# Índice

Contenido

[Índice 1](#_Toc160888358)

[Modelo de datos 1](#_Toc160888359)

[Arquitectura del proyecto 2](#_Toc160888360)

[Crear la base de datos con EF 5](#_Toc160888361)

[Clase DataContext 6](#_Toc160888362)

[Appsettings 7](#_Toc160888363)

[Inyección de dependencias Servicio SQlServer 8](#_Toc160888364)

[Ejecutar proyectos de forma múltiple 8](#_Toc160888365)

[Migración de entidades hacia la base de datos SQL SERVER 10](#_Toc160888366)

[Crear un nuevo Branch en GIT para publicar la nueva versión del proyecto en el repositorio GIT. 10](#_Toc160888367)

# Modelo de datos

Vamos a crear un sencillo sistema de información para una Veterinaria, que va a utilizar el siguiente modelo de datos:

Owner

Pet

Pet Type

History

Service Type

Agenda

1

\*

1

1

1

1

1

\*

\*

\*

\*

\*

User

UserRole

Role

Manager

1

1

1

1

1

1

\*

\*

# Arquitectura del proyecto

SQL Server on Azure

.NET Core 8 API

Blazor WebAssembly Application

Library Class



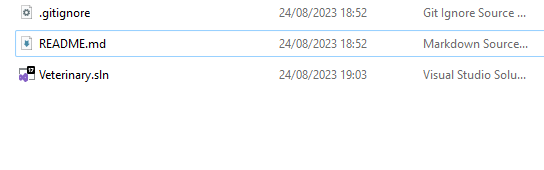
**Crear proyecto**

Vamos a crear esta estructura en Visual Studio (asegúrese de poner todos los proyectos en el mismo directorio C://Projects)

Crear un nuevo repositorio GITHUB, usar gitignore, copiar ruta, repositorio private

Clonar proyecto git desde Visual Studio C://Projects/Veterinary

* Nuevo proyecto **Solution Blank** llamado **Veterinary**. Dentro de la raíz C://Projects al final la .sln queda el ícono dentro de Projects🡪(Veterinary.sln)

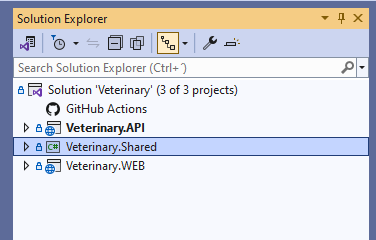


* Sobre el ícono Veterinary.sln del explorador de soluciones oprimimos click derecho y presionamos Open
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **Class Library**, llamado **Veterinary.Shared**

Ubicación: C:\Projects/Veterinary (Borramos la class1.cs que se genera)

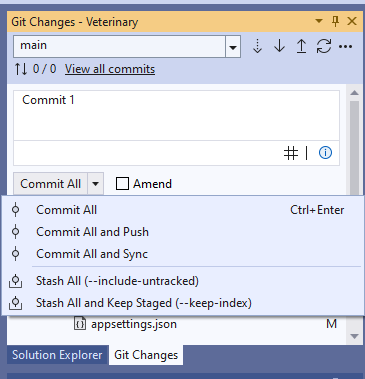
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **ASP.NET Core Web API**, llamado **Veterinary.API** Ubicación: C:\Projects/Veterinary
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **Blazor WebAssembly Standalone App**, llamado **Veterinary.WEB** Ubicación: C:\Projects/Veterinary

Así debe verse al final , la estructura de los proyectos en el Solution Explorer:



Hacemos el primer **Commit All and Sync** en nuestro repositorio. Pestaña Git Changes

\*(Si en Git Changes no se visualiza el árbol de carpetas de los proyectos, será necesario cerrar la solución, y abrirla de nuevamente) Commit All and Sync



# Crear la base de datos con EF

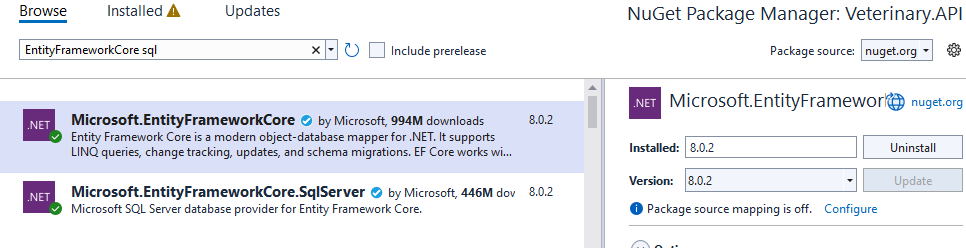
Diagrama

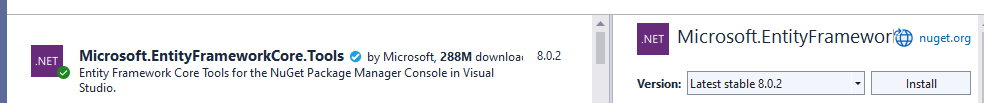
Descripción generada automáticamente

Code First y Database First. En este curso trabajaremos con EF Code First,

Documentación: <https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db>

# Instalar los siguientes Nuggets en el proyecto Veterinary.API





Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

1. Empecemos creando en el proyecto **Veterinary.Shared** la carpeta **Entities** y dentro de esta carpeta la entidad **Owner**:

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class Owner

{

public int Id { get; set; }

[Display(Name = "Document")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Document { get; set; }

[Display(Name = "First Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string LastName { get; set; }

[Display(Name = "Fixed Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string FixedPhone { get; set; }

[Display(Name = "Cell Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string CellPhone { get; set; }

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string Address { get; set; }

public string FullName => $"{FirstName} {LastName}";

public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";

}

# Clase DataContext

1. En el proyecto **API** creamos la carpeta **Data** y dentro de esta la clase **DataContext**:

using Microsoft.EntityFrameworkCore;

using Veterinary.Shared.Entities;

namespace Veterinary.API.Data

{

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<Owner> Owners { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

}

}

}

# Appsettings

1. Configurar el ConnectionStrings en el **appsettings.Development.json** y en **appsettings.json** del proyecto **API**:

{

"ConnectionStrings": {

"DefaultConnection": "Server= MyServer;Database=Veterinary;Encrypt=False;User Id=sa;Password=Abcd1234\*;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

# Inyección de dependencias Servicio SQlServer

1. Configurar la inyección del DataContext en la clase **Program** del proyecto **API**:

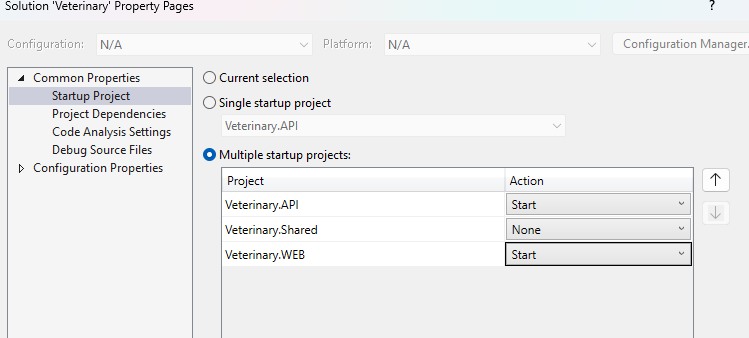
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=DefaultConnection"));

var app = builder.Build();

# Ejecutar proyectos de forma múltiple

1. En el desplegable Startup Projects seleccionar Veterinary.API como proyecto de inicio, abrir Package Manager Console(Tool) , e igualmente elegir Veterinary.API ,como Default Project



# Migración de entidades hacia la base de datos SQL SERVER

1. Correr los siguientes comandos en Package Manager Console:

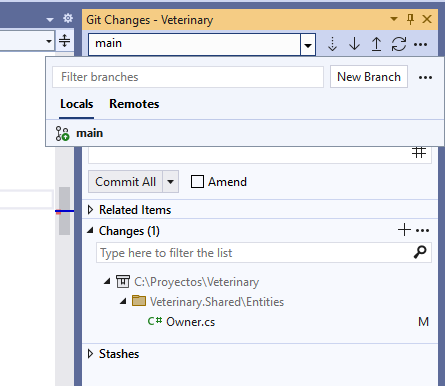
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

add-migration Inicial (este comando crea la migración dentro del proyecto)

update-database (este comando envía la migración hacia SQL SERVER)

# Crear un nuevo Branch en GIT para publicar la nueva versión del proyecto en el repositorio GIT.



Tabla

Descripción generada automáticamente

1. Hacemos nuestro segundo **Commit All and Sync** y luego el Push en una nuevo branch(Si en Git Changes no se visualiza el árbol de carpetas de los proyectos, será necesario cerrar la solución, y abrirla de nuevamente).

# Creando los primeros métodos en el primer controlador

1. En el proyecto **API** en la carpeta **Controllers** creamos la clase **OwnersController**:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Veterinary.API.Data;

using Veterinary.Shared.Entities;

namespace Veterinary.API.Controllers

{

[ApiController]

[Route("/api/Owners")]

public class OwnersController : ControllerBase

{

private readonly DataContext \_context;

public OwnersController(DataContext context)

{

\_context = context;

}

//Get con lista

//Select \* From owners

[HttpGet]

public async Task<ActionResult> Get()

{

return Ok(await \_context.Owners.ToListAsync());

}

[HttpPost]

public async Task<ActionResult> Post(Owner Owner)

{

\_context.Add(Owner);

await \_context.SaveChangesAsync();

return Ok(Owner);

}

[HttpPut]

public async Task<ActionResult> Put(Owner owner)

{

\_context.Update(owner);

await \_context.SaveChangesAsync();

return Ok(owner);

}

[HttpDelete("{id:int}")]

public async Task<ActionResult> Delete(int id)

{

var afectedRows = await \_context.Owners

.Where(x => x.Id == id)

.ExecuteDeleteAsync();

if (afectedRows == 0)

{

return NotFound();

}

return NoContent();

}