**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе №2**

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

Тема: «РАЗРАБОТКА СЛОЯ БИЗНЕС-ЛОГИКИ ПРИЛОЖЕНИЯ»

|  |  |  |
| --- | --- | --- |
| Студент гр. 6305 |  | Белоусов Е.О. |
| Преподаватель |  | Пешехонов К.А. |

Санкт-Петербург

2020

СОДЕРЖАНИЕ

[ЦЕЛЬ РАБОТЫ И ЗАДАНИЕ 3](#_Toc37880043)

[ХОД РАБОТЫ 4](#_Toc37880044)

[Проект 4](#_Toc37880045)

[«Ядро» приложения 4](#_Toc37880046)

[Бизнес-логика приложения 7](#_Toc37880047)

[Модульное тестирование бизнес-логики приложения 10](#_Toc37880048)

[ВЫВОДЫ 27](#_Toc37880049)

# ЦЕЛЬ РАБОТЫ И ЗАДАНИЕ

***Цель работы:*** сформулировать тему проекта и реализовать слой бизнес-логики приложения.

***Постановка задания:***

1. Сформулировать тему проекта приложения ASP.NET Core 3 WebAPI.
2. Реализовать слой бизнес-логики.
3. Покрыть слой бизнес-логики модульными тестами.

# ХОД РАБОТЫ

## Проект

В качестве темы проекта выбрана «Музыкальная библиотека». В приложении должен быть следующий функционал:

* Внесение артиста/группы в базу;
* Внесение композиции в базу;
* Редактирование информации об артисте/группе;
* Редактирование информации о композиции;
* Просмотр информации об артисте/группе;
* Просмотр информации о композиции;
* Удаление артиста/группы из базы;
* Удаление композиции из базы.

## «Ядро» приложения

В процессе проектирования было принято решение вынести модели и различные интерфейсы в отдельный слой – «ядро» (Core).

В первую очередь, созданы 2 модели – Музыка и Артист.

Опишем модель Music.cs:

namespace MusicApp.Core.Models

{

    public class Music

    {

        public int Id { get; set; }

        public int ArtistId { get; set; }

        public string Name { get; set; }

        public Artist Artist { get; set; }

    }

}

Опишем модель Artist.cs:

using System.Collections.Generic;

using System.Collections.ObjectModel;

namespace MusicApp.Core.Models

{

    public class Artist

    {

        public Artist()

        {

            Musics = new Collection<Music>();

        }

        public int Id { get; set; }

        public string Name { get; set; }

        public ICollection<Music> Musics { get; set; }

    }

}

Далее при помощи описанных моделей мы можем реализовать паттерн «Репозиторий», чтобы инкапсулировать операции с 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.

Создадим интерфейс IRepository.cs:

using System;

using System.Collections.Generic;

using System.Linq.Expressions;

using System.Threading.Tasks;

namespace MusicApp.Core.Repositories

{

    public interface IRepository<TEntity> where TEntity : class

    {

        ValueTask<TEntity> GetByIdAsync(int id);

        Task<IEnumerable<TEntity>> GetAllAsync();

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

        Task<TEntity> SingleOrDefaultAsync(Expression<Func<TEntity, bool>> predicate);

        Task AddAsync(TEntity entity);

        Task AddRangeAsync(IEnumerable<TEntity> entities);

        void Remove(TEntity entity);

        void RemoveRange(IEnumerable<TEntity> entities);

    }

}

Создадим интерфейс IMusicRepository.cs:

using System.Collections.Generic;

using System.Threading.Tasks;

using MusicApp.Core.Models;

namespace MusicApp.Core.Repositories

{

    public interface IMusicRepository : IRepository<Music>

    {

        Task<Music> GetWithArtistByIdAsync(int id);

        Task<IEnumerable<Music>> GetAllWithArtistAsync();

        Task<IEnumerable<Music>> GetAllWithArtistByArtistIdAsync(int artistId);

        Task<bool> IsExists(int id);

    }

}

Создадим интерфейс IArtistRepository.cs:

using System.Collections.Generic;

using System.Threading.Tasks;

using MusicApp.Core.Models;

namespace MusicApp.Core.Repositories

{

    public interface IArtistRepository : IRepository<Artist>

    {

        Task<Artist> GetWithMusicsByIdAsync(int id);

        Task<IEnumerable<Artist>> GetAllWithMusicsAsync();

        Task<bool> IsExists(int id);

    }

}

Создадим интерфейс IUnitOfWork.cs:

using System;

using System.Threading.Tasks;

using MusicApp.Core.Repositories;

namespace MusicApp.Core

{

    public interface IUnitOfWork : IDisposable

    {

        IMusicRepository Musics { get; }

        IArtistRepository Artists { get; }

        Task<int> CommitAsync();

    }

}

Теперь создадим интерфейсы сервисов, которые в дальнейшем будут реализованы в слое бизнес-логики (Business Logic Layer, BLL) приложения.

