

No. Doc 001	Sistema	Fecha de Realizacion	Responsable
S/N	PRUEBA-TECNICA	14/08/2024	Victor Hugo Soto Valencia

DOCUMENTACION TECNICA Y FUNCIONALIDAD

Objetivo

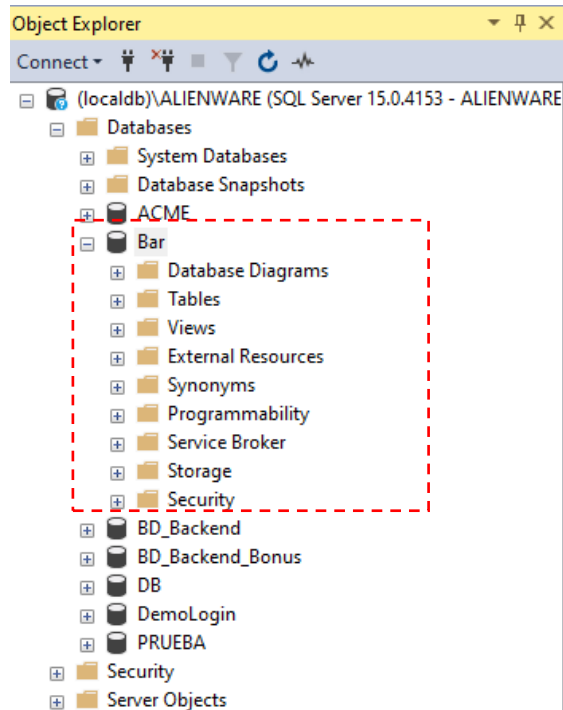
Esta documentación tiene como objetivo el poder mostrar los pasos consiguientes a la realización de la Prueba Técnica solicitada por medio del cual se especifican los procesos y ajustes para el Sistema Prueba Técnica_V1.

Observaciones

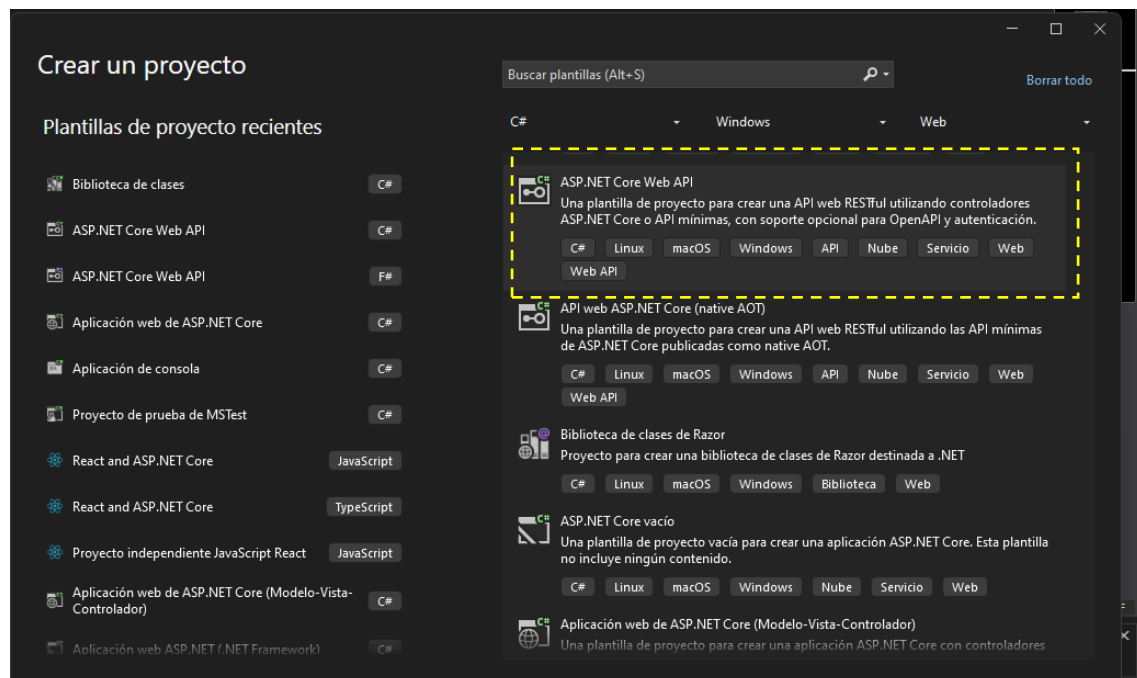
En la realización previa al desarrollo, se tomaron en cuenta la ejecución y uso de **ORM-CodeFirst** y las entidades de paquetes **NuGet Microsoft.EntityFrameworkCore.Tools** y **Microsoft.EntityFrameworkCore.SqlServer** como paquetes principales en el uso, además de también crear el proyecto bajo la característica de ser **WEB_API_Net Core 8.0.1**. Del mismo modo se creo la respectiva base de datos con **SQLServer** con la que el sistema establece conexión y su presunta utilidad con los Métodos **Web Api** correspondientes al ya conocido **CRUD**.

