



Pontificia Universidad
JAVERIANA
Colombia

Laboratorio 1

Arquitectura de Software

Ing. Andrés Sánchez

Adrián Eduardo Ruiz Cerquera

Miguel Ángel Márquez Posso

Juan Felipe Galvis Vargas

Pontificia Universidad Javeriana

Bogotá, Colombia

Noviembre de 2025

Marco Conceptual.....	4
Arquitectura Monolítica.....	4
Definición.....	4
Historia y Evolución.....	4
Ventajas.....	4
Desventajas.....	4
Casos de uso y aplicación.....	4
Patrón MVC (Model-View-Controller).....	5
Definición.....	5
Historia y Evolución.....	5
Ventajas.....	5
Desventajas.....	5
Casos de uso y aplicación.....	5
DAO (Data Access Object).....	6
Definición.....	6
Historia y Evolución.....	6
Ventajas.....	6
Desventajas.....	6
Casos de uso y aplicación.....	6
.NET y ASP.NET Core.....	7
Definición.....	7
Historia y Evolución.....	7
Ventajas.....	7
Desventajas.....	7
Casos de uso y aplicación.....	7
Microsoft SQL Server 2022.....	8
Definición.....	8
Historia y Evolución.....	8
Ventajas.....	8
Desventajas.....	8
Casos de uso y aplicación.....	8
REST (Representational State Transfer).....	9
Definición.....	9
Historia y Evolución.....	9
Ventajas.....	9
Desventajas.....	9
Casos de uso y aplicación.....	10
Swagger / OpenAPI.....	10
Definición.....	10
Historia y Evolución.....	10
Ventajas.....	10
Desventajas.....	10
Casos de uso y aplicación.....	10
Docker.....	11

Definición.....	11
Historia y Evolución.....	11
Ventajas.....	11
Desventajas.....	11
Casos de uso y aplicación.....	11
Diseño.....	12
Diagrama Alto Nivel.....	12
Diagrama de Contexto.....	12
Diagrama de Contenedores.....	13
Diagrama de Componentes.....	14
Diagrama de Despliegue.....	15
Entidades.....	15
Procedimiento.....	16
Creación del proyecto y repositorio.....	16
Base de datos.....	17
Instalación de dependencias.....	18
Entidades.....	18
Controladores y API Restful.....	19
Interfaz.....	19
Swagger.....	20
Uso de Docker.....	20
Despliegue.....	22
Conclusiones y lecciones aprendidas.....	34
Referencias.....	35

Marco Conceptual

Arquitectura Monolítica

Definición

Una arquitectura monolítica es un estilo en el que toda la aplicación (capa de presentación, lógica de negocio y acceso a datos) se empaqueta, despliega y ejecuta como una única unidad (proceso/artefacto). En términos prácticos, el código comparte el mismo espacio de memoria y los módulos internos se comunican por llamadas internas (en memoria), no por la red. Esto reduce la latencia y facilita transacciones locales (consistencia fuerte en una sola base de datos), a costa de un mayor acoplamiento entre componentes.

Historia y Evolución

Durante décadas fue el estándar de facto porque permitió simplificar el desarrollo y la operación en servidores físicos o máquinas virtuales. Con la masificación de la nube y la necesidad de despliegues independientes, surgieron alternativas distribuidas como los microservicios. Aun así, prácticas modernas recomiendan con frecuencia un enfoque monolith-first: iniciar con un monolito bien modularizado y, solo cuando existan motivos claros (escala, independencia de equipos, límites de dominio definidos), extraer servicios.

Ventajas

- Simplicidad operativa: Un único paquete y pipeline de despliegue.
- Depuración y trazabilidad más directas: Todo ocurre en el mismo proceso.
- Rendimiento intra-proceso: Llamadas en memoria y transacciones ACID locales.
- Menor costo inicial: Menos infraestructura distribuida y menor complejidad de observabilidad.

Desventajas

- Acoplamiento alto: Pequeños cambios requieren redeploy completo.
- Escalado principalmente vertical: Difícil escalar selectivamente solo un módulo caliente.
- Barreras organizacionales: Equipos grandes compiten por el mismo repositorio y ciclos de release.
- Riesgo de “big ball of mud” si no se aplican límites y modularidad interna.

Casos de uso y aplicación

Adecuado para MVPs y equipos pequeños, dominios aún en descubrimiento, o sistemas con fuerte lógica transaccional que necesitan coherencia estricta. En entornos académicos o de laboratorio, facilita la integración de patrones (MVC, DAO) y tecnologías (ASP.NET Core, SQL Server) en un único artefacto antes de particionar.

Patrón MVC (Model-View-Controller)

Definición

MVC separa responsabilidades en tres componentes: El Modelo (estado y reglas de dominio), la Vista (presentación/plantillas) y el Controlador (coordina entradas del usuario, invoca lógica y selecciona vistas). Esta separación promueve la cohesión y facilita pruebas unitarias del dominio sin acoplarse a la UI.

Historia y Evolución

El patrón nació en el contexto de Smalltalk en Xerox PARC (finales de los 70). Luego se adaptó a interfaces gráficas y, más tarde, a la Web: los controladores procesan solicitudes HTTP, el modelo encapsula la lógica de negocio/datos y las vistas renderizan HTML/JSON. Frameworks web modernos (por ejemplo, ASP.NET Core MVC) ofrecen convenciones, enrutamiento y plantillas para acelerar esta estructura.

Ventajas

- Separación clara de preocupaciones: Cambios de UI no rompen el dominio.
- Reutilización y testabilidad: El modelo es verificable sin navegador.
- Orden en equipos: Cada rol trabaja en su capa con contratos explícitos.

Desventajas

- Riesgo de controladores “gordos” si la lógica no se empuja a servicios/modelo.
- Curva de aprendizaje cuando se combinan muchas capas (validación, filtros, vistas, rutas).
- Sobre Ingeniería para prototipos triviales si no se limita el alcance.

Casos de uso y aplicación

Se aplica ampliamente en proyectos donde es necesario mantener una separación clara entre la lógica del negocio, la interfaz del usuario y el control del flujo de información. Es muy útil en sistemas de gestión de información como aplicaciones académicas, hospitalarias o empresariales. Por ejemplo, en una plataforma de gestión de proyectos, los modelos representan los datos (proyectos, tareas, usuarios), los controladores manejan las solicitudes y actualizan la lógica (creación, edición, asignación de tareas) y las vistas muestran los resultados al usuario. También se aplica en sistemas de registro médico, donde los modelos almacenan la información de los pacientes, los controladores administran las consultas o exámenes y las vistas muestran los historiales clínicos o resultados de laboratorio.

Asimismo, MVC se usa en proyectos como tiendas en línea, portales de reservas, sistemas de control de inventario o plataformas de aprendizaje virtual, donde existen múltiples vistas (formularios, listados, reportes) que dependen de modelos bien definidos. En una tienda, por ejemplo, los modelos gestionan los productos, los controladores manejan el carrito de compras y las vistas muestran las páginas de catálogo o pago. En un sistema académico, los modelos representan cursos y estudiantes,

los controladores procesan matrículas y las vistas generan boletines o certificados. Gracias a esta estructura, MVC permite desarrollar aplicaciones más ordenadas, fáciles de mantener y de probar, lo que lo hace ideal tanto para proyectos monolíticos como para arquitecturas más modernas basadas en servicios web.

DAO (Data Access Object)

Definición

DAO es un patrón que encapsula el acceso a datos (consultas, inserciones, transacciones y manejo de conexiones), de modo que la lógica de negocio dependa de una interfaz estable y no de detalles de la base de datos o del proveedor. Permite intercambiar motores (SQL/NoSQL) o estrategias (ORM/SQL crudo) con mínimo impacto en la capa superior.

Historia y Evolución

Se popularizó en el catálogo Core J2EE Patterns. En el ecosistema .NET convive con repositorios y ORMs como Entity Framework Core. Muchos equipos usan interfaces DAO para contratos estables y delegan la traducción objeto-relacional al ORM, reduciendo duplicación y acoplamientos.

Ventajas

- Aislamiento de la persistencia: Cambios en el almacenamiento no rompen el dominio.
- Testabilidad: Se pueden simular DAOs con dobles (mocks/stubs) sin tocar la base real.
- Contratos claros: CRUD y consultas especializados por agregado/entidad.

Desventajas

- Más clases/boilerplate si se usa además un ORM.
- Riesgo de sobre-abstracción (duplicar capacidades del ORM sin valor adicional).
- Si el dominio evoluciona rápido, mantener interfaces sincronizadas exige disciplina.

Casos de uso y aplicación

El patrón DAO (Data Access Object) se utiliza para separar la lógica de negocio del acceso a la base de datos, permitiendo que las operaciones de lectura, escritura, actualización o eliminación de información estén encapsuladas en clases específicas que se encargan únicamente de comunicarse con el sistema de almacenamiento. Esto facilita que los controladores y modelos del sistema trabajen con datos sin preocuparse por los detalles técnicos de conexión, consultas o transacciones. Por ejemplo, en una aplicación médica, un DAO de pacientes se encargaría de obtener, registrar o actualizar la información de cada historia clínica en la base de datos, mientras que en un sistema académico, un DAO de estudiantes gestionaría las operaciones relacionadas con matrículas o calificaciones.