Создадим интерфейс для обработки логики музыки IMusicService.cs:

using System.Collections.Generic;

using System.Threading.Tasks;

using MusicApp.Core.Models;

namespace MusicApp.Core.Services

{

    public interface IMusicService

    {

        Task<Music> CreateMusic(Music newMusic);

        Task<Music> GetMusicById(int id);

        Task<IEnumerable<Music>> GetAllWithArtist();

        Task<IEnumerable<Music>> GetMusicsByArtistId(int artistId);

        Task UpdateMusic(Music musicToBeUpdated, Music music);

        Task DeleteMusic(Music music);

    }

}

Создадим интерфейс для обработки логики артистов IArtistService.cs:

using System.Collections.Generic;

using System.Threading.Tasks;

using MusicApp.Core.Models;

namespace MusicApp.Core.Services

{

    public interface IArtistService

    {

        Task<Artist> CreateArtist(Artist newArtist);

        Task<Artist> GetArtistById(int id);

        Task<IEnumerable<Artist>> GetAllArtists();

        Task UpdateArtist(Artist artistToBeUpdated, Artist artist);

        Task DeleteArtist(Artist artist);

    }

}

## Бизнес-логика приложения

BLL ответственен за основную логику разрабатываемого приложения, а также за работу со слоем доступа к данным (Data Access Layer, DAL) и слоем представления данных.

Реализуем два интерфейса сервисов, созданных в Core-слое.

Реализация MusicService.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading.Tasks;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Services;

namespace MusicApp.BLL

{

    public class MusicService : IMusicService

    {

        private readonly IUnitOfWork \_unitOfWork;

        public MusicService(IUnitOfWork unitOfWork)

        {

            \_unitOfWork = unitOfWork;

        }

        public async Task<Music> CreateMusic(Music newMusic)

        {

            if (newMusic is null)

                throw new NullReferenceException();

            await \_unitOfWork.Musics.AddAsync(newMusic);

            await \_unitOfWork.CommitAsync();

            return newMusic;

        }

        public async Task<Music> GetMusicById(int id)

        {

            return await \_unitOfWork.Musics.GetWithArtistByIdAsync(id);

        }

        public async Task<IEnumerable<Music>> GetMusicsByArtistId(int artistId)

        {

            return await \_unitOfWork.Musics.GetAllWithArtistByArtistIdAsync(artistId);

        }

        public async Task<IEnumerable<Music>> GetAllWithArtist()

        {

            return await \_unitOfWork.Musics.GetAllWithArtistAsync();

        }

        public async Task UpdateMusic(int id, Music music)

        {

            if (!await \_unitOfWork.Musics.IsExists(id))

                throw new NullReferenceException();

            if (music.Name.Length <= 0 || music.Name.Length > 50 || music.ArtistId <= 0)

                throw new InvalidDataException();

            var musicToBeUpdated = await GetMusicById(id);

            musicToBeUpdated.Name = music.Name;

            musicToBeUpdated.ArtistId = music.ArtistId;

            await \_unitOfWork.CommitAsync();

        }

        public async Task DeleteMusic(Music music)

        {

            if (!await \_unitOfWork.Artists.IsExists(music.Id))

                throw new NullReferenceException();

            \_unitOfWork.Musics.Remove(music);

            await \_unitOfWork.CommitAsync();

        }

    }

}

Реализация ArtistService.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading.Tasks;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Services;

namespace MusicApp.BLL

{

    public class ArtistService : IArtistService

    {

        private readonly IUnitOfWork \_unitOfWork;

        public ArtistService(IUnitOfWork unitOfWork)

        {

            \_unitOfWork = unitOfWork;

        }

        public async Task<Artist> CreateArtist(Artist newArtist)

        {

            if (newArtist is null)

                throw new NullReferenceException();

            await \_unitOfWork.Artists.AddAsync(newArtist);

            await \_unitOfWork.CommitAsync();

            return newArtist;