Se crea la cadena de conexión y los campos necesarios para la conexión de los cuales en modo **localhost** se establecen y permiten la funcionalidad requerida para la muestra de ejecución y funcionalidad establecida para la práctica.

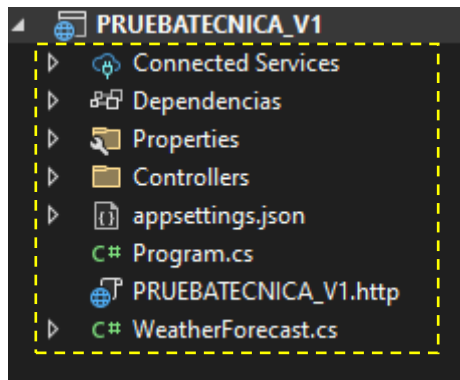
De forma habitual y requerida se crea una Base de datos con el objetivo de tener el contenedor requerido a la programación a ejecutar, a continuación, se muestra la Base de datos Creada en la siguiente imagen



A continuación, se crea un nuevo proyecto en visual Studio **ASP.NET CORE WEB API**

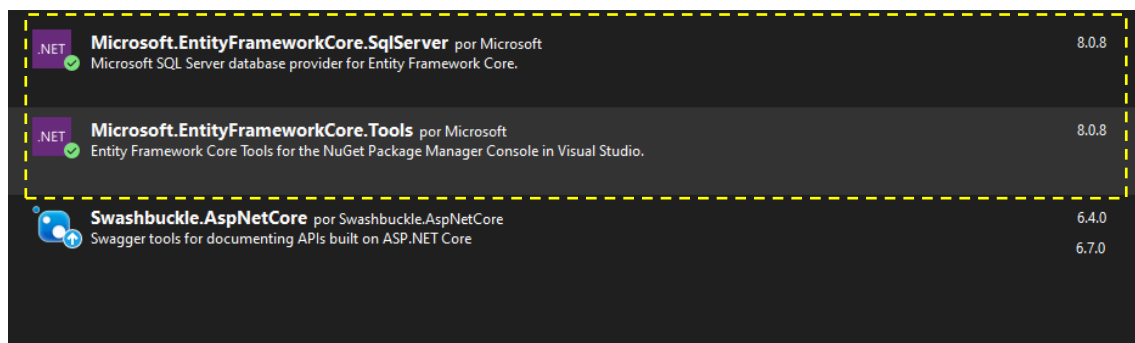


El proyecto se creará con la versión **Net Core 8.0.1** con la cual la carga de **NuGet** requeridos para el **ORM** CodeFirts son necesarios.



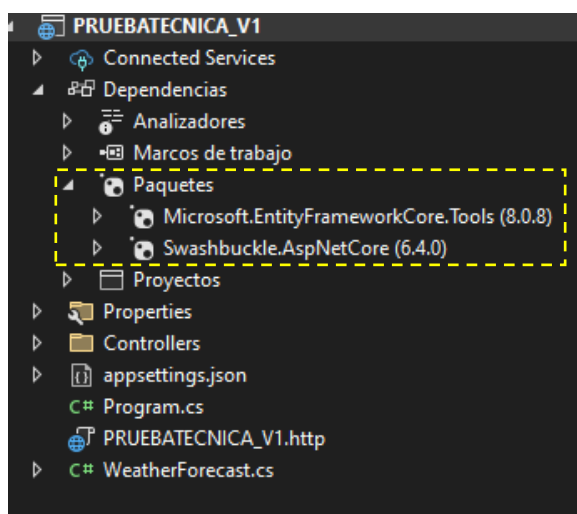
Vista proyecto creado con la estandarización de componentes genéricos creados para el aplicativo.

