fl

Chandradev's Blog

Asp.net MVC Core,WebAPI,C#,Sql Server,EF Core,Angular,Javascript,Blazor

Creating Web Api Core 3.0 layer using Dapper and .net standard 2.0

December 12, 2019December 13, 2019 Chandradev

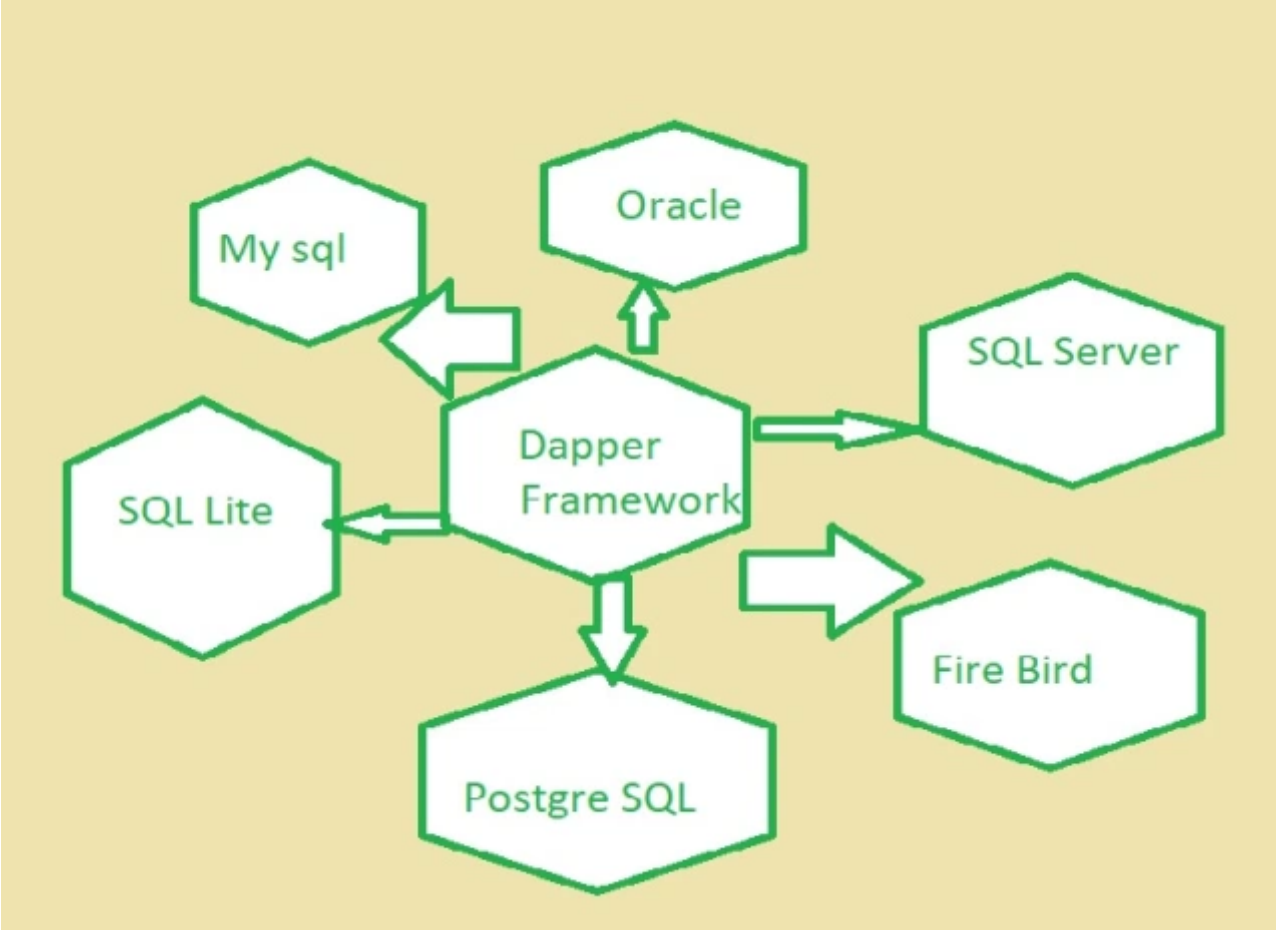
i
Rate This

Recently I got a chance to work with dapper using .net standard 2.0 So i thought to write my learning with you in this small blog post.

