

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. Codigos de status.....	11
2. Programación Intermedia.....	12
2.1. Usando Docker Con Mysql.....	12
2.1.1. Instalación .....	13
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. Proyecto Base (Habilitando Migraciones).....	26
3.4. Proyecto Base (Fluent Api).....	30
3.5. Repositorios, Unidad de trabajo y métodos de extensión .....	34
3.5.1. Repositorios .....	34
3.5.2. Unidades de trabajo .....	34
3.5.3 Reutilización de Código.....	35
3.5.4. Métodos de extensión .....	37
3.2. Conexión a bases de datos.....	39
3.2.1. Arquitectura Básica.....	39

## 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.

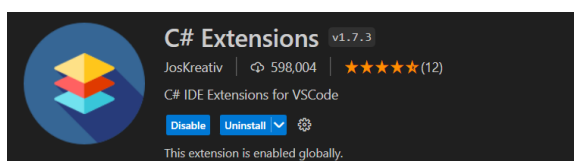
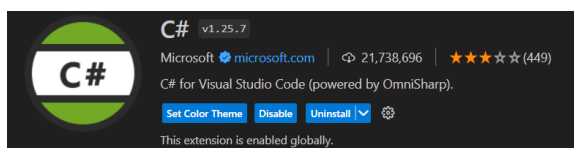
**Conocimientos de programación:** Para desarrollar en .NET Core, es necesario tener conocimientos de programación en C# u otros lenguajes compatibles con .NET Core, como F# o Visual Basic. Familiarizarte con los conceptos de programación orientada a objetos y los principios básicos de .NET Framework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

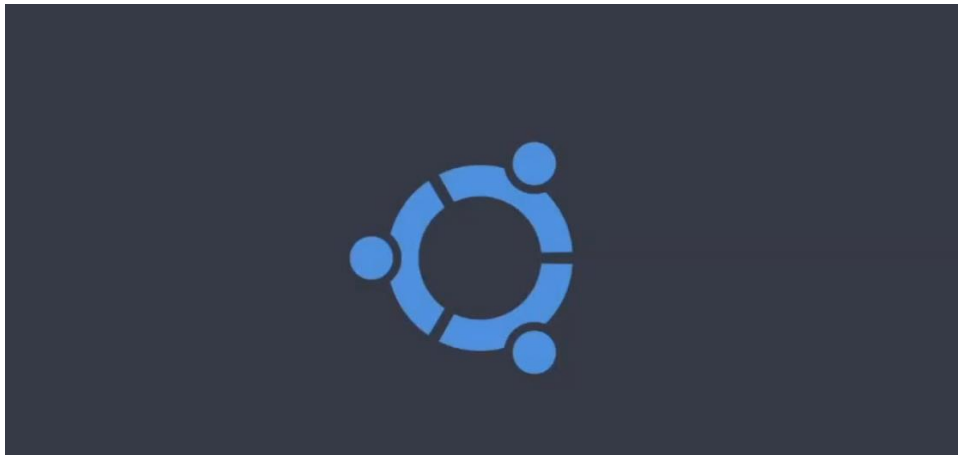
### Software necesario

- 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.

```
sputnik-monitor@linux: ~  
sputnik-monitor@linux:~$ sudo apt remove 'dotnet*'  
[sudo] password for sputnik-monitor:   
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 'dotnet6' 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:~$
```

