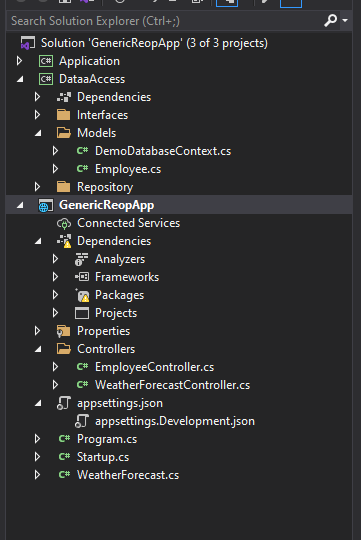
**Generic repository pattern with UnitOfWork in .net core web api**

1. Create **.Net core web api** project
2. Create **DataAccess** .net core class library into solution



1. Install below library into **DataAccess class** library using **NuGet package manager**

**Microsoft.EntityFrameworkCore.SqlServer**

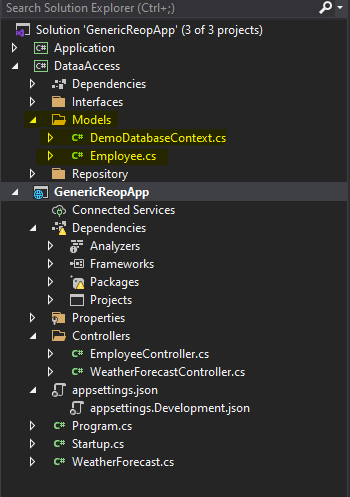
**Microsoft.EntityFrameworkCore.SqlServer.Design**

**Microsoft.EntityFrameworkCore.Tools**

1. Create **Model** folder in **DataAccess** class library
2. Execute below query using **Package manager console** to generate **DBContext** class and **entities**

**PM> Scaffold-DbContext "Server=DESKTOP-3DF0FM6\SQLEXPRESS;Database=DemoDatabase;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models**

After executing this command it will generate below files in Models folder



1. Open **DemoDatabaseContext.cs** file and comment the **OnConfiguring** method
2. Add connection string in **appsettings.json** file

"ConnectionStrings": {

"DemoDBConnection": "Server=DESKTOP-3DF0FM6\\SQLEXPRESS;Database=DemoDatabase;Trusted\_Connection=True"

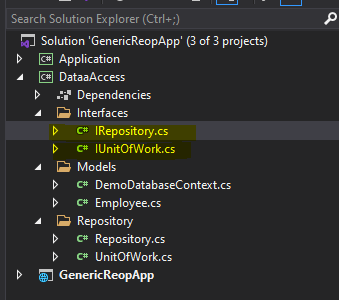
},

1. Open **startup.cs** file and add below into **ConfigureServices()** method

**services.AddDbContext<DemoDatabaseContext>(item => item.UseSqlServer**

**(Configuration.GetConnectionString("DemoDBConnection")));**

1. Create **Interface** folder into **DataAccess** class library and add below interface into this folder-



public interface IRepository<T> where T : class

{

Task<IEnumerable<T>> Get();

Task<IEnumerable<T>> Get(Expression<Func<T, bool>> predicate);

T GetById(int id);

Task Add(T entity);

void Update(T entity);

void Delete(T entity);

int SaveChanges();

Task<int> SaveChangesAsync();

}

public interface IUnitOfWork : IDisposable

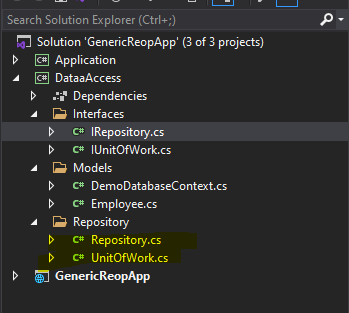
{

DbContext Context { get; }

//void Commit();

}

1. Create **Repository** folder into **DataAccess** class library and add below **classes** into this folder-



public class UnitOfWork : IUnitOfWork

{

public DbContext Context { get; }

public UnitOfWork(DemoDatabaseContext context)

{

Context = context;

}

//public void Commit()

//{

// Context.SaveChanges();

//}

public void Dispose()

{

Context.Dispose();

