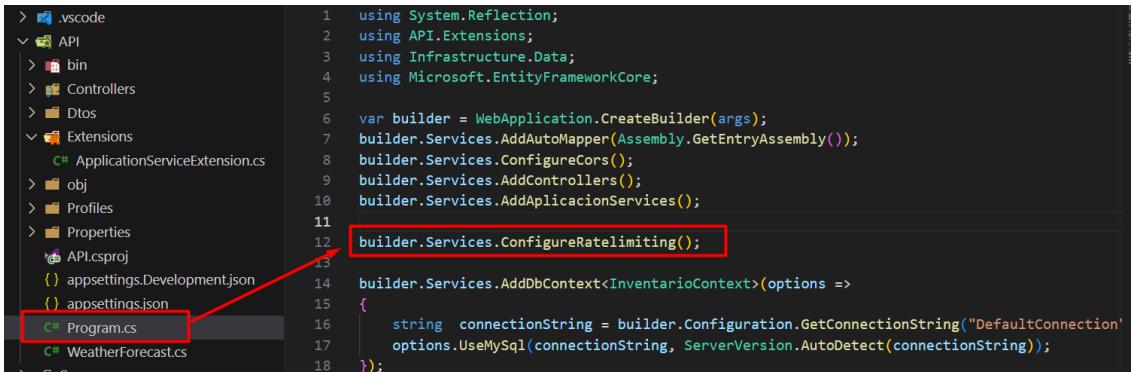
```
{  
  
    services.AddMemoryCache();  
  
    services.AddSingleton<IRateLimitConfiguration, RateLimitConfiguration>();  
    services.AddInMemoryRateLimiting();  
    services.Configure<IpRateLimitOptions>(options =>  
    {  
        options.EnableEndpointRateLimiting = true;  
        options.StackBlockedRequests = false;  
        options.HttpStatusCode = 429;  
    });  
}
```

```

        options.ReallpHeader = "X-Real-IP";
        options.GeneralRules = new List<RateLimitRule>
        {
            new RateLimitRule
            {
                Endpoint = "*",
                Period = "10s",
                Limit = 2
            }
        };
    });
}

```

Agregar el servicio al gestor de dependencias Program.cs



The screenshot shows a file explorer on the left and a code editor on the right. The file explorer displays the project structure:

- .vscode
- API
 - bin
 - Controllers
 - Dtos
 - Extensions
 - ApplicationServiceExtension.cs
 - obj
 - Profiles
 - Properties
 - API.csproj
 - appsettings.Development.json
 - appsettings.json
 - Program.cs
 - WeatherForecast.cs

The code editor shows the `Program.cs` file with the following content:

```

1  using System.Reflection;
2  using API.Extensions;
3  using Infrastructure.Data;
4  using Microsoft.EntityFrameworkCore;
5
6  var builder = WebApplication.CreateBuilder(args);
7  builder.Services.AddAutoMapper(Assembly.GetEntryAssembly());
8  builder.Services.ConfigureCors();
9  builder.Services.AddControllers();
10 builder.Services.AddAplicacionServices();
11
12 builder.Services.ConfigureRatelimiting();
13
14 builder.Services.AddDbContext<InventarioContext>(options =>
15 {
16     string connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
17     options.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString));
18 });

```

```

using System.Reflection;
using API.Extensions;
using Infrastructure.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddAutoMapper(Assembly.GetEntryAssembly());
builder.Services.ConfigureCors();
builder.Services.AddControllers();
builder.Services.AddAplicacionServices();

builder.Services.ConfigureRateLimiting();

builder.Services.AddDbContext<InventarioContext>(options =>
{
    string connectionString =
builder.Configuration.GetConnectionString("DefaultConnection");
    optionsBuilder.UseMySql(connectionString,
ServerVersion.AutoDetect(connectionString));
});

```

Realizar el uso del Middleware para poder realizar uso de la restricción.

```

1  using System.Reflection;
2  using API.Extensions;
3  using Infrastructure.Data;
4  using Microsoft.EntityFrameworkCore;
5  using AspNetCoreRateLimit;
6
7  var builder = WebApplication.CreateBuilder(args);
8  builder.Services.ConfigureRateLimiting();
9  builder.Services.AddAutoMapper(Assembly.GetEntryAs
10 builder.Services.ConfigureCors();
11 builder.Services.AddControllers();
12 builder.Services.AddAplicacionServices();
13

```

using System.Reflection;

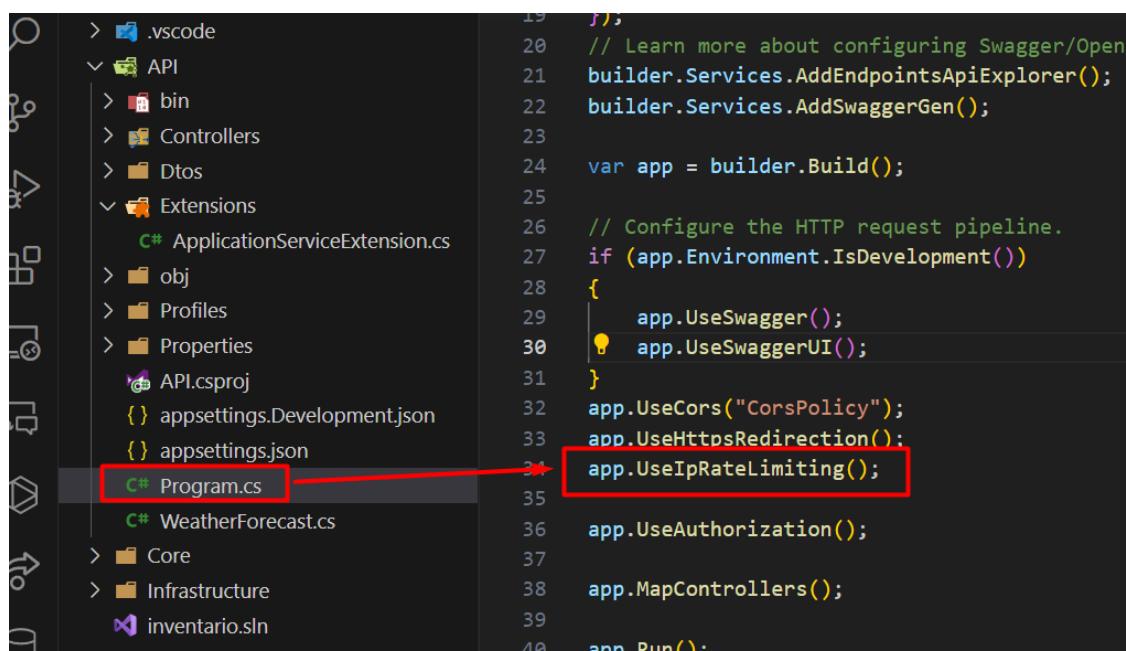
using API.Extensions;

```

using Infrastructure.Data;
using Microsoft.EntityFrameworkCore;
using AspNetCoreRateLimit;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddAutoMapper(Assembly.GetEntryAssembly());
builder.Services.ConfigureCors();
builder.Services.AddControllers();
builder.Services.AddAplicacionServices();

```



```

19    },
20    // Learn more about configuring Swagger/OpenAPI
21    builder.Services.AddEndpointsApiExplorer();
22    builder.Services.AddSwaggerGen();
23
24    var app = builder.Build();
25
26    // Configure the HTTP request pipeline.
27    if (app.Environment.IsDevelopment())
28    {
29        app.UseSwagger();
30        app.UseSwaggerUI();
31    }
32    app.UseCors("CorsPolicy");
33    app.UseHttpsRedirection();
34    app.UseIpRateLimiting();
35
36    app.UseAuthorization();
37
38    app.MapControllers();
39
40    app.Run();

```

```

// Learn more about configuring Swagger/OpenAPI
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

```

```

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();

```

```

        app.UseSwaggerUI();

    }

    app.UseCors("CorsPolicy");
    app.UseHttpsRedirection();
    app.UseIpRateLimiting();

    app.UseAuthorization();

    app.MapControllers();

    app.Run();

```

Cuando se realiza la petición al endpoint se retorna el siguiente encabezado:

NAME	VALUE
Content-Type	application/json; charset=utf-8
Date	Mon, 24 Jul 2023 23:51:10 GMT
Server	Kestrel
Transfer-Encoding	chunked
X-Rate-Limit-Limit	10s
X-Rate-Limit-Remaining	1
X-Rate-Limit-Reset	2023-07-24T23:51:20.8463848Z

3.7.2. Versionado Apis con Query String o Encabezado

El versionado de APIs es una práctica común en el desarrollo de aplicaciones web y servicios API (Application Programming Interface) que implica asignar un número de versión a cada versión importante de la API. Esto permite introducir cambios y mejoras en la API sin romper la compatibilidad con las versiones anteriores.

