

# Laboratorio 1 - Implementación de Monolito con Patrón MVC y DAO



Juan Manuel Durán Rueda

Nicolas Alexander Sanchez Laiton

David Alfonso Anteliz Bonilla

Pontificia universidad javeriana

Arquitectura de software

31/10/2023

## Marco conceptual

### Arquitectura Monolítica:

Una arquitectura monolítica es un enfoque de diseño de software en el que una aplicación se desarrolla como una única unidad, donde todas las funcionalidades y componentes están integrados en una sola aplicación. En este enfoque, la aplicación se compila en un solo código base y se ejecuta en un solo proceso.

### Patrón MVC (Modelo-Vista-Controlador):

MVC es un patrón de diseño arquitectónico que divide una aplicación en tres componentes principales: Modelo, Vista y Controlador. El Modelo se encarga de la lógica de negocio y los datos, la Vista se encarga de la presentación y la interfaz de usuario, y el Controlador actúa como intermediario que maneja las solicitudes del usuario y coordina la comunicación entre el Modelo y la Vista.

### DAO (Data Access Object):

DAO es un patrón de diseño que se utiliza para abstraer la capa de acceso a datos de una aplicación. Proporciona una interfaz para acceder a una fuente de datos, como una base de datos, y oculta los detalles de la implementación subyacente.

### Tecnologías Utilizadas:

**.NET 7:** .NET es un marco de desarrollo de software ampliamente utilizado para desarrollar aplicaciones en una variedad de plataformas.

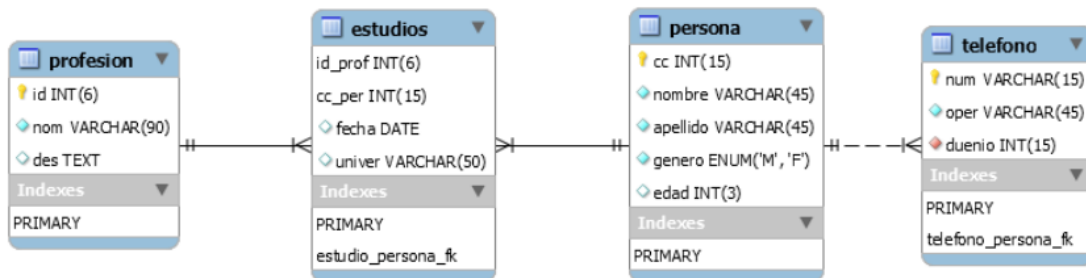
**MS SQL Server 2022:** MS SQL Server es un sistema de gestión de bases de datos relacionales desarrollado por Microsoft.

**REST (Representational State Transfer):** REST es un estilo arquitectónico para el diseño de servicios web que utiliza operaciones HTTP estándar (GET, POST, PUT, DELETE) para interactuar con recursos.

**Swagger 3:** Swagger es una herramienta que facilita la documentación y el desarrollo de APIs RESTful.

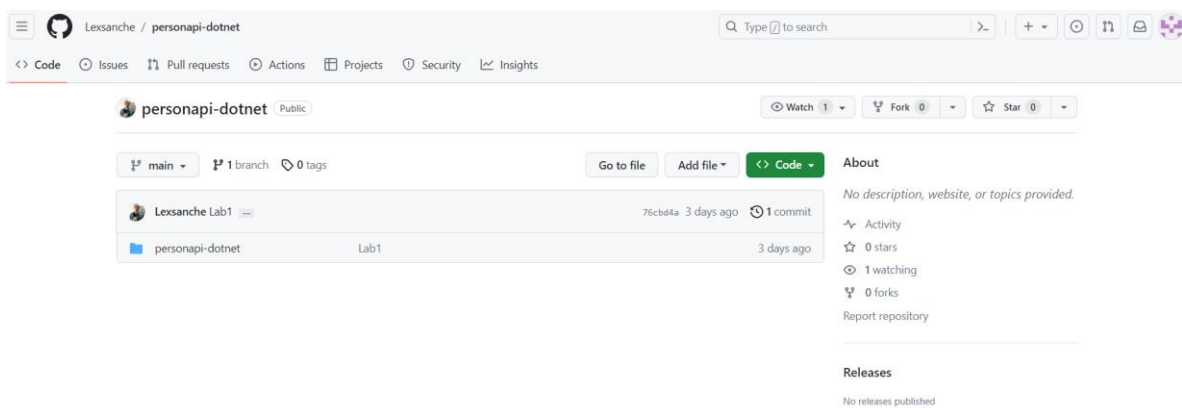
## Diseño

Para el diseño de este taller desarrollamos un CRUD para un modelo de datos basado en 4 tablas, las cuales se encuentran distribuidas de la siguiente manera:



## Procedimiento

Empezamos el procedimiento para la elaboración de este taller creando el repositorio personapi-dotnet.



Como segundo paso vamos a la página: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads> donde podremos descargar SQL Server 2022, como observamos en la imagen, descargaremos la versión express.

## Or, download a free specialized edition



### Developer

SQL Server 2022 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment.

