**Laboratorio 1**

**Isabella Blanco Chaparro  
Jairo Andrés sierra Combariza**

**Imagen que contiene Patrón de fondo

Descripción generada automáticamente**

**Facultad de Ingeniería**

**Arquitectura de Software**

**PONTIFICIA UNIVERSIDAD JAVERIANA**

**BOGOTÁ D.C**

**2024**

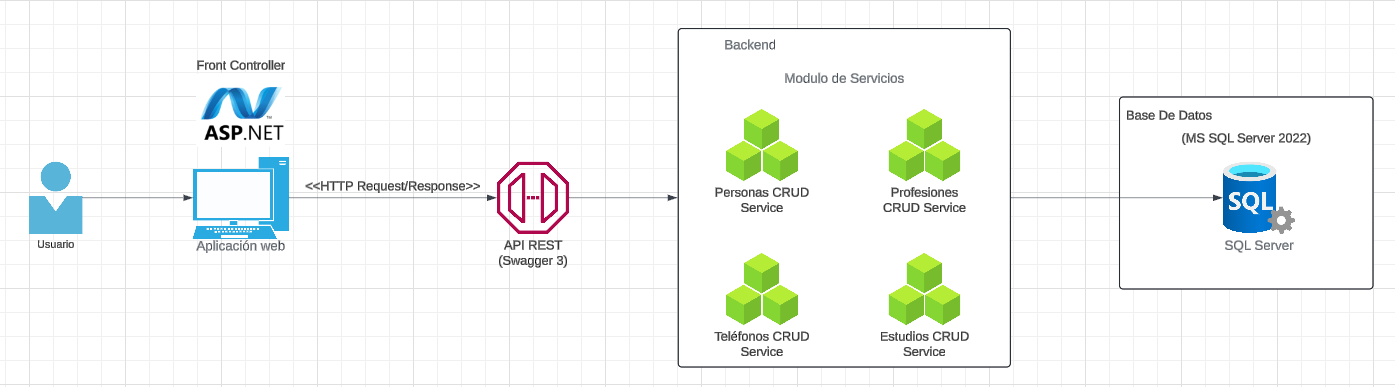
**Marco Conceptual**

En este laboratorio, se trabajará en la implementación de un sistema basado en un modelo monolítico utilizando el patrón Modelo-Vista-Controlador (MVC). Además, se utilizará .NET 7 como framework de desarrollo y Microsoft SQL Server como sistema de gestión de bases de datos.

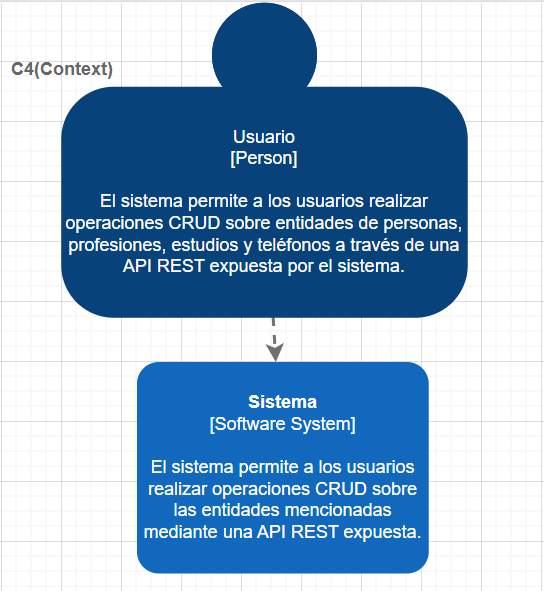
* **Arquitectura Monolítica:** Una arquitectura monolítica es un estilo de diseño en el que una aplicación se construye como un único bloque indivisible, en donde todos sus componentes, desde la interfaz de usuario hasta la lógica de negocio y la base de datos, están interconectados en un solo programa.
* **Patrón MVC:** El patrón MVC es un patrón de diseño que separa las responsabilidades en tres componentes principales:
  + **Modelo (Model):** Representa la estructura de los datos de la aplicación y contiene la lógica de negocio. En este laboratorio, los modelos estarán vinculados a la base de datos a través de Entity Framework Core.
  + **Vista (View):** Encargada de la representación visual de los datos. En ASP.NET Core, las vistas se crean utilizando archivos Razor que permiten mezclar código C y HTML.
  + **Controlador (Controller):** Actúa como intermediario entre el modelo y la vista. Gestiona las peticiones del usuario, interactúa con los modelos y selecciona la vista adecuada para renderizar.
* **Entity Framework Core:** Entity Framework Core es un ORM (Object-Relational Mapper) que facilita el trabajo con bases de datos al permitir manipular los datos como objetos. A través de EF Core, generaremos las clases entidad basadas en el esquema de la base de datos y gestionaremos las operaciones de consulta y manipulación de datos.
* **Conexión a Base de Datos:** Se configurará una cadena de conexión en el archivo ‘appsettings.json’ para conectarse a la base de datos persona\_db en SQL Server.
* **MS SQL Server 2022:** es un sistema de gestión de bases de datos relacional utilizado para almacenar y gestionar los datos de la aplicación. Para este laboratorio, se empleará una versión Express de SQL Server para manejar la base de datos persona\_db, que almacenará los datos relacionados con el modelo CRUD.
* **API RESTful:** El término REST (Representational State Transfer) hace referencia a un estilo arquitectónico para servicios web que utiliza operaciones HTTP (GET, POST, PUT, DELETE) para interactuar con los recursos. En este laboratorio, se crearán varios endpoints RESTful para realizar operaciones CRUD sobre las entidades generadas a partir de la base de datos.
* **Swagger:** Es una herramienta que permite documentar y probar de manera interactiva los servicios web REST. Proporciona una interfaz visual que ayuda a los desarrolladores y usuarios a entender cómo interactuar con la API. Se implementará Swagger en el proyecto para facilitar la documentación de los endpoints y la interacción con ellos.
* **CRUD (Create, Read, Update, Delete):** Se refiere a las cuatro operaciones básicas que se pueden realizar sobre los datos en un sistema de bases de datos o un API. En el contexto del laboratorio, se implementarán endpoints para la creación, lectura, actualización y eliminación de registros en la base de datos.

**Diseño**

* **Diagrama de Arquitectura de Alto Nivel:**



* **C4 Model:** 
  + **C4 Context:** Este es el nivel más alto de abstracción que da una vista de cómo los actores (usuario) interactúan con el sistema de gestión implementado en este laboratorio. El **usuario** hace solicitudes (CRUD) que son manejadas por el **sistema** (ASP.NET Core) que interactúa con una base de datos para realizar las operaciones solicitadas.