.NET y ASP.NET Core

Definición

.NET es una plataforma de desarrollo gratuita, multiplataforma y de código abierto para crear aplicaciones web, móviles, escritorio, IoT y más. ASP.NET Core es el framework web de alto rendimiento sobre .NET que ofrece MVC, Razor Pages, Minimal APIs, SignalR (tiempo real) y middleware para construir APIs y sitios modernos.

Historia y Evolución

La plataforma pasó de .NET Framework (Windows) a .NET Core (multiplataforma) y, desde 2020, converge en .NET 5+ con cadencia anual. Actualmente .NET 8 es versión LTS (soporte extendido) y .NET 9 es STS (soporte estándar). ASP.NET Core heredó el nombre tras la transición desde ASP.NET 5 y consolidó DI, configuración, logging y pipeline de middleware como pilares del desarrollo web moderno.

Ventajas

- Rendimiento competitivo y baja huella.
- Multiplataforma (Windows, Linux, macOS) y contenedores listos.
- Ecosistema maduro (NuGet), plantillas productivas y soporte LTS.
- Integración nativa con OpenAPI/Swagger y herramientas de pruebas.

Desventajas

- Cambios de nombres/versionado pueden confundir a equipos nuevos.
- Algunas bibliotecas antiguas de .NET Framework no son compatibles.
- Requiere comprender el pipeline (middlewares, DI) para sacarle provecho.

Casos de uso y aplicación

Se utilizan para desarrollar una gran variedad de soluciones empresariales, web y móviles debido a su alto rendimiento, escalabilidad y compatibilidad multiplataforma. Son ampliamente empleados en sistemas corporativos como plataformas de gestión de recursos humanos, contabilidad o inventarios, así como en portales académicos y hospitalarios donde se requiere seguridad, trazabilidad y conexión con bases de datos robustas como SQL Server. También se usan en sitios web dinámicos, tiendas en línea, sistemas de reservas y plataformas de banca digital, donde ASP.NET permite crear interfaces web modernas y APIs REST que conectan con aplicaciones móviles o servicios externos. Gracias a su integración con herramientas como Visual Studio, Azure y Docker, .NET es una de las tecnologías más comunes en el desarrollo de aplicaciones empresariales que exigen estabilidad, rendimiento y mantenimiento a largo plazo.

Microsoft SQL Server 2022

Definición

Motor de base de datos relacional para cargas OLTP y analíticas, con alta disponibilidad, seguridad avanzada y capacidades híbridas con Azure. La edición 2022 (16.x) enfatiza integración con analítica casi en tiempo real y gobierno de datos.

Historia y Evolución

- Azure Synapse Link para SQL Server 2022: replica cambios operacionales hacia Synapse Dedicated SQL Pool con impacto mínimo, habilitando analítica casi en tiempo real sin ETL complejo.
- Microsoft Purview: políticas de acceso y gobierno sobre instancias inscritas con Azure Arc.
- Intelligent Query Processing (IQP) ampliado: incluye Parameter Sensitive Plan (PSP) para mitigar ‘parameter sniffing’ cacheando múltiples planes según selectividad; también retroalimentaciones de cardinalidad y paralelismo.
- Query Store habilitado por defecto en bases nuevas: registro histórico de planes/ejecuciones que habilita optimizaciones del motor y diagnósticos más simples.
- Seguridad de red moderna: soporte de TLS 1.3 cuando se usa TDS 8.0 (con los requisitos de SO/driver), endureciendo el cifrado en tránsito.

Ventajas

- Mejoras de rendimiento ‘sin tocar código’ vía IQP.
- Integración nativa con servicios Azure para analítica y gobierno.
- Endurecimiento criptográfico (TLS 1.3 con TDS 8.0) y auditoría inmutable (Ledger) para escenarios regulados.

Desventajas

- Complejidad de operación de features híbridos (Arc, Purview, Synapse Link).
- Requiere entender compatibilidad de controladores/OS para TLS 1.3 y TDS 8.0.
- Licenciamiento empresarial y curva de aprendizaje para aprovechar IQP/Query Store.

Casos de uso y aplicación

Se aplica en una amplia gama de entornos donde se requiere gestionar grandes volúmenes de información con alta confiabilidad y seguridad. En el ámbito empresarial, es común encontrarlo en sistemas financieros, contables, de recursos humanos y de gestión de inventarios, donde se necesita mantener la integridad de los datos y realizar consultas complejas en tiempo real. También se usa en organizaciones de salud para almacenar historias clínicas electrónicas, resultados de laboratorio y registros médicos con control de acceso, auditoría y respaldo automatizado. En el sector educativo, permite administrar bases de datos con información de estudiantes, matrículas y calificaciones, garantizando disponibilidad y consistencia en todo momento.

Además, SQL Server es ampliamente usado en proyectos de análisis de datos y Business Intelligence, gracias a herramientas integradas como SQL Server Analysis Services (SSAS), Integration Services (SSIS) y Reporting Services (SSRS), que facilitan la creación de reportes, paneles de control y procesos ETL (extracción, transformación y carga). En la actualidad, su versión 2022 ha fortalecido la conexión con la nube mediante Azure Synapse Link y Microsoft Purview, lo que permite combinar datos locales con análisis en tiempo real y gobernanza centralizada. Por esto, SQL Server no solo funciona como base de datos transaccional, sino también como plataforma analítica y de integración, adaptable a escenarios modernos de inteligencia empresarial, aplicaciones web, móviles y sistemas distribuidos.

REST (Representational State Transfer)

Definición

REST es un estilo arquitectónico para sistemas hipermedia donde los recursos se identifican con URIs y se manipulan mediante representaciones (JSON/HTML/XML) sobre HTTP. Impone restricciones: cliente-servidor, sin estado (stateless), cacheable, interfaz uniforme (métodos, códigos, tipos de medios), sistema en capas y, opcionalmente, código bajo demanda.

Historia y Evolución

Formulado por Roy T. Fielding en su tesis (2000), modela la Web como sistema distribuido y ha guiado el diseño de APIs HTTP. Con el tiempo, las organizaciones estandarizaron contratos con OpenAPI para reducir ambigüedad y habilitar tooling.

Ventajas

- Permite una comunicación sencilla y universal entre sistemas al usar los métodos estándar de HTTP, lo que facilita la interoperabilidad entre distintas plataformas.
- Es escalable y eficiente, ya que al ser sin estado cada petición es independiente, permitiendo distribuir la carga y atender múltiples solicitudes simultáneamente.
- Favorece la integración de aplicaciones web, móviles o de terceros, promoviendo la reutilización de recursos y la conexión entre diferentes entornos.

Desventajas

- No existe una estandarización estricta en su diseño, lo que puede generar inconsistencias entre distintas implementaciones de APIs.
- No resulta ideal para operaciones que requieren transacciones o procesos coordinados, porque cada solicitud es independiente de las demás.
- Al no mantener sesiones, la autenticación y la gestión de permisos deben realizarse en cada petición, aumentando la complejidad y el riesgo de errores de seguridad.

Casos de uso y aplicación

APIs CRUD, integración B2B, backends móviles y microservicios. En el aula, es el protocolo natural para exponer controladores MVC de ASP.NET Core como endpoints JSON, documentados con OpenAPI.

Swagger / OpenAPI

Definición

OpenAPI es la especificación estándar (orientada a máquinas) para describir APIs HTTP. Swagger es un conjunto de herramientas de SmartBear y comunidad que trabaja con OpenAPI (p. ej., Swagger UI, Editor, Codegen, SwaggerHub). En .NET, librerías como Swashbuckle generan el documento OpenAPI desde los controladores/anotaciones.

Historia y Evolución

La especificación Swagger fue donada a la Linux Foundation (2015) y renombrada como OpenAPI. La versión 3.1 alineó completamente la compatibilidad con JSON Schema 2020-12, mejorando la validación y la reutilización de modelos.

Ventajas

- Genera documentación automática e interactiva de las APIs.
- Permite probar los endpoints fácilmente desde el navegador.
- Mejora la comunicación y estandarización entre equipos.

Desventajas

- Requiere mantener sincronía entre la API y la documentación.
- Tiene una curva de aprendizaje inicial para su configuración.
- Puede exponer información sensible si no se protege en producción.

Casos de uso y aplicación

Definir y versionar contratos REST; publicar documentación exploratoria (Swagger UI) y probar endpoints; gobernar el ciclo de vida de APIs. En ASP.NET Core, añadir Swashbuckle permite descubrir y probar controladores desde el navegador.

Docker

Definición

Docker es una plataforma de contenedores que empaqueta aplicaciones y sus dependencias en imágenes reproducibles para ejecutarlas como contenedores aislados a nivel de proceso. Un contenedor arranca rápido, es portable entre entornos (dev/QA/prod) y favorece despliegues consistentes.

Historia y Evolución

Desde 2013 popularizó el uso de cgroups y namespaces de Linux como base del aislamiento. Hoy es estándar de facto para CI/CD y despliegue, construyendo imágenes a partir de un Dockerfile y orquestando servicios multi-contenedor con Docker Compose.

Ventajas

- Facilita la portabilidad de aplicaciones entre distintos entornos.
- Reduce problemas de configuración al incluir dependencias en contenedores.
- Acelera el despliegue y la escalabilidad de los sistemas.