Se procede a instalar Paquetes NuGet para el uso de BD.

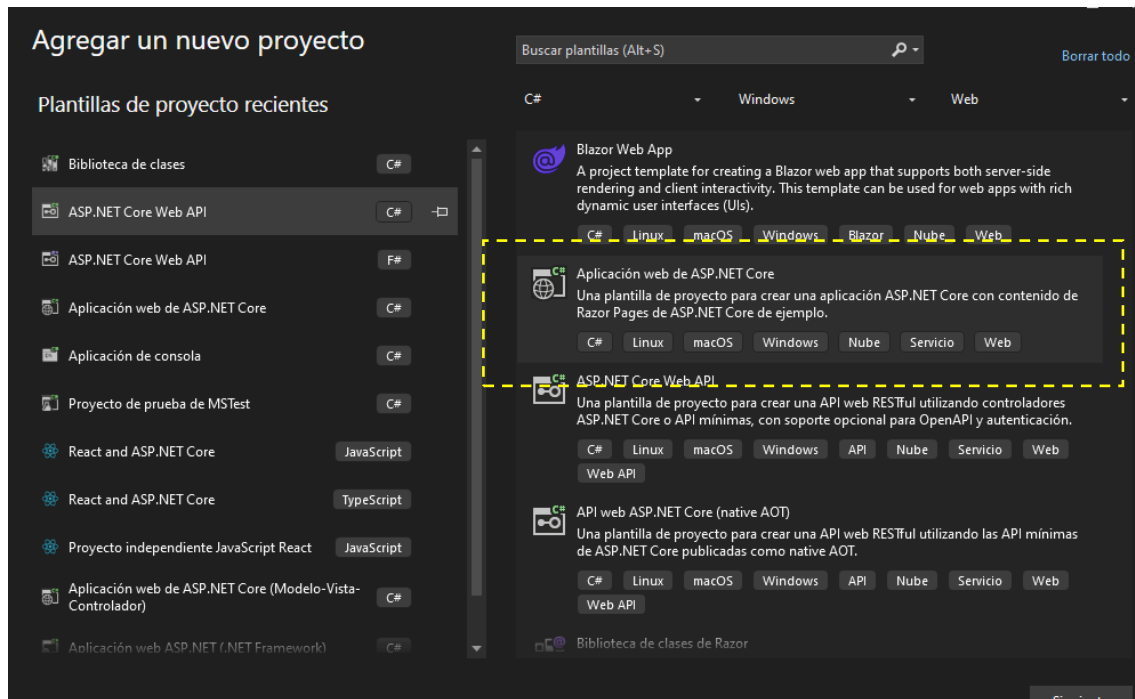


Se instalan los paquetes correspondientes ya sea por medio de la herramienta o por medio de la consola de Comandos predeterminada, tomar en cuenta que la versión sea compatible con la del proyecto no inferior a 8.0.1

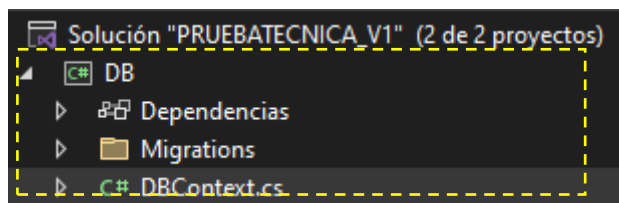
Vista dependencias donde el paquete se encuentra instalado correctamente en el Proyecto.



