Desarrollo de Web APIs con ASP:NET Core.

Prerrequisitos: Iniciar Visual Studio y realizar un clic en crear proyecto

| En la siguiente imagen seleccione la opción “ASP:NET Core Web API”, y realice un clic en siguiente. |  |
| --- | --- |
| Colocar un nombre, ejemplo: “apiServicio” y realizar un clic en el botón siguiente. |  |
| Seleccione el framework de destino y realice un clic en “Crear”. |  |
| Visual Studio debería mostrar la siguiente interface. Realice un clic en el botón “IIS Express” o presione la tecla “f5” para iniciar la aplicación. |  |
| Se debería iniciar un navegador que muestre la siguiente pantalla. Realice un clic en el botón GET |  |
| Realice un clic en el botón Try it out |  |
| Realice un clic en el botón “Execute”. |  |
| Recordemos algunos tipos de respuesta:  200 OK:  Descripción: solicitud exitosa.  400 Bad Request:  Descripción: Indica un error del lado del cliente, como parámetros de solicitud incorrectos.  401 Unauthorized:  Descripción: Se requiere una autenticación valida.  500 Internal Server Error:  Descripción: error interno del servidor. |  |
| Archivo “appsettings.json”, para esta etapa lo que haremos es crear la cadena de conexión a mi motor de Base de datos que para esta ocasión utilizaremos SQL Server. |  |
| La estructura de carpetas que utilizaremos será la siguiente:  Models: Contendrá las clases y se definirá sus atributos  Business: Contendrá las clases que se comuniquen con la Base de Datos.  Services: Contendrá las clases que se comunicaran con las clases de Negicio.  Controllers: Contendrá las clases que podrán ser consultadas por el usuario y/o aplicaciones. |  |
| Para el ejemplo práctico se utilizara altas y bajas de la tabla Rol: | CREATE TABLE [dbo].[Rol](  [Id] [int] IDENTITY(1,1) NOT NULL,  [NombreRol] [nvarchar](max) NULL,  ) ON [PRIMARY]  GO |
| Realizaremos la creación de la clase para los atributos en la Carpeta Models |  |
| La clase contendrá solo 2 atributos | namespace apiServicio.Models  {  public class Rol  {  public int Id { get; set; }  public string NombreRol { get; set; }  }  } |
| Ejemplo de creación de clases:  Clic derecho en el lugar donde requerimos crear la clase->Agregar->Clase |  |
| Para este ejemplo crearemos una clase con el termino Repository.  Esta clase será la que se comunicará directamente con la Base de Datos. |  |
| Repetiremos el paso de crear clase, pero esta vez crearemos la interface: IRolRepository.cs |  |
| Una vez creada la clase y su repositorio, lo que haremos es crear el constructor y hacer referencia a la cadena de conexión de Base de datos.  Nota: tomar en cuenta que se hereda de la interface IRolRepository.  public class RolRepository : IRolRepository |  |
| En diversas partes del código Visual Studio cuenta con un asistente el cual nos brinda opciones para solucionar los errores de código, en este ejemplo movemos el mouse donde indica el error, realizamos un clic derecho y contamos con las opciones. |  |
| En la Clase RolRepositore.cs, contaremos con 2 métodos inicialmente:  1.- GetList()  Método que devuelve la lista de todos los registros de roles en la Base de datos  2.- AgregaActualiza(Rol l, string t)  Método que agrega o actualiza el registro de un rol, tomando como parámetro una variable t(c=create,u=update). |  |
| En el archivo “IRolRepository.cs”, debe contener el nombre de los métodos, sin la aclaración de si es un método público o privado, así también debe terminar con un “;”, y no cuenta con un constructor. | using apiServicio.Models;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  namespace apiServicio.Business.Contracts  {  public interface IRolRepository  {  Task<List<Rol>> GetList();  Task<Rol> AgregaActualiza(Rol l, string t);  }  } |
| Código Clase: RolService.cs | public class RolService : IRolService  {  private readonly IRolRepository \_IRolRepository;  public RolService(IRolRepository tempI)  {  \_IRolRepository = tempI;  }  public Task<List<Rol>> GetList()  {  return \_IRolRepository.GetList();  }  public Task<Rol> AgregaActualiza(Rol l, string t)  {  return \_IRolRepository.AgregaActualiza(l, t);  }  } |
| Código Interface: IRolService.cs | public interface IRolService  {  Task<List<Rol>> GetList();  Task<Rol> AgregaActualiza(Rol l, string t);  } |
| Creamos la clase:  RolController.cs  En la carpeta  Controllers | [ApiController]  [Route("api/[controller]")]  public class RolController  {  private readonly IRolService \_IRolService;  public RolController(IRolService iTemp)  {  this.\_IRolService = iTemp;  }  [HttpGet]  public async Task<List<Rol>> GetList()  {  return await \_IRolService.GetList();  }  [HttpPost("AgregaActualiza")]  public async Task<Rol> AgregaActualiza(  int Id,string NombreRol, string t)  {  Rol l = new Rol();  l.Id = Id;  l.NombreRol = NombreRol;  return await \_IRolService.AgregaActualiza(l, t);  }  } |
| Finalmente en la clase  Startup.cs  Agregamos la comunicación entre al interface y las clases que utilizamos | program.cs  var builder = WebApplication.CreateBuilder(args);  // Add services to the container.  builder.Services.AddControllers();  builder.Services.AddTransient<IRolRepository, RolRepository>(); // Registro de la dependencia  builder.Services.AddTransient<IRolService, RolService>(); // Registro de la dependencia |
| Presione f5 |  |