Desventajas

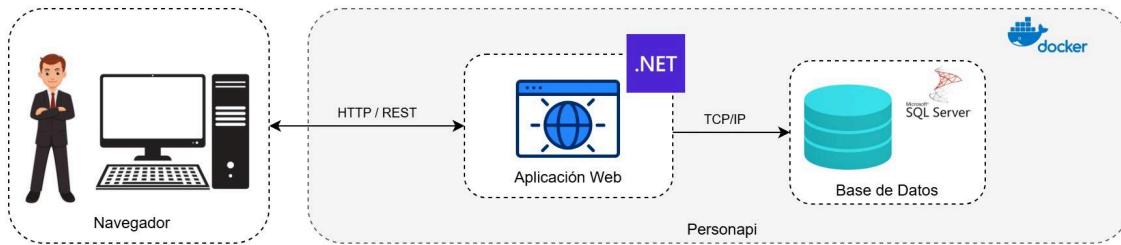
- Requiere conocimientos técnicos para gestionar redes y volúmenes.
- Puede complicar la seguridad si no se controlan las imágenes y accesos.
- Consumo recursos del sistema cuando se ejecutan muchos contenedores.

Casos de uso y aplicación

Empaquetar backends ASP.NET Core, ejecutar SQL Server para desarrollo, correr pruebas de integración y preparar despliegues a Kubernetes. En un laboratorio, un docker-compose.yml puede levantar simultáneamente la API y la base de datos para un monolito MVC con DAO.

Diseño

Diagrama Alto Nivel

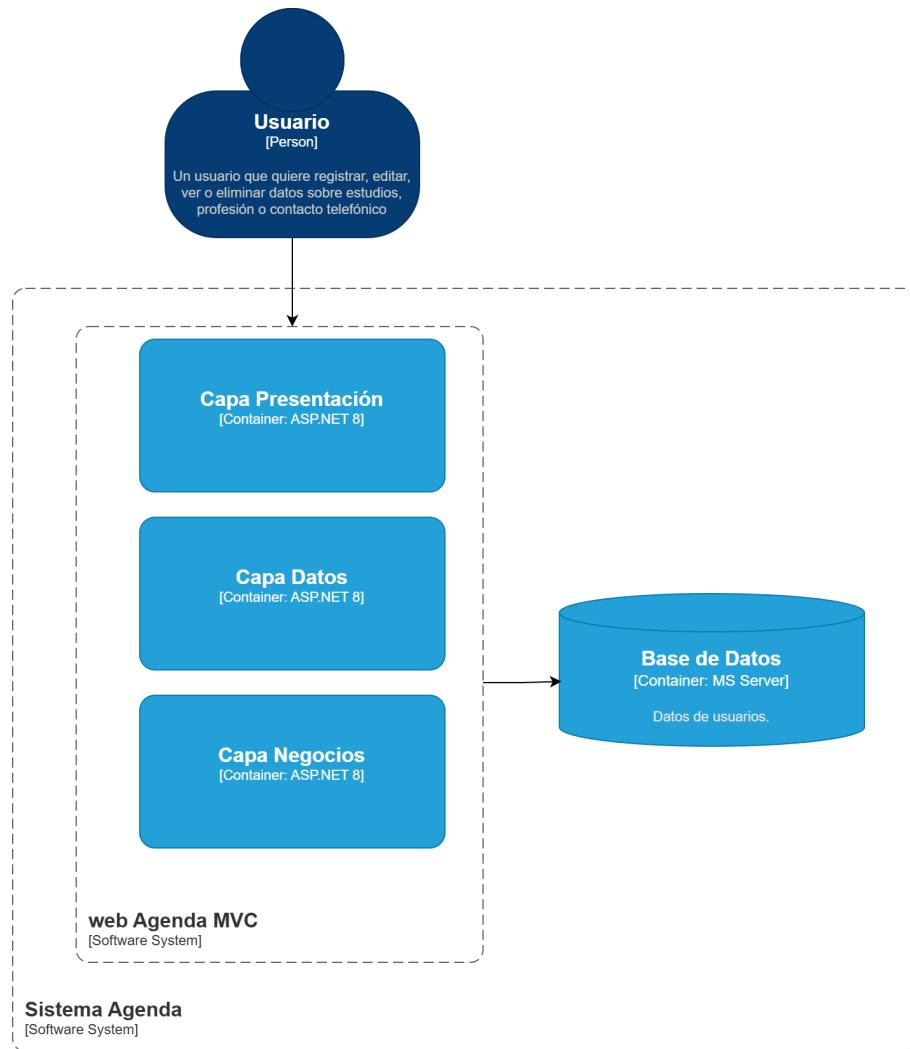


El usuario interactúa desde un navegador enviando peticiones mediante el protocolo HTTP/REST hacia la aplicación web, que actúa como el servidor encargado de procesar la lógica del negocio y devolver las respuestas correspondientes. Esta aplicación, a su vez, se comunica con la base de datos SQL Server a través del protocolo TCP/IP, estableciendo una conexión directa y eficiente para consultar, registrar o actualizar información. La base de datos se ejecuta dentro de un contenedor Docker, lo que garantiza un entorno aislado, portable y reproducible para la gestión de datos. En conjunto, el esquema ilustra un flujo completo desde el usuario hasta la capa de almacenamiento, mostrando cómo se integran las tecnologías web, de servidor y de base de datos en una estructura coherente y moderna.

Diagrama de Contexto

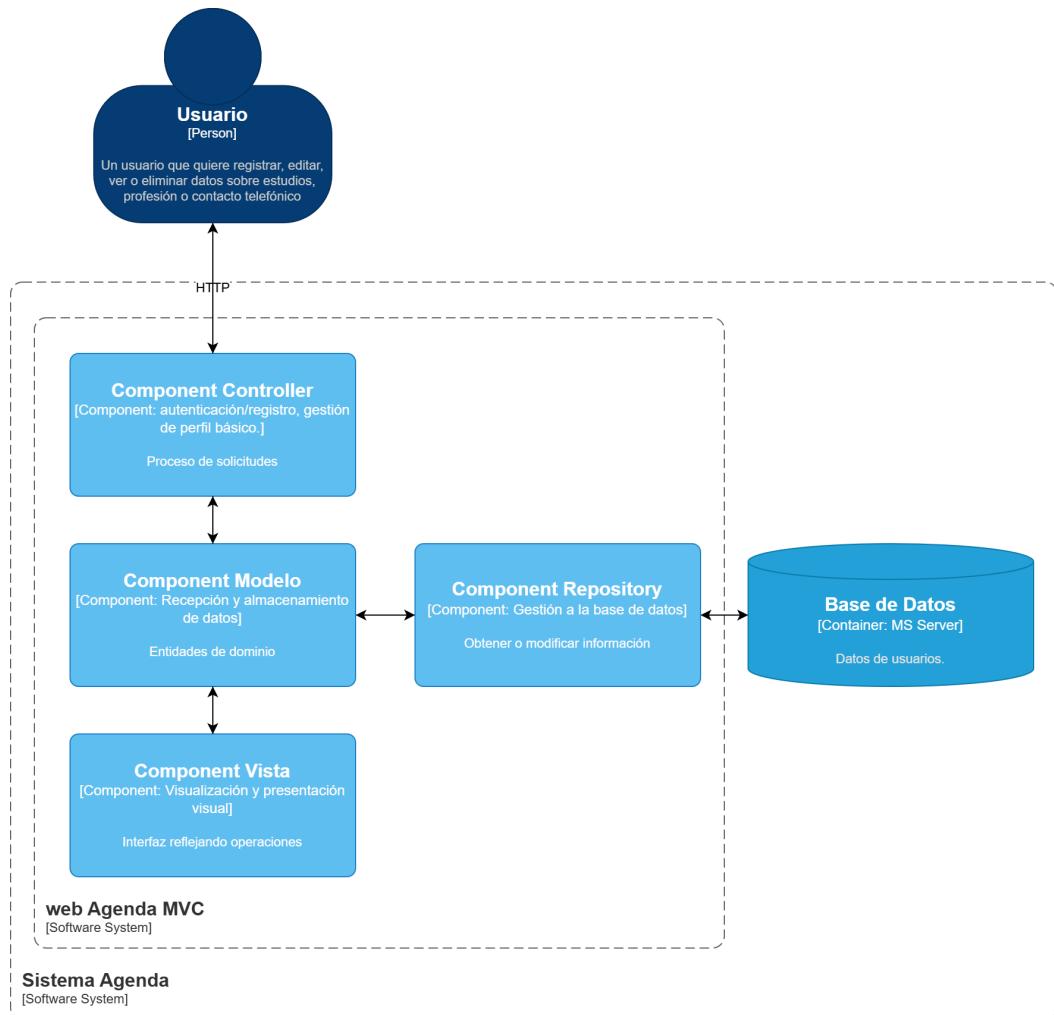


Diagrama de Contenedores



El diagrama de contenedores representa la estructura interna del Sistema Agenda, mostrando cómo se organizan sus componentes principales. El usuario interactúa con la aplicación web Agenda MVC, desarrollada en ASP.NET 8, que está dividida en tres capas: la capa de presentación, encargada de la interfaz y la interacción con el usuario; la capa de negocio, que contiene la lógica y las reglas del sistema; y la capa de datos, que gestiona la comunicación con la base de datos. Esta última se ejecuta en un contenedor de Microsoft SQL Server, donde se almacenan los datos personales, académicos y profesionales de los usuarios. En conjunto, el diagrama muestra cómo cada componente cumple una función específica dentro del sistema, garantizando una arquitectura organizada, modular y mantenible.

