Министерство транспорта Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«РУТ (МИИТ)»

Институт транспортной техники и систем управления
Кафедра «Управление и защита информации»

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Основы построения защищенных баз данных»
на тему «Агентство недвижимости»

Выполнили: ст. гр. ТКИ-542
Дроздов А.Д.
Пономарев А.Д.

Проверил: доц., к.т.н.
Васильева М. А.

Москва 2024

Оглавление

ЦЕЛЬ КУРСОВОГО ПРОЕКТА

Изучить современные технологии ORM, разработать приложение с базой данных, умеющее отрабатывать операции CRUD.

Стек технологий: язык программирования C#, ORM – EF Core, NUnit, PostgreSQL, stylecop.

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

1. Выбрать предметную область.

В соответствии с заданием (выбрать предметную область самостоятельно) описать предметную область, выделить основные сущности, формализовать задачу будущего приложения.

2. Разработать приложение с учетом выбранной технологии ORM. Покрыть все сущностные классы и классы отображения тестами.

3. Разработать репозиторий для работы с БД.

Описание предметной области.

База данных разработана с целью создания заявок по объектам недвижимости. Конечный вариант базы данных содержит в себе информацию о доступных объектах недвижимости, их характеристиках, условиях продажи или аренды, а также данные о потенциальных покупателях, арендаторах и риэлторов.

UML-ДИАГРАММА

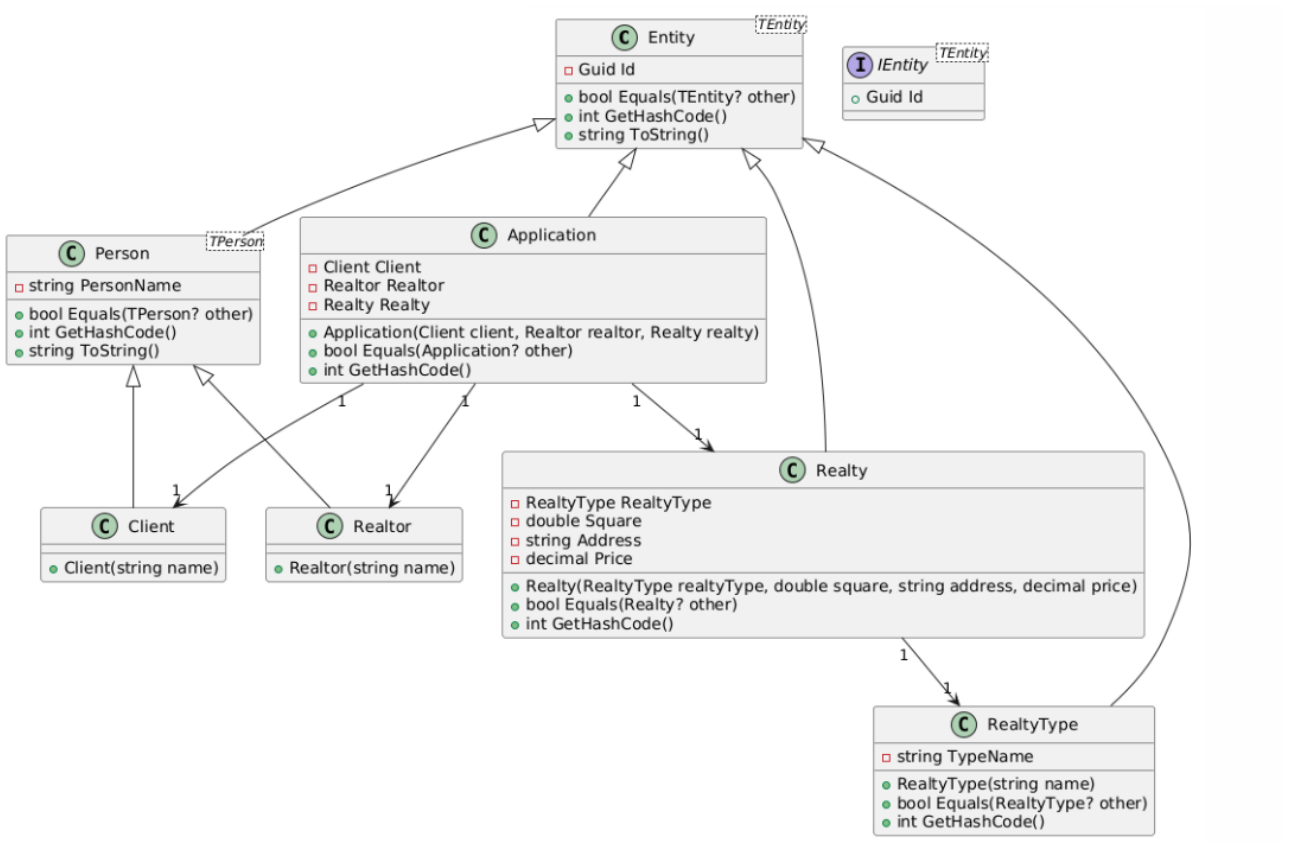


Рисунок 1 – UML – диаграмма

КОД ПРОГРАММЫ

RealtyType.cs

```
// <copyright file="RealtyType.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Domain
{
    using System;
    using Staff;

    /// <summary>
    /// Класс, представляющий тип недвижимости.
    /// </summary>
    public sealed class RealtyType : Entity<RealtyType>, IEquatable<RealtyType>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="RealtyType"/> class.
        /// </summary>
        /// <param name="name">Имя типа недвижимости.</param>
        public RealtyType(string name)
        {
            this.Id = Guid.Empty;
            this.TypeName = name.TrimOrNull() ?? throw new ArgumentNullException(nameof(name));
        }

        [Obsolete("For ORM only")]
        private RealtyType()
        {
        }

        /// <summary>
        /// Получает идентификатор типа недвижимости.
        /// </summary>
        public Guid Id { get; }

        /// <summary>
        /// Получает имя типа недвижимости.
        /// </summary>
        public string TypeName { get; }

        /// <inheritdoc/>
        public bool Equals(RealtyType? other)
        {
            if (other is null)
            {
                return false;
            }

            if (ReferenceEquals(this, other))
            {
                return true;
            }

            return this.TypeName == other.TypeName;
        }
        /// <inheritdoc/>
        public override bool Equals(object? obj)
        {
            return this.Equals(obj as RealtyType);
        }

        /// <inheritdoc/>
        public override int GetHashCode() => HashCode.Combine(this.TypeName);
    }
}
```