[Download now](#)

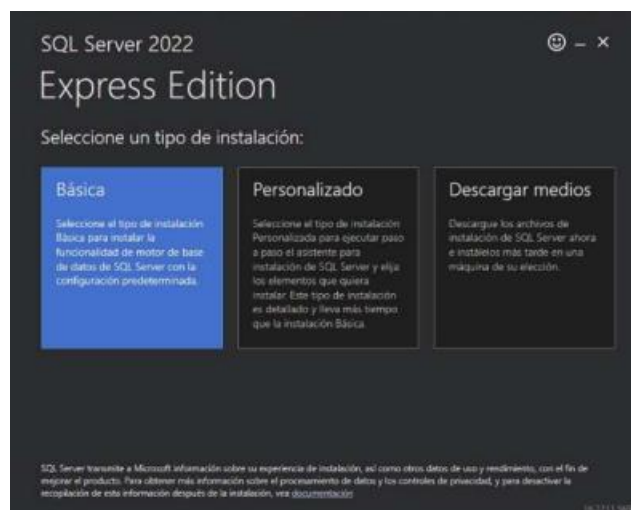


### Express

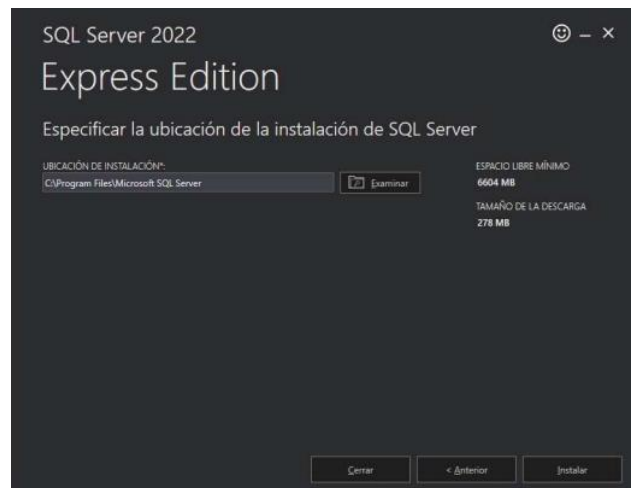
SQL Server 2022 Express is a free edition of SQL Server, ideal for development and production for desktop, web, and small server applications.

[Download now](#)

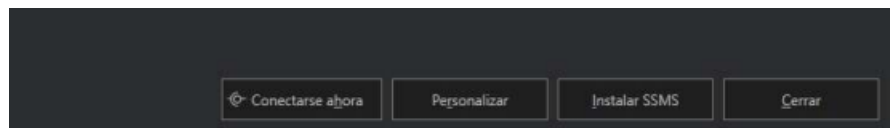
Una vez descargado el instalador de SQL Server escogeremos el tipo de instalación básica.



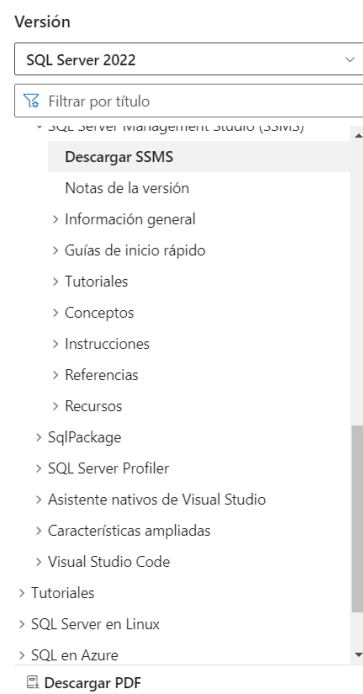
Luego procederemos a escoger la ubicación para el archivo de instalación



Después de instalado, procederemos a instalar el SSMS (SQL Server Management Studio).



En el enlace <https://learn.microsoft.com/es-es/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16> seleccionamos la descarga gratuita y procedemos a instalarlo.



## Descarga de SQL Server Management Studio (SSMS)

Artículo • 10/08/2023 • 49 colaboradores

[Comentarios](#)

### En este artículo

- [Descargar SSMS](#)
- [Idiomas disponibles](#)
- [Novedades](#)
- [Versiones anteriores](#)
- [Mostrar 8 más](#)

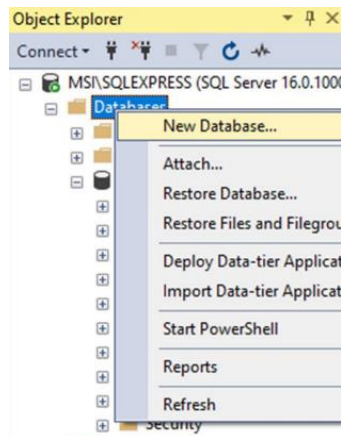
Se aplica a: [SQL Server](#) [Azure SQL Database](#) [Azure SQL Managed Instance](#) [Azure Synapse Analytics](#) [Punto de conexión SQL en Microsoft Fabric](#) [Almacenamiento en Microsoft Fabric](#)

SQL Server Management Studio (SSMS) es un entorno integrado para administrar cualquier infraestructura de SQL, desde SQL Server a Azure SQL Database. SSMS proporciona herramientas para configurar, supervisar y administrar instancias de SQL Server y bases de datos. Use SSMS para implementar, supervisar y actualizar los componentes de nivel de datos que usan las aplicaciones, además de compilar consultas y scripts.