        }

        public async Task<Artist> GetArtistById(int id)

        {

            return await \_unitOfWork.Artists.GetByIdAsync(id);

        }

        public async Task<IEnumerable<Artist>> GetAllArtists()

        {

            return await \_unitOfWork.Artists.GetAllAsync();

        }

        public async Task UpdateArtist(int id, Artist artist)

        {

            if (!await \_unitOfWork.Artists.IsExists(id))

                throw new NullReferenceException();

            if (artist.Name.Length == 0 || artist.Name.Length > 50)

                throw new InvalidDataException();

            var artistToBeUpdated = await GetArtistById(id);

            artistToBeUpdated.Name = artist.Name;

            await \_unitOfWork.CommitAsync();

        }

        public async Task DeleteArtist(Artist artist)

        {

            if (!await \_unitOfWork.Artists.IsExists(artist.Id))

                throw new NullReferenceException();

            \_unitOfWork.Artists.Remove(artist);

            await \_unitOfWork.CommitAsync();

        }

    }

}

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

            services.AddTransient<IMusicService, MusicService>();

            services.AddTransient<IArtistService, ArtistService>();

В данном случае созданные сервисы имеют тип Transient, при котором новый объект сервиса создаётся при каждом обращении к сервису.

## Модульное тестирование бизнес-логики приложения

Для создания модульных тестов (unit tests) используется фрэймворк NUnit - наиболее популярный фрэймворк для unit-тестирования приложений на платформе .NET, позволяющий быстро написать и автоматически проверить модульные тесты. Помимо этого, для создания мок-объектов при тестировании используется фреймворк Moq.

Для MusicService.cs были описаны тесты, представленные далее.

Содержание CreateMusicTests.cs:

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class CreateMusicTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IMusicRepository> musicRepo, Dictionary<int, Music> dbCollection) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var musicRepo = new Mock<IMusicRepository>(MockBehavior.Strict);

            var dbCollection = new Dictionary<int, Music>

            {

                [26] = new Music

                {

                    Id = 26,

                    ArtistId = 26,

                    Name = "Delete Track"

                },

                [27] = new Music

                {

                    Id = 27,

                    ArtistId = 27,

                    Name = "Track"

                }

            };

            unitOfWork.SetupGet(e => e.Musics).Returns(musicRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            musicRepo.Setup(e => e.AddAsync(It.IsAny<Music>()))

                     .Callback((Music newMusic) => { dbCollection.Add(newMusic.Id, newMusic); })

                     .Returns((Music \_) => Task.CompletedTask);

            return (unitOfWork, musicRepo, dbCollection);

        }

        [Test]

        public async Task CreateMusic\_FullInfo\_Success()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollection) = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            var music = new Music

            {

                Id = 28,

                Name = "New Track"

            };

            // Act

            await service.CreateMusic(music);

            // Assert

            Assert.IsTrue(dbCollection.ContainsKey(music.Id));

        }

        [Test]

        public void CreateMusic\_NullObject\_NullReferenceException()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollection) = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            // Act + Assert

            Assert.ThrowsAsync<NullReferenceException>(async () => await service.CreateMusic(null));

        }

    }

}

Содержание DeleteMusicTests.cs:

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class DeleteMusicTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IMusicRepository> musicRepo, Dictionary<int, Music> dbCollectionMusic) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var musicRepo = new Mock<IMusicRepository>(MockBehavior.Strict);

            var artistRepo = new Mock<IArtistRepository>(MockBehavior.Strict);

            var dbCollectionMusic = new Dictionary<int, Music>

            {

                [26] = new Music

                {

                    Id = 26,

                    ArtistId = 26,

                    Name = "Delete Track"

                },

                [27] = new Music

                {

                    Id = 27,

                    ArtistId = 27,

                    Name = "Track"

                }

            };

            var dbCollectionArtists = new Dictionary<int, Artist>

            {

                [26] = new Artist

                {

                    Id = 26,

                    Name = "Group"

                },

                [27] = new Artist

                {

                    Id = 27,

                    Name = "Other Group"

                }

            };

            unitOfWork.SetupGet(e => e.Musics).Returns(musicRepo.Object);

            unitOfWork.SetupGet(e => e.Artists).Returns(artistRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            musicRepo.Setup(e => e.IsExists(It.IsAny<int>()))

                     .ReturnsAsync((int id) => dbCollectionMusic.ContainsKey(id));

            musicRepo.Setup(e => e.Remove(It.IsAny<Music>()))