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 'dotnet6' 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 **sudo 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 liblttng-ust-common1
  liblttng-ust-ctl5 liblttng-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 liblttng-ust-common1
  liblttng-ust-ctl5 liblttng-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 liblttng-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:

```
sputnik-monitor@linux: ~
.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.App]
  Microsoft.NETCore.App 7.0.5 [/usr/lib/dotnet/shared/Microsoft.NETCore.App]

Other architectures found:
  None

Environment variables:
  Not set

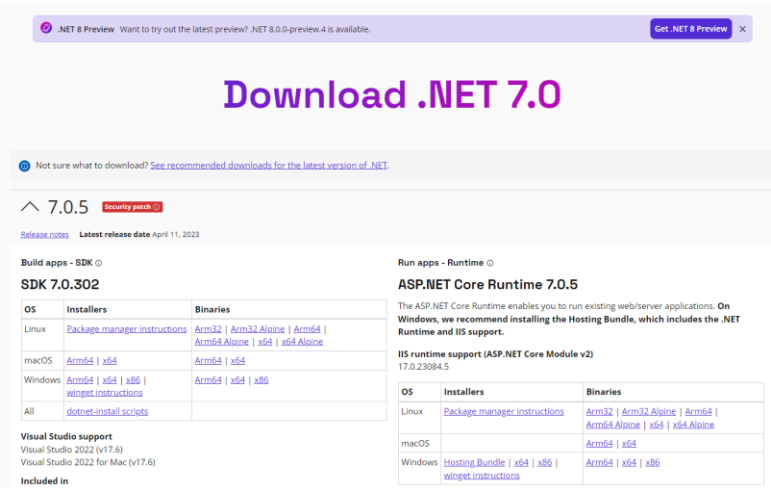
global.json file:
  Not found

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>

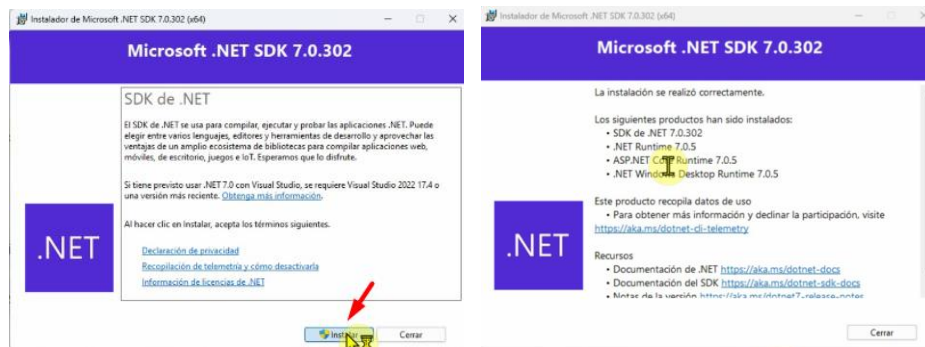


- Descargar el SDK 7.0 teniendo en cuenta la arquitectura del sistema operativo ya se 32 (x86) o 64(x64) bits.

## SDK 7.0.302

OS	Installers	Binaries
Linux	<a href="#">Package manager instructions</a>	<a href="#">Arm32</a>   <a href="#">Arm32 Alpine</a>   <a href="#">Arm64</a>   <a href="#">Arm64 Alpine</a>   <a href="#">x64</a>   <a href="#">x64 Alpine</a>
macOS	<a href="#">Arm64</a>   <a href="#">x64</a>	<a href="#">Arm64</a>   <a href="#">x64</a>
Windows	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>   <a href="#">winget instructions</a>	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>
All	<a href="#">dotnet-install scripts</a>	

- 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:** Solicita 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](http://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. Codigos 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