Cuando desarrollamos una API, los clientes que la consumen dependen de su estructura y comportamiento. Si realizamos cambios en la API sin un sistema de versionado adecuado, los clientes existentes podrían dejar de funcionar o experimentar errores.

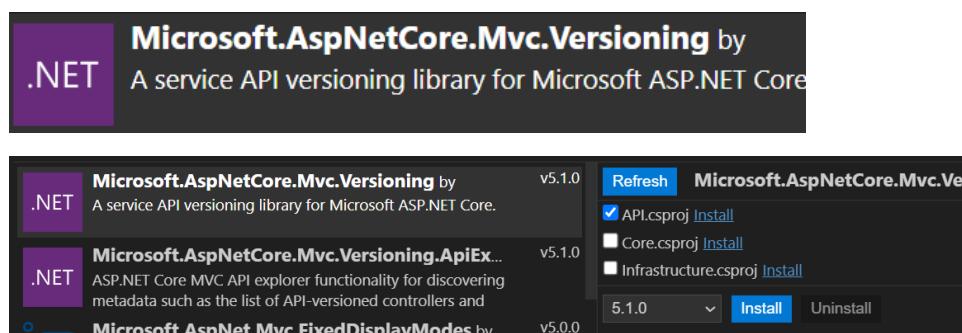
inesperados al intentar consumir la nueva versión de la API. Para evitar esto, se utiliza el versionado de APIs.

El versionado de APIs nos permite:

- **Compatibilidad:** Al asignar una versión a la API, aseguramos que los clientes existentes puedan seguir utilizando la versión anterior de la API sin problemas, incluso si se introduce una nueva versión con cambios.
- **Introducir mejoras y correcciones:** Podemos realizar mejoras, agregar nuevas funcionalidades o corregir errores en una nueva versión de la API sin afectar a los clientes que siguen utilizando versiones anteriores.
- **Control de cambios:** Permite un control más claro sobre qué cambios y características están disponibles en cada versión.
- **Fomentar la adopción de nuevas versiones:** Al tener un sistema de versionado, los clientes pueden optar por actualizar a una nueva versión de la API cuando lo deseen y cuando estén listos para manejar los cambios.

Para aplicar el versionado del Api en NetCore siga las siguientes instrucciones:

- Instale el siguiente paquete en el proyecto API.



- En el archivo `C# ApplicationServiceExtension.cs` agregue el siguiente código inicial.

```
public static void ConfigureApiVersioning(this IServiceCollection services){  
    services.AddApiVersioning(options => {  
  
    });  
}
```

```
public static void ConfigureApiVersioning(this IServiceCollection services){  
    services.AddApiVersioning(options => {  
  
    });  
}
```

Agregue el servicio a él inyector de dependencias. `C# Program.cs`

```

INVENTARIO
  > .vscode
  < API
    > bin
    > Controllers
    > Dtos
    < Extensions
      < ApplicationServiceExtension.cs
    > obj
    > Profiles
    > Properties
      < API.csproj
      { } appsettings.Development.json
      { } appsettings.json
    < Program.cs
    < WeatherForecast.cs
  > Core
  > Infrastructure
  < inventario.sln

```

API > C# Program.cs

```

1  using System.Reflection;
2  using API.Extensions;
3  using Infrastructure.Data;
4  using Microsoft.EntityFrameworkCore;
5  using AspNetCoreRateLimit;
6
7  var builder = WebApplication.CreateBuilder(args);
8  builder.Services.ConfigureRateLimiting();
9  builder.Services.ConfigureApiVersioning();
10 builder.Services.AddAutoMapper(Assembly.GetEntryAssembly());
11 builder.Services.ConfigureCors();
12 builder.Services.AddControllers();
13 builder.Services.AddAplicacionServices();
14
15 builder.Services.AddDbContext<InventarioContext>(options =>
16 {
17     string connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
18     options.UseSqlServer(connectionString, ServerVersion.AutoDetect(connectionString));
19 });
20 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore-swagger-ui
21

```

```

using System.Reflection;
using API.Extensions;
using Infrastructure.Data;
using Microsoft.EntityFrameworkCore;
using AspNetCoreRateLimit;

var builder = WebApplication.CreateBuilder(args);
builder.Services.ConfigureRateLimiting();
builder.Services.ConfigureApiVersioning();
builder.Services.AddAutoMapper(Assembly.GetEntryAssembly());
builder.Services.ConfigureCors();
builder.Services.AddControllers();
builder.Services.AddAplicacionServices();

builder.Services.ConfigureRateLimiting();

builder.Services.AddDbContext<InventarioContext>(options =>
{
    string connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    options.UseSqlServer(connectionString,
        ServerVersion.AutoDetect(connectionString));
});

```

Si se trata de consulta un endpoint del WebApi da el siguiente error:

```
1 {
2   "error": {
3     "code": "ApiVersionUnspecified",
4     "message": "An API version is required, but was not specified.",
5     "innerError": null
6   }
7 }
```

```
{
  "error": {
    "code": "ApiVersionUnspecified",
    "message": "An API version is required, but was not specified.",
    "innerError": null
  }
}
```

Este error ocurre porque no se ha establecido una versión inicial del WebApi. Para corregir el error se debe configura la versión en el método **ConfigureApiVersioning** que fue creado en el archivo de extensión **C# ApplicationServiceExtension.cs**.

```
public static void ConfigureApiVersioning(this IServiceCollection services){
  services.AddApiVersioning(options => {
    options.DefaultApiVersion = new ApiVersion(1, 0);
    options.AssumeDefaultVersionWhenUnspecified = true;
  });
}
```

```
public static void ConfigureApiVersioning(this IServiceCollection services){
  services.AddApiVersioning(options => {
    options.DefaultApiVersion = new ApiVersion(1, 0);
    options.AssumeDefaultVersionWhenUnspecified = true;
  });
}
```

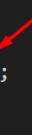
- **public static void ConfigureApiVersioning(this IServiceCollection services):** Esta es una extensión de método estático que extiende la clase **IServiceCollection**, lo que permite agregar la configuración de versionado de APIs como parte de la configuración de servicios en la clase **Startup**.

- **services.AddApiVersioning(options => { ... })**: Aquí, estamos agregando el middleware de versionado de APIs al contenedor de servicios de la aplicación. El método **AddApiVersioning** toma una función de configuración como parámetro, representada por **options => { ... }**.
- **options.DefaultApiVersion = new ApiVersion(1, 0);**: Esta línea establece la versión predeterminada de la API. Si el cliente no especifica una versión en la solicitud, se utilizará la versión 1.0 de forma predeterminada.
- **options.AssumeDefaultVersionWhenUnspecified = true;**: Esta línea indica que si el cliente no proporciona explícitamente un número de versión en la solicitud, se asumirá la versión predeterminada especificada en la línea anterior.

