SUSTENTACIÓN DE URBANO TINEO, PABLO JAMIRO

**Desarrollado en C# con .Net Core**

**Problema 1. Solución:**

La solución fue desarrollada con la plantilla de Proyecto Web Api.

Para la creación de la base de datos en memoria hice uso del administrador de paquetes Nuget, instalando Microsoft.EntityFrameworkCore.InMemory.

Después creé la carpeta “db” donde puse los archivos palabras.100.txt y palabras.txt, llamada “db”, porque sus archivos los usaremos para alimentar la base de datos en memoria.

A continuación, en la carpeta Models creé las clases:

* Proveedor: con sus atributos id y nombre, y un constructor público:

Proveedor (id, nombre).

public class Proveedor

{

        public int id {get; set;}

        public string name {get; set;}

        public Proveedor (int id, string name){

            this.id= id;

            this.name = name;

        }

    }

* ProveedorContext: encargada del almacenamiento de datos de cierta entidad.

    public class ProveedorContext : DbContext

    {

public ProveedorContext(DbContextOptions<ProveedorContext> options): base (options){

        }

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

    }

En el archivo Startup.cs, en su método de configuración de servicios (ConfigureServices) agregué al parámetro “services” el DbContext de Proveedor, haciendo uso de lambda para la inicialización de este contexto en memoria:

services.AddDbContext<ProveedorContext>(ops => ops.UseInMemoryDatabase(“ProveedorDB”));

Luego de haberse creado el ProveedorController heredando de ControllerBase:

public class ProveedorController: ControllerBase

* Declaré un objeto de privado constante de tipo ProveedorContext llamado “\_context”:

private readonly ProveedorContext \_context;

* Crear el constructor de ProveedorController con un parámetro de tipo ProveedorContext:

   public ProveedorController (ProveedorContext context)

   {

       \_context = context;

   }

* Declaré la variable “leído” de tipo booleano inicializado en false, más adelante veremos su uso:

static bool leido = false;

* Desarrollé el método CargaDatos() de tipo void que consta de los siguientes pasos:

1. Un arreglo de strings para leer cada fila de archivo palabras.txt de la siguiente forma:

string[] líneas = System.IO.File.ReadAllLines(@”db\palabras.txt”);

1. Un contador inicializado con 1:

int contador = 1;

1. Hacer uso de un foreach para leer cada línea guarda en el arreglo “líneas”, que construirá un Proveedor a la vez haciendo uso del contador antes creado usándolo como id, por consiguiente, aumentando el contador en uno, por último, agregando el Proveedor al \_context, todo esto de la siguiente manera:

foreach(string linea in lineas){

var reg = new Proveedor(contador, linea);

contador++;

\_context.Proveedor.Add(reg);

}

1. Por último, guardando los cambios:

\_context.SaveChanges();

* Este método lo estoy ejecutando cuando se construye por primera vez el ProveedorController al llamarse mediante la api, evitando que se construya por segunda vez con la variable “leído”, de la siguiente manera:

public ProveedorController(ProveedorContext context)

   {

     \_context = context;

     if(leido==false){

        CargaDatos();

        leido= true;

     }

   }

**Problema 3. Solución:**

La arquitectura usada va enfocada a la arquitectura MVC, creando las entidades y Contextos en la carpeta Models; los controladores, que servirán como puntos de llamada para los endpoints del api, en Controllers; y en este caso como no es una aplicación web no se incluye las vistas, pero si documentación sobre cómo usar la api. Hago uso de está arquitectura porque la considero que es ordenada y de fácil entendimiento.