TECNOLOGIA EN DESARROLLO DE SOFTWARE

PROGRAMACION DE SOFTWARE

MATEO VALLEJO ÁLVAREZ

Parcial No°1[20%]

ITM

2024

Ejercicio tipo parcial

- Crear una conexión a base de datos desde el lenguaje de programación C# de .NET, la conexión se deberá realizar con la metodología Code First de Entity Framework, el programa será un servicio WEB API, que cree un elemento para administrar pagos de personas en el sistema. Cada vez que se registre un pago, se deberá realizar un cálculo para saber cuántas personas se han atendido y cuantas han pagado en promedio.

*****Los campos de la entidad Persona deben ser mínimo 5

*****Los campos deben ser mínimo de 3 tipos distintos, no todos debes ser textos

*****Máximo deben existir 2 tablas o entidades, mínimo 1 tabla

IDE = Visual Studio 2022

Lenguaje = C#

Framework = .NET

Aplicación = WEB API

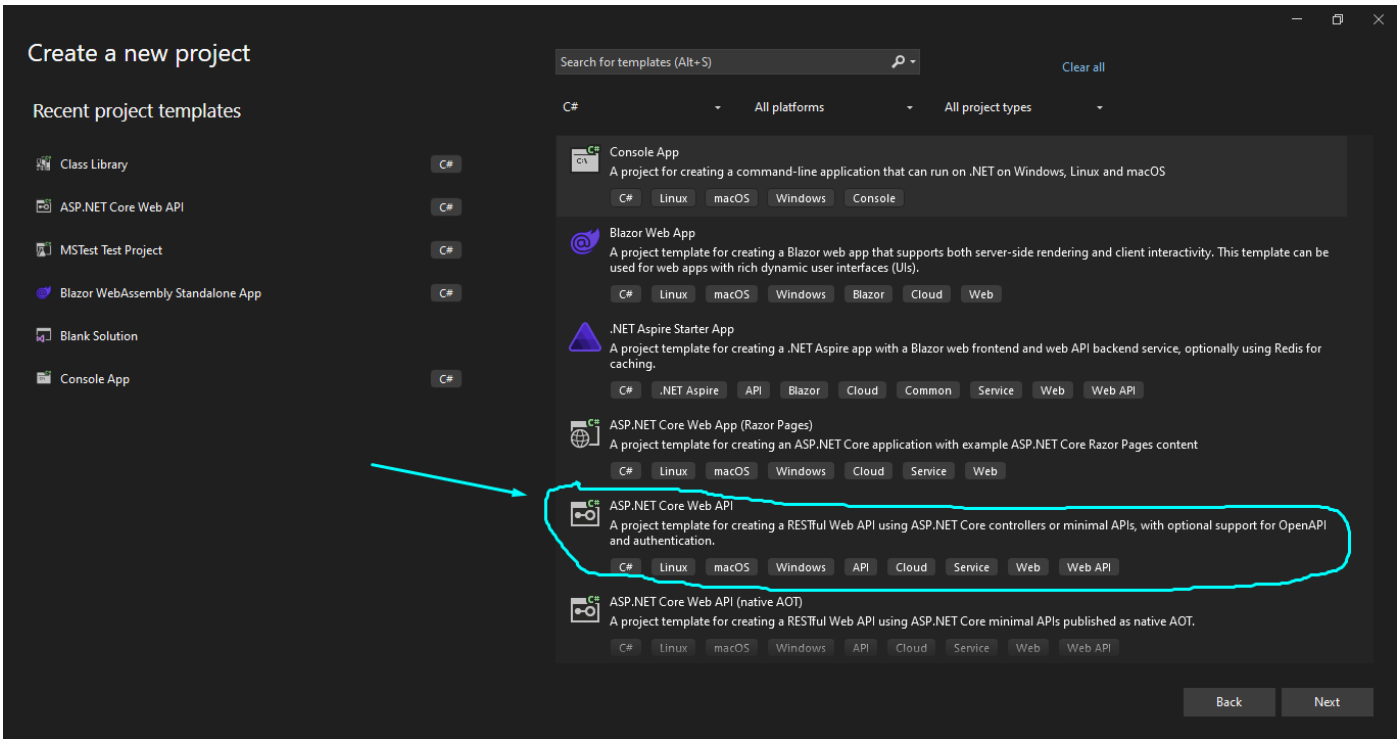
Data Base = SQL SERVER Management Studio 2022