A continuación. Vamos a trabajar. El límite de peticiones haciendo uso. Del Query String. Para esto siga las siguientes instrucciones:

1. ingrese al archivo. **C# ApplicationServiceExtension.cs** y modifique el método **ConfigureApiVersioning**.

```
public static void ConfigureApiVersioning(this IServiceCollection services){
    services.AddApiVersioning(options => {
        options.DefaultApiVersion = new ApiVersion(1, 0);
        options.AssumeDefaultVersionWhenUnspecified = true;
        options.ApiVersionReader = new QueryStringApiVersionReader("v");
    });
}
```



```
public static void ConfigureApiVersioning(this IServiceCollection services){
    services.AddApiVersioning(options => {
        options.DefaultApiVersion = new ApiVersion(1, 0);
        options.AssumeDefaultVersionWhenUnspecified = true;
        options.ApiVersionReader = new QueryStringApiVersionReader("v");
    });
}
```

2. En los Controladores de la aplicación especifique cuál de los query string corresponden a la versión 1 o a una nueva versión.

```
[HttpGet]
[ApiVersion("1.0")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 references
public async Task<ActionResult<IEnumerable<PaisesDto>>> Get()
{
    var pais = await unitofwork.Paises.GetAllAsync();
    return this.mapper.Map<List<PaisesDto>>(pais);
}
```

```
[HttpGet]
[ApiVersion("1.0")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
```

```
public async Task<ActionResult<IEnumerable<PaisesDto>>> Get()
{
    var pais = await unitofwork.Paises.GetAllAsync();
    return this.mapper.Map<List<PaisesDto>>(pais);
}
```

En esta imagen se puede observar el decorador [ApiVersion("1.0")] que permite establecer una versión específica.

```
[HttpGet]
[ApiVersion("1.1")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
0 references
public async Task<ActionResult<IEnumerable<PaisesDto>>> Get11()
{
    var pais = await unitofwork.Paises.GetAllAsync();
    return this.mapper.Map<List<PaisesDto>>(pais);
}
```

```
[HttpGet]
[ApiVersion("1.1")]
[ProducesResponseType(StatusCodes.Status200OK)]
```

```
[ProducesResponseType(StatusCodes.Status400BadRequest)]
```

```
public async Task<ActionResult<IEnumerable<PaisesDto>>> Get11()
{
    var pais = await unitofwork.Paises.GetAllAsync();
    return this.mapper.Map<List<PaisesDto>>(pais);
}
```

Para llamar un api de versión diferente debe especificar el endPoint y la versión que desea acceder

`GET ▾ http://localhost:5000/api/Pais?v=1.1`

3.7.3. Versionado desde el Header

El versionado desde el encabezado (Header) en ASP.NET Core se refiere a una técnica para especificar la versión de una API directamente en la solicitud HTTP mediante el uso de encabezados personalizados. En lugar de incluir la versión en la URL o en el cuerpo de la solicitud, se agrega una cabecera (header) especial que indica la versión que se debe utilizar para procesar la solicitud.

Configuración:

a. Modifique el método de extensión `ConfigureApiVersioning()` que se encuentra en la clase

`C# ApplicationServiceExtension.cs`

```
public static void ConfigureApiVersioning(this IServiceCollection services){
    services.AddApiVersioning(options => {
        options.DefaultApiVersion = new ApiVersion(1, 0);
        options.AssumeDefaultVersionWhenUnspecified = true;
        //options.ApiVersionReader = new QueryStringApiVersionReader("v");
        options.ApiVersionReader = new HeaderApiVersionReader("X-Version"); //()
        options.ReportApiVersions = true;
    });
}
```

```
public static void ConfigureApiVersioning(this IServiceCollection services){
    services.AddApiVersioning(options => {
        options.DefaultApiVersion = new ApiVersion(1,0);
        options.AssumeDefaultVersionWhenUnspecified = true;
        //options.ApiVersionReader = new QueryStringApiVersionReader("v");
        options.ApiVersionReader = new HeaderApiVersionReader("X-Version"); //()
        options.ReportApiVersions = true;
    });
}
```

```
});  
}
```

- b. Ejecute el servidor y modifique el header en insomnia. Ver imagen siguiente.

The screenshot shows the Insomnia REST client interface. A red arrow points to the 'Add' button in the 'Headers' list. Another red box highlights the 'X-version' header entry, which is set to '1.1'. The response preview shows a JSON array with two items, each containing 'idCod' and 'nombrePais' fields.

```
1: [  
2:   {  
3:     "idCod": "001",  
4:     "nombrePais": "Colombia"  
5:   },  
6:   {  
7:     "idCod": "002",  
8:     "nombrePais": "Estados Unidos"  
9:   }  
10]
```

en Net CORE se puede combinar los diferentes métodos de La versión. A continuación, reino veremos cómo se aplica estos métodos.

```
public static void ConfigureApiVersioning(this IServiceCollection services){  
    services.AddApiVersioning(options => {  
        options.DefaultApiVersion = new ApiVersion(1, 0);  
        options.AssumeDefaultVersionWhenUnspecified = true;  
        //options.ApiVersionReader = new QueryStringApiVersionReader("v");  
        //options.ApiVersionReader = new HeaderApiVersionReader("X-Version"); //()  
        options.ApiVersionReader = ApiVersionReader.Combine(  
            new QueryStringApiVersionReader("v"),  
            new HeaderApiVersionReader("X-Version")  
        );  
        options.ReportApiVersions = true;  
    });  
}
```

```
public static void ConfigureApiVersioning(this IServiceCollection services){  
    services.AddApiVersioning(options => {  
        options.DefaultApiVersion = new ApiVersion(1,0);  
        options.AssumeDefaultVersionWhenUnspecified = true;  
        //options.ApiVersionReader = new QueryStringApiVersionReader("v");  
        //options.ApiVersionReader = new HeaderApiVersionReader("X-Version"); //()  
        options.ApiVersionReader = ApiVersionReader.Combine(  
            new QueryStringApiVersionReader("v"),  
            new HeaderApiVersionReader("X-Version")  
        )  
        options.ReportApiVersions = true;  
    });  
}
```

NAME	VALUE
Content-Type	application/json; charset=utf-8
Date	Tue, 25 Jul 2023 20:05:37 GMT
Server	Kestrel
Transfer-Encoding	chunked
api-supported-versions	1.0, 1.1
X-Rate-Limit-Limit	10s
X-Rate-Limit-Remaining	1
X-Rate-Limit-Reset	2023-07-25T20:05:47.6703356Z

3.7.4. Paginación en Net Core

- a. Cree una carpeta llamada Helper en el proyecto API
- b. Cree una clase llamada Pager.cs. Agregue el siguiente código.

```
namespace API.Helpers;
public class Pager<T> where T : class
{
    public string Search { get; private set; }
    public int PageIndex { get; private set; }
    public int PageSize { get; private set; }
    public int Total { get; private set; }
    public IEnumerable<T> Registers { get; private set; }

    public Pager(IEnumerable<T> registers, int total, int pageIndex,
        int pageSize, string search)
    {
        Registers = registers;
        Total = total;
       PageIndex = pageIndex;
        PageSize = pageSize;
        Search = search;
    }

    public int TotalPages
    {
        get
        {
            return (int)Math.Ceiling(Total / (double)PageSize);
        }
    }

    public bool HasPreviousPage
    {
        get
        {
            return (PageIndex > 1);
        }
    }

    public bool HasNextPage
    {
        get
        {
            return (PageIndex < TotalPages);
        }
    }
}
```

```
namespace API.Helpers;

public class Pager<T> where T : class
{
    public string Search { get; private set; }

    public int PageIndex { get; private set; }
```

```
public int PageSize { get; private set; }

public int Total { get; private set; }

public IEnumerable<T> Registers { get; private set; }

public Pager(IEnumerable<T> registers, int total, int pageIndex, int pageSize, string search)

{
    Registers = registers;
    Total = total;
   PageIndex = pageIndex;
    PageSize = pageSize;
    Search = search;
}

public int TotalPages

{
    get
    {
        return (int)Math.Ceiling(Total / (double)PageSize);
    }
}

public bool HasPreviousPage

{
    get
    {
        return (PageIndex > 1);
    }
}

public bool HasNextPage
```

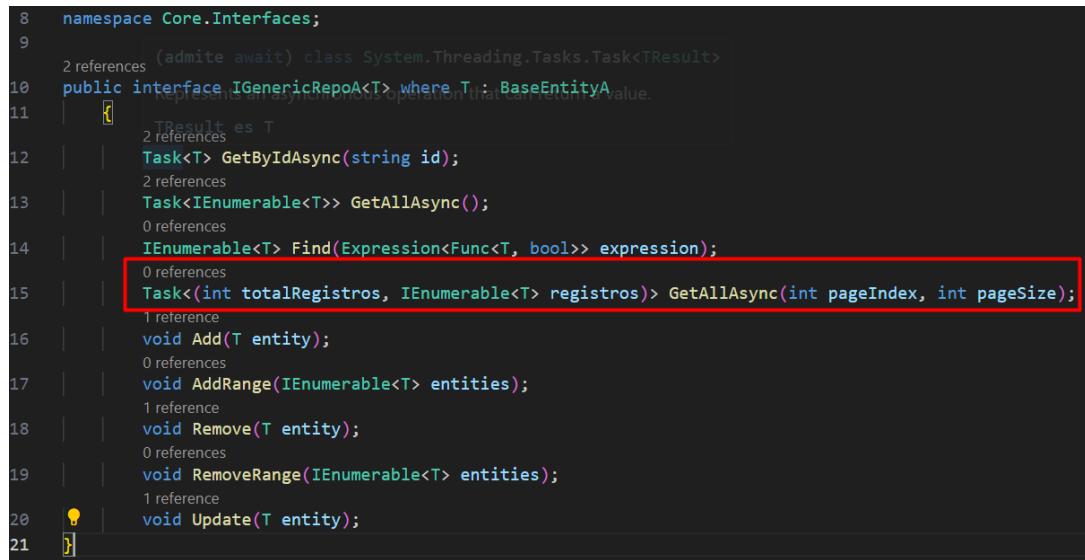
```

{
    get
    {
        return (PageIndex < TotalPages);
    }
}

