**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 .NET»**

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

|  |  |  |
| --- | --- | --- |
| Студент гр. 6305 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Буракаев Д. А. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пешехонов К. А. |

Санкт-Петербург

2020

**Содержание**

[**Цель работы** 3](#_Toc38639466)

[**Задание** 3](#_Toc38639467)

[**Ход работы** 4](#_Toc38639468)

[**Сущности** 4](#_Toc38639469)

[**Фреймворк сущностей** 5](#_Toc38639470)

[**Репозиторий** 7](#_Toc38639471)

[**Вывод** 10](#_Toc38639472)

# **Цель работы**

Целью данной лабораторной работы является реализация слоя доступа к данным (**DAL**) приложения.

# **Задание**

1. Ознакомиться с принципом работы Entity Framework Code First.

2. Реализовать слой доступа к данным.

# **Ход работы**

## **Сущности**

Сущности, базовые классы приложения, необходимые для селекции отдельных объектов:

*Quest.cs*

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAL.Entities

{

public class Quest

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public int PlayersLimit { get; set; }

public double Price { get; set; }

public int Duration { get; set; }

}

}

*Reserv.cs*

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAL.Entities

{

public class Reserv

{

public int Id { get; set; }

public virtual Quest Quest { get; set; }

public string DateTimeSection { get; set; }

public double Cost { get; set; }

}

}

## **Фреймворк сущностей**

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).

*DataContext.cs*

using DAL.Entities;

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAL.EF

{

public class DataContext : DbContext

{

public DbSet<Quest> Quests { get; set; }

public DbSet<Reserv> Reserves { get; set; }

static DataContext()

{

Database.SetInitializer<DataContext>(new QuestDbInitializer());

}

public DataContext(string connectionString) : base(connectionString) { }

}

public class QuestDbInitializer : DropCreateDatabaseIfModelChanges<DataContext>

{

protected override void Seed(DataContext db)

{

db.Quests.Add(new Quest { Name = "MadMax", Description = "MadMax description", Duration = 3, PlayersLimit = 12, Price = 600 });

db.Quests.Add(new Quest { Name = "New York", Description = "New York description", Duration = 2, PlayersLimit = 10, Price = 400 });

db.Quests.Add(new Quest { Name = "Integration", Description = "Integration description", Duration = 2, PlayersLimit = 4, Price = 550 });

db.Quests.Add(new Quest { Name = "Son of Sam", Description = "Son of Sam description", Duration = 2, PlayersLimit = 10, Price = 500 });

db.Quests.Add(new Quest { Name = "Deprecated", Description = "Deprecated description", Duration = 2, PlayersLimit = 8, Price = 210 });

db.Quests.Add(new Quest { Name = "Last of us", Description = "Last of us description", Duration = 1, PlayersLimit = 8, Price = 200 });

db.Quests.Add(new Quest { Name = "Injection", Description = "Injection description", Duration = 3, PlayersLimit = 12, Price = 600 });

db.Quests.Add(new Quest { Name = "All inclusive", Description = "All inclusive description", Duration = 2, PlayersLimit = 10, Price = 400 });

db.Quests.Add(new Quest { Name = "Just one minute", Description = "Just one minute description", Duration = 2, PlayersLimit = 4, Price = 550 });

db.Quests.Add(new Quest { Name = "Mirror hole", Description = "Mirror hole description", Duration = 2, PlayersLimit = 10, Price = 500 });

db.Quests.Add(new Quest { Name = "Smoothering black", Description = "Smoothering black description", Duration = 2, PlayersLimit = 8, Price = 210 });

db.Quests.Add(new Quest { Name = "Find yourself", Description = "Find yourself description", Duration = 1, PlayersLimit = 8, Price = 200 });

db.SaveChanges();

}

}

}

## **Репозиторий**

Далее при помощи описанных моделей мы можем реализовать паттерн репозиторий приложения, чтобы инкапсулировать операции. Проводимых над базой данных. Для упрощения работы с репозиториями реализуем паттерн «Unit of Work», с ним мы можем быть уверены, что все репозитории будут использовать один и тот же контекст данных.

Оба перечисленных паттерна реализуются внутри Entity Framework (EF), однако решение об их реализации внутри разрабатываемого приложения приняты с целью ослабления непосредственной связи с EF. При этом есть понимание, что применение названных паттернов может являться излишним.

*IRepository.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAL.Interfaces

{

public interface IRepository<TEntity> where TEntity : class

{

IEnumerable<TEntity> GetAll();

TEntity Get(int id);

void Create(TEntity item);

void Update(TEntity item);

void Delete(int id);

IEnumerable<TEntity> Find(Func<TEntity, bool> predicate);

}

}

*IUnitOfWork.cs*

using DAL.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAL.Interfaces

{

public interface IUnitOfWork : IDisposable

{

IRepository<Quest> Quests { get; }

IRepository<Reserv> Reserves { get; }

void Save();

}

}

*GenericRepository.cs*

using DAL.EF;

using DAL.Entities;

using DAL.Interfaces;

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAL.Repositories

{

public class GenericRepository<TEntity> : IRepository<TEntity> where TEntity : class

{

internal DataContext context;

internal DbSet<TEntity> dbSet;

public GenericRepository(DataContext context)

{

this.context = context;

this.dbSet = context.Set<TEntity>();

}

public virtual IEnumerable<TEntity> GetAll()

{

IQueryable<TEntity> query = dbSet;

return query.ToList();

}

public TEntity Get(int id)

{

return dbSet.Find(id);

}

public void Create(TEntity entity)

{

dbSet.Add(entity);

context.SaveChanges();

}

public void Update(TEntity entity)

{

context.Entry(entity).State = EntityState.Modified;

context.SaveChanges();

}

public IEnumerable<TEntity> Find(Func<TEntity, Boolean> predicate)

{

return dbSet.Where(predicate);

}

public void Delete(int id)

{

TEntity entity = dbSet.Find(id);

if (entity != null)

dbSet.Remove(entity);

context.SaveChanges();

}

}

}

*EFUnitOfWork.cs*

using DAL.EF;

using DAL.Entities;

using DAL.Interfaces;

using System;

namespace DAL.Repositories

{

public class EFUnitOfWork : IUnitOfWork

{

private DataContext db;

private GenericRepository<Quest> questRepository;

private GenericRepository<Reserv> reservRepository;

public EFUnitOfWork(string connectionString)

{

db = new DataContext(connectionString);

}

public IRepository<Quest> Quests

{

get

{

if (questRepository == null)

questRepository = new GenericRepository<Quest>(db);

return questRepository;

}

}

public IRepository<Reserv> Reserves

{

get

{

if (reservRepository == null)

reservRepository = new GenericRepository<Reserv>(db);

return reservRepository;

}

}

public void Save()

{

db.SaveChanges();

}

private bool disposed = false;

public virtual void Dispose(bool disposing)

{

if (!this.disposed)

{

if (disposing)

{

db.Dispose();

}

this.disposed = true;

}

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

}

}

# **Вывод**

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