1. Crear proyecto ASP.NET Core Web API
2. Crear 3 Folders en el proyecto

* Data
* Entities
* Repositories

1. Crear la entidad ‘Product’ en la carpeta ‘Entities’ con las siguientes propiedades:

public class Product

{

public int ProductId { get; set; }

public string ProductName { get; set; }

public string ProductDescription { get; set; }

public int ProductPrice { get; set; }

public int ProductStock { get; set; }

}

1. Instalar los nugets:

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Tools
* Microsoft.EntityFrameworkCore.SQLServer
* Microsoft.EntityFrameworkCore.Design
* Microsoft.Data.SqlClient

1. Crear la clase ‘DbContextClass’ en la carpeta ‘Data’ así:

public class DbContextClass : DbContext

{

protected readonly IConfiguration Configuration;

public DbContextClass(IConfiguration configuration)

{

Configuration = configuration;

}

protected override void OnConfiguring(DbContextOptionsBuilder options)

{

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

}

public DbSet<Product> Product { get; set; }

}

1. Crear IProductService y ProductService en la carpeta ‘Repositories’ así:

public interface IProductService

{

public Task<List<Product>> GetProductListAsync();

public Task<IEnumerable<Product>> GetProductByIdAsync(int Id);

public Task<int> AddProductAsync(Product product);

public Task<int> UpdateProductAsync(Product product);

public Task<int> DeleteProductAsync(int Id);

}

public class ProductService : IProductService

{

private readonly DbContextClass \_dbContext;

public ProductService(DbContextClass dbContext)

{

this.\_dbContext = dbContext;

}

public async Task<List<Product>> GetProductListAsync()

{

var products = await this.\_dbContext.Product.FromSqlRaw<Product>("spGetProductList").ToListAsync();

return products;

}

public async Task<IEnumerable<Product>> GetProductByIdAsync(int productId)

{

var productIdParam = new SqlParameter("@productId", productId);

var product = await Task.Run(()=> this.\_dbContext.Product.FromSqlRaw(@"exec spGetProductById @productId", productIdParam).ToListAsync());

return product;

}

public async Task<int> AddProductAsync(Product product)

{

var paratemers = new List<SqlParameter>();

paratemers.Add(new SqlParameter("@productName", product.ProductName));

paratemers.Add(new SqlParameter("@productDescription", product.ProductDescription));

paratemers.Add(new SqlParameter("@productPrice", product.ProductPrice));

paratemers.Add(new SqlParameter("@productStock", product.ProductStock));

var result = await Task.Run(()=> this.\_dbContext.Database.ExecuteSqlRawAsync(@"exec spCreateProduct @productName, @productDescription, @productPrice, @productStock", paratemers.ToArray()));

return result;

}

public async Task<int> UpdateProductAsync(Product product)

{

var paratemers = new List<SqlParameter>();

paratemers.Add(new SqlParameter("@productId", product.ProductId));

paratemers.Add(new SqlParameter("@productName", product.ProductName));

paratemers.Add(new SqlParameter("@productDescription", product.ProductDescription));

paratemers.Add(new SqlParameter("@productPrice", product.ProductPrice));

paratemers.Add(new SqlParameter("@productStock", product.ProductStock));

var result = await Task.Run(() => this.\_dbContext.Database.ExecuteSqlRawAsync(@"exec spUpdateProduct @productId, @productName, @productDescription, @productPrice, @productStock", paratemers.ToArray()));

return result;

}

public async Task<int> DeleteProductAsync(int productId)

{

var result = await Task.Run(()=> this.\_dbContext.Database.ExecuteSqlInterpolatedAsync($"spDeleteProduct {productId}"));

return result;

}

}

1. Modificar el archivo ‘appsettings.json’ escribiendo la cadena de conexión

"ConnectionStrings": {

"ConexionSQLServer": "server=CESARIOS-YOGA;database=Test2;Trusted\_Connection=true;MultipleActiveResultSets=true;TrustServerCertificate=true;"

}

1. Modificar el archivo ‘Program.cs’ registrado los servicios.

builder.Services.AddScoped<IProductService, ProductService>();

builder.Services.AddDbContext<DbContextClass>();

1. Construir el controlador ‘Product’ así:

[Route("api/[controller]")]

[ApiController]

public class ProductController : ControllerBase

{

private readonly IProductService \_productService;

public ProductController(IProductService productService)

