Pontificia Universidad Javeriana

Arquitectura de Software

Laboratorio 1

GRUPO 1

Andres Felipe Velez Echeverry
Victor Sanchez Cardenas
Victor Peñaranda Florez

03 de Noviembre de 2025

Facultad de Ingeniería Sistemas

Bogotá D.C. 2025

Marco Teórico

Introducción

En el desarrollo de software, el diseño de la arquitectura es una fase fundamental que define la estructura general del sistema, las tecnologías empleadas y los patrones de diseño que se aplicarán. Este laboratorio se centra en la construcción de una aplicación monolítica utilizando los patrones Modelo-Vista-Controlador (MVC) y Data Access Object (DAO), aprovechando el entorno de desarrollo ASP.NET Core, el ORM Entity Framework Core y la base de datos relacional SQL Server 2022. Se busca aplicar conceptos fundamentales de arquitectura de software de manera práctica, integrando herramientas modernas que promuevan la calidad, mantenibilidad y organización del código.

Arquitectura de Software

La arquitectura de software es el conjunto de decisiones estructurales que determinan la organización interna de un sistema. Incluye la definición de componentes, sus relaciones y restricciones, así como los estilos y patrones que guían su implementación.

Una buena arquitectura permite:

- Satisfacer atributos de calidad (mantenibilidad, escalabilidad, seguridad).
- Facilitar la evolución del sistema a lo largo del tiempo.
- Promover la reutilización y separación de responsabilidades.

En este laboratorio se implementa una arquitectura monolítica, que será descrita a continuación.

Arquitectura Monolítica

La arquitectura monolítica es un estilo en el que todos los componentes del sistema (interfaz de usuario, lógica de negocio y acceso a datos) están integrados en una única aplicación desplegable. Es una aproximación común en proyectos pequeños o medianos por su simplicidad inicial.

Características:

- Todos los módulos se ejecutan en el mismo proceso.
- Compartición directa de memoria entre capas.
- Una sola base de código y despliegue unificado.

Ventajas:

- Menor complejidad inicial en desarrollo y despliegue.
- Facilidad para pruebas integradas y depuración.
- Mejor rendimiento por evitar comunicación entre servicios.

Desventajas:

- Dificultad para escalar componentes por separado.
- Riesgo de acoplamiento excesivo entre capas.

- Complejidad creciente a medida que el sistema crece.

Patrón Modelo-Vista-Controlador (MVC)

El patrón MVC divide la aplicación en tres capas principales:

- Modelo: Representa los datos y la lógica de negocio del sistema. Aquí se ubican las entidades y el acceso a la base de datos.
- Vista: Encargada de la presentación de los datos al usuario. En aplicaciones web puede incluir páginas Razor o respuestas JSON para APIs.
- Controlador: Intermediario que procesa las solicitudes del usuario, ejecuta la lógica necesaria y selecciona la vista apropiada.

Beneficios del patrón MVC:

- Favorece la separación de responsabilidades.
- Permite el desarrollo paralelo del frontend y backend.
- Facilita la prueba y mantenimiento del sistema.

Patrón DAO (Data Access Object)

El patrón DAO es una capa de abstracción que encapsula el acceso a la base de datos. Su propósito es desacoplar la lógica de negocio de los detalles específicos del almacenamiento persistente.

Componentes típicos:

- Interfaz DAO: Define las operaciones básicas (CRUD).
- Clase DAO concreta: Implementa los métodos usando SQL o un ORM.
- Uso en servicios/controladores: Permite a la lógica de negocio interactuar con los datos sin depender del almacenamiento.

Ventajas del patrón DAO:

- Modularidad y bajo acoplamiento.
- Facilita el mantenimiento y pruebas unitarias.
- Posibilidad de cambiar el motor de base de datos sin afectar la lógica de negocio.

Entity Framework Core (EF Core)

Entity Framework Core es un ORM (Object-Relational Mapper) para .NET que permite interactuar con bases de datos relacionales usando clases de C#. Soporta múltiples bases como SQL Server, PostgreSQL y MySQL.

Funcionalidades clave:

- Traducción automática entre clases C# y tablas SQL.
- Soporte para migraciones y scaffolding.
- Consultas LINQ para operaciones CRUD.

Enfoques:

- Database-First: Se parte de una base de datos existente (como en este laboratorio).
- Code-First: Se diseña el modelo en código y se generan las tablas desde allí.
- EF Core permite integrar el patrón DAO fácilmente, ya que permite centralizar todas las operaciones de datos en repositorios.

API REST y ASP.NET Core

Una API RESTful es una interfaz que permite el acceso a recursos a través de métodos HTTP estándar como GET, POST, PUT y DELETE. Su diseño sigue principios como:

- Uso de URLs como identificadores de recursos.
- Comunicación sin estado entre cliente y servidor.
- Intercambio de datos en formatos como JSON o XML.

ASP.NET Core es un framework moderno, multiplataforma y de alto rendimiento para crear aplicaciones web y APIs. Al usar ASP.NET Core en conjunto con MVC y EF Core, se puede implementar una API REST completa de manera eficiente y modular.

Swagger y documentación de APIs

Swagger (OpenAPI) es una herramienta que permite generar documentación interactiva de APIs REST. Se integra automáticamente con ASP.NET Core para mostrar todos los endpoints disponibles, sus parámetros, métodos HTTP y posibles respuestas.

Beneficios:

- Facilita la exploración y prueba de la API desde el navegador.
- Documentación autogenerada y actualizada.
- Mejora la colaboración entre desarrolladores frontend y backend.

SQL Server 2022

SQL Server es un sistema de gestión de bases de datos relacional desarrollado por Microsoft. En su edición Express, es una opción gratuita y potente para el desarrollo académico y profesional.

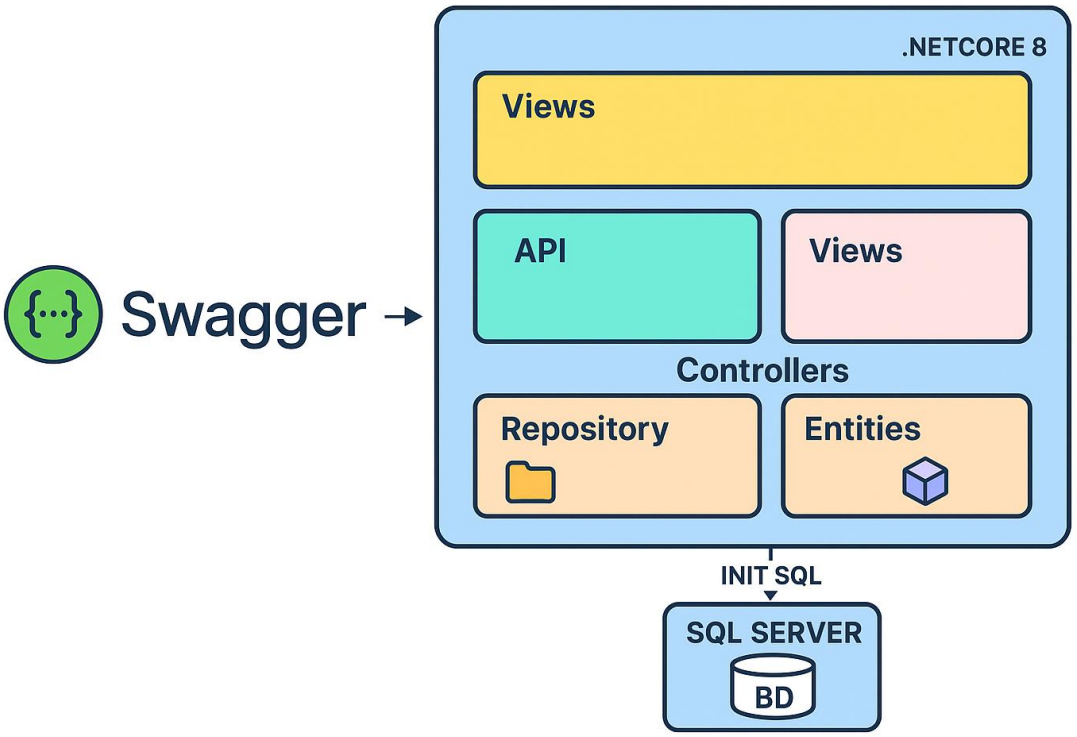
Características clave:

- Lenguaje estándar SQL.
- Compatibilidad con herramientas como SQL Server Management Studio (SSMS).
- Alta integración con .NET y Entity Framework Core.