```

- **namespace API.Helpers;**: Declaración del espacio de nombres (namespace) donde se encuentra la clase **Pager<T>**. Los espacios de nombres se utilizan para organizar y agrupar clases relacionadas dentro de un proyecto.
- **public class Pager<T> where T : class**: Declaración de la clase genérica **Pager<T>**, donde **T** es un parámetro de tipo que puede ser cualquier clase (**where T : class** indica que **T** debe ser una clase). Esto permite que la clase **Pager<T>** sea utilizada con diferentes tipos de datos.
- Propiedades de la clase:
- **Search**: Una cadena que representa el término de búsqueda utilizado para filtrar los datos.
- **PageIndex**: Un entero que indica el número de página actual.
- **PageSize**: Un entero que indica la cantidad de elementos que se muestran por página.
- **Total**: Un entero que representa el número total de elementos en la colección que se está paginando.
- **Registers**: Una colección genérica **IEnumerable<T>** que contiene los elementos que se muestran en la página actual.
- Constructor de la clase: La clase tiene un constructor que toma los siguientes parámetros:
 - **registers**: Una colección de elementos que se mostrarán en la página actual.
 - **total**: El número total de elementos en la colección sin paginar.
 - **pageIndex**: El número de página actual.
 - **pageSize**: La cantidad de elementos que se muestran por página.
 - **search**: El término de búsqueda utilizado para filtrar los datos.
- Propiedades adicionales:
- **TotalPages**: Una propiedad de solo lectura que calcula el número total de páginas necesarias para mostrar todos los elementos de la colección paginada.
- **HasPreviousPage**: Una propiedad de solo lectura que indica si hay una página anterior a la página actual.
- **HasNextPage**: Una propiedad de solo lectura que indica si hay una página siguiente a la página actual.

En el repositorio genérico se debe agregar un nuevo método que permita agregar funcionalidad de paginación.



```
8 namespace Core.Interfaces;
9 2 references (admite await) class System.Threading.Tasks.Task<TResult>
10 public interface IGenericRepoA<T> where T : BaseEntityA
11 {
12     TResult es T
13     Task<T> GetByIdAsync(string id);
14     Task<IEnumerable<T>> GetAllAsync();
15     IEnumerable<T> Find(Expression<Func<T, bool>> expression);
16     Task<(int totalRegistros, IEnumerable<T> registros)> GetAllAsync(int pageIndex, int pageSize);
17     void Add(T entity);
18     void AddRange(IEnumerable<T> entities);
19     void Remove(T entity);
20     void RemoveRange(IEnumerable<T> entities);
21     void Update(T entity);
}
```

namespace Core.Interfaces;

```
public interface IGenericRepoA<T> where T : BaseEntityA
{
    Task<T> GetByIdAsync(string id);
    Task<IEnumerable<T>> GetAllAsync();
    IEnumerable<T> Find(Expression<Func<T, bool>> expression);
    Task<(int totalRegistros, IEnumerable<T> registros)> GetAllAsync(int pageIndex, int pageSize)
        void Add(T entity);
        void AddRange(IEnumerable<T> entities);
        void Remove(T entity);
        void RemoveRange(IEnumerable<T> entities);
        void Update(T entity);
}
```

En la implementación del repositorio genérico agregar el siguiente método:

```

0 references
public virtual async Task<(int totalRegistros, IEnumerable<T> registros)> GetAllAsync(int pageIndex, int pageSize)
{
    var totalRegistros = await _context.Set<T>().CountAsync();
    var registros = await _context.Set<T>()
        .Skip((pageIndex - 1) * pageSize)
        .Take(pageSize)
        .ToListAsync();
    return (totalRegistros, registros);
}

```

```

public virtual async Task<(int totalRegistros, IEnumerable<T> registros)>
GetAllAsync(int pageIndex, int pageSize)

{
    var totalRegistros = await _context.Set<T>().CountAsync();

    var registros = await _context.Set<T>()
        .Skip((pageIndex - 1) * pageSize)
        .Take(pageSize)
        .ToListAsync();

    return (totalRegistros, registros);
}

```

En el repositorio de cada Entidad para el ejemplo práctico PaisRepository agregar el siguiente código:

```

public override async Task<(int totalRegistros, IEnumerable<Pais> registros)> GetAllAsync(int pageIndex,
int pageSize){
    var totalRegistros = await _context.Paises
        .CountAsync();
    var registros = await _context.Paises
        .Include(u => u.Estados)
        .Skip((pageIndex - 1) * pageSize)
        .Take(pageSize)
        .ToListAsync();
    return (totalRegistros, registros);
}

```

```

public override async Task<(int totalRegistros, IEnumerable<Pais> registros)>
GetAllAsync(int pageIndex, int pageSize)

{
    var totalRegistros = await _context.Paises.CountAsync();

    var registros = await _context.Paises
        .Include(u => u.Estados)
        .Skip((pageIndex - 1) * pageSize)
        .Take(pageSize)
        .ToListAsync();

    return (totalRegistros, registros);
}

```

}

En la carpeta helpers crear un nuevo archivo llamado params el permitirá definir la configuración de la paginación.

```
1  namespace API.Helpers;
2  public class Params
3  {
4      private int _pageSize = 5;
5      private const int MaxPageSize = 50;
6      private int _pageIndex = 1;
7      private string _search;
8      public int PageSize
9      {
10         get => _pageSize;
11         set => _pageSize = (value > MaxPageSize) ? MaxPageSize : value;
12     }
13     public intPageIndex
14     {
15         get => _pageIndex;
16         set => _pageIndex = (value <= 0) ? 1 : value;
17     }
18     public string Search
19     {
20         get => _search;
21         set => _search = (!String.IsNullOrEmpty(value)) ? value.ToLower() : "";
22     }
23 }
```

```
namespace API.Helpers;
public class Params
{
    private int _pageIndex = 5;
    private const int MaxPageSize = 50;
    private int _pageIndex = 1;
    private string _search;
    public int PageSize
    {
        get => _pageSize;
        set => _pageSize = (value > MaxPageSize) ? MaxPageSize : value;
    }
    public int PagelIndex
```

```

{
    get => _pageIndex;
    set => _pageIndex = (value <= 0) ? 1 : value;
}
public string Search
{
    get => _search;
    set => _search = (!String.IsNullOrEmpty(value)) ? value.ToLower() : "";
}
}

```

- Se define la clase **Params** con los siguientes campos privados:
- **_pageSize**: Representa el tamaño de página para la paginación y se inicializa con el valor 5.
- **MaxPageSize**: Es una constante que define el tamaño máximo de página permitido y se establece en 50.
- **_pageIndex**: Representa el índice de la página actual y se inicializa con el valor 1.
- **_search**: Representa el término de búsqueda y se inicializa con un valor nulo.
- Se definen propiedades públicas para acceder y controlar los campos privados:
- **PageSize**: Propiedad de lectura y escritura para el campo **_pageSize**. Al establecer este valor, se verifica si supera el **MaxPageSize** y, de ser así, se ajusta al máximo permitido.
- **PageIndex**: Propiedad de lectura y escritura para el campo **_pageIndex**. Si se establece un valor menor o igual a cero, se ajusta automáticamente a 1.
- **Search**: Propiedad de lectura y escritura para el campo **_search**. Al establecer este valor, se verifica si es nulo o vacío. Si no es nulo, se convierte a minúsculas; de lo contrario, se establece como una cadena vacía.

3.8 JWT

JSON web token es un estándar abierto que define una manera compacta para asegurar la transmisión de información entre diferentes partes. La información puede ser verificada y certificada porque está digitalmente firmada. JWT puede ser firmada usando una llave secreta con el algoritmo HMAC o un par de llaves usando RSA o ECDSA.