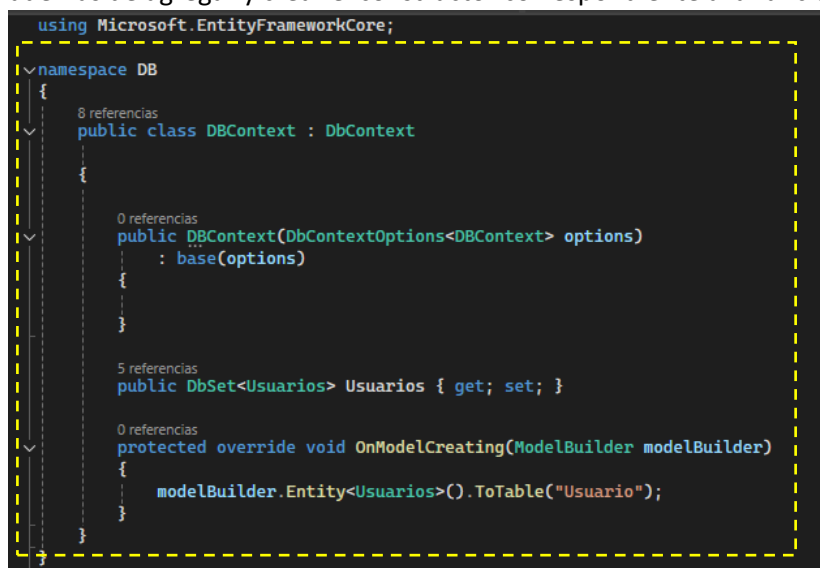
A continuación, se deben de crear un nuevo dentro de nuestra solución con el objetivo de generar nuestra Biblioteca de clases correspondiente a la conexión y creación de componentes necesarios para la ejecución del aplicativo.



Después solo nombramos nuestra clase creada dentro de nuestra Biblioteca.



Acto seguido proseguimos con la creación de nuestra clase correspondiente a la conexión además de agregar y crear el constructor correspondiente a la funcionalidad.



Cabe aclarar que los campos necesarios a la ejecución requieren la creación y la Instancia de la clase **DbContext**.

Después de crear el constructor correspondiente proseguimos a crear el **Modelo** encargado a determinar los campos necesarios para nuestra tabla, en este apartado se utilizó como ejemplo la Tabla Usuarios donde y en forma genérica solo se agregaron los datos **UsuarioID**, **Nombre**, **Apellido**, **Correo** y **Ocupación**. Con su respectivo tipo de dato, en este caso con **int** y con **String** para los Textos, además de poder agregar una pequeña validación de caracteres.

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DB
{
    5 referencias
    public class Usuarios
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        0 referencias
        public int UsuarioID { get; set; }

        [MaxLength(50, ErrorMessage = "El campo {0} debe tener maximo {1} caracteres.")]
        2 referencias
        public required string Nombre { get; set; }
        [MaxLength(50, ErrorMessage = "El campo {0} debe tener maximo {1} caracteres.")]
        2 referencias
        public required string Apellido { get; set; }
        [MaxLength(50, ErrorMessage = "El campo {0} debe tener maximo {1} caracteres.")]
        2 referencias
        public required string Correo { get; set; }
        [MaxLength(50, ErrorMessage = "El campo {0} debe tener maximo {1} caracteres.")]
        2 referencias
        public string Ocupacion { get; set; }
    }
}
```

También a funcionalidad para la ejecución y la creación de la tabla se utilizaron las palabras reservadas utilizadas para Modelo de datos en este caso para la generación de la Tabla Usuarios.

```
[Key]
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
```

Dato a tomar en cuenta es el auto incrementable el cual nos permite insertar Usuarios de forma consecutiva y 1,1.

A continuación, dentro de la Clase **Program** se crear el método correspondiente a la ejecución del **ORM CodeFirst**, donde claramente se permite realizar la ejecución y utilización de las palabras reservadas e identificables para el Framework utilizado.

