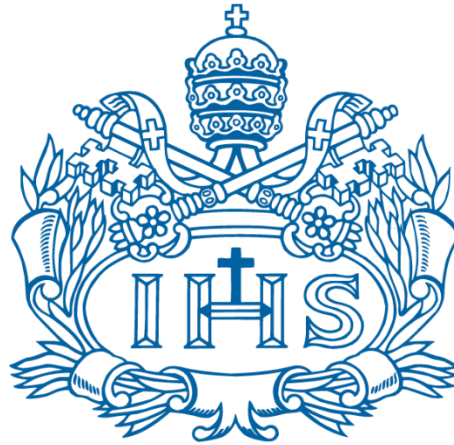




Pontificia Universidad Javeriana



Laboratorio 1

Arquitectura de Software

Ingeniería de Sistemas

Profesor:

Andrés Armando Sánchez Martín

Estudiantes:

Juan Sebastian Galeano González

Juan Esteban Reyes Tausa

Esteban Hernando Pedraza Solano

Bogotá



14 de marzo de 2024

Contenido

Introducción.....	2
Marco Conceptual.....	3
.Net.....	3
SQL Server 2022.....	4
Swagger 3.....	5
Docker.....	6
Docker Compose.....	7
Diseño	8
Procedimiento	10
1. Crear el Repositorio.....	10
2. Instalar SQL Server 2019 Express modo Básico	12
3. Instalar SQL Server Management Studio 18	13
4. Crear la base de datos	14
5. Crear las tablas según el modelo	16
6. Instalar Visual Studio Community 2022.....	19
7. Clonar el repositorio local	20
8. Visual Studio Community 2022	20
9. Creación Docker Compose	38
10. Hacer push al repositorio	40
11. Crear tag y hacer un release.....	42
Conclusiones y lecciones aprendidas	44
Referencias	45

Introducción

Este proyecto se centra en la implementación de una aplicación monolítica utilizando el patrón de diseño Modelo-Vista-Controlador (MVC) junto con el patrón Data Access Object (DAO). La aplicación se construirá sobre la plataforma .NET 7, integrando una base de datos



MS SQL Server 2022, y exponiendo funcionalidad a través de una API REST documentada con Swagger 3.

El principal objetivo del proyecto es desarrollar un conjunto completo de operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre un modelo de datos relacional, el cual describe entidades como personas, estudios, profesiones y teléfonos. Estas entidades están interrelacionadas y serán gestionadas a través de vistas y endpoints RESTful, brindando una interfaz eficiente para manipular los datos.

Además de la implementación del código fuente, se entregarán scripts SQL (DDL y DML) para crear y poblar la base de datos, así como una documentación exhaustiva que incluye el marco conceptual, el diseño, los pasos para configurar el ambiente y las conclusiones extraídas del desarrollo del proyecto.

Marco Conceptual

.Net

.NET es una plataforma de desarrollo abierta, gratuita y multiplataforma, diseñada para construir diversos tipos de aplicaciones. Acepta múltiples lenguajes de programación, siendo C# el más popular. Esta plataforma proporciona un entorno de ejecución de alto rendimiento que muchas aplicaciones a gran escala utilizan en producción.



El diseño de .NET se enfoca en la productividad, rendimiento, seguridad y confiabilidad. Ofrece administración automática de memoria a través de su recolector de elementos no utilizados (GC) y promueve la seguridad de tipos y la protección de la memoria mediante un uso estricto de los compiladores de lenguajes. Las aplicaciones desarrolladas con .NET pueden ejecutarse de forma concurrente gracias a primitivas de simultaneidad como `async/await` y `Task`. Además, .NET incluye un amplio conjunto de bibliotecas optimizadas para diferentes arquitecturas y sistemas operativos.

Uno de los aspectos más importantes de .NET es su adaptabilidad para distintos dominios de programación, desde aplicaciones en la nube, hasta sistemas cliente o de juegos. Esto es posible gracias a implementaciones especializadas como ASP.NET Core para aplicaciones web y Windows Forms para aplicaciones de escritorio.

En términos de componentes, .NET incluye un entorno de ejecución que ejecuta el código de las aplicaciones, bibliotecas que proveen utilidades como el análisis de JSON, un compilador para traducir código fuente a ejecutable y diversas herramientas para la creación y monitoreo de aplicaciones.

Por último, la plataforma es mantenida por Microsoft y la comunidad a través de la fundación .NET y GitHub, con actualizaciones periódicas que garantizan que los usuarios puedan implementar aplicaciones seguras y confiables en producción.

SQL Server 2022

SQL Server 2022 es la última versión importante del producto SQL Server de Microsoft. Esta versión se presenta como una plataforma de datos híbrida, impulsada por innovaciones en seguridad, rendimiento, disponibilidad y virtualización de datos. Está diseñada para ayudar a las organizaciones a modernizar su infraestructura de datos, proporcionando capacidades conectadas a la nube y permitiendo escenarios de datos híbridos.

Entre las principales capacidades de SQL Server 2022 se encuentran el aumento del rendimiento sin necesidad de cambios en el código, la protección de la integridad de los datos mediante blockchain y la escalabilidad mejorada. Además, SQL Server 2022 permite a los desarrolladores conectar datos relacionales con data lakes que contienen archivos Parquet y tablas Delta utilizando Azure o proveedores de almacenamiento compatibles con S3.

Uno de los aspectos más destacados de SQL Server 2022 es su flexibilidad en las opciones de despliegue. Se puede implementar en sistemas operativos Windows y Linux, así como en máquinas virtuales y plataformas de contenedores como Kubernetes. Los desarrolladores también cuentan con nuevas capacidades en el lenguaje T-SQL, lo que facilita la creación de aplicaciones que aprovechan la plataforma.

SQL Server 2022 ofrece una variedad de ediciones que permiten su uso en diferentes entornos, desde versiones de evaluación y desarrollo hasta versiones empresariales con



acceso ilimitado a recursos. Además, introduce un nuevo modelo de licenciamiento basado en el pago por uso, lo que permite a las organizaciones adaptar el costo a su uso real.

Con esta versión, Microsoft ha eliminado algunas características que estaban presentes en versiones anteriores, como los entornos de ejecución para R, Python y Java, y ha retirado funcionalidades como Polybase con Hadoop y Stretch Database. A cambio, ha introducido mejoras significativas en inteligencia de consultas, virtualización de datos y seguridad.

Swagger 3

Swagger es un conjunto de herramientas para la creación, documentación y consumo de API REST, el cual sigue la OpenAPI Specification (OAS). Swagger 3.0 se basa en la versión 3.0 de esta especificación, que es un estándar ampliamente aceptado para describir interfaces HTTP, permitiendo que tanto personas como máquinas entiendan y se integren con APIs de forma estandarizada.

Swagger no solo proporciona una forma de definir APIs de manera consistente y clara, sino que también ofrece diversas herramientas para generar documentación interactiva, automatizar pruebas, y generar código de cliente y servidor a partir de una descripción de API. Esto lo convierte en una herramienta fundamental para desarrolladores y equipos que buscan implementar servicios web con estándares abiertos.

Componentes principales de Swagger 3:

- a) OpenAPI Specification (OAS): La OpenAPI Specification es un estándar independiente del lenguaje que define la estructura de una API. Un documento de OpenAPI describe las operaciones disponibles en una API, los datos que espera recibir y los formatos que produce. Swagger utiliza esta especificación como base para todas sus herramientas.
- b) Documentación interactiva: Una de las características clave de Swagger es la capacidad de generar una interfaz gráfica y dinámica donde los usuarios pueden probar directamente los endpoints de la API. Esta documentación interactiva se basa en el archivo de definición de OpenAPI.
- c) Generación de código: Swagger facilita la generación de código tanto del lado del cliente como del servidor. Esto se hace automáticamente a partir del archivo de especificación, eliminando errores manuales y asegurando la consistencia entre el código y la API.
- d) Validación y pruebas automáticas: Gracias a la definición precisa de las operaciones y los datos esperados por la API, Swagger permite generar automáticamente pruebas que verifican la conformidad del servicio con su especificación.
- e) Soporte de múltiples formatos: Swagger soporta tanto JSON como YAML para la definición de las APIs. Este soporte flexible permite que los desarrolladores elijan el



formato con el que se sientan más cómodos o que sea más adecuado para su flujo de trabajo.

Swagger 3 es compatible con tecnologías como .NET y facilita la creación de APIs RESTful. En un entorno .NET, Swagger se integra de manera fluida a través de bibliotecas como Swashbuckle y NSwag, que permiten generar automáticamente la definición de OpenAPI a partir del código .NET, lo que simplifica el desarrollo y mantenimiento de las APIs.

Docker

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones mediante el uso de contenedores. Estos contenedores permiten separar las aplicaciones de la infraestructura, facilitando una entrega de software más rápida y eficiente. Docker ayuda a gestionar tanto las aplicaciones como la infraestructura con los mismos procesos, reduciendo el tiempo entre la creación de código y su implementación en producción.

Docker permite empaquetar y ejecutar aplicaciones dentro de contenedores, entornos aislados y ligeros que contienen todo lo necesario para que la aplicación funcione. Esto asegura que los contenedores se ejecuten de la misma manera en cualquier entorno, sin depender de las configuraciones del sistema host. Los contenedores son ideales para compartir, probar y desplegar aplicaciones de forma coherente.

Docker es útil para la entrega rápida y consistente de aplicaciones. Facilita la integración continua (CI) y entrega continua (CD) al permitir a los desarrolladores trabajar en contenedores estandarizados. También es altamente portable, permitiendo que los contenedores se ejecuten en laptops, servidores locales o en la nube, y facilita el escalado dinámico según las necesidades del negocio.

Docker sigue una arquitectura cliente-servidor. El cliente Docker envía comandos al Docker daemon (dockerd), que se encarga de construir y gestionar contenedores, imágenes y otros objetos. El cliente puede interactuar con el daemon tanto de forma local como remota a través de una API REST. Docker también incluye herramientas como Docker Compose, que permite gestionar aplicaciones formadas por varios contenedores.

Los registros almacenan imágenes de Docker. Docker Hub es el registro público predeterminado, pero es posible configurar registros privados. Al usar comandos como `docker pull` o `docker push`, se pueden descargar o subir imágenes desde y hacia estos registros.

Los principales objetos de Docker son imágenes y contenedores. Las imágenes son plantillas de solo lectura que se utilizan para crear contenedores. Los contenedores son instancias ejecutables de estas imágenes, con su propio sistema de archivos, redes y almacenamiento, permitiendo la ejecución de aplicaciones de manera aislada del sistema host.



Docker Compose

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones compuestas por varios contenedores. Facilita el desarrollo y despliegue al manejar múltiples servicios a la vez mediante un archivo de configuración en formato YAML.

Compose permite gestionar toda la pila de tu aplicación, incluyendo servicios, redes y volúmenes, desde un solo archivo YAML. Con un solo comando (`docker-compose up`), se pueden crear y arrancar todos los servicios definidos en dicho archivo de manera automática.

Docker Compose es útil en diferentes entornos, ya sea en producción, desarrollo, pruebas o integraciones continuas (CI). Además, es compatible con la gestión del ciclo de vida completo de una aplicación, desde su arranque hasta su parada o reconstrucción.

Compose permite:

Iniciar, detener y reconstruir servicios.

- Ver el estado de los servicios en ejecución.
- Monitorizar los logs de los servicios.
- Ejecutar comandos temporales en servicios específicos.

Docker Compose es una herramienta esencial para manejar aplicaciones de múltiples contenedores de manera eficiente y coherente.

En el marco conceptual de este laboratorio, cada tecnología o herramienta seleccionada tiene un propósito claro para el desarrollo eficiente de aplicaciones modernas. .NET nos permitirá construir aplicaciones robustas y escalables, aprovechando sus capacidades de manejo de memoria, seguridad y soporte para múltiples lenguajes de programación. SQL Server 2022 servirá como nuestra plataforma de gestión de datos, aportando nuevas funcionalidades para la integración de datos híbridos y mejorando el rendimiento sin necesidad de modificar código.

Swagger 3 facilitará la creación, documentación y pruebas de nuestras APIs REST, garantizando una integración fluida entre los diferentes servicios y proporcionando una interfaz interactiva para validación. Docker y Docker Compose serán esenciales para la contenedorización y orquestación de nuestros servicios, permitiendo que las aplicaciones se ejecuten de forma consistente en cualquier entorno. Docker Compose, en particular, nos ayudará a gestionar aplicaciones multicontenedor, simplificando el arranque y la coordinación de los servicios en desarrollo, pruebas y producción.

Este conjunto de herramientas nos proporcionará una plataforma sólida para gestionar aplicaciones modernas, asegurando eficiencia, seguridad y escalabilidad en todo el ciclo de vida del desarrollo.

Diseño

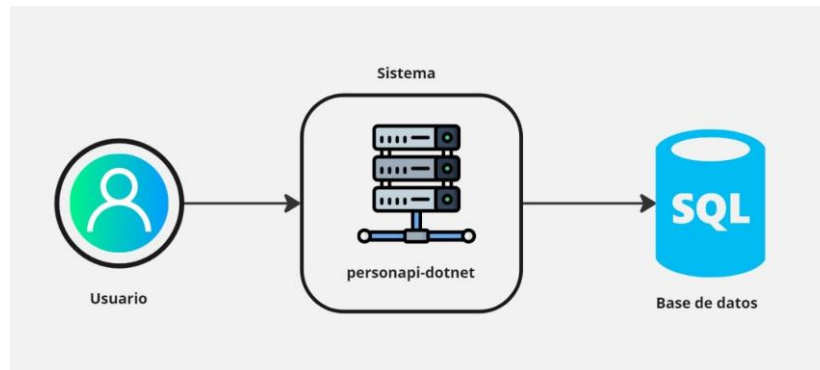


Diagrama de contexto

Arquitectura general: El sistema se diseñó como una aplicación monolítica con el patrón MVC, donde:

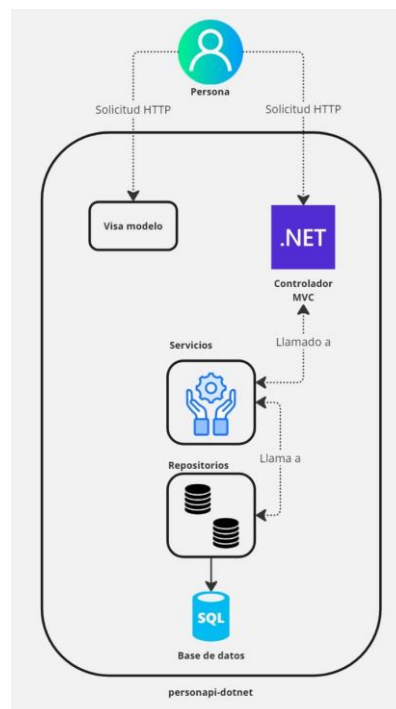


Diagrama de arquitectura general



Modelo: Incluye las clases que representan las entidades y la lógica de negocio, generadas a partir de la base de datos utilizando Entity Framework.