Realty.cs

```
// <copyright file="Realty.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace Domain
{
    using System;
    using Staff;

    /// <summary>
    /// Класс, представляющий недвижимость.
    /// </summary>
    public sealed class Realty : Entity<Realty>, IEquatable<Realty>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="Realty"/> class.
        /// </summary>
        /// <param name="realtyType">Тип недвижимости.</param>
        /// <param name="square">Площадь.</param>
        /// <param name="address">Адрес.</param>
        /// <param name="price">Цена.</param>
        public Realty(RealtyType realtyType, double square, string address, decimal price)
        {
            this.Id = Guid.Empty;

            if (realtyType is null)
            {
                throw new ArgumentNullException(nameof(realtyType), "RealtyType не может быть null.");
            }

            ArgumentOutOfRangeException.ThrowIfNegativeOrZero(square);
            ArgumentOutOfRangeException.ThrowIfNegativeOrZero(price);

            this.RealtyType = realtyType;
            this.Square = square;
            this.Address = address.TrimOrNull() ?? throw new ArgumentNullException(nameof(address));
            this.Price = price;
        }

        [Obsolete("For ORM only")]
        private Realty()
        {
        }

        /// <summary>
        /// Получает идентификатор недвижимости.
        /// </summary>
        public Guid Id { get; }

        /// <summary>
        /// Получает тип недвижимости.
        /// </summary>
        public RealtyType RealtyType { get; }

        /// <summary>
        /// Получает площадь недвижимости.
        /// </summary>
        public double Square { get; }

        /// <summary>
        /// Получает адрес недвижимости.
        /// </summary>
        public string Address { get; }

        /// <summary>
        /// Получает или задает цену недвижимости.
        /// </summary>
        public decimal Price { get; set; }

        /// <inheritdoc/>
        public bool Equals(Realty? other)
        {
            if (other is null)
            {
                return false;
            }

            if (ReferenceEquals(this, other))
            {
                return true;
            }

            return this.Id == other.Id &&
                   this.RealtyType.Equals(other.RealtyType) &&
                   this.Square == other.Square &&
                   this.Address == other.Address &&
                   this.Price == other.Price;
        }

        /// <inheritdoc/>
        public override bool Equals(object? obj)
        {
            return this.Equals(obj as Realty);
        }

        /// <inheritdoc/>
        public override int GetHashCode() => HashCode.Combine(this.Id, this.RealtyType, this.Square, this.Address, this.Price);
    }
}
```

Client.cs

```
// <copyright file="Client.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Domain
```

```
{
    using System;
    using System.Xml.Linq;

    /// <summary>
    /// Класс, представляющий клиента.
    /// </summary>
    public class Client : Person<Client>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="Client"/> class.
        /// </summary>
        /// <param name="name">Имя клиента.</param>
         public Client(string name)
           : base(name)
        {
        }

        /// <summary>
        /// Инициализирует новый экземпляр класса <see cref="Client"/>.
        /// </summary>
        [Obsolete("For ORM only", true)]
        private Client()
            : base(string.Empty)
        {
        }

    }
}
```

Realtor.cs

```
// <copyright file="Realtor.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Domain
{
    using System;

    /// <summary>
    /// Класс, представляющий риэлтора.
    /// </summary>
    public class Realtor : Person<Realtor>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="Realtor"/> class.
        /// </summary>
        /// <param name="name">Имя риэлтора.</param>
        public Realtor(string name)
            : base(name)
        {
        }
        /// <summary>
        /// Инициализирует новый экземпляр класса <see cref="Client"/>.
        /// </summary>
        [Obsolete("For ORM only", true)]
        private Realtor()
            : base(string.Empty)
        {
        }
    }
}
```

Application.cs

```
// <copyright file="Application.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace Domain
{
    using System;

    /// <summary>
    /// Класс, представляющий заявку на сделку с недвижимостью.
    /// </summary>
    public sealed class Application : Entity<Application>, IEquatable<Application>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="Application"/> class.
        /// </summary>
        /// <param name="client">Клиент.</param>
        /// <param name="realtor">Риэлтор.</param>
        /// <param name="realty">Недвижимость.</param>
        public Application(Client client, Realtor realtor, Realty realty)
        {
            this.Id = Guid.Empty;
            this.Client = client ?? throw new ArgumentNullException(nameof(client));
            this.Realtor = realtor ?? throw new ArgumentNullException(nameof(realtor));
            this.Realty = realty ?? throw new ArgumentNullException(nameof(realty));
        }

        [Obsolete("For ORM only")]
        private Application()
        {
        }

        /// <summary>
        /// Получает идентификатор заявки.
        /// </summary>
        public Guid Id { get; }

        /// <summary>
        /// Получает клиента.
        /// </summary>
        public Client Client { get; }

        /// <summary>
        /// Получает риэлтора.
        /// </summary>
        public Realtor Realtor { get; }

        /// <summary>
        /// Получает недвижимость.
        /// </summary>
        public Realty Realty { get; }

        /// <inheritdoc/>
        public bool Equals(Application? other)
        {
            if (other is null)
            {
                return false;
            }

            if (ReferenceEquals(this, other))
            {
                return true;
            }

            return this.Id == other.Id &&
                   this.Client.Equals(other.Client) &&
                   this.Realtor.Equals(other.Realtor) &&
                   this.Realty.Equals(other.Realty);
        }

        /// <inheritdoc/>
        public override bool Equals(object? obj)
        {
            return this.Equals(obj as Application);
        }

        /// <inheritdoc/>
        public override int GetHashCode() => HashCode.Combine(this.Id, this.Client, this.Realtor, this.Realty);
    }
}
```

Entity.cs

```
// <copyright file="Entity.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Domain
{
    using System;

    /// <summary>
    /// Базовая сущность.
    /// </summary>
    /// <typeparam name="TEntity"> Тип конкретной сущности. </typeparam>
    public abstract class Entity<TEntity> : IEntity<TEntity>
        where TEntity : class, IEntity<TEntity>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="Entity{TEntity}"/> class.
        /// Инициализирует новый экземпляр класса <see cref="Entity{TEntity}"/>.
        /// </summary>
        protected Entity() => this.Id = Guid.Empty;

        /// <inheritdoc cref="IEntity{TEntity}.Id"/>
        public virtual Guid Id { get; protected set; }

        /// <inheritdoc cref="object.ToString"/>
        public override string ToString() => $"[{this.Id}]";

        /// <inheritdoc cref="object.Equals(object?)"/>
        public override bool Equals(object? obj)
        {
            return ReferenceEquals(this, obj)
            || (obj is TEntity entity && this.Equals(entity));
        }

        /// <inheritdoc/>
        public virtual bool Equals(TEntity? other)
        {
            return other is not null
                && this.GetType() == other.GetType()
                && this.Id == other.Id;
        }

        /// <inheritdoc/>
        // @NOTE: В случае проблемы заменить на object.GetHashCode().
        public override int GetHashCode() => this.Id.GetHashCode();
    }
}
```

IEntity.cs

```
// <copyright file="IEntity.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Domain
{
    using System;

    /// <summary>
    /// Интерфейс для сущностей, имеющих идентификатор.
    /// </summary>
    /// <typeparam name="TEntity">Тип сущности.</typeparam>
    public interface IEntity<TEntity> : IEquatable<TEntity>
        where TEntity : class, IEntity<TEntity>
    {
        /// <summary>
        /// Получает идентификатор сущности.
        /// </summary>
        Guid Id { get; }
    }
}
```

Person.cs

```
// <copyright file="Person.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Domain
{
    using System;
    using System.Diagnostics.CodeAnalysis;

    /// <summary>
    /// Абстрактный класс, представляющий человека.
    /// </summary>
    public abstract class Person<TPerson> : Entity<TPerson>
       where TPerson : Person<TPerson>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="Person{TPerson}"/> class.
        /// </summary>
        /// <param name="name">Имя человека.</param>
        protected Person(string name)
        {
            this.Id = Guid.Empty;
            this.PersonName = name ?? throw new ArgumentNullException(nameof(name));
        }

        /// <summary>
        /// Получает идентификатор человека.
        /// </summary>
        public Guid Id { get; }

        /// <summary>
        /// Получает имя человека.
        /// </summary>
        public string PersonName { get; }

        /// <inheritdoc/>
        public bool Equals(TPerson? other)
        {
            if (other is null)
            {
                return false;
            }

            if (ReferenceEquals(this, other))
            {
                return true;
            }

            return this.Id == other.Id &&
                   string.Equals(this.PersonName, other.PersonName, StringComparison.OrdinalIgnoreCase);
        }

        /// <inheritdoc/>
        public override bool Equals(object? obj)
        {
            return this.Equals(obj as TPerson);
        }

        /// <inheritdoc/>
        public override int GetHashCode()
        {
            return HashCode.Combine(this.Id, StringComparer.OrdinalIgnoreCase.GetHashCode(this.PersonName));
        }

        /// <inheritdoc/>
        public override string ToString()
        {
            return $"{this.PersonName} (ID: {this.Id})";
        }
    }
}
```