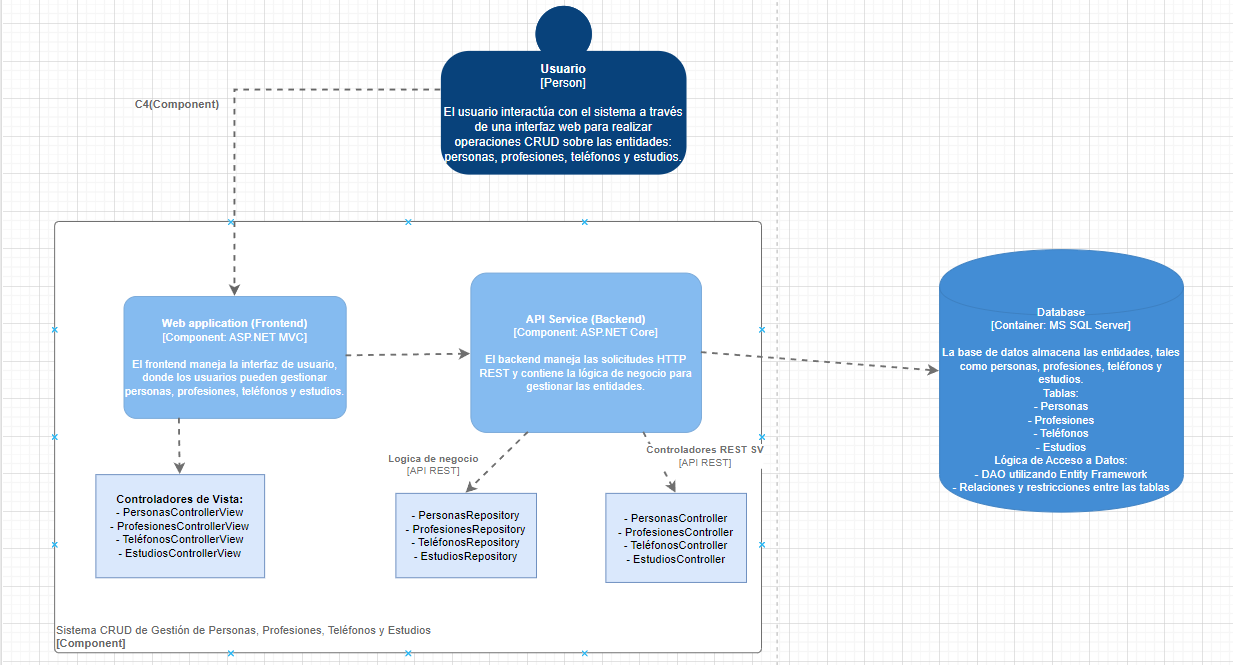


* + **C4 Container:** Se muestra cómo el frontend (interfaz web) y el backend (lógica y API) se conectan entre sí, y cómo interactúan con la base de datos que persiste la información.

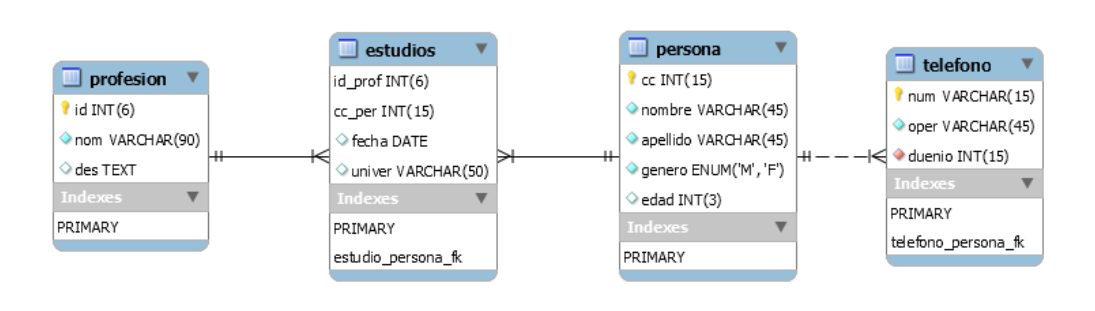
Diagrama

Descripción generada automáticamente

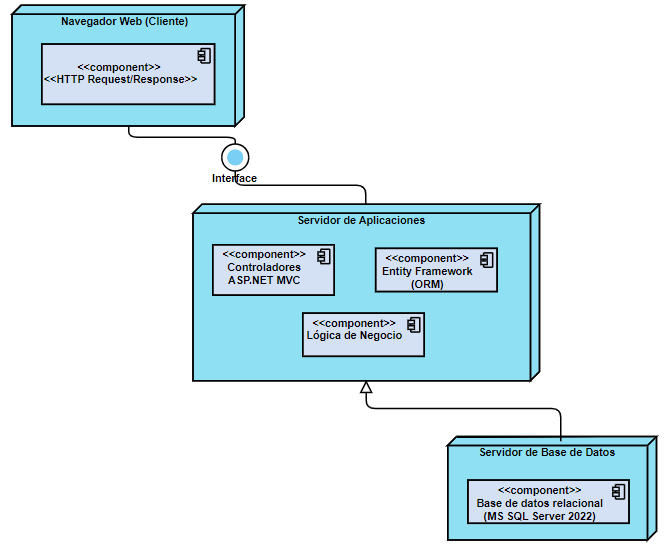
* + **C4 Component:** Muestra cómo los componentes se interrelacionan entre el frontend, el backend y la base de datos, mientras que los servicios de backend manejan la lógica de negocio y los controladores REST exponen las operaciones CRUD.



* **C4 Class UML:**



* **C4 Despliegue**



**Procedimiento**

1. **Creación del Repositorio Git:**

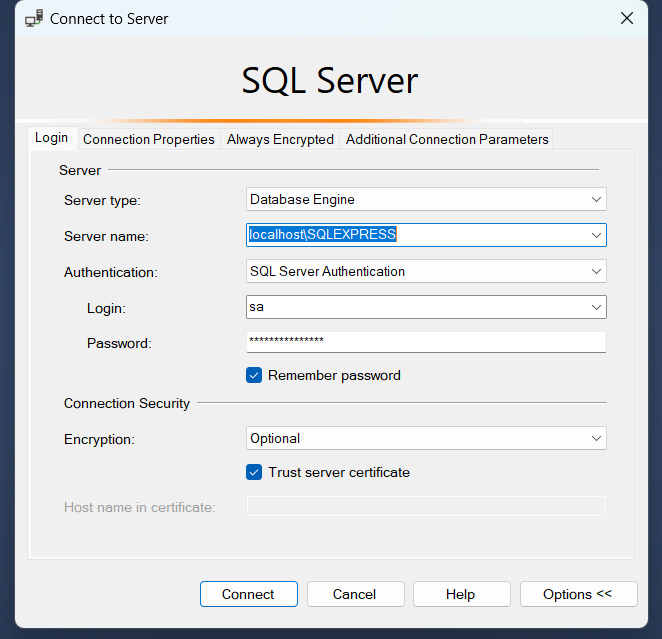
* Crear un repositorio público en GitHub llamado ‘personapi-dotnet’.
* Clonar el repositorio local a partir del remoto.