En este laboratorio, se utiliza SQL Server 2022 como base para realizar el scaffolding y generar automáticamente las entidades del modelo de datos.

Diseño(C4)

Diagrama de alto nivel

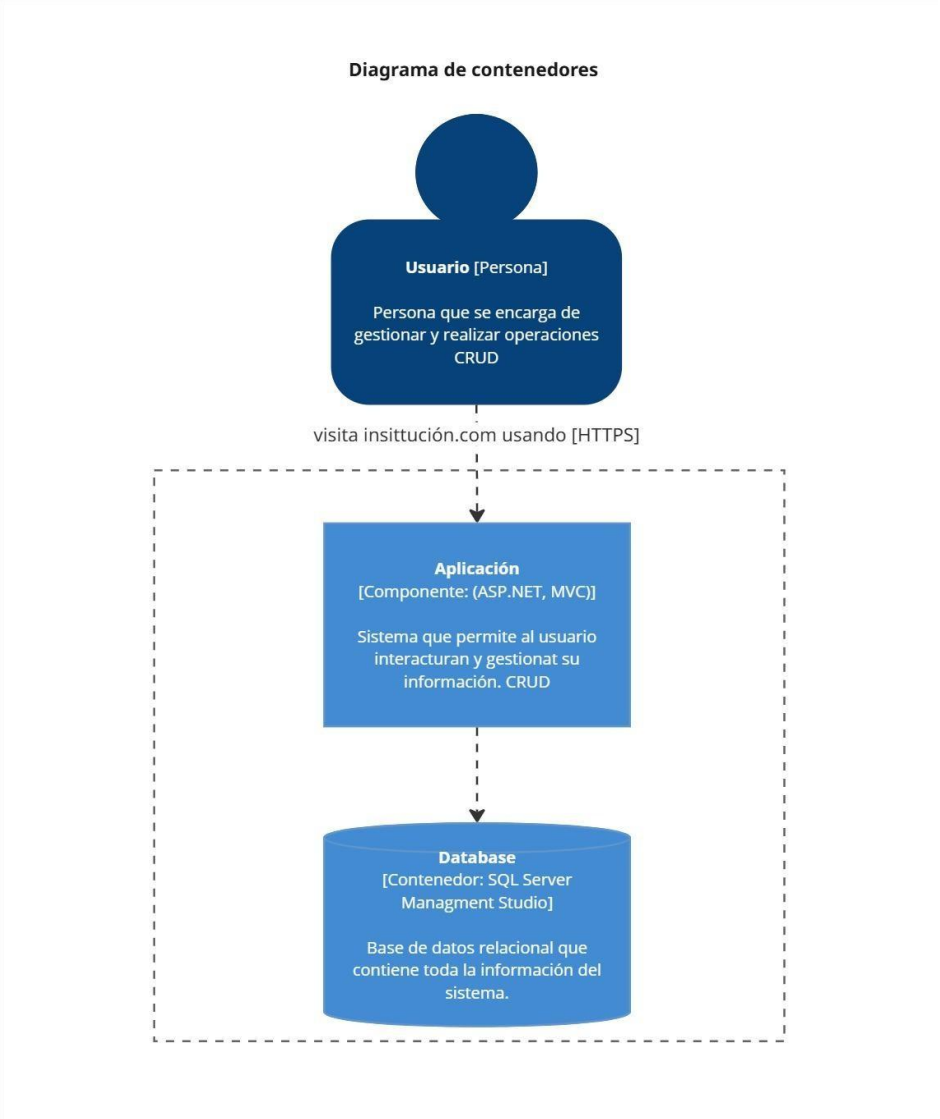


Este diagrama representa la arquitectura de alto nivel de la aplicación monolítica desarrollada en .NET 8 utilizando el patrón MVC (Modelo-Vista-Controlador). La estructura está claramente dividida en tres capas: las **vistas**, que conforman la interfaz gráfica del usuario; los **controladores**, organizados en controladores de API y de vistas, encargados de procesar las peticiones y coordinar la lógica de negocio; y los **modelos**, que contienen las entidades del dominio y los repositorios para el acceso a datos. La aplicación se comunica con una base de datos en **SQL Server**, la cual se inicializa mediante un archivo **init.sql**. Además, el sistema expone una interfaz REST documentada con **Swagger**, permitiendo probar los endpoints de forma interactiva.

Diagrama de contexto

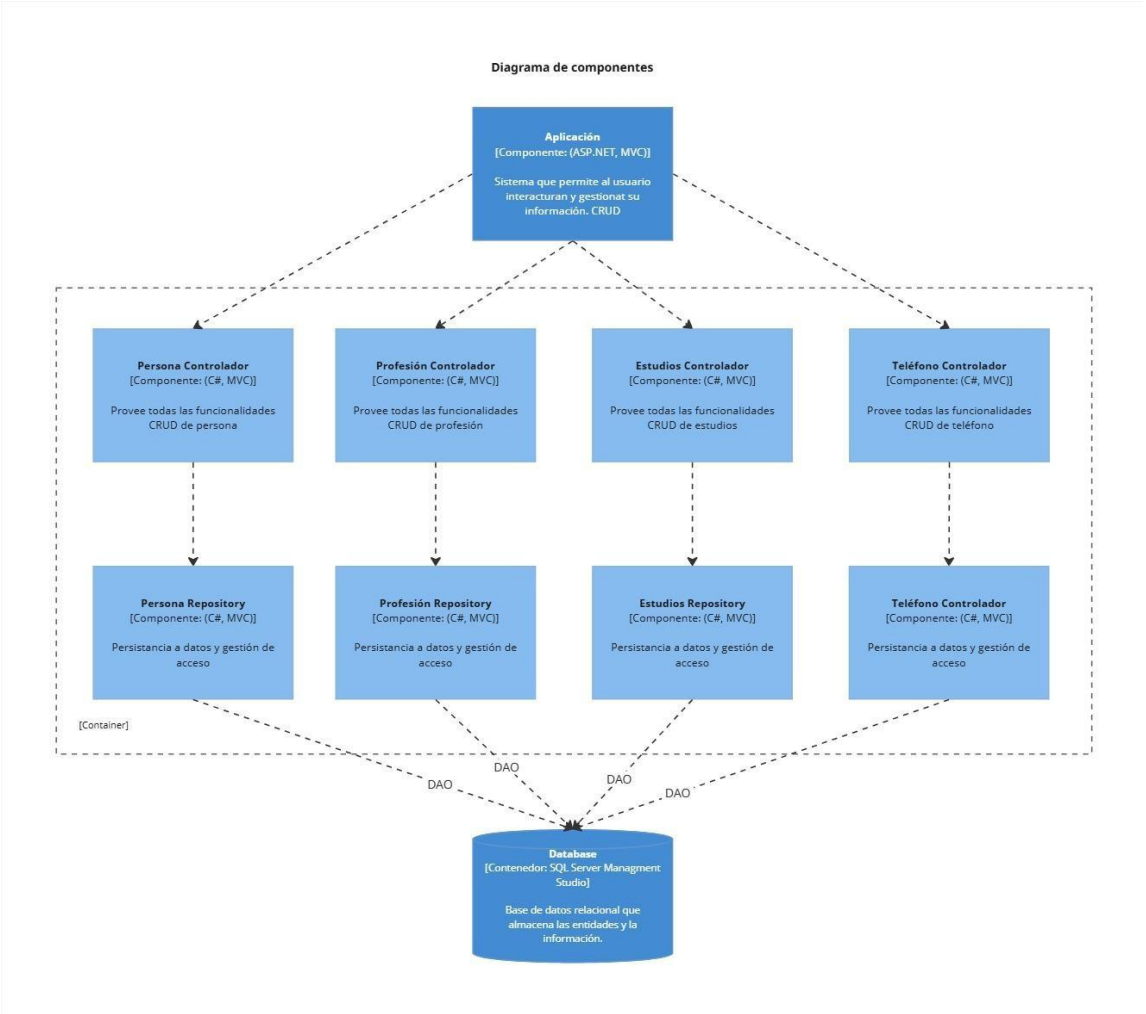
Este diagrama refleja cómo el usuario, quien representa a cualquier persona que accede a la aplicación web, se encarga de realizar operaciones básicas como crear, leer, actualizar y eliminar datos (CRUD). El sistema, identificado como "Dotnet", actúa como intermediario que permite a los usuarios interactuar con la plataforma de forma efectiva para gestionar su información personal o relacionada con las entidades del dominio.