```
using DB;
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Writers;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<DBContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("DBConnection"))
);
```

Cabe a aclarar que se Nombra el nombre de nuestra nueva cadena de conexión requerida para el uso del Motor **DB SQL Server**.

Dentro de la misma clase **Program** se establece también el código encargado a la ejecución de **CodeFirst** el cual por base de Comando ejecutar y realiza la migración de los datos de nuestro Modelo el cual se llama **Usuarios**.

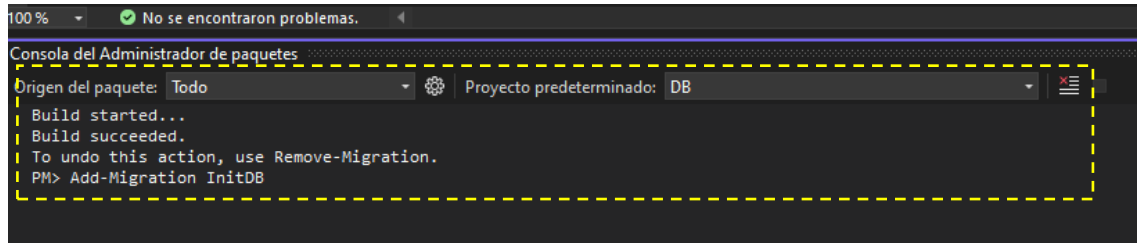
```
using (var scope = app.Services.CreateScope())
{
    var context = scope.ServiceProvider.GetRequiredService<DBContext>();
    context.Database.Migrate();
}
```

Se agrega la Cadena de conexión en el apartado **appsettings.json**

```
Esquema: https://json.schemastore.org/appsettings.json
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "AllowedHosts": "*",
9   "ConnectionStrings": {
10    "DBConnection": "server=(localdb)\\ALIENWARE; database=Bar; Trusted_Connection=true; integrated security=true;"
11  }
12 }
13
```

Se realizan pruebas con el compilado de la aplicación y por medio de la depuración y ejecución,

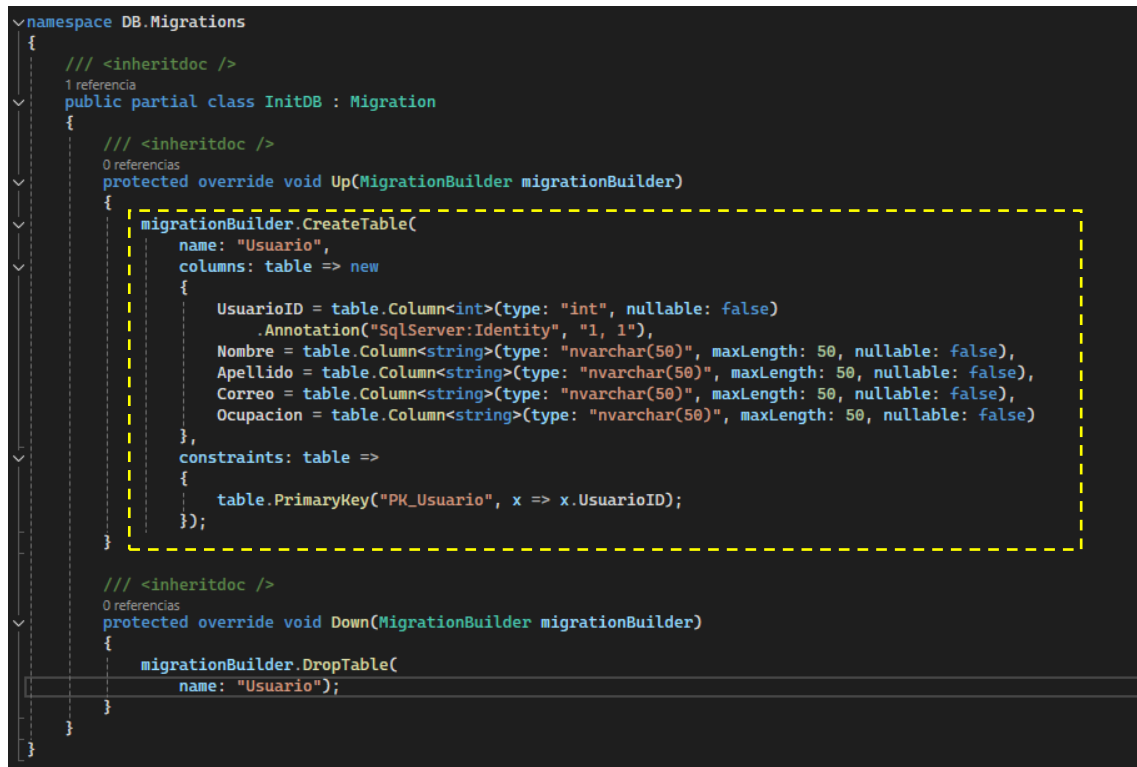
Después se realiza la selección del proyecto que en este caso sería **DB** y acto siguiente proseguimos a realizar la ejecución del comando **Add-Migrated initDB**