Firstly we will know what is the dapper ?

Dapper is micro ORM(Object Relational Mapper) framework which helps to map database directly with C# object. It is developed by StackOverflow team and released as open source.

Advantages of Dapper



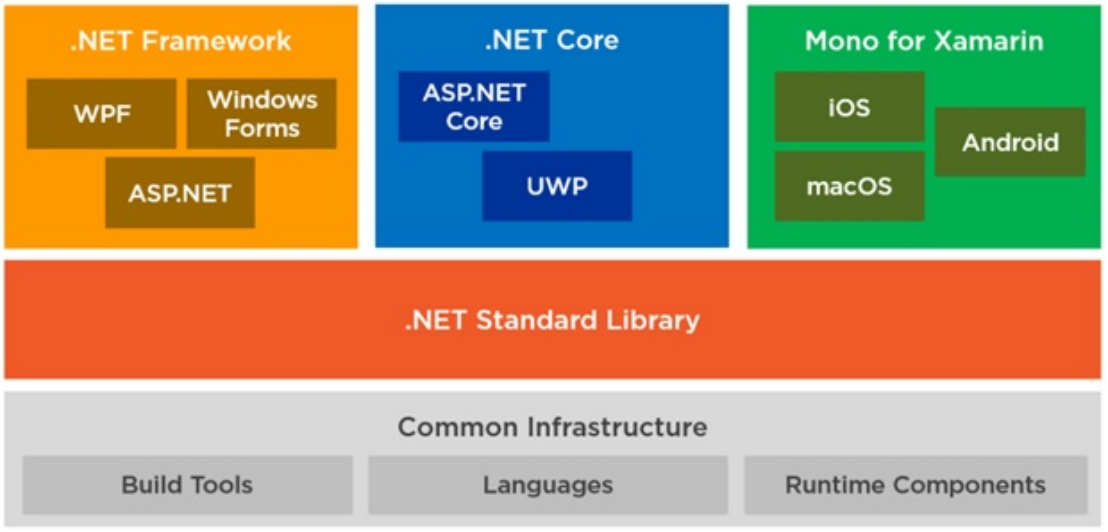
(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/dapper/)

1. It is a high-performance data access system
2. It is database independent
3. Same code work with all relational database.
4. Fewer lines of code.
5. Easy Handling of SQL Query.
6. Easy Handling of Stored Procedure.
7. Dapper also allows fetching multiple data based on multiple inputs.
8. Get generic result for simple or complex data type.
9. ease of use
10. It provides support for both static and dynamic object binding using transactions, stored procedures, or bulk inserts of data.

In this post I have also used .net standard 2.0, so we have to know what is the advantage of this

