Unit of work

به زبان ساده unit of work به عنوان یک مدیر عمل می کند که تغییرات انجام شده بر روی داده ها را دنبال می کند زمانی که از دیتابیس استفاده می کنید. و زمانی که کار شما تمام شد تمام تغییرات را یک باره ذخیره می کند. این الگو با استفاده از یک کلاس مشترک databse context مدیریت چندین repository را ساده می کند.

اساسا به ما کمک می کند تا coordinate کردن کار چندین repository را انجام دهیم. یعنی هماهنگ کردن.

مثال

فرض کنید که چندین entity داریم با نام های book , player , student و حالا می خواهیم repository این ها را به همراه interface مربوطبه در program.cs تزریق کنیم. اما با استفاده از الگوی unit of work می توانیم فقط آن را به programs.cs معرفی کنیم و استفاده کنیم.

Without Unit Of Work

//...

// add repositories and their interfaces to the DI container

builder.Services.AddScoped<IPlayerRepository, PlayerRepository>();

builder.Services.AddScoped<IBookRepository, BookRepository>();

builder.Services.AddScoped<IStudentRepository, StudentRepository>();

//...

With Unit Of Work

//...

// add UnitOfWork and its interface to the DI container

builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();

//...

**IUnitOfWork**

یک interface است که از property های تشکیل می شود که هر کدام از آن ها از نوع interface مربوط به repository هر کدام از entity ها است همچنین یک متد برای ذخیره کرن تغییرات نیز دارد.

public interface IUnitOfWork

{

IPlayerRepository Players { get; }

// we have only get because we don't want to set the repository. setting the repository will be done in the UnitOfWork class

Task CompleteAsync(); // this method will save all the changes made to the database {

**UnitOfWork Implementation**

پیاده سازی مربوط به interface را انجام می دهد

public class UnitOfWork:IUnitOfWork, IDisposable // IDisposable is used to free unmanaged resources

{

private readonly DataContext \_context;

private readonly ILogger \_logger;

public IPlayerRepository Players { get; private set; }

// constructor will take the context and logger factory as parameters

public UnitOfWork(

DataContext context,

ILoggerFactory loggerFactory

)

{

\_context = context;

\_logger = loggerFactory.CreateLogger("logs");

Players = new PlayerRepository(\_context, \_logger);

}

public async Task CompleteAsync()

{

await \_context.SaveChangesAsync();

}

public void Dispose()

{

\_context.Dispose();

}

اکنون با تزریق unitofwork به لایه های مثل controller می توانیم دسترسی کاملی به دیتا های خود داشته باشیم.

Medium

امروزه در توسعه نرم افزار اطمینان از کارایی مدیریت داده ها و ثبات آن ها می تواند تضمین یک نرم افزار قوی باشد. که الگوی unit of work به توسعه دهندگان این امکان را می دهد که به این موهم دست یابند در زمانی که با Database سرکار دارند

تعریف :

یک الگو است که برای مدیریت تراکنش های دیتابیس هماهنگی تغییرات بین entity های مختلف در اپلیکیشن کاربرد دارد. در سی شارپ این الگو پلی است میان لایه business و data access که ارائه می دهد یک مکانیسم مرکزی برای change tracking و اطمینان از atomic بودن و commit و rollback

این الگو به همراه repository پیاده سازی می شود.

public interface IRepository<T> where T : class  
{  
 IQueryable<T> GetAll();  
 void Add(T entity);  
 void Remove(T entity);  
}  
  
public class Repository<T> : IRepository<T> where T : class  
{  
 private readonly DbSet<T> \_dbSet;  
  
 public Repository(DbSet<T> dbSet)  
 {  
 \_dbSet = dbSet;  
 }  
  
 public IQueryable<T> GetAll()  
 {  
 return \_dbSet.AsQueryable();  
 }  
  
 public void Add(T entity)  
 {  
 \_dbSet.Add(entity);  
 }  
  
 public void Remove(T entity)  
 {  
 \_dbSet.Remove(entity);  
 }  
}

**Advantages of Using Unit of Work in C#:**

**ثبات در تغییرات داده ها : این اطمینان را میدهد که تغییرات در چندین entity باید commit یا rollback شوند و یکپارچگی داده ها حفظ می شود.**

**کاهش فراخوانی db: با استفاده از change tracker و unitOfWork و commit کردن در یک transaction تعداد فراخوانی های db کاهش یافته و سبب افزاریش کارایی می شود.**

**تمرکز بر business : توسعه دهندگان به جای تمرکز بر منطق تراکنش و پیچیدگی های آن بر روی بیزنس تمرکز می کنند.**

**rollback راحت : در صورتی که خطای رخ دهد امکان rollback راحت وجود دارد.**

**c-sharpcorner**

**The Repository Pattern**

ریپوزیتوری چیزی نیست جز یک class که برای یک entity تعریف می شود به همراه تمامی operation های که برای آن entity ممکن است. به طور مثال برای موجودیت customer تمام متد های مورد نیاز برای عملیات crud را باید repository داشته باشد و به روش generic و non generic قابل پیاده سازی است.

یکی از مزیت های unit of work این است که می توان تمامی عملیات های insert update delete را در یک transaction انجام داد به جای چندین فراخوانی db