```
100 %  No se encontraron problemas.
Consola del Administrador de paquetes
Origen del paquete: Todo  Proyecto predeterminado: DB
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> Add-Migration InitDB
```

Al terminar se mostrará de forma automática la creación de la tabla con sus respectivos datos,

Ojo aún no se crea en **BD** para que esto suceda haya que ejecutar la solución de forma que este haga el registro y la inserción de la tabla **Usuarios**

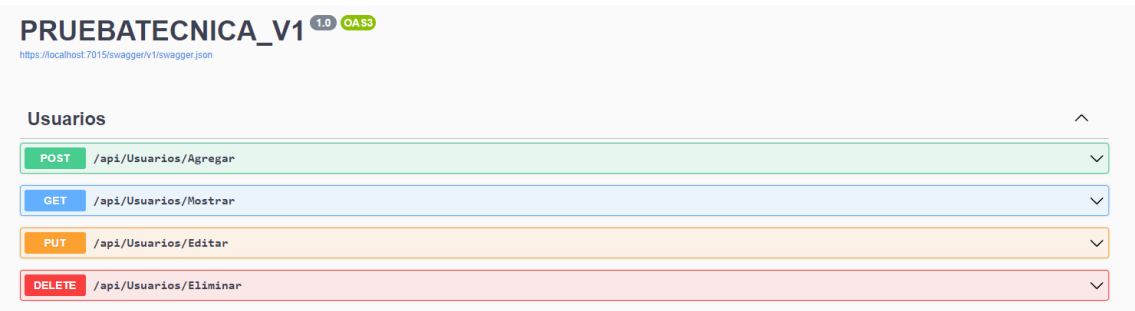


```
namespace DB.Migrations
{
    /// <inheritdoc />
    1 referencia
    public partial class InitDB : Migration
    {
        /// <inheritdoc />
        0 referencias
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Usuario",
                columns: table => new
                {
                    UsuarioID = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Nombre = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),
                    Apellido = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),
                    Correo = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),
                    Ocupacion = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Usuario", x => x.UsuarioID);
                });
        }