Advantages of .net standard 2.0

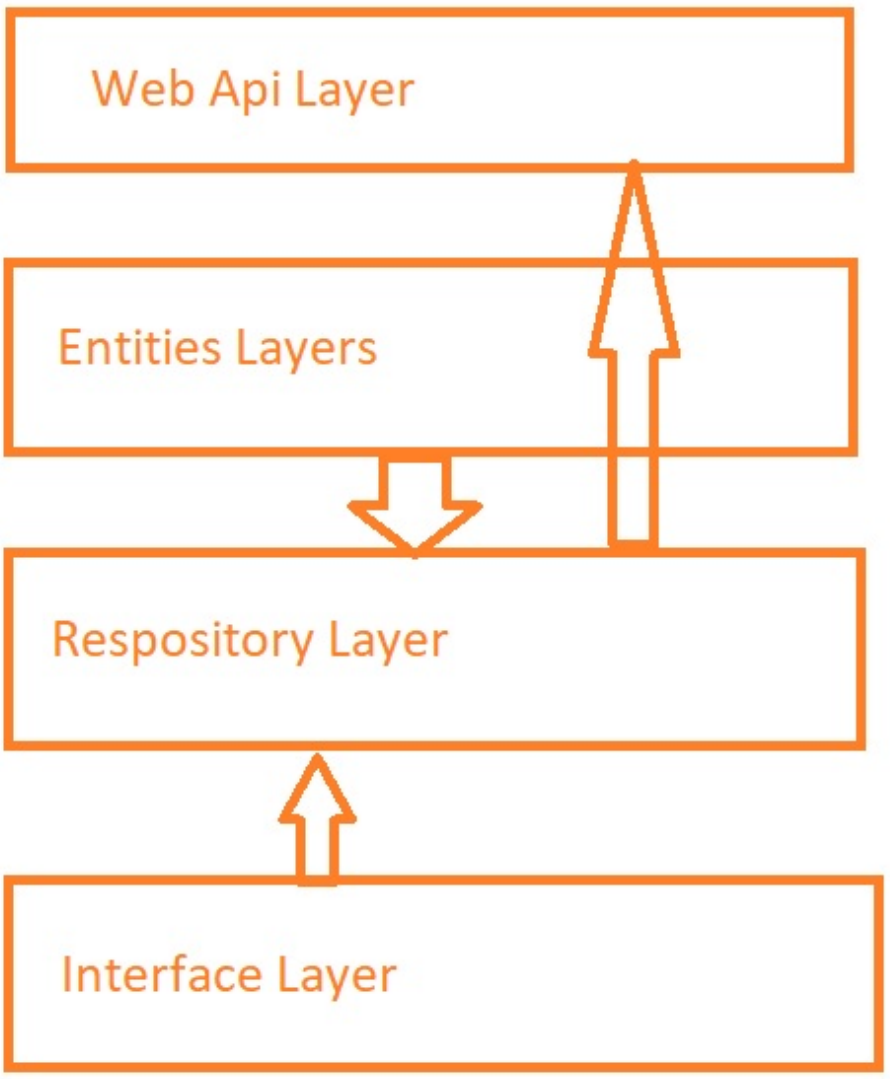
>> Same code can be shared on all type of .net application. This will be portable. This means you can write and can use in applications that run on multiple platforms. Their purpose is to share code between applications.



(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/netstandard/)

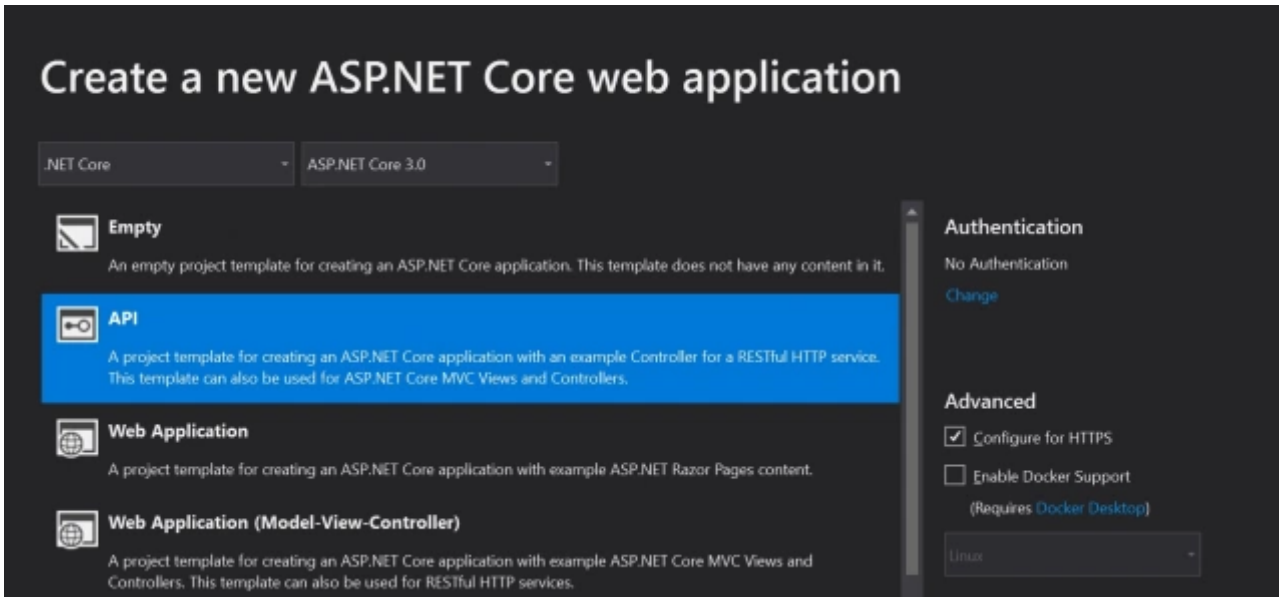
Now we will see how to implement in asp.net core web api