Vista: Compuesta por las páginas web y vistas que muestran la información al usuario, utilizando Razor para el renderizado de las plantillas.

Controlador: Maneja la comunicación entre el modelo y la vista, gestionando las solicitudes del usuario y devolviendo la respuesta adecuada.

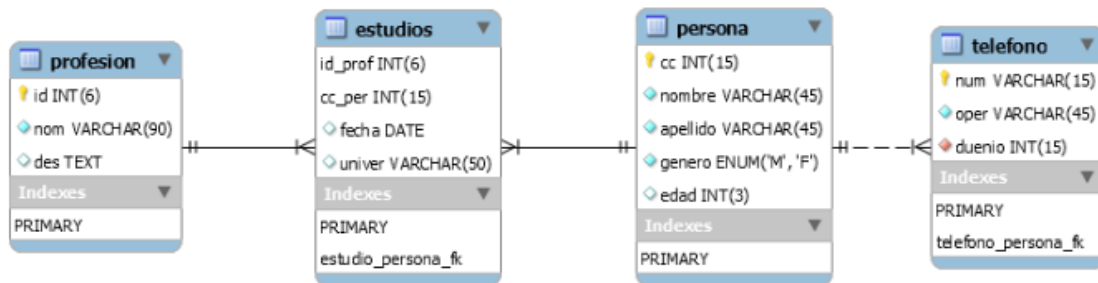
Diseño de la base de datos: Se implementó un esquema relacional con las siguientes tablas:

Persona: Información básica de cada persona.

Profesion: Detalles relacionados con las profesiones.

Estudios: Información sobre los estudios realizados.

Telefono: Datos de contacto. Las tablas están interrelacionadas para permitir consultas y operaciones CRUD complejas.



Modelo Base de Datos Proporcionado

Patrón DAO (Data Access Object): Los repositorios implementan el patrón DAO para interactuar con la base de datos de manera abstracta, lo que facilita cambiar el backend sin afectar el resto de la aplicación.



Implementación de la API REST con Swagger: La API se documentó con Swagger, proporcionando una interfaz interactiva para probar los endpoints y ver la estructura de los datos.

Manejo de la configuración: La cadena de conexión a la base de datos y otras configuraciones relevantes se gestionaron a través del archivo appsettings.json en .NET, lo que permite una administración centralizada de la configuración del sistema.

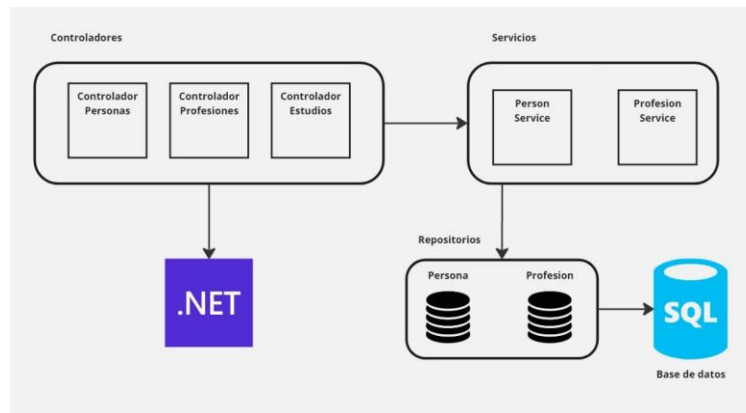


Diagrama de contenedores

Procedimiento

1. Crear el Repositorio.

El primer paso consiste en crear un repositorio Git público en GitHub (o cualquier sistema Git), nombrado personapi-dotnet, donde se almacenará y gestionará el código fuente del proyecto.




Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*


Owner * **Repository name ***


 el-sebas-galeano / personapi-dotnet

✔ personapi-dotnet is available.

Great repository names are short and memorable. Need inspiration? How about **studious-train** ?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: **None**


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None**

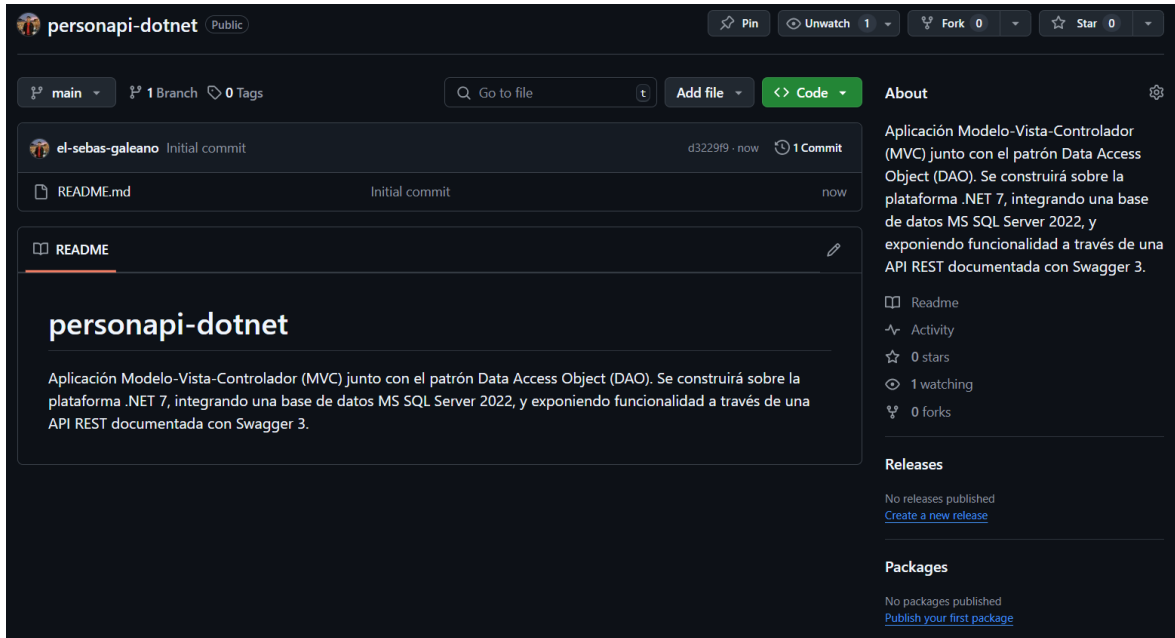
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

A continuación, se presenta una imagen del repositorio creado en GitHub, el cual se ha nombrado "personapi-dotnet".



2. Instalar SQL Server 2019 Express modo Básico

El segundo paso consiste en descargar SQL Server 2019 Express. Para ello, accedo al siguiente enlace: [Descarga de SQL Server 2019 Express](#). Allí, busco la opción adecuada para la descarga y sigo las instrucciones proporcionadas en la página para completar el proceso.

Microsoft® SQL Server® 2019 Express

Microsoft® SQL Server® 2019 Express es un sistema de administración de datos gratuito, eficaz y confiable que ofrece un almacén de datos completo y confiable para sitios web ligeros y aplicaciones de escritorio.

¡Importante! Al seleccionar un idioma a continuación, todo el contenido de la página cambiará completamente a ese idioma en forma dinámica.

Seleccionar idioma

Español

Descargar

Expandir todo | [Contraer todo](#)

▼ Detalles

Versión:

15.0.2000.5

File Name:

SQL2019-SSSEI-Expr.exe

Date Published:

18/4/2022

File Size:

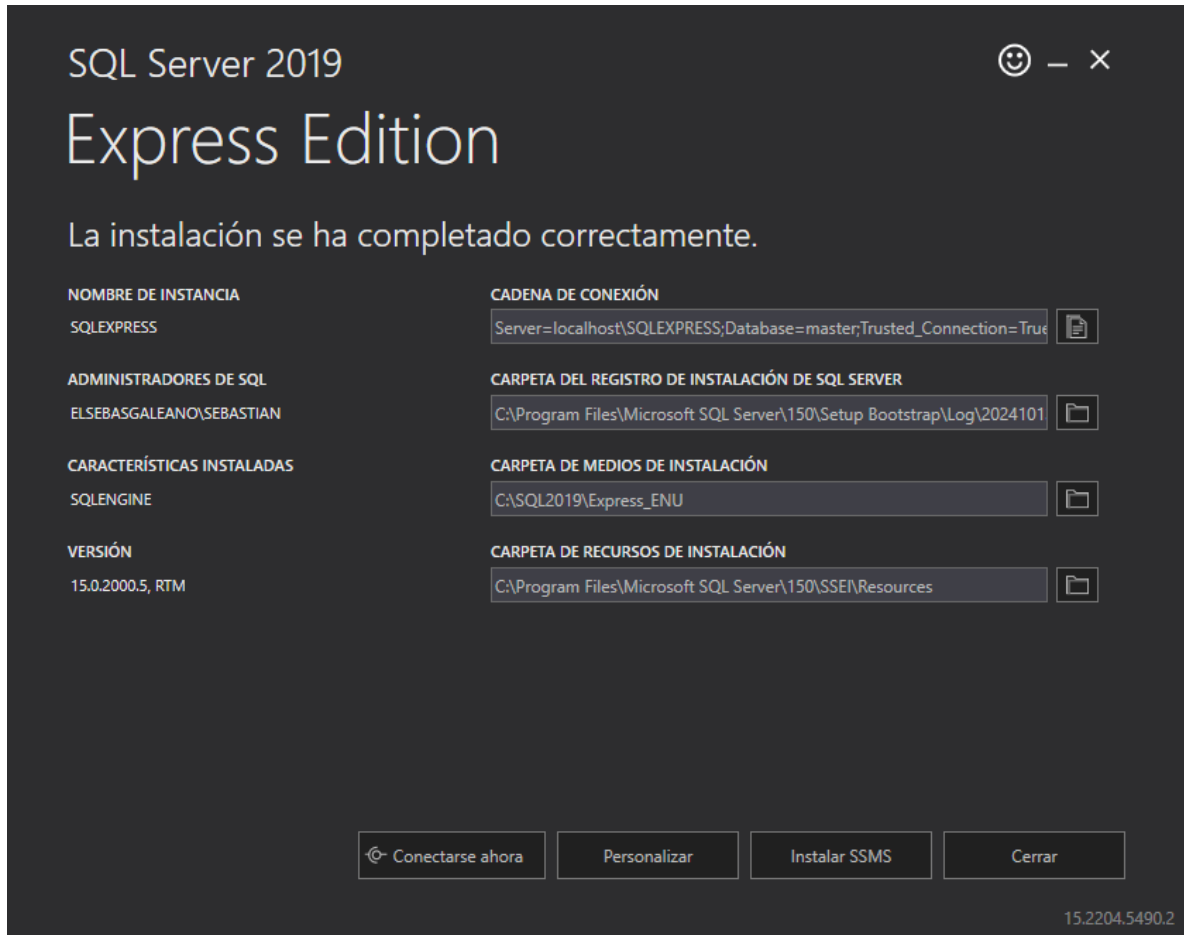
6.1 MB

Ejecuto el instalador de SQL Server 2019 Express que descargué previamente. Al iniciar el instalador, selecciono la opción de instalación en modo básico. Esta opción me permite realizar una instalación rápida y sencilla, configurando automáticamente las opciones recomendadas. Sigo las instrucciones en pantalla hasta completar la instalación. Una vez

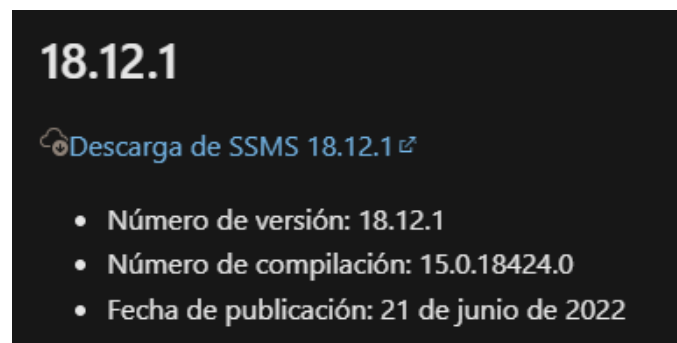


finalizada, confirmo que SQL Server 2019 Express se haya instalado correctamente en mi sistema.