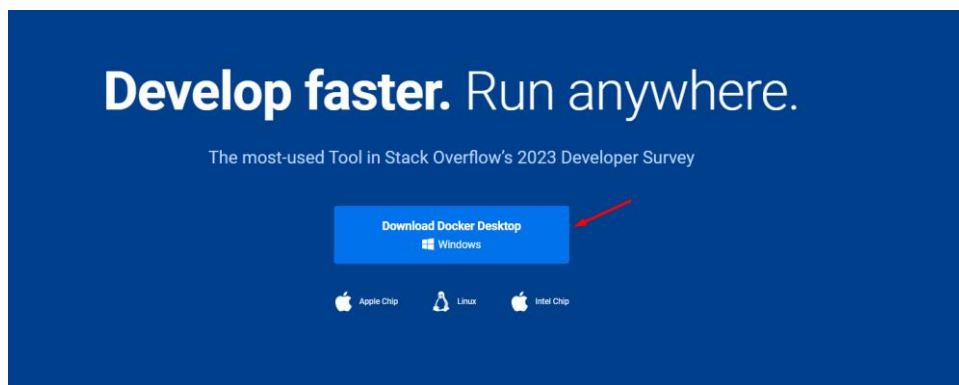
desarrolladores crear entornos de desarrollo consistentes y reproducibles, facilita la colaboración entre equipos y simplifica el proceso de implementación y escalabilidad de aplicaciones.

### 2.1.1. Instalación

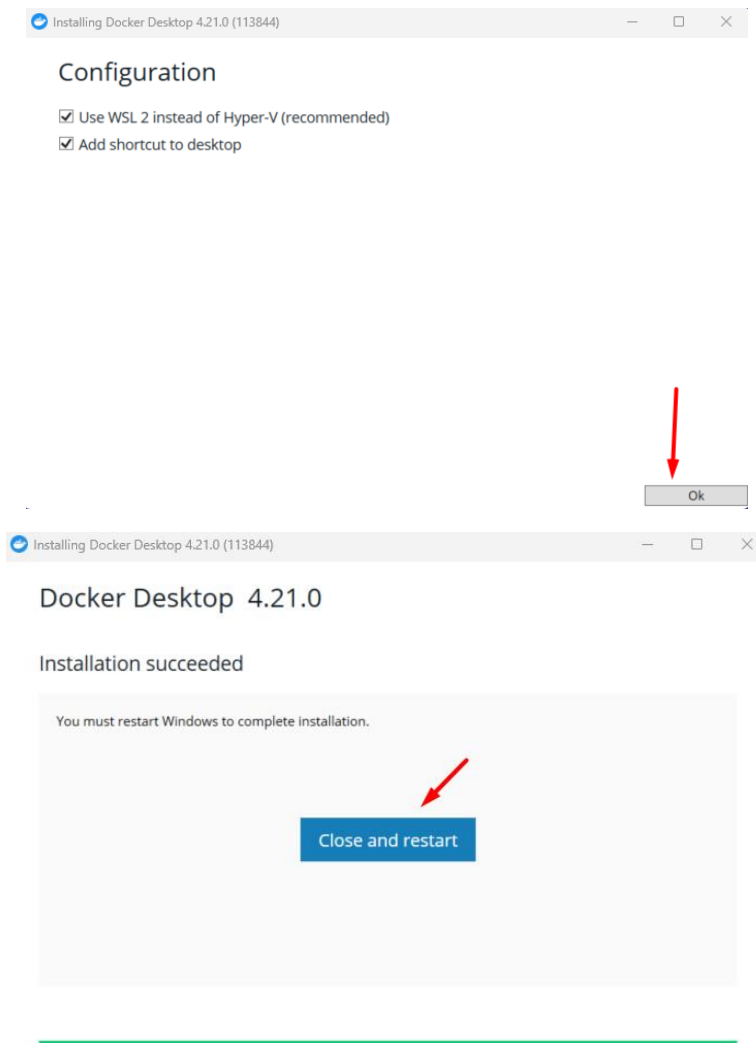
Linux : <https://docs.docker.com/desktop/install/ubuntu/>

- Instalación gnome
  - `sudo apt install gnome-terminal`
- Eliminar instalaciones previas
  - `sudo apt remove docker-desktop`
- Para finalizar ejecute
  - `rm -r $HOME/.docker/desktop`
  - `sudo rm /usr/local/bin/com.docker.cli`
  - `udo apt purge docker-desktop`
- Instalacion docker Desk
  - Actualizacion repositorio : <https://docs.docker.com/engine/install/ubuntu/#set-up-the-repository>
    - `sudo apt-get update`
    - `sudo apt-get install ca-certificates curl gnupg`
  - Add Docker's official GPG ke
    - `sudo install -m 0755 -d /etc/apt/keyrings`
    - `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg -dearmor -o /etc/apt/keyrings/docker.gpg`
    - `sudo chmod a+r /etc/apt/keyrings/docker.gpg`
  - Use the following command to set up the repository:
    - `echo \`
    - `"deb [arch="$(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

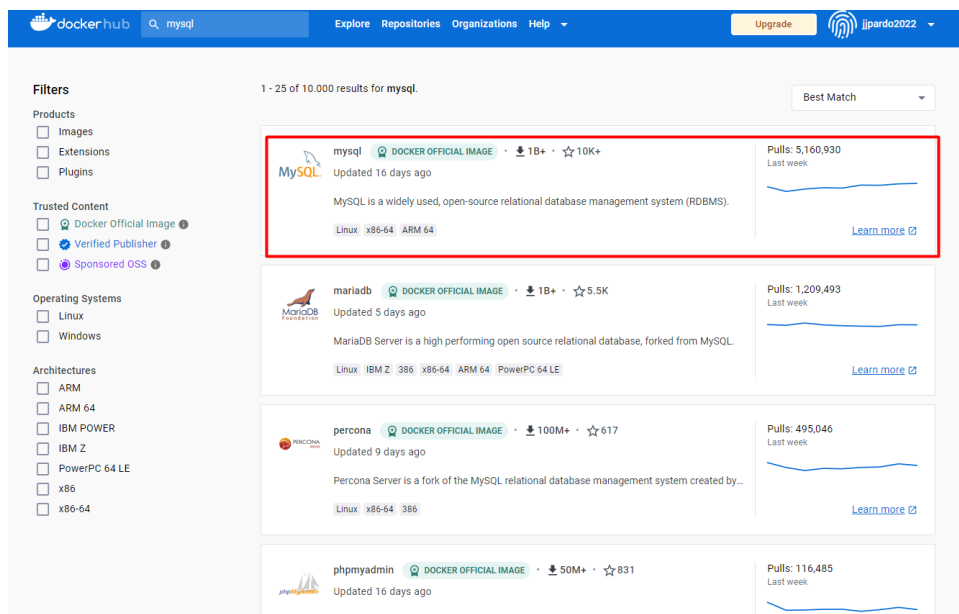


- Ejecute el Asistente de instalación

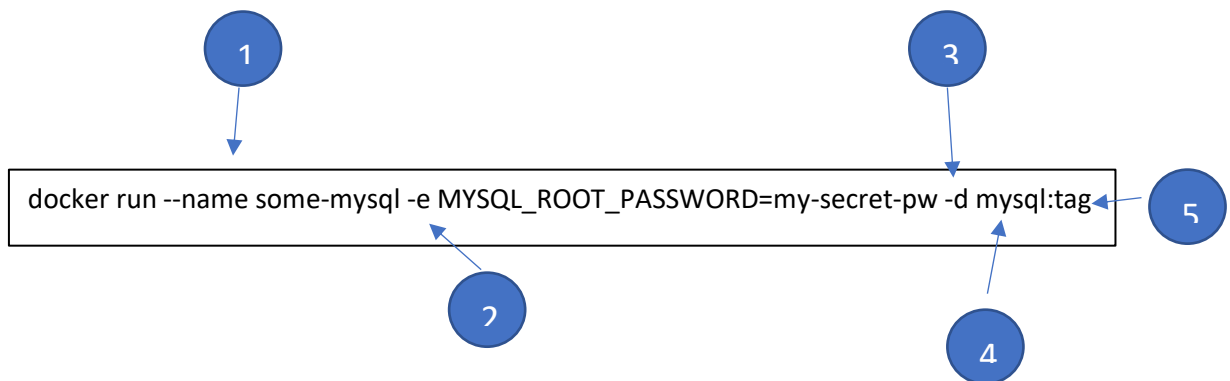


### 2.1.2. Configuración de Imagen Mysql en docker

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



- 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 desees 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`

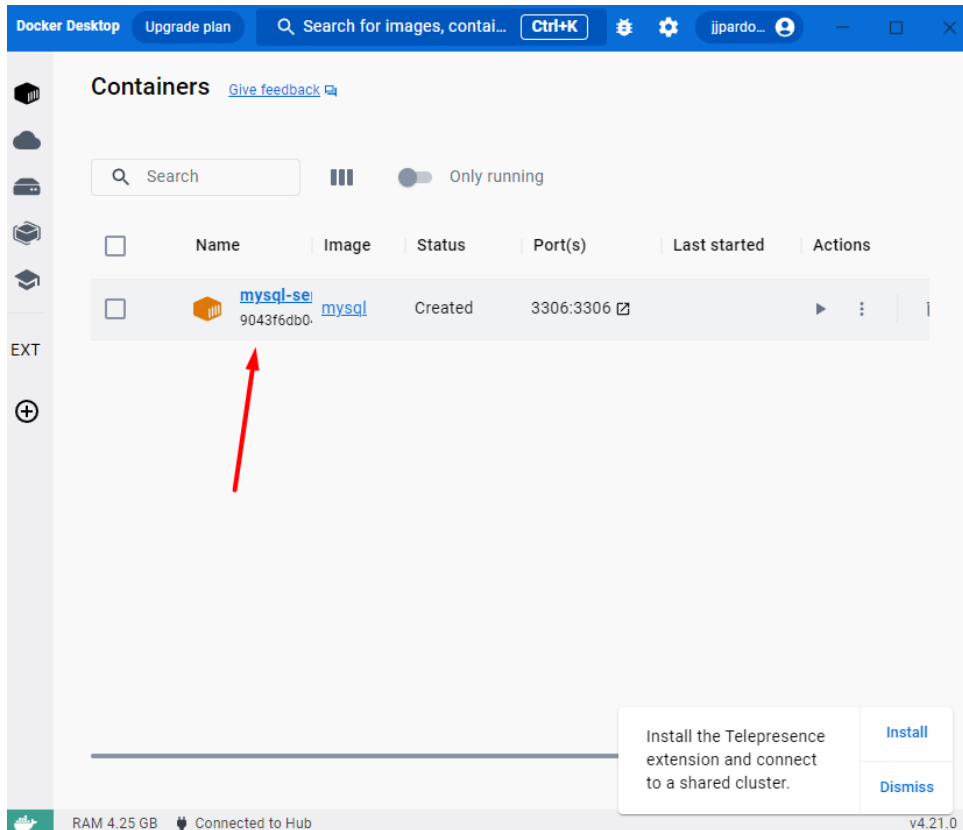
Si el contenedor de Mysql no existe lo descarga de forma automatica.