ПРОХОЖДЕНИЕ ТЕСТОВ

PersonTest.cs

```
// <copyright file="PersonTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DomainTests
{
   using System;
   using NUnit.Framework;
   using Domain;

   [TestFixture]
   public class PersonTests
   {
      [Test]
      [TestCase(typeof(Realtor))]
      [TestCase(typeof(Client))]
      public void Constructor_NullName_ThrowsArgumentNullException(Type personType)
      {
         // Arrange
         string name = null;

         // Act & Assert
         Assert.Throws<ArgumentNullException>(() => new Realtor(name));
         Assert.Throws<ArgumentNullException>(() => new Client(name));
      }
   }
}
```

RealtyTest.cs

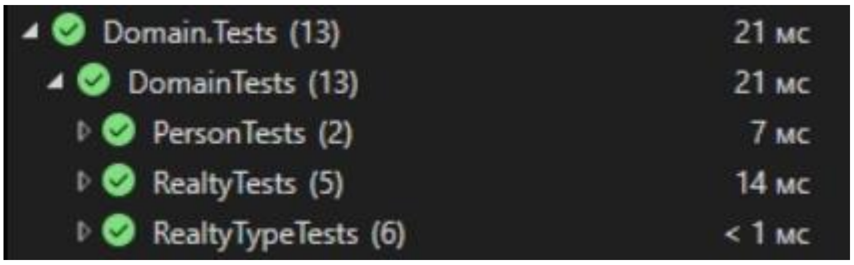
```
// <copyright file="RealtyTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DomainTests
{
   using System;
   using NUnit.Framework;
   using Domain;

   [TestFixture]
   public class RealtyTests
   {
      private RealtyType _realtyType;
      private double _square;
      private string _address;
      private decimal _price;

      [SetUp]
      public void Setup()
      {
         _realtyType = new RealtyType("House");
         _square = 100.0;
```

```
            _address = "123 Main St";
            _price = 500000.0m;
        }

        [Test]
        public void Constructor_NullRealtyType_ThrowsArgumentNullException()
        {
            // Arrange
            RealtyType realtyType = null;

            // Act & Assert
            Assert.Throws<ArgumentNullException>(() => new Realty(realtyType, _square, _address, _price));
        }

        [Test]
        public void Constructor_NegativeSquare_ThrowsArgumentOutOfRangeException()
        {
            // Arrange
            double square = -100.0;

            // Act & Assert
            Assert.Throws<ArgumentOutOfRangeException>(() => new Realty(_realtyType, square, _address, _price));
        }

        [Test]
        public void Constructor_NegativePrice_ThrowsArgumentOutOfRangeException()
        {
            // Arrange
            decimal price = -500000.0m;

            // Act & Assert
            Assert.Throws<ArgumentOutOfRangeException>(() => new Realty(_realtyType, _square, _address, price));
        }

        [Test]
        public void Constructor_NullAddress_ThrowsArgumentNullException()
        {
            // Arrange
            string address = null;

            // Act & Assert
            Assert.Throws<ArgumentNullException>(() => new Realty(_realtyType, _square, address, _price));
        }

        [Test]
        public void Constructor_EmptyAddress_ThrowsArgumentNullException()
        {
            // Arrange
            string address = string.Empty;

            // Act & Assert
            Assert.Throws<ArgumentNullException>(() => new Realty(_realtyType, _square, address, _price));
        }
    }
}
```

RealtyTypeTest.cs

```
// <copyright file="RealtyTypeTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DomainTests
{
   using NUnit.Framework;
   using System;
   using Domain;

   [TestFixture]
   public class RealtyTypeTests
   {
      [Test]
      public void Constructor_NullName_ThrowsArgumentNullException()
      {
         // Arrange
         string name = null;

         // Act & Assert
         Assert.Throws<ArgumentNullException>(() => new RealtyType(name));
      }

      [Test]
      public void Constructor_EmptyName_ThrowsArgumentNullException()
      {
         // Arrange
         string name = string.Empty;

         // Act & Assert
         Assert.Throws<ArgumentNullException>(() => new RealtyType(name));
      }

      [Test]
      public void Equals_SameInstance_ReturnsTrue()
      {
         // Arrange
         var realtyType = new RealtyType("Apartment");

         // Act
         var result = realtyType.Equals(realtyType);

         // Assert
         Assert.IsTrue(result);
      }

      [Test]
      public void Equals_DifferentInstancesWithSameName_ReturnsTrue()
      {
         // Arrange
         var realtyType1 = new RealtyType("Apartment");
         var realtyType2 = new RealtyType("Apartment");

         // Act
         var result = realtyType1.Equals(realtyType2);

         // Assert
```

```
            Assert.IsTrue(result);
        }

        [Test]
        public void Equals_DifferentInstancesWithDifferentName_ReturnsFalse()
        {
            // Arrange
            var realtyType1 = new RealtyType("Apartment");
            var realtyType2 = new RealtyType("House");

            // Act
            var result = realtyType1.Equals(realtyType2);

            // Assert
            Assert.IsFalse(result);
        }

        [Test]
        public void Equals_NullInstance_ReturnsFalse()
        {
            // Arrange
            var realtyType = new RealtyType("Apartment");

            // Act
            var result = realtyType.Equals(null);

            // Assert
            Assert.IsFalse(result);
        }
    }
}
```



```
◢ ✓ Domain.Tests (13)          21 мс
  ◢ ✓ DomainTests (13)         21 мс
    ▷ ✓ PersonTests (2)         7 мс
    ▷ ✓ RealtyTests (5)        14 мс
    ▷ ✓ RealtyTypeTests (6)    < 1 мс
```

Рисунок 2 - Результат прохождения тестов

НАСТРОЙКА МИГРАЦИИ

ApplicationConfiguration.cs

```
// <copyright file="ApplicationConfiguration.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DataAccessLayer.Configurations
{
    using Microsoft.EntityFrameworkCore;
    using Microsoft.EntityFrameworkCore.Metadata.Builders;
    using Domain;

    public class ApplicationConfiguration : IEntityTypeConfiguration<Application>
    {
        public void Configure(EntityTypeBuilder<Application> builder)
        {
            builder.HasKey(a => a.Id);
            builder.HasOne(a => a.Client).WithMany().HasForeignKey("ClientId");
            builder.HasOne(a => a.Realtor).WithMany().HasForeignKey("RealtorId");
            builder.HasOne(a => a.Realty).WithMany().HasForeignKey("RealtyId");
        }
    }
}
```

ClientConfiguration.cs

```
// <copyright file="ClientConfiguration.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DataAccessLayer.Configurations
{
    using Microsoft.EntityFrameworkCore;
    using Microsoft.EntityFrameworkCore.Metadata.Builders;
    using Domain;

    public class ClientConfiguration : IEntityTypeConfiguration<Client>
    {
        public void Configure(EntityTypeBuilder<Client> builder)
        {
            builder.HasKey(c => c.Id);
            builder.Property(c => c.PersonName).IsRequired().HasMaxLength(100);
        }
    }
}
```