1. **Instalación de SQL Server:**

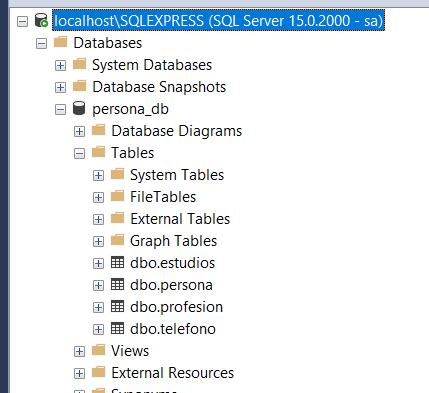
* Instalar SQL Server 2019 Express.
* Instalar SQL Server Management Studio 18.

1. **Configuración de la Base de Datos:**

* Abrir SQL Server Management Studio y conectarse al servidor local (localhost\SQLEXPRESS).



* Crear una nueva base de datos llamada persona\_db y asignar la propiedad de la base de datos al usuario sa (administrador del sistema).
* Crear las tablas necesarias en persona\_db según el modelo de datos definido, usando los scripts DDL proporcionados o diseñados.



1. **Instalación de Visual Studio Community 2022:**

* Descargar e instalar Visual Studio Community 2022.
* Durante la instalación, asegurarse de incluir 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.

1. **Creación del proyecto en Visual Studio:**

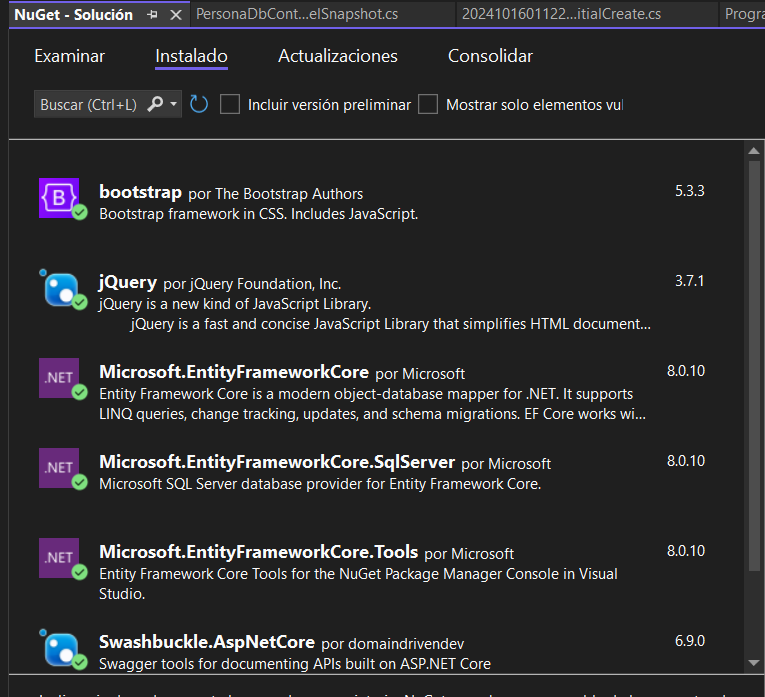
* Clonar el repositorio local desde Git.
* Abrir Visual Studio Community 2022 y crear un nuevo proyecto seleccionando la plantilla Aplicación web de ASP.NET Core (Modelo-Vista-Controlador).
* Nombrar la aplicación como personapi-dotnet.
* Configurar el proyecto con el framework .NET 6.0, sin autenticación y sin configuración HTTPS.

1. **Configuración de la conexión a SQL Server:**

* En el menú Ver de Visual Studio, activar la vista Explorador de objetos de SQL Server.
* Agregar y probar la conexión a la instancia de SQL Server local (express).

1. **Instalación de paquetes NuGet para Entity Framework:**

* Abrir la Consola del Administrador de Paquetes en el menú Herramientas > Administrador de Paquetes NuGet > Consola del Administrador de Paquetes.
* En el explorador de soluciones, hacer clic derecho en Dependencias y seleccionar Administrar Paquetes NuGet.
* Instalar los siguientes paquetes:
  + Microsoft.EntityFrameworkCore.
  + Microsoft.EntityFrameworkCore.SqlServer.
  + Microsoft.EntityFrameworkCore.Tools.

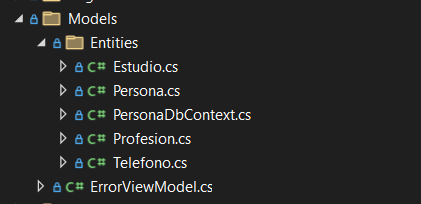


1. **Generación de las entidades con Scaffold-DbContext:**

* En la Consola del Administrador de Paquetes, ejecutar el siguiente comando para generar las entidades y el contexto a partir de la base de datos:

***Scaffold-DbContext "Server=localhost\SQLEXPRESS;Database=persona\_db;Trusted\_Connection=True;TrustServerCertificate=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models/Entities***

* Este comando creará las clases entidad en la carpeta Models/Entities del proyecto basadas en las tablas de la base de datos persona\_db.



1. **Configuración del archivo appsettings.json:**

* En el archivo **appsettings.json** del proyecto, agregar la cadena de conexión a la base de datos:

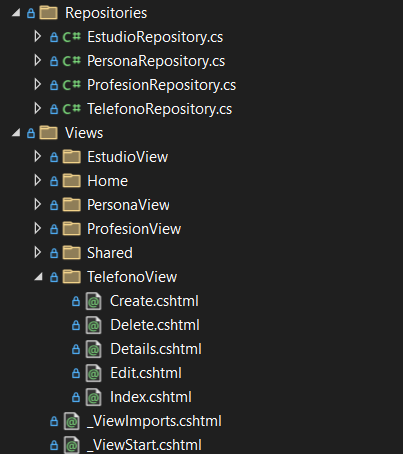
***"ConnectionStrings": {***

***"DefaultConnection": "Server=localhost\\SQLEXPRESS;Database=persona\_db;Trusted\_Connection=True;TrustServerCertificate=true"***

***}***

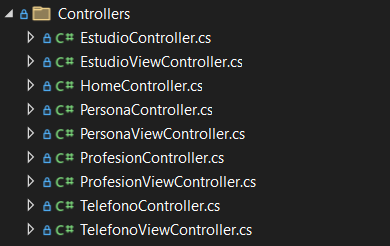
1. **Creación de interfaces y repositorios:**

* Crear una carpeta llamada views y dentro de ella definir las interfaces que describen las operaciones CRUD.
* Crear una carpeta llamada Repositories para implementar los repositorios que interactúan con la base de datos usando Entity Framework Core.



1. **Creación de controladores:**

* Crear controladores en la carpeta Controllers para manejar las operaciones de cada entidad generada. Asegurarse de implementar los métodos CRUD (Create, Read, Update, Delete) en los controladores.