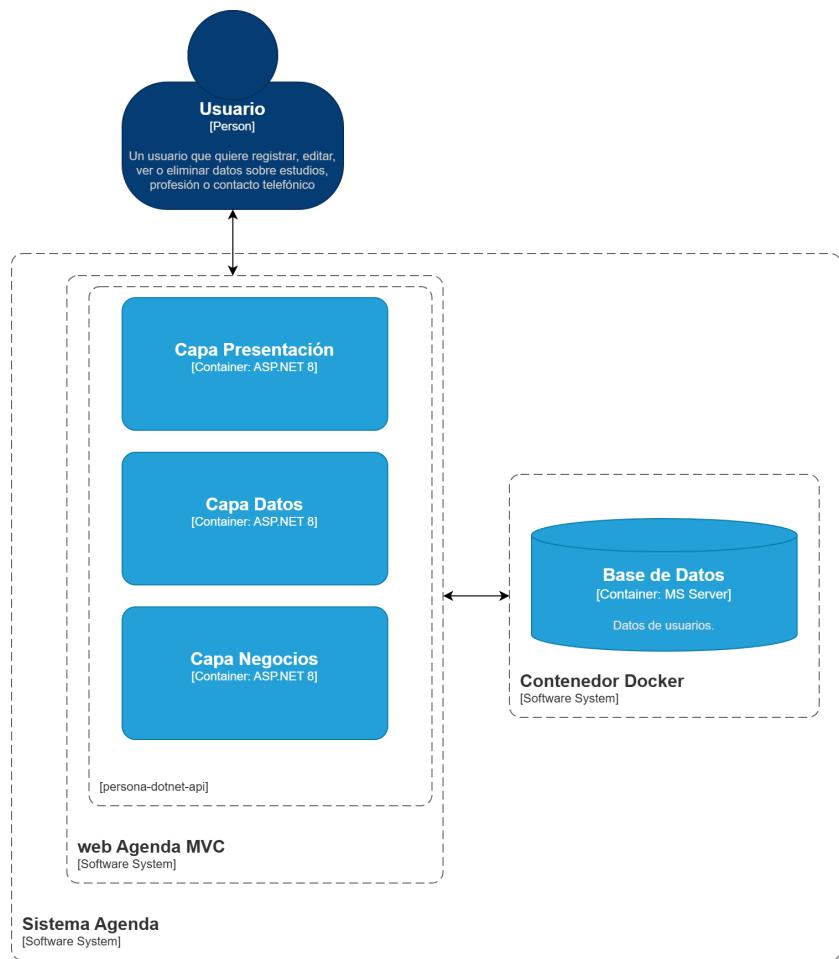
Diagrama de Componentes



El diagrama de componentes representa la estructura interna del sistema web Agenda MVC, mostrando cómo se conectan los elementos que permiten su funcionamiento. El usuario interactúa con el sistema mediante solicitudes HTTP, que son recibidas por el Controller, encargado de procesar las peticiones, validar los datos y coordinar las acciones necesarias, como autenticación, registro o edición de perfiles. Luego, el Modelo gestiona las entidades de dominio, es decir, los datos que representan la información del usuario, y los organiza para ser almacenados o modificados.

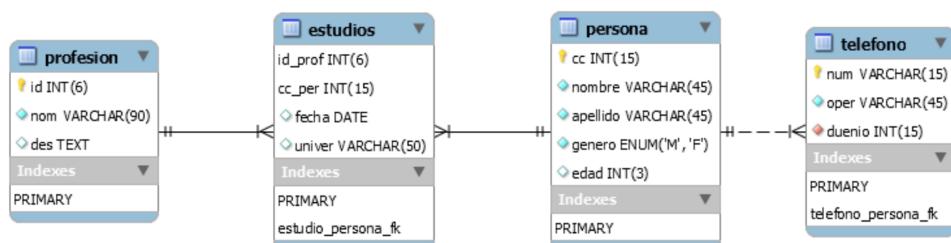
El Repositorio actúa como intermediario entre el modelo y la base de datos, encargándose de realizar las operaciones de lectura o escritura necesarias mediante consultas a SQL Server. Finalmente, la Vista muestra los resultados al usuario, reflejando los cambios o la información solicitada en una interfaz visual clara. En conjunto, el diagrama ilustra el flujo de datos dentro del patrón MVC: desde la interacción del usuario hasta la persistencia de los datos, destacando cómo cada componente cumple un rol específico para mantener una arquitectura ordenada, modular y fácil de mantener.

Diagrama de Despliegue



El diagrama de despliegue muestra cómo el sistema Agenda MVC se ejecuta en un entorno distribuido donde el usuario accede desde un navegador y se conecta a la aplicación desarrollada en ASP.NET 8, compuesta por tres capas: presentación, negocio y datos, que trabajan de forma integrada para procesar solicitudes, aplicar reglas lógicas y manejar la información. La base de datos se encuentra en un contenedor Docker que ejecuta Microsoft SQL Server, lo que garantiza un entorno aislado y estable para el almacenamiento de los datos. La comunicación entre la aplicación y la base de datos se realiza mediante una conexión segura, permitiendo un flujo constante de información y un despliegue flexible, portátil y de fácil mantenimiento.

Entidades



La **entidad profesión** corresponde a la tabla profesion, donde se registra la información sobre las distintas ocupaciones o áreas laborales que puede ejercer una persona. Cada profesión cuenta con un identificador único (**id**), un nombre (**nom**) y una descripción más detallada (**des**), que permiten clasificarla y diferenciarla dentro del sistema. Esta entidad está relacionada con la de **Estudios** a través de una relación de uno a muchos, ya que una misma profesión puede estar vinculada a varios estudios realizados por diferentes personas.

La **entidad estudios** se representa en la tabla estudios, encargada de almacenar los datos académicos o formativos de las personas. Entre sus principales atributos se encuentran **id_prof**, que enlaza cada estudio con la profesión correspondiente, y **cc_per**, que relaciona el registro con la persona que cursó dicho estudio. También incluye la **fecha** en que se realizó y el nombre de la **universidad** donde se llevó a cabo. Su estructura refleja una relación de muchos a uno con las entidades **Persona** y **Profesión**, indicando que cada estudio pertenece a un solo individuo y está asociado a una profesión específica.

La **entidad persona**, representada por la tabla persona, almacena los datos personales de los usuarios del sistema. Cada registro se identifica mediante la **cédula de ciudadanía (cc)**, que actúa como clave primaria. Además, contiene atributos como el **nombre**, **apellido**, **género** (masculino o femenino) y **edad**. Esta entidad se relaciona con **Estudios** y **Teléfono** mediante una relación de uno a muchos, lo que permite que una persona pueda tener varios estudios registrados y más de un número telefónico asociado.

Por último, la **entidad teléfono** se define en la tabla telefono, donde se guardan los números de contacto de las personas. Sus principales campos son el **número de teléfono (num)**, el **operador (oper)** y el **dueño (duenio)**, que es una clave foránea que vincula el número con la persona propietaria mediante su cédula. La relación entre **Teléfono** y **Persona** es de muchos a uno, ya que un mismo individuo puede tener varios teléfonos registrados, pero cada número pertenece únicamente a una persona.

Procedimiento

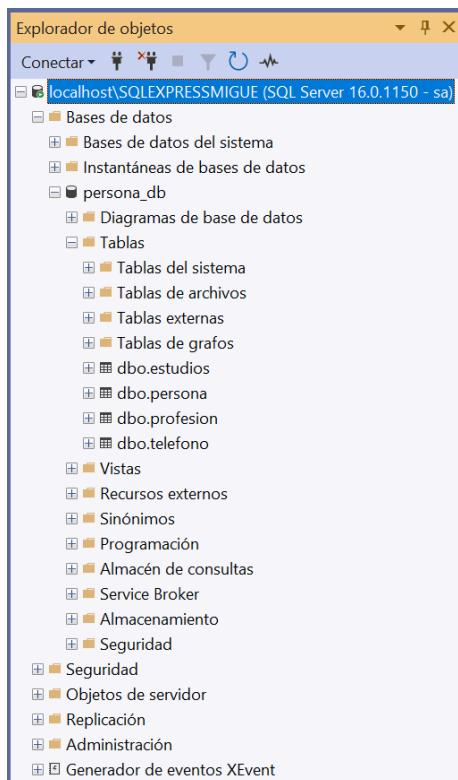
Creación del proyecto y repositorio

Primero se crea un repositorio público en GitHub llamado personapi-dotnet, que servirá para guardar y versionar todo el código del proyecto. Luego, el repositorio se clona de forma local para poder trabajar desde el equipo en Visual Studio. Dentro de este entorno, se crea un nuevo proyecto con la plantilla Aplicación web ASP.NET Core (Modelo-Vista-Controlador), asegurándose de usar .NET (en este caso estaba disponible para utilizar el 8.0), sin autenticación ni configuración HTTPS. El nombre del proyecto debe coincidir con el del repositorio para mantener coherencia y facilidad al momento de hacer push y tag.

