Arquite-ctura

BackEnd

APPINVENTORY

Introducción

La arquitectura del sistema APPInventory está conformado por cinco capas (WebApiExam_API, Models, Aplication, Infrastructure, DataAccess).

A continuación, vamos a realizar una descripción a grandes rasgos de cada una.

API

Esta capa está encargada de recepción de peticiones http y envió de respuestas http, a su vez se definen los métodos expuestos que la capa de Front-End podrá visualizar.

Esta capa soporta las siguientes peticiones:

API	Descripción	Cuerpo de petición	Cuerpo de respuesta	Variables
Get api/[action]	Petición para obtener registros de la BD.	N/A	Objeto que contiene registros y mensaje con resultado de la operación.	N/A
Post api/[action]	Petición para insertar registro en la BD.	Objeto que contiene el registro a insertar.	Objeto que contiene mensaje con resultado de la operación.	Contiene una o más variables
Put api/[action]	Petición para actualizar registro en la BD.	Objeto que contiene el registro a actualizar.	Objeto que contiene mensaje con resultado de la operación.	Contiene una o más variables

Delete api/[action]	Petición para eliminar resgistros de la BD.	Objeto que contiene el registro a eliminar.	Objeto que contiene mensaje con resultado de la operación.	Contiene una o más variables

Definición típica de clase Api (Ejemplo):

```
using System;                                          //Definición de librerías a utilizar
using Microsoft.AspNetCore.Mvc;
using WebApiExam_API.Models;
using System.IO;
using Microsoft.AspNetCore.Hosting;
using WebApiExam_API.Repository;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using WebApiExam_API.Interfaces;


namespace WebApiExam_API.Controllers    //Namespace, colocar en su categoría
{
    [Route("api/[controller]")] // Se define la ruta que estará expuesta en red.
    [ApiController]       // Se defina para indicar que soporta HTTP API
    public class ProductsController : Controller
    {
        private readonly IWebHostEnvironment _env;
        private readonly IProductsRepository _product; // Instancia de capa de Repository que conecta al DBcontext mediante EntityFramework

        public ProductsController(IWebHostEnvironment env, IProductsRepository Product)
        {
            _env = env;
            _product = Product ?? throw new ArgumentNullException(nameof(Product)); // Instancia para consumir API
        }

        //Definición de API (get, post,put, delete)
        [HttpGet]
        [Route("GetProducts")]          // Se define la ruta que estará expuesta en red.
        public async Task<IActionResult> Get()
        {
            return Ok(await _product.GetProducts());
        }

        //Definición de API (get, post,put, delete) con cuerpo de objeto, para subir archivos
        [HttpPost]
        [Route("AddProduct")]
        public async Task<IActionResult> Post(Products prod)
        {

            var result = await _product.InsertProduct(prod);
            if (result == null)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "Something Went Wrong");
            }
```

```
            return Ok("Added Successfully");
        }


        //Definición de API (get, post,put, delete) con variables
        [HttpDelete("{id}")]
        public JsonResult Delete(int id)
        {
            var result = _product.DeleteProduct(id);
            return new JsonResult("Deleted Successfully");
        }



    }
      }
```

Infrastructure

Contiene la lógica de negocio de la aplicación, aquí es el punto de interacción entre las llamadas desde los controladores del front y las llamadas necesarias del repositorio.

Definición de la capa Infrastructure:

REPOSITORY

```
using INVENTORY.Aplication;
using INVENTORY.Aplication.Aplication;
using System;
using System.Collections.Generic;       //Definición de librerías a utilizar
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using INVENTORY.DataAccess;
using INVENTORYModels.Modelos;
using Dapper;
using System.Data;
using INVENTORYModels.Comun;

namespace INVENTORY.Infrastructure.Repository//Namespace, colocar en su categoría
{
    public class ProductsWebRepository : IProductsWebRepository
    {

        private IDataAccess _Context { get; }

        public ProductsWebRepository(IDataAccess context) //Inyecta el DbContext para que accesa a la capa de datos

        {
            _Context = context;
              }
          //Definición de métodos de negocio
  public async Task<string> AddAsync(TbProducts entity)
        {
             //Declaración de parámetros que necesita el SP
            var paramdata = new
            {
                Name = entity.Name,
             Category = entity.Category,
```

```
            Color = entity.Color,
            UnitPrice = entity.UnitPrice,
            AvailableQuantity = entity.AvailableQuantity            };
```