RealtorConfiguration.cs

```
// <copyright file="RealtorConfiguration.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DataAccessLayer.Configurations
{
    using Microsoft.EntityFrameworkCore;
    using Microsoft.EntityFrameworkCore.Metadata.Builders;
    using Domain;

    public class RealtorConfiguration : IEntityTypeConfiguration<Realtor>
    {
        public void Configure(EntityTypeBuilder<Realtor> builder)
        {
            builder.HasKey(r => r.Id);
            builder.Property(r => r.PersonName).IsRequired().HasMaxLength(100);
        }
    }
}
```

RealtyConfiguration.cs

```
// <copyright file="RealtyConfiguration.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DataAccessLayer.Configurations
{
    using Microsoft.EntityFrameworkCore;
    using Microsoft.EntityFrameworkCore.Metadata.Builders;
    using Domain;

    public class RealtyConfiguration : IEntityTypeConfiguration<Realty>
    {
        public void Configure(EntityTypeBuilder<Realty> builder)
        {
            builder.HasKey(r => r.Id);
            builder.Property(r => r.Square).IsRequired();
            builder.Property(r => r.Address).IsRequired().HasMaxLength(200);
            builder.Property(r => r.Price).IsRequired();
            builder.HasOne(r => r.RealtyType).WithMany().HasForeignKey("RealtyTypeId");
        }
    }
}
```

RealtyTypeConfiguration.cs

```
// <copyright file="RealtyTypeConfiguration.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace DataAccessLayer.Configurations
{
    using Microsoft.EntityFrameworkCore;
    using Microsoft.EntityFrameworkCore.Metadata.Builders;
    using Domain;

    public class RealtyTypeConfiguration : IEntityTypeConfiguration<RealtyType>
    {
        public void Configure(EntityTypeBuilder<RealtyType> builder)
        {
            builder.HasKey(rt => rt.Id);
            builder.Property(rt => rt.TypeName).IsRequired().HasMaxLength(100);
        }
    }
}
```