Diagrama de contenedores



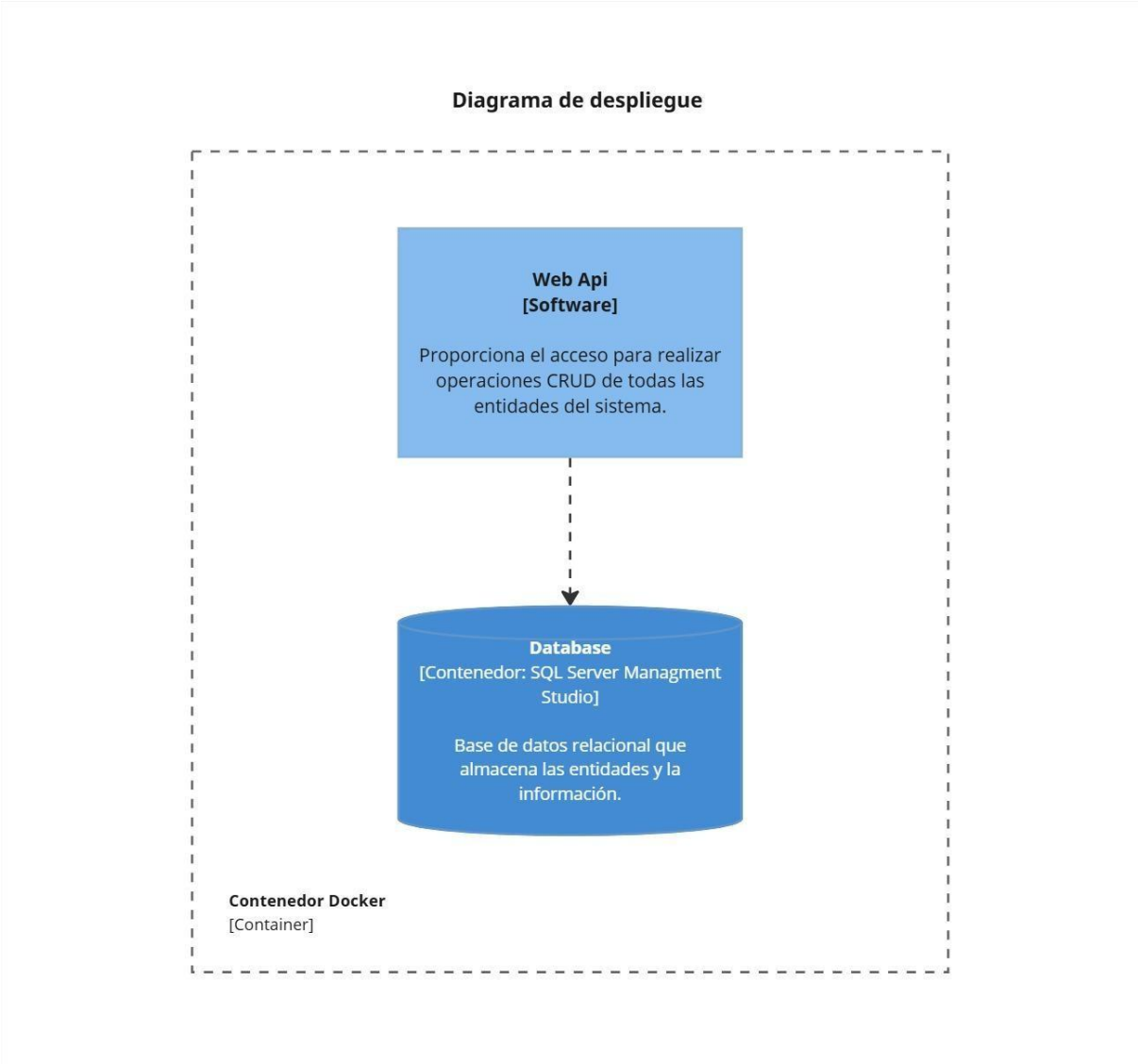
El diagrama de contenedores representa cómo el usuario interactúa con el sistema a través de un navegador web usando HTTPS. El usuario accede a la aplicación web desarrollada con ASP.NET MVC, la cual permite realizar operaciones CRUD para gestionar la información. Esta aplicación se comunica con una base de datos relacional en SQL Server Management Studio, donde se almacena toda la información del sistema. La estructura permite identificar claramente los roles de cada contenedor en el funcionamiento general del sistema.

Diagrama de componentes



El diagrama de componentes muestra la estructura interna del sistema, donde se evidencia la separación de responsabilidades entre controladores, repositorios y la base de datos. La SPA se comunica vía HTTP con los controladores específicos (Persona, Profesión, Estudios y Teléfono), encargados de las operaciones CRUD de cada entidad. Estos controladores acceden a los datos mediante repositorios que se encargan de la persistencia y el acceso a la base de datos SQL Server, donde se almacena toda la información del sistema.

