**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра вычислительной техники**

**ОТЧЁТ**

**по лабораторной работе №3**

**по дисциплине «Программирование в среде dotNET»**

**Тема: “Разработка слоя доступа к данным приложения”**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6305 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Силинский А.С. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пешехонов К.А. |

Санкт-Петербург

2020

**Оглавление**

[**Цель работы** 3](#_Toc38253462)

[**Задание** 3](#_Toc38253463)

[**Ход работы** 3](#_Toc38253464)

[**Выводы** 3](#_Toc38253465)

[**Приложение** 4](#_Toc38253466)

# **Цель работы**

Целью данной лабораторной работы является разработка слоя доступа к данным приложения.

# **Задание**

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

# **Ход работы**

Для работы с данными необходимо создать отдельный слой доступа к данным (Data Access Layer, DAL). В этом слое необходимо будет реализовать подключение к самой базе данных. Также здесь происходит настройка конфигураций для сущностей, реализация репозиториев. Для упрощения работы с базами данных существует библиотека Entity Framework (EF). С ее помощью мы можем задать конфигурацию для сущностей (указать связи, первичные ключи) и затем подключить саму БД в слое API как зависимость через инъекцию.

Код исходных файлов данного модуля можно посмотреть в приложении к данной лабораторной работе.

# **Выводы**

В ходе выполнения данной лабораторной работы был реализован слой доступа к данным приложения.

# **Приложение**

***FestivalConfiguration.cs***

using FestivalWebApp.Core.Models;  
using Microsoft.EntityFrameworkCore;  
using Microsoft.EntityFrameworkCore.Metadata.Builders;  
  
namespace FestivalWebApp.DAL.Configurations  
{  
 public class FestivalConfiguration : IEntityTypeConfiguration<Festival>  
 {  
 private const int *FestivalNameMaxSize* = 50;  
 private const int *FestivalDescriptionMaxSize* = 200;  
  
 public void Configure(EntityTypeBuilder<Festival> builder)  
 **{** builder.HasKey(festival => festival.Id);  
  
 builder.Property(festival => festival.Id)  
 .UseIdentityColumn();  
  
 builder.Property(festival => festival.Name)  
 .IsRequired()  
 .HasMaxLength(*FestivalNameMaxSize*);  
  
 builder.Property(festival => festival.Description)  
 .HasMaxLength(*FestivalDescriptionMaxSize*);  
  
 builder.Property(festival => festival.Date)  
 .IsRequired();  
  
 builder.HasMany(f => f.Participants)  
 .WithOne(p => p.Festival)  
 .HasForeignKey(p => p.FestivalId);  
  
 builder.ToTable("Festivals");  
 **}** }  
}

***ParticipantConfiguration.cs***

using FestivalWebApp.Core.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace FestivalWebApp.DAL.Configurations

{

public class ParticipantConfiguration : IEntityTypeConfiguration<Participant>

{

private const int NameMaxSize = 50;

private const int SecondNameMaxSize = 50;

public void Configure(EntityTypeBuilder<Participant> builder)

{

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

builder.Property(participant => participant.Id)

.UseIdentityColumn();

builder.Property(participant => participant.Name)

.IsRequired()

.HasMaxLength(NameMaxSize);

builder.Property(participant => participant.SecondName)

.IsRequired()

.HasMaxLength(SecondNameMaxSize);

builder.ToTable("Participants");

}

}

}

***FestivalRepository.cs***

using System.Collections.Generic;  
using System.Threading.Tasks;  
using FestivalWebApp.Core.Models;  
using FestivalWebApp.Core.Repositories;  
using Microsoft.EntityFrameworkCore;  
  
namespace FestivalWebApp.DAL.Repositories  
{  
 public class FestivalRepository : IFestivalRepository  
 {  
 private readonly FestivalDatabaseContext \_context;  
  
 public FestivalRepository(FestivalDatabaseContext context)  
 {  
 \_context = context;  
 }  
  
 public async Task<Festival> GetFestivalById(int id)  
 {  
 return await \_context.Festivals.FindAsync(id);  
 }  
  
 public async Task<bool> IsExist(int id)  
 {  
 return await \_context.Festivals.FindAsync(id) != null;  
 }  
  
 public async Task<IEnumerable<Festival>> GetAllFestivals()  
 {  
 return await \_context.Festivals.ToListAsync();  
 }  
  
 public async Task<Festival> AddFestival(Festival festival)  
 {  
 await \_context.Festivals.AddAsync(festival);  
 await \_context.SaveChangesAsync();  
 return festival;  
 }  
  
 public async Task UpdateFestival(Festival festival)  
 {  
 \_context.Festivals.Update(festival);  
 await \_context.SaveChangesAsync();  
 }  
  
 public async Task RemoveFestival(Festival festival)  
 {  
 \_context.Festivals.Remove(festival);  
 await \_context.SaveChangesAsync();  
 }  
 }  
}

***ParticipantRepository.cs***

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FestivalWebApp.Core.Models;

using FestivalWebApp.Core.Repositories;

using Microsoft.EntityFrameworkCore;

namespace FestivalWebApp.DAL.Repositories

{

public class ParticipantRepository : IParticipantRepository

{

private readonly FestivalDatabaseContext \_context;

public ParticipantRepository(FestivalDatabaseContext context)

{

\_context = context;

}

public async Task<Participant> GetParticipantById(int id)

{

return await \_context.Participants.FindAsync(id);

}

public async Task<IEnumerable<Participant>> GetAllParticipants()

{

return await \_context.Participants.ToListAsync();

}

public async Task<IEnumerable<Participant>> GetParticipantsByFestivalId(int festivalId)

{

return await \_context.Participants.Where(p => p.FestivalId == festivalId)

.ToListAsync();

}

public async Task<Participant> AddParticipant(Participant participant)

{

await \_context.Participants.AddAsync(participant);

await \_context.SaveChangesAsync();

return participant;

}

public async Task UpdateParticipant(Participant participant)

{

\_context.Participants.Update(participant);

await \_context.SaveChangesAsync();

}

public async Task<bool> IsExist(int id)

{

return await \_context.Participants.FindAsync(id) != null;

}

public async Task RemoveParticipant(Participant participant)

{

\_context.Participants.Remove(participant);

await \_context.SaveChangesAsync();

}

}

}

***FestivalDatabaseContext.cs***

using FestivalWebApp.Core.Models;  
using FestivalWebApp.DAL.Configurations;  
using Microsoft.EntityFrameworkCore;  
  
namespace FestivalWebApp.DAL  
{  
 public class FestivalDatabaseContext : DbContext  
 {  
 public FestivalDatabaseContext(DbContextOptions<FestivalDatabaseContext> options) : base(options)  
 {  
 }  
  
 public DbSet<Festival> Festivals { get; set; }  
  
 public DbSet<Participant> Participants { get; set; }  
  
 protected override void OnModelCreating(ModelBuilder modelBuilder)  
 {  
 modelBuilder.ApplyConfiguration(new FestivalConfiguration());  
 modelBuilder.ApplyConfiguration(new ParticipantConfiguration());  
 }  
 }  
}