ER-ДИАГРАММА

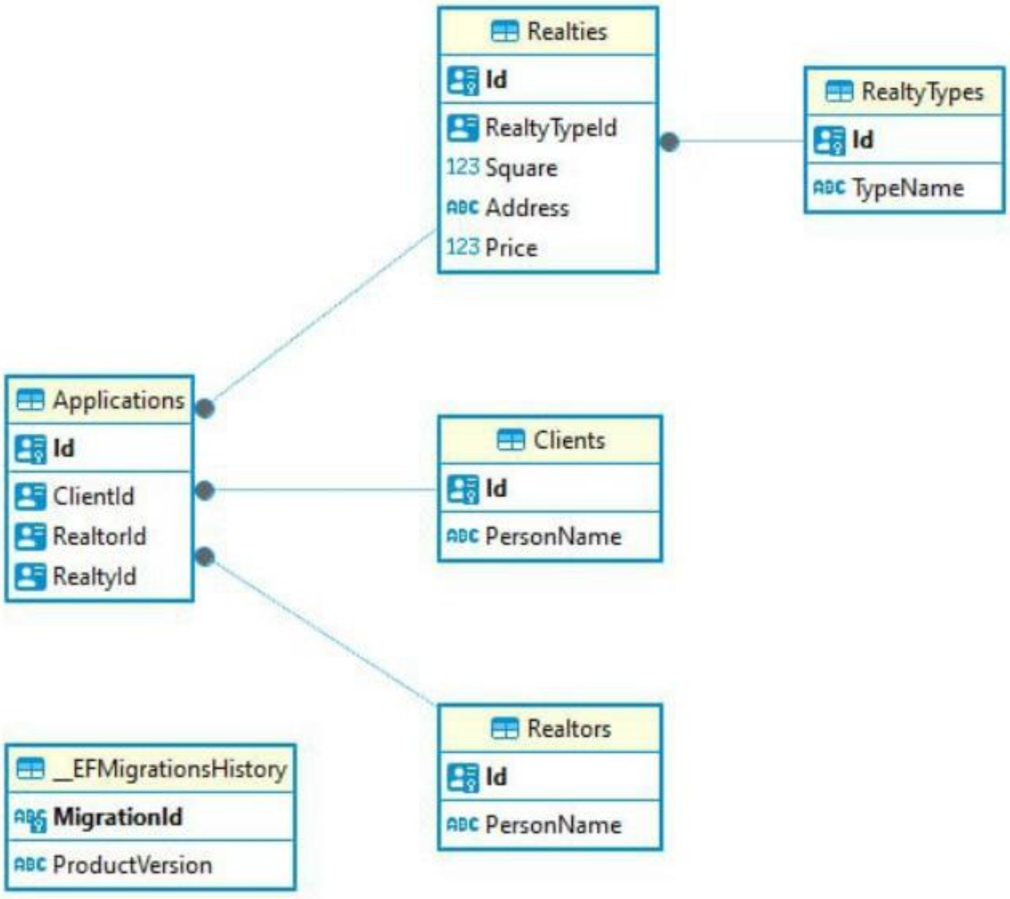


Рисунок 3 – ER-диаграмма

ТЕСТЫ НА МИГРАЦИИ

ApplicationConfigurationTests.cs

```
// <copyright file="ApplicationConfigurationTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace DataAccessLayer.Testss
{
    using DataAccessLayer.Testss;
    using Domain;
    using NUnit.Framework;
    using System;
    using System.Linq;
    using System.Reflection;
    using Microsoft.EntityFrameworkCore;
```

```
/// <summary>
/// Тесты для <see cref="ApplicationConfiguration"/>.
/// </summary>
[TestFixture]
internal sealed class ApplicationConfigurationTests : BaseConfigurationTests
{
    [TearDown]
    public void TearDown()
    {
        this.DataContext.ChangeTracker.Clear();
    }

    [Test]
    public void Application_Configuration_Should_Set_Primary_Key()
    {
        // Arrange
        var application = CreateApplication();

        // Act
        this.DataContext.Applications.Add(application);
        this.DataContext.SaveChanges();
        this.DataContext.ChangeTracker.Clear();

        // Assert
        var result = this.DataContext.Applications.Find(application.Id);
        Assert.That(result, Is.Not.Null);
    }

    private static Application CreateApplication(Client client = null, Realtor realtor = null, Realty realty = null)
    {
        var application = (Application)Activator.CreateInstance(typeof(Application), nonPublic: true);

        var idProperty = typeof(Application).GetProperty("Id", BindingFlags.Public | BindingFlags.Instance);
        if (idProperty != null && idProperty.CanWrite)
        {
            idProperty.SetValue(application, 1);
        }

        var clientProperty = typeof(Application).GetProperty("Client", BindingFlags.Public | BindingFlags.Instance);
        if (clientProperty != null && clientProperty.CanWrite)
        {
            clientProperty.SetValue(application, client);
        }

        var realtorProperty = typeof(Application).GetProperty("Realtor", BindingFlags.Public | BindingFlags.Instance);
        if (realtorProperty != null && realtorProperty.CanWrite)
        {
            realtorProperty.SetValue(application, realtor);
        }

        var realtyProperty = typeof(Application).GetProperty("Realty", BindingFlags.Public | BindingFlags.Instance);
        if (realtyProperty != null && realtyProperty.CanWrite)
        {
            realtyProperty.SetValue(application, realty);
        }

        return application;
    }

    private static Client CreateClient(string name)
    {
```

```
    var client = (Client)Activator.CreateInstance(typeof(Client), nonPublic: true);

    var idProperty = typeof(Client).GetProperty("Id", BindingFlags.Public | BindingFlags.Instance);
    if (idProperty != null && idProperty.CanWrite)
    {
        idProperty.SetValue(client, 1);
    }

    var nameProperty = typeof(Client).GetProperty("Name", BindingFlags.Public | BindingFlags.Instance);
    if (nameProperty != null && nameProperty.CanWrite)
    {
        nameProperty.SetValue(client, name);
    }

    return client;
}

private static Realtor CreateRealtor(string name)
{
    var realtor = (Realtor)Activator.CreateInstance(typeof(Realtor), nonPublic: true);

    var idProperty = typeof(Realtor).GetProperty("Id", BindingFlags.Public | BindingFlags.Instance);
    if (idProperty != null && idProperty.CanWrite)
    {
        idProperty.SetValue(realtor, 1);
    }

    var nameProperty = typeof(Realtor).GetProperty("Name", BindingFlags.Public | BindingFlags.Instance);
    if (nameProperty != null && nameProperty.CanWrite)
    {
        nameProperty.SetValue(realtor, name);
    }

    return realtor;
}

private static Realty CreateRealty(RealtyType realtyType, string address, decimal price, double square)
{
    var realty = (Realty)Activator.CreateInstance(typeof(Realty), nonPublic: true);

    var idProperty = typeof(Realty).GetProperty("Id", BindingFlags.Public | BindingFlags.Instance);
    if (idProperty != null && idProperty.CanWrite)
    {
        idProperty.SetValue(realty, 1);
    }

    var addressProperty = typeof(Realty).GetProperty("Address", BindingFlags.Public | BindingFlags.Instance);
    if (addressProperty != null && addressProperty.CanWrite)
    {
        addressProperty.SetValue(realty, address);
    }

    var priceProperty = typeof(Realty).GetProperty("Price", BindingFlags.Public | BindingFlags.Instance);
    if (priceProperty != null && priceProperty.CanWrite)
    {
        priceProperty.SetValue(realty, price);
    }

    var squareProperty = typeof(Realty).GetProperty("Square", BindingFlags.Public | BindingFlags.Instance);
    if (squareProperty != null && squareProperty.CanWrite)
    {
```

```
                squareProperty.SetValue(realty, square);
            }

            var realtyTypeProperty = typeof(Realty).GetProperty("RealtyType", BindingFlags.Public | BindingFlags.Instance);
            if (realtyTypeProperty != null && realtyTypeProperty.CanWrite)
            {
                realtyTypeProperty.SetValue(realty, realtyType);
            }

            return realty;
        }

        private static RealtyType CreateRealtyType(string typeName)
        {
            var realtyType = (RealtyType)Activator.CreateInstance(typeof(RealtyType), nonPublic: true);

            var idProperty = typeof(RealtyType).GetProperty("Id", BindingFlags.Public | BindingFlags.Instance);
            if (idProperty != null && idProperty.CanWrite)
            {
                idProperty.SetValue(realtyType, 1);
            }

            var typeNameProperty = typeof(RealtyType).GetProperty("TypeName", BindingFlags.Public | BindingFlags.Instance);
            if (typeNameProperty != null && typeNameProperty.CanWrite)
            {
                typeNameProperty.SetValue(realtyType, typeName);
            }

            return realtyType;
        }
    }
}
```

BaseConfigurationTests.cs

```
// <copyright file="BaseConfigurationTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace DataAccessLayer.Testss
{
    using System;
    using DataAccessLayer;
    using Microsoft.EntityFrameworkCore;
    using Microsoft.Extensions.DependencyInjection;
    using Microsoft.Extensions.Logging;

    /// <summary>
    /// Базовый тип для реализации модульных тестов конфигураций (<see cref="IEntityTypeConfiguration{TEntity}"/>).
    /// </summary>
    internal abstract class BaseConfigurationTests
    {
        /// <summary>
        /// Инициализирует новый экземпляр класса <see cref="BaseConfigurationTests"/>.
        /// </summary>
        /// <param name="minimumLogLevel">Минимальный уровень логируемых сообщений.</param>
        /// <exception cref="Exception">В случае невозможности построения/получения контекста доступа к
данным.</exception>
        protected BaseConfigurationTests(LogLevel minimumLogLevel = LogLevel.Debug)
        {
            this.DataContext = new ServiceCollection()
                .AddDbContext<DataContext>(
                    options => options
                        .UseInMemoryDatabase($"InMemoryDB_{Guid.NewGuid()}")
                        .EnableDetailedErrors()
                        .EnableSensitiveDataLogging()
                        .LogTo(Console.WriteLine, minimumLogLevel))
                .BuildServiceProvider()
                .GetService<DataContext>()
                ?? throw new Exception($"Cannot get {typeof(DataContext).FullName} from DI");
        }

        /// <summary>
        /// Контекст доступа к данным.
        /// </summary>
        protected DataContext DataContext { get; }
    }
}
```

ClientConfigurationTests.cs

```
// <copyright file="ClientConfigurationTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace DataAccessLayer.Testss
{
    using DataAccessLayer.Configurations;
    using Domain;
    using NUnit.Framework;

    /// <summary>
    /// Тесты для <see cref="ClientConfiguration"/>.
    /// </summary>
    [TestFixture]
    internal sealed class ClientConfigurationTests : BaseConfigurationTests
    {
        [TearDown]
        public void TearDown()
        {
            this.DataContext.ChangeTracker.Clear();
        }

        [Test]
        public void AddClientToDatabase_Success()
        {
            // arrange
            var client = new Client("Иван Ивановп");

            // act
            _ = this.DataContext.Add(client);
            _ = this.DataContext.SaveChanges();
            this.DataContext.ChangeTracker.Clear();

            // assert
            var result = this.DataContext.Find<Client>(client.Id);

            Assert.That(result, Is.Not.Null);
            Assert.That(result!.PersonName, Is.EqualTo(client.PersonName));
        }
    }
}
```

RealtorConfigurationTests.cs

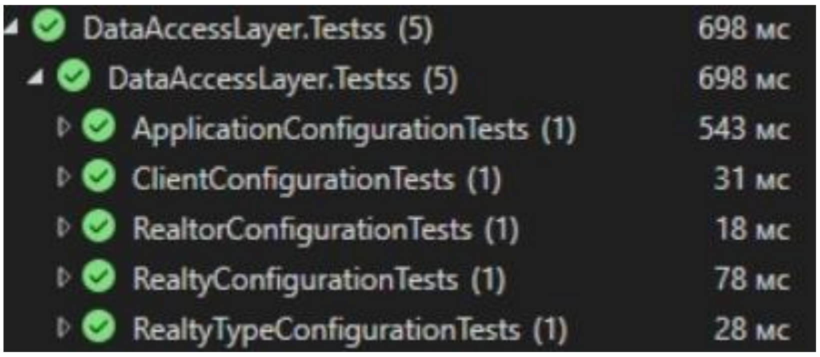
```
// <copyright file="RealtorConfigurationTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
// <copyright file="RealtorConfigurationTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace DataAccessLayer.Testss
{
    using DataAccessLayer.Configurations;
    using DataAccessLayer.Testss;
    using Domain;
    using NUnit.Framework;

    /// <summary>
    /// Тесты для <see cref="RealtorConfiguration"/>.
    /// </summary>
    [TestFixture]
    internal sealed class RealtorConfigurationTests : BaseConfigurationTests
    {
        [TearDown]
        public void TearDown()
        {
            this.DataContext.ChangeTracker.Clear();
        }

        [Test]
        public void AddRealtorToDatabase_Success()
        {
            // arrange
            var realtor = new Realtor("Иван Иванов");

            // act
            _ = this.DataContext.Add(realtor);
            _ = this.DataContext.SaveChanges();
            this.DataContext.ChangeTracker.Clear();

            // assert
            var result = this.DataContext.Find<Realtor>(realtor.Id);

            Assert.That(result, Is.Not.Null);
            Assert.That(result!.PersonName, Is.EqualTo(realtor.PersonName));
        }
    }
}
```