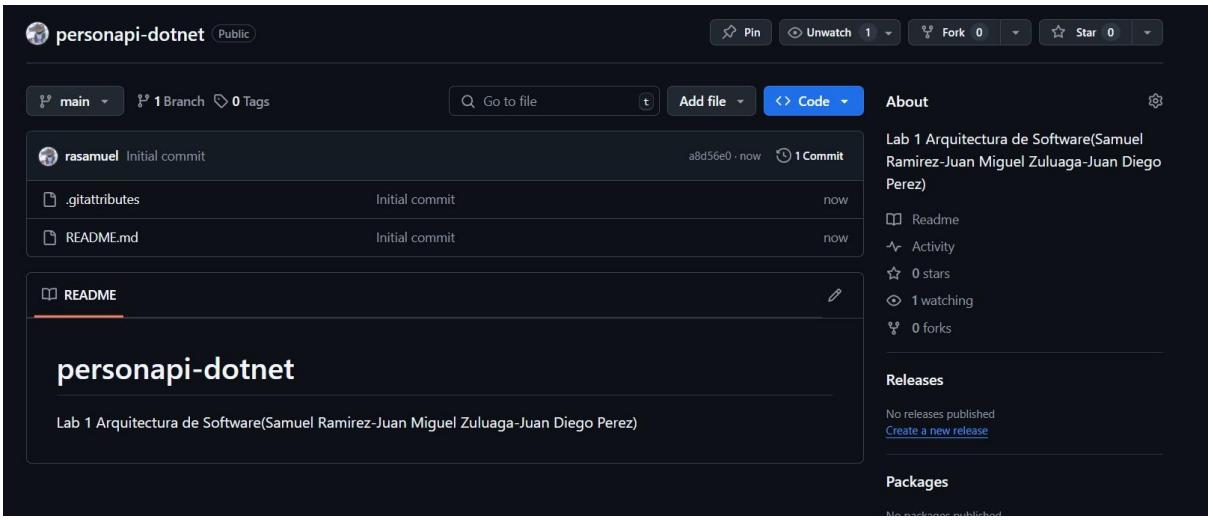
Diagrama de despliegue



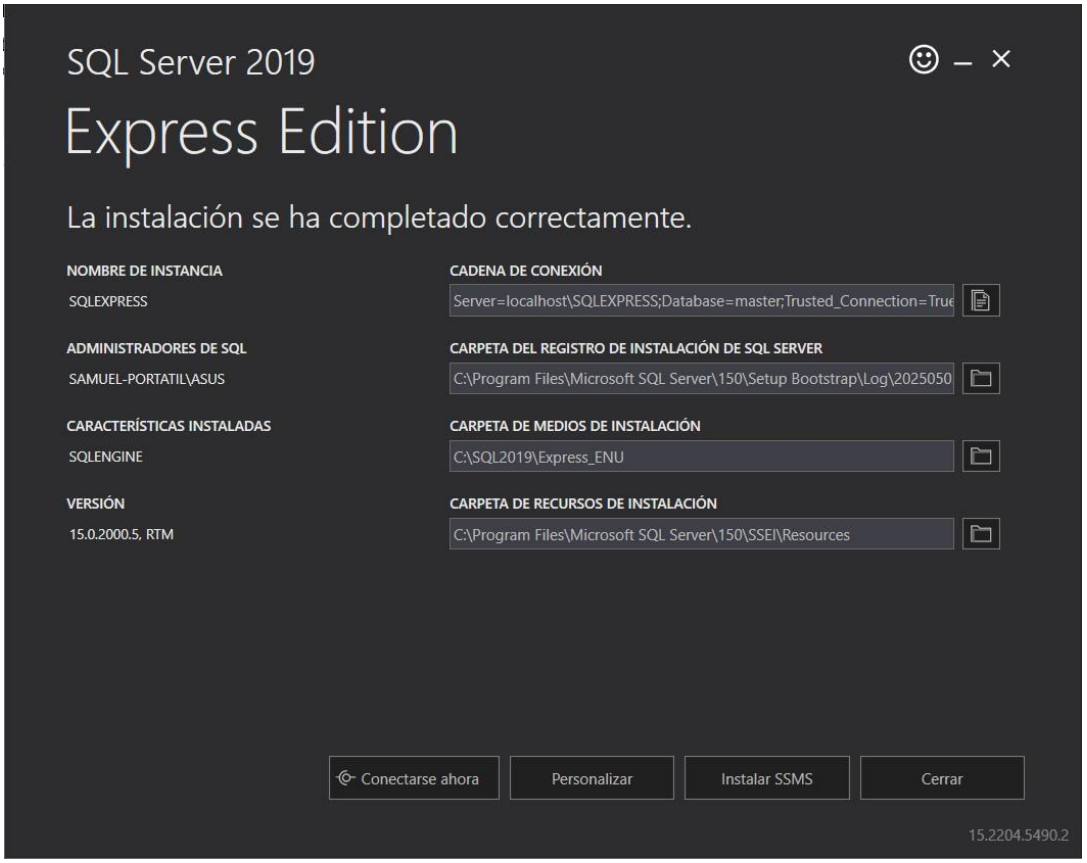
El diagrama de despliegue muestra cómo se distribuyen los componentes del sistema dentro de un contenedor Docker. La Web API gestiona las operaciones CRUD de todas las entidades y se comunica directamente con la base de datos alojada en SQL Server Management Studio, donde se almacena toda la información del sistema.

Procedimiento

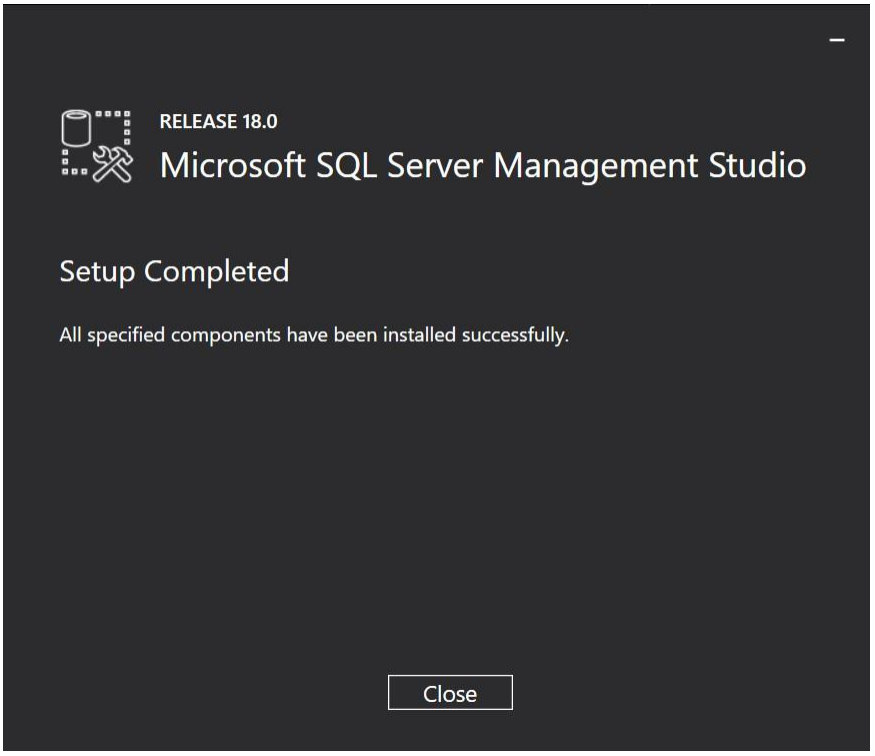
1. crear el repositorio git publico en github o en cualquier sistema git nombre del repositorio debe ser personapi-dotnet



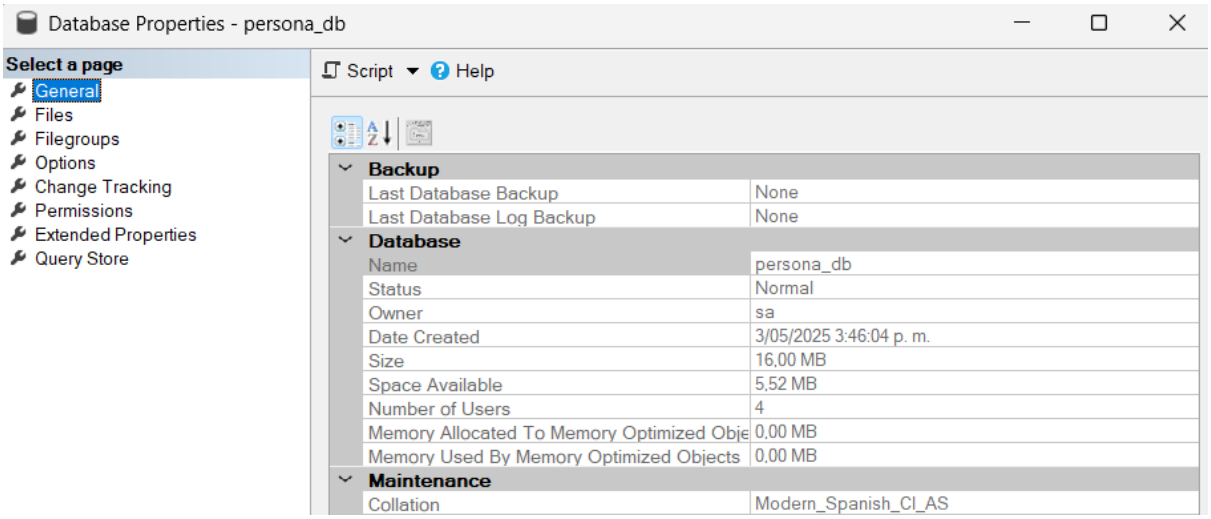
2. instalar SQL Server 2019 Express modo Básico