3. Instalar SQL Server Management Studio 18

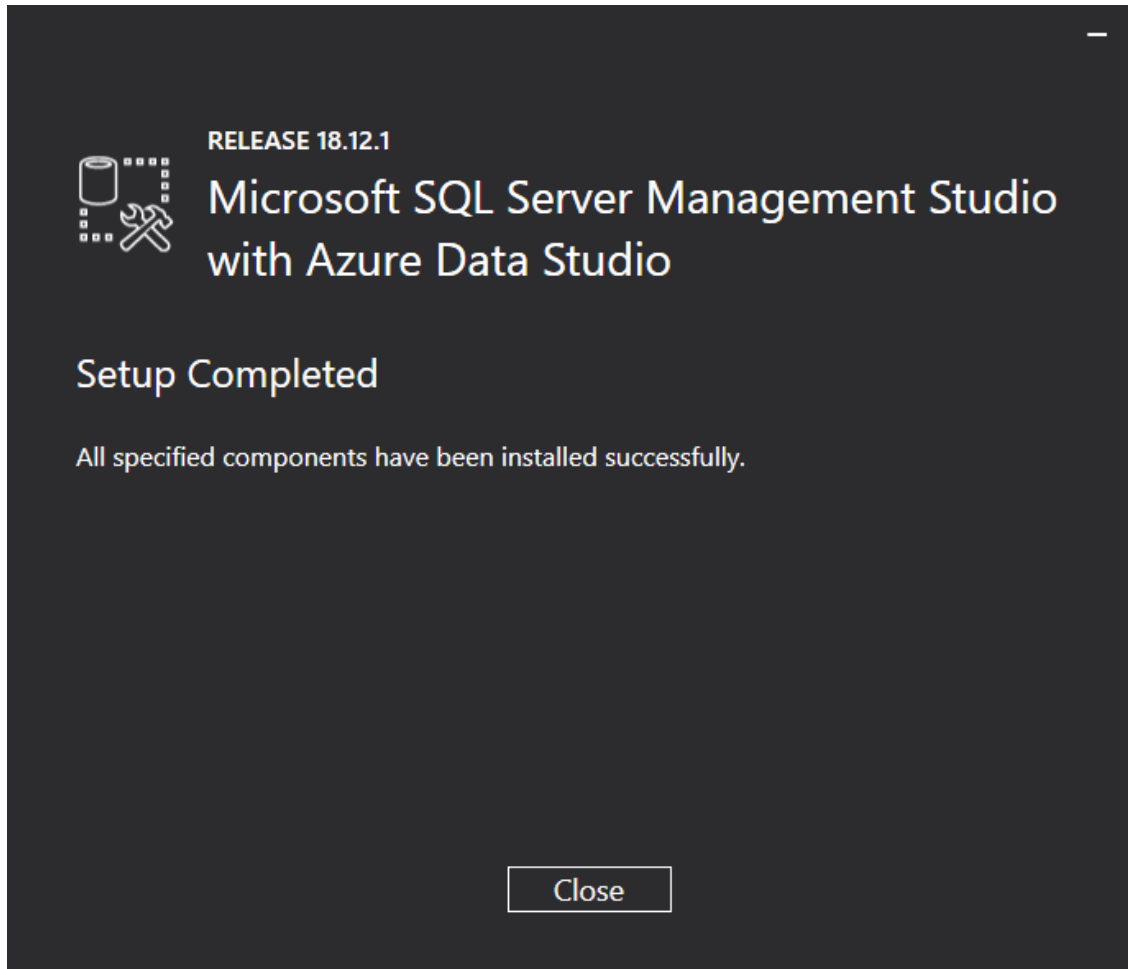


Descargo el instalador de SQL Server Management Studio (SSMS) 18.12.1 desde el siguiente enlace: [Documentación de SSMS](#).





Una vez que se completa la descarga, ejecuto el instalador. Sigo las instrucciones en pantalla para llevar a cabo la instalación de SSMS, lo que me permitirá administrar y gestionar mis bases de datos en SQL Server de manera eficiente.



4. Crear la base de datos

Hemos creado la base de datos `persona_db` en SQL Server Management Studio (SSMS). Para lograrlo, nos conectamos al servidor local, hicimos clic derecho en "Bases de Datos", seleccionamos "Nueva base de datos" e ingresamos el nombre `persona_db`. La base de datos ya está lista para su uso.



New Database

Select a page
General
Options
Filegroups

Script Help

Database name:

Owner:

☒ Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize	Path
persona_db	ROWS...	PRIMARY	8	By 64 MB, Unlimited	C:\...
persona_db_log	LOG	Not Applicable	8	By 64 MB, Unlimited	C:\...

Server: ELSEBASGALEANO\SQLEXPRESS
Connection: ELSEBASGALEANO\SEBASTIAN
[View connection properties](#)

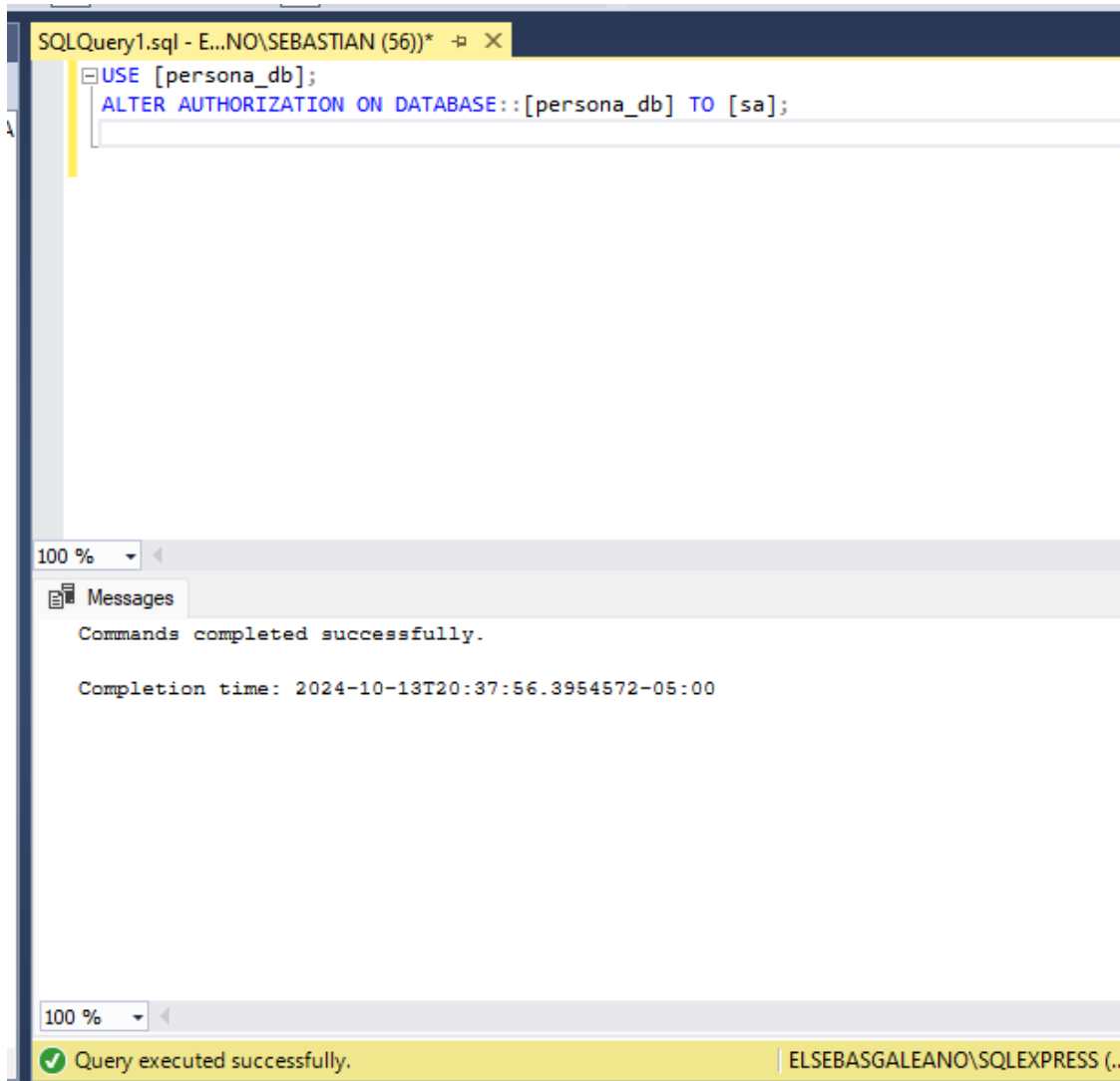
Progress
Ready

Add Remove

OK Cancel

Luego, se asignó la propiedad de la base de datos `persona_db` al usuario `sa`. Para hacerlo, ejecutamos el siguiente comando en el entorno de SQL Server Management Studio (SSMS):

```
SQLQuery1.sql - E...NO\SEBASTIAN (56)) * X  
USE [persona_db];  
ALTER AUTHORIZATION ON DATABASE::[persona_db] TO [sa];
```



5. Crear las tablas según el modelo

Empezamos por crear la tabla de "profesión" en la base de datos persona_db utilizando el siguiente script SQL:



```
SQLQuery1.sql - E...NO\SEBASTIAN (56))*  X
CREATE TABLE profesion (
    id INT IDENTITY(1,1) PRIMARY KEY,
    nom VARCHAR(90),
    des TEXT
);
```

Luego creamos la tabla de "persona" en la base de datos persona_db utilizando el siguiente script SQL:

```
SQLQuery1.sql - E...NO\SEBASTIAN (56))*  X
CREATE TABLE persona (
    cc BIGINT PRIMARY KEY,
    nombre VARCHAR(45),
    apellido VARCHAR(45),
    genero CHAR(1),
    edad INT
);
```

Luego creamos la tabla de "estudios" en la base de datos persona_db utilizando el siguiente script SQL:

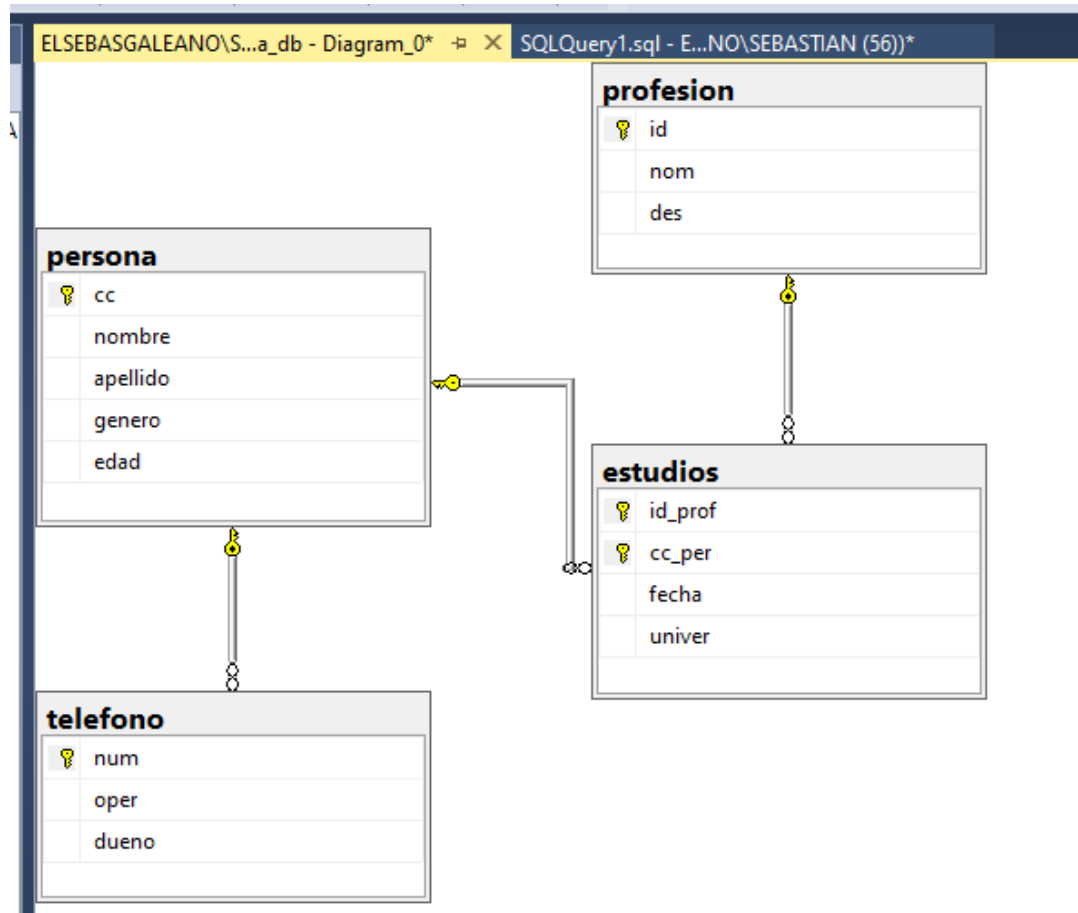
```
SQLQuery1.sql - E...NO\SEBASTIAN (56))*  X
CREATE TABLE estudios (
    id_prof INT,
    cc_per BIGINT,
    fecha DATE,
    univer VARCHAR(50),
    PRIMARY KEY (id_prof, cc_per),
    FOREIGN KEY (id_prof) REFERENCES profesion(id),
    FOREIGN KEY (cc_per) REFERENCES persona(cc)
);
```



Luego creamos la tabla de "telefono" en la base de datos persona_db utilizando el siguiente script SQL

```
SQLQuery1.sql - E...NO\SEBASTIAN (56))* X
CREATE TABLE telefono (
    num VARCHAR(15) PRIMARY KEY,
    oper VARCHAR(45),
    dueno BIGINT,
    FOREIGN KEY (dueno) REFERENCES persona(cc)
);
```

Por último, el diagrama resultante fue el siguiente:

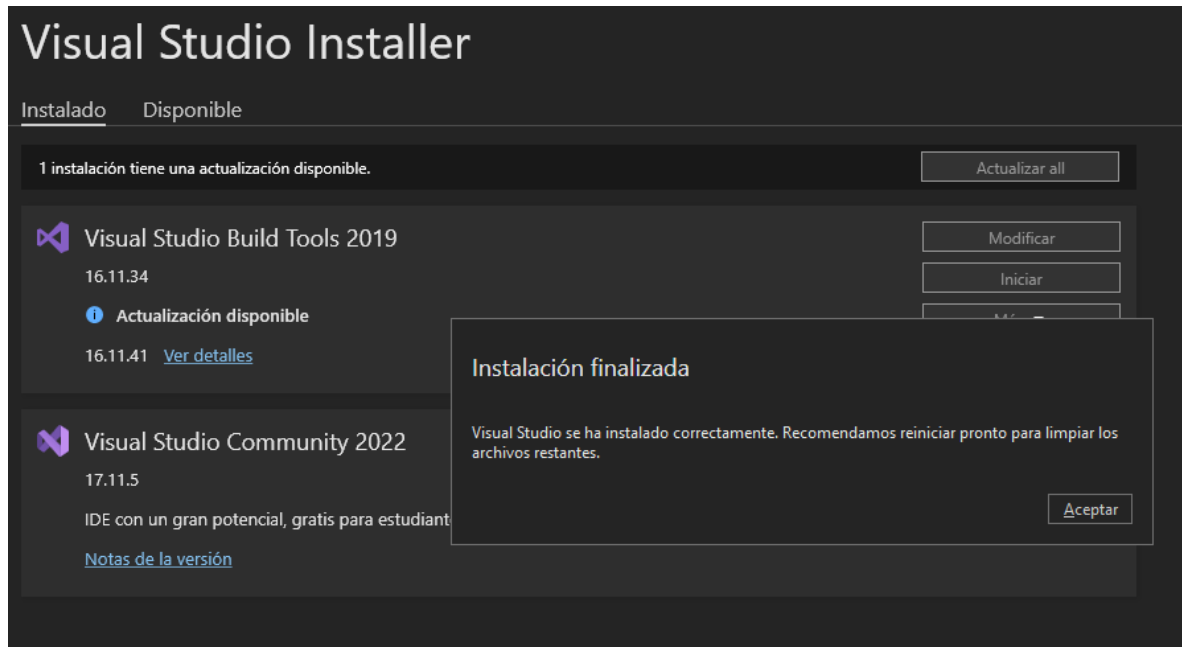


6. Instalar Visual Studio Community 2022

Procedemos a descargar Visual Studio Community 2022 desde el siguiente enlace: [Visual Studio Community 2022](#). Esta herramienta es esencial para el desarrollo de aplicaciones en .NET, ya que proporciona un entorno de desarrollo integrado (IDE) que facilita la creación, depuración y gestión de proyectos.