```

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-tok
ens/
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
41cc33cd9912: Pull complete
07bbc8bdf52a: Pull complete
6d88f83726a9: Download complete
cf5c7d5d33f7: Downloading [====>] 6.454MB/58.6MB
9db3175a2a66: Download complete
feaadeb27fa9: Downloading [>] 1.071MB/56.57MB
cf91e7784414: Waiting
b1778db1c329: 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 búfer de protocolo	proto		Web/gRPC
Archivo de la solución	sln,solution		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 Infrastructure:

```
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:

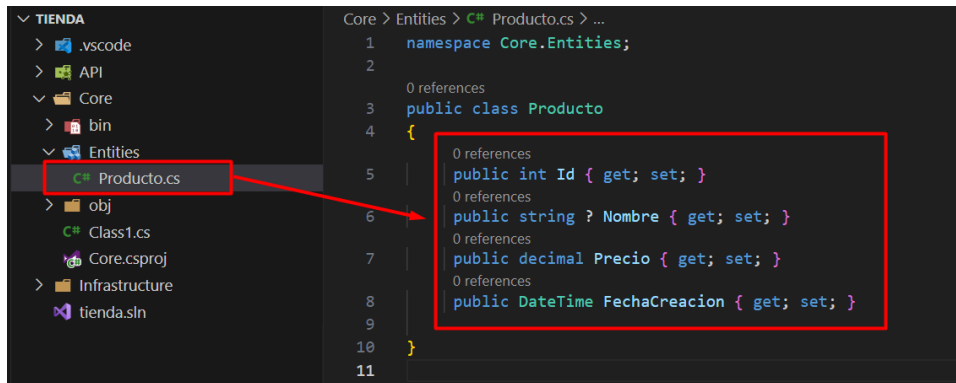
```
PS D:\projectsNetCore\tienda> cd ..\Infrastructure\  
PS D:\projectsNetCore\tienda\Infrastructure> dotnet add reference ..\Core\  
Se ha agregado la referencia "..\Core\Core.csproj" al proyecto.  
PS D:\projectsNetCore\tienda\Infrastructure> |
```