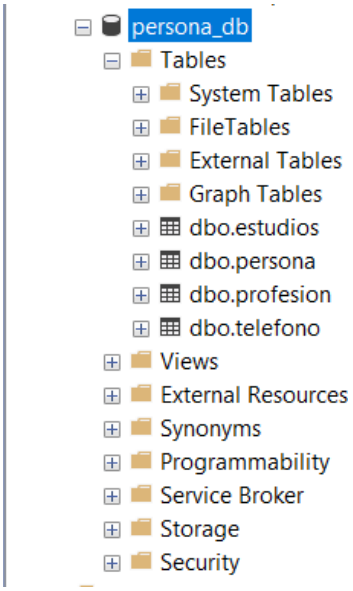
3. instalar SQL Server Management Studio 18



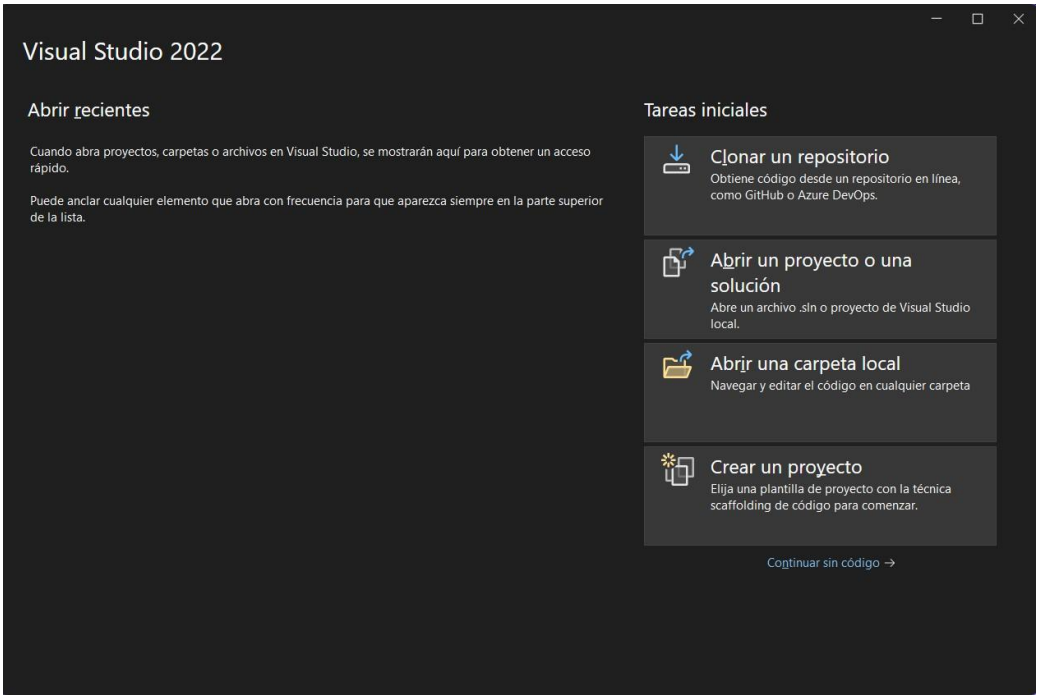
4. crear la base de datos llamada persona_db y darle la propiedad al usuario sa



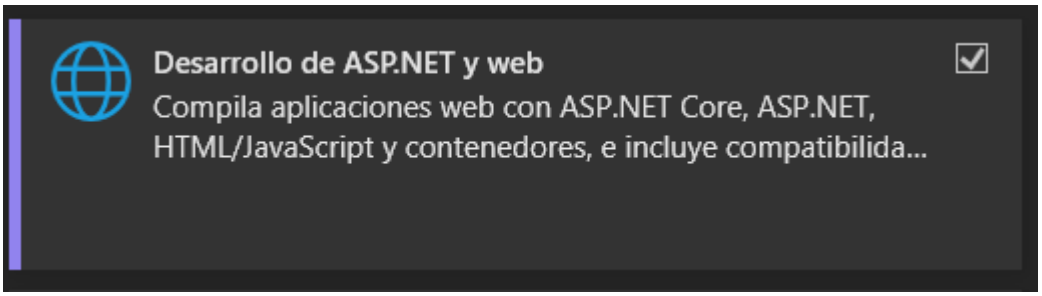
5. crear las tablas según el modelo



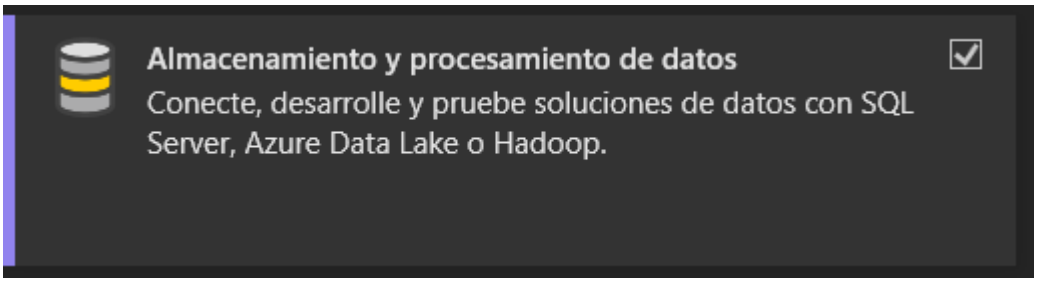
6. instalar Visual Studio Community 2022 con los complementos



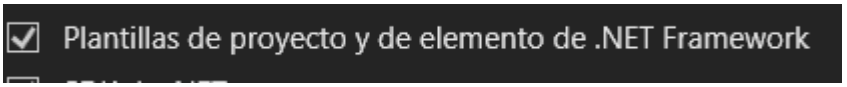
6.1. Desarrollo ASP.NET y web



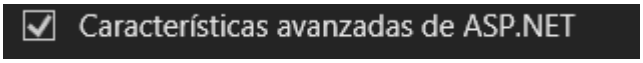
6.2. Almacenamiento y procesamiento de datos