1. **Documentación y pruebas con Swagger:**

* Integrar Swagger en el proyecto para documentar y probar los endpoints. Para ello, modificar el archivo Program.cs según la configuración del proyecto:

***services.AddSwaggerGen(c => {***

***c.SwaggerDoc("v1", new OpenApiInfo { Title = "PersonAPI", Version = "v1" });***

***});***

* Ejecutar la aplicación y navegar a la interfaz de Swagger en http://localhost:<puerto>/swagger para ver la documentación de la API y probar los endpoints.

1. **Implementación de Docker:**

* Crear un archivo **Dockerfile** en la raíz del proyecto para contener la configuración de la imagen Docker
* Construir la imagen Docker, esto generará una imagen llamada personapi-dotnet, que incluye todos los componentes necesarios para ejecutar la aplicación.:

***docker build -t personapi-dotnet***

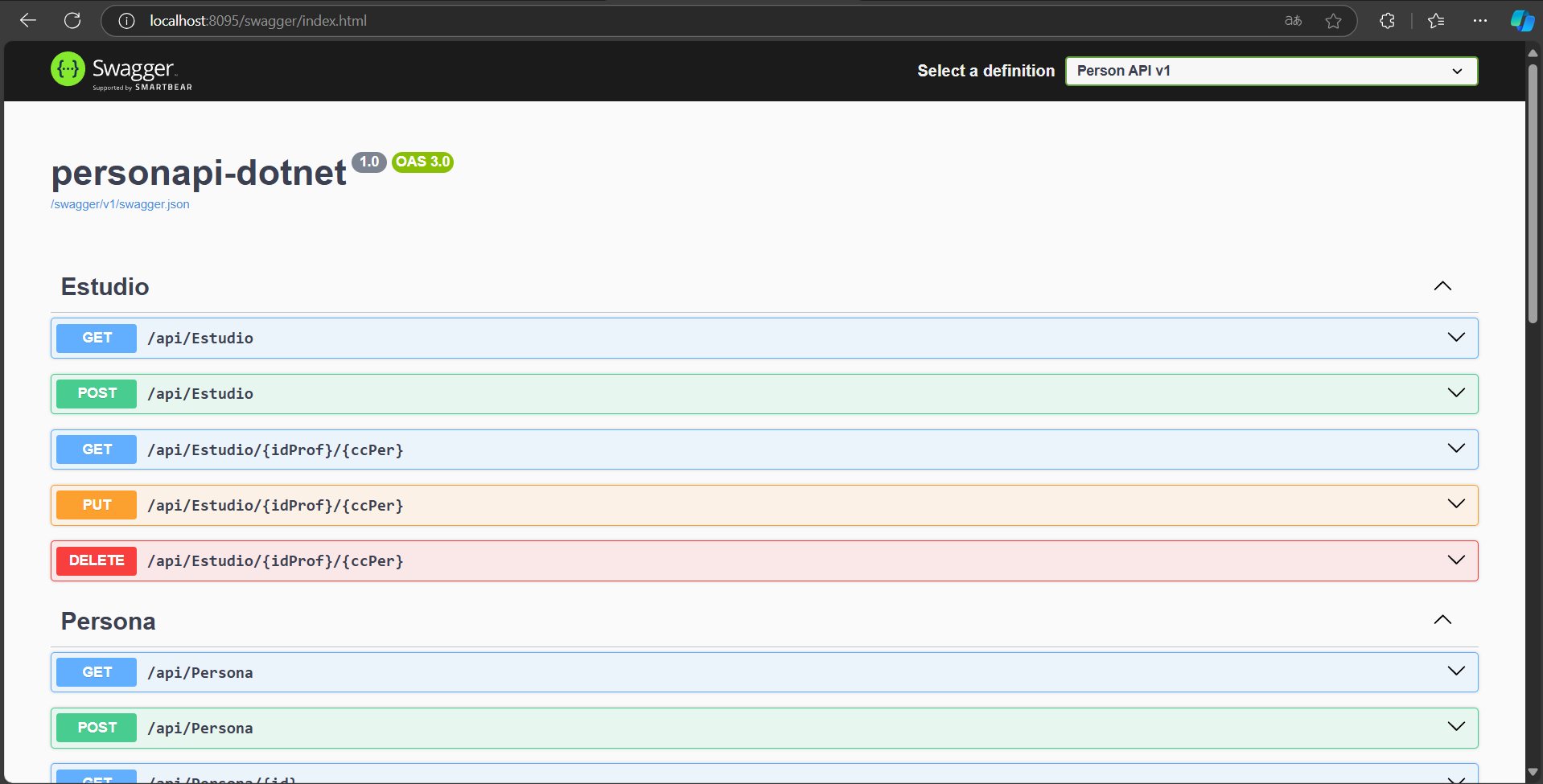
* Ejecutar la aplicación en un contenedor:

***docker run -p 8095:80 personapi-dotnet***

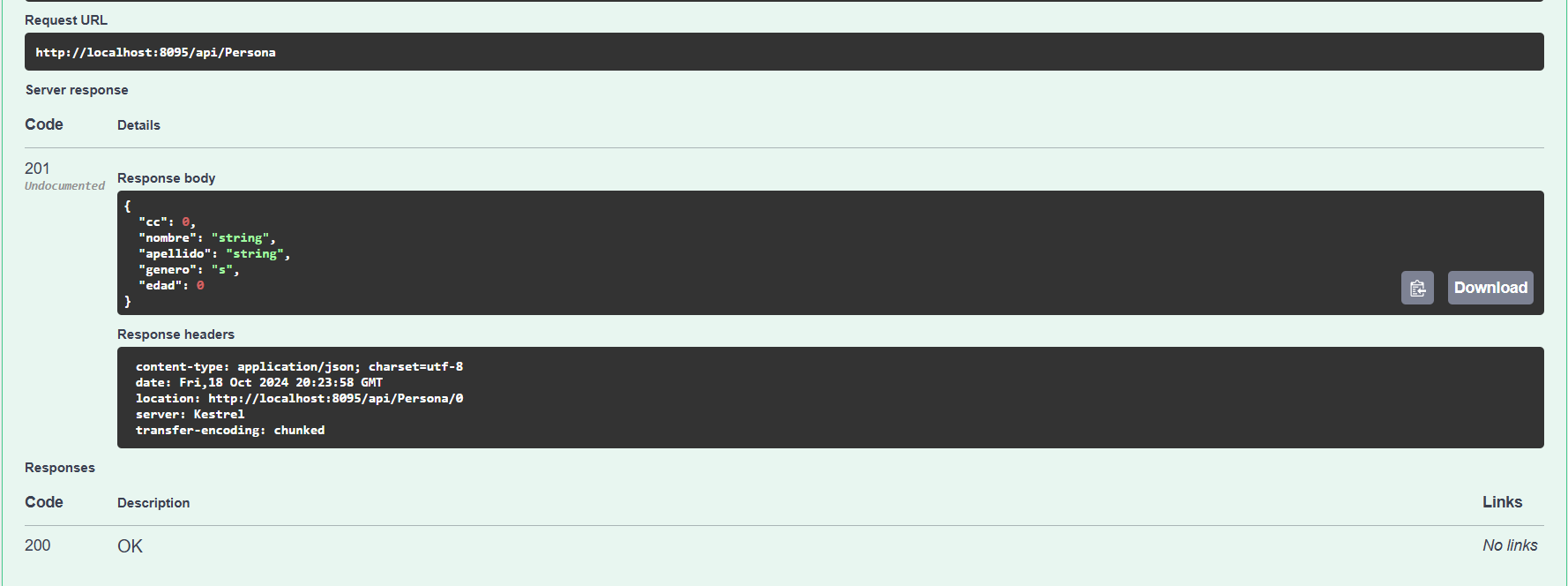
* Levantamiento de servicios con ‘docker-compose’, se iniciará tanto la aplicación ASP.NET Core como la base de datos SQL Server en contenedores separados pero conectados.
* Una vez que los contenedores están en ejecución, la aplicación será accesible desde ‘http://localhost:8080’.

