CRUD using ASP.NET Core

Generic Layer Based Repository Architectural pattern

Shahriar Biggo
Shoaibshahriar29@gmail.com
https://www.linkedin.com/in/biggo29/

CRUD using ASP.NET Core GENERIC Layer Based Repository Architectural pattern

Introduction

Prerequisite

1. MS SQL SERVER
2. MS VISUAL STUDIO 19
3. ASP.NET CORE 3.0+

Create table

For reference we will take 2 tables. **Employee** and **Department**. Where there will be relation between employee and department table. To create the table, follow the instruction below.

1. We will use MS SQL Server for database.
2. Open Sql Server management studio.
3. Right click on **Tables** > New… > Table…
4. Insert column name and size of the field.
5. Set primary key and right click on the primary key. Go to properties.
6. Set Identity Specification Yes, (Is Identity) yes.
7. Save the table.

BIGGO.TEST - dbo....EST - dbo.Employee | BIGGO.TEST - dbo....T - dbo.

Column Name	Data Type	Allow Nulls
EmpId	int	☐
EmpName	nvarchar(50)	☐
EmpAge	int	☐
EmpGender	bit	☐
DeptId	int	☐
EmpEmail	nvarchar(50)	☑
EmpPhoto	varbinary(MAX)	☑
		☐

Column Properties

Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes
Is Columnset	No
Is Sparse	No
Merge-published	No
Not For Replication	No
Replicated	No

(Is Identity)

BIGGO.TEST - dbo....EST - dbo.Employee | BIGGO.TEST - dbo....T - dbo.Department

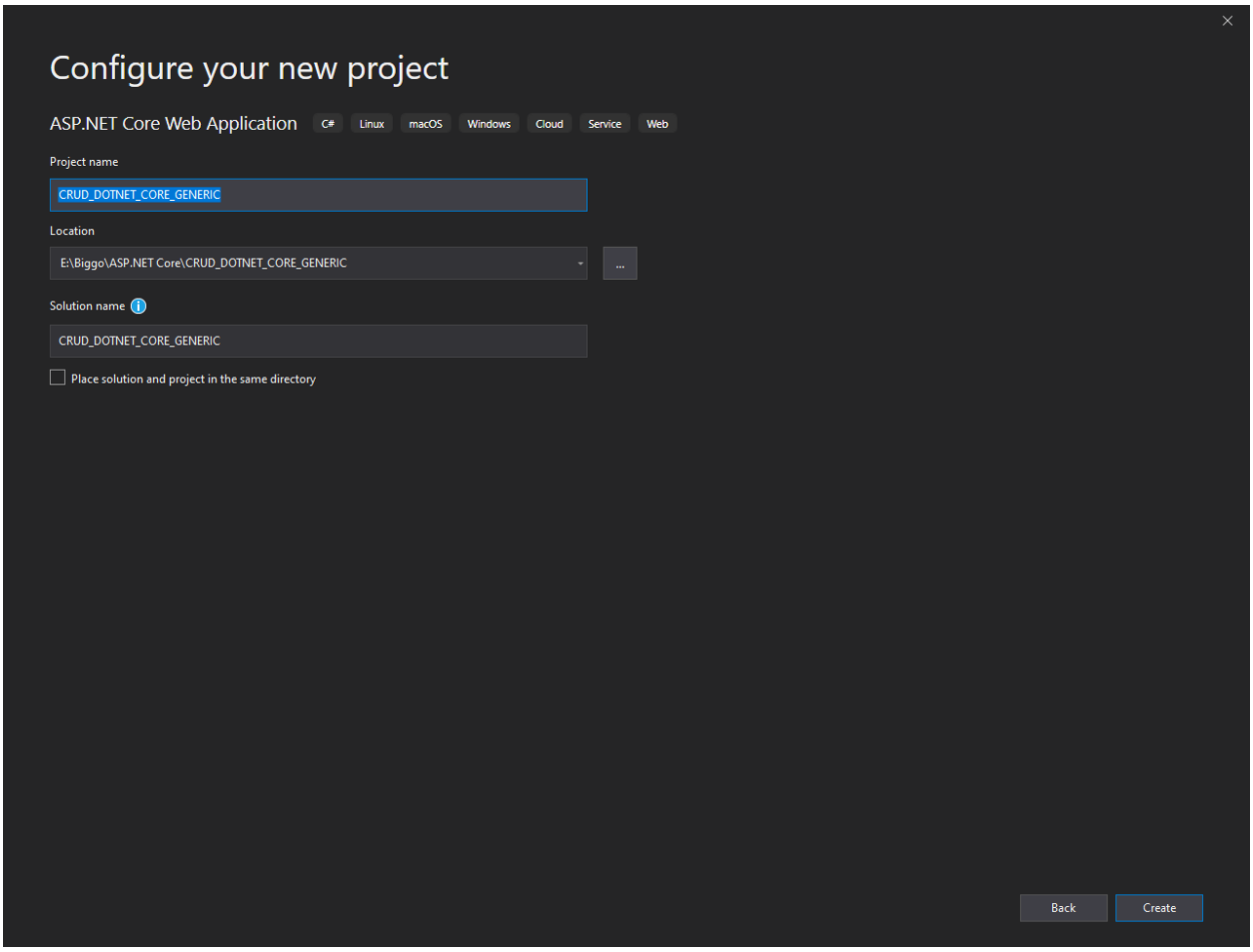
Column Name	Data Type	Allow Nulls
DeptId	int	☐
DeptName	nvarchar(50)	☐
		☐

Column Properties

(General)	
(Name)	DeptId
Allow Nulls	No
Data Type	int
Default Value or Binding	
Table Designer	
Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes
Is Columnset	No
Is Sparse	No
Merge-published	No
Not For Replication	No
Replicated	No
RowGuid	No
Size	4

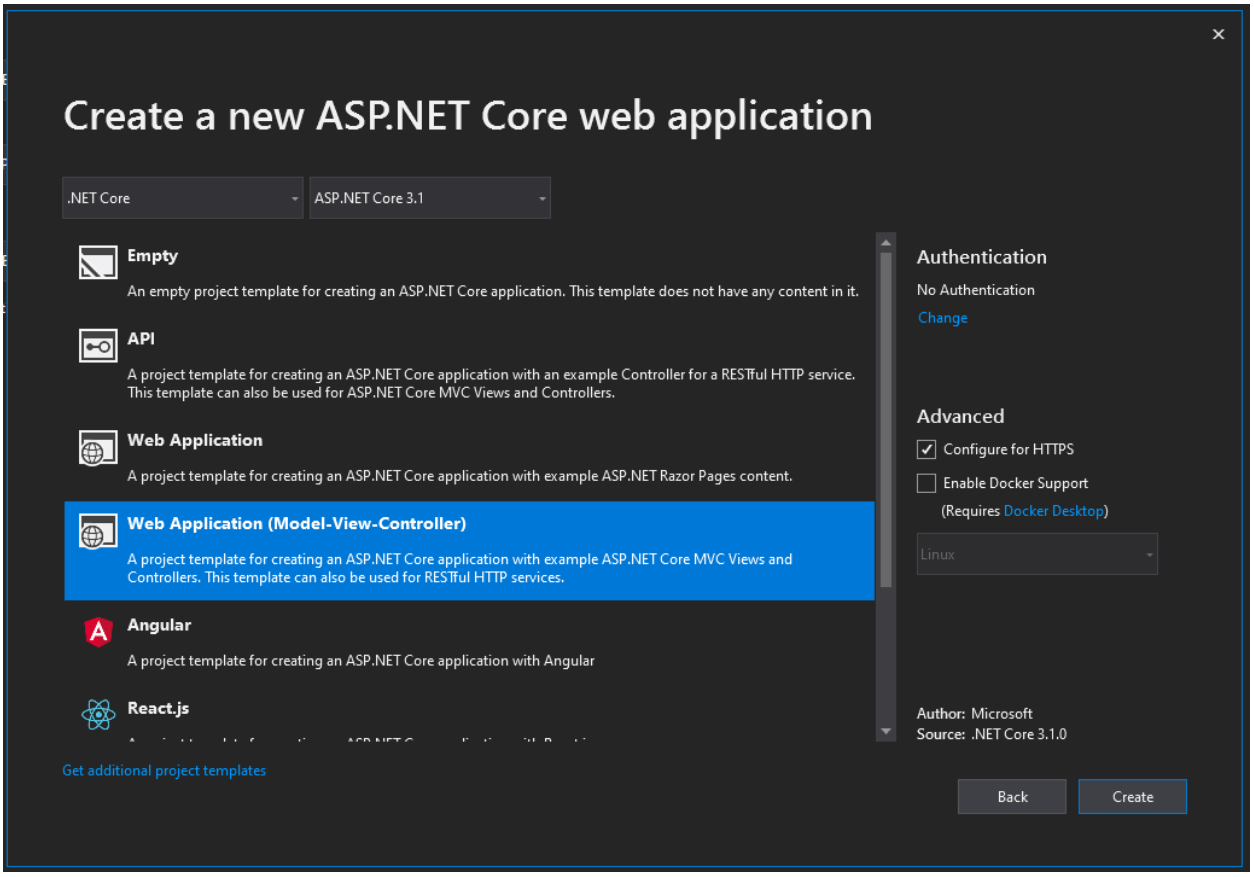
Crete Project

- Open MS Visual Studio 19, Make sure it has ASP.NET Core 3.0+ installed.
- Create new project. Select ASP.NET CORE Web Application