{

this.\_productService = productService;

}

[HttpGet("getProductList")]

public async Task<IActionResult> GetProductListAsync()

{

try

{

var products = await this.\_productService.GetProductListAsync();

return Ok(products);

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

[HttpGet("getProductById")]

public async Task<IActionResult> GetProductByIdAsync(int idProduct)

{

try

{

var product = await this.\_productService.GetProductByIdAsync(idProduct);

return Ok(product);

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

[HttpPost("createProduct")]

public async Task<IActionResult> CreateProductAsync(Product product)

{

try

{

if (product != null)

{

var result = await this.\_productService.AddProductAsync(product);

return Ok(result);

}

else

{

return BadRequest("El objeto no puede ser nulo");

}

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

[HttpPut("updateProduct")]

public async Task<IActionResult> UpdateProductAsync(Product product)

{

try

{

if (product != null)

{

var result = await this.\_productService.UpdateProductAsync(product);

return Ok(result);

}

else

{

return BadRequest("El objeto no puede ser nulo");

}

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

[HttpDelete("deleteProduct")]

public async Task<IActionResult> DeleteProductAsync(int idProduct)

{

try

{

var result = await this.\_productService.DeleteProductAsync(idProduct);

return Ok(result);

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

}

1. Ejecutar la migración desde el ‘Package Manager Console’

* add-migration "Initial"
* update-database

1. Crear Los procedimientos almacenados

CREATE OR ALTER PROCEDURE [dbo].[spGetProductList]

AS

BEGIN

SELECT \* FROM dbo.Product

END

CREATE OR ALTER PROCEDURE [dbo].[spGetProductById]

@ProductId int

AS

BEGIN

SELECT

ProductId,

ProductName,

ProductDescription,

ProductPrice,

ProductStock

FROM dbo.Product where ProductId = @ProductId

END

CREATE OR ALTER PROCEDURE [dbo].[spCreateProduct]

@ProductName [nvarchar](max),

@ProductDescription [nvarchar](max),

@ProductPrice int,

@ProductStock int

AS

BEGIN

INSERT INTO dbo.Product

(

ProductName,

ProductDescription,

ProductPrice,

ProductStock

)

VALUES

(

@ProductName,

@ProductDescription,

@ProductPrice,

@ProductStock

)

END

CREATE OR ALTER PROCEDURE [dbo].[spUpdateProduct]

@ProductId int,

@ProductName [nvarchar](max),

@ProductDescription [nvarchar](max),

@ProductPrice int,

@ProductStock int

AS

BEGIN

UPDATE dbo.Product

SET

ProductName = @ProductName,

ProductDescription = @ProductDescription,

ProductPrice = @ProductPrice,

ProductStock = @ProductStock

WHERE ProductId = @ProductId

END

CREATE OR ALTER PROCEDURE [dbo].[spDeleteProduct]

@ProductId int

AS

BEGIN

DELETE FROM dbo.Product where ProductId = @ProductId

END

1. Hacer las peticiones con Postman así:

* <http://localhost:5147/api/Product/getProductList>
* <http://localhost:5147/api/Product/getProductById?idProduct=3>
* <http://localhost:5147/api/Product/createProduct>

{

    "ProductId":"0",

    "ProductName":"Product 4",

    "ProductDescription":"Description 4",

    "ProductPrice":"4",

    "ProductStock":"4"

}

* <http://localhost:5147/api/Product/updateProduct>

{

    "ProductId":"3",

    "ProductName":"Product tres",

    "ProductDescription":"Description tres",

    "ProductPrice":"33",

    "ProductStock":"33"

}

* <http://localhost:5147/api/Product/deleteProduct?idProduct=3>

1. Crear una nueva tabla (Entidad)
2. Crear la clase en la carpeta ‘Entities’ con todas sus propiedades.
3. Agregar el respectivo DbSet en la clase ‘DbContextClass’ :

* public DbSet<Vehicle> Vehicle { get; set; }

1. Agregar la interfaz de servicio y la clase de servicio en la carpeta ‘Repositories’ y realizar su correspondiente implementación

* IVehicleService.cs
* VehicleService.cs

1. Agregar el servicio en la clase ‘Program.cs’:

* builder.Services.AddScoped<IVehicleService, VehicleService>();

1. Ejecutar la migración desde el ‘Package Manager Console’

* add-migration "Add vehicle entity"
* update-database