6.3. Plantillas de proyecto y elementos de .Net Framework

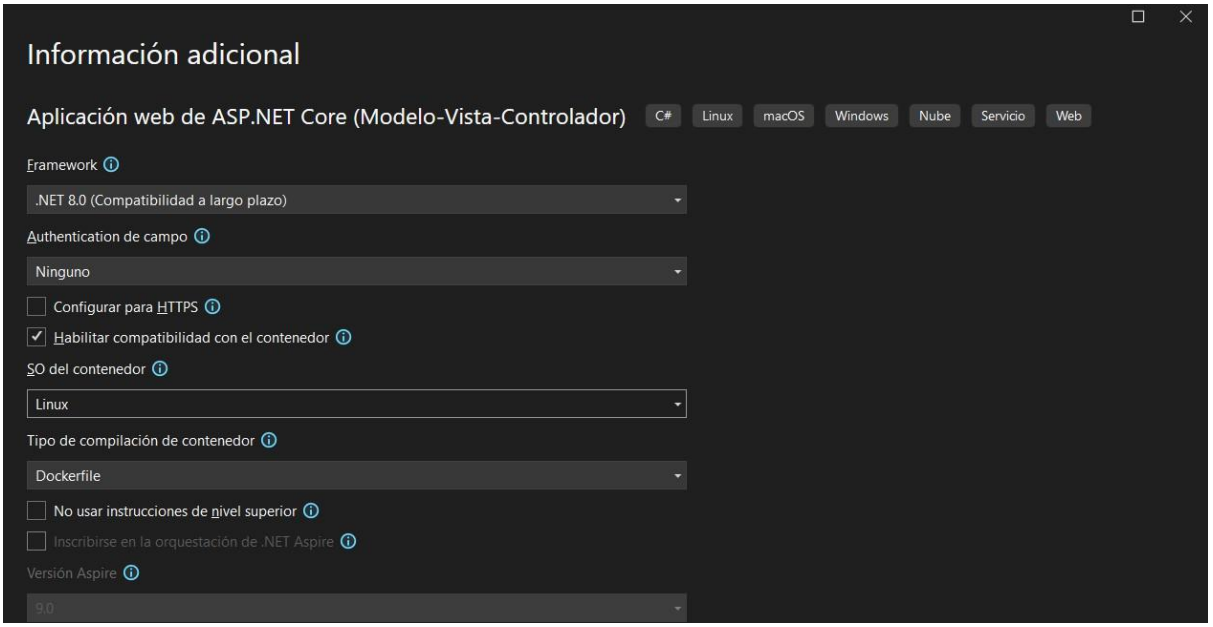


6.4. Caracteristicas avanzadas de ASP.NET



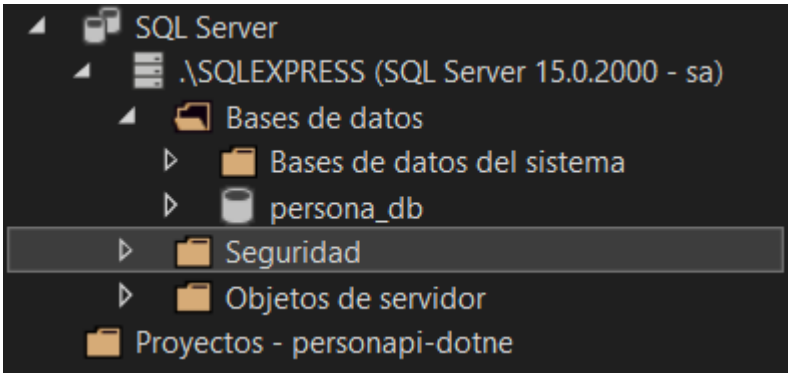
7. clonar el repositorio local git a partir del remoto creado previamente en Visual Studio Community 2022
   7.1. crear un proyecto
   7.2. seleccionar la plantilla Aplicación web de ASP.NET Core (Modelo-Vista-Controlador)
   7.3. el nombre de la aplicación debe ser el mismo del repo personapi-dotnet

7.4. Framework .NET 6.0 sin autenticación y sin configuración HTTPS

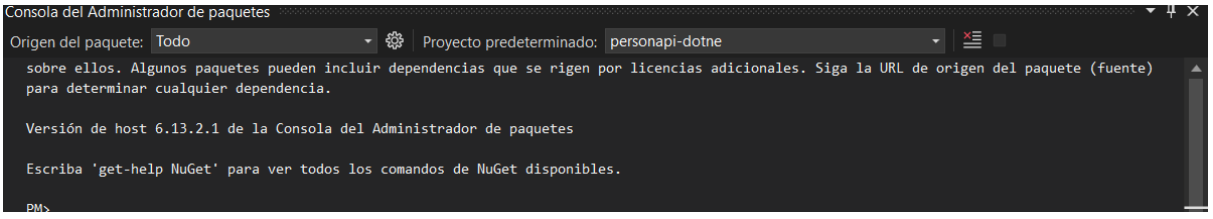


7.5. en el menu Ver activar la vista de Explorador de objetos de SQL Server

7.6. agregar y probar la conexión de tipo local express



7.7. ir al menu Herramientas>Administrador de paquetes NuGet>Consola del Administrador de paquetes

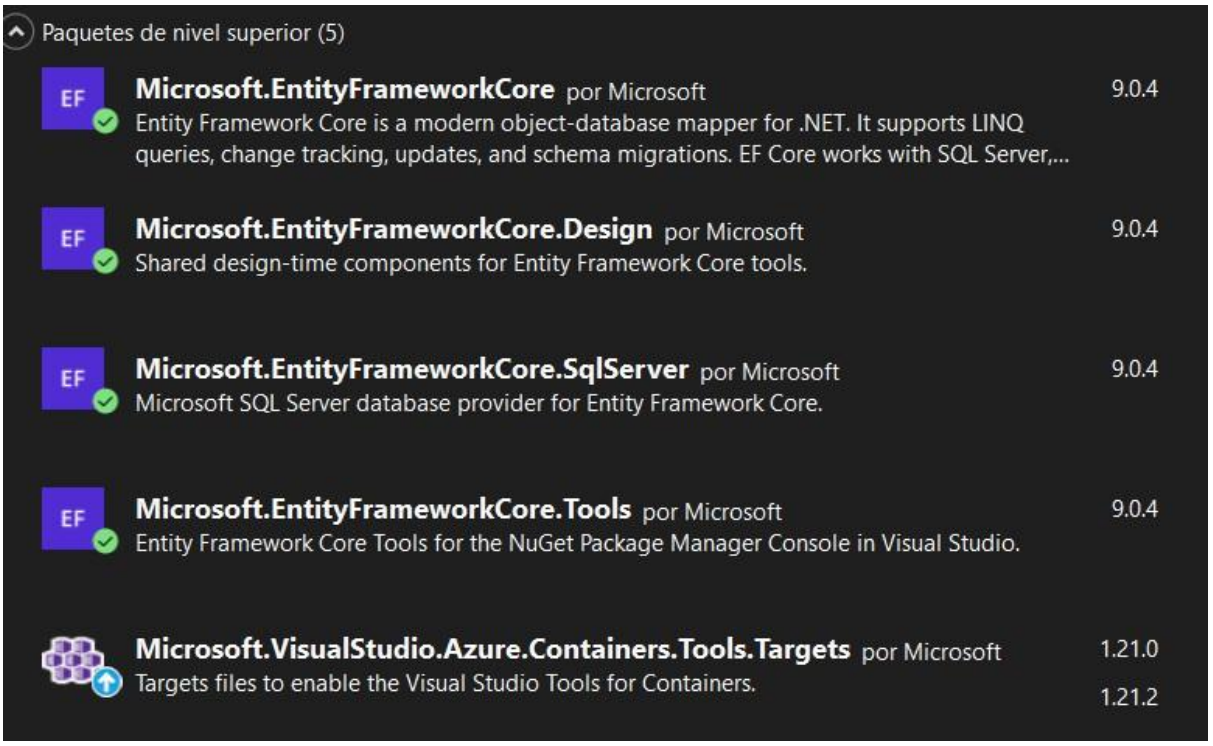


7.8. En el explorador de soluciones, hacer clic derecho en dependencias e ir a Administrar paquetes NuGet e instalar

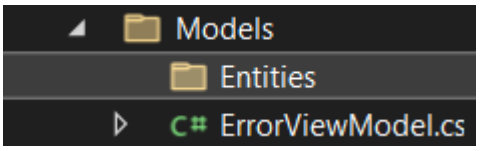
7.8.1. Microsoft.EntityFrameworkCore

7.8.2. Microsoft.EntityFrameworkCore.SqlServer

7.8.3. Microsoft.EntityFrameworkCore.Tools



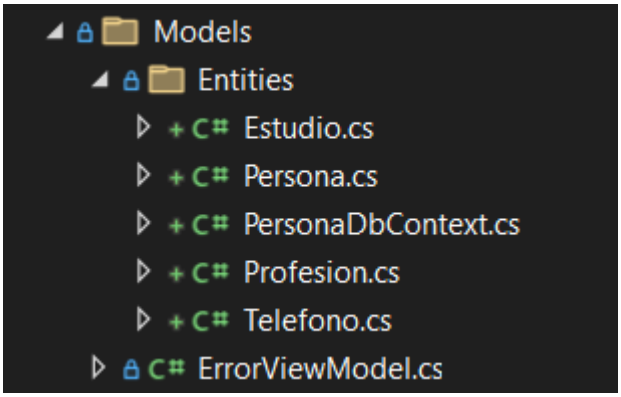
7.9. crear entidades, en el explorador de soluciones, en la carpeta Models hacer clic derecho y en agregar agregar una carpeta llamada Entities



7.10. en la Consola del Administrador de paquetes escribir

7.10.1. Scaffold-DbContext "Server=localhost\SQLEXPRESS;Database=persona_db;Trusted_Connection=True;TrustServerCertificate=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models/Entities