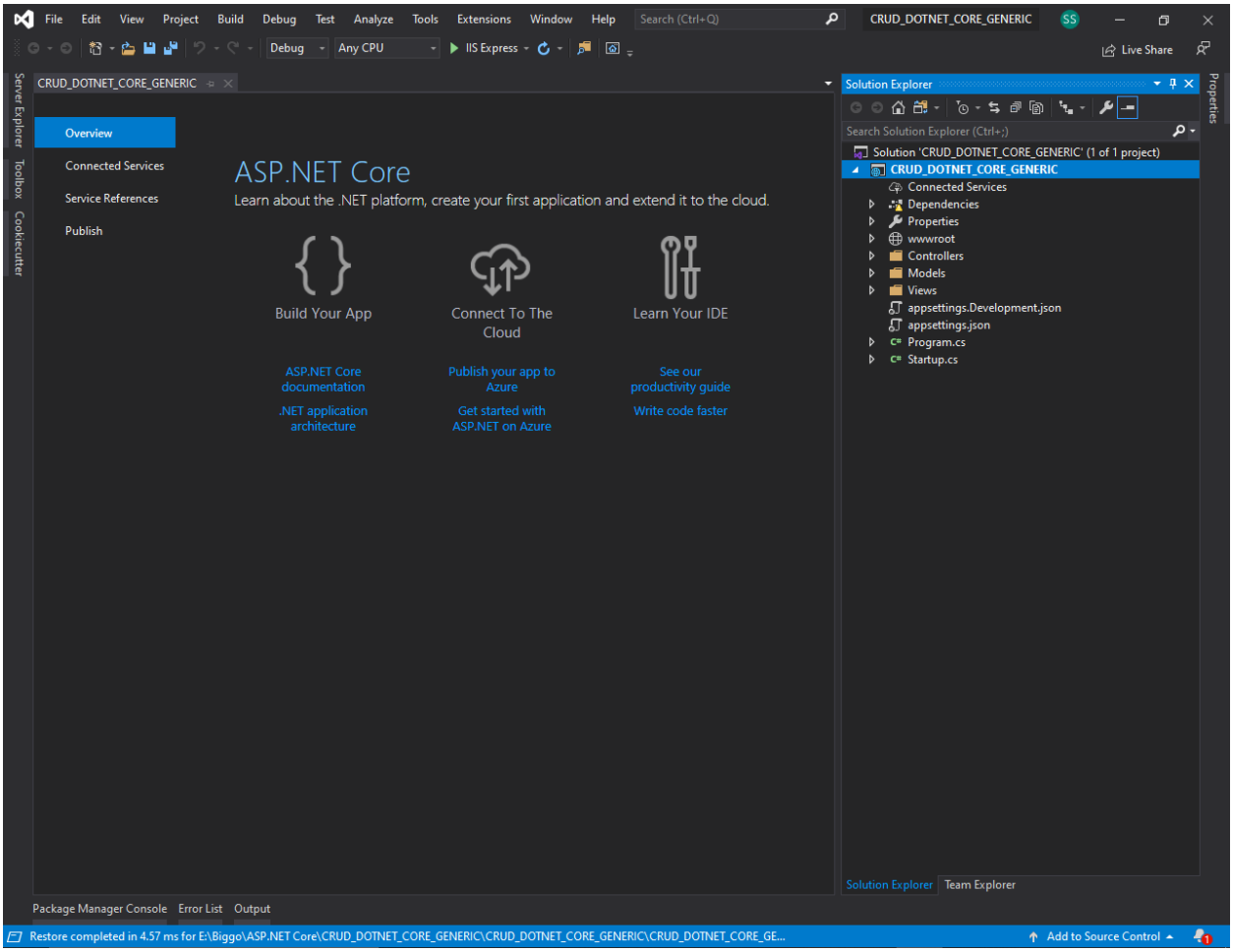
- Insert project and solution name. click on Create

- Select Web Application(Model-View-Controller)



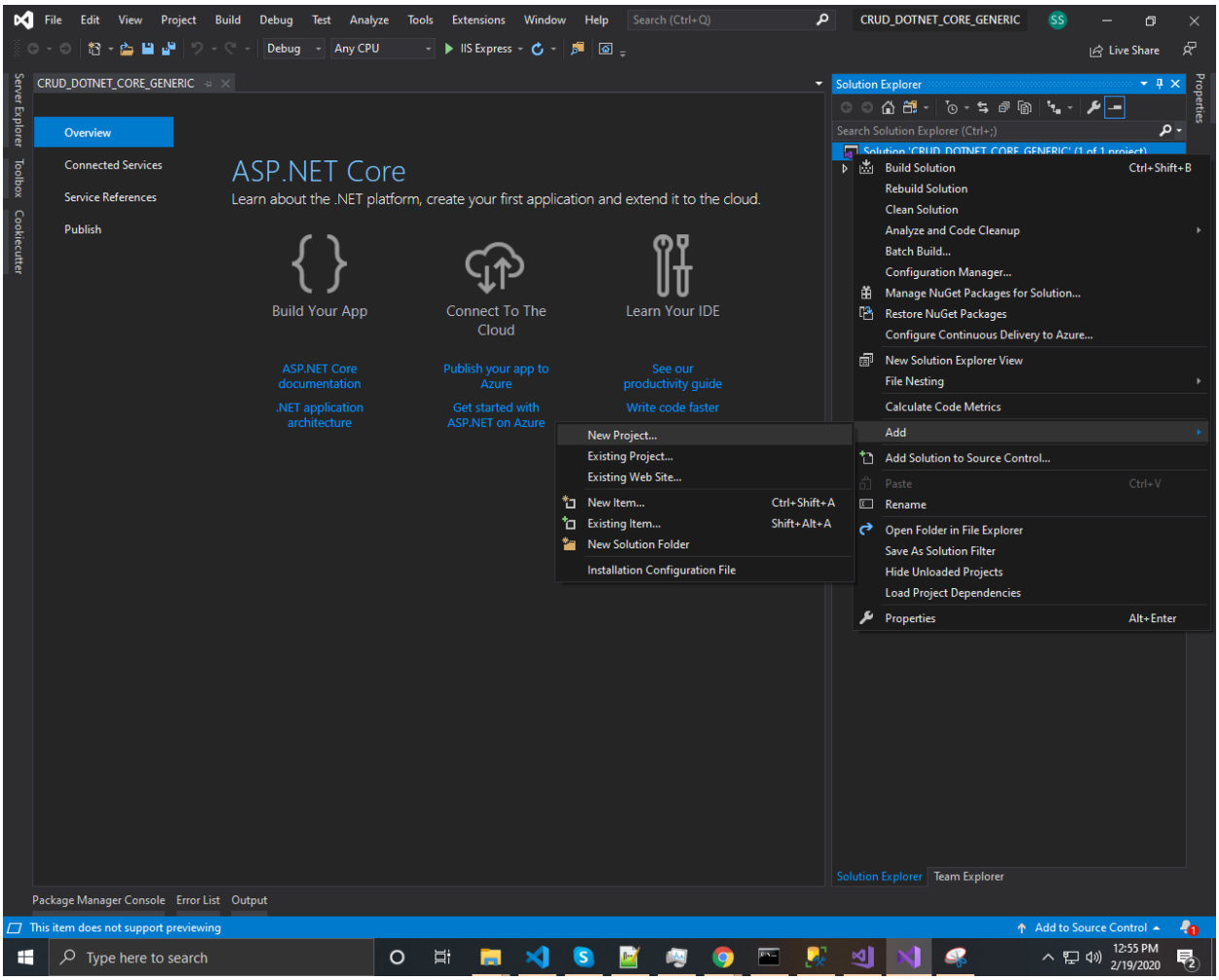
- Click on create.

- After creating the project will be initially looked like the following screenshot