RealtyConfigurationTests.cs

```
// <copyright file="RealtyConfigurationTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace DataAccessLayer.Testss
{
    using DataAccessLayer.Configurations;
    using Domain;
    using NUnit.Framework;
    using System.Linq;
    using Microsoft.EntityFrameworkCore;

    /// <summary>
    /// Тесты для <see cref="RealtyConfiguration"/>.
    /// </summary>
    [TestFixture]
    internal sealed class RealtyConfigurationTests : BaseConfigurationTests
    {
        [TearDown]
        public void TearDown()
        {
            this.DataContext.ChangeTracker.Clear();
        }

        [Test]
        public void AddRealtyToDatabase_Success()
        {
            // arrange
            var realtyType = new RealtyType("House");
            var realty = new Realty(realtyType, 100.0, "123 Main St", 500000.0m);

            // Добавляем RealtyType в контекст данных
            this.DataContext.RealtyTypes.Add(realtyType);
            this.DataContext.SaveChanges();

            // act
            this.DataContext.Realties.Add(realty);
            this.DataContext.SaveChanges();
            this.DataContext.ChangeTracker.Clear();

            // assert
            var result = this.DataContext.Realties
                .Include(r => r.RealtyType)
                .FirstOrDefault(r => r.Id == realty.Id);

            Assert.That(result, Is.Not.Null);
            Assert.That(result!.RealtyType, Is.EqualTo(realty.RealtyType));
            Assert.That(result!.Square, Is.EqualTo(realty.Square));
            Assert.That(result!.Address, Is.EqualTo(realty.Address));
            Assert.That(result!.Price, Is.EqualTo(realty.Price));
        }
    }
}
```

RealtyTypeConfigurationTests.cs

```
// <copyright file="RealtyTypeConfigurationTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>
namespace DataAccessLayer.Testss
{
    using DataAccessLayer.Testss;
    using Domain;
    using NUnit.Framework;
    using System;
    using System.Linq;
    using System.Reflection;
    using Microsoft.EntityFrameworkCore;

    /// <summary>
    /// Тесты для <see cref="RealtyTypeConfiguration"/>.
    /// </summary>
    [TestFixture]
    internal sealed class RealtyTypeConfigurationTests : BaseConfigurationTests
    {
        [TearDown]
        public void TearDown()
        {
            this.DataContext.ChangeTracker.Clear();
        }

        [Test]
        public void RealtyType_Configuration_Should_Set_MaxLength_For_TypeName()
        {
            // Arrange
            var longTypeName = new string('a', 101); // 101 characters, which exceeds the max length
            var realtyType = CreateRealtyType(longTypeName);

            // Act & Assert
            Assert.Throws<DbUpdateException>(() =>
            {
                this.DataContext.RealtyTypes.Add(realtyType);
                this.DataContext.SaveChanges();
            });
        }

        private static RealtyType CreateRealtyType(string typeName)
        {
            var realtyType = (RealtyType)Activator.CreateInstance(typeof(RealtyType), nonPublic: true);

            var idProperty = typeof(RealtyType).GetProperty("Id", BindingFlags.Public | BindingFlags.Instance);
            if (idProperty != null && idProperty.CanWrite)
            {
                idProperty.SetValue(realtyType, 1);
            }

            var typeNameProperty = typeof(RealtyType).GetProperty("TypeName", BindingFlags.Public | BindingFlags.Instance);
            if (typeNameProperty != null && typeNameProperty.CanWrite)
            {
                typeNameProperty.SetValue(realtyType, typeName);
            }

            return realtyType;
        }
    }
}
```



```
◢ ✓ DataAccessLayer.Testss (5)                    698 мс
  ◢ ✓ DataAccessLayer.Testss (5)                  698 мс
    ▷ ✓ ApplicationConfigurationTests (1)         543 мс
    ▷ ✓ ClientConfigurationTests (1)               31 мс
    ▷ ✓ RealtorConfigurationTests (1)              18 мс
    ▷ ✓ RealtyConfigurationTests (1)               78 мс
    ▷ ✓ RealtyTypeConfigurationTests (1)           28 мс
```

Рисунок 4 – Результат прохождения тестов

РЕПОЗИТОРИИ

ApplicationRepository.cs

```
// <copyright file="ApplicationRepository.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository
{
    using System;
    using System.Linq;
    using DataAccessLayer;
    using Domain;
    using Microsoft.EntityFrameworkCore;

    /// <summary>
    /// Репозиторий для класса <see cref="Domain.Application"/>.
    /// </summary>
    public sealed class ApplicationRepository : BaseRepository<Application>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="ApplicationRepository"/> class.
        /// Инициализирует новый экземпляр класса <see cref="ApplicationRepository"/>.
        /// </summary>
        /// <param name="dataContext">Контекст доступа к данным.</param>
        /// <exception cref="ArgumentNullException">
        /// В случае если <paramref name="dataContext"/> – <see langword="null"/>.
        /// </exception>
        public ApplicationRepository(DataContext dataContext)
            : base(dataContext)
        {
        }

        /// <summary>
        /// Получает все заявки.
        /// </summary>
        /// <returns>Заявки.</returns>
        public override IQueryable<Application> GetAll()
        {
            return this.DataContext.Applications
                .Include(application => application.Client)
                .Include(application => application.Realtor)
                .Include(application => application.Realty);
        }
    }
}
```