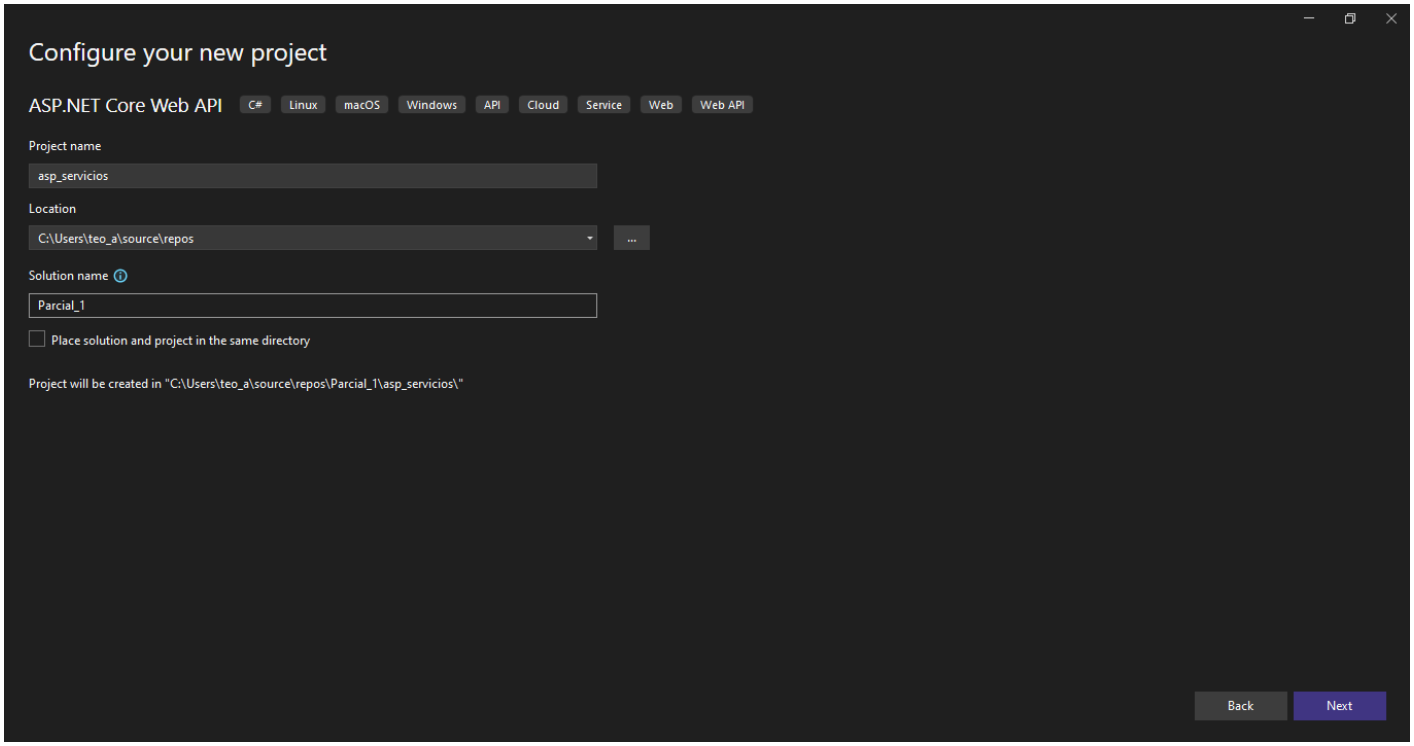
Metodologia = Code First

Modalidad = Presencial

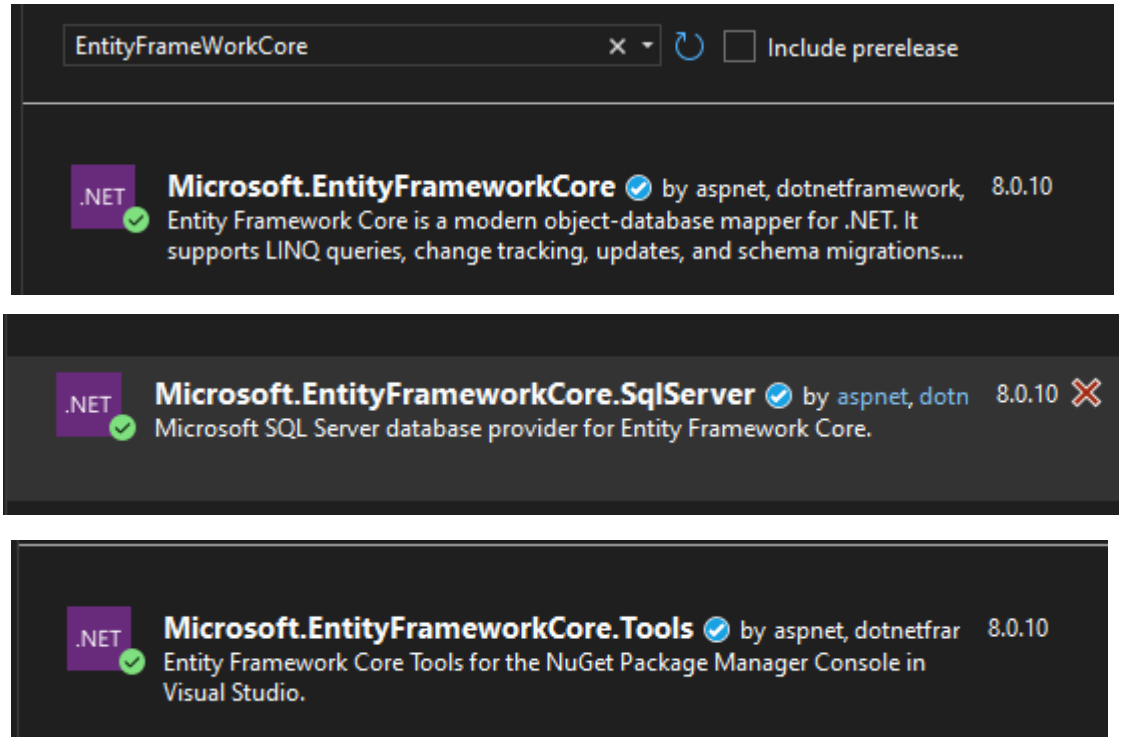
I. CREACION DEL PROYECTO

A. ASP.NET Core Web API

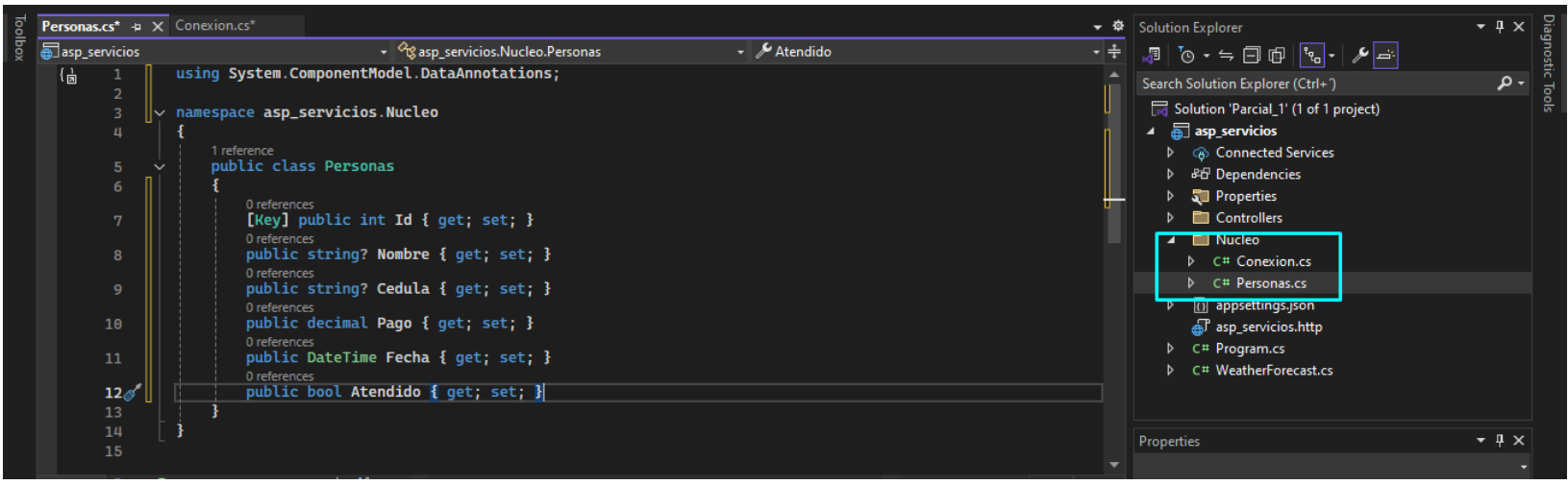


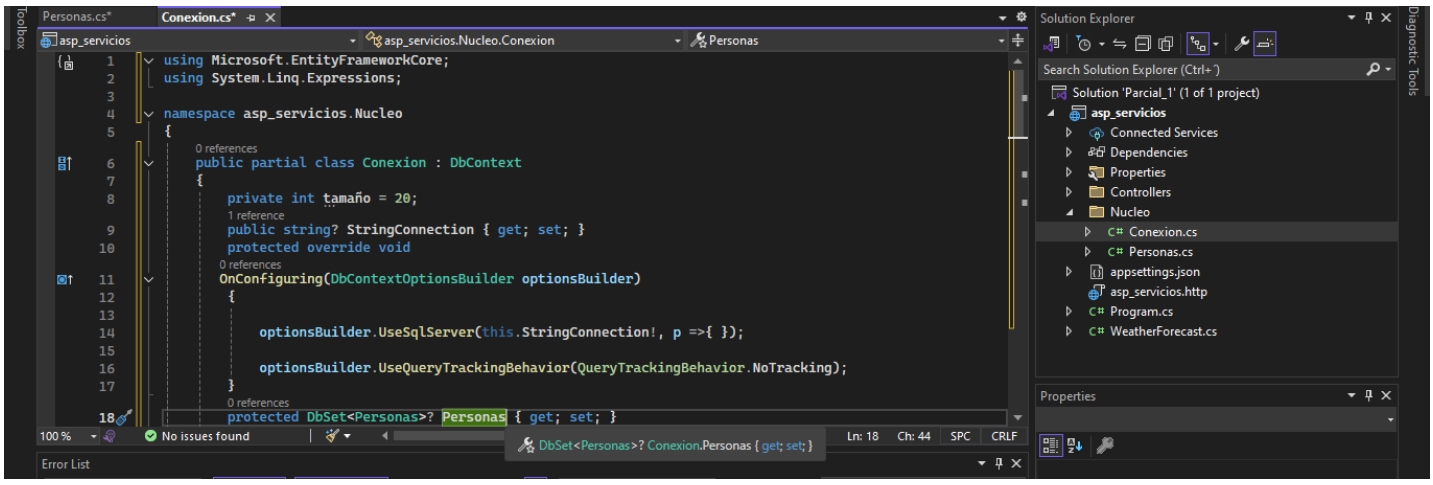


INSTALACION DE NUGGET



AGREGAR CLASES EN LA CARPETA NUCLEO





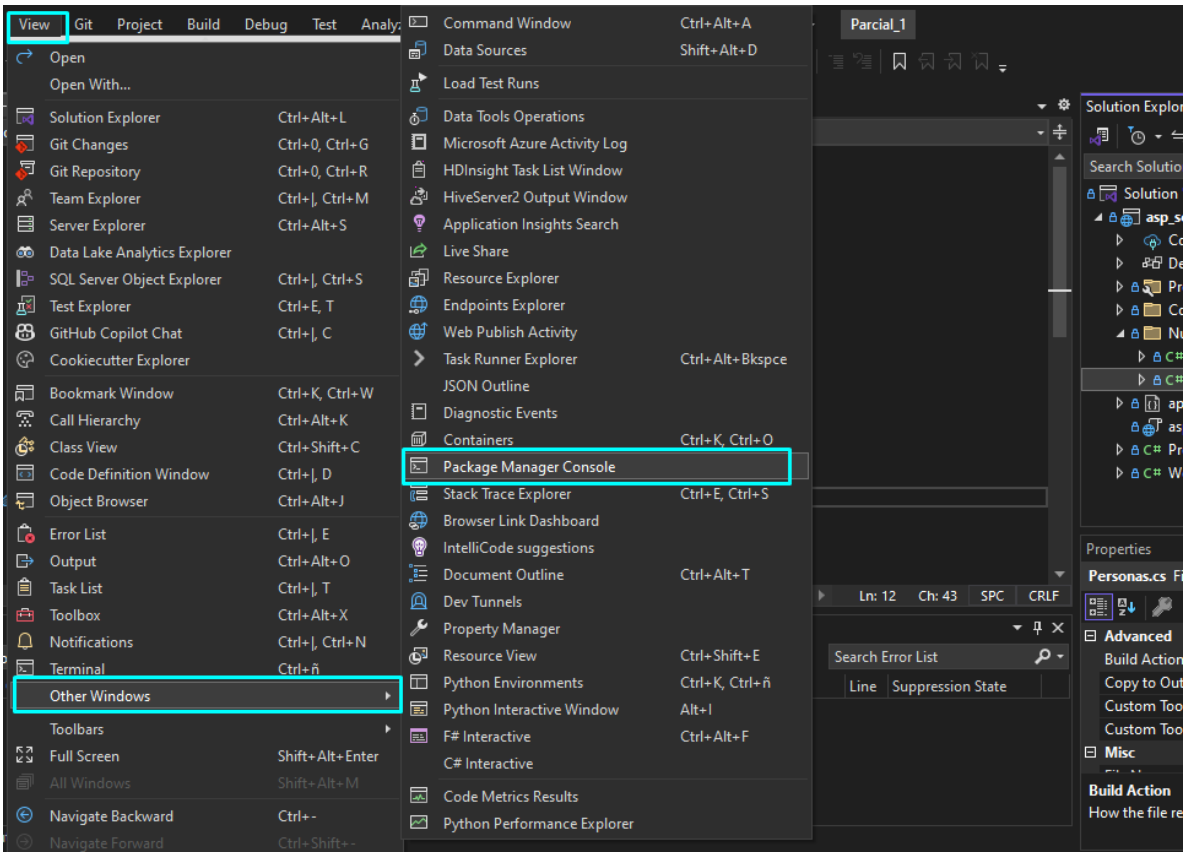
CODIGO CONEXIÓN:

```
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;

namespace asp_servicios.Nucleo
{
   public partial class Conexion : DbContext