Use SSMS para consultar, diseñar y administrar bases de datos y almacenes de datos, estén donde estén, en el equipo local o en la nube.

Para los clientes que necesitan un complemento multiplataforma para SSMS a fin de administrar SQL

Una vez instalado el SSMS, crearemos la base de datos persona\_db.



Después de crear la base de datos, se crearán las tablas definidas con anterioridad en el modelo, las cuales nos dan una guía en el archivo “persona\_ddl\_mysql.sql”

```
-- Table `arq_per_db`.`persona`
-----
CREATE TABLE IF NOT EXISTS `arq_per_db`.`persona` (
  `cc` INT(15) NOT NULL,
  `nombre` VARCHAR(45) NOT NULL,
  `apellido` VARCHAR(45) NOT NULL,
  `genero` ENUM('M', 'F') NOT NULL,
  `edad` INT(3) NULL DEFAULT NULL,
  PRIMARY KEY (`cc`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- Table `arq_per_db`.`profesion`
-----
CREATE TABLE IF NOT EXISTS `arq_per_db`.`profesion` (
  `id` INT(6) NOT NULL,
  `nom` VARCHAR(90) NOT NULL,
  `des` TEXT NULL DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- Table `arq_per_db`.`estudios`
-----
CREATE TABLE IF NOT EXISTS `arq_per_db`.`estudios` (
  `id_prof` INT(6) NOT NULL,
  `cc_per` INT(15) NOT NULL,
  `fecha` DATE NULL DEFAULT NULL,
  `univer` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`id_prof`, `cc_per`),
  INDEX `estudio_persona_fk` (`cc_per` ASC) VISIBLE,
  CONSTRAINT `estudio_persona_fk`
    FOREIGN KEY (`cc_per`)
      REFERENCES `arq_per_db`.`persona` (`cc`),
  CONSTRAINT `estudio_profesion_fk`
    FOREIGN KEY (`id_prof`)
```

Luego debemos instalar Visual Studio: <https://visualstudio.microsoft.com/es/free-developer-offers/>



### Visual Studio Community

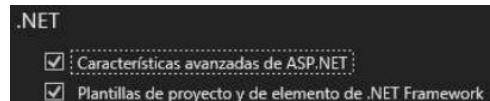
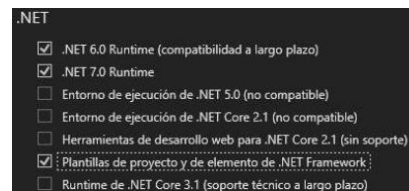
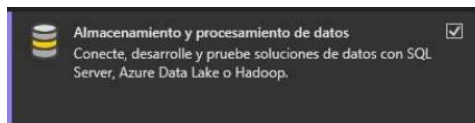
El mejor IDE completo para desarrolladores de .NET y C++ en Windows. Completamente equipado con una buena matriz de herramientas y características para elevar y mejorar todas las etapas del desarrollo de software.

[Más información →](#)

Descarga gratuita

Después de instalar Visual Studio debemos instalar los siguientes complementos:

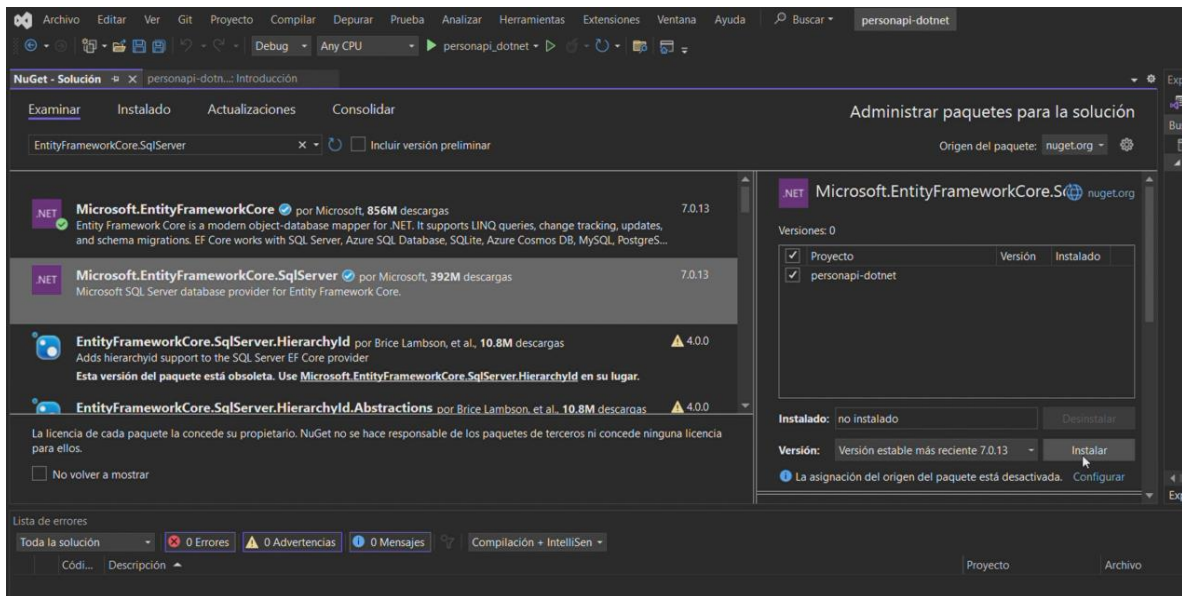
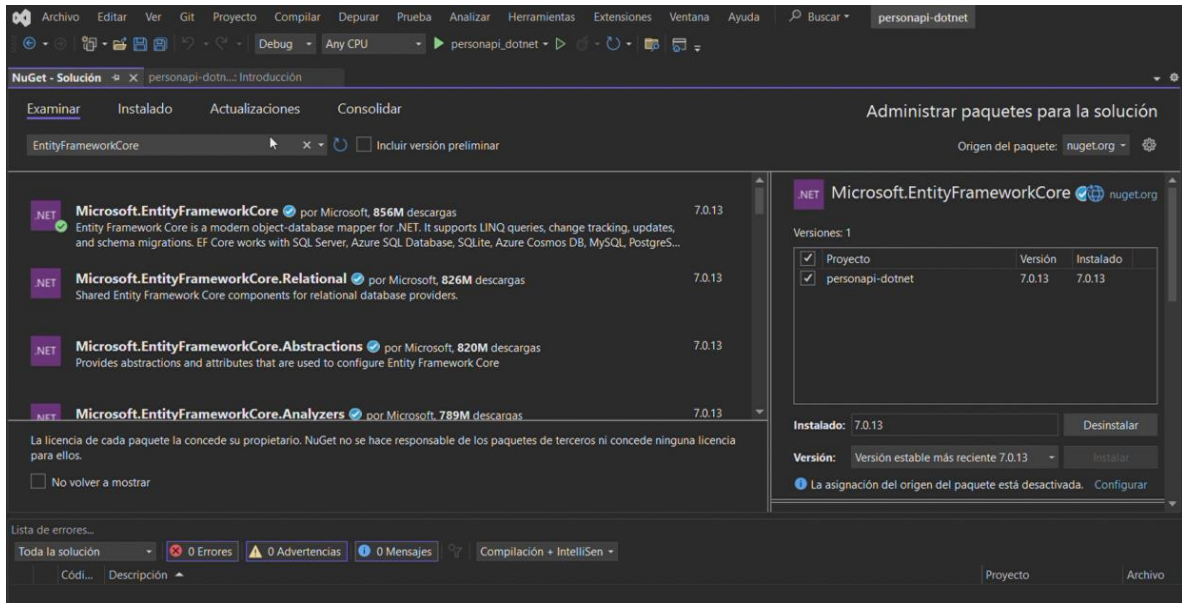
- Desarrollo ASP.NET y web
- Almacenamiento y procesamiento de datos
- Plantillas de proyecto y elementos de .NET Framework
- Características avanzadas de ASP.NET



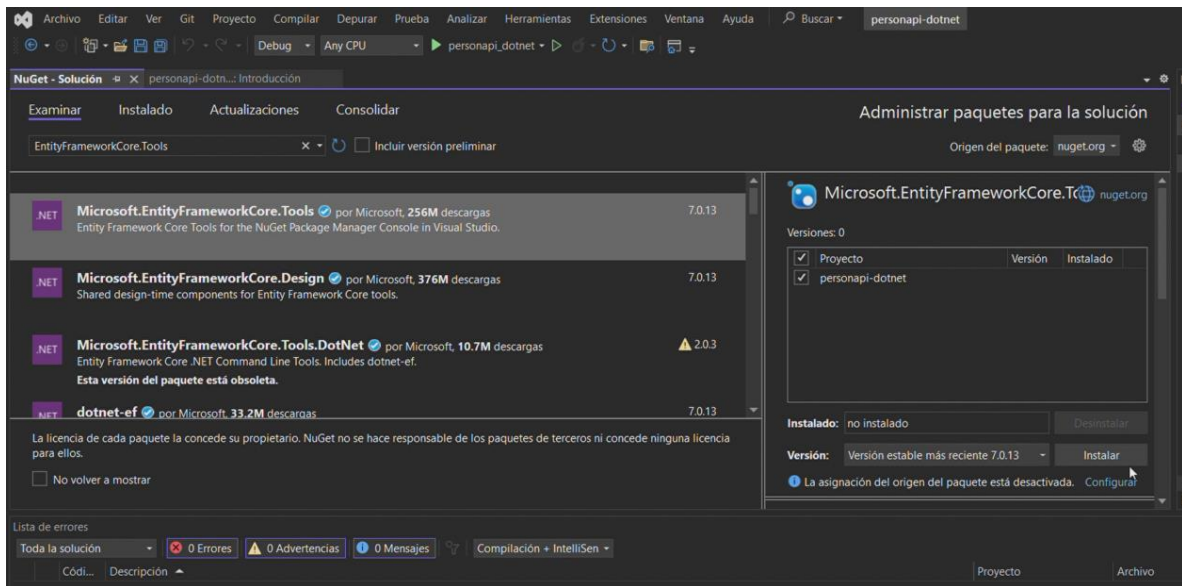
Después de crear el proyecto debemos instalar diferentes paquetes para esto nos dirigimos al menú de herramientas, luego a administrador de paquetes NuGet y por último a Consola del Administrador de paquetes.