```
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> |
```

```
> .vscode  
> API  
> Core  
> Infrastructure  
> tienda.sln
```

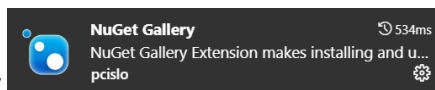
### 3.2. Proyecto Base (Entidades)

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 que se muestran en las imágenes siguientes:

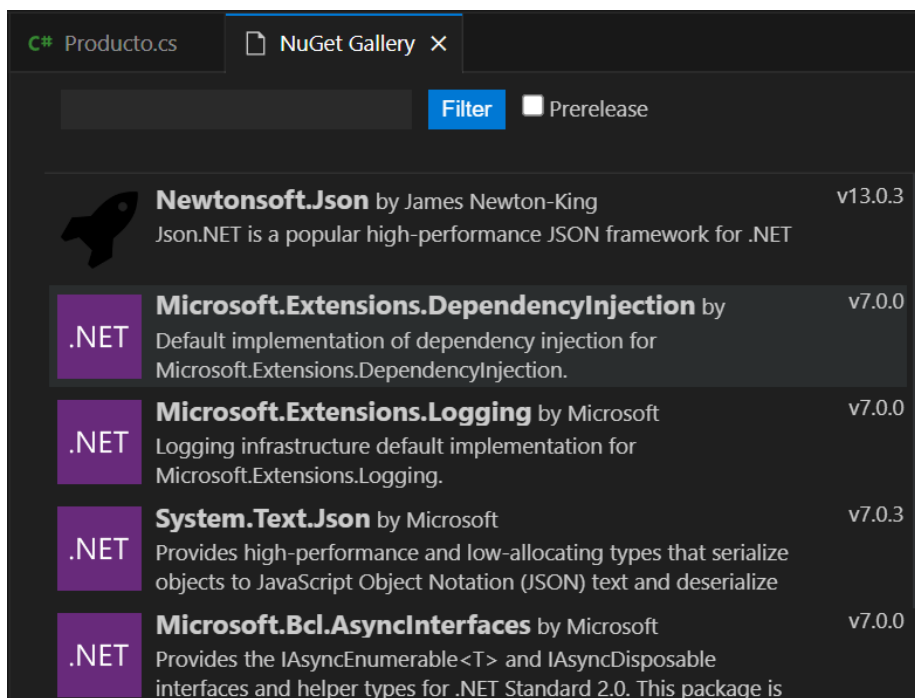
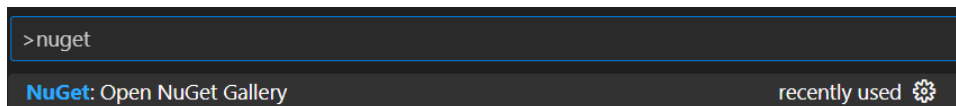


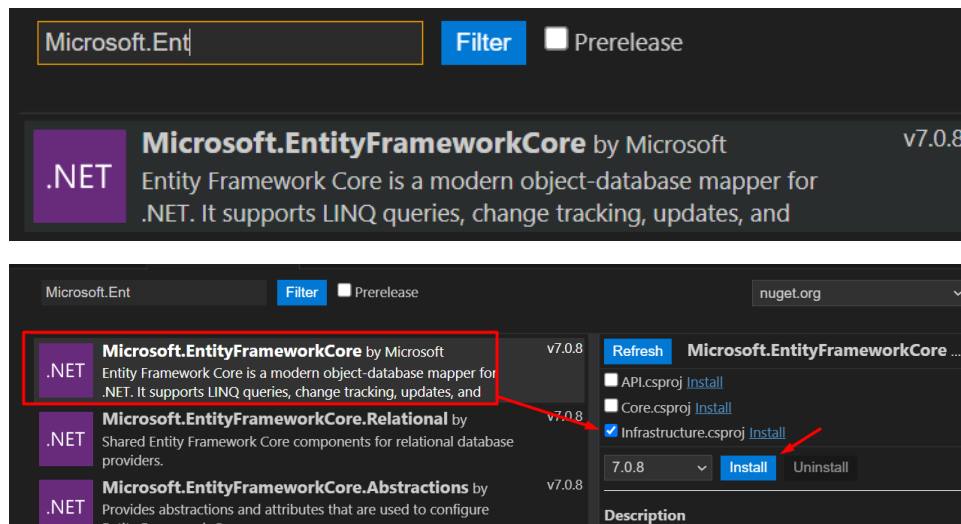
A continuación, se deben agregar los paquetes de entity framework para poder trabajar con todo el esquema de punto net para esto podemos hacerlo de diferentes maneras desde la línea de comandos o haciendo uso de visual studio code. En esta ocasión vamos a realizar la instalación de estos paquetes usando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

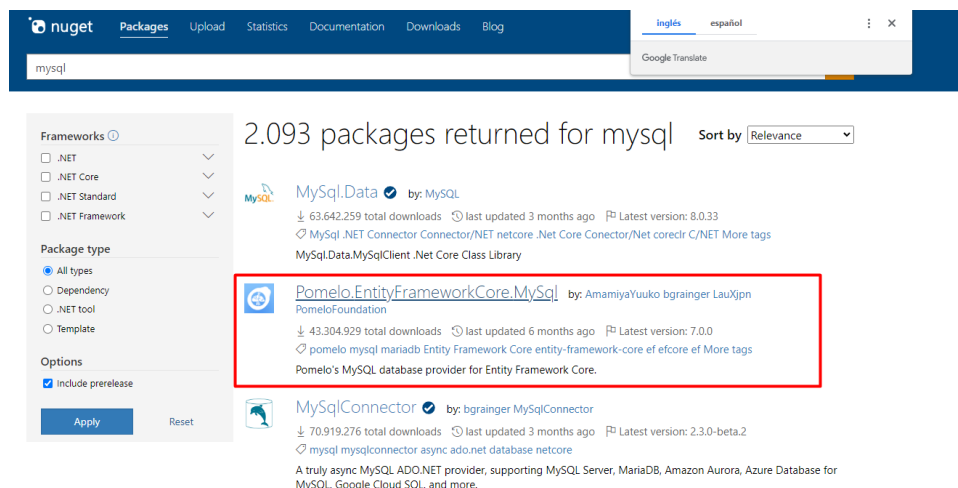


de Nuget gallery

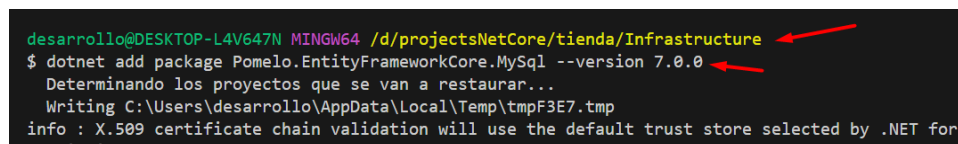




Instalación proveedor Mysql : para realizar la instalación del proveedor de Mysql debemos realizarlo a través de la línea de comandos, para eso debemos ingresar a la página web de los nugets. <https://www.nuget.org/>. quién es buscador escribimos la palabra mysql. Cuando se realiza la búsqueda aparecen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

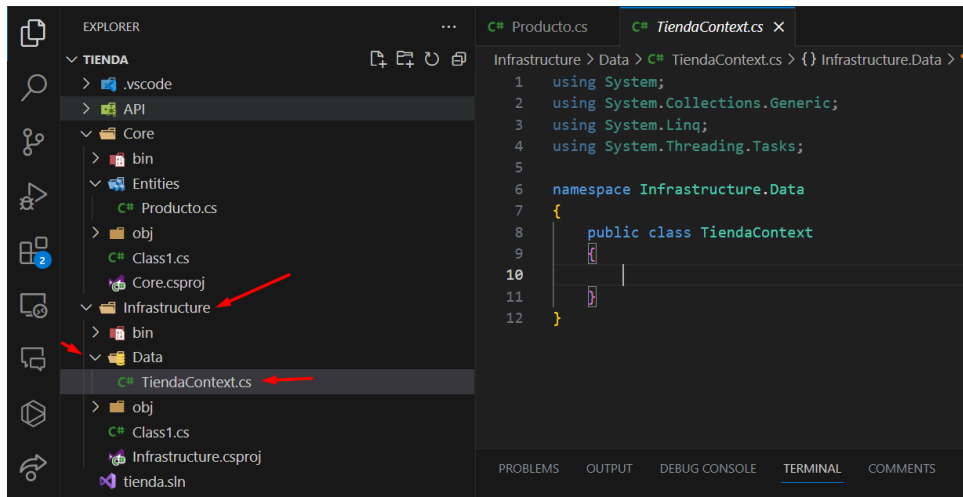


Y por último ejecutamos el comando **dotnet add package Pomelo.EntityFrameworkCore.MySql --version 7.0.0** en la terminal de visual studio code. Es importante tener en cuenta que la instalación la debemos realizar dentro de la carpeta infraestructura.

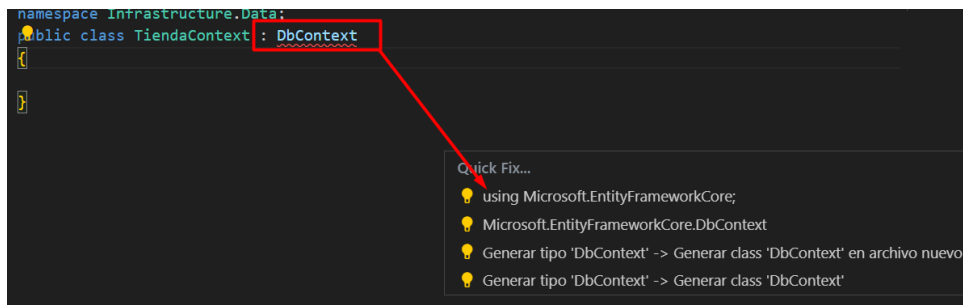


El siguiente paso que vamos a realizar es la creación de la clase de contexto la cual se utilizará para la conexión a la base de datos. Para una organización efectiva del proyecto la clase de contexto se crea dentro de la carpeta infraestructura y una carpeta interna llamada data. Como buena práctica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

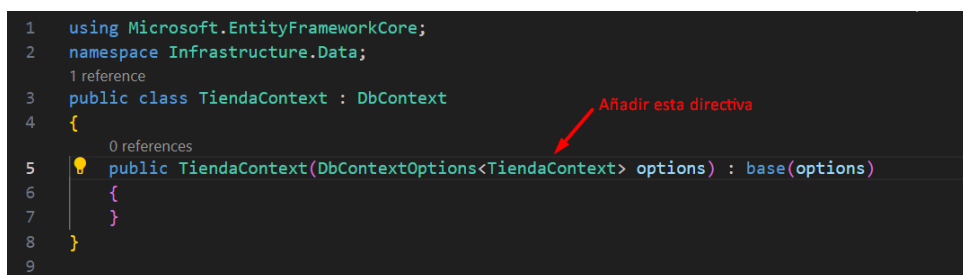




La clase tiendaContext heredará de la clase DbContext la cual pertenece al entity framework core.



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



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