Aunque los JWT se pueden cifrar para proporcionar también el secreto entre las partes, nos centraremos en los tokens firmados. Los tokens firmados pueden verificar la integridad de las reclamaciones contenidas en ellos, mientras que los tokens cifrados ocultan esas reclamaciones de otras partes. Cuando los tokens se firman utilizando pares

de claves públicas/privadas, la firma también certifica que solo la parte que tiene la clave privada es la que la firmó.

Cuando se recomienda usar JWT

Autorización: Este es el escenario más común para usar JWT. Una vez que el usuario haya iniciado sesión, cada solicitud posterior incluirá el JWT, lo que permitirá al usuario acceder a las rutas, servicios y recursos que se permiten con ese token.

Intercambio de información: los tokens web JSON son una buena forma de transmitir información de forma segura entre las partes. Debido a que los JWT se pueden firmar, por ejemplo, usando pares de claves públicas/privadas, puede estar seguro de que los remitentes son quienes dicen ser. Además, como la firma se calcula utilizando el encabezado y la carga útil, también puede verificar que el contenido no ha sido manipulado.

3.8.1 Estructura JWT

En su forma más compacta los JWT están estructurados en tres grandes partes separadas por un punto(.). A continuación se explican cada una de las partes que componen un JWT.

Header

El encabezado normalmente consta de dos partes: el tipo de token, que es JWT, y el algoritmo de firma que se está utilizando, como HMAC SHA256 o RSA.

Payload

La segunda parte del token es la carga útil, que contiene las reclamaciones. Las reclamaciones son declaraciones sobre una entidad (normalmente, el usuario) y datos adicionales. Hay tres tipos de reclamaciones: registradas, públicas y privadas.

- Reclamaciones registradas: Estas son un conjunto de reclamaciones predefinidas que no son obligatorias, pero se recomiendan, para proporcionar un conjunto de reclamaciones útiles e interoperables.
- Reclamaciones públicas: Estas pueden ser definidas a voluntad por aquellos que usan JWT. Pero para evitar colisiones, deben definirse en el registro de tokens web IANA JSON o definirse como un URI que contenga un espacio de nombres resistente a las colisiones.
- Reclamaciones privadas: Estas son las reclamaciones personalizadas creadas para compartir información entre las partes que acuerdan usarlas y no son reclamaciones registradas ni públicas.

Signature

Para crear la parte de la firma tienes que tomar el encabezado codificado, el codificado del Payload, un secreto, el algoritmo especificado en el encabezado, y firmarlo.

La firma se utiliza para verificar que el mensaje no se cambió en el camino y, en el caso de los tokens firmados con una clave privada, también puede verificar que el remitente del JWT es quien dice ser.

3.8.1.1. Autenticación en NetCore

Para la implementación de JWT en Netcore siga los siguientes pasos teniendo en cuenta el nivel de 4 capas *Persistencia, Aplicación, Dominio, WebApi”.

1. Genere el proyecto Seguridad de tipo classlib.
2. Asocie el proyecto Seguridad a la solución.
3. Instale el paquete `"System.IdentityModel.Tokens.Jwt"` en el proyecto seguridad.
4. Establezca referencia entre el proyecto Seguridad y aplicación.
5. Establezca referencia entre el proyecto WebpApi y Seguridad.
6. En la carpeta Helpers que se encuentra en el proyecto WebApi cree las clases Autorizacion y JWT.

En la clase JWT aplique el siguiente código.

```
1  namespace ApiIncidencias.Helpers;
2
3      3 references
4  public class JWT
5  {
6      0 references
7      |  public string Key { get; set; }
8      |  0 references
9      |  public string Issuer { get; set; }
10     |  0 references
11     |  public string Audience { get; set; }
12     |  0 references
13     |  public double DurationInMinutes { get; set; }
14 }
```

```
namespace ApiIncidencias.Helpers;
public class JWT
{
    public string Key { get; set; }
    public string Issuer { get; set; }
    public string Audience { get; set; }
    public double DurationInMinutes { get; set; }
}
```

En la clase Autorizacion aplique el siguiente codigo

```
1  namespace ApiIncidencias.Helpers;
2  public class Autorizacion
3  {
4      2 references
5      public enum Roles
6      {
7          0 references
8          Administrador,
9          0 references
10         Gerente,
11         1 reference
12         Empleado
13     }
14     1 reference
15     public const Roles rol_predeterminado = Roles.Empleado;
16 }
```

```
namespace ApiIncidencias.Helpers;
public class Autorizacion
{
    public enum Roles
    {
        Administrador,
        Gerente,
        Empleado
    }
    public const Roles rol_predeterminado = Roles.Empleado;
}
```

En el método de extensión aplique el siguiente código.

```

public static void AddAplicacionServices(this IServiceCollection services)
{
    //services.AddScoped(typeof(IGenericRepository<>),typeof(GenericRepository<>));
    //services.AddScoped<IPaisInterface,PaisRepository>();
    //services.AddScoped<ITipoPersona,TipoPersonaRepository>(); //
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("using Microsoft.IdentityModel.Tokens.El tema
mas importante del bacnd no funcino.System.Demo"));
    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(opt =>
    {
        opt.TokenValidationParameters = new TokenValidationParameters
        [
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = key,
            ValidateAudience = false,
            ValidateIssuer = false
        ];
    });
    services.AddScoped<IJwtGenerador, JwtGenerador>();
    services.AddScoped<IPasswordHasher<Usuario>, PasswordHasher<Usuario>>();
    services.AddScoped<IUserService, UserService>();
    services.AddScoped<IUnitOfWork, UnitOfWork>();
}

```

```

public static void AddAplicacionServices(this IServiceCollection services)
{
    //services.AddScoped(typeof(IGenericRepository<>),typeof(GenericRepository<>));
    //services.AddScoped<IPaisInterface, PaisRepository>();
    //services.AddScoped<ITipoPersona,TipoPersonaRepository>(); //

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("using
Microsoft.IdentityModel.Tokens. El tema más importante del backend no
funciono.System.Demo"));

    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(opt =>
    {
        opt.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = key,
            ValidateAudience = false,
            ValidateIssuer = false
        };
    });
    services.AddScoped<IJwtGenerador, JwtGenerator>();
    services.AddScoped<IPasswordHasher<Usuario>, PasswordHasher<Usuario>>();
    services.AddScoped<IUserService, UserService>();
    services.AddScoped<IUnitOfWork, unitOfWork>();
}

```

```
}
```

Cree las clases Usuario, rol y UsuarioRol.

```
1  namespace Dominio;
2
3  public class Usuario : BaseEntityA
4  {
5      10 references
6      public string Username { get; set; }
7      3 references
8      public string Email { get; set; }
9      4 references
10     public string Password { get; set; }
11     5 references
12     public ICollection<Rol> Roles { get; set; } = new HashSet<Rol>();
13     1 reference
14     public ICollection<UsuariosRoles> UsuariosRoles { get; set; }
15 }
```

```
namespace Dominio;
```

```
public class Usuario : BaseEntityA
{
    public string Username { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public ICollection<Rol> Roles { get; set; } = new HashSet<Rol>();
    public ICollection<UsuariosRoles> UsuariosRoles { get; set; }
}
```

```
1  namespace Dominio;
2
3  public class Rol : BaseEntityA
4  {
5      3 references
6      public string Nombre { get; set; }
7      1 reference
8      public ICollection<Usuario> Usuarios { get; set; } = new HashSet<Usuario>();
9      1 reference
10     public ICollection<UsuariosRoles> UsuariosRoles { get; set; }
11 }
```

```
namespace Dominio;
```

```

public class Rol : BaseEntityA
{
    public string Nombre { get; set; }

    public ICollection<Usuario> Usuarios { get; set; } = new HashSet<Usuario>();

    public ICollection<UsuariosRoles> UsuariosRoles { get; set; }

}

```

```

1   namespace Dominio;
2
3   3 references
4   public class UsuariosRoles
5   {
6       2 references
7       |   public int UsuarioId { get; set; }
8       |   1 reference
9       |   public Usuario Usuario { get; set; }
10      |   2 references
11      |   public int RolId { get; set; }
12      |   1 reference
13      |   public Rol Rol { get; set; }
14   }