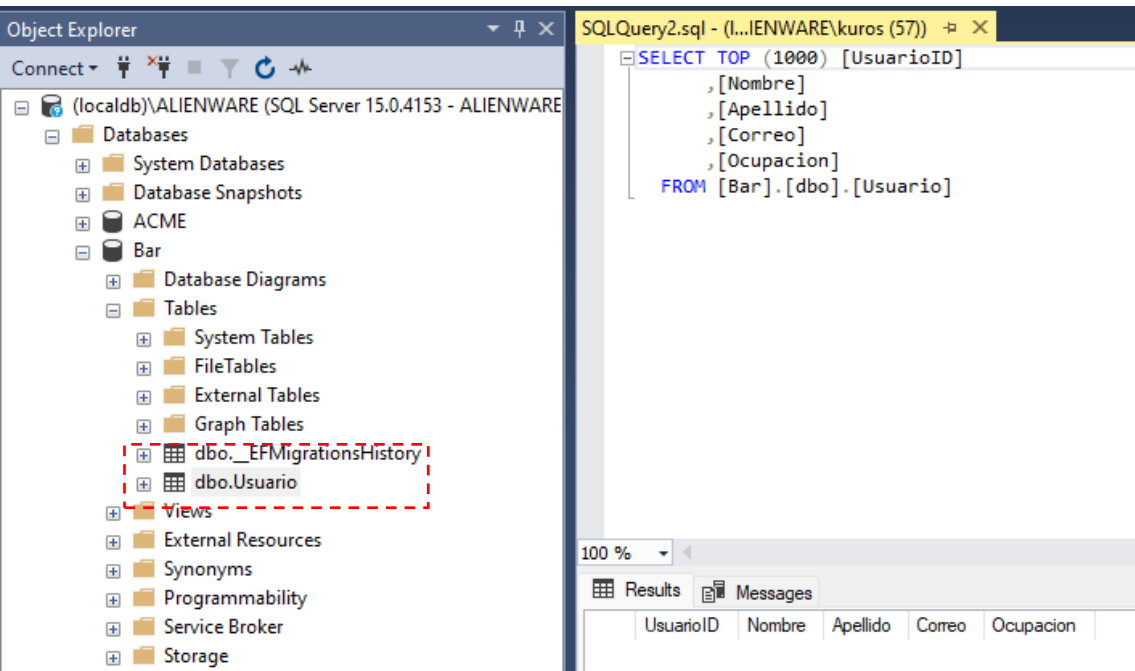
        /// <inheritdoc />
        0 referencias
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Usuario");
        }
    }
}
```

Al terminar lo mencionado y al hacer correr la solución ya tenemos dentro de Base de Datos nuestra tabla correspondiente a **Usuarios**.

Se muestra primera vista de nuestra tabla en Web API



Se muestra primera vista de nuestra tabla en la base de datos.



Usuario			
	Column Name	Condensed Type	Nullable
🔑	UsuarioID	int	No
	Nombre	nvarchar(50)	No
	Apellido	nvarchar(50)	No
	Correo	nvarchar(50)	No
	Ocupacion	nvarchar(50)	No

Cuando se halla realizado la primera ejecución cabe recordar que nuestro Método creado en la clase **program** esta aun habilitada en este caso para no generar creación de tabla Usuarios continua y que esta no nos elimine los datos que insertemos hay que solo comentar el Método.

```
//using (var scope = app.Services.CreateScope())
//{
//    var context = scope.ServiceProvider.GetRequiredService<DBContext>();
//    context.Database.Migrate();
//}
```

A continuación, proseguimos con la Creación de **Controlador** correspondiente a la funcionalidad del CRUD con el objetivo de realizar las peticiones Solicitadas para el **WEB API**.

```
namespace PRUEBATECNICA_V1.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UsuariosController : ControllerBase
    {
        private DBContext _Context;
        public UsuariosController(DBContext context)
        {
            _Context = context;
        }

        [HttpPost]
        [Route("Agregar")]
        public async Task<IActionResult> CrearUsuario(Usuarios usuario)
        {
            await _Context.Usuarios.AddAsync(usuario);
            await _Context.SaveChangesAsync();
            return Ok();
        }

        [HttpGet]
        [Route("Mostrar")]
        public async Task<ActionResult<IEnumerable<Usuarios>>> ListaUsuarios()
        {
            var usuarios = await _Context.Usuarios.ToListAsync();
            return Ok(usuarios);
        }

        [Route("Editar")]
        public async Task<IActionResult> ActualizarUsuario(int id, Usuarios usuarios)
        {
            var UsuarioExistente = await _Context.Usuarios.FindAsync(id);
            UsuarioExistente.Nombre = usuarios.Nombre;
            UsuarioExistente.Apellido = usuarios.Apellido;
            UsuarioExistente.Correo = usuarios.Correo;
            UsuarioExistente.Ocupacion = usuarios.Ocupacion;
            await _Context.SaveChangesAsync();
            return Ok();
        }