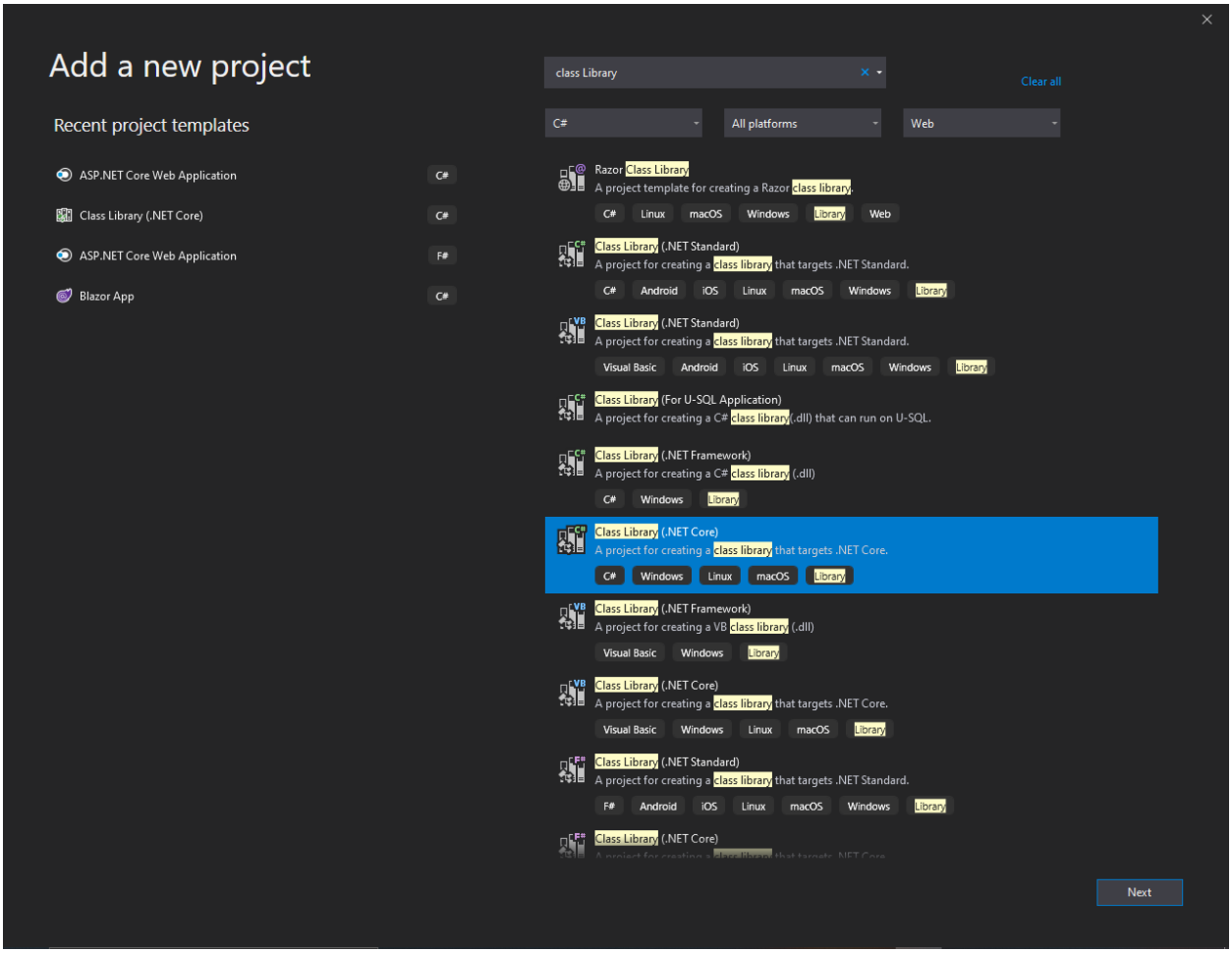


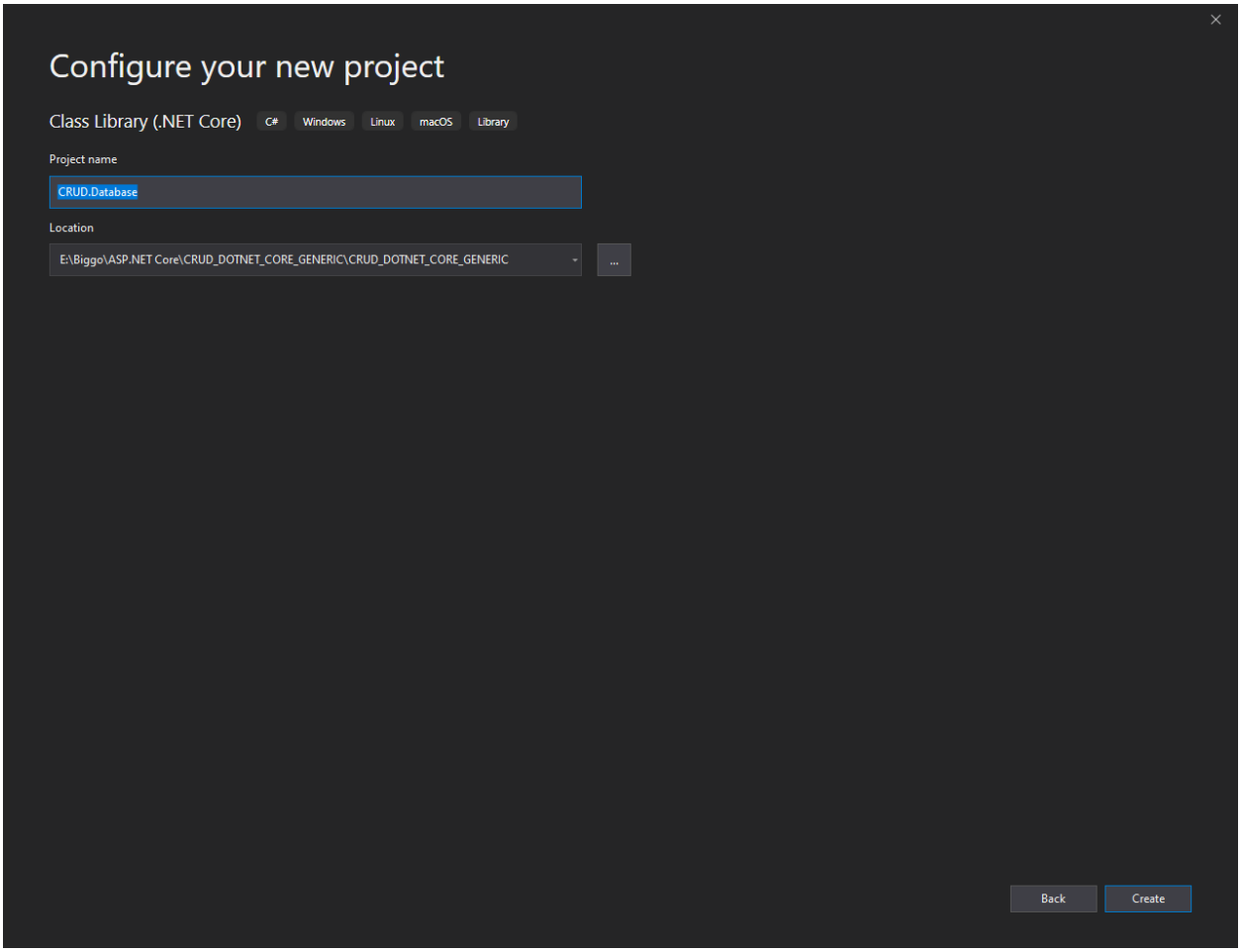
Create Class library

- Now to will create the layer for the project using class library project.
- Right click on the solution file. Add > New Project…
- We will use 4 class libraries.
    - CRUD.Core
    - CRUD.Database
    - CRUD.IOC
    - CRUD.Service

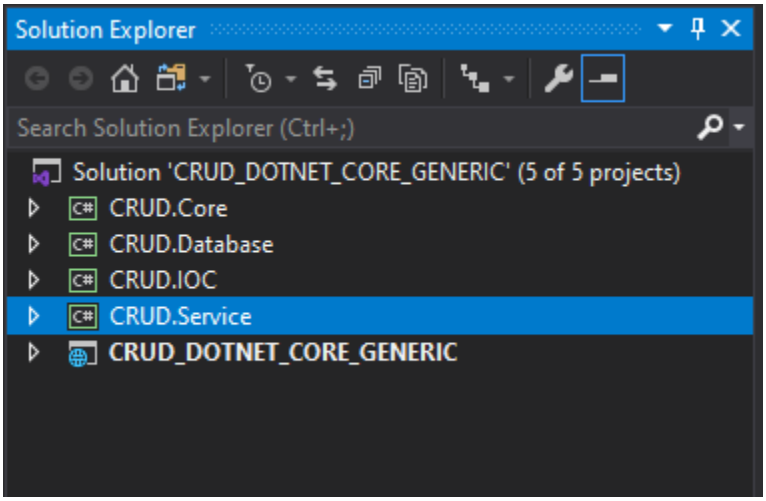


- Search for “Class Library”. Select Class Library(.NET Core) C# project and click next.
- Insert name for the project.




Configure your new project
Class Library (.NET Core)
C#
Windows
Linux
macOS
Library
Project name
CRUD.Database
Location
E:\Biggo\ASP.NET Core\CRUD_DOTNET_CORE_GENERIC\CRUD_DOTNET_CORE_GENERIC
...
Back
Create

- After creating all 4 class library project the layout will be look like following.