```

namespace Dominio;

```

public class UsuariosRoles
{
    public int Usuarioid { get; set; }

    public Usuario Usuario { get; set; }

    public int RolId { get; set; }

    public Rol Rol { get; set; }

}

```

Agregue los DbSet de las clases creadas al DbContext

```
Persistencia > C# ApilncidenciasContext.cs > ApilncidenciasContext
15     public DbSet<TipoPersona> TipoPersonas { get; set; }
16     0 references
17     public DbSet<TrainerSalon> TrainerSalones { get; set; }
18     0 references
19     public DbSet<Departamento> Departamentos { get; set; }
20     2 references
21     public DbSet<Pais> Paises { get; set; }
22     0 references
23     public DbSet<Genero> Generos { get; set; }
24     0 references
25     public DbSet<Rol> Roles { get; set; }
26     1 reference
27     public DbSet<Usuario> Usuarios { get; set; }
28
29 }
```

```
public DbSet<TipoPersona> TipoPersonas { get; set; }

public DbSet<TrainerSalon> TrainerSalones { get; set; }

public DbSet<Departamento> Departamentos { get; set; }

public DbSet<Pais> Paises { get; set; }

public DbSet<Genero> Generos { get; set; }

public DbSet<Rol> Roles { get; set; }

public DbSet<Usuario> Usuarios { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Pais>().HasIndex(idx => idx.NombrePais).IsUnique();
    modelBuilder.Entity<TrainerSalon>().HasKey(r => new { r.IdPerTrainerFk, r.});
    base.OnModelCreating(modelBuilder);
    modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
}
```

En el proyecto Aplicación cree una carpeta y nómbrala Contratos y dentro de dicha carpeta cree una interfaz y llámela **C# IJwtGenerador.cs**. Aplique el siguiente código a la Interfaz.

```

1  using Dominio;
2
3  namespace Aplicacion.Contratos;
4
5  4 references
6  public interface IJwtGenerador
7  {
8      2 references
9      string CrearToken(Usuario usuario);
10 }

```

using Dominio;

namespace Aplicacion.Contratos;

```

public interface IJwtGenerador
{
    string CrearToken(Usuario usuario);
}

```

Implemente la interfaz creando una nueva clase en el proyecto Seguridad. Para una mejor organización del proyecto cree una carpeta y llamela TokenSeguridad y cree una clase llamada

C# JwtGenerador.cs M

```

1  using System.IdentityModel.Tokens.Jwt;
2  using System.Security.Claims;
3  using System.Text;
4  using Aplicacion.Contratos;
5  using Dominio;
6  using Microsoft.IdentityModel.Tokens;
7  namespace Seguridad.TokenSeguridad;
8  public class JwtGenerador : IJwtGenerador
9  {
10     2 references
11     public string CrearToken(Usuario usuario)
12     {
13         var claims = new List<Claim>(
14             new Claim(JwtRegisteredClaimNames.NameId, usuario.Username)
15         );
16         var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("using Microsoft.IdentityModel.Tokens.El tema mas importante del backend no
17         funciona, System.Demo"));
18         var credenciales = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);
19         var tokenDescripcion = new SecurityTokenDescriptor
20         {
21             Subject = new ClaimsIdentity(claims),
22             Expires = DateTime.Now.AddDays(20),
23             SigningCredentials = credenciales
24         };
25         var tokenManejador = new JwtSecurityTokenHandler();
26         var token = tokenManejador.CreateToken(tokenDescripcion);
27         return tokenManejador.WriteToken(token);
28     }
}

```

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

```

using System.Text;
using Aplicacion.Contratos;
using Dominio;
using Microsoft.IdentityModel.Tokens;
namespace Seguridad.TokenSeguridad;

public class JwtGenerator : IJwtGenerador
{
    public string CrearToken(Usuario usuario)
    {
        var claims = new List<Claim>{
            new Claim(JwtRegisteredClaimNames.NameId, usuario.Username)
        };

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("using
Microsoft.IdentityModel.Tokens. El tema más importante del backend no
funciono.System.Demo"));

        var credenciales = new SigningCredentials(key,
SecurityAlgorithms.HmacSha512Signature);

        var tokenDescripcion = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(claims),
            Expires = DateTime.Now.AddDays(20),
            SigningCredentials = credenciales
        };

        var tokenManejador = new JwtSecurityTokenHandler();
        var token = new tokenManejador.CrearToken(tokenDescripcion);
        return tokenManejador.WriteToken(token);
    }
}

```

CrearToken(): Este método genera un token JWT para el usuario especificado. El usuario está representado por el objeto Usuario. El método primero crea una lista de afirmaciones, que incluye el nombre de usuario del usuario. El método luego crea una

clave de seguridad simétrica, que se utiliza para firmar el token. El método luego crea un descriptor de token de seguridad, que especifica el sujeto del token, la fecha de vencimiento del token y las credenciales de firma. El método luego crea un manejador de tokens de seguridad JWT, que se utiliza para crear el token. Finalmente, el método devuelve la cadena de tokens.

En el proyecto WebApi cree la carpeta Services y genere la siguiente Interfaz y su Implementación en la misma carpeta.

C# IUserService.cs Implemente el siguiente código en la clase

```
1  using ApilIncidentes.Dtos;
2
3  namespace ApilIncidentes.Services;
4
5  public interface IUserService
6  {
7      Task<string> RegisterAsync(RegisterDto model);
8      Task<DatosUsuarioDto> GetTokenAsync(LoginDto model);
9      Task<string> AddRoleAsync(AddRoleDto model);
10 }
```

```
using ApilIncidentes.Dtos;
namespace ApilIncidentes.Services;

public interface IUserService
{
    Task<string> RegisterAsync(RegisterDto model);
    Task<DatosUsuarioDto> GetTokenAsync(LoginDto model);
    Task<string> AddRoleAsync(AddRoleDto model);
}
```

- **RegisterAsync()**: Este método se utiliza para registrar un nuevo usuario. El método toma un objeto RegisterDto como parámetro. El objeto RegisterDto contiene el nombre, la dirección de correo electrónico y la contraseña del usuario.
- **GetTokenAsync()**: Este método se utiliza para obtener un token para un usuario conectado. El método toma un objeto LoginDto como parámetro. El objeto LoginDto contiene la dirección de correo electrónico y la contraseña del usuario.
- **AddRoleAsync()**: Este método se utiliza para agregar un rol a un usuario. El método toma un objeto AddRoleDto como parámetro. El objeto AddRoleDto contiene el ID del usuario y el nombre del rol.

- La interfaz `IUserService` se declara en el espacio de nombres `ApiIncidentes.Services`. La interfaz toma el espacio de nombres `ApiIncidentes.Dtos` como una declaración `using`. Este espacio de nombres contiene los objetos `RegisterDto`, `LoginDto` y `AddRoleDto`.
- El método `RegisterAsync()` tiene un tipo de retorno `Task<string>`. Esto significa que el método devuelve una tarea que eventualmente se completará con un valor de cadena. El método toma un objeto `RegisterDto` como parámetro. El objeto `RegisterDto` contiene el nombre, la dirección de correo electrónico y la contraseña del usuario.
- El método `GetTokenAsync()` tiene un tipo de retorno `Task<DatosUsuarioDto>`. Esto significa que el método devuelve una tarea que eventualmente se completará con un objeto `DatosUsuarioDto`. El objeto `DatosUsuarioDto` contiene los datos del usuario, como el ID del usuario, el nombre, la dirección de correo electrónico y los roles. El método toma un objeto `LoginDto` como parámetro. El objeto `LoginDto` contiene la dirección de correo electrónico y la contraseña del usuario.
- El método `AddRoleAsync()` tiene un tipo de retorno `Task<string>`. Esto significa que el método devuelve una tarea que eventualmente se completará con un valor de cadena. El método toma un objeto `AddRoleDto` como parámetro. El objeto `AddRoleDto` contiene el ID del usuario y el nombre del rol.