Se realizó la instalación de los siguientes complementos:

- Desarrollo ASP.NET y web
- Almacenamiento y procesamiento de datos
- Plantillas de proyecto y elementos de .Net Framework
- Características avanzadas de ASP.NET



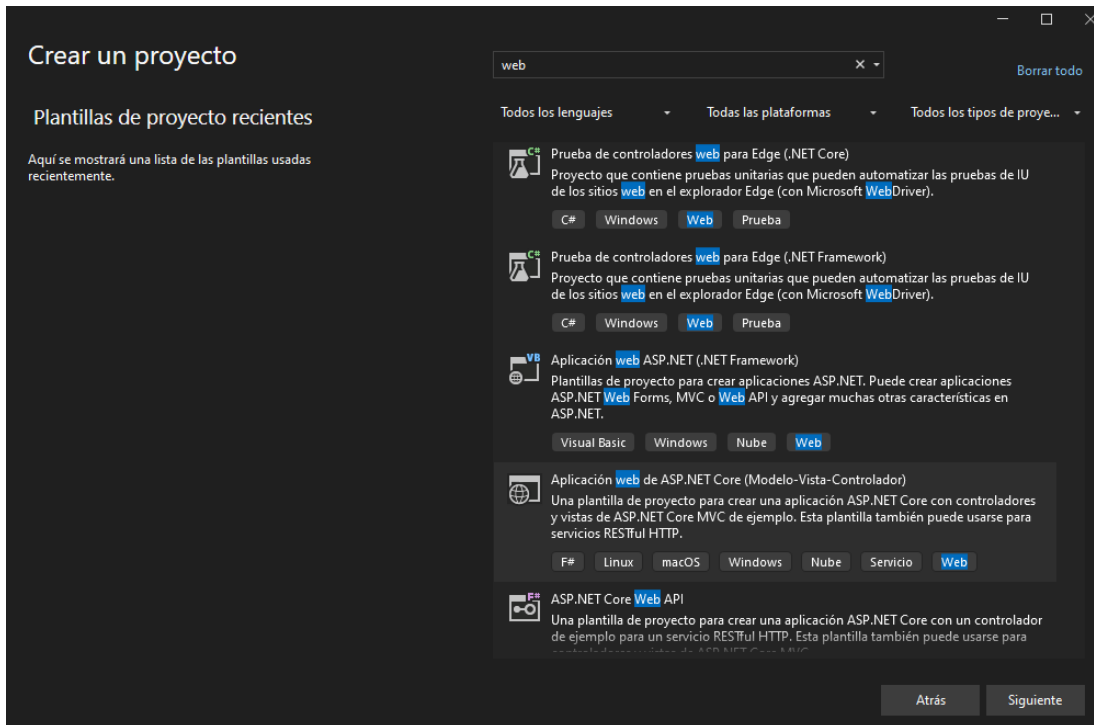
7. Clonar el repositorio local

Procedemos a clonar el repositorio creado en el paso 1:

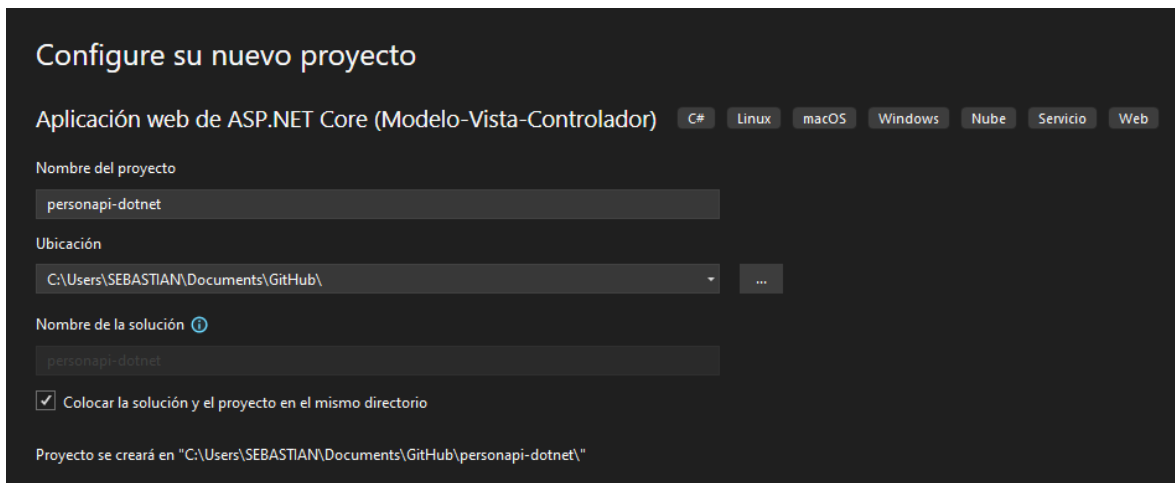
```
SEBASTIAN@elsebasgaleano MINGW64 ~/Documents/GitHub
$ git clone https://github.com/el-sebas-galeano/personapi-dotnet.git
Cloning into 'personapi-dotnet'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

8. Visual Studio Community 2022

Creación del proyecto en Visual Studio Community 2022:



Configuramos el nombre de la aplicación:



Configuramos la versión de .NET, la autenticación y la opción de ejecución en contenedor para crear un dockerfile:



Información adicional

Aplicación web de ASP.NET Core (Modelo-Vista-Controlador) C# Linux macOS Windows Nube Servicio Web

Framework ?
.NET 8.0 (Compatibilidad a largo plazo)

Authentication de campo ?
Ninguno

☐ Configurar para HTTPS ?

☒ Habilitar compatibilidad con el contenedor ?

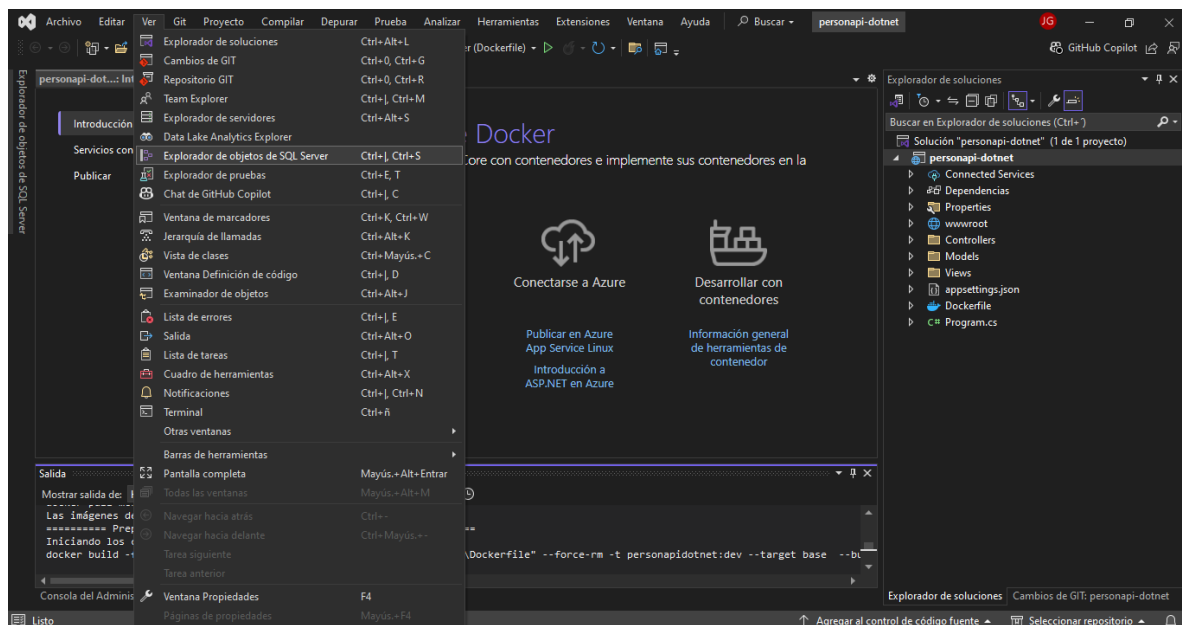
SO del contenedor ?
Linux

Tipo de compilación de contenedor ?
Dockerfile

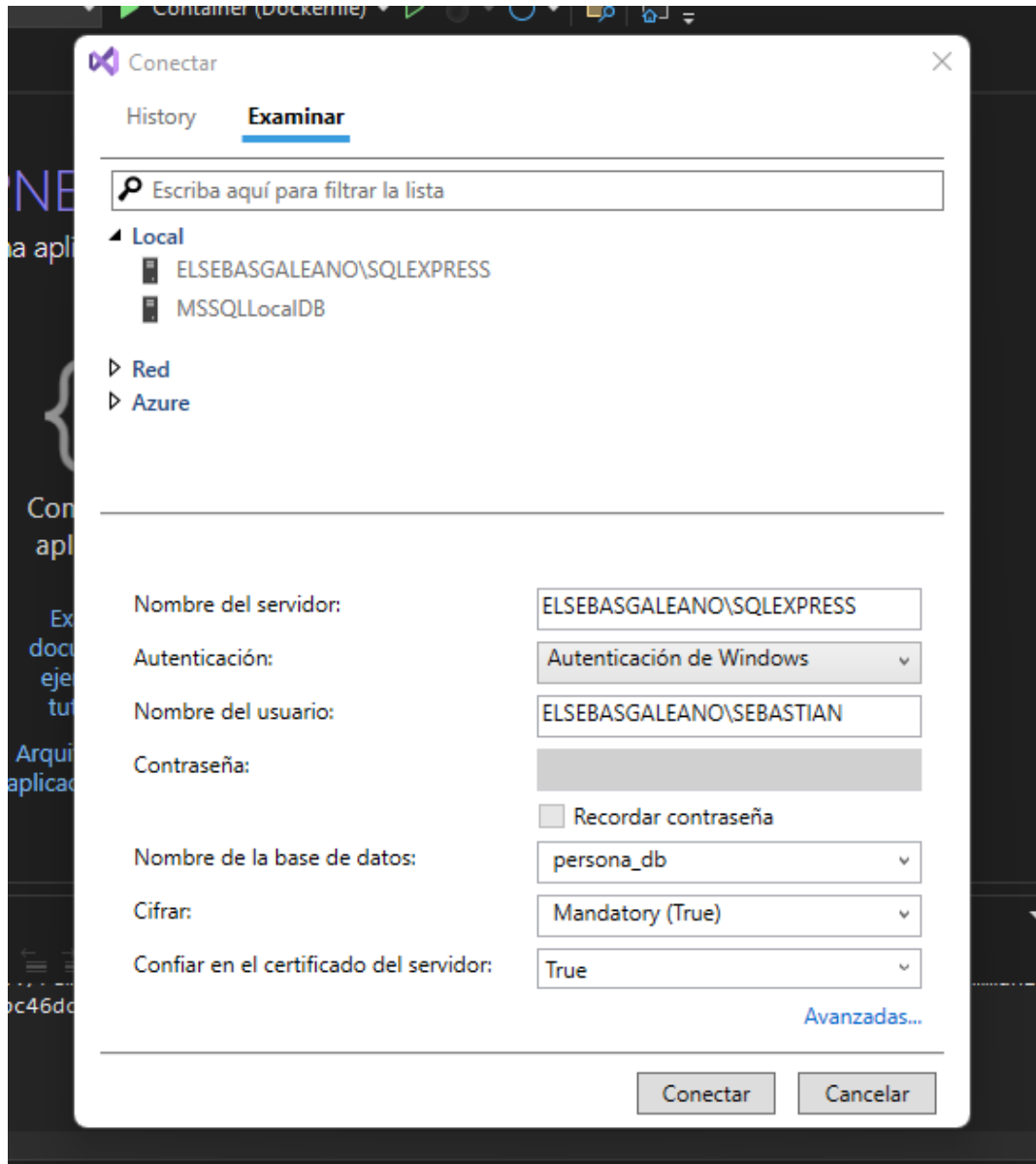
☐ No usar instrucciones de nivel superior ?

☐ Inscribirse en la orquestación de .NET Aspire ?

Una vez creada la aplicación, en la opción Ver activamos el explorador de objetos de SQL Server:



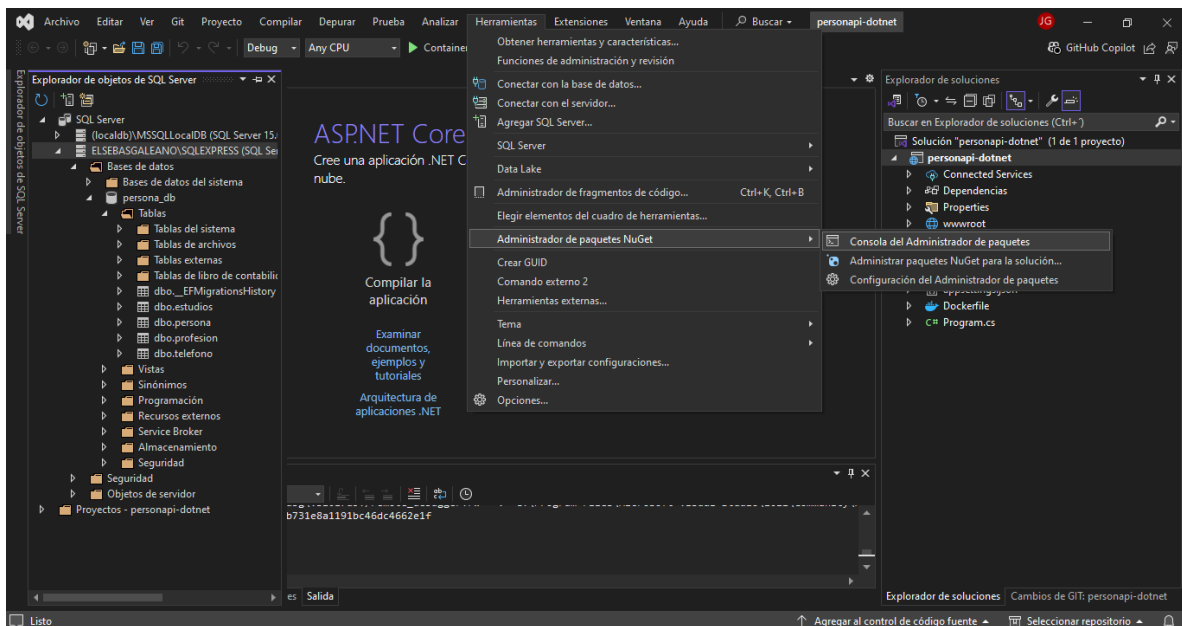
Creamos una conexión a la base de datos creada anteriormente:



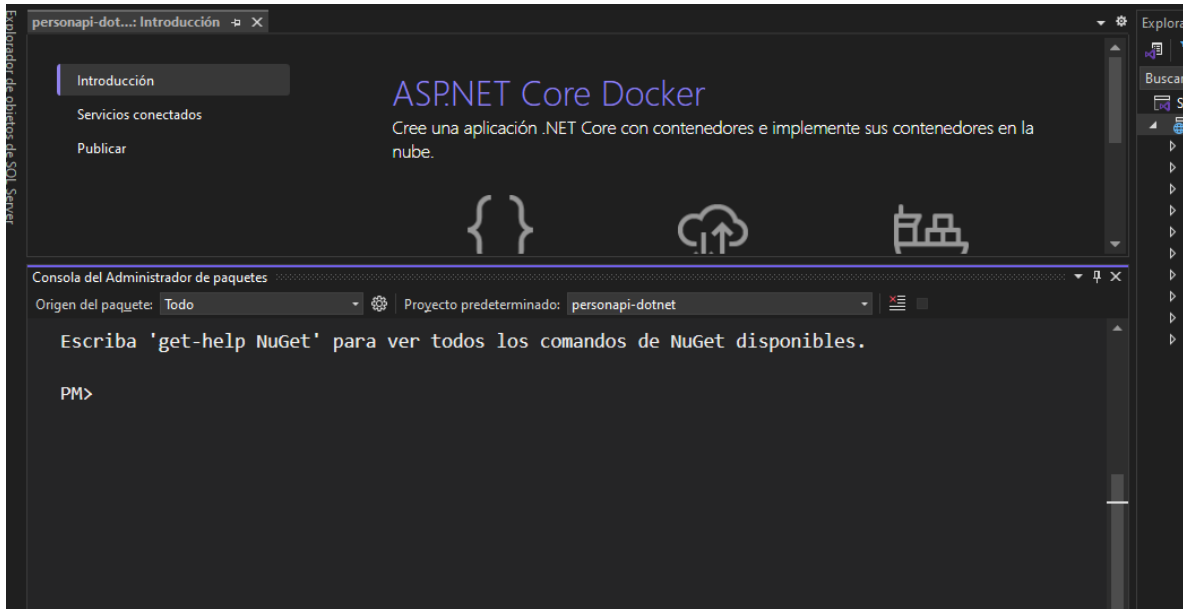
De esta manera, se debería observar la conexión con nuestra base de datos:



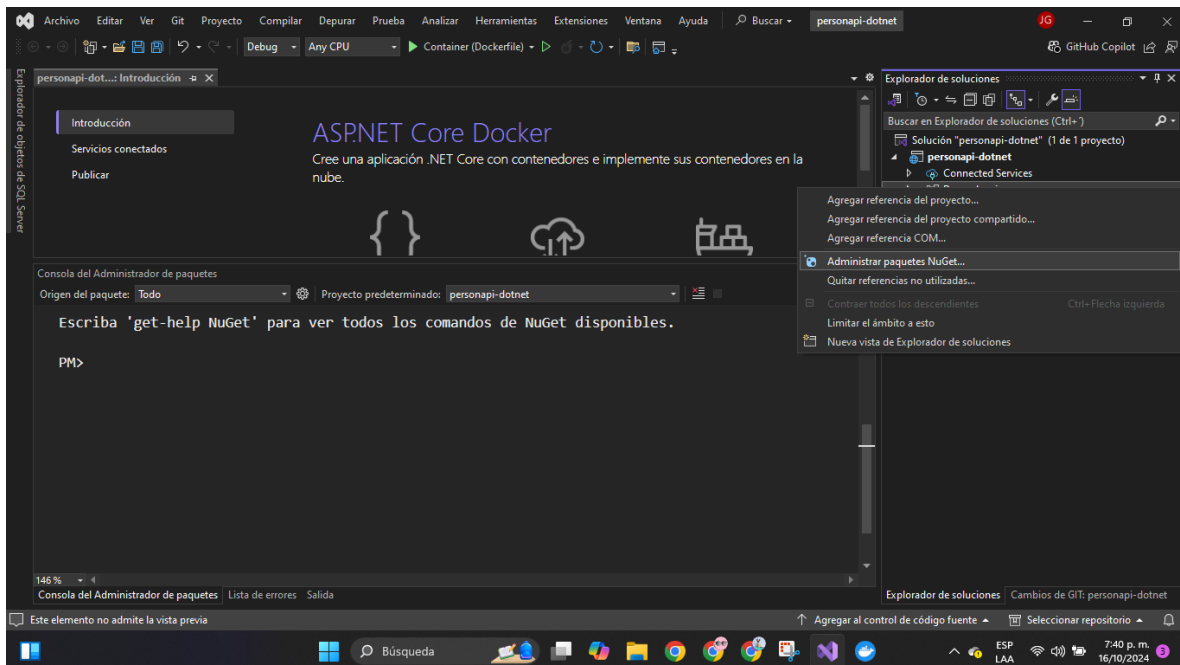
Procedemos a abrir la consola de administración de paquetes NuGet:



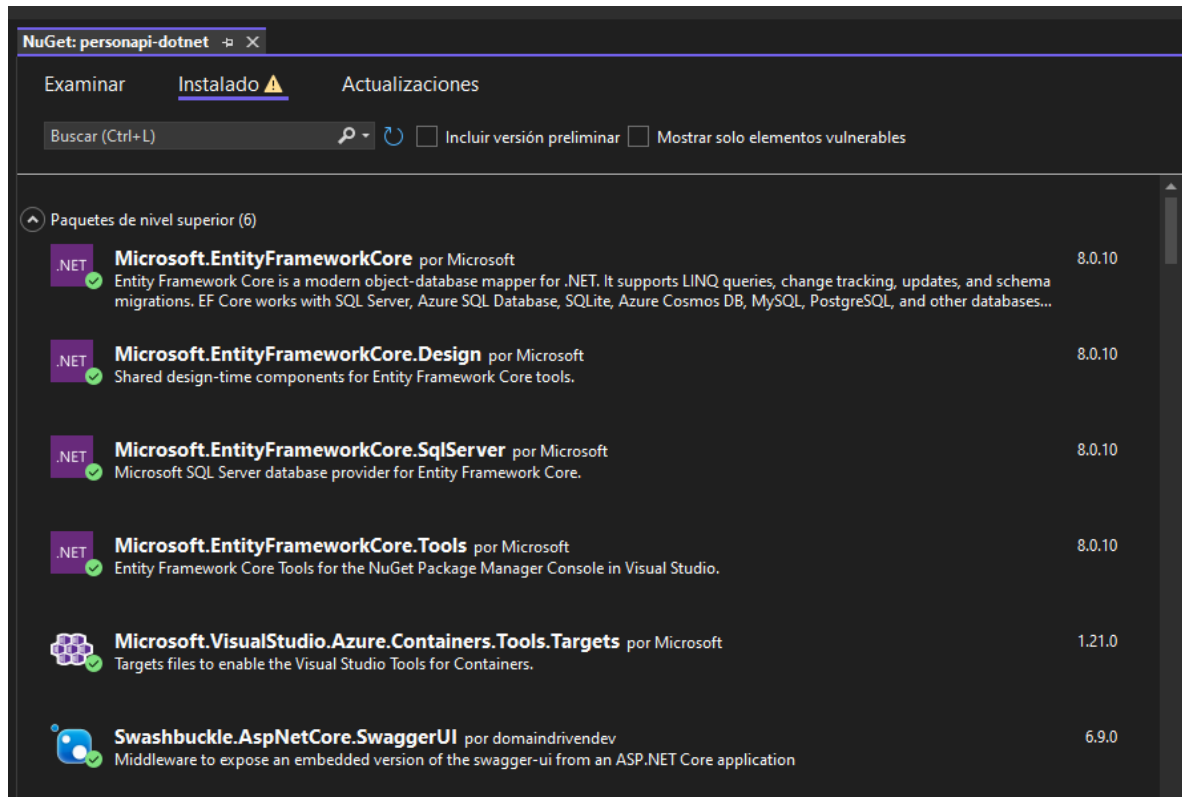
Se abre la consola de administración:



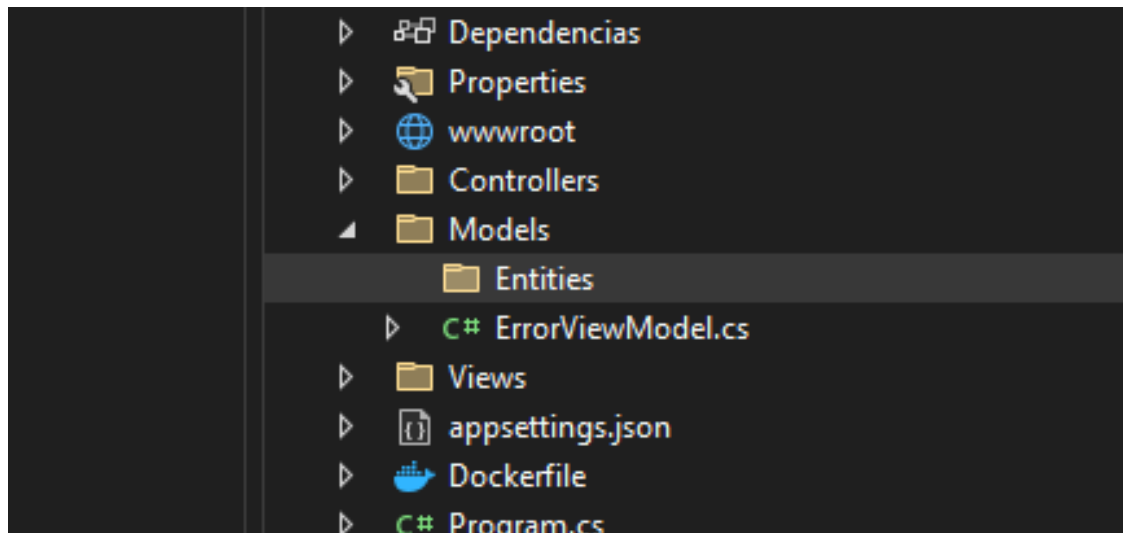
En Dependencias, abrimos la opción de administrar paquetes NuGet:



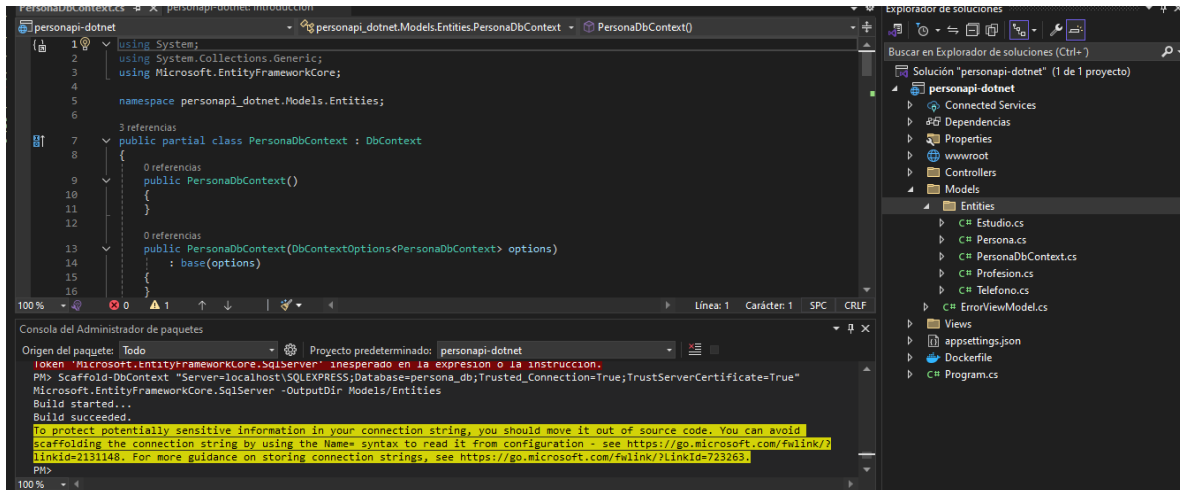
Instalamos los siguientes paquetes:



Creamos una carpeta llamada Entities en Models:



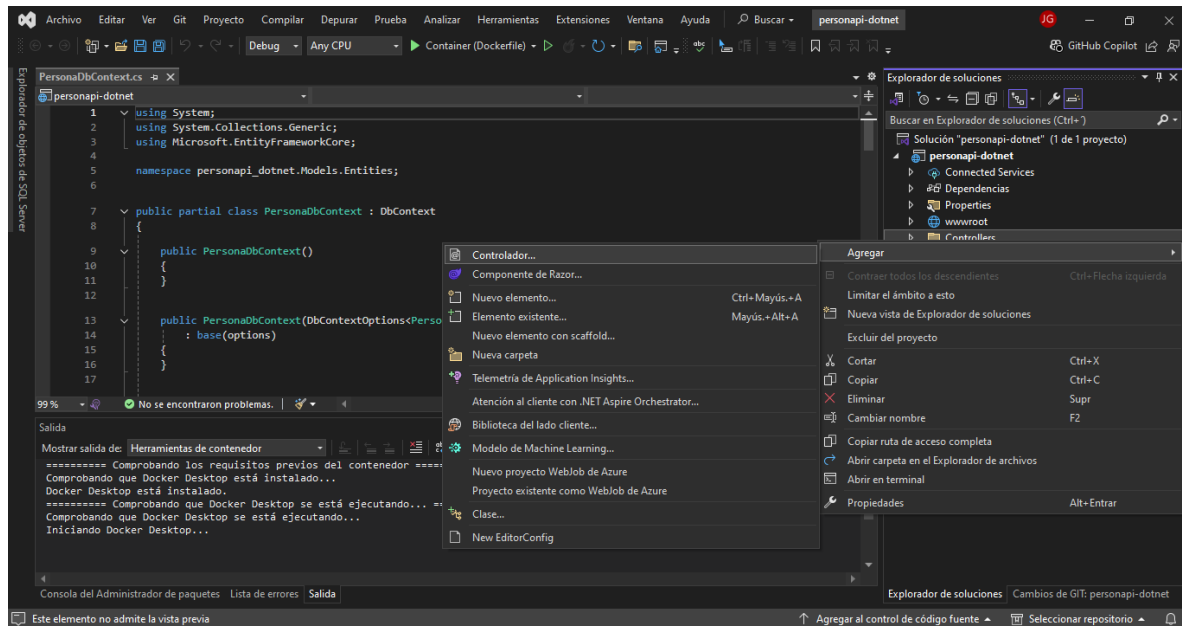
Corremos el siguiente comando para obtener las entidades de la base de datos:
"Server=localhost\SQLEXPRESS;Database=persona_db;Trusted_Connection=True;TrustServerCertificate=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models/Entities



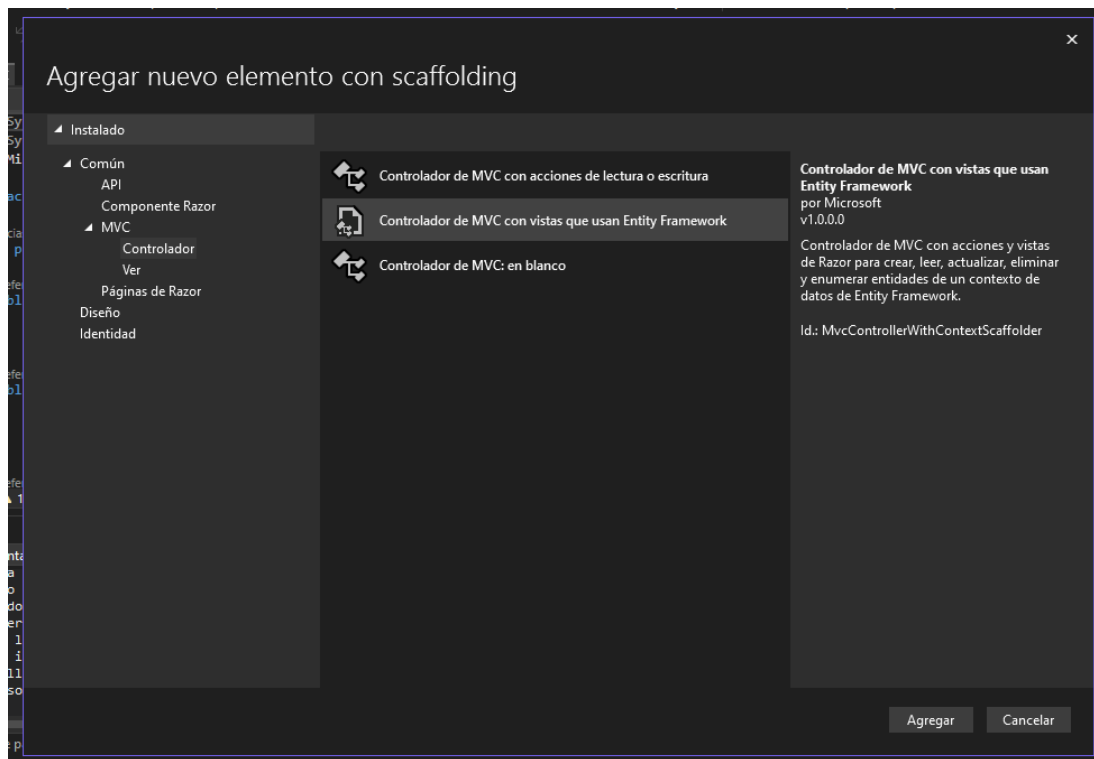
Agregamos la cadena de conexión a appsettings.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection":
    "Server=ms_sql,1433;Database=persona_db;User
    ID=sa;Password=TuPassword123!;TrustServerCert
    ificate=true"
  },
  "AllowedHosts": "*"
}
```