                     .Callback((Music newMusic) => { dbCollectionMusic.Remove(newMusic.Id); });

            artistRepo.Setup(e => e.IsExists(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollectionArtists.ContainsKey(id));

            artistRepo.Setup(e => e.Remove(It.IsAny<Artist>()))

                      .Callback((Artist newArtist) => { dbCollectionArtists.Remove(newArtist.Id); });

            return (unitOfWork, musicRepo, dbCollectionMusic);

        }

        [Test]

        public async Task DeleteMusic\_TargetItem\_Success()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollectionMusic) = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            var music = new Music

            {

                Id = 26,

                Name = "Delete Track"

            };

            // Act

            await service.DeleteMusic(music);

            // Assert

            Assert.IsFalse(dbCollectionMusic.ContainsKey(26));

        }

        [Test]

        public void DeleteMusic\_ItemDoesNotExists\_NullReferenceException()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollectionMusic) = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            var music = new Music

            {

                Id = 0,

                Name = "Delete Track"

            };

            // Act + Assert

            Assert.ThrowsAsync<NullReferenceException>(async () => await service.DeleteMusic(music));

        }

    }

}

Содержание GetMusicByIdTests.cs:

using System.Collections.Generic;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class GetMusicByIdTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IMusicRepository> musicRepo, Dictionary<int, Music> dbCollectionMusic) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var musicRepo = new Mock<IMusicRepository>(MockBehavior.Strict);

            var artistRepo = new Mock<IArtistRepository>(MockBehavior.Strict);

            var dbCollectionMusic = new Dictionary<int, Music>

            {

                [26] = new Music

                {

                    Id = 26,

                    ArtistId = 26,

                    Name = "Delete Track"

                },

                [27] = new Music

                {

                    Id = 27,

                    ArtistId = 27,

                    Name = "Track"

                }

            };

            var dbCollectionArtists = new Dictionary<int, Artist>

            {

                [26] = new Artist

                {

                    Id = 26,

                    Name = "Group"

                },

                [27] = new Artist

                {

                    Id = 27,

                    Name = "Other Group"

                }

            };

            unitOfWork.SetupGet(e => e.Musics).Returns(musicRepo.Object);

            unitOfWork.SetupGet(e => e.Artists).Returns(artistRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            musicRepo.Setup(e => e.GetWithArtistByIdAsync(It.IsAny<int>()))

                     .ReturnsAsync((int id) => dbCollectionMusic[id]);

            artistRepo.Setup(e => e.IsExists(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollectionArtists.ContainsKey(id));

            return (unitOfWork, musicRepo, dbCollectionMusic);

        }

        [Test]

        public async Task GetMusicById\_ItemExists\_Success()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollectionMusic) = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            // Act

            var music = await service.GetMusicById(27);

            // Assert

            Assert.AreEqual(music, dbCollectionMusic[27]);

        }

        [Test]

        public void GetMusicById\_ItemDoesNotExists\_KeyNotFoundException()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollectionMusic) = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            // Act + Assert

            Assert.ThrowsAsync<KeyNotFoundException>(async () => await service.GetMusicById(0));

        }

    }

}

Содержание UpdateMusicTests.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class UpdateMusicTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IMusicRepository> musicRepo, Dictionary<int, Music> dbCollectionMusic) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var musicRepo = new Mock<IMusicRepository>(MockBehavior.Strict);

            var artistRepo = new Mock<IArtistRepository>(MockBehavior.Strict);

            var dbCollectionMusic = new Dictionary<int, Music>

            {

                [26] = new Music

                {

                    Id = 26,

                    ArtistId = 26,

                    Name = "Delete Track"

                },

                [27] = new Music

                {

                    Id = 27,

                    ArtistId = 27,

                    Name = "Track"

                }

            };

            var dbCollectionArtists = new Dictionary<int, Artist>

            {

                [26] = new Artist

                {

                    Id = 26,

                    Name = "Group"

                },

                [27] = new Artist

                {

                    Id = 27,

                    Name = "Other Group"

                }

            };

            unitOfWork.SetupGet(e => e.Musics).Returns(musicRepo.Object);

            unitOfWork.SetupGet(e => e.Artists).Returns(artistRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            musicRepo.Setup(e => e.GetWithArtistByIdAsync(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollectionMusic[id]);

            musicRepo.Setup(e => e.IsExists(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollectionMusic.ContainsKey(id));

            artistRepo.Setup(e => e.GetByIdAsync(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollectionArtists[id]);

            artistRepo.Setup(e => e.IsExists(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollectionArtists.ContainsKey(id));

            return (unitOfWork, musicRepo, dbCollectionMusic);