   {
      private int tamaño = 20;
      public string? StringConnection { get; set; }
      protected override void
     OnConfiguring(DbContextOptionsBuilder optionsBuilder)
      {

         optionsBuilder.UseSqlServer(this.StringConnection!, p =>{ });

         optionsBuilder.UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking);
      }
      protected DbSet<Personas>? Personas { get; set; }
      public virtual DbSet<T> ObtenerSet<T>() where T : class, new()
      {
         return this.Set<T>();
```

```
}
public virtual List<T> Listar<T>() where T : class, new()
{
   return this.Set<T>()
   .Take(tamaño)
   .ToList();
}
public virtual List<T> Buscar<T>(Expression<Func<T, bool>> condiciones) where T : class, new()
{
   return this.Set<T>()
   .Where(condiciones)
   .Take(tamaño)
   .ToList();
}
public virtual bool Existe<T>(Expression<Func<T, bool>> condiciones) where T : class, new()
{
   return this.Set<T>().Any(condiciones);
}
public virtual void Guardar<T>(T entidad) where T : class, new()
{
   this.Set<T>().Add(entidad);
}
public virtual void Modificar<T>(T entidad) where T : class
{
   var entry = this.Entry(entidad);
   entry.State = EntityState.Modified;
}
public virtual void Borrar<T>(T entidad) where T : class, new()
{
   this.Set<T>().Remove(entidad);
}
public virtual void Separar<T>(T entidad) where T : class, new()
```

```
        {
            this.Entry(entidad).State =
           EntityState.Detached;
        }
        public virtual void GuardarCambios()
        {
            this.SaveChanges();
        }

    }
}
```