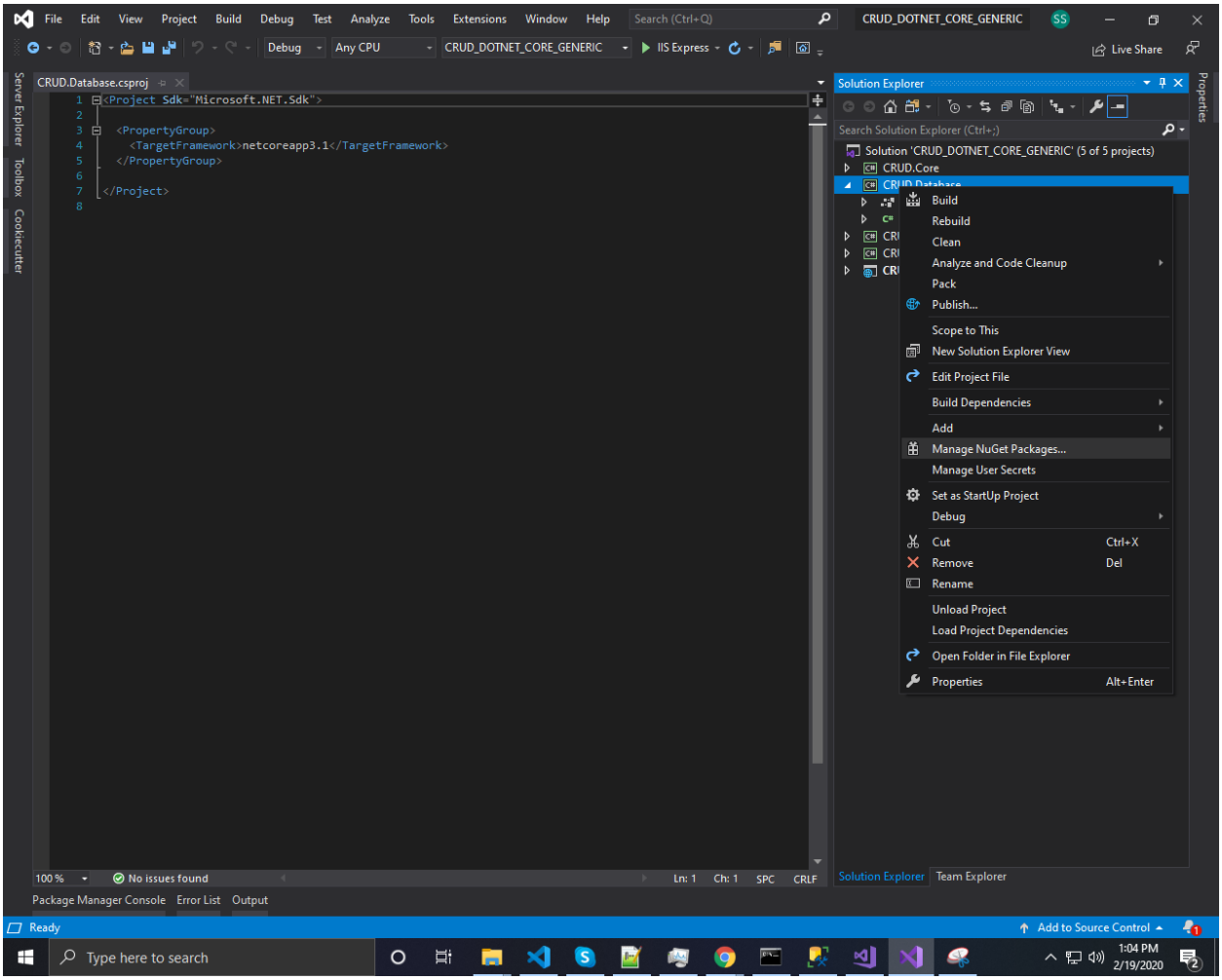
CRUD.Database Class library

Purpose

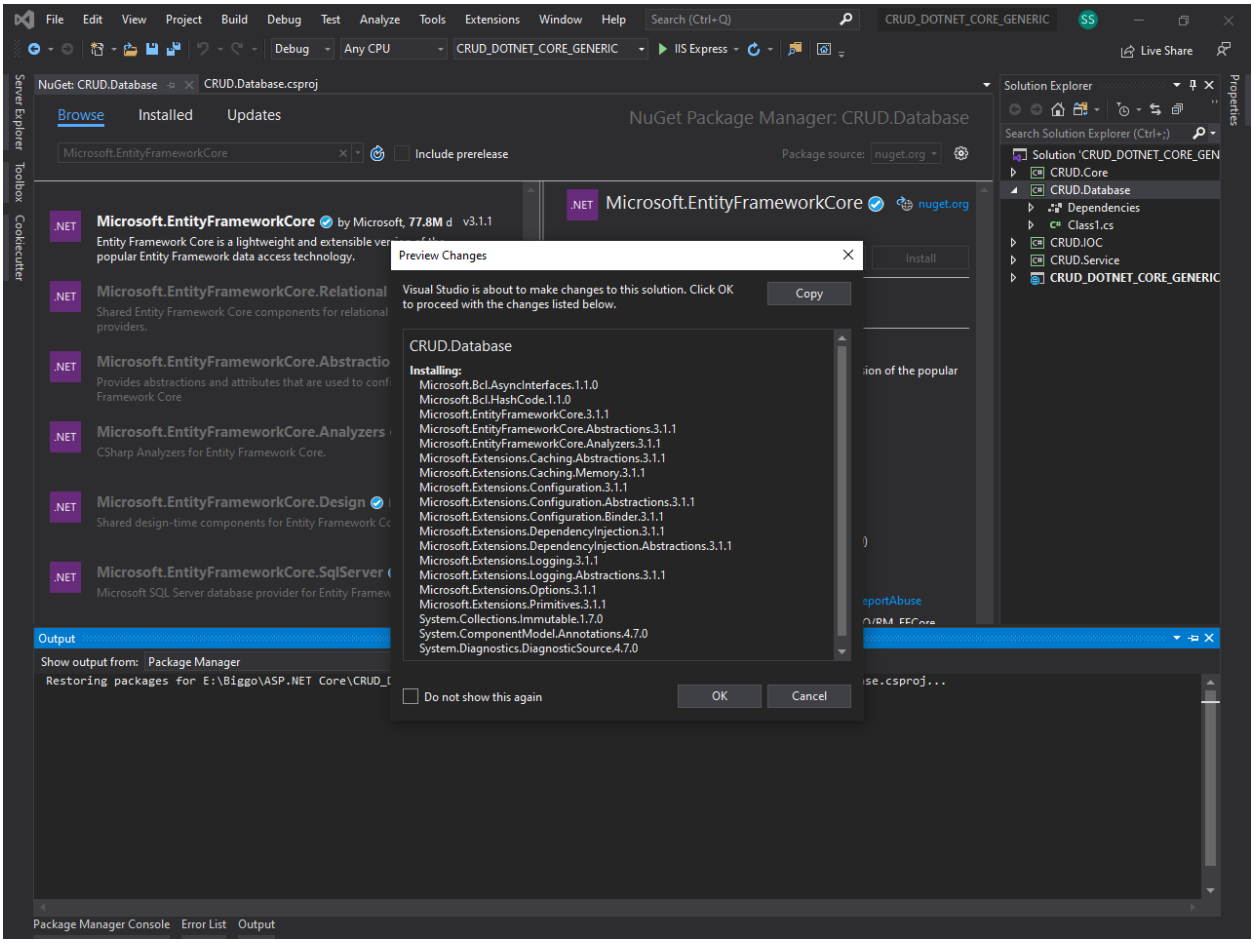
- This class library project is the database layer AKA data access layer or DAL.
- We keep isolated the database from the main project.
- If any changes are required in the database, we just change in the class file and pass the .dll file the main project.
-

Adding Dependencies

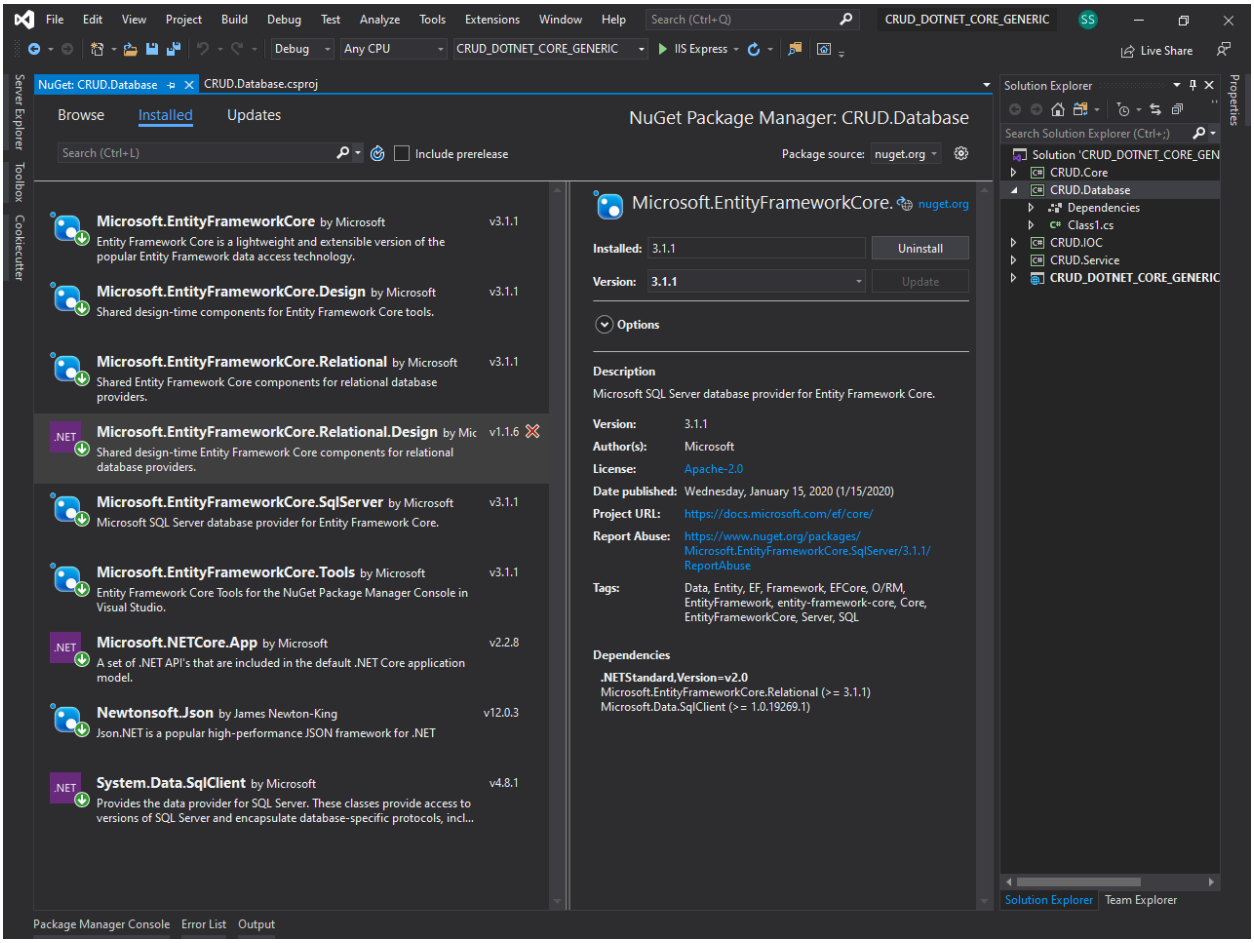
- Right click on the CRUD.Database project. Set as startup project.
- Right click on the project > Manage NuGet Packages…



- Search for the following and install them
  - Microsoft.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.Design
  - Microsoft.EntityFrameworkCore.Relational
  - Microsoft.EntityFrameworkCore.Relational.Design
  - Microsoft.EntityFrameworkCore.SqlServer
  - Microsoft.EntityFrameworkCore.Tools
  - Microsoft.NETCore.App
  - Newtonsoft.Json
  - System.Data.SqlClient