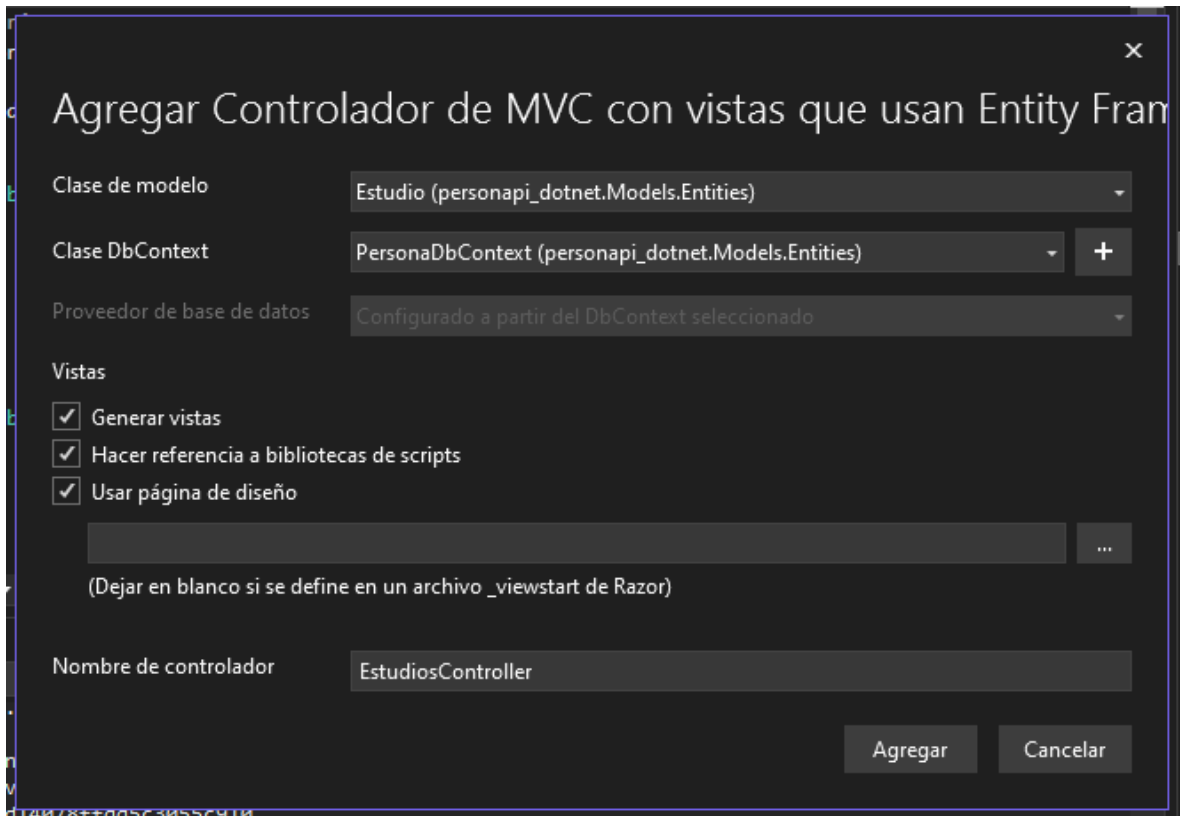
Procedemos a crear los controladores mediante el entity framework:



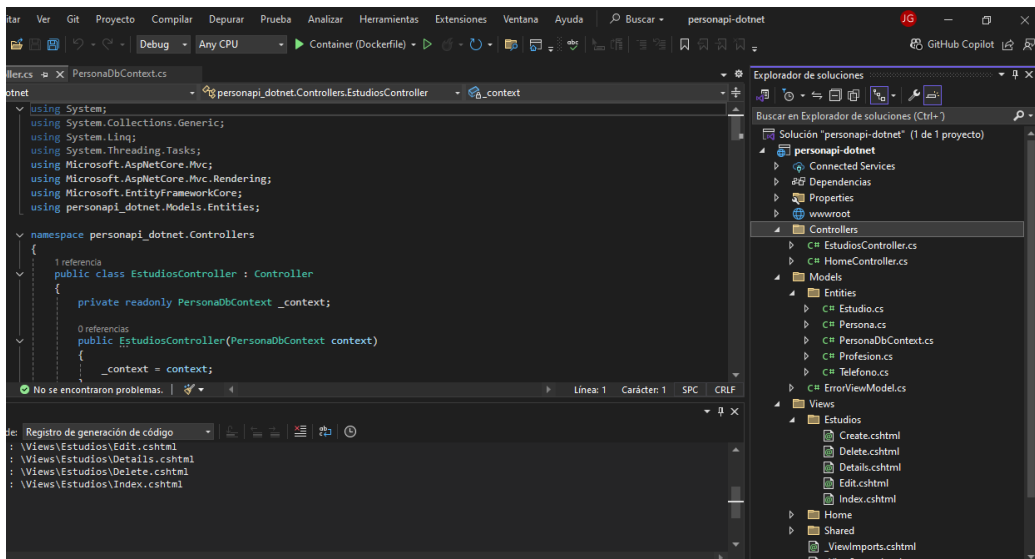
Elegimos la plantilla de Controlador para vistas:



Seleccionamos la entidad y el contexto de la base de datos para crear el controlador:



De esta manera, se crea el controlador y las views asociadas a la entidad para crear, actualizar y eliminar entidades.