```

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.

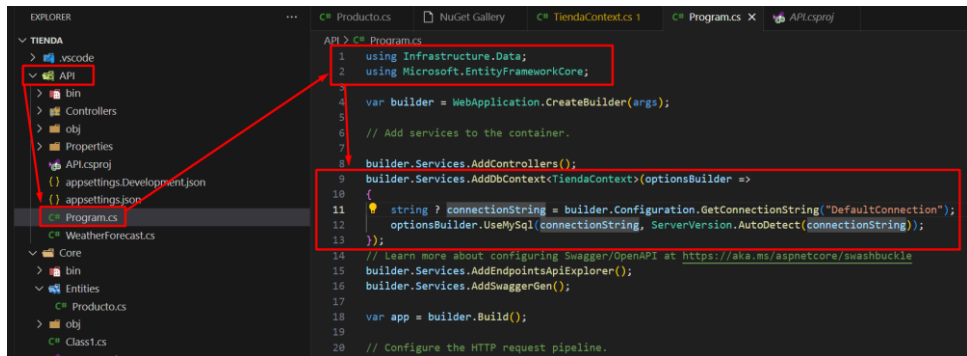
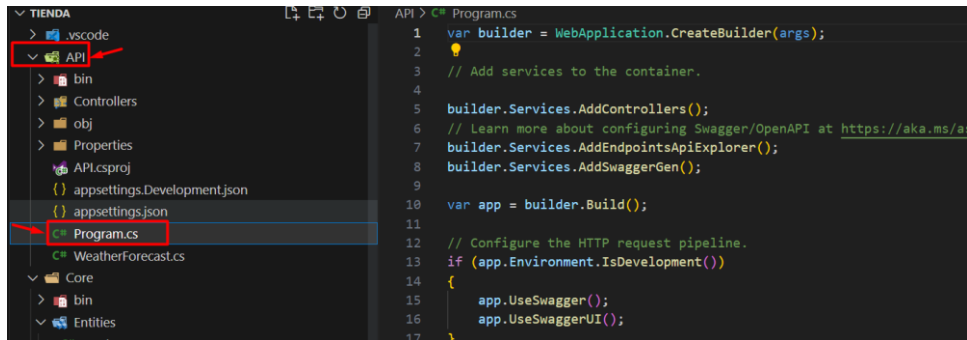
El contenedor de inyección de dependencias mantiene un registro de las dependencias y sus configuraciones correspondientes. Cuando se solicita una instancia de un objeto que tiene dependencias, el contenedor se encarga de crear y proporcionar esas dependencias automáticamente.

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.