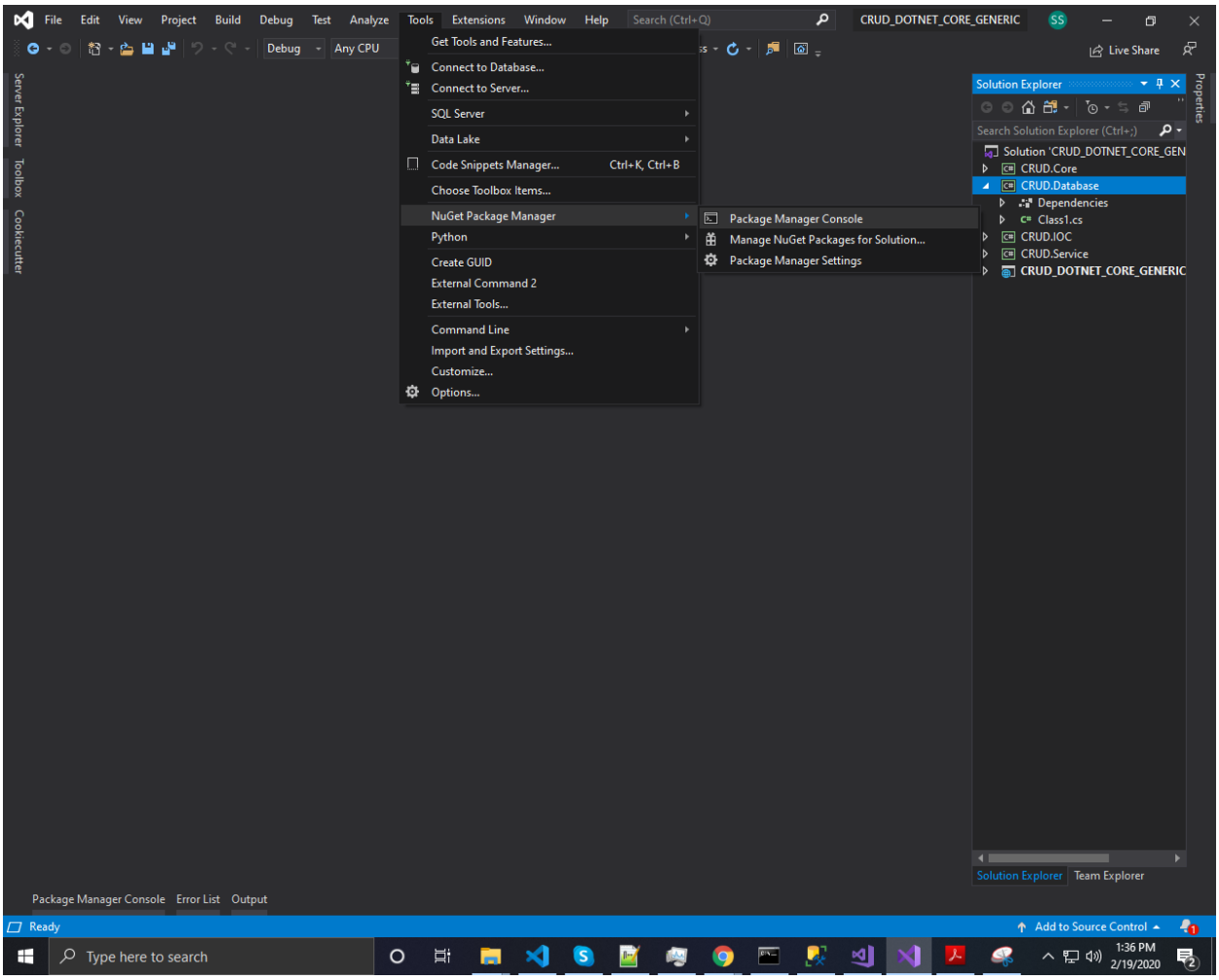

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
CRUD_DOTNET_CORE_GENERIC
Debug
Any CPU
CRUD_DOTNET_CORE_GENERIC
IIS Express
Live Share
NuGet: CRUD.Database
CRUD.Database.csproj
Browse
Installed
Updates
NuGet Package Manager: CRUD.Database
Include prerelease
Package source: nuget.org
Microsoft.EntityFrameworkCore by Microsoft, 77.8M d v3.1.1
Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.
Microsoft.EntityFrameworkCore.Relational
Shared Entity Framework Core components for relational providers.
Microsoft.EntityFrameworkCore.Abstractions
Provides abstractions and attributes that are used to configure Entity Framework Core
Microsoft.EntityFrameworkCore.Analyzers
CSharp Analyzers for Entity Framework Core.
Microsoft.EntityFrameworkCore.Design
Shared design-time components for Entity Framework Core
Microsoft.EntityFrameworkCore.SqlServer
Microsoft SQL Server database provider for Entity Framework Core
Microsoft.EntityFrameworkCore nuget.org
Install
Preview Changes
Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.
Copy
CRUD.Database
Installing:
Microsoft.Bcl.AsyncInterfaces.1.1.0
Microsoft.Bcl.HashCode.1.1.0
Microsoft.EntityFrameworkCore.3.1.1
Microsoft.EntityFrameworkCore.Abstractions.3.1.1
Microsoft.EntityFrameworkCore.Analyzers.3.1.1
Microsoft.Extensions.Caching.Abstractions.3.1.1
Microsoft.Extensions.Caching.Memory.3.1.1
Microsoft.Extensions.Configuration.3.1.1
Microsoft.Extensions.Configuration.Abstractions.3.1.1
Microsoft.Extensions.Configuration.Binder.3.1.1
Microsoft.Extensions.DependencyInjection.3.1.1
Microsoft.Extensions.DependencyInjection.Abstractions.3.1.1
Microsoft.Extensions.Logging.3.1.1
Microsoft.Extensions.Logging.Abstractions.3.1.1
Microsoft.Extensions.Options.3.1.1
Microsoft.Extensions.Primitives.3.1.1
System.Collections.Immutable.1.7.0
System.ComponentModel.Annotations.4.7.0
System.Diagnostics.DiagnosticSource.4.7.0
Do not show this again
OK
Cancel
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'CRUD_DOTNET_CORE_GEN
CRUD.Core
CRUD.Database
Dependencies
Class1.cs
CRUD.IOC
CRUD.Service
CRUD_DOTNET_CORE_GENERIC
Output
Show output from: Package Manager
Restoring packages for E:\Biggo\ASP.NET Core\CRUD_D
Package Manager Console
Error List
Output

- After installing the packages, the install packages will be look like the following.

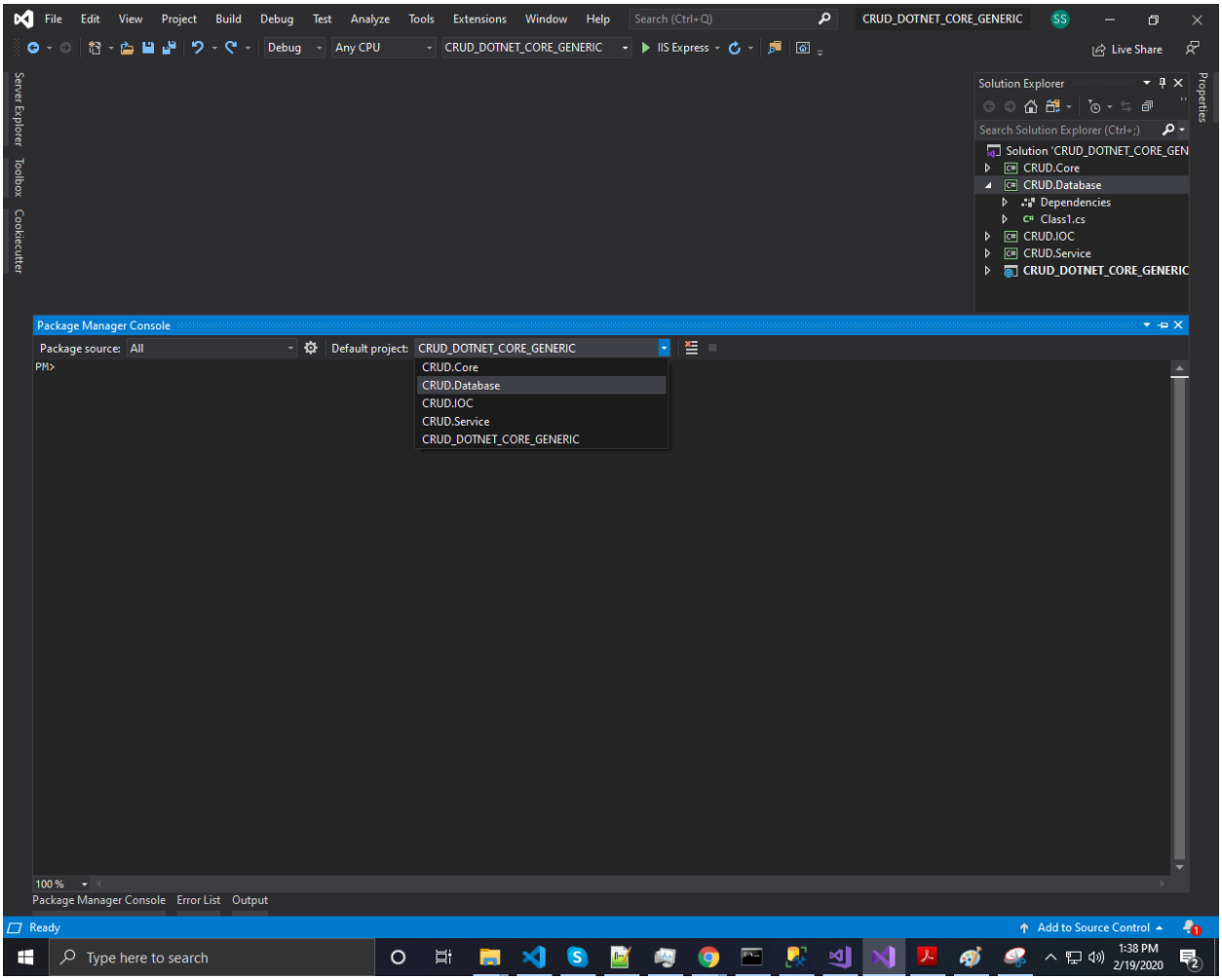


EF Core. DB First Creating model from existing DB.

- Right click on the Database class library project > NuGet Package Manager > Package Manager Console



- Select CRUD.Databse as the default project.