In this demo I m going to design the simple demo web api as given below simple architecture



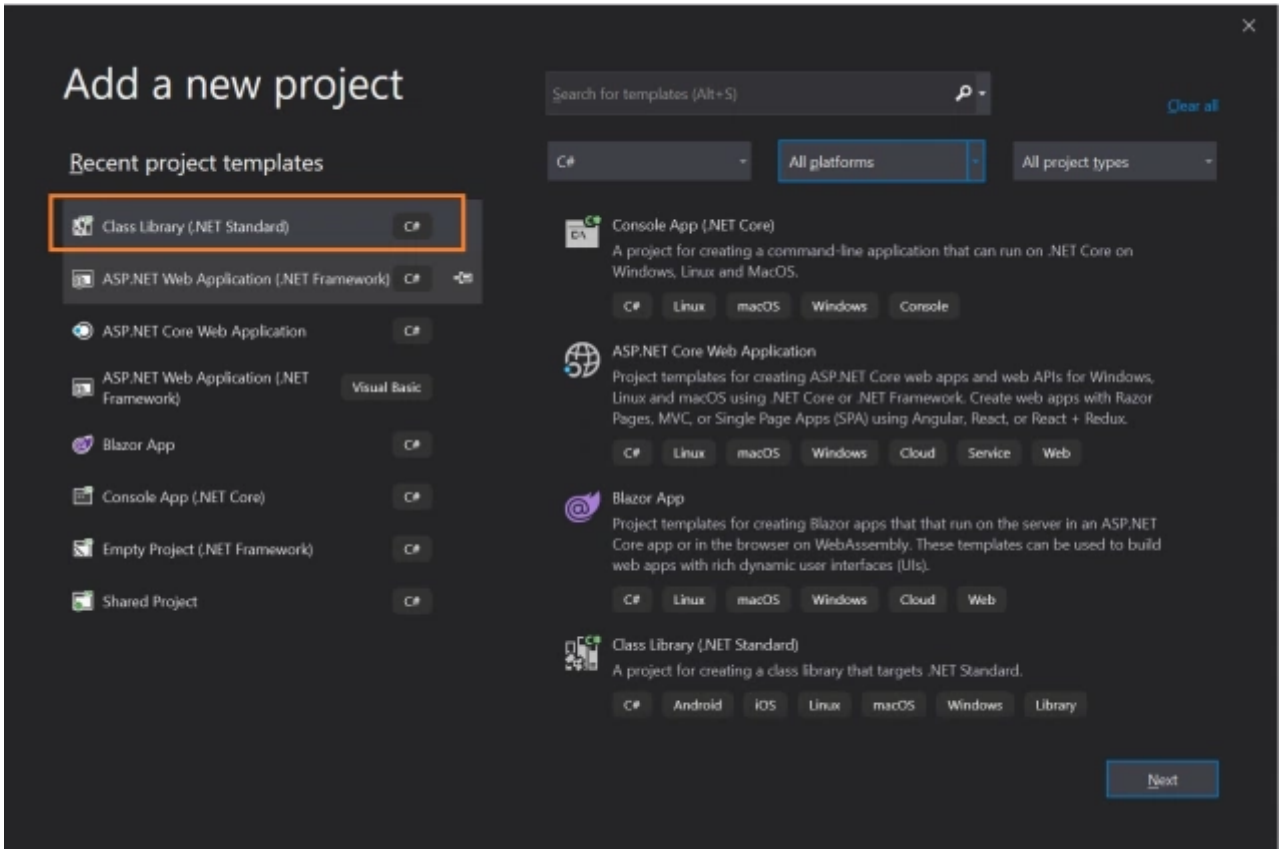
(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/project_architecture/)

Step 1: Now we will Create the blank empty web api project like this



(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/emptyproject-2/)