The screenshot shows a GitHub repository page for 'personapi-dotnet'. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation is a search bar and a button to add a file. The main content area shows a list of commits from 'adrianrruiz' with details like commit message, file, time, and author. A 'README' file is shown below the commits. To the right, there's an 'About' section with a note about no description, website, or topics provided, and sections for Activity, Stars, Forks, Releases (with one release titled 'FINALIZADO - 03/11/2025'), and Packages.

Base de datos

Se instala SQL Server 2022 Express (modo básico) y SQL Server Management Studio para gestionar la base de datos. Luego se crea una base llamada persona_db, asignando permisos al usuario sa. Con base en el modelo proporcionado, se crean las tablas necesarias (Persona, Teléfono, Profesión, Estudios) respetando las claves primarias y foráneas. Esta estructura será el punto de partida para generar las entidades dentro del proyecto [ASP.NET](#).

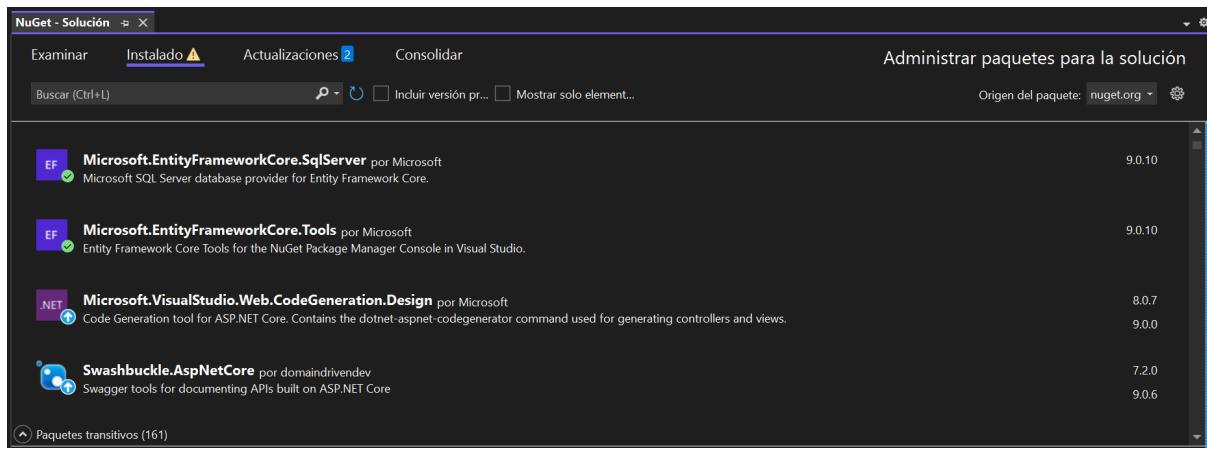
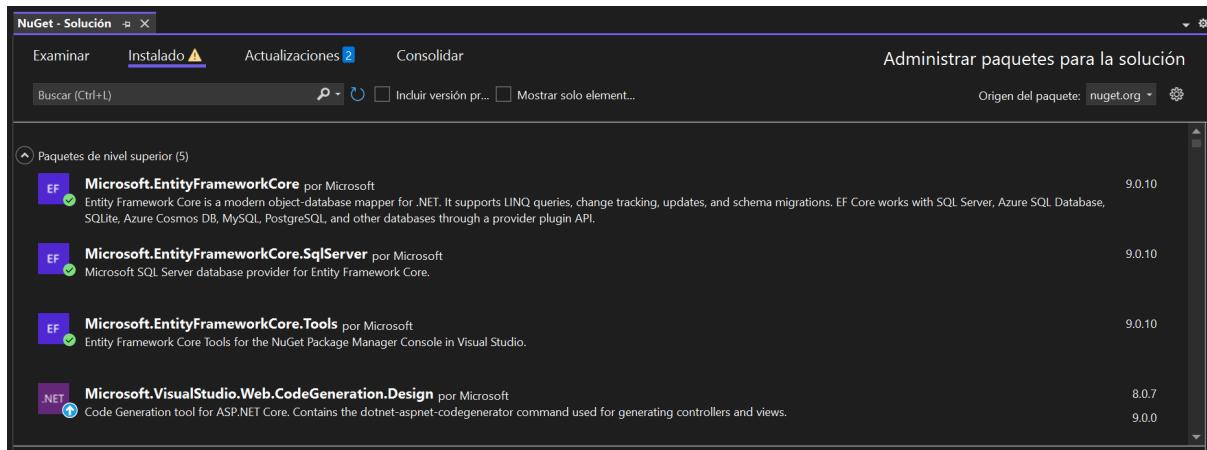


Instalación de dependencias

Dentro de Visual Studio, se abren las opciones de Administrador de paquetes NuGet y se instalan las bibliotecas esenciales de Entity Framework Core:

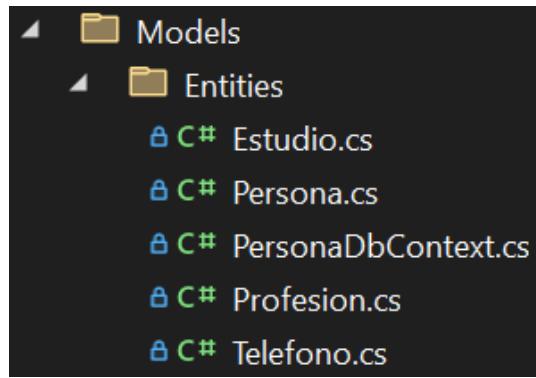
- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools

Estas dependencias permiten conectar la aplicación con SQL Server y manejar los datos mediante un modelo de entidades. También se activa la vista Explorador de objetos de SQL Server para probar la conexión con la base de datos local.



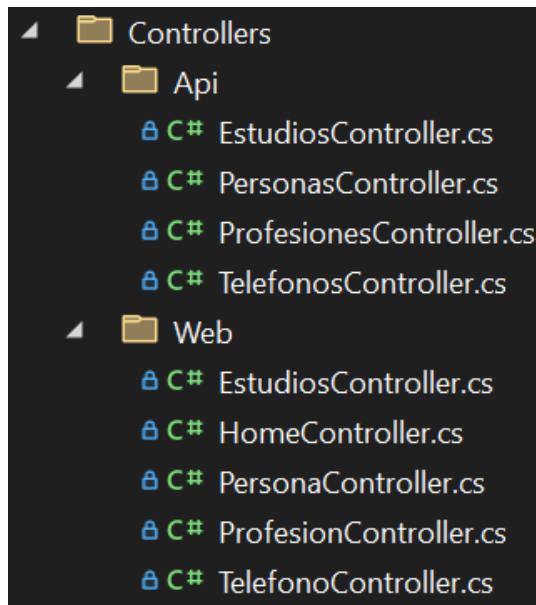
Entidades

En la carpeta *Models*, se crea una subcarpeta llamada *Entities*, que contendrá las clases generadas a partir de las tablas de la base de datos. Desde la Consola del Administrador de paquetes, se ejecuta el comando *Scaffold-DbContext*, el cual genera automáticamente las entidades y el contexto de conexión a partir del esquema existente en SQL Server. Estas clases representan la estructura del modelo de datos que se utilizará dentro del sistema.