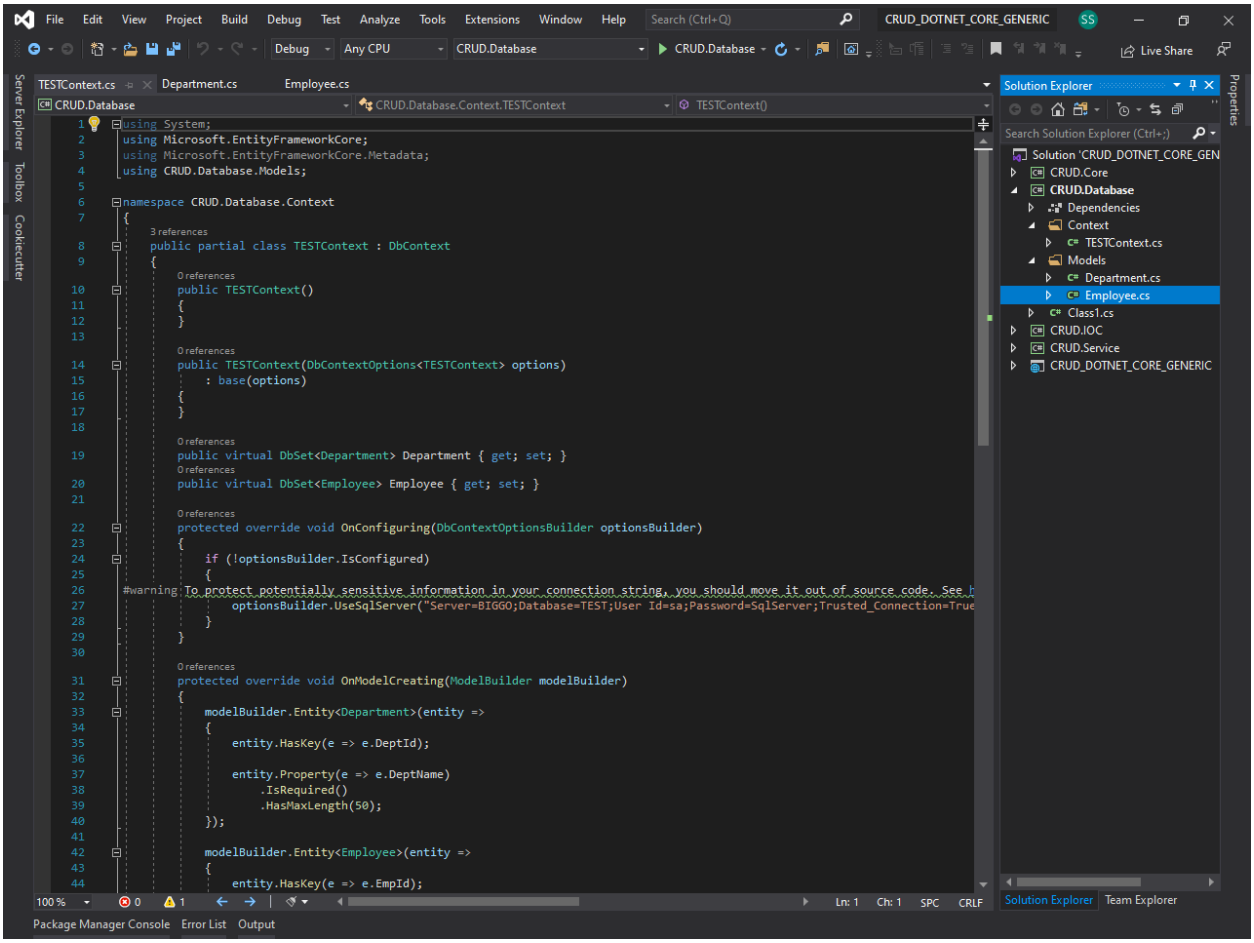


Scaffold-DbContext

- Type the scaffold command in the console. And the output will be the following

```
PM> Scaffold-DbContext "Server=BIGGO;Database=TEST;User
Id=sa;Password=SqlServer;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -f -ContextDir Context -Context
TESTContext
Build started...
Build succeeded.
```

- After Build successful the database project will the look like the following.



SqlOption

- Create a class in CRUD.Database directory to get connection string from main project.

```
using System;

namespace CRUD.Database
{
    public class SqlOption
    {
        public string ConnectionString { get; set; }
    }
}
```

Edit the 'TESTContext'

- Add a new constructor that will provided connection string to other/main project.

```
        public TESTContext(IOptions<SqlOption> connectionString)
        {
            this._connectionString = connectionString.Value;
        }
```

Edit `OnConfiguring` method

- Add the below line of code to get the connection string from appsettings.json file.

```
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
                //optionsBuilder.UseSqlServer("Server=BIGGO;Database=TEST;User
Id=sa;Password=SqlServer;Trusted_Connection=True;");
                optionsBuilder.UseSqlServer(this._connectionString.ConnectionString);
            }
        }
```

Repository

- Create new directory named 'Repository'.
- We will create two separated class and two separated interface in the Repository directory as following.
    - IRawRepository.cs
    - IRepository.cs
    - RawRepository.cs
    - Repository.cs

IRawRepository

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Text;

namespace CRUD.Database.Repository
{
    public interface IRawRepository
```

```
    {
        int Count(params SqlParameter[] parm);
        List<TEntity> Get<TEntity>(params SqlParameter[] parm) where TEntity : class, new();
        List<TEntity> GetWithoutParam<TEntity>() where TEntity : class, new();
        TEntity GetSingle<TEntity>(params SqlParameter[] parm) where TEntity : class, new();
        List<TEntity> CallSP<TEntity>(params SqlParameter[] parm) where TEntity : class,
new();
        int CallSPW(params SqlParameter[] parm);
    }
}
```

RawRepository

```
using CRUD.Database.Context;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;
using System.Linq;
using System.Reflection;
using System.Text;

namespace CRUD.Database.Repository
{
    public class RawRepository : IRawRepository, IDisposable
    {
        private readonly TESTContext _context;
        private readonly string _sql;
        private readonly DbConnection _connection;

        private bool _disposed;

        //Openning connection using Dependency injection
        public RawRepository(TESTContext context, string sql)
        {
            _context = context;
            _sql = sql;
            _connection = _context.Database.GetDbConnection();
            if(_connection.State!= ConnectionState.Open)
            {
                _connection.Open();
            }
        }

        //counter of an Entity
        public int Count(params SqlParameter[] parm)
        {
            int count = 1;
            //using (var connection = _context.Database.GetDbConnection())
            //{
            //connection.Open();
            using (var command = _connection.CreateCommand())
            {
                command.CommandText = _sql;
                command.Parameters.AddRange(parm);
                var reader = command.ExecuteReader();
                while(reader.Read())
                {
                    count = reader.GetInt16(0);
                }
                //connection close
                reader.Close();
            }
            return count;
        }

        //Convert Value
        private T ConvertValue<T, U>(U value) where U : IConvertible
        {
            return (T)Convert.ChangeType(value, typeof(T));
        }

        //reader
        private TEntity reader<TEntity>(DbDataReader reder) where TEntity : class, new()
        {
            var obj = new TEntity();
```

```
            Type t = obj.GetType();
            foreach (PropertyInfo propInfo in t.GetProperties())
            {
                if (propInfo.PropertyType.IsClass)
                {
                    object propVal = propInfo.GetValue(obj, null);
                    //setValsRecursive(propVal, value);
                }
                if (Enumerable.Range(0, reder.FieldCount).Any(i => string.Equals(reder.GetName(i),
propInfo.Name, StringComparison.OrdinalIgnoreCase)) && reder[propInfo.Name] != DBNull.Value)
                {
                    propInfo.SetValue(obj, reder[propInfo.Name], null);
                }

            }
            return obj;
        }

        //Get Data with parameters
        public List<TEntity> Get<TEntity>(params SqlParameter[] parm) where TEntity : class, new()
        {
            List<TEntity> data = new List<TEntity>();
            Type obj = new TEntity().GetType();
            using(var command = _connection.CreateCommand())
            {
                command.CommandText = _sql;
                command.Parameters.AddRange(parm);
                var reader = command.ExecuteReader();
                while(reader.Read())
                {
                    data.Add(reader<TEntity>(reader));
                }
                //connection close
                reader.Close();
            }

            return data;
        }

        //Get Data without param
        public List<TEntity> GetWithoutParam<TEntity>() where TEntity : class, new()
        {
            List<TEntity> data = new List<TEntity>();
            Type obj = new TEntity().GetType();
            using(var command = _connection.CreateCommand())
            {
                command.CommandText = _sql;
                var reader = command.ExecuteReader();
                while(reader.Read())
                {
                    data.Add(reader<TEntity>(reader));
                }
                //close connection
                reader.Close();
            }
            return data;
        }

        //Get Single Data
        public TEntity GetSingle<TEntity>(params SqlParameter[] parm) where TEntity : class, new()
        {
            int rowCount = 0;
            TEntity data = new TEntity();
            Type obj = new TEntity().GetType();
            using(var command = _connection.CreateCommand())
            {
                command.CommandText = _sql;
                var reader = command.ExecuteReader();
                while(reader.Read())
```

```
            {
                if(rowCount > 0)
                {
                    throw new Exception();
                }
                data = (reader<TEntity>(reader));
                rowCount++;
            }
            //close conneciton
            reader.Close();
        }
        return data;
    }

    //Fetch list of data calling from SP
    public List<TEntity> CallSP<TEntity>(params SqlParameter[] parm) where TEntity : class, new()
    {
        List<TEntity> data = new List<TEntity>();
        Type obj = new TEntity().GetType();
        using(var command = _connection.CreateCommand())
        {
            command.CommandText = _sql;
            command.Parameters.AddRange(parm);
            var reader = command.ExecuteReader();
            while(reader.Read())
            {
                data.Add(reader<TEntity>(reader));
            }
            //close connection
            reader.Close();
        }
        return data;
    }

    //SP check
    public int CallSPW(params SqlParameter[] parm)
    {
        using(var command = _connection.CreateCommand())
        {
            command.CommandText = _sql;
            command.CommandType = CommandType.StoredProcedure;
            var reader = command.ExecuteReader();
            while(reader.Read())
            {
                var res = reader["result"];
            }
            //close connection
            reader.Close();
        }
        return 1;
    }

    //dispose
    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    public virtual void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {
                if (_connection.State == ConnectionState.Open)
                {
                    _connection.Close();
                }
```

```
                }
            }
            _disposed = true;
        }

        private new List<SqlParameter> convert(params SqlParameter[] sqlParameters)
        {
            List<SqlParameter> _sqlParameters = new List<SqlParameter>();
            foreach(SqlParameter sqlParameter in sqlParameters)
            {
                _sqlParameters.Add(new SqlParameter(sqlParameter.ParameterName, sqlParameter.Value));
            }
            return _sqlParameters;
        }

        private SqlDataReader ExecuteSqlCommand(SqlParameter parm)
        {
            using(var connection = _context.Database.GetDbConnection())
            {
                connection.Open();
                using(var command = _connection.CreateCommand())
                {
                    command.CommandText = _sql;
                    return (SqlDataReader)command.ExecuteReader();
                }
            }
        }
    }
}
```

IRepository

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;

namespace CRUD.Database.Repository
{
    public interface IRepository<TEntity>
    {
        int Count(Expression<Func<TEntity, bool>> filter);
        IEnumerable<TEntity> Get(Expression<Func<TEntity, bool>> filter);
        IEnumerable<TEntity> GetQuery(Expression<Func<TEntity, bool>> filter);

        TEntity GetSingle(Expression<Func<TEntity, bool>> filter);
        TEntity GetFirstOrDefault(Expression<Func<TEntity, bool>> filter);

        IEnumerable<TEntity> GetAll();

        IQueryable<TEntity> GetQueryAll();

        void Insert(TEntity entity);

        void Update(TEntity entity);

        void Delete(TEntity entity);

        void Delete(Expression<Func<TEntity, bool>> filter);

        //RawSqlRepository GetParsedOrDefaultValue(string sql);
    }
}
```