SIGUIENTE PASO



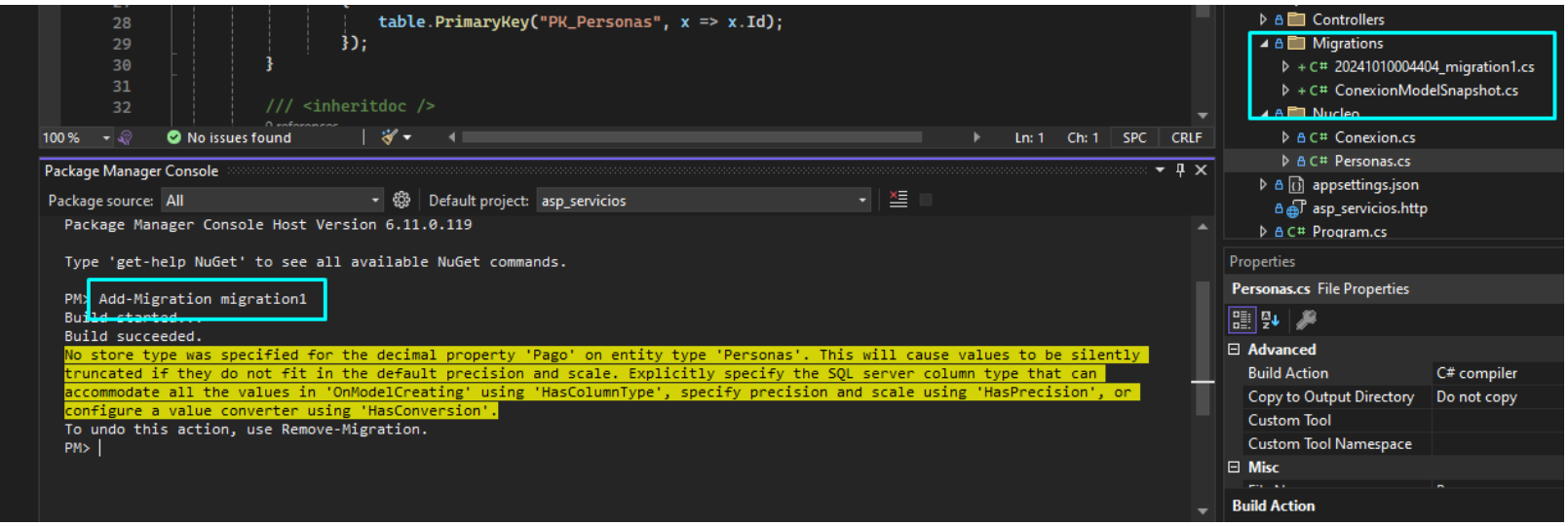
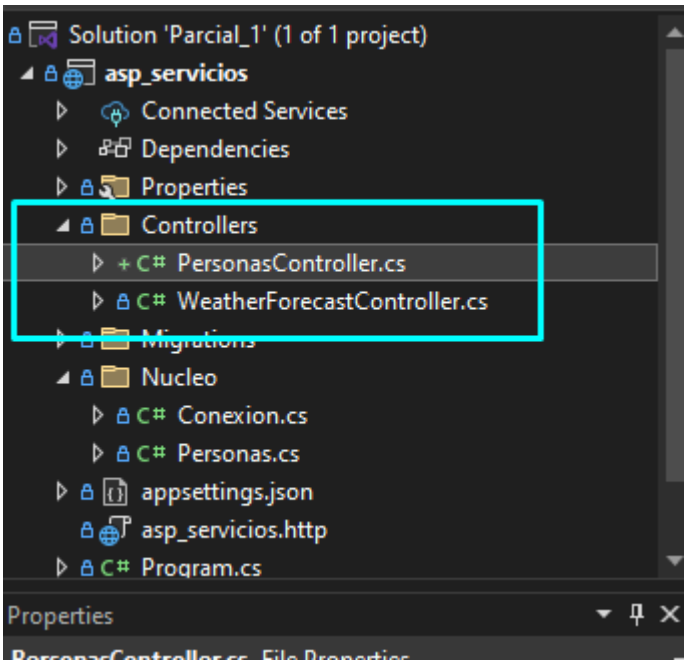
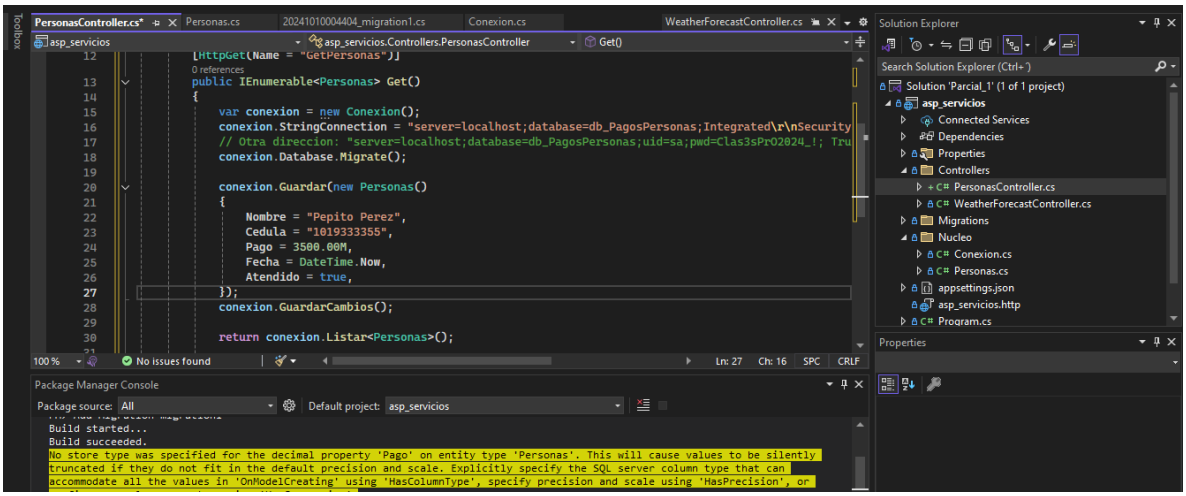