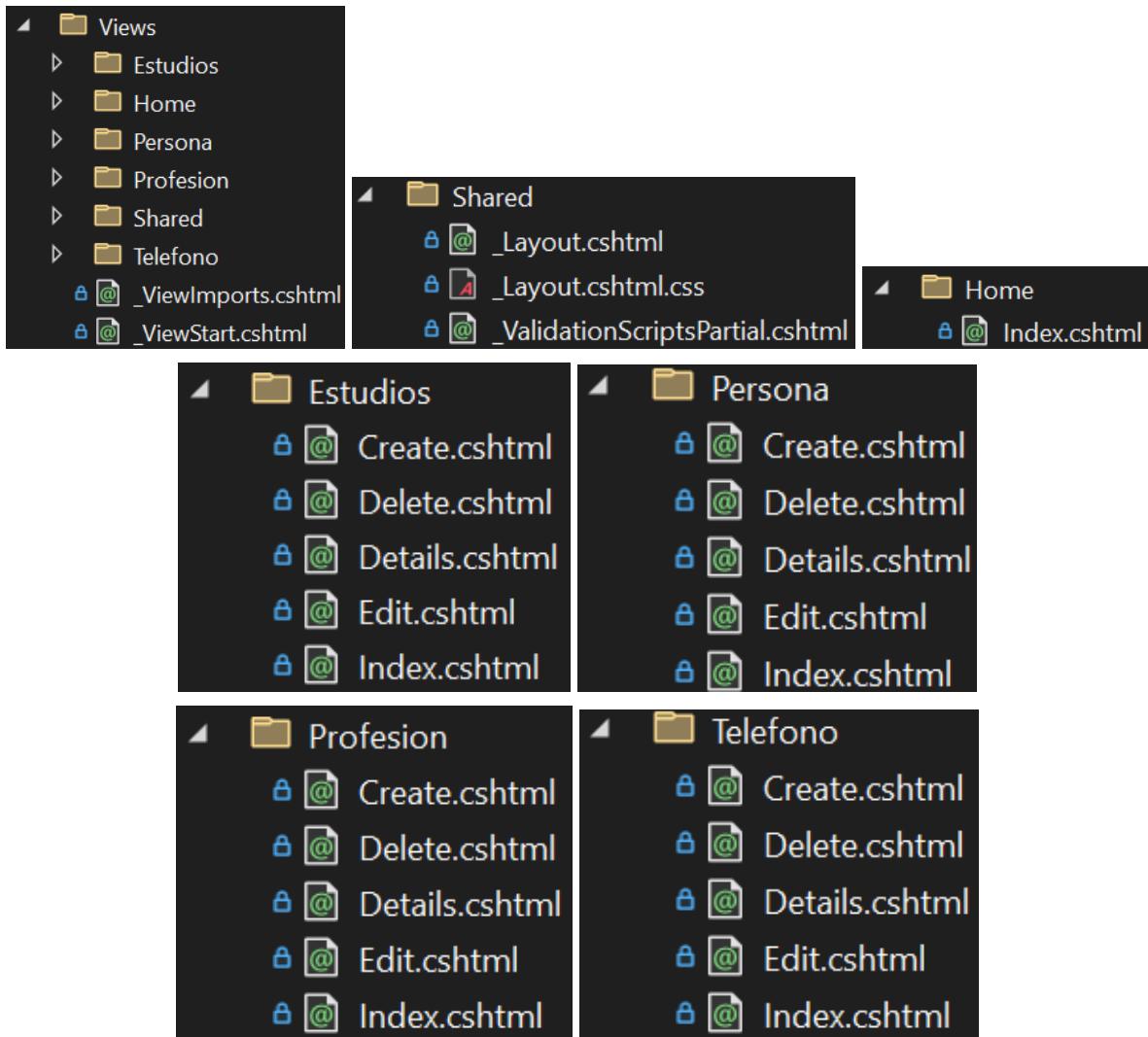
Controladores y API Restful

Una vez definidas las entidades, se crean los controladores, que son los encargados de manejar las peticiones del usuario y coordinar la lógica del sistema. Cada controlador se asocia a una entidad principal (por ejemplo, PersonaController o ProfesionController) y expone las rutas necesarias para realizar las operaciones CRUD (crear, leer, actualizar, eliminar). A partir de los controladores, se define la API RESTful, estableciendo los endpoints que permitirán acceder a los datos de cada entidad mediante solicitudes HTTP. Estas rutas se estructuran con métodos como GET, POST, PUT y DELETE, asegurando una comunicación estándar entre el cliente y el servidor. En esta etapa también puede integrarse Swagger para documentar y probar la API de manera visual e interactiva.



Interfaz

Aunque el enfoque principal es la API, el proyecto MVC permite visualizar los datos en una interfaz web básica. Las vistas muestran la información de las tablas y permiten probar la interacción entre el usuario y los controladores de forma sencilla. Esta capa hace que la aplicación sea más intuitiva y funcional para quienes la usan desde el navegador.

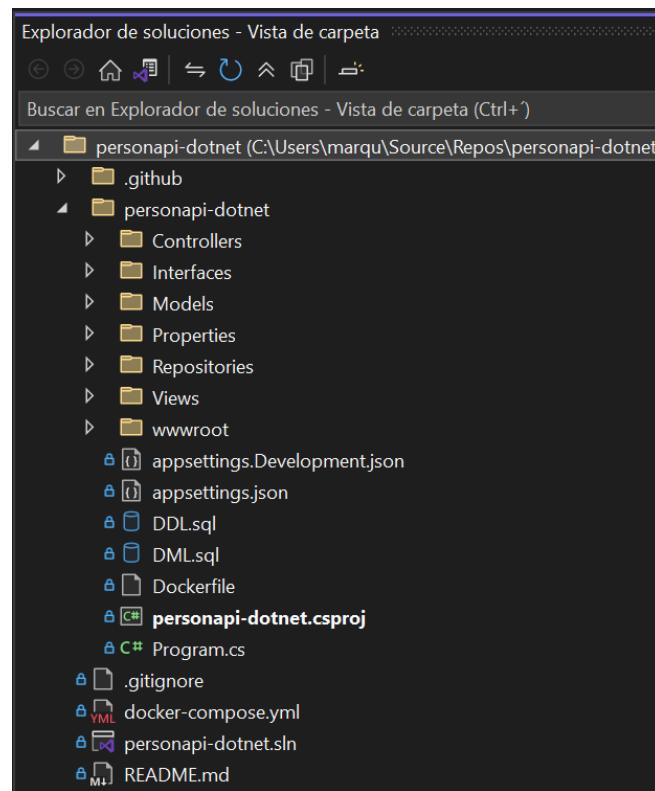


Swagger

```
// Add Swagger
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo
    {
        Title = "Persona API",
        Version = "v1",
        Description = "API REST para gestión de personas, profesiones, estudios y teléfonos"
    });
});
```

Uso de Docker

Finalmente, se configura Docker para contener la base de datos o incluso toda la aplicación, asegurando que el entorno de ejecución sea uniforme en cualquier equipo. Se crea un archivo docker-compose.yml que define los servicios necesarios (por ejemplo, el contenedor de SQL Server). Esto facilita el despliegue, evita conflictos de configuración y permite probar el sistema completo de manera aislada.



```
services:  
  sqlserver:  
    image: mcr.microsoft.com/mssql/server:2022-latest  
    container_name: sqlserver  
    environment:  
      - ACCEPT_EULA=Y  
      - SA_PASSWORD=Arqui2025!  
    ports:  
      - "1433:1433"  
    volumes:  
      - sql_data:/var/opt/mssql  
      - ./scripts:/docker-entrypoint-initdb.d  
    restart: unless-stopped  
  
  personapi:  
    build:  
      context: ./personapi-dotnet  
      dockerfile: Dockerfile  
    container_name: personapi_dotnet  
    ports:  
      - "8080:8080"  
    environment:  
      - ASPNETCORE_ENVIRONMENT=Development  
    depends_on:  
      - sqlserver  
    restart: unless-stopped
```

```

12      | - ./scripts:/docker-entrypoint-initdb.d
13      | restart: unless-stopped
14
15  personapi:
16    build:
17      context: ./personapi-dotnet
18      dockerfile: Dockerfile
19      container_name: personapi_dotnet
20
21    ports:
22      - "8080:8080"
23      environment:
24        - ASPNETCORE_ENVIRONMENT=Development
25    depends_on:
26      - sqlserver
27      restart: unless-stopped
28
29  ngrok-personapi:
30    image: ngrok/ngrok:latest
31    restart: unless-stopped
32    command: http --url=smolderingly-unlarge-mariann.ngrok-free.dev personapi:8080 --log=stdout
33    environment:
34      - NGROK_AUTHTOKEN=${NGROK_AUTHTOKEN}
35    depends_on:
36      - personapi
37
38  volumes:
39    - sql_data:

```

No se encontraron problemas.

Línea: 1 | Carácter: 1 | SPC | CRLF

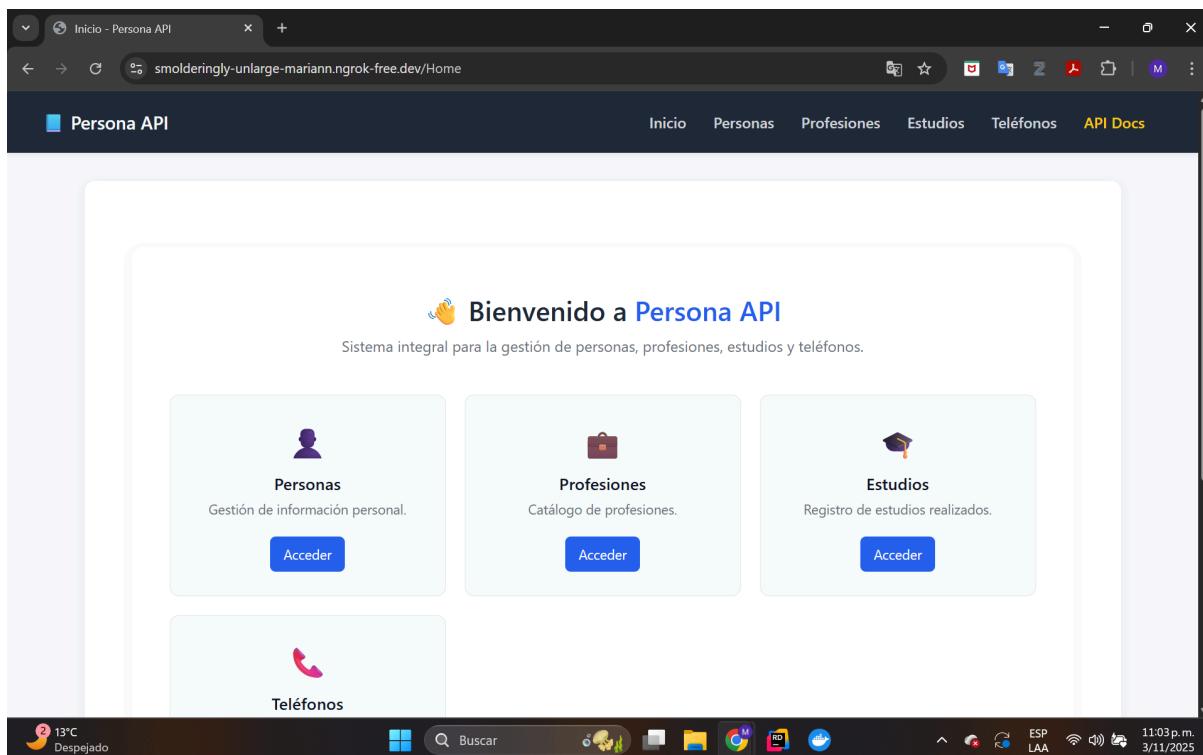
Al finalizar, se realiza el push de los cambios al repositorio remoto y se crea un TAG en GitHub para marcar la versión final del laboratorio.