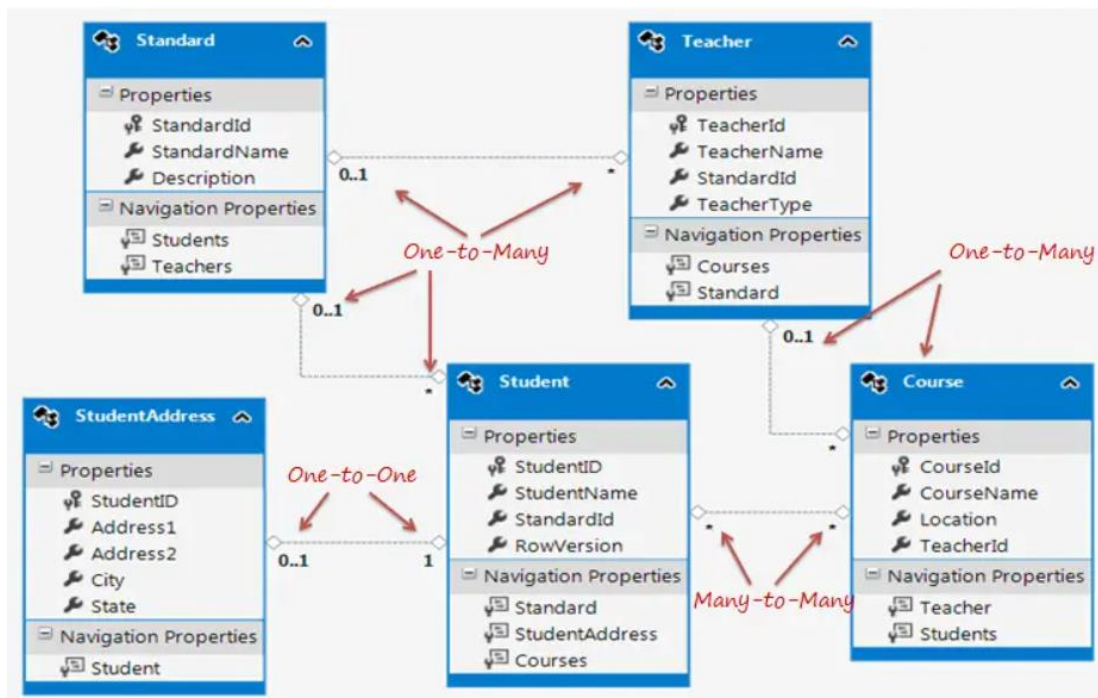
```
builder.Services.AddDbContext<TiendaContext>(optionsBuilder =>
{
    string ? connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    optionsBuilder.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString));
});
```

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 Uno

Como se puede observar en la figura anterior, Student y StudentAddress tienen una relación Uno a Uno (cero o uno). Un estudiante puede tener solo una dirección o ninguna. Entity Framework agrega la propiedad de navegación de referencia Student a la entidad StudentAddress y la entidad de navegación StudentAddress a la entidad Student. Además, la entidad StudentAddress tiene la propiedad StudentId como clave primaria y clave externa (ForeignKey), lo que hace que sea una relación uno a uno.

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

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



```

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>

```

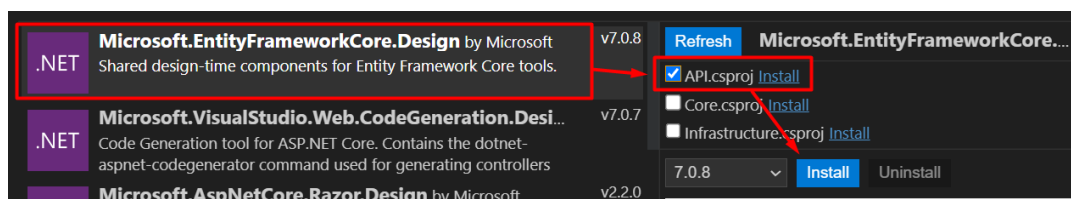
Para poder habilitar las migraciones debemos encontrarnos dentro de la carpeta principal de tienda y ejecutar el siguiente comando: `dotnet ef migrations add InitialCreate --project ./Infrastructure/ --startup-project ./API/ --output-dir ./Data/Migrations`

```

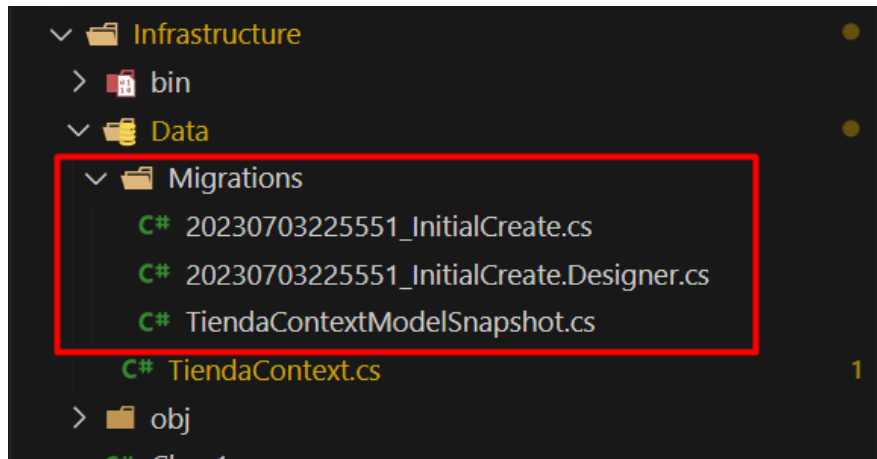
PS D:\projectsNetCore\tienda> dotnet ef migrations add InitialCreate --project ./Infrastructure/ --startup-project ./API/ --output-dir ./Data/Migrations
Build started...
Build succeeded.
Your startup project 'API' doesn't reference Microsoft.EntityFrameworkCore.Design. This package is required for the 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+8fffc3fe3d 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

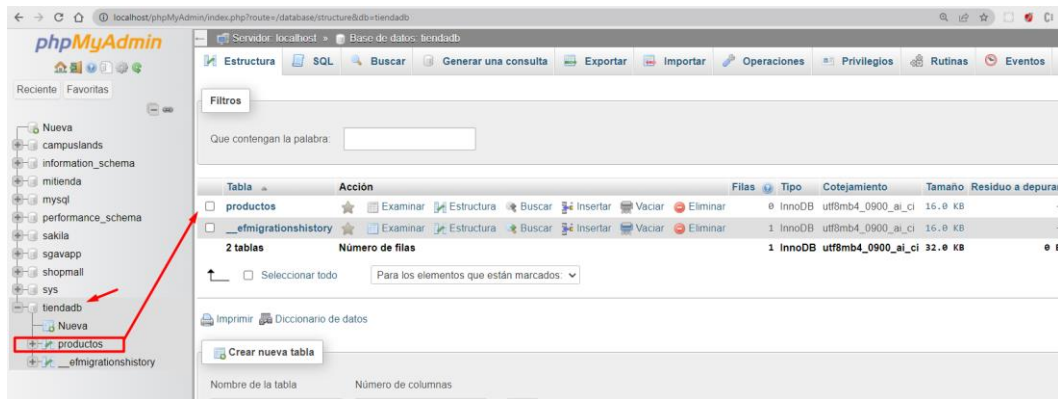
Tiempo transcurrido 00:00:01.07
PS D:\projectsNetCore\tienda>
```

- 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.