Repository

```
using CRUD.Database.Context;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;

using CRUD.Database.Models;

namespace CRUD.Database.Repository
{
    public class Repository<TEntity> : IRepository<TEntity>
        where TEntity : class
    {
        private readonly TESTContext _context;
        private readonly DbSet<TEntity> _entities;

        public Repository(TESTContext context)
        {
            _context = context;
            _entities = context.Set<TEntity>();
        }

        public int Count(Expression<Func<TEntity, bool>> filter)
        {
            return _entities.Count(filter);
        }

        public IEnumerable<TEntity> Get(Expression<Func<TEntity, bool>> filter)
        {
            return GetQuery(filter).ToList();
        }

        public IEnumerable<TEntity> GetQuery(Expression<Func<TEntity, bool>> filter)
        {
            return _entities.Where(filter);
        }


        public TEntity GetFirstOrDefault(Expression<Func<TEntity, bool>> filter)
        {
            return Get(filter).FirstOrDefault();
        }
        public TEntity GetSingle(Expression<Func<TEntity, bool>> filter)
        {
            return Get(filter).Single();
        }

        public IEnumerable<TEntity> GetAll()
        {
            return GetQueryAll().ToList();
        }

        public IQueryable<TEntity> GetQueryAll()
        {
            return _entities;
        }

        public void Insert(TEntity entity)
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity));
            }
```

```
            _entities.Add(entity);
        }

        public void Update(TEntity entity)
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity));
            }

            try
            {
                _context.Set<TEntity>().Attach(entity);
            }
            catch (Exception e)
            {
                _context.Entry(entity).State = EntityState.Detached;
                _context.Set<TEntity>().Attach(entity);
            }

            _context.Entry(entity).State = EntityState.Modified;
        }

        public void Delete(TEntity entity)
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity));
            }

            _entities.Remove(entity);
        }

        public void Delete(Expression<Func<TEntity, bool>> filter)
        {
            foreach (var entity in Get(filter))
            {
                Delete(entity);
            }
        }

        //public RawSqlRepository GetParsedOrDefaultValue(string sql)
        //{
        //return new RawSqlRepository(_context, sql);
        //}
    }
}
```

UnitOfWork

- Create new directory named 'UnitOfWork'.
- We will create one class and one interface in the UnitOfWork directory as following.
  - IUnitOfWork.cs
  - UnitOfWork.cs

IUnitOfWork

```
using CRUD.Database.Repository;
using System;
using System.Collections.Generic;
using System.Text;

namespace CRUD.Database.UnitOfWork
{
    public interface IUnitOfWork
    {
        void Save();

        IRepository<TData> Repository<TData>() where TData : class;
        IRawRepository Repository(string sql);
    }
}
```

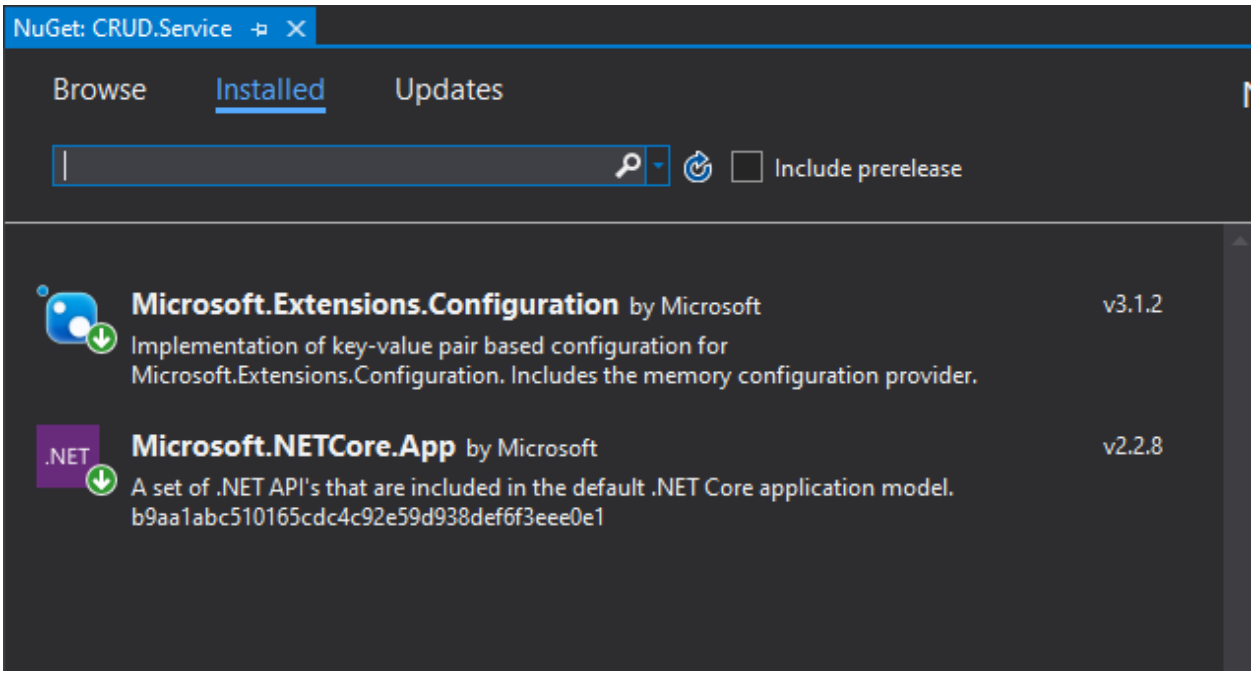
CRUD.Service class library

Purpose

- This layer is known as all Business layer.
- This class library has all the business logic.

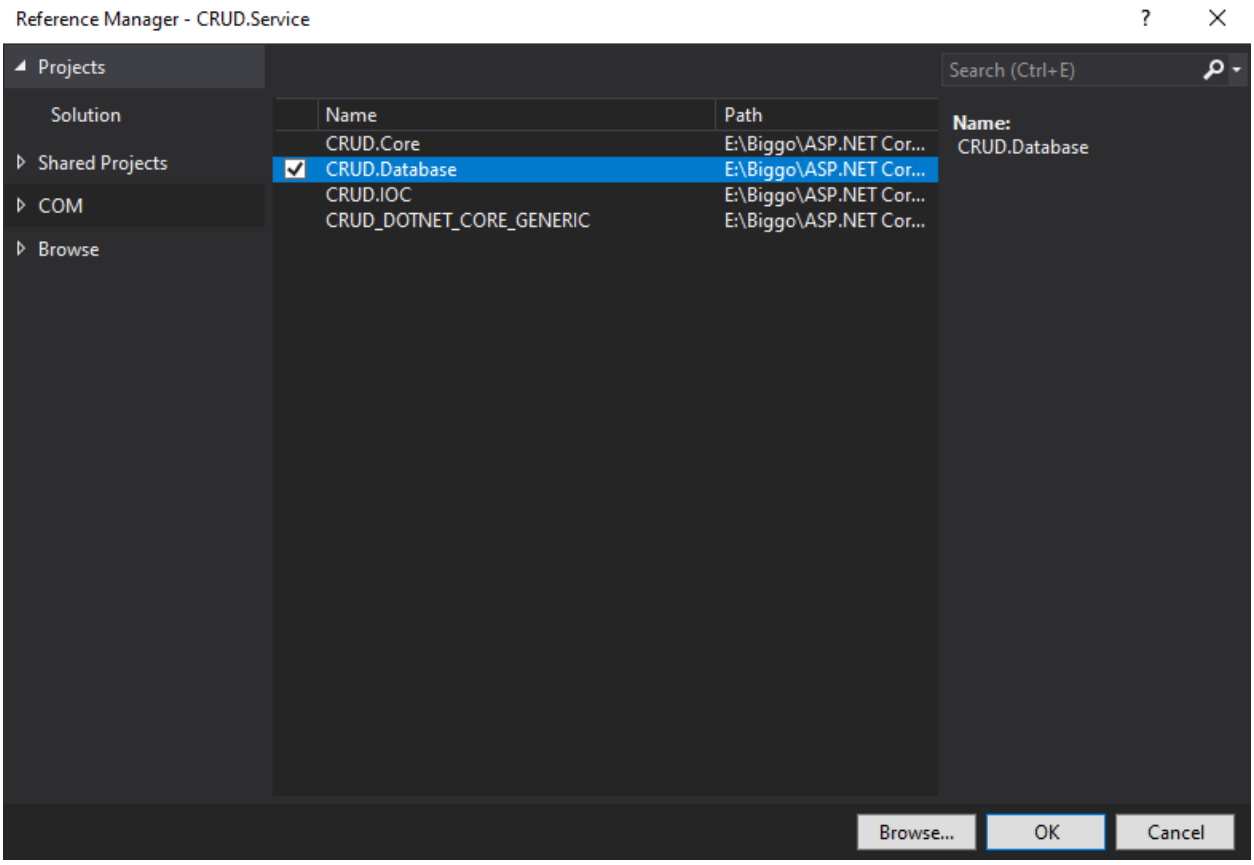
Adding Dependencies

- Right click on the project > Manage NuGet Packages…
- Search for the following and install them
  - Microsoft.Extensions.Configuration
  - Microsoft.NETCore.App



Adding reference

- Add reference of CRUD.Database to the CRUD.Service class library project.
- Right click on the Dependencies add reference and tick 'CRUD.Database' and click OK.



AppSettings.cs

- Create a class named 'AppSettings.cs' where we can fetch the connection string from the main application.

```
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Text;

namespace CRUD.Service
{
    public class AppSettings
    {
        public IConfiguration _configuration;
        public AppSettings(IConfiguration configuration)
        {
            _configuration = configuration;
        }
        public string conString()
        {
            return _configuration["ConnectionStrings:DefaultConnection"];
        }
    }
}
```

Interface

- Create directory named 'Interface'
- In this directory all the implementation of the project will be declared.
- This follows ***Interfaces Segregation Principle*** of the **SOLID** principles.
- We will create two interface named '`IDepartmentService`' and 'IEmployeeService' where the business logics of the projects will be declared but not implemented.