Baserepository.cs

```
// <copyright file="BaseRepository.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository
{
    using System;
using System.Linq;
using System.Linq.Expressions;
using DataAccessLayer;
using Domain;

/// <summary>
/// Базовый класс репозиториев.
/// </summary>
/// <typeparam name="TEntity"> Целевой тип сущности. </typeparam>
public abstract class BaseRepository<TEntity>
   where TEntity : class, IEntity<TEntity>
{
   /// <summary>
   /// Initializes a new instance of the <see cref="BaseRepository{TEntity}"/> class.
   /// Инициализирует новый экземпляр класса <see cref="BaseRepository{TEntity}"/>.
   /// </summary>
   /// <param name="dataContext">Контекст доступа к данным.</param>
   protected BaseRepository(DataContext dataContext)
   {
      this.DataContext = dataContext ?? throw new ArgumentNullException(nameof(dataContext));
   }

   /// <summary>
   /// Контекст доступа к данным.
   /// </summary>
   public DataContext DataContext { get; }

   /// <summary>
   /// Создает сущность.
   /// </summary>
   /// <param name="entity">Сущность.</param>
   /// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>
   /// <returns>Контекст доступа к сущности.</returns>
   public TEntity Create(TEntity entity, bool saveNow = true)
   {
      var result = this.DataContext.Set<TEntity>().Add(entity).Entity;
      _ = this.Save(saveNow);
      return result;
   }

   /// <summary>
   /// Удаляет сущность.
   /// </summary>
   /// <param name="entity">Сущность.</param>
   /// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>
   /// <returns>Измененный контекст доступа к сущности.</returns>
   public TEntity Delete(TEntity entity, bool saveNow = true)
   {
      var result = this.DataContext.Set<TEntity>().Remove(entity).Entity;
      _ = this.Save(saveNow);
      return result;
   }

   /// <summary>
   /// Поиск множества сущностей по предикату (<paramref name="predicate"/>).
   /// </summary>
   /// <param name="predicate">Предикат, которому должна удовлетворять сушность.</param>
   /// <returns>Множество (<see cref="IQueryable{TEntity}"/>) всех сущностей.</returns>
   public IQueryable<TEntity> Filter(Expression<Func<TEntity, bool>> predicate) => this.GetAll().Where(predicate);

   /// <summary>
```

```
        /// Поиск сущности по предикату (<paramref name="predicate"/>).
        /// </summary>
        /// <param name="predicate">Предикат, которому должна удовлетворять сущность.</param>
        /// <returns>Сущность или <see langword="null"/>.</returns>
        public TEntity? Find(Expression<Func<TEntity, bool>> predicate) => this.GetAll().FirstOrDefault(predicate);

        /// <summary>
        /// Получение конкретной сущности по её идентификатору.
        /// </summary>
        /// <param name="id">Идентификатор сущности.</param>
        /// <returns>Сушность.</returns>
        public TEntity? Get(Guid id) => this.GetAll().SingleOrDefault(entity => entity.Id == id);

        /// <summary>
        /// Получение всех сущностей.
        /// </summary>
        /// <returns> Множество (<see cref="IQueryable{TEntity}"/>) всех сущностей.</returns>
        public abstract IQueryable<TEntity> GetAll();

        /// <summary>
        /// Изменяет Сущность.
        /// </summary>
        /// <param name="entity">Сушность.</param>
        /// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>
        /// <returns>Измененный контекст доступа к сущности.</returns>
        public TEntity Update(TEntity entity, bool saveNow = true)
        {
            var result = this.DataContext.Set<TEntity>().Update(entity).Entity;
            _ = this.Save(saveNow);
            return result;
        }

        /// <summary>
        /// Сохраняет контекст в БД.
        /// </summary>
        /// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>
        /// <returns>Количество измененных сущностей.</returns>
        private int Save(bool saveNow = true)
        {
            return saveNow
                ? this.DataContext.SaveChanges()
                : 0;
        }
    }
}
```

ClientRepository.cs

```
// <copyright file="ClientRepository.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository
{
    using System;
    using System.Linq;
    using DataAccessLayer;
    using Domain;
    using Microsoft.EntityFrameworkCore;

    /// <summary>
    /// Репозиторий для класса <see cref="Domain.Client"/>.
    /// </summary>
    public sealed class ClientRepository : BaseRepository<Client>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="ClientRepository"/> class.
        /// Инициализирует новый экземпляр класса <see cref="ClientRepository"/>.
        /// </summary>
        /// <param name="dataContext">Контекст доступа к данным.</param>
        /// <exception cref="ArgumentNullException">
        /// В случае если <paramref name="dataContext"/> – <see langword="null"/>.
        /// </exception>
        public ClientRepository(DataContext dataContext)
            : base(dataContext)
        {
        }

        /// <summary>
        /// Получает всех клиентов.
        /// </summary>
        /// <returns>Клиенты.</returns>
        public override IQueryable<Client> GetAll()
        {
            return this.DataContext.Clients;
        }
    }
}
```

RealtorRepository.cs

```
// <copyright file="RealtorRepository.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository
{
    using System;
    using System.Linq;
    using DataAccessLayer;
    using Domain;
    using Microsoft.EntityFrameworkCore;

    /// <summary>
    /// Репозиторий для класса <see cref="Domain.Realtor"/>.
    /// </summary>
    public sealed class RealtorRepository : BaseRepository<Realtor>
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="RealtorRepository"/> class.
        /// Инициализирует новый экземпляр класса <see cref="RealtorRepository"/>.
        /// </summary>
        /// <param name="dataContext">Контекст доступа к данным.</param>
        /// <exception cref="ArgumentNullException">
        /// В случае если <paramref name="dataContext"/> – <see langword="null"/>.
        /// </exception>
        public RealtorRepository(DataContext dataContext)
            : base(dataContext)
        {
        }

        /// <summary>
        /// Получает всех риэлторов.
        /// </summary>
        /// <returns>Риэлторы.</returns>
        public override IQueryable<Realtor> GetAll()
        {
            return this.DataContext.Realtors;
        }
    }
}
```

RealtyRepository.cs

```
// <copyright file="RealtyRepository.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository
{
    using System;
    using System.Linq;
    using DataAccessLayer;
    using Domain;
    using Microsoft.EntityFrameworkCore;

    /// <summary>
    /// Репозиторий для класса <see cref="Domain.Realty"/>.
    /// </summary>
    public sealed class RealtyRepository : BaseRepository<Realty>
    {
        /// <summary>
        /// Инициализирует новый экземпляр класса <see cref="RealtyRepository"/>.
        /// </summary>
        /// <param name="dataContext">Контекст доступа к данным.</param>
        /// <exception cref="ArgumentNullException">
        /// В случае если <paramref name="dataContext"/> – <see langword="null"/>.
        /// </exception>
        public RealtyRepository(DataContext dataContext)
            : base(dataContext)
        {
        }

        /// <summary>
        /// Получает все объекты недвижимости.
        /// </summary>
        /// <returns>Объекты недвижимости.</returns>
        public override IQueryable<Realty> GetAll()
        {
            return this.DataContext.Realties
                .Include(realty => realty.RealtyType);
        }
    }
}
```

RealtyTypeRepository.cs

```
// <copyright file="RealtyTypeRepository.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository
{
    using System;
    using System.Linq;
    using DataAccessLayer;
    using Domain;
    using Microsoft.EntityFrameworkCore;

    /// <summary>
    /// Репозиторий для класса <see cref="Domain.RealtyType"/>.
    /// </summary>
    public sealed class RealtyTypeRepository : BaseRepository<RealtyType>
    {
        /// <summary>
        /// Инициализирует новый экземпляр класса <see cref="RealtyTypeRepository"/>.
        /// </summary>
        /// <param name="dataContext">Контекст доступа к данным.</param>
        /// <exception cref="ArgumentNullException">
        /// В случае если <paramref name="dataContext"/> – <see langword="null"/>.
        /// </exception>
        public RealtyTypeRepository(DataContext dataContext)
            : base(dataContext)
        {
        }

        /// <summary>
        /// Получает все типы недвижимости.
        /// </summary>
        /// <returns>Типы недвижимости.</returns>
        public override IQueryable<RealtyType> GetAll()
        {
            return this.DataContext.RealtyTypes;
        }
    }
}
```