TAG: <https://github.com/MiguelMarquezPosso/personapi-dotnet/releases/tag/FINISHED>

Despliegue

Link: <https://smolderingly-unlarge-mariann.ngrok-free.dev/Home>

A continuación, se pueden observar imágenes sobre la interfaz con todas sus funcionalidades:



Personas
Gestión de información personal.
[Acceder](#)

Profesiones
Catálogo de profesiones.
[Acceder](#)

Estudios
Registro de estudios realizados.
[Acceder](#)

Teléfonos
Gestión de números telefónicos.
[Acceder](#)

[Ver Documentación API](#)

Personas

[Create New](#)

Cc	Nombre	Apellido	Genero	Edad	
1010	Juan	Pérez	M	28	Edit Details Delete
1020	Laura	Gómez	F	32	Edit Details Delete
1030	Andrés	Torres	M	24	Edit Details Delete
1040	María	Rodríguez	F	45	Edit Details Delete

Create - Persona API

smolderingly-unlarge-mariann.ngrok-free.dev/Persona/Create

Persona API

Inicio Personas Profesiones Estudios Teléfonos API Docs

Create

Persona

Cédula

Nombre

Apellido

Género

Edad

[Create](#)

[Back to List](#)

13°C Despejado Buscar ESP LAA 11:04 p.m. 3/11/2025

Edit - Persona API

smolderingly-unlarge-mariann.ngrok-free.dev/Persona/Edit/1010

Persona API

Inicio Personas Profesiones Estudios Teléfonos API Docs

Edit

Persona

Nombre

Apellido

Género

Edad

[Save](#)

[Back to List](#)

© 2025 - Persona API · GitHub

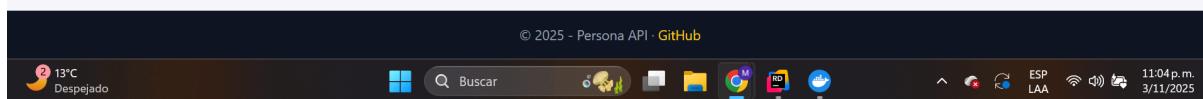
13°C Despejado Buscar ESP LAA 11:04 p.m. 3/11/2025

Details

Persona

Cc	1010
Nombre	Juan
Apellido	Pérez
Genero	M
Edad	28

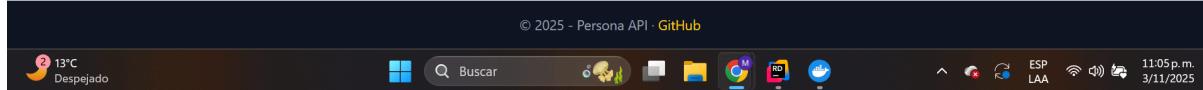
[Edit](#) | [Back to List](#)



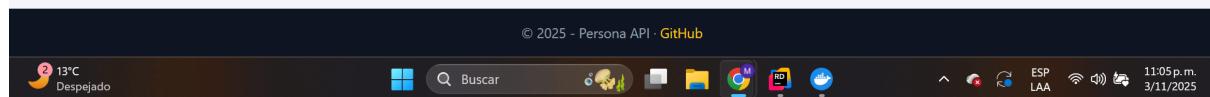
Profesiones

[Create New](#)

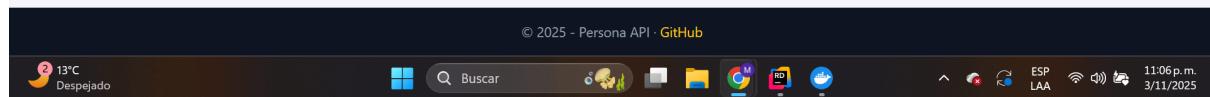
Id	Nombre	Descripción	
1	Ingeniero de Sistemas	Diseña y mantiene sistemas informáticos.	Edit Details Delete
2	Médico	Atiende y diagnostica pacientes.	Edit Details Delete
3	Arquitecto	Diseña planos y estructuras de edificaciones.	Edit Details Delete
4	Docente	Imparte clases en instituciones educativas.	Edit Details Delete



The screenshot shows a web browser window with the title "Create - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Profesion/Create". The page has a dark header with the "Persona API" logo and navigation links for "Inicio", "Personas", "Profesiones", "Estudios", "Teléfonos", and "API Docs". The main content area is titled "Create" and "Profesión". It contains three input fields: "Id" (empty), "Nombre" (empty), and "Descripción" (empty). Below the fields is a blue "Create" button and a link "Back to List".



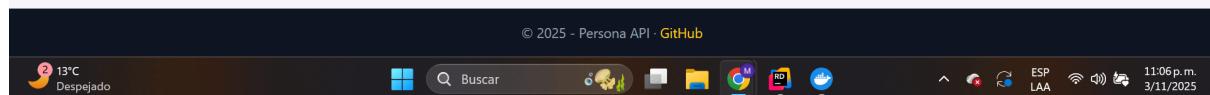
The screenshot shows a web browser window with the title "Edit - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Profesion/Edit/1". The page has a dark header with the "Persona API" logo and navigation links for "Inicio", "Personas", "Profesiones", "Estudios", "Teléfonos", and "API Docs". The main content area is titled "Edit" and "Profesión". It contains two input fields: "Nombre" (containing "Ingeniero de Sistemas") and "Descripción" (containing "Diseña y mantiene sistemas informáticos"). Below the fields is a blue "Save" button and a link "Back to List".



The screenshot shows a web browser window with the title "Details - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Profesion/Details/1". The page has a dark header with the "Persona API" logo and navigation links for Inicio, Personas, Profesiones, Estudios, Teléfonos, and API Docs. The main content area has a light gray background and features a large heading "Details" followed by "Profesión". Below this, there are two data rows:

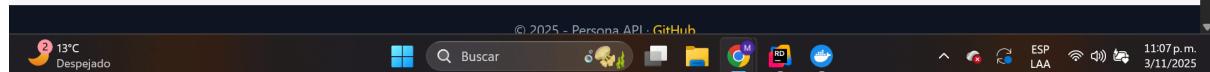
Nombre	Ingeniero de Sistemas
Descripción	Diseña y mantiene sistemas informáticos.

At the bottom of the content area are two blue links: "Edit" and "Back to List".



The screenshot shows a web browser window with the title "Estudios - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Estudios". The page has a dark header with the "Persona API" logo and navigation links for Inicio, Personas, Profesiones, Estudios, Teléfonos, and API Docs. The main content area has a light gray background and features a section titled "Estudios" with a graduation cap icon. Below this is a blue button labeled "Crear Nuevo Estudio". A table displays four rows of study data:

Profesión ID	Persona CC	Fecha	Universidad	Persona	Profesión	Acciones
1	1010	06/15/2020 00:00:00	Pontificia Universidad Javeriana	Juan Pérez	Ingeniero de Sistemas	<button>Editar</button> <button>Detalles</button> <button>Eliminar</button>
2	1020	12/20/2018 00:00:00	Universidad del Rosario	Laura Gómez	Médico	<button>Editar</button> <button>Detalles</button> <button>Eliminar</button>
3	1030	11/05/2022 00:00:00	Universidad Nacional de Colombia	Andrés Torres	Arquitecto	<button>Editar</button> <button>Detalles</button> <button>Eliminar</button>
4	1040	09/10/2010 00:00:00	Universidad de Antioquia	María Rodríguez	Docente	<button>Editar</button> <button>Detalles</button> <button>Eliminar</button>



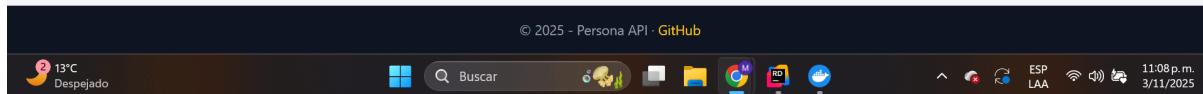
The screenshot shows a web browser window titled "Create - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Estudios/Create". The page has a dark header with the "Persona API" logo and navigation links for "Inicio", "Personas", "Profesiones", "Estudios", "Teléfonos", and "API Docs". The main content area is titled "Create" and "Estudio". It contains four input fields: "Profesión" (Ingeniero de Sistemas), "Persona" (Juan), "Fecha" (dd/mm/aaaa --:-- ----), and "Universidad" (empty). Below the fields are a "Create" button and a "Back to List" link.

The screenshot shows a web browser window titled "Edit - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Estudios/Edit?idProf=1&ccPer=1010". The page has a dark header with the "Persona API" logo and navigation links for "Inicio", "Personas", "Profesiones", "Estudios", "Teléfonos", and "API Docs". The main content area is titled "Edit" and "Estudio". It contains two input fields: "Fecha" (15/06/2020 12:00 a.m.) and "Universidad" (Pontificia Universidad Javeriana). Below the fields are a "Save" button and a "Back to List" link.