}

}

public class Repository<T> : IRepository<T> where T : class

{

private readonly IUnitOfWork \_unitOfWork;

protected DbSet<T> dbSet;

protected DbContext dbContext;

public Repository(IUnitOfWork unitOfWork)

{

//\_unitOfWork = unitOfWork;

\_unitOfWork = unitOfWork;

if(\_unitOfWork.Context != null)

{

dbContext = \_unitOfWork.Context;

dbSet = dbContext.Set<T>();

}

}

public async Task<IEnumerable<T>> Get()

{

//return \_unitOfWork.Context.Set<T>().AsEnumerable<T>();

return await dbSet.ToListAsync();

}

public async Task<IEnumerable<T>> Get(System.Linq.Expressions.Expression<Func<T,

bool>> predicate)

{

//return \_unitOfWork.Context.Set<T>().Where(predicate).AsEnumerable<T>();

return await dbSet.Where(predicate).ToListAsync();

}

public T GetById(int id)

{

return dbSet.Find(id);

}

public async Task Add(T entity)

{

// \_unitOfWork.Context.Set<T>().Add(entity);

await dbSet.AddAsync(entity);

}

public void Update(T entity)

{

// \_unitOfWork.Context.Set<T>().Update(entity);

dbSet.Update(entity);

}

public void Delete(T entity)

{

//T existing = \_unitOfWork.Context.Set<T>().Find(entity);

//if (existing != null) \_unitOfWork.Context.Set<T>().Remove(existing);

//T existing = dbSet.Find(entity);

//if (existing != null)

//{

// dbSet.Remove(existing);

//}

dbSet.Remove(entity);

}

public int SaveChanges()

{

return \_unitOfWork.Context.SaveChanges();

}

public async Task<int> SaveChangesAsync()

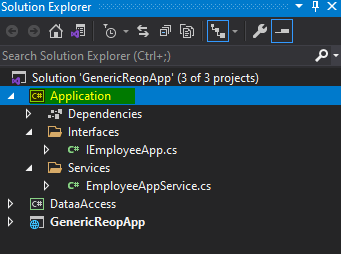
{

return await \_unitOfWork.Context.SaveChangesAsync();

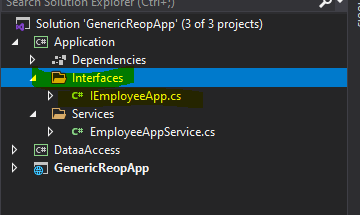
}

}

1. Create **Application ->** .net core class library into solution



1. Create **Interface** folder into **Application** class library and add below interface into this folder-



public interface IEmployeeApp

{

Task<IEnumerable<Employee>> GetEmployeeList();

Task<IEnumerable<Employee>> GetEmployeeById(Expression<Func<Employee, bool>>

predicate);

Employee GetEmployeeById(int id);

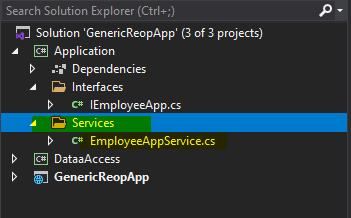
Task<string> AddEmployee(Employee employee);

Task<string> UpdateEmployee(Employee employee);

Task<string> DeleteEmployee(int id);

}

1. Create **Services** folder into **Application** class library and add below service into this folder-



public class EmployeeAppService : IEmployeeApp

{

public readonly IRepository<Employee> \_employeeRepository;

public EmployeeAppService(IRepository<Employee> employeeRepository)

{

\_employeeRepository = employeeRepository;

}

public async Task<IEnumerable<Employee>> GetEmployeeList()

{

var result = await \_employeeRepository.Get();

if (result == null)

{

throw new Exception("error");

}

return result;

}

public async Task<IEnumerable<Employee>>

GetEmployeeById(Expression<Func<Employee, bool>> predicate)

{

var result = await \_employeeRepository.Get(predicate);

if (result == null)

{

throw new Exception("error");

}

return result;

}

public Employee GetEmployeeById(int id)

{

if (id <= 0)

{

throw new Exception("error");

}

var result = \_employeeRepository.GetById(id);

if (result == null)

{

throw new Exception("error");

}

return result;

}

public async Task<string> AddEmployee(Employee employee)