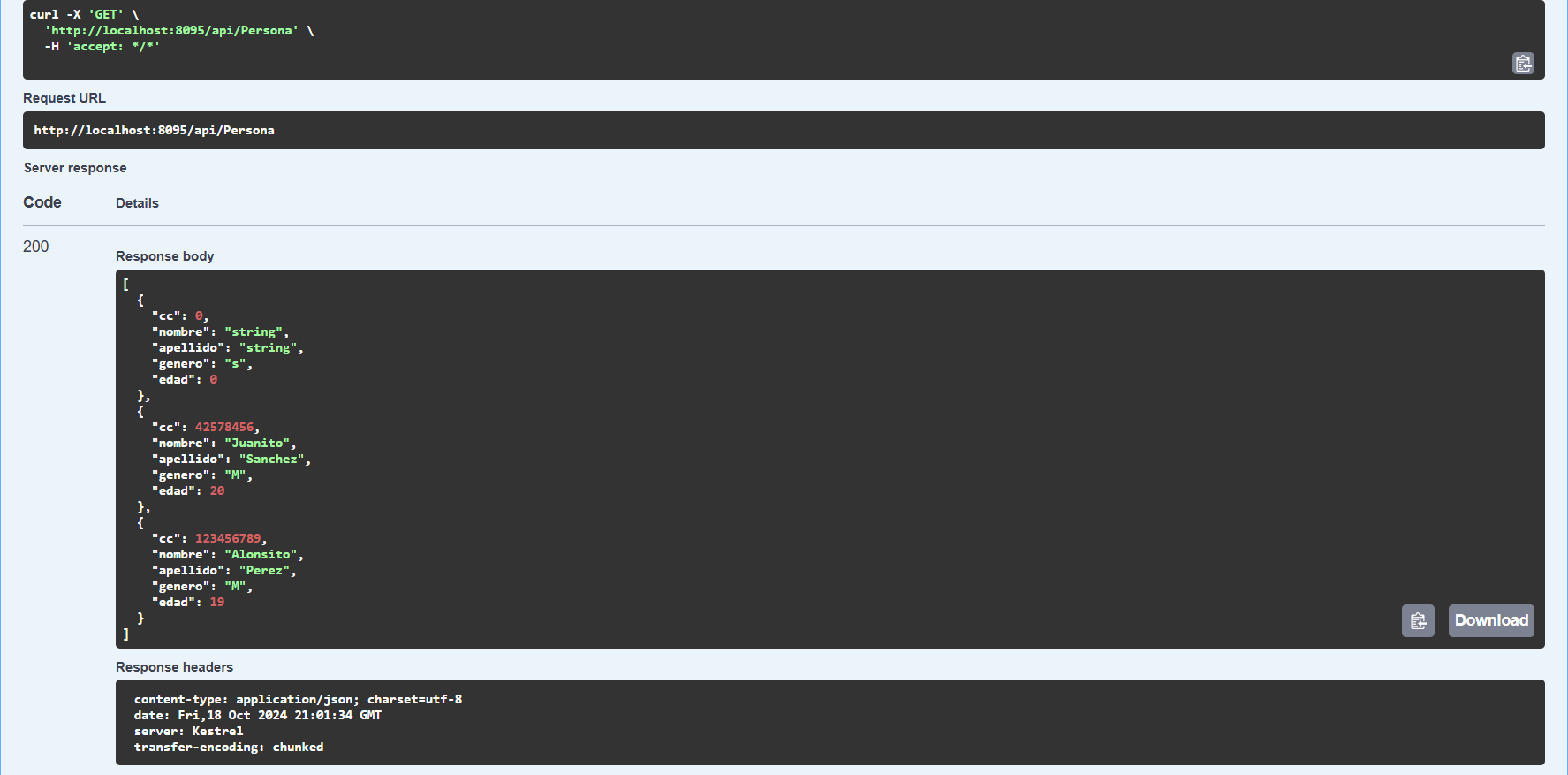
1. **Despliegue y pruebas:**

* Realizar el despliegue de la aplicación y probar todas las funcionalidades CRUD a través de Swagger.