C# UserService.cs

Implemente el siguiente código en la clase.

```
1  using System.IdentityModel.Tokens.Jwt;
2  using System.Security.Claims;
3  using System.Text;
4  using ApiIncidencias.Dtos;
5  using ApiIncidencias.Helpers;
6  using Aplicacion.Contratos;
7  using Dominio;
8  using Dominio.Interfaces;
9  using Microsoft.AspNetCore.Identity;
10 using Microsoft.Extensions.Options;
11 using Microsoft.IdentityModel.Tokens;
12
13 namespace ApiIncidencias.Services;
14
15 public class UserService : IUserService
16 {
17     private readonly JWT _jwt;
18     private readonly IUnitOfWork _unitOfWork;
19     private readonly IPasswordHasher<Usuario> _passwordHasher;
20     private readonly IJwtGenerador _jwtGenerador;
21
22     public UserService(IUnitOfWork unitOfWork, IOptions<JWT> jwt,
23         IPasswordHasher<Usuario> passwordHasher, IJwtGenerador jwtGenerador)
24     {
25         _jwt = jwt.Value;
26         _unitOfWork = unitOfWork;
27         _passwordHasher = passwordHasher;
28         _jwtGenerador = jwtGenerador;
29     }
30
31     public async Task<string> RegisterAsync(RegisterDto registerDto)
32     {
33         var usuario = new Usuario
34         {
35
36             Email = registerDto.Email,
37             Username = registerDto.Username,
38
39         };
40
41         usuario.Password = _passwordHasher.HashPassword(usuario, registerDto.Password);
42
43         var usuarioExiste = _unitOfWork.Usuarios
44             .Find(u => u.Username.ToLower() == registerDto.Username.ToLower())
45             .FirstOrDefault();
46
47         if (usuarioExiste == null)
48         {
49             var rolPredeterminado = _unitOfWork.Roles
50                 .Find(u => u.Nombre == Autorizacion.rol_predeterminado.ToString())
51                 .First();
```

```

52     try
53     {
54         usuario.Roles.Add(rolPredeterminado);
55         _unitOfWork.Usuarios.Add(usuario);
56         await _unitOfWork.SaveAsync();
57
58         return $"El usuario {registerDto.Username} ha sido registrado exitosamente";
59     }
60     catch (Exception ex)
61     {
62         var message = ex.Message;
63         return $"Error: {message}";
64     }
65 }
66 else
67 {
68     return $"El usuario con {registerDto.Username} ya se encuentra registrado.";
69 }
70 }

71 public async Task<string> AddRoleAsync(AddRoleDto model)
72 {
73     var usuario = await _unitOfWork.Usuarios
74         .GetByUsernameAsync(model.Username);
75     if (usuario == null)
76     {
77         return $"No existe algún usuario registrado con la cuenta {model.Username}.";
78     }
79     var resultado = _passwordHasher.VerifyHashedPassword(usuario, usuario.Password, model.Password);
80     if (resultado == PasswordVerificationResult.Success)
81     {
82         var rolExiste = _unitOfWork.Roles
83             .Find(u => u.Nombre.ToLower() == model.Role.ToLower())
84             .FirstOrDefault();
85         if (rolExiste != null)
86         {
87             var usuarioTieneRol = usuario.Roles
88                 .Any(u => u.Id == rolExiste.Id);
89
90             if (usuarioTieneRol == false)
91             {
92                 usuario.Roles.Add(rolExiste);
93                 _unitOfWork.Usuarios.Update(usuario);
94                 await _unitOfWork.SaveAsync();
95             }
96             return $"Rol {model.Role} agregado a la cuenta {model.Username} de forma exitosa.";
97         }
98         return $"Rol {model.Role} no encontrado.";
99     }
100    return $"Credenciales incorrectas para el usuario {usuario.Username}.";
101 }

```

```

using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using ApilIncidentes.Dtos;
using ApilIncidentes.Helpers;
using Aplicacion.Contratos;
using Dominio;
using Dominio.Interfaces;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;

namespace ApilIncidentes.Services;

public class UserService : IUserService

```

```

{
    private readonly JWT _jwt;
    private readonly IUnitOfWork _unitOfWork;
    private readonly IPasswordHasher<Usuario> _passwordHasher;
    private readonly IJwtGenerador _jwtGenerador;
    public UserService (IUnitOfWork unitOfWork, IOptions<JWT> jwt,
IPasswordHasher<Usuario> passwordHasher, IJwtGenerador jwtGenerador);
    {
        _jwt = jwt.Value;
        _unitOfWork = _unitOfWork;
        _passwordHasher = passwordHasher;
        _jwtGenerador = jwtGenerador;
    }
}

public async Task<string> RegisterAsync(RegisterDto registerDto)
{
    var usuario = new Usuario
    {

        Email = registerDto.Email,
        Username = registerDto.Username

    };
    usuario.Password = _passwordHasher.HashPassword(usuario, registerDto.Password);

    var usuarioExiste = _unitOfWork.Usuarios
        .Find(u => u.Username.ToLower() == registerDto.Username.ToLower())
        .FirstOrDefault();

    if (usuarioExiste == null)
    {
        var rolPredeterminado = _unitOfWork.Roles
            .Find(u => u.Nombre == Autorizacion.rol_predeterminado.ToString())
            .First();

        try
        {
            usuario.Roles.Add(rolPredeterminado);
            _unitOfWork.Usuarios.Add(usuario);
            await _unitOfWork.SaveAsync();

            return $"El usuario {registerDto.Username} ha sido registrado exitosamente"
        }
        catch (Exception ex)
        {
            var message = ex.Message
            return $"Error: {message}";
        }
    }
}

```

```

    }
    else
    {
        return $"El usuario con {registerDto.Username} ya se encuentra registrado.";
    }
}

public async Task<string> AddRoleAsync(AddRoleDto model)
{
    var usuario = await _unitOfWork.Usuarios
        .GetByUsernameAsync(model.Username);

    if (usuario == null)
    {
        return $"No existe algún usuario registrado con la cuenta {model.Username}.";
    }

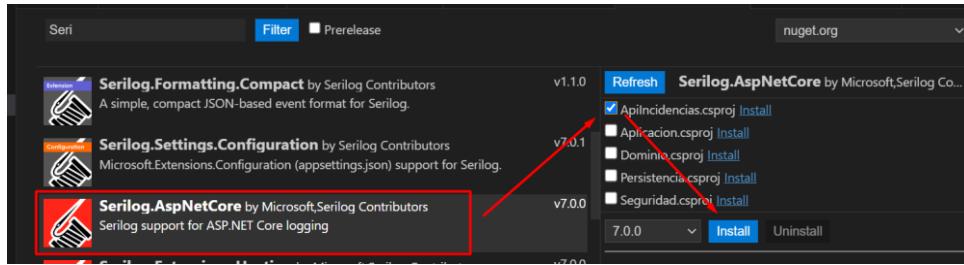
    var resultado = _passwordHasher.VerifyHashedPassword(usuario, usuario.Password,
model.Password);
    if (resultado == PasswordVerificationResult.Success)
    {
        var rolExiste = _unitOfWork.Roles
            .Find(u => u.Nombre.ToLower() == model.Role.ToLower())
            .FirstDefault();
        if(rolExiste != null)
        {
            var usuarioTieneRol = usuario.Roles
                .Any(u => u.Id == rolExiste.Id);

            if (usuarioTieneRol == false)
            {
                usuario.Roles.Add(rolExiste);
                _unitOfWork.Usuarios.Update(usuario);
                await _unitOfWork.SaveAsync();
            }
            return $"Rol {model.Role} agregado a la cuenta {model.Username} de forma exitosa";
        }
        return $"Rol {model.Role} no encontrado.";
    }
    return $"Credenciales incorrectas para el usuario {usuario.Username}";
}

```

3.9. Bitácoras o Logs

Netcore en su estructura interna posee un manejador de logs pero no permite el almacenamiento de los logs en archivos y bases de datos, para este tipo de requerimiento se usará un paquete de terceros llamado Serilog; esta librería se instala en el proyecto WebApi.