Repetimos el proceso con las demás entidades para crear su controlador y vistas. A continuación, se observa uno de los archivos de vista creado automáticamente:



```

@model personapi_dotnet.Models.Entities.Estudio

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

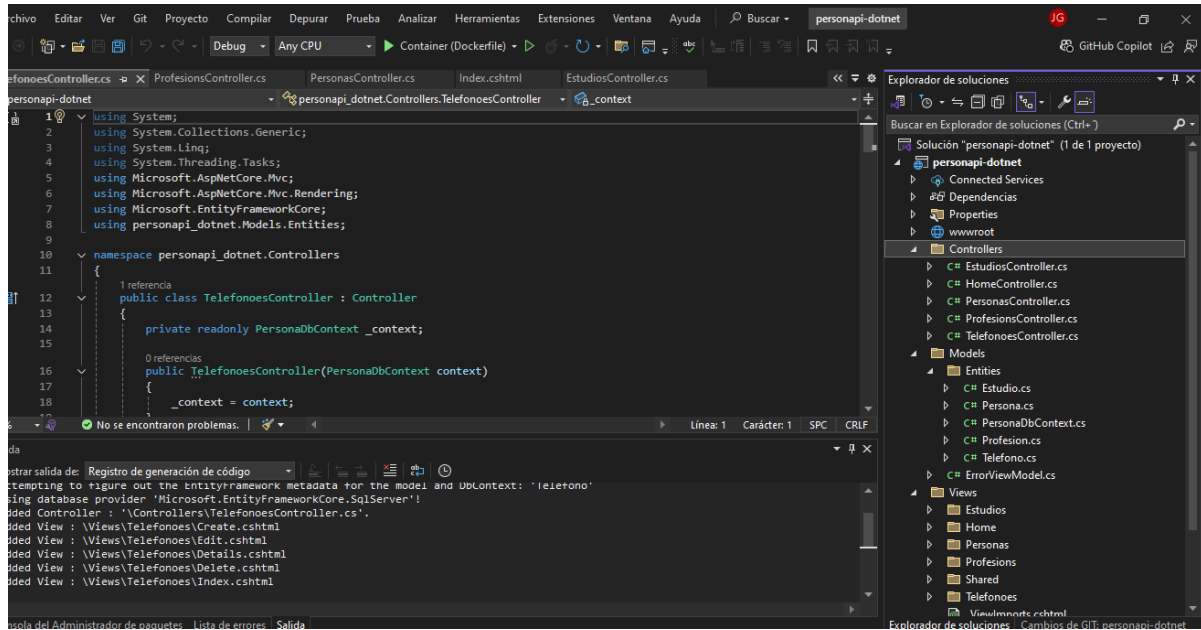
<h4>Estudio</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="IdProf" class="control-label"></label>
                <select asp-for="IdProf" class="form-control" asp-items="ViewBag.IdProf"></select>
            </div>
            <div class="form-group">
                <label asp-for="CcPer" class="control-label"></label>
                <select asp-for="CcPer" class="form-control" asp-items="ViewBag.CcPer"></select>
            </div>
            <div class="form-group">
                <label asp-for="Fecha" class="control-label"></label>
                <input asp-for="Fecha" class="form-control" />
                <span asp-validation-for="Fecha" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Univer" class="control-label"></label>
                <input asp-for="Univer" class="form-control" />
                <span asp-validation-for="Univer" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

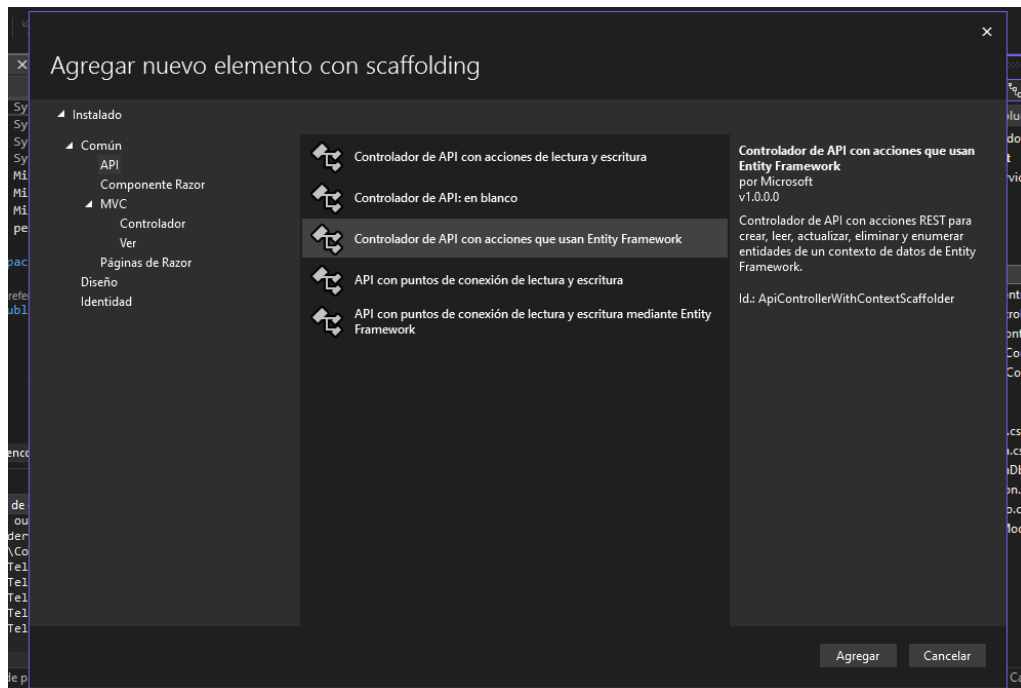
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

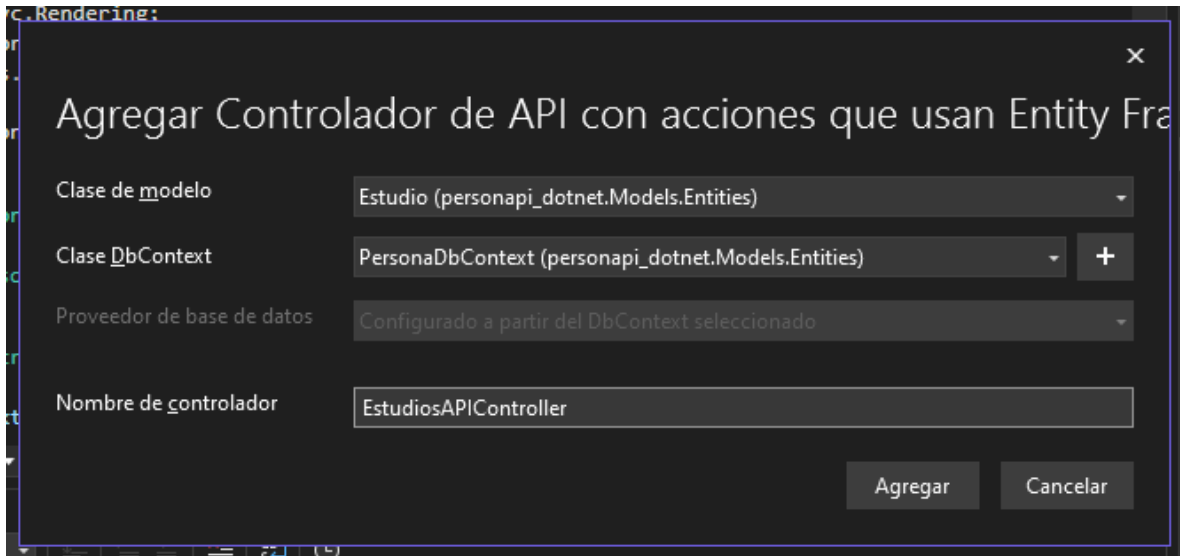
En la imagen siguiente se observa la creación de todos los controladores y las carpetas de las vistas creadas:



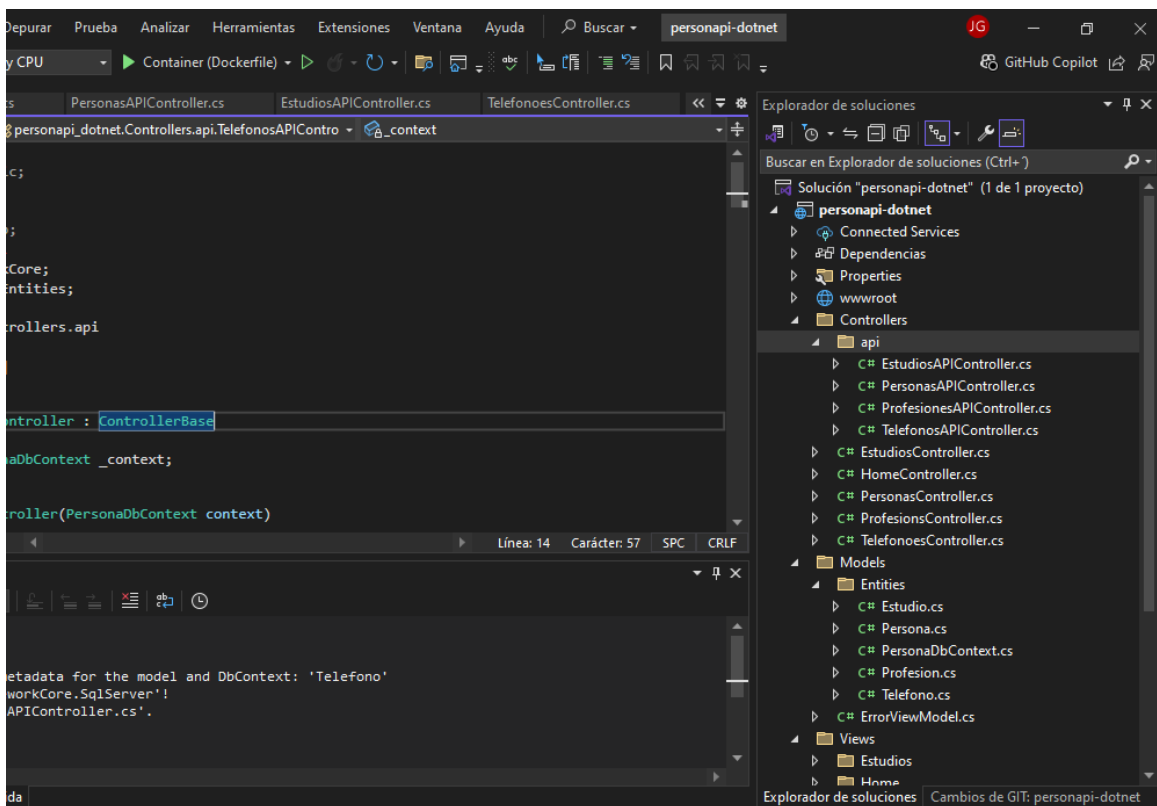
Ahora crearemos los controladores para nuestra API Restful. Creamos una carpeta llamada “api” en los controladores, donde los crearemos mediante entity framework:



Escogemos la entidad y el contexto de la base de datos:



De esta manera, quedan creados nuestros controladores para api:



Agregamos los controladores en Views, en la carpeta Shared y el archivo Layout:



```

<ul class="navbar-nav flex-grow-1">
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-
action="Privacy">Privacy</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Persona" asp-
action="Index">Persona</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Profesion" asp-
action="Index">Profesion</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Estudio" asp-
action="Index">Estudio</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Telefono" asp-
action="Index">Telefono</a>
  </li>
</ul>

```

En la consola de administración de paquetes añadimos los siguientes paquetes:

```

Se ha detenido la canalización.
PM> dotnet add package Swashbuckle.AspNetCore
Determinando los proyectos que se van a restaurar...
Writing C:\Users\SEBASTIAN\AppData\Local\Temp\tmpgqkhyz.tmp
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza
predeterminado seleccionado por .NET para la firma de código.
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza
predeterminado seleccionado por .NET para la marca de tiempo.

```

```

PM> dotnet add package Swashbuckle.AspNetCore.SwaggerUI
Determinando los proyectos que se van a restaurar...
Writing C:\Users\SEBASTIAN\AppData\Local\Temp\tmpinkq2e.tmp
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza
predeterminado seleccionado por .NET para la firma de código.
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza
predeterminado seleccionado por .NET para la marca de tiempo.

```

Por último, configuramos nuestro Program.cs para añadir la configuración de swagger y la configuración para la base de datos:



```
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;
using personapi_dotnet.Models.Entities;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "Persona API", Version = "v1" });
});

builder.Services.AddDbContext<PersonaDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseExceptionHandler("/Home/Error");
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "API v1");
    });
}
else
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "API v1");
    });
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

Corremos el comando dotnet build para construir nuestro programa.



```
PM> dotnet build
MSBUILD : error MSB1011: Especifique el archivo de proyecto o de soluci3n que se va a utilizar ya que esta carpeta contiene m3s de un archivo de proyecto o de soluci3n.

Hay actualizaciones de carga de trabajo disponibles. Ejecute "dotnet workload list" para obtener m3s informaci3n.
PM> dotnet build
Determinando los proyectos que se van a restaurar...
Todos los proyectos est3n actualizados para la restauraci3n.
C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\Models\Entities\PersonaDbContext.cs(27,10): warning CS1030: #advertencia: 'To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the 'Name' syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see https://go.microsoft.com/fwlink/?linkid=723263.' [C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\personapi-dotnet.csproj]
C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\Views\Telefonos\Index.cshtml(31,47): warning CS8602: Desreferencia de una referencia posiblemente NULL. [C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\personapi-dotnet.csproj]
C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\Views\Telefonos\Delete.cshtml(24,39): warning CS8602: Desreferencia de una referencia posiblemente NULL. [C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\personapi-dotnet.csproj]
C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\Views\Telefonos\Details.cshtml(23,39): warning CS8602: Desreferencia de una referencia posiblemente NULL. [C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\personapi-dotnet.csproj]
personapi-dotnet -> C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\bin\Debug\net6.0\personapi-dotnet.dll

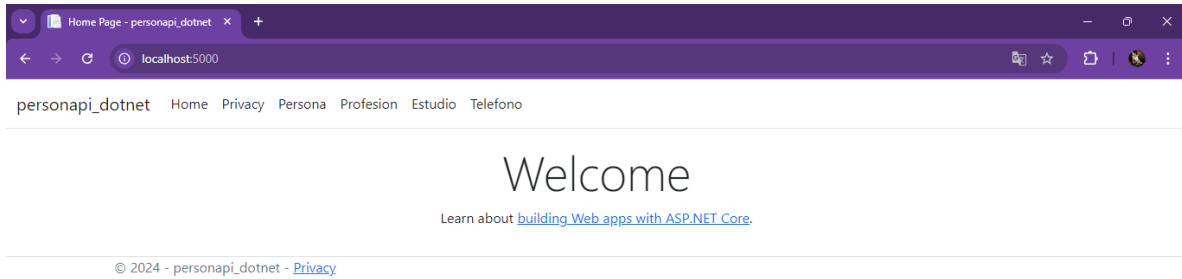
Compilaci3n correcta.

C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\Models\Entities\PersonaDbContext.cs(27,10): warning CS1030: #advertencia: 'To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the 'Name' syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see https://go.microsoft.com/fwlink/?linkid=723263.' [C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\personapi-dotnet.csproj]
C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\Views\Telefonos\Index.cshtml(31,47): warning CS8602: Desreferencia de una referencia posiblemente NULL. [C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet\personapi-dotnet.csproj]
```

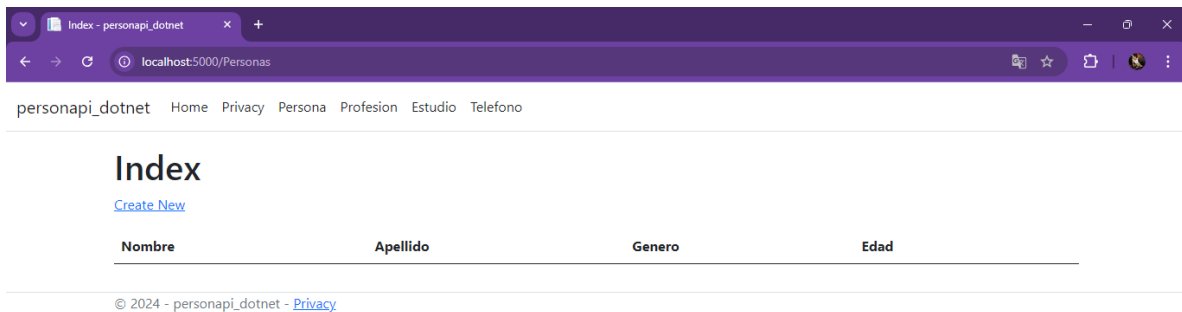
Corremos el comando dotnet build para ejecutarlo:

```
PM> dotnet run
Compilando...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\SEBASTIAN\Documents\GitHub\personapi-dotnet
```