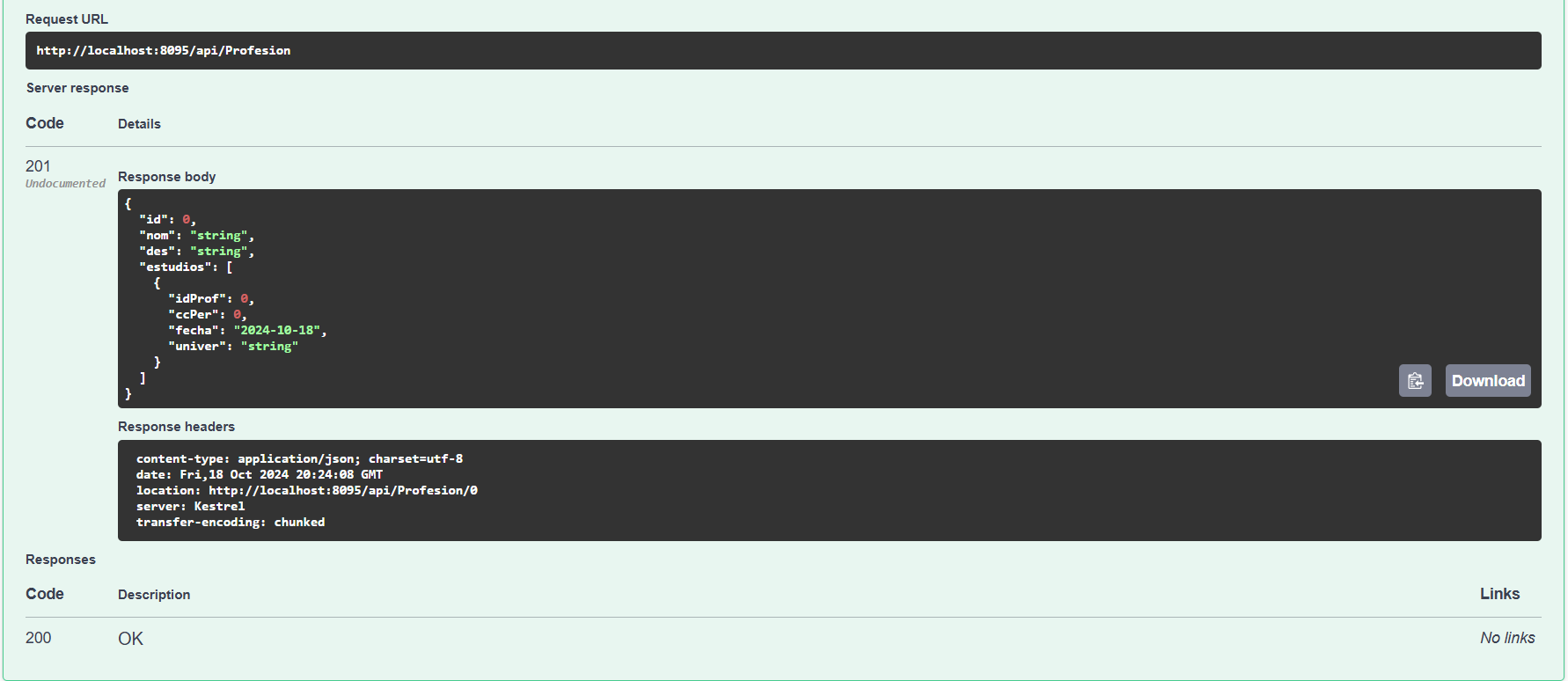


* En la imagen se muestran los endpoints disponibles para realizar las pruebas CRUD (Crear, Leer, Actualizar, Eliminar). Permite demostrar que los endpoints están operativos y listos para ser probados. Swagger facilita la validación de las operaciones de la API sin necesidad de usar herramientas externas.
* Pruebas de endpoints con Swagger:
* Creacion de Persona:

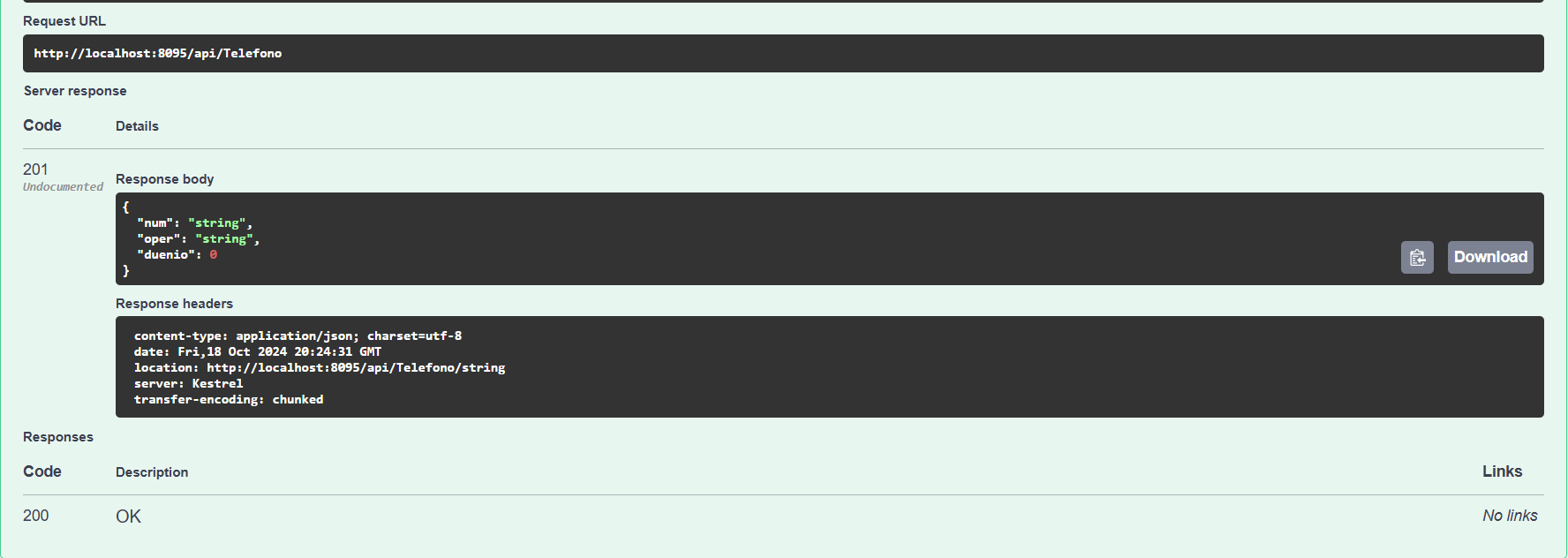


* Get Persona:  
  