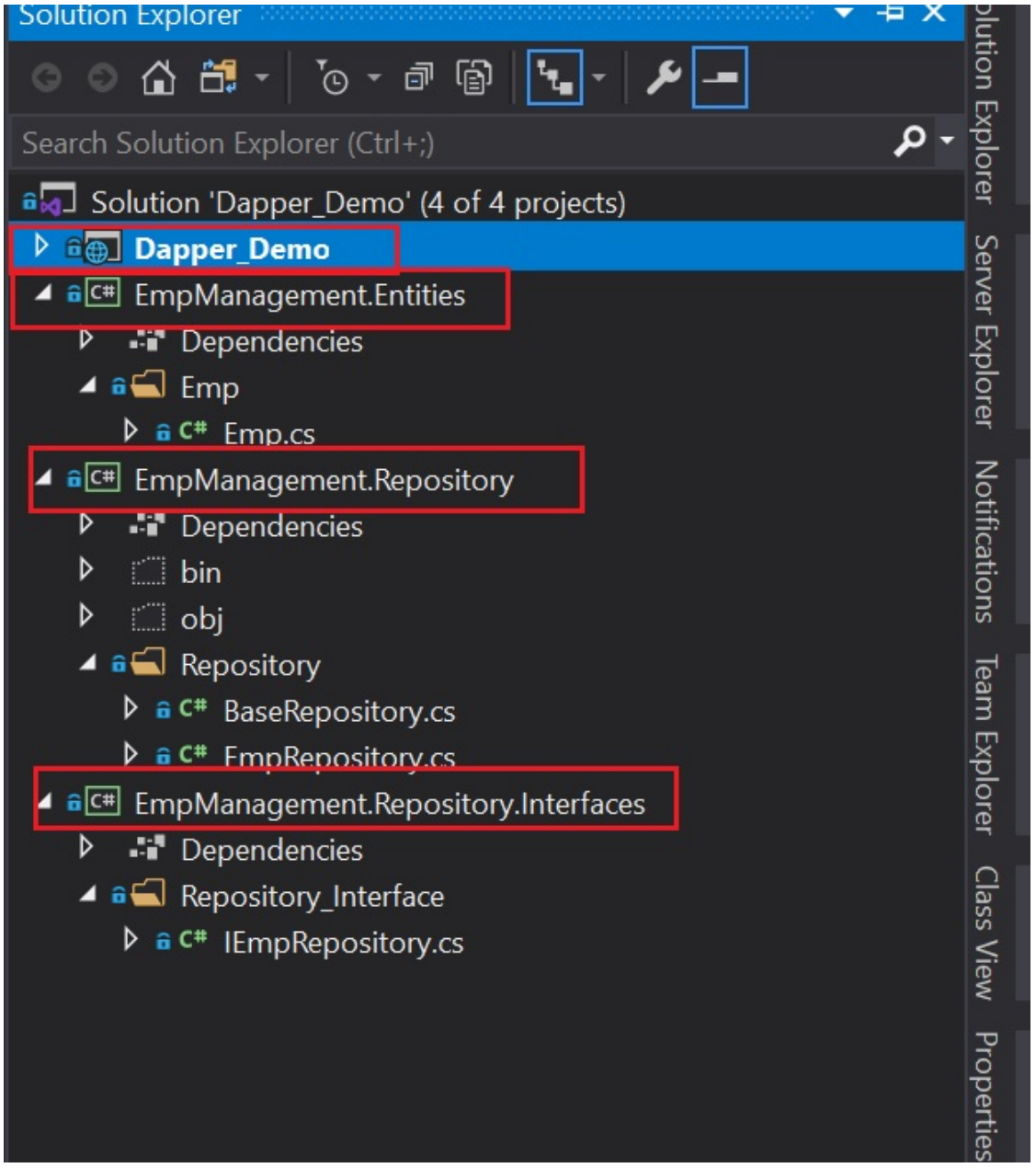
Step 2: Right click on solution explorer and create the EmpManagement.Entities layers using .net standard 2.0 like this



(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/projectlayer/)

Step 3: Follow the same steps and create the **EmpManagement.Repository**

Step 4: Follow the same steps and create the **EmpManagement.Repository.Interfaces**



(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/projectstructure/)