Abrimos en el puerto que nos dice la consola:



Accedemos al controlador de Persona:



Creamos una persona:



Create - personapi_dotnet

localhost:5000/Personas/Create

personapi_dotnet Home Privacy Persona Profesion Estudio Telefono

Create

Persona

Cc
123456

Nombre
Sebastian Esteban

Apellido
Reyes Galeano

Genero
F

Edad
33

Create

[Back to List](#)

© 2024 - personapi_dotnet - [Privacy](#)

Confirmamos la creación:

Index - personapi_dotnet

localhost:5000/Personas

personapi_dotnet Home Privacy Persona Profesion Estudio Telefono

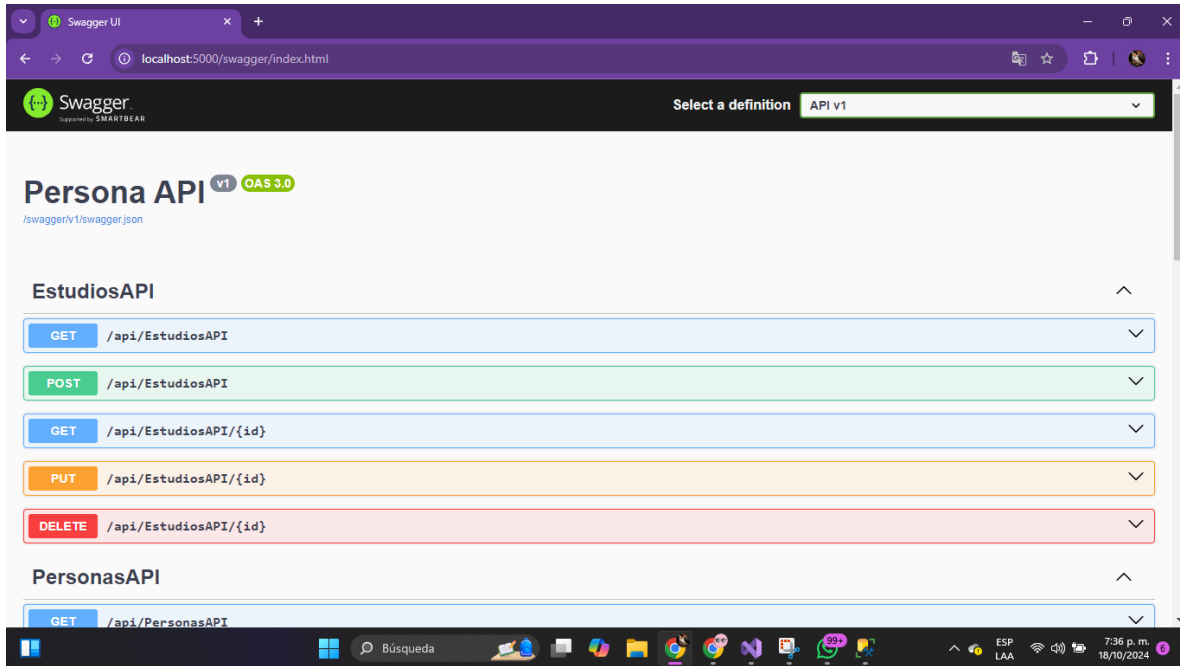
Index

[Create New](#)

Nombre	Apellido	Genero	Edad	
Sebastian Esteban	Reyes Galeano	F	33	Edit Details Delete

© 2024 - personapi_dotnet - [Privacy](#)

Ahora accedemos a swagger, añadiendo /swagger a la ruta inicial, en este caso localhost:5000:



9. Creación Docker Compose

El archivo docker-compose.yml define los servicios y la infraestructura para el laboratorio que incluye una base de datos SQL Server (ms_sql) y una aplicación .NET (personapi-dotnet). Esta es la estructura básica que permite levantar estos servicios y configurar la comunicación entre ellos:



```
1 version: '3.9'
2
3 services:
4   data:
5     build: ./db-config
6     container_name: ms_sql
7     image: ms_sql
8     restart: always
9     environment:
10      ...
11     healthcheck:
12      ...
13     ports:
14       - "1433:1433"
15     networks:
16       - lab-net-1
17
18   personapi-dotnet:
19     build: ./personapi-dotnet
20     container_name: personapi-dotnet
21     image: personapidotnet
22     ports:
23       - "32770:8080"
24     depends_on:
25       data:
26         condition: service_healthy
27     environment:
28       ...
29     networks:
30       - lab-net-1
31
32 networks:
33   lab-net-1:
34     driver: bridge
```

Descripción:

- **services:** define los dos contenedores que se van a ejecutar: data (para SQL Server) y personapi-dotnet (para la aplicación .NET).
- **build: ./carpeta:** Indica que Docker debe construir la imagen para el servicio a partir del Dockerfile ubicado en la carpeta.
- **container_name: nombre-contenedor:** Define el nombre del contenedor para la aplicación .NET.
- **image: nombre-imagen:** Define el nombre de la imagen que se creará para este contenedor.
- **ports: "HHHH:CCCC":** Expone el puerto CCCC del contenedor (donde se ejecuta la aplicación .NET) en el puerto HHHH de la máquina host.



- **depends_on:** Esta línea asegura que el contenedor (en este caso personapi-dotnet) no se inicie hasta que el servicio de SQL Server (data) esté saludable y completamente inicializado. La condición `service_healthy` especifica que el contenedor .NET debe esperar a que SQL Server pase el healthcheck.
- **environment:** Aquí se especifican variables de entorno que usaran los servicios.
- **healthcheck:** Configura un chequeo de salud para monitorear si el servicio (en este caso el de SQL Server) está activo y funcionando correctamente.
- **networks:** nombre-red: Define una red personalizada (en este caso llamada lab-net-1) que conecta ambos servicios (data y personapi-dotnet). Esta red es de tipo bridge, lo que significa que Docker creará una red aislada para que los contenedores puedan comunicarse entre sí, pero no con otros servicios fuera de esa red.

Luego de la configuración de este archivo *docker-compose.yml*, se utiliza el comando:



```
docker-compose up --build
```

Esto permite construir las imágenes y correr ambos contenedores en la red indicada.

10. Hacer push al repositorio

Paso 1: Clonar el repositorio (si no se ha hecho)

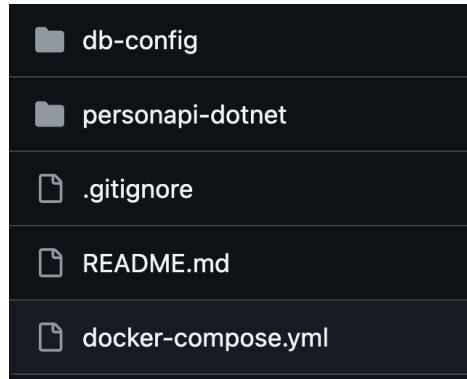
Si aún no se ha clonado el repositorio, hay que usar el siguiente comando para clonar el repositorio remoto a tu máquina local:



```
git clone https://github.com/el-sebas-galeano/personapi-dotnet.git
```

Paso 2: Agregar los cambios al repositorio local

Antes de hacer push, debes agregar los archivos que desea enviar del repositorio local. En este caso, debe contener los siguientes archivos y directorios:



Luego, hay que agregar los archivos modificados al área de preparación (staging area) con el comando:

```
git add .
```

Paso 3: Confirmar los cambios:

Después de agregar los archivos al área de preparación, se debe hacer commit a los cambios junto con un mensaje:

```
git commit -m "Descripción de los cambios realizados"
```

Paso 4: Hacer push al repositorio remoto

Una vez que los cambios han sido confirmados, deben ser enviados al repositorio remoto. En la rama principal (main o master), el comando sería:

```
git push origin main
```



11. Crear tag y hacer un release

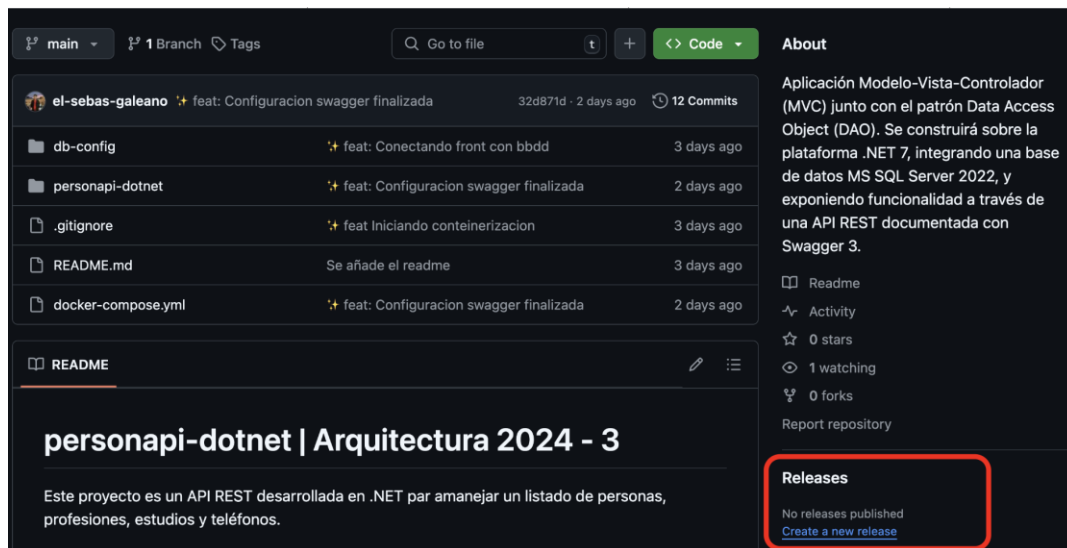
Paso 1: Crear un tag en Git

Se debe crear un tag en el repositorio local y enviado al repositorio remoto. Esto se hace de la siguiente manera:

```
# Crear un tag anotado  
git tag -a v1.0 -m "Versión 1.0: Primera  
versión de lanzamiento"  
  
# Enviar el tag al repositorio remoto  
git push origin v1.0
```

Paso 2: Crear un nuevo release

En la página del repositorio de GitHub, dirijase a la sección de Releases y haga click en el botón "Create a new release".



Paso 3: Llenar los campos del formulario:



The screenshot shows the GitHub 'Create a new release' form. At the top, there's a dropdown for the tag version, currently set to 'v1.0'. Next to it are 'Target: main' and 'Previous tag: auto' dropdowns, and a 'Generate release notes' button. Below these is a text input field for the release title, which contains 'Entrega del laboratorio'. Underneath the title is a 'Write' tab with a rich text editor for describing the release. At the bottom, there's a checkbox for 'Set as a pre-release' and two buttons: 'Publish release' and 'Save draft'.

- Tag version: Seleccionar el tag que se creó previamente (v1.0 en este caso). Si el tag no existe aún, puedes crearlo directamente en este formulario.
- Release title: Proporcionar un título para el release, en este caso: "Entrega del laboratorio".
- Descripción: Opcional
- Una vez rellenados todos los campos, haga click en el botón "Publish release".



Conclusiones y lecciones aprendidas

Conclusiones:

- Integración de tecnologías: La combinación de .NET 7, SQL Server 2022 y Swagger 3 permite implementar aplicaciones monolíticas robustas, con un enfoque modular y buenas prácticas en el manejo de datos y servicios web. El uso del patrón MVC facilita la separación de responsabilidades y mejora la mantenibilidad del código.
- Eficiencia en el desarrollo: Utilizar herramientas integradas como Visual Studio y Entity Framework simplifica tareas complejas, como la gestión de la base de datos y la generación de código. Esto agiliza el proceso de desarrollo y reduce los errores comunes en la integración con la base de datos.
- Documentación y pruebas automáticas: La integración de Swagger facilita la documentación y prueba de la API, lo que resulta beneficioso para la colaboración y la verificación del correcto funcionamiento de los servicios RESTful implementados.
- Limitaciones y desafíos: Durante la implementación, se presentaron desafíos como la correcta configuración de la cadena de conexión y el uso de paquetes NuGet, lo que resalta la importancia de entender las configuraciones de entorno y la gestión de dependencias en .NET.

Lecciones aprendidas:



- Importancia de un entorno de desarrollo configurado adecuadamente: Tener todas las herramientas y configuraciones listas antes de comenzar permite un flujo de trabajo más fluido y evita problemas de compatibilidad.
- Dominio de Entity Framework y la generación de código desde bases de datos existentes: Aprender a usar Scaffold-DbContext para generar las entidades a partir de la base de datos mejora la productividad y asegura que el modelo de datos esté alineado con la base de datos.
- Creación y configuración de repositorio con Docker Compose. Muy chévere la herramienta, facilita mucho la creación de muchos contenedores y la interconexión entre ellos.

Referencias

1. Microsoft. (2024, abril 2). Introducción a .NET. Microsoft Learn. <https://learn.microsoft.com/es-es/dotnet/fundamentals/>
2. Microsoft. (2024). Introduction to SQL Server 2022. Microsoft Learn. <https://learn.microsoft.com/en-us/training/modules/introduction-to-sql-server-2022/>
3. Swagger. (2023). OpenAPI Specification, Version 3.1.0. Swagger.io. <https://swagger.io/specification/>
4. Docker, Inc. (n.d.). Docker Documentation. Docker. <https://docs.docker.com/>