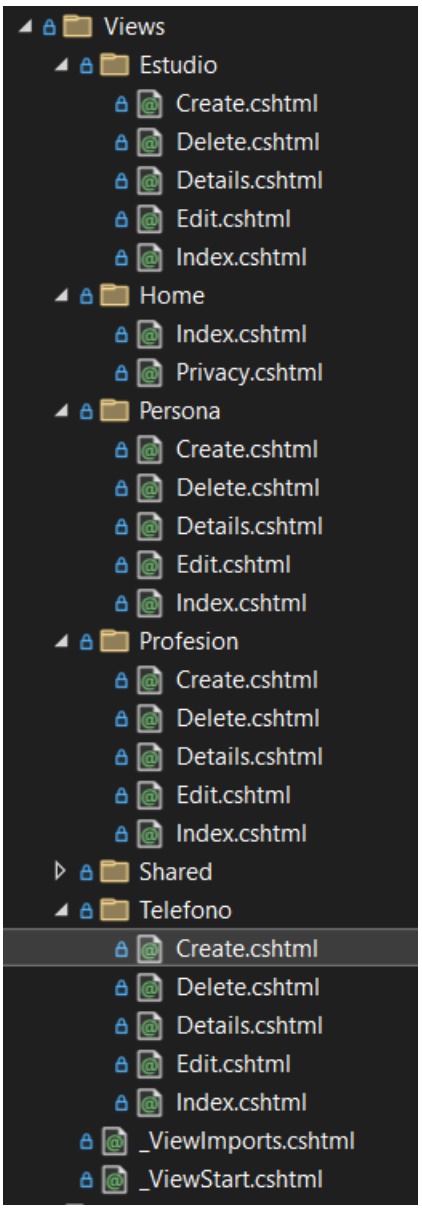
7.10.2. se crean las clases entidad a partir de las tablas existentes de la base de datos y el contexto



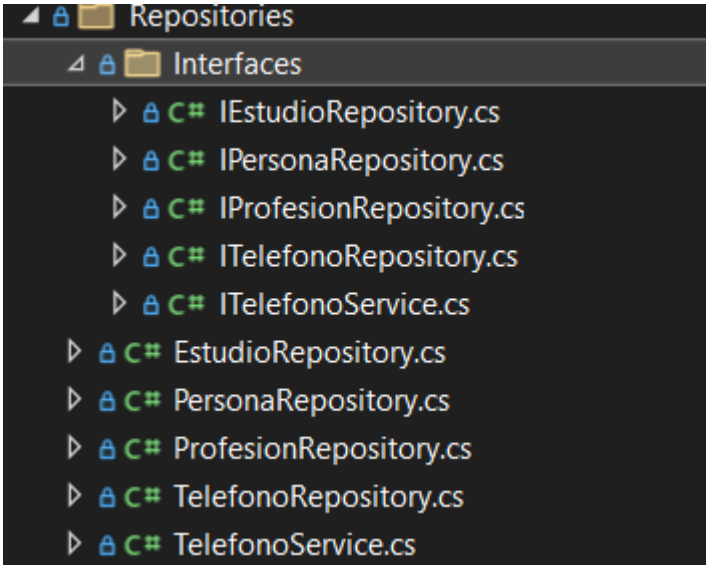
7.10.3. agregar la cadena de coneccion en appsettings.json