        }

        [Test]

        public async Task UpdateMusic\_FullInfo\_Success()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollectionMusic)  = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            var music = new Music

            {

                ArtistId = 27,

                Name = "New Track"

            };

            // Act

            await service.UpdateMusic(27, music);

            // Assert

            Assert.AreEqual((await unitOfWork.Object.Musics.GetWithArtistByIdAsync(27)).Name, music.Name);

        }

        [Test]

        public void UpdateMusic\_EmptyName\_InvalidDataException()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollectionMusic)  = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            var music = new Music()

            {

                Name = ""

            };

            // Act + Assert

            Assert.ThrowsAsync<InvalidDataException>(async () => await service.UpdateMusic(27, music));

        }

        [Test]

        public void UpdateMusic\_NoItemForUpdate\_NullReferenceException()

        {

            // Arrange

            var (unitOfWork, musicRepo, dbCollectionMusic)  = GetMocks();

            var service = new MusicService(unitOfWork.Object);

            var music = new Music()

            {

                Name = "Update Track"

            };

            // Act + Assert

            Assert.ThrowsAsync<NullReferenceException>(async () => await service.UpdateMusic(0, music));

        }

    }

}

Для ArtistService.cs были описаны тесты, представленные далее.

Содержание CreateArtistTests.cs:

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class CreateArtistTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IArtistRepository> artistRepo, Dictionary<int, Artist> dbCollection) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var artistRepo = new Mock<IArtistRepository>(MockBehavior.Strict);

            var dbCollection = new Dictionary<int, Artist>

            {

                [26] = new Artist

                {

                    Id = 26,

                    Name = "Delete Group"

                },

                [27] = new Artist

                {

                    Id = 27,

                    Name = "Group"

                }

            };

            unitOfWork.SetupGet(e => e.Artists).Returns(artistRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            artistRepo.Setup(e => e.AddAsync(It.IsAny<Artist>()))

                      .Callback((Artist newArtist) => { dbCollection.Add(newArtist.Id, newArtist); })

                      .Returns((Artist \_) => Task.CompletedTask);

            return (unitOfWork, artistRepo, dbCollection);

        }

        [Test]

        public async Task CreateArtist\_FullInfo\_Success()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection) = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            var artist = new Artist

            {

                Id = 28,

                Name = "New Group"

            };

            // Act

            await service.CreateArtist(artist);

            // Assert

            Assert.IsTrue(dbCollection.ContainsKey(artist.Id));

        }

        [Test]

        public void CreateArtist\_NullObject\_NullReferenceException()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection) = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            // Act + Assert

            Assert.ThrowsAsync<NullReferenceException>(async () => await service.CreateArtist(null));

        }

    }

}

Содержание DeleteArtistTests.cs:

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class DeleteArtistTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IArtistRepository> artistRepo, Dictionary<int, Artist> dbCollection) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var artistRepo = new Mock<IArtistRepository>(MockBehavior.Strict);

            var dbCollection = new Dictionary<int, Artist>

            {

                [26] = new Artist

                {

                    Id = 26,

                    Name = "Delete Group"

                },

                [27] = new Artist

                {

                    Id = 27,

                    Name = "Group"

                }

            };

            unitOfWork.SetupGet(e => e.Artists).Returns(artistRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            artistRepo.Setup(e => e.IsExists(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollection.ContainsKey(id));

            artistRepo.Setup(e => e.Remove(It.IsAny<Artist>()))

                      .Callback((Artist newArtist) => { dbCollection.Remove(newArtist.Id); });

            return (unitOfWork, artistRepo, dbCollection);

        }

        [Test]

        public async Task DeleteArtist\_TargetItem\_Success()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection) = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            var artist = new Artist

            {

                Id = 26,

                Name = "Delete Group"

            };

            // Act

            await service.DeleteArtist(artist);

            // Assert

            Assert.IsFalse(dbCollection.ContainsKey(26));

        }

        [Test]

        public void DeleteArtist\_ItemDoesNotExists\_NullReferenceException()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection) = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            var artist = new Artist

            {

                Id = 0,

                Name = "Delete Group"

            };