Step 5: Create the table and sp in database like this

```
USE [EmpManagement]
GO
/****** Object:  StoredProcedure [dbo].[AddEmp]    Script Date: 12/12/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[AddEmp]
   @EmpName nvarchar(50),
    @EmpAddress nvarchar(50),
    @EmailId nvarchar(50),
    @MobileNum nvarchar(50)

AS
```

```
15
16  BEGIN
17  SET NOCOUNT ON;
18  insert into tblEmp(EmpName, EmpAddress, EmailId, MobileNum)
19        values(@EmpName, @EmpAddress, @EmailId, @MobileNum)
20  END
21
22  GO
23  /****** Object:  StoredProcedure [dbo].[DeleteEmp]    Script Date: 12/
24  SET ANSI_NULLS ON
25  GO
26  SET QUOTED_IDENTIFIER ON
27  GO
28  CREATE PROCEDURE [dbo].[DeleteEmp]
29     @Id int
30
31  AS
32
33  BEGIN
34  SET NOCOUNT ON;
35  DELETE from tblEmp where    Id=@Id
36  END
37
38  GO
39  /****** Object:  StoredProcedure [dbo].[GetAllEmps]    Script Date: 12
40  SET ANSI_NULLS ON
41  GO
42  SET QUOTED_IDENTIFIER ON
43  GO
44  CREATE PROCEDURE [dbo].[GetAllEmps]
45
46
47  AS
48
49  BEGIN
50  SET NOCOUNT ON;
51  SELECT * from TBLEMP
52  END
53
54  GO
55  /****** Object:  StoredProcedure [dbo].[GetEmpById]    Script Date: 12
56  SET ANSI_NULLS ON
57  GO
58  SET QUOTED_IDENTIFIER ON
59  GO
60  CREATE PROCEDURE [dbo].[GetEmpById]
61    @Id int
62
63  AS
64
65  BEGIN
66  SET NOCOUNT ON;
67  SELECT * from TBLEMP where Id=@Id
68  END
69
70  GO
71  /****** Object:  StoredProcedure [dbo].[UpdateEmp]    Script Date: 12/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[UpdateEmp]
  @Id int,
  @EmpName nvarchar(50),
  @EmpAddress nvarchar(50),
  @EmailId nvarchar(50),
  @MobileNum nvarchar(50)

AS
BEGIN
SET NOCOUNT ON;
UPDATE tblEmp SET EmpName=@EmpName,EmpAddress=@EmpAddress,EmailId=@Ema
END

GO
/****** Object:  Table [dbo].[tblEmp]    Script Date: 12/12/2019 10:14
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblEmp](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [EmpName] [nvarchar](50) NULL,
    [EmpAddress] [nvarchar](50) NULL,
    [EmailId] [nvarchar](50) NULL,
    [MobileNum] [nvarchar](50) NULL,
 CONSTRAINT [PK_tblEmp] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
) ON [PRIMARY]

GO
```

Step 6: Go to EmpManagement.Entities layer and Create the Emp Entities class like this

```
using System;

namespace DataManagement.Entities
{
    public class Emp
    {
        public int Id { get; set; }
        public string EmpName { get; set; }
        public string EmpAddress { get; set; }
        public string EmailId { get; set; }
        public string MobileNum { get; set; }
    }
}
```

Step 7: Go to EmpManagement.Repository.Interfaces layer and Create the generic interface like this

```
using System.Collections.Generic;

namespace DataManagement.Repository.Interfaces
{

    public interface IEmpRepository<T> where T : class
    {
        IEnumerable<T> GetAllEmp();
        T GetEmpById(int id);
        void AddEmp(T entity);
        void DeleteEmp(int id);
        void UpdateEmp(T entity);
    }


}
```

Step 8: Create the BaseRepository.cs file in EmpManagement.Repository layer like this

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace DataManagement.Repository
{
    public class BaseRepository : IDisposable
    {
        protected IDbConnection con;
        public BaseRepository()
        {
            string connectionString = "Data Source=localhost;Initial Ca
            con = new SqlConnection(connectionString);
        }
        public void Dispose()
        {
            throw new NotImplementedException();
        }
    }
}
```

Step 9: Create the BaseRepository.cs file in EmpManagement.Repository layer like this

```
using Dapper;
using DataManagement.Entities;
using DataManagement.Repository;
using DataManagement.Repository.Interfaces;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;

namespace EmpManagement.Repository
{
    public  class EmpRepository<T> : BaseRepository, IEmpRepository<Em
     {
```

