МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Программирование в среде dotNET»

Тема: «РАЗРАБОТКА СЛОЯ ДОСТУПА К ДАННЫМ ПРИЛОЖЕНИЯ»

Студент гр. 6305 _____ Белоусов Е.О.

Преподаватель _____ Пешехонов К.А.

Санкт-Петербург

2020

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ И ЗАДАНИЕ

Цель работы: реализовать слой доступа к данным приложения.

Постановка задания:

1. Ознакомиться с принципом работы Entity Framework Code First.
2. Реализовать слой доступа к данным.

ХОД РАБОТЫ

Слой доступа к данным

Слой доступа к данным (Data Access Layer, DAL) является важной составляющей приложения, поскольку с помощью него мы взаимодействуем с нашей базой данных. Для упрощения работы .NET Core предоставляет сильное ORM-решение – Entity Framework (EF). EF позволяет автоматически связать обычные классы языка C# с таблицами в нашей базе данных. В первую очередь, EF Core нацелен на работу с MS SQL Server, хотя и поддерживает работу с рядом других систем управления базами данных (СУБД).

В данном проекте используется MS SQL Server Express 2019 и Entity Framework Core (в DAL добавлены следующие зависимости: `EntityFrameworkCore`, `EntityFrameworkCore.Design`, `EntityFrameworkCore.SqlServer`).

В отдельных файлах опишем поведение наших моделей.

Реализация `MusicConfiguration.cs`:

```
using MusicApp.Core.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MusicApp.DAL.Configurations
{
    public class MusicConfiguration : IEntityTypeConfiguration<Music>
    {
        public void Configure(EntityTypeBuilder<Music> builder)
        {
            const int maxLength = 50;

            builder.HasKey(m => m.Id);

            builder.Property(m => m.Id)
                   .UseIdentityColumn();

            builder.Property(m => m.Name)
                   .IsRequired()
                   .HasMaxLength(maxLength);

            builder.HasOne(m => m.Artist)
                   .WithMany(a => a.Musics)
                   .HasForeignKey(m => m.ArtistId);

            builder.ToTable("Musics");
        }
```

```
    }
}
```

Реализация `ArtistConfiguration.cs`:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using MusicApp.Core.Models;

namespace MusicApp.DAL.Configurations
{
    public class ArtistConfiguration : IEntityTypeConfiguration<Artist>
    {
        public void Configure(EntityTypeBuilder<Artist> builder)
        {
            const int maxLength = 50;

            builder.HasKey(a => a.Id);

            builder.Property(m => m.Id)
                   .UseIdentityColumn();

            builder.Property(m => m.Name)
                   .IsRequired()
                   .HasMaxLength(maxLength);

            builder.ToTable("Artists");
        }
    }
}
```

Теперь можем реализовать DbContext – класс, определяющий контекст данных и используемый для взаимодействия с базой данных.

Реализация `MusicAppDbContext.cs`:

```
using Microsoft.EntityFrameworkCore;
using MusicApp.Core.Models;
using MusicApp.DAL.Configurations;

namespace MusicApp.DAL
{
    public class MusicAppDbContext : DbContext
    {
        public DbSet<Music> Musics { get; set; }
        public DbSet<Artist> Artists { get; set; }

        public MusicAppDbContext(DbContextOptions<MusicAppDbContext> options) : bas
e(options) { }
```

```
        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.ApplyConfiguration(new MusicConfiguration());
            builder.ApplyConfiguration(new ArtistConfiguration());
        }
    }
}
```

Реализуем интерфейсы Repository и UnitOfWork, разработанные в Core-слое.

Реализация `Repository.cs`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using MusicApp.Core.Repositories;

namespace MusicApp.DAL.Repositories
{
    public abstract class Repository<TEntity> : IRepository<TEntity> where TEntity
: class
    {
        protected readonly DbContext Context;

        protected Repository(DbContext context)
        {
            Context = context;
        }

        public async Task AddAsync(TEntity entity)
        {
            await Context.Set<TEntity>().AddAsync(entity);
        }

        public async Task AddRangeAsync(IEnumerable<TEntity> entities)
        {
            await Context.Set<TEntity>().AddRangeAsync(entities);
        }

        public IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> predicate)
        {
            return Context.Set<TEntity>().Where(predicate);
        }

        public ValueTask<TEntity> GetByIdAsync(int id)
        {
```

```
            return Context.Set<TEntity>().FindAsync(id);
        }

        public async Task<IEnumerable<TEntity>> GetAllAsync()
        {
            return  await Context.Set<TEntity>().ToListAsync();
        }

        public void Remove(TEntity entity)
        {
            Context.Set<TEntity>().Remove(entity);
        }

        public void RemoveRange(IEnumerable<TEntity> entities)
        {
            Context.Set<TEntity>().RemoveRange(entities);
        }

        public Task<TEntity> SingleOrDefaultAsync(Expression<Func<TEntity, bool>> p
redicate)
        {
            return Context.Set<TEntity>().SingleOrDefaultAsync(predicate);
        }
    }
}
```