table.PrimaryKey("PK_Personas", x => x.Id);
});
}
/// <inheritdoc />
100 %
No issues found
Ln: 1
Ch: 1
SPC
CRLF
Package Manager Console
Package source: All
Default project: asp_servicios
Package Manager Console Host Version 6.11.0.119
Type 'get-help NuGet' to see all available NuGet commands.
PM> Add-Migration migration1
Build started...
Build succeeded.
No store type was specified for the decimal property 'Pago' on entity type 'Personas'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value converter using 'HasConversion'.
To undo this action, use Remove-Migration.
PM>
Controllers
Migrations
20241010004404_migration1.cs
ConexionModelSnapshot.cs
Nucleo
Conexion.cs
Personas.cs
appsettings.json
asp_servicios.http
Program.cs
Properties
Personas.cs File Properties
Advanced
Build Action
C# compiler
Copy to Output Directory
Do not copy
Custom Tool
Custom Tool Namespace
Misc
Build Action


Solution 'Parcial_1' (1 of 1 project)
asp_servicios
Connected Services
Dependencies
Properties
Controllers
PersonasController.cs
WeatherForecastController.cs
Migrations
Nucleo
Conexion.cs
Personas.cs
appsettings.json
asp_servicios.http
Program.cs
Properties



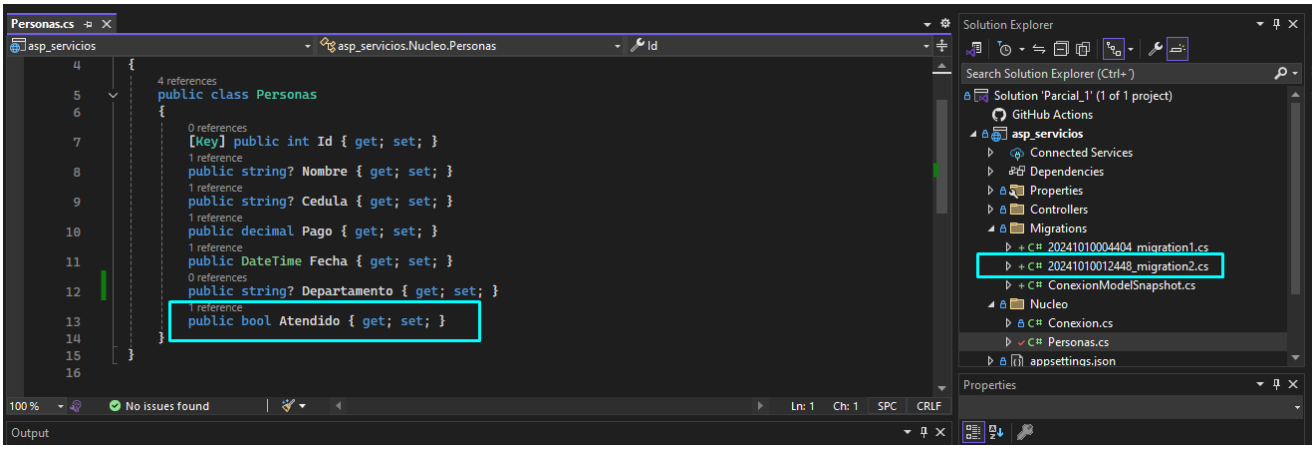
CODIGO PersonasController

```
using Microsoft.AspNetCore.Mvc;

using asp_servicios.Nucleo;

using Microsoft.EntityFrameworkCore;


namespace asp_servicios.Controllers

{

   [ApiController]

   [Route("[controller]")]

   public class PersonasController : ControllerBase

   {


      [HttpGet(Name = "GetPersonas")]

      public IEnumerable<Personas> Get()

      {

         var conexion = new Conexion();

         conexion.StringConnection =
"server=localhost;database=db_PagosPersonas;Integrated\r\nSecurity=True;TrustServerCertificate=true;";

         // Otra direccion: "server=localhost;database=db_PagosPersonas;uid=sa;pwd=Clas3sPrO2024_!;
TrustServerCertificate = true";

         conexion.Database.Migrate();
```