//Llamado a la capa de repositorio, cualquier tipo de operación (select, insert, update, delete)

//Aquí se pueden manipular todo tipo de reglas de negocio,

```
            var result = await this._Context.EjecutaAsync("Insert_Product", paramdata);
            return result.ToString();
              }
```

UNIT OF WORK

```
using INVENTORY.Aplication.Aplication;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace INVENTORY.Infrastructure.Repository
{
    public class UnitOfWork : IUnitOfWork
    {
        public UnitOfWork(IProductsWebRepository ProductsWebRepository)
```

//Inyecta el Repositorio para que pueda realizar las operaciones de consulta, inserción, actualización y eliminación de registros.

```
        {
            ProductsWeb = ProductsWebRepository;
        }

        public IProductsWebRepository ProductsWeb { get; set; }



    }
      }
```

ServiceCollectionExtension

Se hace la dependencia de inyección a todos los servicios que se van a utilizar.

```
using INVENTORY.Aplication.Aplication;
using INVENTORY.DataAccess.Context;
using INVENTORY.DataAccess;
using INVENTORY.Infrastructure.Repository;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace INVENTORY.Infrastructure
{
    public static class ServiceCollectionExtension
    {
        public static void RegisterServices(this IServiceCollection services, IConfiguration
configuracion)
```

```
        {
            string connectionString = configuracion.GetConnectionString("DefaultConnection");

services.AddTransient<IDapperContext, DapperContext>(c => new
DapperContext(connectionString));
services.AddTransient<IDataAccess, SqlDataAccess>();
services.AddTransient<IProductsWebRepository, ProductsWebRepository>();

services.AddTransient<IUnitOfWork, UnitOfWork>();
        }


    }
      }
```

Aplication

Contiene las interfaces. Estas interfaces incluyen abstracciones para las operaciones que se llevarán a cabo mediante la infraestructura.

```
using INVENTORYModels.Comun;
using INVENTORYModels.Modelos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace INVENTORY.Aplication.Aplication
{
    public interface IRepository<T> where T : class
    {
         Task<string> AddAsync(T entity);

 List<TbProducts> GetAllSync();
```

DataAccess

Contiene las interfaces e implementaciones de acceso a datos, estas implementaciones incluyen DbContext con Dapper y Microsoft.Data.SqlClient.

DbContext Microsoft.Data.SqlClient

```
using INVENTORY.DataAccess.Context;
using Dapper;
using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace INVENTORY.DataAccess;

    public class SqlDataAccess : IDataAccess
    {

        private readonly IDapperContext context;
```

```
        public SqlDataAccess(IDapperContext context)
        {
            this.context = context;

        }

        public async Task<int> EjecutaAsync(string query, object param, CommandType
tipoComando = CommandType.StoredProcedure)
        {
            using SqlConnection cn = (SqlConnection)this.context.CreateConnection;

            return await cn.ExecuteAsync(query, param, commandType: tipoComando);
              }
```

DbContext Dapper

```
using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace INVENTORY.DataAccess.Context
{
    public class DapperContext : IDapperContext
    {

        private readonly string _connectionString;
        private IDbConnection _connection;

        public DapperContext(string cadenadeConexion)
        {
            _connectionString = cadenadeConexion;
        }

        public IDbConnection CreateConnection
        {
            get
            {
                _connection = new SqlConnection(_connectionString);
                return _connection;
            }
        }
        public void Dispose()
        {
            if (_connection != null && _connection.State == ConnectionState.Open)
            {
                _connection.Close();
            }
        }



    }
      }
```

Models

Esta capa contiene todas las entidades asociadas a la base de datos, es necesario definir algunos atributos y anotaciones para integrar el funcionamiento con Dapper.

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Inventory.Models.Models //Namespace, colocar en su categoría
{
    public class Products
    {
        [Key]       ] //Especificar que campo se tiene como llave primaria
        public int ProductId { get; set; }    //Campos necesarios
        [Required]
      [DataType(DataType.Text)]
      public string Name { get; set; }
      [Required]
      [DataType(DataType.Text)]
      public string Category { get; set; }
      [Required]
      [DataType(DataType.Text)]
      public string Color { get; set; }
      [Required]
      [DataType(DataType.Currency)]
      public decimal UnitPrice { get; set; }
      public int AvailableQuantity { get; set; }

    }
}
```