* Creación de Estudio



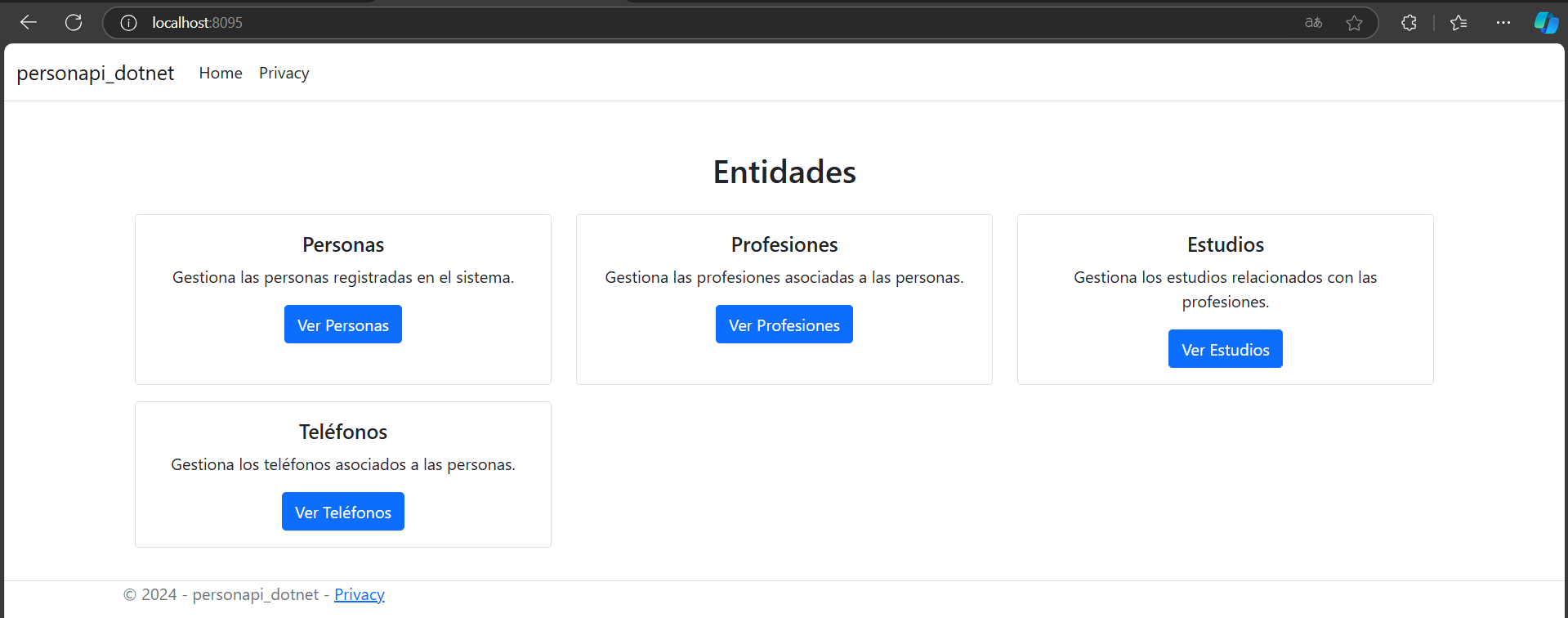
* Get Estudio   
  
* Creación de Profesion  
  
* Get Profesión

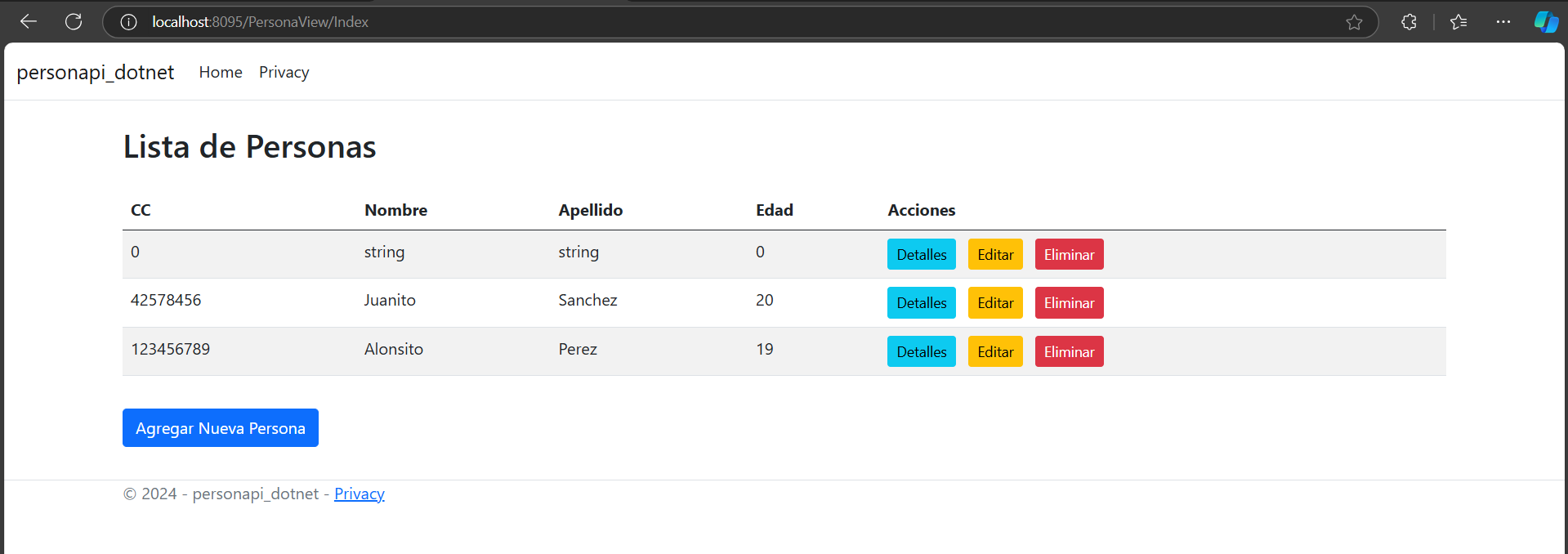


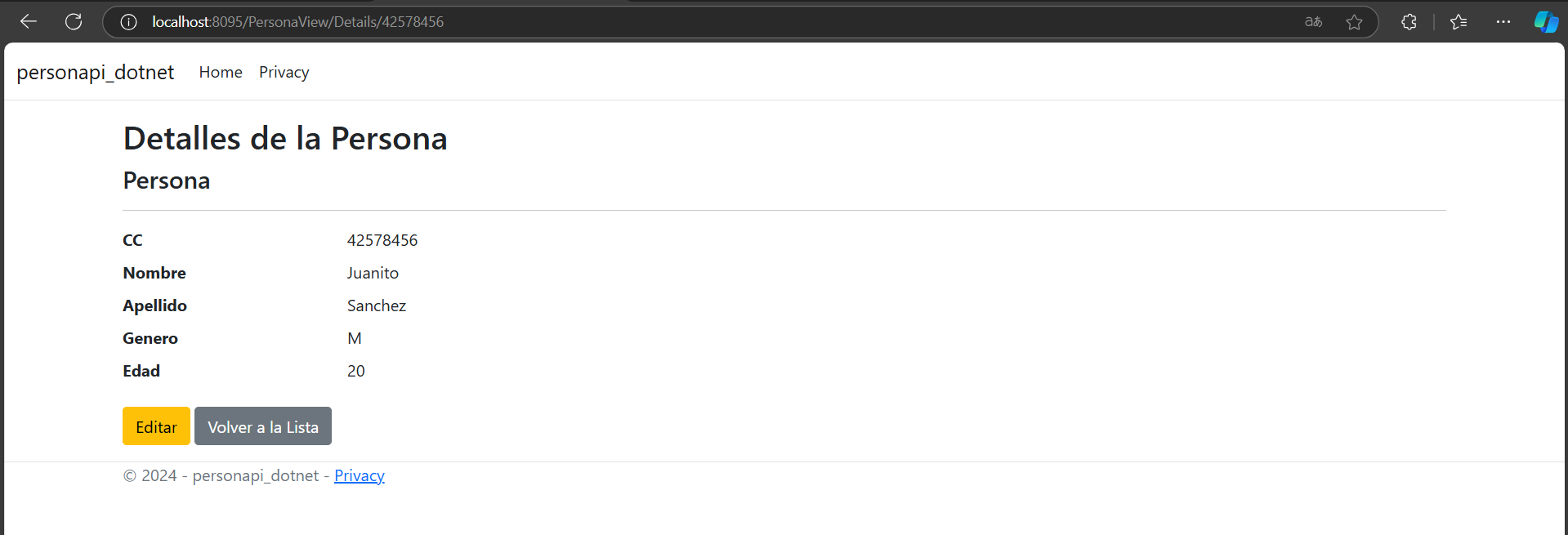
* Creación de Telefono  
  
* Get Telefono

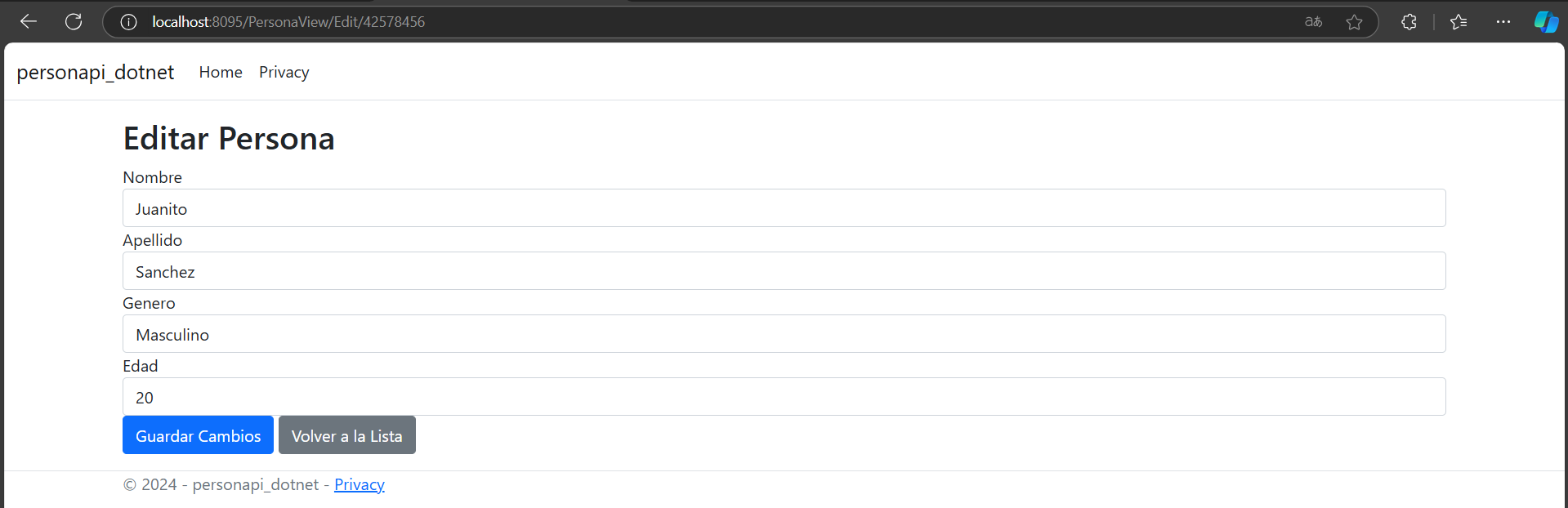


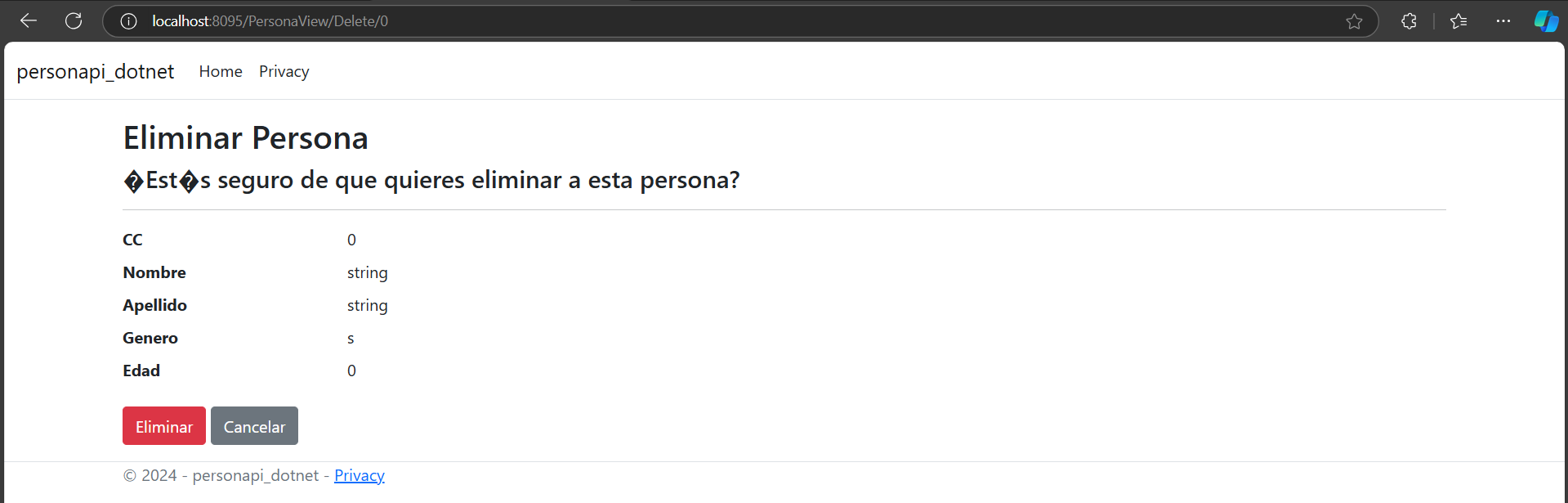
**Pantallas:** Las pantallas y los procedimientos de las operaciones CRUD mostrados para la entidad Persona se repiten de la misma manera para las entidades Estudio, Profesión y Teléfono. Los pasos son idénticos en estructura y funcionalidad.











1. **Versionamiento y creación de TAG:**

* Hacer push de todo el código al repositorio remoto en GitHub.
* Crear un TAG en el repositorio con la versión final del laboratorio.

1. **Entrega final:**

* Subir al repositorio el archivo README con la configuración del entorno, los pasos para la compilación y el despliegue.

**Conclusiones y Lecciones Aprendidas**

Durante este laboratorio, se aprendió a desarrollar una aplicación monolítica basada en el patrón MVC utilizando ASP.NET Core. También se comprendió el funcionamiento de Entity Framework Core para gestionar la interacción con bases de datos y cómo estructurar el acceso a los datos con el patrón DAO. El uso de herramientas como SQL Server y Visual Studio facilitó el desarrollo del sistema, desde la creación de la base de datos hasta la implementación del CRUD.

**Referencias:**

* https://www.youtube.com/watch?v=6nT-RjMEG0o&ab\_channel=hdeleon.net
* https://www.youtube.com/watch?v=28LjewDjaz4&ab\_channel=hdeleon.net
* https://dev.to/veronicaguamann/api-con-aspnet-mvc-6-y-sql-server-mediante-entity-framework-core-6-code-first-parte-1-2i05
* https://dev.to/veronicaguamann/api-con-aspnet-mvc-6-y-sql-server-mediante-entity-framework-core-6-code-first-parte-2-4lbg
* https://www.c-sharpcorner.com/article/building-asp-net-web-api-in-net-core-with-entity-framework/
* https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/entity-data-model