IDepartmentService.cs

- Right click on the department directory and add two interface named '`IDepartmentService.cs`' and '`IEmployeeService.cs`'. you can add as many interface as per your requirement.
- Add the following lines of code or customize them on your need.

```
using CRUD.Database.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace CRUD.Service.Interface
{
    public interface IDepartmentService
    {
        List<Department> GetAllDepartments();
        Department GetDepartmentId(int id);
        Department GetDepartmentByName(string name);
        void CreateDepartment(Department department);
        void UpdateDepartment(Department department);
    }
}
```

IEmployeeService

```
using CRUD.Database.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace CRUD.Service.Interface
{
    public interface IEmployeeService
    {
        List<Employee> GetAllEmployees();
        Employee GetEmployeeById(int id);
        Employee GetEmployeeByname(string name);
        void CreateEmployee(Employee employee);
        void UpdateEmployee(Employee employee);
        void DeleteEmployee(Employee employee);
    }
}
```

Implementation

- Create directory named 'Implementation'
- In this directory all the implementation which were declared on the interface will be implemented.
- This follows the ***'Dependency Inversion Principle'*** of the **SOLID** principles.
- We will create two class named '`DepartmentService.cs`' and '' which will inherit the declaration from the interface that were created before.

DepartmentService

```
using CRUD.Database.Models;
using CRUD.Database.UnitOfWork;
using CRUD.Service.Interface;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CRUD.Service.Implementation
{
    public class DepartmentService : IDepartmentService
    {
        private readonly IUnitOfWork _unitOfWork;
        private AppSettings _appSettings;

        public DepartmentService(IUnitOfWork unitOfWork, AppSettings appSettings)
        {
            _unitOfWork = unitOfWork;
            _appSettings = appSettings;
        }

        public List<Department> GetAllDepartments()
        {
            var list = _unitOfWork.Repository<Department>().GetAll().ToList();
            return list;
        }

        public Department GetDepartmentId(int id)
        {
            return _unitOfWork.Repository<Department>().GetFirstOrDefault(a => a.DeptId == id);
        }

        public Department GetDepartmentByName(string name)
        {
            return _unitOfWork.Repository<Department>().GetFirstOrDefault(a => a.DeptName.Trim() == name.Trim());
        }

        public void CreateDepartment(Department department)
        {
            _unitOfWork.Repository<Department>().Insert(department);
            _unitOfWork.Save();
        }

        public void UpdateDepartment(Department department)
        {
            _unitOfWork.Repository<Department>().Update(department);
            _unitOfWork.Save();
        }
    }
}
```

EmployeeService

```
using CRUD.Database.Models;
using CRUD.Database.UnitOfWork;
using CRUD.Service.Interface;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CRUD.Service.Implementation
{
    public class EmployeeService : IEmployeeService
    {
        private readonly IUnitOfWork _unitOfWork;
        private AppSettings _appSettings;

        public EmployeeService(IUnitOfWork unitOfWork, AppSettings appSettings)
        {
            _unitOfWork = unitOfWork;
            _appSettings = appSettings;
        }

        public List<Employee> GetAllEmployees()
        {
            var list = _unitOfWork.Repository<Employee>().GetAll().ToList();
            return list;
        }

        public Employee GetEmployeeById(int id)
        {
            return _unitOfWork.Repository<Employee>().GetFirstOrDefault(a => a.EmpId == id);
        }

        public Employee GetEmployeeByname(string name)
        {
            return _unitOfWork.Repository<Employee>().GetFirstOrDefault(a => a.EmpName.Trim() ==
name.Trim());
        }

        public void CreateEmployee(Employee employee)
        {
            _unitOfWork.Repository<Employee>().Insert(employee);
            _unitOfWork.Save();
        }

        public void UpdateEmployee(Employee employee)
        {
            _unitOfWork.Repository<Employee>().Update(employee);
            _unitOfWork.Save();
        }

        public void DeleteEmployee(Employee employee)
        {
            _unitOfWork.Repository<Employee>().Delete(employee);
            _unitOfWork.Save();
        }
    }
}
```

CRUD_DOTNET_CORE_GENERIC

Purpose

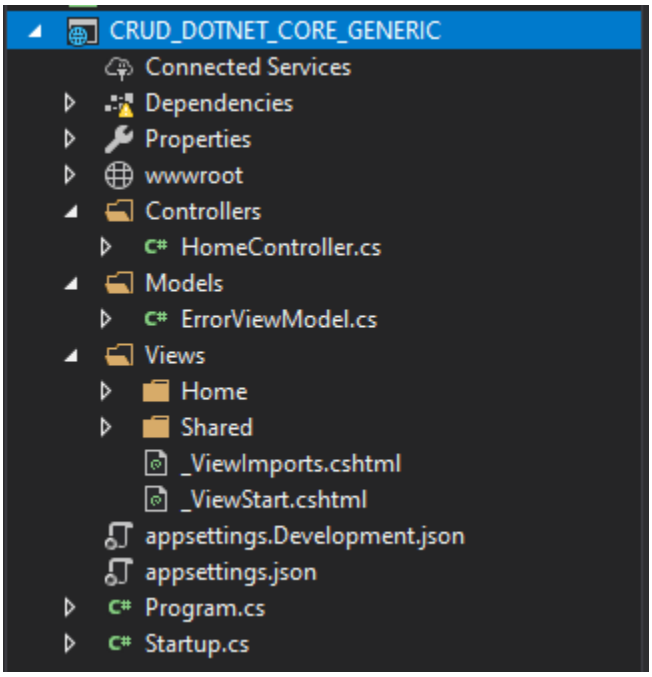
- This is the main project.
- We kept isolated the Data Access Layer and Business Layer in different class library project.
- We will add the reference of the Data Access Layer and Business Layer in this project.
- Here we have Models, Views and Controller.
- Besides we have wwwroot folder where we keep all the libraries of css and javascripts.
- If we started the project with angular we had a directory named 'src' where we need to implement the front end logic.
- But as we're using razor view, we will stick to .cshtml files.

Scope of improvement

- CORS policy.
- Adding JWT.
- Exposing APIs.
- Adding Symmetric or Asymmetric encryption.
- Adding Swagger tools for documenting APIs built on ASP.NET Core

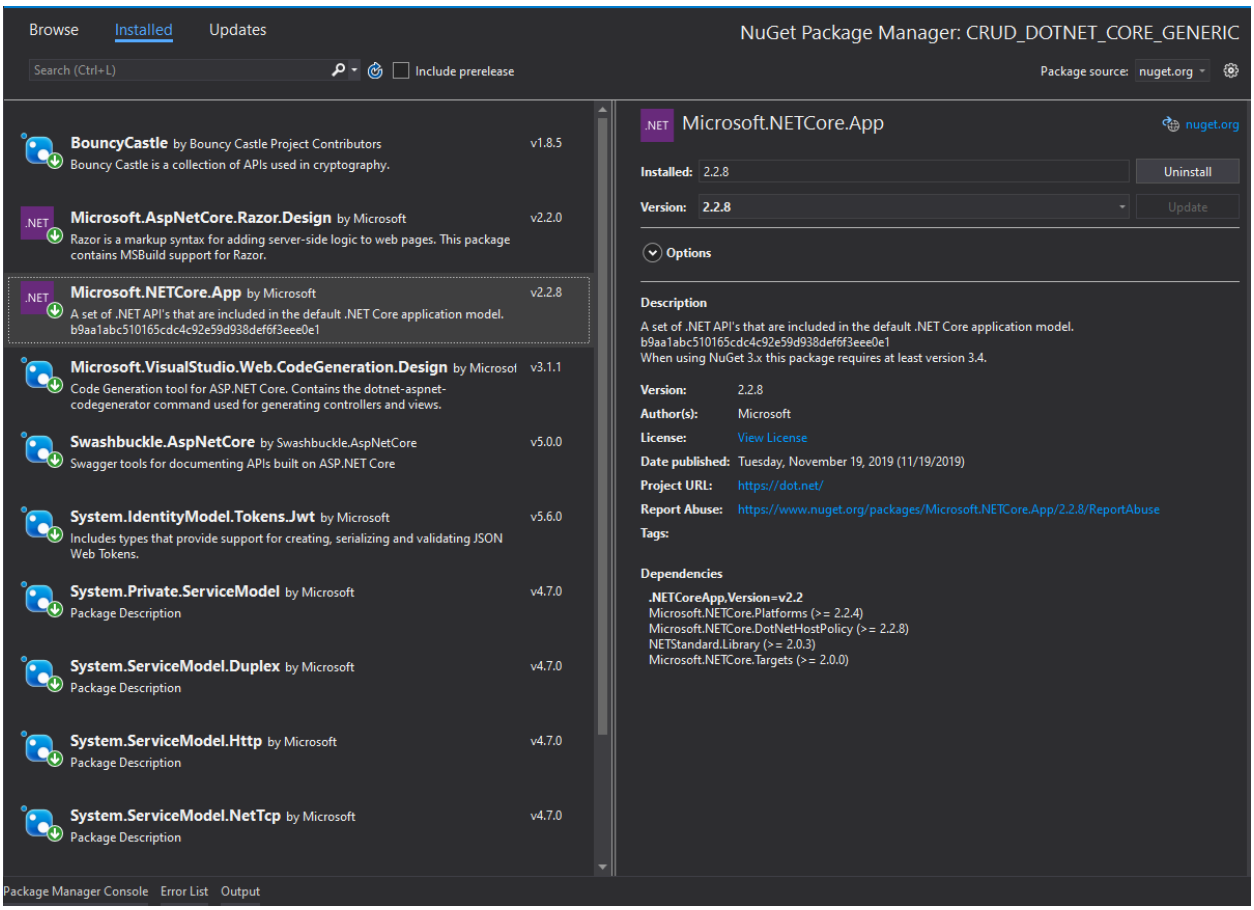
Default Project Structure

- The following screenshot is the default project structure of the project.



Adding Dependencies

- Right click on the project > Manage NuGet Packages…
- Search for the following and install them
  - BouncyCastle
  - Microsoft.AspNetCore.Razor.Design
  - Microsoft.NETCore.App
  - Microsoft.VisualStudio.Web.CodeGeneration.Design
  - Swashbuckle.AspNetCore
  - System.IdentityModel.Tokens.Jwt
  - System.Private.ServiceModel
  - System.ServiceModel.Duplex
  - System.ServiceModel.Http
  - System.ServiceModel.NetTcp
  - System.ServiceModel.Security



  - If you face problem any problem regarding build remove the ‘Microsoft.AspNetCore.Razor.Design’ from the NuGet package manager.

Adding Reference

- Add reference of CRUD.Database and CRUD.Service class library project.
- Right click on the Dependencies add reference and tick 'CRUD.Database', 'CRUD.Service' and click OK.