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "ConnectionStrings": {
        "DefaultConnection": "Server=localhost\\SQLEXPRESS;Database=persona_db;User Id=sa;Password=1027801475;Tru
    }
}
```

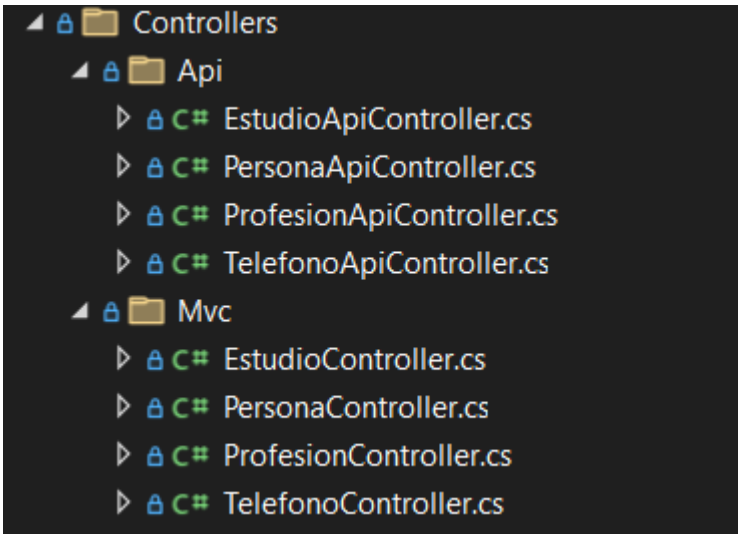
7.11. crear interfaces



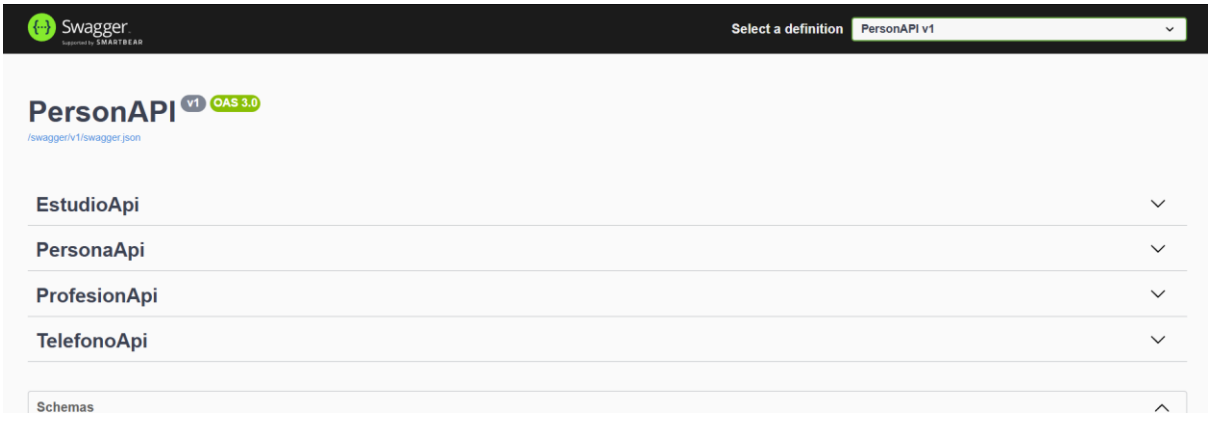
7.12. crear repositorios

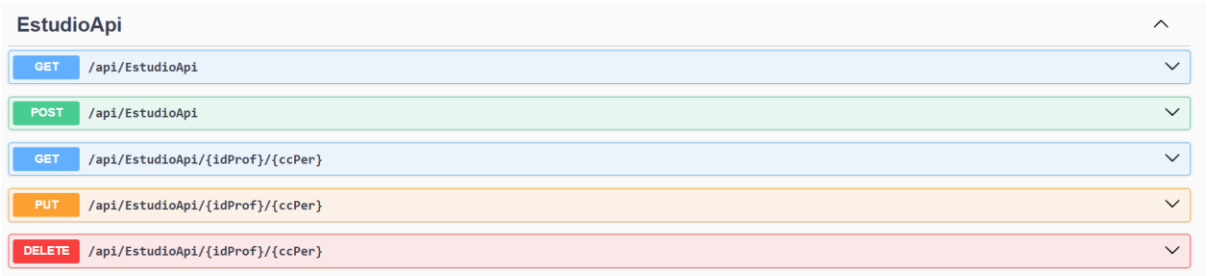


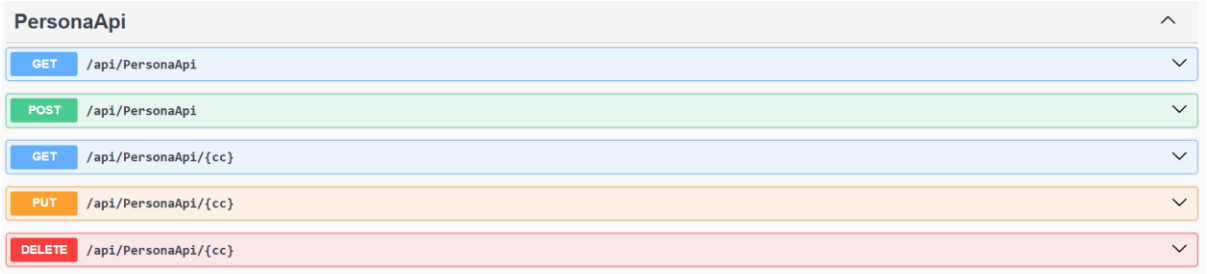
7.13. crear controladores

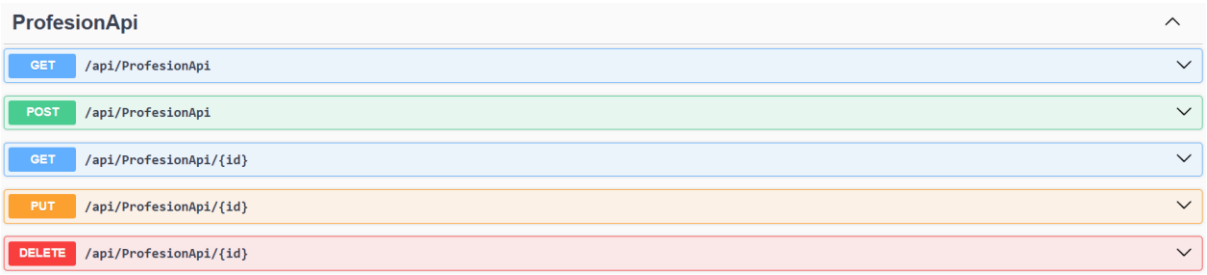


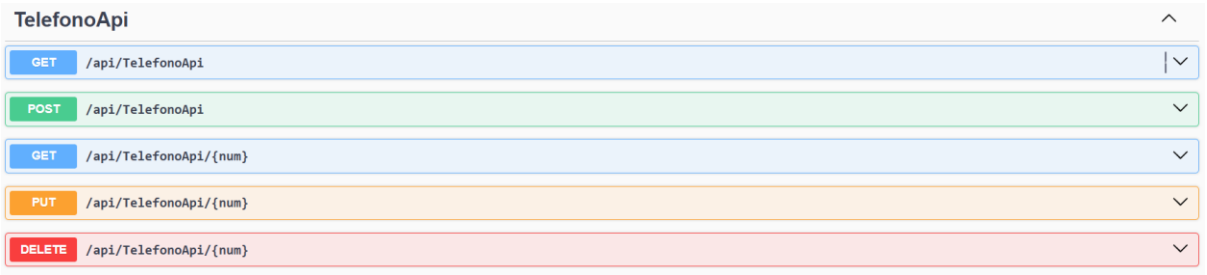
7.14. uso de swagger



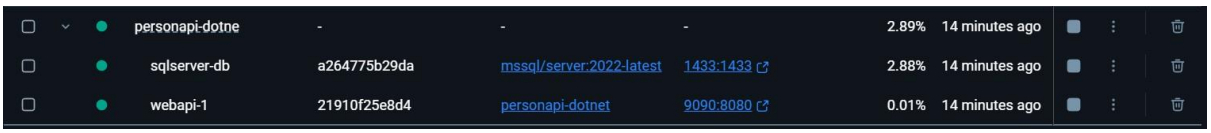


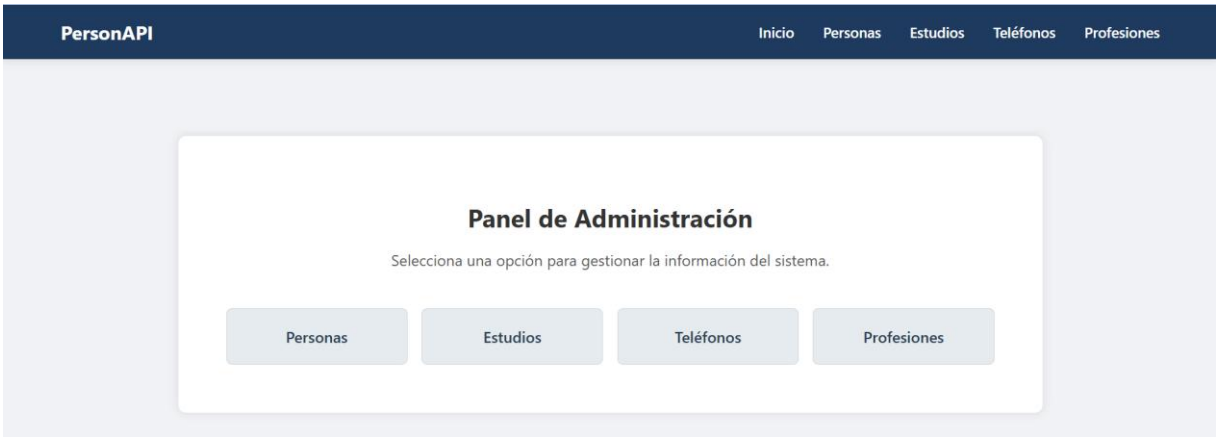






7.15. desplegar





8. hacer push al repositorio
9. crear TAG

Conclusiones y lecciones aprendidas

- **Lecciones Clave sobre ASP.NET Core y Buenas Prácticas**
  - ***Arquitectura moderna y modular:***
    - ASP.NET Core permite desarrollar aplicaciones web modernas de manera estructurada, promoviendo una arquitectura limpia y mantenible mediante la separación de responsabilidades.
  - ***Desacoplamiento mediante repositorios:***
    - Implementar el patrón repositorio ayuda a separar la lógica de negocio del acceso a datos, lo que facilita la realización de pruebas unitarias y el mantenimiento del sistema.
  - ***Integración y escalabilidad con MVC y API REST:***
    - La separación entre controladores MVC (para vistas) y controladores API (para servicios REST) permite atender tanto a usuarios humanos como a aplicaciones externas desde el mismo backend.
  - ***Contenerización con Docker:***
    - Docker simplifica la configuración del entorno de desarrollo, mejora la portabilidad entre equipos y facilita la implementación en diferentes entornos (desarrollo, pruebas, producción).
- **Buenas Prácticas Técnicas con Entity Framework y Validación**
  - ***Validación robusta del lado del servidor:***
    - Validar modelos en el backend es esencial, especialmente cuando se omiten propiedades de navegación al recibir datos desde formularios o clientes API.
  - ***Gestión adecuada de relaciones en EF Core:***
    - Al crear o actualizar entidades con claves foráneas, es crucial asegurar que las relaciones estén correctamente configuradas para evitar errores de integridad referencial.
  - ***Uso estratégico de ModelState.Remove():***
    - Esta función permite ignorar ciertas propiedades en la validación del modelo, útil cuando los formularios o peticiones no envían campos de

navegación.

    - **Seguimiento y depuración con logs:**
        - Utilizar Console.WriteLine o el sistema de logging de ASP.NET Core facilita la detección y resolución de errores durante la validación o ejecución.
- **Conclusión General**
    - Adoptar buenas prácticas como el uso de patrones (repositorio, inyección de dependencias), la validación adecuada de datos, el uso de logs y la contenerización con Docker, permite desarrollar sistemas ASP.NET Core más robustos, escalables y fáciles de mantener. Además, comprender a fondo el funcionamiento de Entity Framework y la validación del modelo es esencial para evitar errores comunes y garantizar una experiencia de desarrollo más fluida.
- **Link Tag Github**
    - **https://github.com/rasamuel/personapi-dotnet/releases/tag/entrega**

Referencias

- https://learn.microsoft.com/es-es/ef/core/
- https://learn.microsoft.com/es-es/aspnet/core/mvc/overview?view=aspnetcore-7.0
- https://learn.microsoft.com/es-es/aspnet/core/tutorials/getting-started-with-swashbuckle?view
  =aspnetcore-7.0&tabs=visual-studio
- https://restfulapi.net
- https://dotnet.microsoft.com/en-us/apps/aspnet