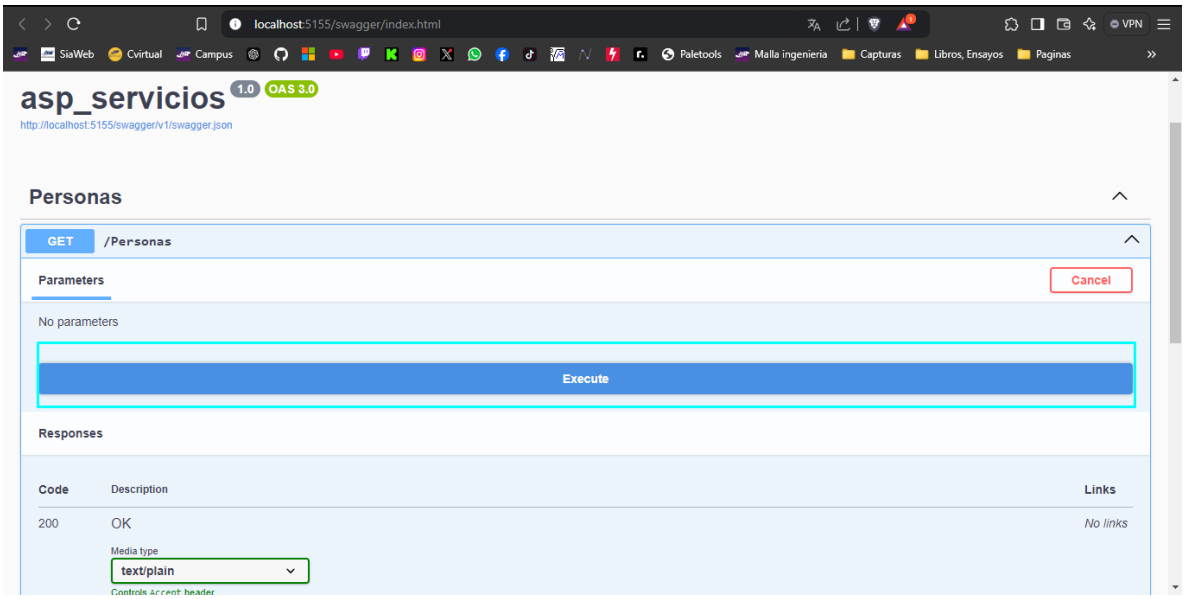
```
            conexion.Guardar(new Personas()
            {
                Nombre = "Pepito Perez",
                Cedula = "1019333355",
                Pago = 3500.00M,
                Fecha = DateTime.Now,
                Atendido = true,
            });
            conexion.GuardarCambios();


            return conexion.Listar<Personas>();


        }
    }
}
```

En el caso que nos olvidemos o queramos agregar otro campo se hace un migration2 y agrega una modificación creando otra columna

(El nuevo campo se agrega en la clase del núcleo)



Ejecutamos la aplicación



Y si todo esta correcto y estamos conectados a la base de datos (Se creara automáticamente con el nombre que le pongamos) tendremos respuestas del API