En dicho explorador instalaremos estos paquetes como se ve en las siguientes imágenes:

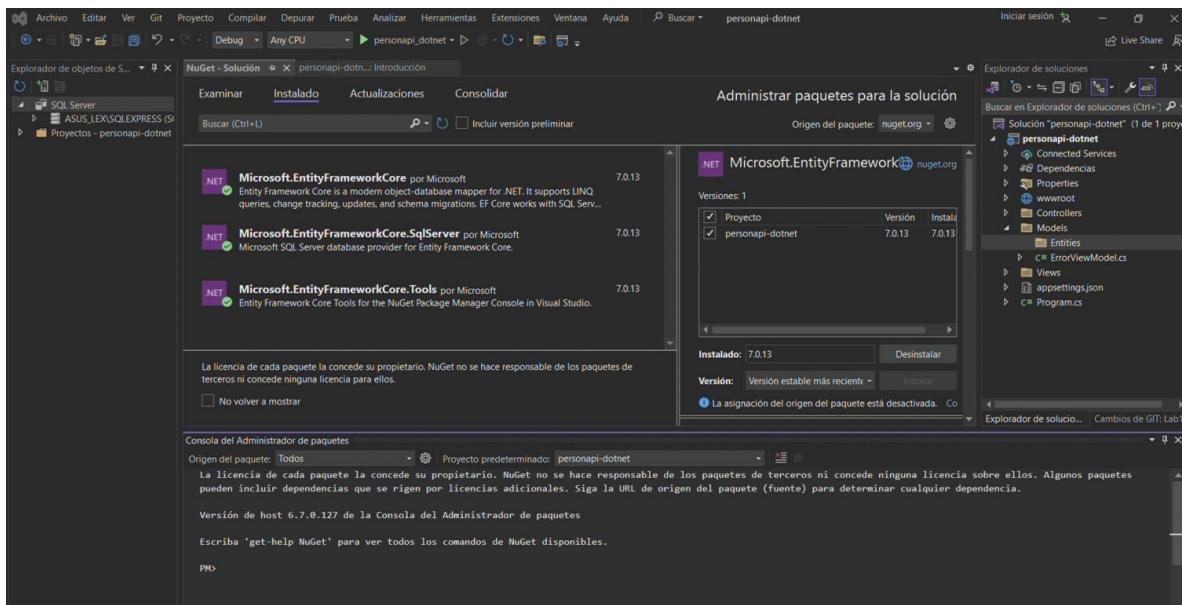
- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools



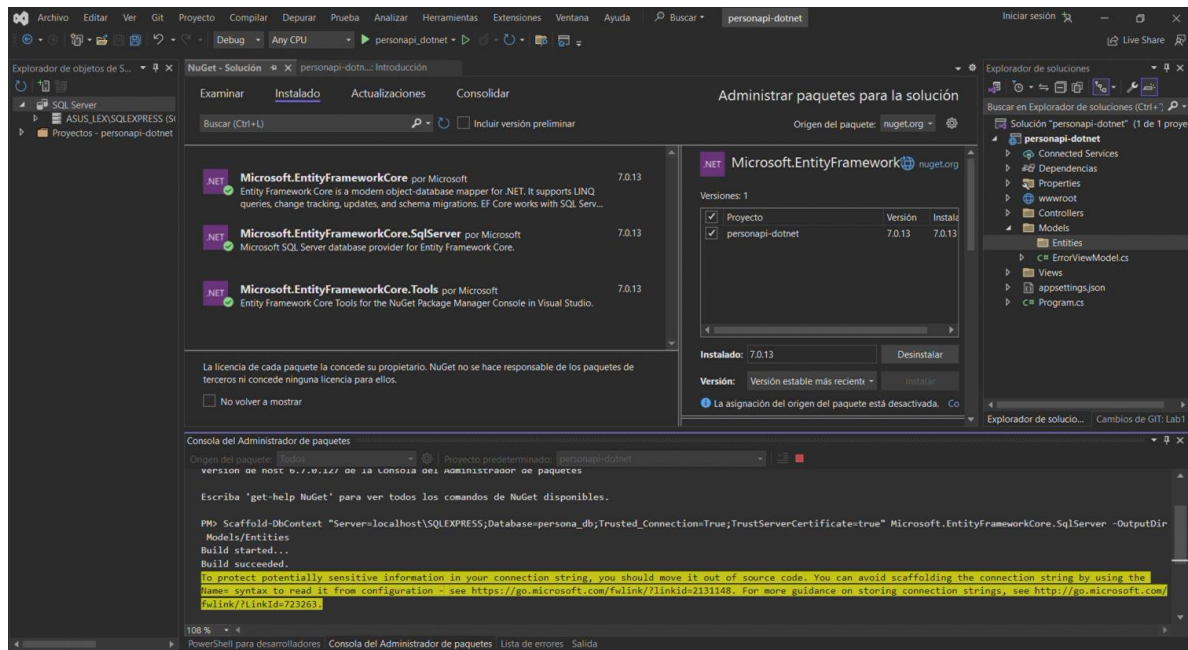




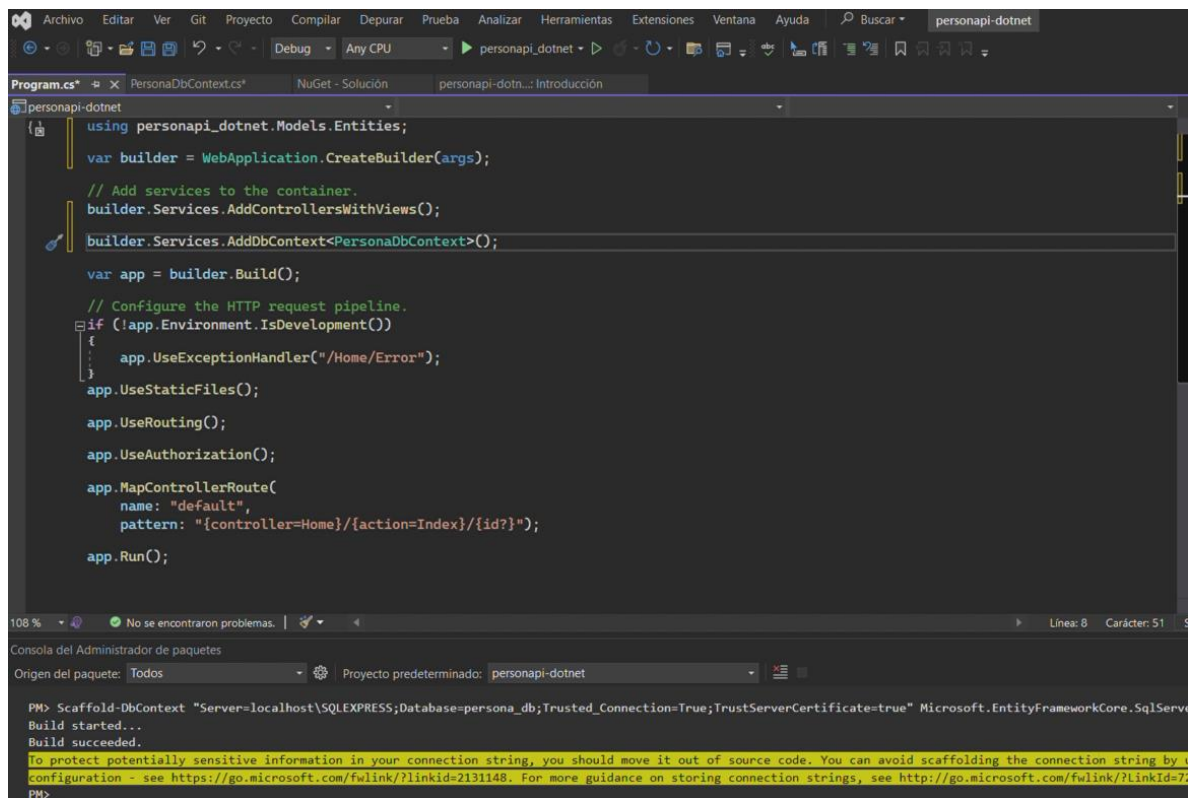
Procedemos a establecer conexión con la base de datos:



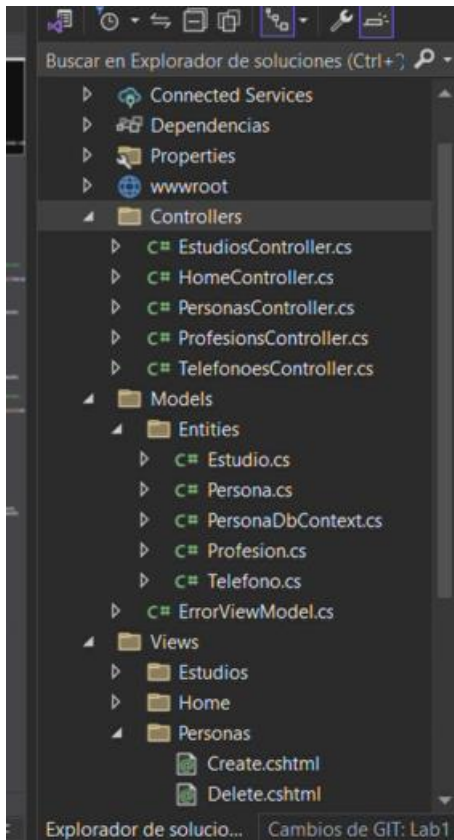
En seguida procedemos con la creación de entidades, para esto creamos una carpeta llamada entities:



Luego de esto en el archivo Program.cs, añadir:  
builder.Services.AddDbContext<PersonaDbContext>();



Creamos los controllers y views correspondientes:







Para la integración con swagger, se debe abrir la terminal e ingresar el comando “Install-Package Swashbuckle.AspNetCore -Version 6.2.3”

```
Build succeeded.
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 N
configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723263.
PM> Install-Package Swashbuckle.AspNetCore -Version 6.2.3
Restaurando paquetes para C:\Users\nicol\Documents\Arqui\lab1\personapi-dotnet\personapi-dotnet\personapi-dotnet.csproj...
GET https://api.nuget.org/v3-flatcontainer/swashbuckle.aspnetcore/index.json
0.5 %
```

```
Consola del Administrador de paquetes
Origen del paquete: Todos
Proyecto predeterminado: personapi-dotnet
OK https://api.nuget.org/v3-flatcontainer/swashbuckle.aspnetcore.swaggerui/6.2.3/swashbuckle.aspnetcore.swaggerui.6.2.3.nupkg 377 ms
OK https://api.nuget.org/v3-flatcontainer/swashbuckle.aspnetcore.swaggerui/6.2.3/swashbuckle.aspnetcore.swaggerui.6.2.3.nupkg 383 ms
Se instaló Swashbuckle.AspNetCore 6.2.3 de https://api.nuget.org/v3/index.json con el hash de contenido cnzQm8Le+hInsw25Vw10H0CPXpY1/szcvmqZ312v+QyrlBwAm00g6R1yHB18s/mLeywC+Rg209ndz:01UNYQ==.
Se instaló Swashbuckle.AspNetCore.Swagger 6.2.3 de https://api.nuget.org/v3/index.json con el hash de contenido xQ7J1sl0Bm8g/qHWPcvk031VvUI8BwFC98KSh68T5y50AnBNctfac7XRSEZf+eD7/MasvANncTqwZYfmQ==.
Se instaló Swashbuckle.AspNetCore.SwaggerGen 6.2.3 de https://api.nuget.org/v3/index.json con el hash de contenido +Xq7Wd9KCfcXlnbJVFNgY8ITdP2TRY1pbt6IKzDw5Fufxd91BfNDtCt+/wKwX701BBFwX1dnn02/V072A==.
Se instaló Microsoft.Extensions.ApiDescription.Server 3.0.0 de https://api.nuget.org/v3/index.json con el hash de contenido LMdQ/76F6sOCsl1f7+Xh3fS/wKUE5ryeXAPcoCnuwOQT55m0p571gDh/pHghuGz/e+AmEQb7pRgB+
vut8m=.
Se instaló Swashbuckle.AspNetCore.SwaggerUI 6.2.3 de https://api.nuget.org/v3/index.json con el hash de contenido bCR187uKTVb4G+KURmMBlQrL64St04dEFZcFqIM67Zc05r/H47E083yblMYDvFNd01DCv8mPcrz91/VEHQ5g==.
Instalando el paquete NuGet Swashbuckle.AspNetCore 6.2.3.
Generación de archivo MSBuild C:\Users\nicol\Documents\Arqui\lab1\personapi-dotnet\personapi-dotnet\personapi-dotnet\obj\personapi-dotnet.csproj.nuget.g.props.
Generación de archivo MSBuild C:\Users\nicol\Documents\Arqui\lab1\personapi-dotnet\personapi-dotnet\personapi-dotnet\obj\personapi-dotnet.csproj.nuget.g.targets.
Escribiendo el archivo de recursos en el disco. Ruta de acceso: C:\Users\nicol\Documents\Arqui\lab1\personapi-dotnet\personapi-dotnet\personapi-dotnet\obj\project.assets.json
Se ha restaurado C:\Users\nicol\Documents\Arqui\lab1\personapi-dotnet\personapi-dotnet\personapi-dotnet\personapi-dotnet.csproj (en 3.8 s).
'Microsoft.Extensions.ApiDescription.Server 3.0.0' se instaló correctamente en personapi-dotnet
'Microsoft.OpenApi 1.2.3' se instaló correctamente en personapi-dotnet
'Swashbuckle.AspNetCore 6.2.3' se instaló correctamente en personapi-dotnet
'Swashbuckle.AspNetCore.Swagger 6.2.3' se instaló correctamente en personapi-dotnet
'Swashbuckle.AspNetCore.SwaggerGen 6.2.3' se instaló correctamente en personapi-dotnet
'Swashbuckle.AspNetCore.SwaggerUI 6.2.3' se instaló correctamente en personapi-dotnet
La ejecución de acciones de NuGet tardó 384 ms
```

Instalar las siguientes extensiones mediante el Nuget Package for solutions en las versiones más recientes.

	<b>Swashbuckle.AspNetCore</b> por Swashbuckle.AspNetCore Swagger tools for documenting APIs built on ASP.NET Core	6.2.3 6.5.0
	<b>Swashbuckle.AspNetCore.Swagger</b> por Swashbuckle.AspNetCore.Swagger Middleware to expose Swagger JSON endpoints from APIs built on ASP.NET Core	6.5.0
	<b>Swashbuckle.AspNetCore.SwaggerGen</b> por Swashbuckle.AspNetCore.SwaggerGen Swagger Generator for APIs built on ASP.NET Core	6.5.0
	<b>Swashbuckle.AspNetCore.SwaggerUI</b> por Swashbuckle.AspNetCore.SwaggerUI Middleware to expose an embedded version of the swagger-ui from an ASP.NET Core application	6.5.0