En el contenedor de dependencias Agregar el siguiente código.

```

7  using Microsoft.AspNetCore.Mvc.Authorization;
8  using Microsoft.EntityFrameworkCore;
9  using Microsoft.IdentityModel.Tokens;
10 using Persistencia;
11 using Serilog;
12
13 var builder = WebApplication.CreateBuilder(args);
14 var logger = new LoggerConfiguration()
15     .ReadFrom.Configuration(builder.Configuration)
16     .Enrich.FromLogContext()
17     .CreateLogger();
18
19 //builder.Logging.ClearProviders();
20 builder.Logging.AddSerilog(logger);
21 // Add services to the container.
22 //builder.Services.AddJwt(builder.Configuration);
23 builder.Services.AddControllers(options =>
24 {

```

using Microsoft.AspNetCore.Mvc.UseAuthorization;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using Persistencia;

using Serilog;

var builder = WebApplication.CreateBuilder(args);

var logger = new LoggerConfiguration()

.ReadFrom.Configuration(builder.Configuration)

.Enrich.FromLogContext()

.CreateLogger();

//builder.Logging.ClearProviders();

builder.Logging.AddSerilog(logger);

```

// Add services to the container.

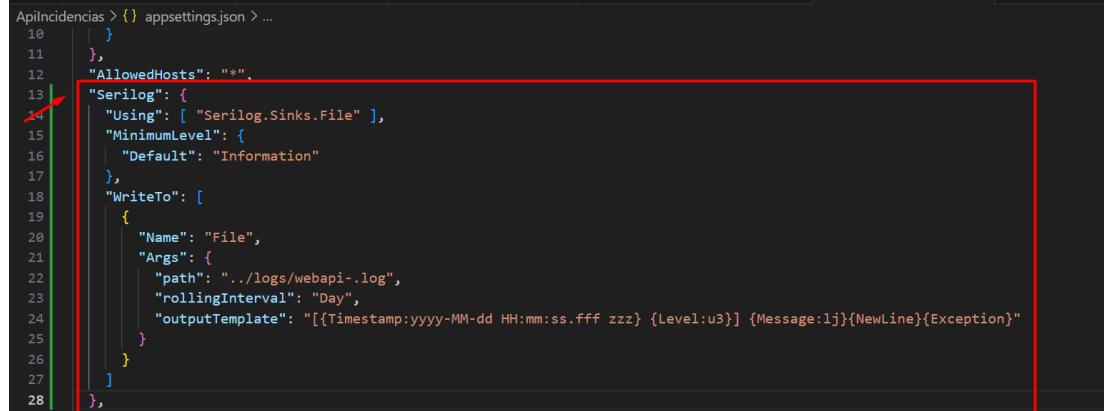
// builder.Services.AddJwt(builder.Configuration);

builder.Services.AddControllers(options =>

{

```

En el archivo de configuración global appsetting agregar el siguiente código:



```

Aplicaciones > {} appsettings.json > ...
10
11
12
13 "AllowedHosts": "*",
14
15 "Serilog": {
16     "Using": [ "Serilog.Sinks.File" ],
17     "MinimumLevel": {
18         "Default": "Information"
19     },
20     "WriteTo": [
21         {
22             "Name": "File",
23             "Args": {
24                 "path": ".../logs/webapi-.log",
25                 "rollingInterval": "Day",
26                 "outputTemplate": "[{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} {Level:u3}] {Message:lj}{NewLine}{Exception}"
27             }
28         }
29     ]
30 }

```

```

"AllowedHosts" : "*",

"Serilog" :{

    "Using" : ["Serilog.Sinks.File"],

    "MinimumLevel" : {

        Default : "Information"

    }

    "WriteTo" : [

        {

            "Name" : "File",

            "Args" : {

                "path": ".../logs/webapi-.log",

                "rollingInterval": "Day",

                "outputTemplate" : "[{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz}{Level:u3}]{Message:lj}{NewLine}{Exception}"

            }

        }

    ]

}

```

Agregar validación de excepciones

```
59  using (var scope = app.Services.CreateScope())
60  {
61      var services = scope.ServiceProvider;
62      var loggerFactory = services.GetRequiredService<ILoggerFactory>();
63      try
64      {
65          var context = services.GetRequiredService<ApiIncidenciasContext>();
66          await context.Database.MigrateAsync();
67      }
68      catch (Exception ex)
69      {
70          var _logger = loggerFactory.CreateLogger<Program>();
71          _logger.LogError(ex, "Ocurrio un error durante la migracion");
72      }
73 }
```

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServicesProvider;
    var loggerFactory = services.GetRequiredService<ILoggerFactory>();
    try
    {
        var context = services.GetRequiredService<ApiIncidenciasContext>();
        await context.Database.MigrateAsync();
    }
    catch (Exception ex)
    {
        var _logger = loggerFactory.CreateLogger<Program>();
        _logger.LogError(ex, "Ocurrio un error durante la migración.");
    }
}
```

3.9.1. Personalización de Mensajes de Excepciones

Para personalizar los mensajes que se pueden producir en tiempo de ejecución y se guardados de forma adecuada siga los siguientes pasos:

- Cree una nueva carpeta en Helpers y llamela ApiResponse e implemente el siguiente código:

```
Apilncidencias > Helpers > Errors > C# ApiResponse.cs > ...
1  namespace Apilncidencias.Helpers.Errors;
2
3      1 reference
4  public class ApiResponse
5  {
6      1 reference
7      public int StatusCode { get; set; }
8      1 reference
9      public string Message { get; set; }
10
11      0 references
12  public ApiResponse(int statusCode, string message = null)
13  {
14      1 reference
15      StatusCode = statusCode;
16      Message = message ?? GetDefaultMessage(statusCode);
17
18      1 reference
19  private string ...GetDefaultMessage(int statusCode)
20  {
21      1 reference
22      return statusCode switch
23      {
24          400 => "Has realizado una petición incorrecta.",
25          401 => "Usuario no autorizado.",
26          404 => "El recurso que has intentado solicitar no existe.",
27          405 => "Este método HTTP no está permitido en el servidor.",
28          500 => "Error en el servidor. No eres tú, soy yo. Comunícate con el administrador XD.",
29          _ => throw new NotImplementedException()
30      };
31
32  }
33
34  }
```

```
namespace Apilncidencias.Helpers.Errors;

public class ApiResponse
{
    public int StatusCode { get; set; }

    public string Message { get; set; }

    public ApiResponse(int statusCode, string message = null)
    {
        StatusCode = statusCode;

        Message = message ?? GetDefaultMessage(statusCode);
    }

    private string GetDefaultMessage(int statusCode)
    {
        return statusCode switch
        {
            400 => "Has realizado una petición incorrecta",

```

```

        401 => "Usuario no autorizado",
        404 => "El recurso que has intentado solicitar no existe.",
        405 => "Este método HTTP no está permitido en el servidor.",
        500 => "Error en el servidor. No eres tú, soy yo. Comunicaque con el administrador XD",
        _ => throw new NotImplementedException()

    }

}

}

```

B. En el método Get del controlador agregue una instancia a la clase ApiResponse como se observa en la siguiente imagen.

```

54 [HttpGet("{id}")]
55 [ProducesResponseType(StatusCodes.Status200OK)]
56 [ProducesResponseType(StatusCodes.Status400BadRequest)]
57 [ProducesResponseType(StatusCodes.Status404NotFound)]
0 references
58 public async Task<ActionResult<PaisDto>> Get(string id)
59 {
60     var pais = await _unitOfWork.Paises.GetByIdAsync(id);
61     if (pais == null){
62         return NotFound(new ApiResponse(404,"El país solicitado no existe."));
63     }
64     return _mapper.Map<PaisDto>(pais);
65 }

```

```

[HttpGet("{id}")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]

```

```

public async Task<ActionResult<PaisDto>> Get(string id)
{
    var pais = await _unitOfWork.Paises.GetByIdAsync(id);
    if (pais == null)
    {
        return NotFound(new ApiResponse(404, "El país solicitado no existe."));
    }
    return _mapper.Map<PaisDto>(pais);
}

```

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
1 reference
public async Task<ActionResult<Pais>> Post(PaisDto paisDto){
    var pais = _mapper.Map<Pais>(paisDto);
    this._unitOfWork.Paises.Add(pais);
    await _unitOfWork.SaveChangesAsync();
    if (pais == null)
    {
        return BadRequest(new ApiResponse(400));
    }
    paisDto.Id = pais.Id;
    return CreatedAtAction(nameof(Post), new { id = paisDto.Id }, paisDto);
}
```

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

```
public async Task<ActionResult<Pais>> Post(PaisDto paisDto){
```

```
    var pais = _mapper.Map<Pais>(paisDto);
```

```
    this._unitOfWork.Paises.Add(pais);
```

```
    await _unitOfWork.SaveChangesAsync();
```

```
    if(pais == null)
```

```
{
```

```
    return BadRequest(new ApiResponse(400));
```

```
}
```

```
    paisDto.Id = pais.Id;
```

```
    return CreatedAtAction(nameof(Post), new{id= paisDto.Id}, paisDto);
```

```
}
```