The screenshot shows a web browser window with the title "Details - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Estudios/Details?idProf=1&ccPer=1010". The page has a dark header with the "Persona API" logo and navigation links for Inicio, Personas, Profesiones, Estudios, Teléfonos, and API Docs. The main content area has a light gray background and features a large heading "Details" followed by "Estudio". Below this, there is a table with the following data:

Fecha	06/15/2020 00:00:00
Universidad	Pontificia Universidad Javeriana
Persona	Juan
Cédula	1010
Profesión	Ingeniero de Sistemas

At the bottom of the table, there are two links: "Edit" and "Back to List".

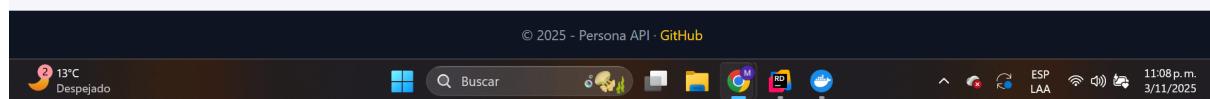


The screenshot shows a web browser window with the title "Index - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Teléfono". The page has a dark header with the "Persona API" logo and navigation links for Inicio, Personas, Profesiones, Estudios, Teléfonos, and API Docs. The main content area has a light gray background and features a large heading "Teléfonos" with a phone icon. Below this, there is a "Create New" link. A table lists telephone numbers along with their owners' names and IDs:

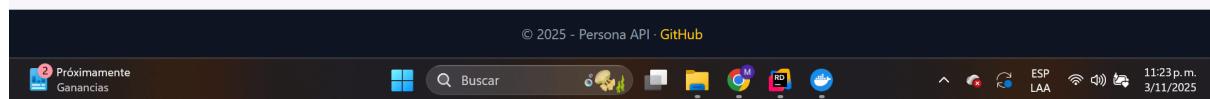
Operador	Número	Nombre del dueño	Cédula del dueño	Actions
Claro	3001112233	Juan	1010	Edit Details Delete
Wom	3014445566	Maria	1040	Edit Details Delete
Tigo	3102223344	Laura	1020	Edit Details Delete
Movistar	3203334455	Andrés	1030	Edit Details Delete



The screenshot shows a web browser window with the title "Create - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Teléfono/Create". The page has a dark header with the "Persona API" logo and navigation links for "Inicio", "Personas", "Profesiones", "Estudios", "Teléfonos", and "API Docs". The main content area is titled "Create" and "Teléfono". It contains three input fields: "Número" (Number), "Operador" (Operator), and "Dueño" (Owner), each with a placeholder value. Below the fields is a blue "Create" button and a link to "Back to List".



The screenshot shows a web browser window with the title "Edit - Persona API". The URL in the address bar is "smolderingly-unlarge-mariann.ngrok-free.dev/Teléfono/Edit/3001112233". The page has a dark header with the "Persona API" logo and navigation links for "Inicio", "Personas", "Profesiones", "Estudios", "Teléfonos", and "API Docs". The main content area is titled "Edit" and "Teléfono". It contains two input fields: "Operador" (Operator) with the value "Claro" and "Dueño" (Owner) with the value "Juan". Below the fields is a blue "Save" button and a link to "Back to List".



Details - Persona API

smolderingly-unlarge-mariann.ngrok-free.dev/Telefono/Details/3001112233

Persona API

Inicio Personas Profesiones Estudios Teléfonos API Docs

Details

Teléfono

Operador	Claro
Número	3001112233
Cédula del dueño	1010
Nombre del dueño	Juan

[Edit](#) | [Back to List](#)

© 2025 - Persona API · GitHub

11°C Despejado Buscar ESP LAA 11:24 p.m. 3/11/2025

Index - Persona API

smolderingly-unlarge-mariann.ngrok-free.dev/swagger/index.html

Swagger Supported by SMARTBEAR

Select a definition Persona API V1

Persona API v1 OAS 3.0

/swagger/v1/swagger.json

API REST para gestión de personas, profesiones, estudios y teléfonos

Estudios

GET	/api/Estudios
POST	/api/Estudios
GET	/api/Estudios/{idProf}/{ccPer}
PUT	/api/Estudios/{idProf}/{ccPer}
DELETE	/api/Estudios/{idProf}/{ccPer}
GET	/api/Estudios/count

Personas

11°C Despejado Buscar ESP LAA 11:24 p.m. 3/11/2025

The screenshot shows the Swagger UI interface for the Personas API. The top navigation bar has tabs for 'Index - Persona API' and 'Swagger UI'. The main content area is titled 'Personas' and lists the following endpoints:

- GET /api/Personas**
- POST /api/Personas**
- GET /api/Personas/{cc}**
- PUT /api/Personas/{cc}**
- DELETE /api/Personas/{cc}**
- GET /api/Personas/count**

Below this, there is a section titled 'Profesiones' which lists similar endpoints:

- GET /api/Profesiones**
- POST /api/Profesiones**
- GET /api/Profesiones/{id}**
- PUT /api/Profesiones/{id}**
- DELETE /api/Profesiones/{id}**

The bottom status bar shows system information: 11°C, Despejado, Buscar, ESP LAA, 11:25 p.m., 3/11/2025.

The screenshot shows the Swagger UI interface for the Profesiones API. The top navigation bar has tabs for 'Index - Persona API' and 'Swagger UI'. The main content area is titled 'Profesiones' and lists the following endpoints:

- GET /api/Profesiones**
- POST /api/Profesiones**
- GET /api/Profesiones/{id}**
- PUT /api/Profesiones/{id}**
- DELETE /api/Profesiones/{id}**
- GET /api/Profesiones/count**

Below this, there is a section titled 'Telefonos' which lists the following endpoints:

- GET /api/Telefonos**
- POST /api/Telefonos**
- GET /api/Telefonos/{num}**
- PUT /api/Telefonos/{num}**
- DELETE /api/Telefonos/{num}**

The bottom status bar shows system information: 11°C, Despejado, Buscar, ESP LAA, 11:25 p.m., 3/11/2025.

The screenshot shows the Swagger UI interface for a Persona API. The main title is "Telefonos". Below it, there are seven API endpoints listed:

- GET /api/Telefonos
- POST /api/Telefonos
- GET /api/Telefonos/{num}
- PUT /api/Telefonos/{num}
- DELETE /api/Telefonos/{num}
- GET /api/Telefonos/count

Below the endpoints is a "Schemas" section containing three items: Estudio, Persona, and Profesion. At the bottom of the screen is a Windows taskbar with various icons and status indicators.

This screenshot is identical to the one above, showing the same API endpoints and schema definitions. The main difference is the expanded "Schemas" section, which now includes an additional item: Telefono.

Conclusiones y lecciones aprendidas

- La implementación del patrón MVC permitió mantener una estructura clara y modular, separando la lógica, la presentación y los datos. Esta organización facilitó el desarrollo, la comprensión del sistema y dejó la aplicación preparada para futuras mejoras o expansiones.
- El uso del patrón DAO ayudó a separar la capa de acceso a datos de la lógica de negocio, brindando flexibilidad para modificar o mejorar la persistencia sin afectar el resto del sistema. Esto aumentó la mantenibilidad y redujo el riesgo de errores en futuras actualizaciones.
- La integración de Swagger 3 para la documentación y prueba de las APIs REST permitió verificar de forma ágil el funcionamiento de los endpoints y ofreció una interfaz visual clara que mejoró la comprensión de las funcionalidades y la colaboración entre desarrolladores.
- La combinación del stack tecnológico seleccionado (ASP.NET, SQL Server, Docker y Swagger) permitió cumplir con los requerimientos del proyecto y garantizó una base sólida para el desarrollo, las pruebas y la documentación, asegurando escalabilidad y estabilidad.
- Una de las principales conclusiones del proyecto es que la correcta aplicación de patrones de diseño y una buena organización del código desde el inicio son esenciales para asegurar la calidad del software. Esto permite que el sistema evolucione fácilmente y se adapte a nuevas necesidades sin comprometer su funcionamiento ni su estructura.
- La implementación del proyecto demostró que el uso de arquitecturas bien definidas mejora la calidad del desarrollo, ya que cada componente cumple una función específica y reduce la dependencia entre módulos.
- Se comprobó que mantener una separación lógica entre capas facilita la detección y corrección de errores, haciendo que el mantenimiento y la depuración del sistema sean procesos mucho más ágiles.
- El empleo de herramientas modernas como Docker y Swagger favoreció la estandarización del entorno de desarrollo y la verificación del correcto funcionamiento de los servicios, evitando inconsistencias entre entornos locales y de producción.
- Finalmente, se concluye que el aprendizaje obtenido en el manejo de patrones, frameworks y herramientas de despliegue constituye una base sólida para proyectos futuros, fortaleciendo las competencias técnicas y el pensamiento estructurado en el desarrollo de software profesional.

Referencias

- Microsoft. Overview of ASP.NET Core. <https://learn.microsoft.com/en-us/aspnet/core/overview>
- .NET – What is .NET? <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>
- .NET: Build. test. deploy. (no date) Microsoft. <https://dotnet.microsoft.com/en-us/>
- Microsoft. .NET support policy (LTS/STS). <https://dotnet.microsoft.com/en-us/platform/support/policy>
- Reenskaug, T. The original MVC reports (Xerox PARC). <https://masters.donntu.ru/2013/fknt/korienev/library/article5.pdf>
- Home (2024) Docker Documentation. <https://docs.docker.com/>