**ASP. NET CORE WEB API**

1. Rename the folder name as “ContactCrud.Api”
2. Now to run project as microservices

* Select solution – and add new empty class

1. ContactCrud.Core

* Make Entity folder inside and define schema

A screenshot of a computer program

Description automatically generated

**Entities-> Contact.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ContactCrud.core.Entities

{

public class Contact

{

public int ContactId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public string PhoneNumber { get; set; }

}

}

1. ContactCrud.Infrastructure

DATA LAYER -> Sql server connection will done here

* Make a Repository folder inside

A screenshot of a computer program

Description automatically generated

Inside ContactRepository.cs : Make Interface in application folder and extend that interface here.

Do Dependency Injection with Constructor

ContactRepository.cs :

using CleanArch.Sql.Queries;

using ContactCrud.Application.Interfaces;

using ContactCrud.core.Entities;

using Dapper;

using Microsoft.Extensions.Configuration;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ContactCrud.Infrastructure.Repository

{

public class ContactRepository: IContactRepository

{

private readonly IConfiguration configuration;

public ContactRepository(IConfiguration configuration)

{

this.configuration = configuration;

}

public async Task<List<Contact>> GetAllAsync()

{

using (IDbConnection connection = new SqlConnection(configuration.GetConnectionString("DBConnection")))

{

connection.Open();

var result = await connection.QueryAsync<Contact>(ContactQueries.AllContact);

return result.ToList();

}

}

}

}

IContactRepository.cs :

using ContactCrud.core.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ContactCrud.Application.Interfaces

{

public interface IContactRepository

{

Task<List<Contact>> GetAllAsync();

}

}

1. ContactCrud.Application

* Make Interfaces folder inside

IContactRepository.cs:

using ContactCrud.core.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ContactCrud.Application.Interfaces

{

public interface IContactRepository

{

Task<List<Contact>> GetAllAsync();

}

}

IContactService.cs:

using ContactCrud.core.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ContactCrud.Application.Interfaces

{

public interface IContactService

{

Task<List<Contact>> GetAllContacts();

}

}

1. ContactCrud.Services

* Make Services folder inside

ContactService.cs

Inherit service Interface

Initialise the IContactRepository Interface with the help of constructor

Make definition of IContactService Interface and inside that call repository data.

**ContactService.cs**

using ContactCrud.Application.Interfaces;

using ContactCrud.core.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ContactCrud.Services.Services

{

public class ContactService: IContactService

{

private readonly IContactRepository contactRepository;

public ContactService(IContactRepository contactRepository) {

this.contactRepository = contactRepository;

}

public Task<List<Contact>> GetAllContacts()

{

var contacts = contactRepository.GetAllAsync();

return contacts;

}

}

}

1. Mapping of Repository and service

Inside Infrastructure folder -> ServiceCollectionExtension.cs

A screen shot of a computer program

Description automatically generated

F) Controller

Make routes here and initialise the service layer using dependencies injection through constructor

using ContactCrud.Api.Models;

using ContactCrud.Application.Interfaces;

using ContactCrud.core.Entities;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System.Data.SqlClient;

namespace ContactCrud.Api.Controllers

{

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

[ApiController]

public class ContactsController : ControllerBase

{

private readonly IContactService contactService;

public ContactsController(IContactService contactService)

{

this.contactService = contactService;

}

[HttpGet]

public async Task<ApiResponse<List<Contact>>> GetAllContacts()

{

var apiResponse = new ApiResponse<List<Contact>>();

try

{

var contacts = await contactService.GetAllContacts();

apiResponse.Success = true;

apiResponse.Result = contacts.ToList();

}

catch (SqlException ex)

{

apiResponse.Success = false;

apiResponse.Message = ex.Message;

}

catch (Exception ex)

{

apiResponse.Success = false;

apiResponse.Message = ex.Message;

}

return apiResponse;

}

}

}

G) Models

ApiResponse.cs

namespace ContactCrud.Api.Models

{

public class ApiResponse<T>

{

public bool Success { get; set; }

public string Message { get; set; }

public T Result { get; set; }

}

}

Mapper

<https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/automapper-in-c-sharp>

* + Using autoMapper