```
14          public void AddEmp(Emp objEmp)
15          {
16              try
17              {
18                  DynamicParameters parameters = new DynamicParameters()
19                  con.Open();
20                  parameters.Add("EmpName", objEmp.EmpName);
21                  parameters.Add("EmpAddress", objEmp.EmpAddress);
22                  parameters.Add("EmailId", objEmp.EmailId);
23                  parameters.Add("MobileNum", objEmp.MobileNum);
24                  SqlMapper.Execute(con, "AddEmp", param: parameters, co
25
26                  //For implementing commandtimeout and transaction
27                  // SqlMapper.Execute(con, "AddEmp", param: parameters,
28                  // Other approach to excecute Storeprocedure in dapper
29                  // con.Execute("AddEmp", parameters, null, null, comma
30
31              }
32              catch (Exception ex)
33              {
34
35                  throw ex;
36              }
37          }
38
39          public void DeleteEmp(int Id)
40          {
41              try
42              {
43                  DynamicParameters parameters = new DynamicParameters()
44                  parameters.Add("Id", Id);
45                  SqlMapper.Execute(con, "DeleteEmp", param: parameters,
46              }
47              catch (Exception)
48              {
49
50                  throw;
51              }
52          }
53
54          public IEnumerable<Emp> GetAllEmp()
55          {
56              try
57              {
58                  return SqlMapper.Query<Emp>(con, "GetAllEmps", command
59              }
60              catch (Exception)
61              {
62
63                  throw;
64              }
65          }
66
67          public Emp GetEmpById(int Id)
68          {
69              try
70              {
```

```
                DynamicParameters parameters = new DynamicParameters()
                parameters.Add("Id", Id);
                return SqlMapper.Query<Emp>(con, "GetEmpById", paramet
            }
            catch (Exception)
            {
                throw;
            }
        }

        public void UpdateEmp(Emp objEmp)
        {
            try
            {
                DynamicParameters parameters = new DynamicParameters()
                con.Open();
                parameters.Add("EmpName", objEmp.EmpName);
                parameters.Add("EmpAddress", objEmp.EmpAddress);
                parameters.Add("EmailId", objEmp.EmailId);
                parameters.Add("MobileNum", objEmp.MobileNum);
                parameters.Add("Id", objEmp.Id);
                SqlMapper.Execute(con, "UpdateEmp", param: parameters,
            }
            catch (Exception)
            {

                throw;
            }

        }
    }
}
```

Step 10 :Create the Emp Controller in WebApi Layer as given below

```
using DataManagement.Entities;
using DataManagement.Repository.Interfaces;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace Dapper_Demo.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class EmpController : ControllerBase
    {

        IEmpRepository<Emp> _empRepository;
        public EmpController(IEmpRepository<Emp> empRepository)

        {
            _empRepository = empRepository;
        }

        // GET: api/Emp
        [HttpGet]
        public IEnumerable<Emp> Get()
        {
            return _empRepository.GetAllEmp();
        }

        // GET: api/Emp/5
        [HttpGet("{id}", Name = "Get")]
        public Emp Get(int id)
        {
            return _empRepository.GetEmpById(id);
        }

        // POST: api/Emp
        [HttpPost]
        public void Post([FromBody] Emp emp)
        {
            _empRepository.AddEmp(emp);
        }

        // PUT: api/Emp/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody] Emp emp)
        {
            _empRepository.UpdateEmp(emp);
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
            _empRepository.DeleteEmp(id);
        }
    }
}
```

Step 11: Register the interface in Startup.cs file like this

(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/interface/)

Step 12: Now run the application you will see the output like this

(https://chandradev819.wordpress.com/2019/12/12/creating-web-api-core-3-0-layer-using-dapper-and-net-standard-2-0/output-5/)

Summary:

In this post we learnt that how to create the web api core standalone layer using dapper and .net standard 2.0. You can download the working code from this github repo

Source code download from github (https://github.com/Chandradev819/Dapper_Demo_With_Asp.netCore.git)

 Asp.net WebApi Core 3.0, Dapper, Web API . .net Standard 2.0, Dapper, Web API Core

One thought on “Creating Web Api Core 3.0 layer using Dapper and .net standard 2.0”

1. **VERY DETAILED**
   September 22, 2020 / 10:03 am

   0

   0

   i
   Rate This

Good one

Reply