Реализация `MusicRepository.cs`:

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using MusicApp.Core.Models;
using MusicApp.Core.Repositories;

namespace MusicApp.DAL.Repositories
{
    public class MusicRepository : Repository<Music>, IMusicRepository
    {
        public MusicRepository(MusicAppDbContext context) : base(context) { }

        public async Task<Music> GetWithArtistByIdAsync(int id)
        {
            return await MyMusicDbContext.Musics.Include(m => m.Artist)
                                                .SingleOrDefaultAsync(m => m.Id ==
id);;
        }

        public async Task<IEnumerable<Music>> GetAllWithArtistAsync()
        {
```

```
            return await MyMusicDbContext.Musics.Include(m => m.Artist)
                                                .ToListAsync();
        }


        public async Task<IEnumerable<Music>> GetAllWithArtistByArtistIdAsync(int a
rtistId)
        {
            return await MyMusicDbContext.Musics.Include(m => m.Artist)
                                                .Where(m => m.ArtistId == artistId)
                                                .ToListAsync();
        }

        public async Task<bool> IsExists(int id)
        {
            return await GetByIdAsync(id) is {};
        }

        private MusicAppDbContext MyMusicDbContext => Context as MusicAppDbContext;
    }
}
```

Реализация `ArtistRepository.cs`:

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using MusicApp.Core.Models;
using MusicApp.Core.Repositories;

namespace MusicApp.DAL.Repositories
{
    public class ArtistRepository : Repository<Artist>, IArtistRepository
    {
        public ArtistRepository(MusicAppDbContext context) : base(context) { }

        public async Task<Artist> GetWithMusicsByIdAsync(int id)
        {
            return await MyMusicDbContext.Artists.Include(a => a.Musics)
                                                 .SingleOrDefaultAsync(a => a.Id ==
 id);
        }

        public async Task<IEnumerable<Artist>> GetAllWithMusicsAsync()
        {
            return await MyMusicDbContext.Artists.Include(a => a.Musics)
                                                 .ToListAsync();
        }

        public async Task<bool> IsExists(int id)
```

```
        {
            return await GetByIdAsync(id) is {};
        }

        private MusicAppDbContext MyMusicDbContext => Context as MusicAppDbContext;
    }
}
```

Реализация `UnitOfWork.cs`:

```
using System.Threading.Tasks;
using MusicApp.Core;
using MusicApp.Core.Repositories;
using MusicApp.DAL.Repositories;

namespace MusicApp.DAL
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly MusicAppDbContext _context;
        private MusicRepository _musicRepository;
        private ArtistRepository _artistRepository;

        public UnitOfWork(MusicAppDbContext context)
        {
            _context = context;
        }

        public IMusicRepository Musics => _musicRepository ??= new MusicRepository(
_context);

        public IArtistRepository Artists => _artistRepository ??= new ArtistReposit
ory(_context);

        public async Task<int> CommitAsync()
        {
            return await _context.SaveChangesAsync();
        }

        public void Dispose()
        {
            _context.Dispose();
        }
    }
}
```

Добавим инъекцию зависимостей (Dependency Injection) UnitOfWork, добавив следующую часть кода в файл `Startup.cs` в метод `ConfigureServices()`:

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

В данном случае, добавлена зависимость как Scoped, при этом типе в течение одного запроса в каждом обращении будет использоваться один и тот же объект сервиса.

Миграции и заполнение таблиц данными

В слое API в файл `appsettings.Development.json` добавим ConnectionStrings.

