


Паттерны слоя данных



Repository

- ▶ паттерн, задача - управление доступом к источнику данных (содержит операции над данными или реализует CRUD-интерфейс)

может работать с разными сущностями



```
public interface IGenericRepository<TEntity> where TEntity : class
{
    void Create(TEntity item);
    TEntity FindById(int id);
    IEnumerable<TEntity> Get();
    IEnumerable<TEntity> Get(Func<TEntity, bool> predicate);
    void Remove(TEntity item);
    void Update(TEntity item);
}
```

позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.


► базовая реализация для репозитория

```
public class EFGenericRepository<TEntity> : IGenericRepository<TEntity> where TEntity :  
class
```

```
{
```

```
    DbContext _context;  
    DbSet<TEntity> _dbSet;
```

ссылка на КОНТЕКСТ
набор DbSet



```
    public EFGenericRepository(DbContext context)  
    {  
        _context = context;  
        _dbSet = context.Set<TEntity>();  
    }
```

```
    public IEnumerable<TEntity> Get()  
    {  
        return _dbSet.AsNoTracking().ToList();  
    }
```

```
    public IEnumerable<TEntity> Get(Func<TEntity, bool> predicate)  
    {  
        return _dbSet.AsNoTracking().Where(predicate).ToList();  
    }
```

```
    public TEntity FindById(int id)  
    {  
        return _dbSet.Find(id);  
    }
```

```
    public void Create(TEntity item)  
    {
```

Преимущества

- ▶ гибкость при работе с разными типами подключений
- ▶ слой абстракции поверх слоя распределения данных (Data Mapper) с запросами
- ▶ сокращение дублирования кода запросов

```
EFGenericRepository<User> userRepo =  
    new EFGenericRepository<User>(new MyDBContext());
```

- ▶ Если репозитории используют одно и то же подключение, то для организации доступа к одному подключению для всех репозиториях приложения используется паттерн - **Unit Of Work**

содержит набор репозиториях и ряд некоторых общих для них функций

```
public class MyDBContext : DbContext  
{  
    public DbSet<User> Users { get; set; }  
    public DbSet<Company> Companies { get; set; }  
}
```

Repository + Unit Of Work

► 1) определяем модели

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Company
{
    public int Id { get; set; }
    public string CName { get; set; }
    public Student Student { get; set; }
}
```

► 2) контекст + паттерн Репозиторий

```
public class StudentContext : DbContext
{
    public DbSet<Student> Students { get; set; }
    public DbSet<Company> Companies { get; set; }
}

interface IRepository<T> where T : class
{
    IEnumerable<T> GetAll();
    T Get(int id);
    void Create(T item);
    void Update(T item);
    void Delete(int id);
}
```

► 3) Реализация репозитория

```
public class StudentRepository : IRepository<Student>
{
    private StudentContext db;

    public StudentRepository(StudentContext context)
    {
        this.db = context;
    }

    public IEnumerable<Student> GetAll()
    {
        return db.Students;
    }

    public Student Get(int id)
    {
        return db.Students.Find(id);
    }

    public void Create(Student student)
    {
        db.Students.Add(student);
    }

    public void Update(Student student)
    {
        db.Entry(student).State = EntityState.Modified;
    }

    public void Delete(int id)
    {
        Student student = db.Students.Find(id);
        if (student != null)
            db.Students.Remove(student);
    }
}
```


► 3) Реализация репозитория

```
public class CompanyRepository : IRepository<Company>
{
    private StudentContext db;

    public CompanyRepository(StudentContext context)
    {
        this.db = context;
    }

    public IEnumerable<Company> GetAll()
    {
        return db.Companies.Include(o => o.Student);
    }

    public Company Get(int id)
    {
        return db.Companies.Find(id);
    }

    public void Create(Company order)
    {
        db.Companies.Add(order);
    }

    public void Update(Company order)
    {
        db.Entry(order).State = EntityState.Modified;
    }

    public void Delete(int id)
    {
        Company order = db.Companies.Find(id);
        if (order != null)
            db.Companies.Remove(order);
    }
}
```

► 4) предоставляет доступ к репозиториям через отдельные свойства и определяет общий контекст для обоих репозиториев

```
public class UnitOfWork : IDisposable
{
    private StudentContext db = new StudentContext();
    private StudentRepository studentRepository;
    private CompanyRepository companyRepository;

    public StudentRepository Students
    {
        get
        {
            if (studentRepository == null)
                studentRepository = new StudentRepository(db);
            return studentRepository;
        }
    }

    public CompanyRepository Company
    {
        get
        {
            if (companyRepository == null)
                companyRepository = new CompanyRepository(db);
            return companyRepository;
        }
    }

    public void Save()
    {
        db.SaveChanges();
    }

    public void Dispose()
```

Хэширование паролей

```
{  
    public class SaltedHash  
    {  
        public string Hash { get; private set; }  
        public string Salt { get; private set; }  
  
        public SaltedHash(string password)  
        {  
            var saltBytes = new byte[32];  
            new Random().NextBytes(saltBytes);  
            Salt = Convert.ToBase64String(saltBytes);  
            var passwordAndSaltBytes = Concat(password, saltBytes);  
            Hash = ComputeHash(passwordAndSaltBytes);  
        }  
    }  
}
```

```

static string ComputeHash(byte[] bytes)
{
    using (var sha256 = SHA256.Create())
    {
        return
Convert.ToBase64String(sha256.ComputeHash(bytes));
    }
}

static byte[] Concat(string password, byte[] saltBytes)
{
    var passwordBytes = Encoding.UTF8.GetBytes(password);
    return passwordBytes.Concat(saltBytes).ToArray();
}

```

Salt	Password
U9hgysRNNIUP3dbaXwMmsiEgbrE4qqGdDYwQatzAu20=	98JEex/cQgeC5SiBIWmosK7sCR84vDt4gKh1bMR9x9o=
U9hgysRNNIUP3dbaXwMmsiEgbrE4qqGdDYwQatzAu20=	WYi36U5doBKg+Kbl9SvO4SR2Kk6+wLE1NQemqPTG3HI=

```
public static bool Verify(string salt, string hash, string password)
{
    var saltBytes = Convert.FromBase64String(salt);
    var passwordAndSaltBytes = Concat(password, saltBytes);
    var hashAttempt = ComputeHash(passwordAndSaltBytes);
    return hash == hashAttempt;
}
}
```