            // Act + Assert

            Assert.ThrowsAsync<NullReferenceException>(async () => await service.DeleteArtist(artist));

        }

    }

}

Содержание GetArtistByIdTests.cs:

using System.Collections.Generic;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class GetArtistByIdTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IArtistRepository> artistRepo, Dictionary<int, Artist> dbCollection) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var artistRepo = new Mock<IArtistRepository>(MockBehavior.Strict);

            var dbCollection = new Dictionary<int, Artist>

            {

                [26] = new Artist

                {

                    Id = 26,

                    Name = "Delete Group"

                },

                [27] = new Artist

                {

                    Id = 27,

                    Name = "Group"

                }

            };

            unitOfWork.SetupGet(e => e.Artists).Returns(artistRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            artistRepo.Setup(e => e.GetByIdAsync(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollection[id]);

            return (unitOfWork, artistRepo, dbCollection);

        }

        [Test]

        public async Task GetArtistById\_ItemExists\_Success()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection) = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            // Act

            var artist = await service.GetArtistById(27);

            // Assert

            Assert.AreEqual(artist, dbCollection[27]);

        }

        [Test]

        public void GetArtistById\_ItemDoesNotExists\_KeyNotFoundException()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection) = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            // Act + Assert

            Assert.ThrowsAsync<KeyNotFoundException>(async () => await service.GetArtistById(0));

        }

    }

}

Содержание UpdateArtistTests.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading.Tasks;

using Moq;

using MusicApp.Core;

using MusicApp.Core.Models;

using MusicApp.Core.Repositories;

using NUnit.Framework;

namespace MusicApp.BLL.Tests

{

    [TestFixture]

    public class UpdateArtistTests

    {

        private static (Mock<IUnitOfWork> unitOfWork, Mock<IArtistRepository> artistRepo, Dictionary<int, Artist> dbCollection) GetMocks()

        {

            var unitOfWork = new Mock<IUnitOfWork>(MockBehavior.Strict);

            var artistRepo = new Mock<IArtistRepository>(MockBehavior.Strict);

            var dbCollection = new Dictionary<int, Artist>

            {

                [26] = new Artist

                {

                    Id = 26,

                    Name = "Delete Group"

                },

                [27] = new Artist

                {

                    Id = 27,

                    Name = "Group"

                }

            };

            unitOfWork.SetupGet(e => e.Artists).Returns(artistRepo.Object);

            unitOfWork.Setup(e => e.CommitAsync()).ReturnsAsync(0);

            artistRepo.Setup(e => e.GetByIdAsync(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollection[id]);

            artistRepo.Setup(e => e.IsExists(It.IsAny<int>()))

                      .ReturnsAsync((int id) => dbCollection.ContainsKey(id));

            return (unitOfWork, artistRepo, dbCollection);

        }

        [Test]

        public async Task UpdateArtist\_FullInfo\_Success()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection)  = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            var artist = new Artist

            {

                Name = "New Group"

            };

            // Act

            await service.UpdateArtist(27, artist);

            // Assert

            Assert.AreEqual((await unitOfWork.Object.Artists.GetByIdAsync(27)).Name, artist.Name);

        }

        [Test]

        public void UpdateArtist\_EmptyName\_InvalidDataException()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection)  = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            var artist = new Artist()

            {

                Name = ""

            };

            // Act + Assert

            Assert.ThrowsAsync<InvalidDataException>(async () => await service.UpdateArtist(27, artist));

        }

        [Test]

        public void UpdateArtist\_NoItemForUpdate\_NullReferenceException()

        {

            // Arrange

            var (unitOfWork, artistRepo, dbCollection)  = GetMocks();

            var service = new ArtistService(unitOfWork.Object);

            var artist = new Artist()

            {

                Name = "Update Group"

            };

            // Act + Assert

            Assert.ThrowsAsync<NullReferenceException>(async () => await service.UpdateArtist(0, artist));

        }

    }

}

Убедимся, что все описанные unit-тесты проходят успешно:

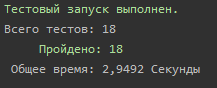


Рисунок 1. Результаты unit-тестирования

# ВЫВОДЫ

В результате выполнения данной лабораторной работы была сформулирована тематика проекта, а также реализованы Core-слой и слой бизнес-логики разрабатываемого приложения. Помимо этого, BLL покрыт модульными тестами, которые позволяют контролировать корректное функционирование приложения. Все описанные тесты были успешно пройдены.