Содержание `appsettings.Development.json`:

```
{
  "ConnectionStrings": {
    "Default": "server=.\\SQLEXPRESS; database=MusicDB; user id=sa; password=28645d
b"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

Добавим MyMusicDbContext, написав в файл `Startup.cs` в метод `ConfigureServices()` следующие строки:

```
services.AddDbContext<MusicAppDbContext>(options => options.UseSqlServe
r(Configuration.GetConnectionString("Default"),
x => x.MigrationsAssembly("MusicApp.DAL")));
```

Кроме того, данный код указывает, что миграции должны выполняться в проекте MusicApp.DAL.

Сгенерируем миграции при помощи следующей команды:

```
dotnet ef --startup-project MusicApp.API/MusicApp.API.csproj migrations add
InitialModel -p MusicApp.DAL/MusicApp.DAL.csproj
```

Затем обновим нашу базу данных:

```
dotnet ef --startup-project MusicApp.API/MusicApp.API.csproj database upgrade
```

Теперь заполним базу данных непосредственно данными. Создадим пустую миграцию:

```
dotnet ef --startup-project MusicApp.API/MusicApp.API.csproj migrations add SeedMusicsAndArtistsTable -p MusicApp.DAL/MusicApp.DAL.csproj
```

В созданном файле добавим новые данные для базы:

```
using Microsoft.EntityFrameworkCore.Migrations;

namespace MusicApp.DAL.Migrations
{
    public partial class SeedMusicsAndArtistsTable : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql("INSERT INTO Artists (Name) Values ('Arctic Monkeys')");
            migrationBuilder.Sql("INSERT INTO Artists (Name) Values ('Linkin Park')");
            migrationBuilder.Sql("INSERT INTO Artists (Name) Values ('Limp Bizkit')");
            migrationBuilder.Sql("INSERT INTO Artists (Name) Values ('Gruppa Skryptonite')");

            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('R U Mine?', (SELECT Id FROM Artists WHERE Name = 'Arctic Monkeys'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Tranquility Base Hotel & Casino', (SELECT Id FROM Artists WHERE Name = 'Arctic Monkeys'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Why Did You Only Call Me When You Are High?', (SELECT Id FROM Artists WHERE Name = 'Arctic Monkeys'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('505', (SELECT Id FROM Artists WHERE Name = 'Arctic Monkeys'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Crying Lightning', (SELECT Id FROM Artists WHERE Name = 'Arctic Monkeys'))");

            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('In The End', (SELECT Id FROM Artists WHERE Name = 'Linkin Park'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Slip', (SELECT Id FROM Artists WHERE Name = 'Linkin Park'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Crawling', (SELECT Id FROM Artists WHERE Name = 'Linkin Park'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Somewhere I Belong', (SELECT Id FROM Artists WHERE Name = 'Linkin Park'))");
```

```
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('When
 They Come for Me', (SELECT Id FROM Artists WHERE Name = 'Linkin Park'))");

            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('My W
ay', (SELECT Id FROM Artists WHERE Name = 'Limp Bizkit'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Roll
in', (SELECT Id FROM Artists WHERE Name = 'Limp Bizkit'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Gold
 Cobra', (SELECT Id FROM Artists WHERE Name = 'Limp Bizkit'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Shot
gun', (SELECT Id FROM Artists WHERE Name = 'Limp Bizkit'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Brea
k Stuff', (SELECT Id FROM Artists WHERE Name = 'Limp Bizkit'))");

            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Lati
nskaya Muzyka', (SELECT Id FROM Artists WHERE Name = 'Gruppa Skryptonite'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Podr
uga', (SELECT Id FROM Artists WHERE Name = 'Gruppa Skryptonite'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Karu
sel', (SELECT Id FROM Artists WHERE Name = 'Gruppa Skryptonite'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Bere
gom', (SELECT Id FROM Artists WHERE Name = 'Gruppa Skryptonite'))");
            migrationBuilder.Sql("INSERT INTO Musics (Name, ArtistId) Values ('Glup
ye I Nenuzhnye', (SELECT Id FROM Artists WHERE Name = 'Gruppa Skryptonite'))");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql("DELETE FROM Musics");

            migrationBuilder.Sql("DELETE FROM Artists");
        }
    }
}
```

ВЫВОДЫ

В результате выполнения данной лабораторной работы был реализован слой доступа к данным, а также создана и заполнена информацией база данных.