{

if (employee == null)

{

throw new Exception("error");

}

await this.\_employeeRepository.Add(employee);

var result = await \_employeeRepository.SaveChangesAsync();

if (result <= 0)

{

return "Record does not saved";

}

return "Record Saved";

}

public async Task<string> UpdateEmployee(Employee employee)

{

if (employee.EmpId <= 0)

{

throw new Exception("error");

}

this.\_employeeRepository.Update(employee);

var result = await \_employeeRepository.SaveChangesAsync();

if (result <= 0)

{

return "Record does not updated";

}

return "Record Updated";

}

public async Task<string> DeleteEmployee(int id)

{

if (id <= 0)

{

throw new Exception("error");

}

var employee = \_employeeRepository.GetById(id);

if (employee == null)

{

throw new Exception("error");

}

\_employeeRepository.Delete(employee);

var result = await \_employeeRepository.SaveChangesAsync();

if (result <= 0)

{

return "Record does not deleted";

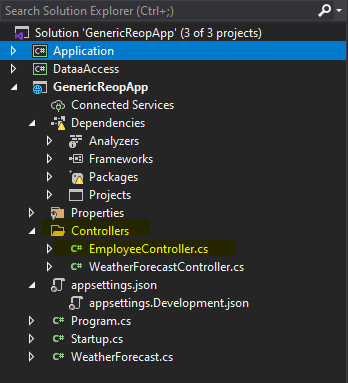
}

return "Record Deleted";

}

}

1. Open **.Net core project** which we have created **in step 1**, Add **Employeecontroller.cs**



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

[ApiController]

public class EmployeeController : ControllerBase

{

public readonly IEmployeeApp \_employeeService;

public EmployeeController(IEmployeeApp employeeService)

{

this.\_employeeService = employeeService;

}

[HttpGet("api/Employee/GetEmployeeList")]

public async Task<IActionResult> Get()

{

var result = await \_employeeService.GetEmployeeList();

return Ok(result);

}

[HttpGet("api/Employee/GetEmployeeById/{id}")]

public async Task<IActionResult> GetEmployeeById(int id)

{

var result = await \_employeeService.GetEmployeeById(x => x.EmpId == id);

return Ok(result);

}

[HttpPost("api/Employee/SaveEmployee")]

public async Task<IActionResult> SaveEmployee(Employee employee)

{

var result = await \_employeeService.AddEmployee(employee);

return Ok(result);

}

[HttpPost("api/Employee/UpdateEmployee")]

public async Task<IActionResult> UpdateEmployee(Employee employee)

{

var result = await \_employeeService.UpdateEmployee(employee);

return Ok(result);

}

[HttpPost("api/Employee/DeleteEmployee")]

public async Task<IActionResult> DeleteEmployee(int id)

{

var result = await \_employeeService.DeleteEmployee(id);

return Ok(result);

}

}

1. Open **startup.cs** file and inject dependency injection into **ConfigureServices** method

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<DemoDatabaseContext>(item => item.UseSqlServer

(Configuration.GetConnectionString("DemoDBConnection")));

services.AddControllers();

**services.AddScoped<IUnitOfWork, UnitOfWork>();**

**services.AddScoped(typeof(IRepository<>), typeof(Repository<>));**

**services.AddScoped<IEmployeeApp, EmployeeAppService>();**

services.AddSwaggerGen();

}

1. To enable **swagger**. Install the below command using **Package manager console**

PM> Install-Package Swashbuckle.AspNetCore -Version 3.1.1

1. Register below **swagger** code to **startup.cs file**

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<DemoDatabaseContext>(item => item.UseSqlServer

(Configuration.GetConnectionString("DemoDBConnection")));

services.AddControllers();

services.AddScoped<IUnitOfWork, UnitOfWork>();

services.AddScoped(typeof(IRepository<>), typeof(Repository<>));

services.AddScoped<IEmployeeApp, EmployeeAppService>();

**services.AddSwaggerGen();**

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

**app.UseSwagger();**

//Enable middleware to serve swagger - ui(HTML, JS, CSS, etc.),

// specifying the Swagger JSON endpoint.

**app.UseSwaggerUI(c =>**

**{**

**c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");**

**});**

}