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

siguientes líneas de código después de:

```
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");
    }
}
```

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.

```
CREATE DATABASE `tiendadb`;
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
```

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	Metodo Fluent api	Uso
Configuracion 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 atributos de anotacion 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.
	ToTable()	Configura la tabla de la base de datos a la que se asigna la entidad.
Property Configuration	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.

### 3.6. 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.

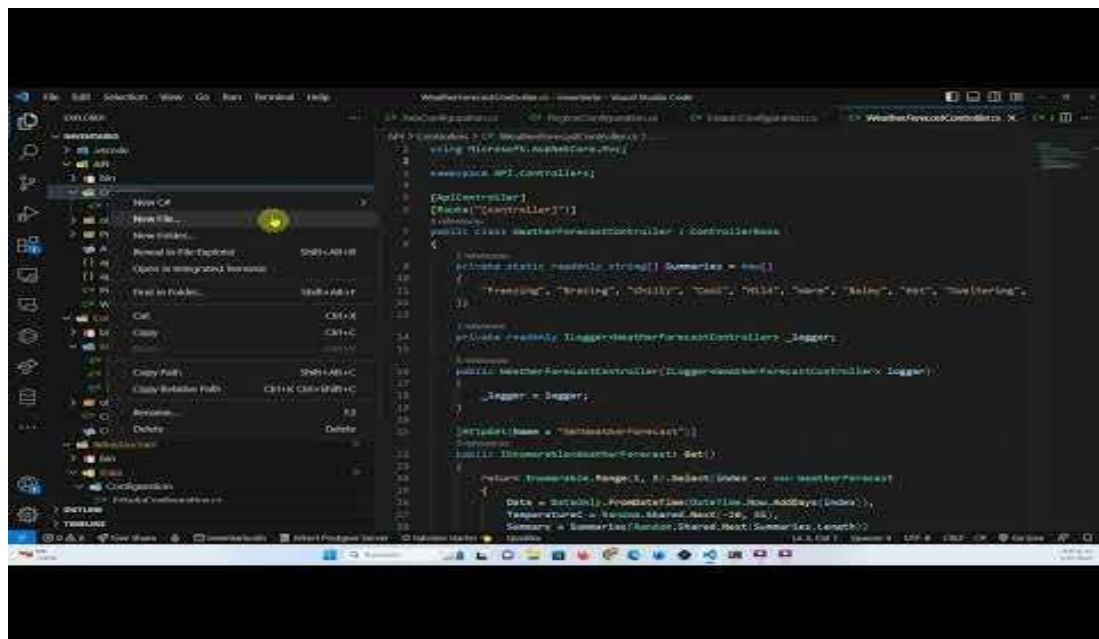
El patrón de Unidad de Trabajo se utiliza comúnmente junto con el patrón de Repositorio para manejar la interacción entre la lógica de negocio de la API y la capa de acceso a datos. La Unidad de Trabajo se encarga de orquestar las operaciones de los repositorios y asegura que se realicen en el contexto 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
    }
}
```

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();
```

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.

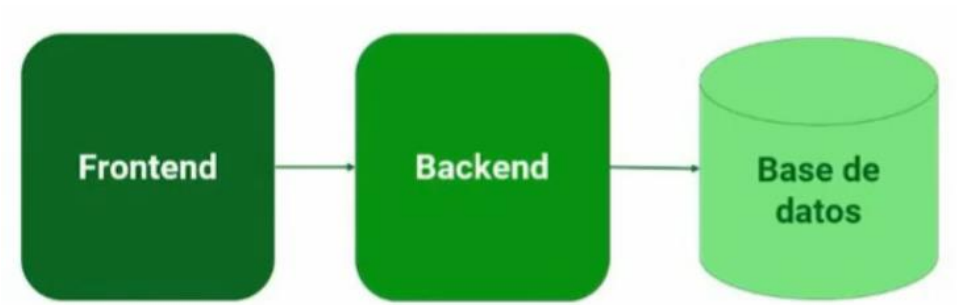
**CORS (Cross-Origin Resource Sharing)** es un mecanismo de seguridad implementado en los navegadores web que controla las solicitudes HTTP realizadas desde un origen (dominio, protocolo y puerto) a otro origen distinto. CORS impone restricciones sobre las solicitudes de recursos (como API) que se originan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

Si estás desarrollando una aplicación web 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 '\*'.

## 3.2. Conexión a bases de datos

### 3.2.1. Arquitectura Básica



Crear proyecto

```
dotnet new webapi --framework net7.0 -o WebAPITiendaV
```

#### Notas

Por aquí les dejo los comandos con las versiones estables y más recientes al 18 mayo de 2022:

- Instalación de EF  
`dotnet add package Microsoft.EntityFrameworkCore --version 6.0.5`
- Instalación para crear una base de datos en memoria  
`dotnet add package Microsoft.EntityFrameworkCore.InMemory --version 6.0.5`
- Instalación para conectarnos con el motor SQL Server  
`dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 6.0.5`
- Instalación para conectarnos con PostgreSQL  
`dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL --version 6.0.4`