Una vez instalados, en el archivo de “Program.cs” agrega las siguientes líneas de código para que el programa ejecute el swagger usando el /swagger.

```
NuGet - Solución  TelefonoesController.cs  EstudiosController.cs  Profesion...
personapi-dotnet
{
    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();
    builder.Services.AddControllersWithViews();

    builder.Services.AddDbContext<PersonaDbContext>();

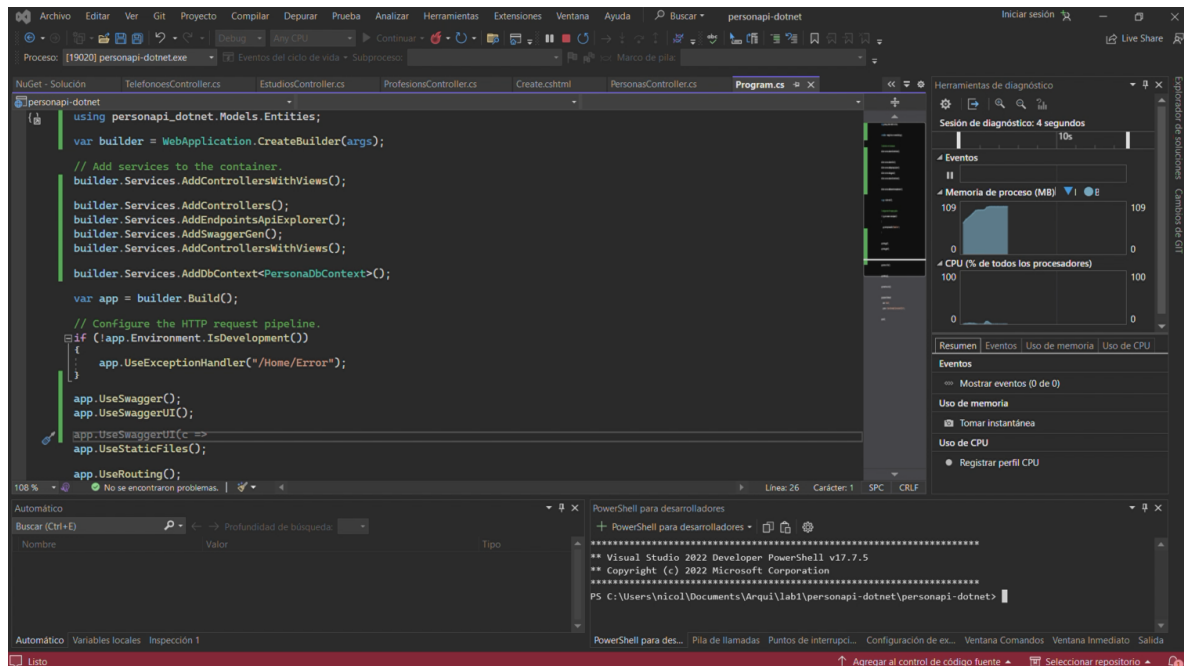
    var app = builder.Build();

    // Configure the HTTP request pipeline.
    if (!app.Environment.IsDevelopment())
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseSwagger();
    app.UseSwaggerUI();
    app.UseSwaggerUI(c =>
    app.UseStaticFiles();
    app.UseRouting();
}
```



Finalmente, ejecutamos y verificamos que todo este en orden.



## Conclusiones

- **Implementación de una Aplicación Monolítica:** La implementación de una aplicación monolítica en este proyecto demuestra la capacidad de crear una aplicación de software integral en la que todas las funcionalidades están integradas en una sola unidad. Esto es útil en escenarios donde la simplicidad y la facilidad de desarrollo son prioritarias.
- **Uso Efectivo del Patrón MVC:** La adopción del patrón Modelo-Vista-Controlador (MVC) ha permitido una separación clara de las preocupaciones en la aplicación. El Modelo maneja la lógica de negocio, la Vista se ocupa de la interfaz de usuario y el Controlador coordina las interacciones. Esto mejora la mantenibilidad y escalabilidad del código.
- **Abstracción de Acceso a Datos con DAO:** La implementación del patrón DAO ha permitido abstraer la capa de acceso a datos, lo que facilita la gestión y la interacción con la base de datos. Esto se traduce en un código más limpio y modular, lo que simplifica las operaciones de base de datos.
- **Tecnologías y Herramientas Actuales:** El uso de tecnologías y herramientas actuales, como .NET 7, MS SQL Server 2022, REST y Swagger 3, demuestra la adaptabilidad del proyecto a las últimas tendencias tecnológicas. Esto garantiza que la aplicación sea capaz de satisfacer las necesidades actuales y futuras.

## Lecciones aprendidas

- **Planificación Previa:** La planificación adecuada del proyecto, incluida la configuración del entorno, la instalación de herramientas y la creación de la base de datos, es esencial para evitar obstáculos en etapas posteriores.
- **Mantenimiento de la Base de Datos:** El mantenimiento y la administración de la base de datos son críticos. Asegurarse de que las tablas y datos se mantengan actualizados y protegidos es fundamental para el funcionamiento correcto de la aplicación.
- **Separación de Responsabilidades:** El uso del patrón MVC y el patrón DAO resalta la importancia de separar las responsabilidades en una aplicación. Esto facilita la escalabilidad y la colaboración en el desarrollo.
- **Adopción de Estándares y Buenas Prácticas:** Seguir estándares y buenas prácticas en el desarrollo de software, como la documentación de API con Swagger, garantiza que la aplicación sea más fácil de entender y utilizar tanto para el equipo de desarrollo como para otros usuarios.

## Referencias:

- SQL Server: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>
- SQL Server Manager Studio: <https://learn.microsoft.com/es-es/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>
- Visual Studio: <https://visualstudio.microsoft.com/es/free-developer-offers>
- GitHub: <https://github.com/>