        [HttpDelete]
        [Route("Eliminar")]
        public async Task<IActionResult> EliminarUsuario(int id)
        {
            var UsuarioBorrado = await _Context.Usuarios.FindAsync(id);
            _Context.Usuarios.Remove(UsuarioBorrado);
            await _Context.SaveChangesAsync();
            return Ok();
        }
    }
}
```

Se realiza la inserción de datos como prueba dentro de **SQLServer** con el objetivo de mostrar datos y realizar las ejecuciones del **WEB API**.

Alienware\LOCALDB....Bar - dbo.Usuario					
Alienware\LOCALDB....8.Bar - Diagram_0*					
SQL					
	UsuarioID	Nombre	Apellido	Correo	Ocupacion
	1	Victor	Soto	victorsotovalen...	Desarrollador
	2	Elena	Diaz	elena@gmail.c...	Desarrollador
	3	Sebastian	Soto	Sebast@gmail....	UIX
	4	Violet	Evergarden	violet@gmail.c...	CEO
▶*	NULL	NULL	NULL	NULL	NULL

Se realizan las pruebas correspondientes al sistema

Método GET

200

Response body

```
[
  {
    "usuarioID": 1,
    "nombre": "Victor",
    "apellido": "Soto",
    "correo": "victorsotovalenci@gmail.com",
    "ocupacion": "Desarrollador"
  },
  {
    "usuarioID": 2,
    "nombre": "Elena",
    "apellido": "Diaz",
    "correo": "elena@gmail.com",
    "ocupacion": "Desarrollador"
  },
  {
    "usuarioID": 3,
    "nombre": "Sebastian",
    "apellido": "Soto",
    "correo": "Sebast@gmail.com",
    "ocupacion": "UIX"
  },
  {
    "usuarioID": 4,
    "nombre": "Violet",
    "apellido": "Evergarden",
    "correo": "violet@gmail.com",
    "ocupacion": "CEO"
  }
]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Thu, 15 Aug 2024 06:22:05 GMT
server: Kestrel
```

Se realiza la prueba y se muestra un estatus 200 Correcto de la consulta.

Se realiza prueba **Método PUT** para editar un registro

PUT

/api/Usuarios/Editar

Parameters

Name	Description
id	
integer(\$int32)	2
(query)	

Request body

```
{
  "usuarioID": 2,
  "nombre": "Prueba",
  "apellido": "Prueba2",
  "correo": "Prueba@gmail.com",
  "ocupacion": "Prueba de ejecucion"
}
```

Curl

```
curl -X 'PUT' \
'https://localhost:7015/api/Usuarios/Editar?id=2' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
  "usuarioID": 2,
  "nombre": "Prueba",
  "apellido": "Prueba2",
  "correo": "Prueba@gmail.com",
  "ocupacion": "Prueba de ejecucion"
}'
```

Request URL

https://localhost:7015/api/Usuarios/Editar?id=2

Server response

Code	Details
200	<div><div>Response headers</div><div>access-control-allow-origin: * content-length: 0 date: Thu, 15 Aug 2024 06:25:32 GMT server: Kestrel</div></div>

Responses

Code	Description
200	Success

Se muestra un estatus 200 correcto de acuerdo al **Método PUT**

Se realiza prueba **Método POST**

Usuarios

POST

/api/Usuarios/Agregar

Parameters

No parameters

Request body

```
{  "usuarioID": 10,  "nombre": "Nombre Prueba",  "apellido": "Prueba",  "correo": "Prueba@gmail.com",  "ocupacion": "Prueba 2"}
```

Se consulta el **Método POST** con el objetivo de mostrar el registro editado y el registro Agregado.

https://localhost:7015/api/Usuarios/Mostrar

Server response

Code	Details
200	<div>Response body<pre>{ "usuarioID": 1, "nombre": "Victor", "apellido": "Soto", "correo": "victorsotovalenci@gmail.com", "ocupacion": "Desarrollador"}, { "usuarioID": 2, "nombre": "Prueba", "apellido": "Prueba2", "correo": "Prueba@gmail.com", "ocupacion": "Prueba de ejecucion"}, { "usuarioID": 3, "nombre": "Sebastian", "apellido": "Soto", "correo": "Sebast@gmail.com", "ocupacion": "UIX"}, { "usuarioID": 4, "nombre": "Violet", "apellido": "Evergarden", "correo": "violet@gmail.com", "ocupacion": "CEO"}}</pre></div> <div>Response headers<pre>content-type: application/json; charset=utf-8 date: Thu, 15 Aug 2024 06:29:47 GMT server: Kestrel</pre></div>

Responses

Code	Description
------	-------------

Vista General Final aplicativo funcional.