ТЕСТЫ НА РЕПОЗИТОРИИ

ApplicationRepositoryTest.cs

```
// <copyright file="ApplicationRepositoryTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository.Tests
{
  using System;
  using System.Linq;
  using Domain;
  using NUnit.Framework;

  /// <summary>
  /// Ìîäóëüíûå òåñòû äëÿ <see cref="ApplicationRepository"/>.
  /// </summary>
  [TestFixture]
  internal sealed class ApplicationRepositoryTests
    : BaseRepositoryTests<ApplicationRepository, Application>
  {
    [SetUp]
    public void SetUp()
    {
      _ = this.DataContext.Database.EnsureCreated();
    }

    [TearDown]
    public void TearDown()
    {
      _ = this.DataContext.Database.EnsureDeleted();
    }

    [Test]
    public void Create_ValidData_Success()
    {
      // arrange
      var client = new Client("John");
      var realtor = new Realtor("Jane");
      var realtyType = new RealtyType("House");
      var realty = new Realty(realtyType, 100.0, "123 Main St", 200000.0m);
      var application = new Application(client, realtor, realty);

      // act
      _ = this.Repository.Create(application);

      // assert
      var result = this.DataContext.Find<Application>(application.Id);
      Assert.That(result, Is.EqualTo(application));
    }

    [Test]
    public void Update_ValidData_Success()
    {
      // arrange
      var client = new Client("John");
      var realtor = new Realtor("Jane");
      var realtyType = new RealtyType("House");
```

```
            var realty = new Realty(realtyType, 100.0, "123 Main St", 200000.0m);
            var application = new Application(client, realtor, realty);

            _ = this.DataContext.Add(application);
            _ = this.DataContext.SaveChanges();

            // act
            realty.Price = 250000.0m; // Èçìåíÿåì öåíó íåäâèæèìîñòè
            _ = this.Repository.Update(application);

            // assert
            var result = this.DataContext.Find<Application>(application.Id)?.Realty.Price;
            Assert.That(result, Is.EqualTo(250000.0m));
        }

        [Test]
        public void Delete_ValidData_Success()
        {
            // arrange
            var client = new Client("John");
            var realtor = new Realtor("Jane");
            var realtyType = new RealtyType("House");
            var realty = new Realty(realtyType, 100.0, "123 Main St", 200000.0m);
            var application = new Application(client, realtor, realty);

            _ = this.DataContext.Add(application);
            _ = this.DataContext.SaveChanges();

            // act
            _ = this.Repository.Delete(application);

            // assert
            var result = this.DataContext.Find<Application>(application.Id);
            Assert.That(result, Is.Null);
        }
    }
}
```

BaseRepositoryTest.cs

```
// <copyright file="BaseRepositoryTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository.Tests
{
    using System;
    using DataAccessLayer;
    using Domain;
    using Microsoft.EntityFrameworkCore;
    using Microsoft.Extensions.DependencyInjection;
    using Microsoft.Extensions.Logging;
    using NUnit.Framework;

    /// <summary>
    /// Áàçîâûé òèï òåñòîâ äëÿ ðåïîçèòîðèåâ.
    /// </summary>
    /// <typeparam name="TRepository"> Öåëåâîé òèï òåñòèðóåìîãî ðåïîçèòîðèÿ. </typeparam>
    /// <typeparam name="TEntity"> Öåëåâîé òèï ñóùíîñòè òåñòèðóåìîãî ðåïîçèòîðèÿ. </typeparam>
    public abstract class BaseRepositoryTests<TRepository, TEntity>
        where TRepository : BaseRepository<TEntity>
        where TEntity : class, IEntity<TEntity>
    {
        private readonly ServiceProvider serviceProvider;

        protected BaseRepositoryTests()
        {
            this.serviceProvider = new ServiceCollection()
                .AddDbContext<DataContext>(
                    builder => builder.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())
                        .EnableDetailedErrors()
                        .EnableSensitiveDataLogging()
                        .LogTo(Console.WriteLine, LogLevel.Error))
                .AddScoped<TRepository>()
                .BuildServiceProvider();
        }

        protected DataContext DataContext
        {
            get => this.serviceProvider.GetService<DataContext>()
                ?? throw new Exception($"Cannot get {typeof(DataContext).Name}");
        }

        protected TRepository Repository
        {
            get => this.serviceProvider.GetService<TRepository>()
                ?? throw new Exception($"Cannot get {typeof(TRepository).Name}");
        }
    }
}
```

ClientRepositoryTest.cs

```
// <copyright file="ClientRepositoryTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository.Tests
{
    using System;
    using System.Linq;
    using Domain;
    using NUnit.Framework;

    /// <summary>
    /// Ìîäóëüíûå òåñòû äëÿ <see cref="ClientRepository"/>.
    /// </summary>
    [TestFixture]
    internal sealed class ClientRepositoryTests
        : BaseRepositoryTests<ClientRepository, Client>
    {
        [SetUp]
        public void SetUp()
        {
            _ = this.DataContext.Database.EnsureCreated();
        }

        [TearDown]
        public void TearDown()
        {
            _ = this.DataContext.Database.EnsureDeleted();
        }

        [Test]
        public void Create_ValidData_Success()
        {
            // arrange
            var client = new Client("John");

            // act
            _ = this.Repository.Create(client);

            // assert
            var result = this.DataContext.Find<Client>(client.Id);
            Assert.That(result, Is.EqualTo(client));
        }

        [Test]
        public void Delete_ValidData_Success()
        {
            // arrange
            var client = new Client("John");
            _ = this.DataContext.Add(client);
            _ = this.DataContext.SaveChanges();

            // act
            _ = this.Repository.Delete(client);

            // assert
            var result = this.DataContext.Find<Client>(client.Id);
            Assert.That(result, Is.Null);
        }
    }
}
```

RealtorRepositoryTest.cs

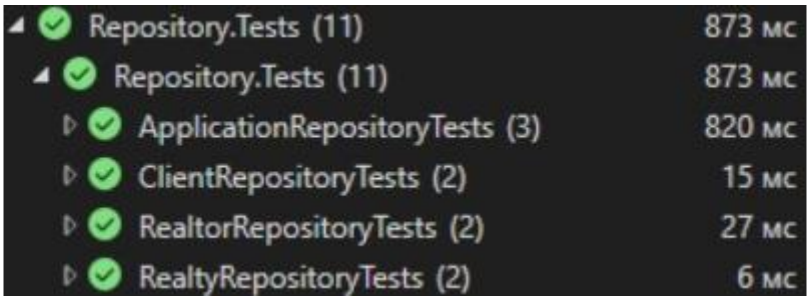
```
// <copyright file="RealtorRepositoryTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository.Tests
{
    using System;
    using System.Linq;
    using Domain;
    using NUnit.Framework;

    /// <summary>
    /// Ìîäóëüíûå òåñòû äëÿ <see cref="RealtorRepository"/>.
    /// </summary>
    [TestFixture]
    internal sealed class RealtorRepositoryTests
        : BaseRepositoryTests<RealtorRepository, Realtor>
    {
        [SetUp]
        public void SetUp()
        {
            _ = this.DataContext.Database.EnsureCreated();
        }

        [TearDown]
        public void TearDown()
        {
            _ = this.DataContext.Database.EnsureDeleted();
        }

        [Test]
        public void Create_ValidData_Success()
        {
            // arrange
            var realtor = new Realtor("Jane");

            // act
            _ = this.Repository.Create(realtor);

            // assert
            var result = this.DataContext.Find<Realtor>(realtor.Id);
            Assert.That(result, Is.EqualTo(realtor));
        }

        [Test]
        public void Delete_ValidData_Success()
        {
            // arrange
            var realtor = new Realtor("Jane");
            _ = this.DataContext.Add(realtor);
            _ = this.DataContext.SaveChanges();

            // act
            _ = this.Repository.Delete(realtor);

            // assert
            var result = this.DataContext.Find<Realtor>(realtor.Id);
            Assert.That(result, Is.Null);
        }
    }
}
```

RealtyRepositoryTest.cs

```
// <copyright file="RealtyRepositoryTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository.Tests
{
    using System;
    using System.Linq;
    using Domain;
    using NUnit.Framework;

    /// <summary>
    /// Ìîäóëüíûå òåñòû äëÿ <see cref="RealtyRepository"/>.
    /// </summary>
    [TestFixture]
    internal sealed class RealtyRepositoryTests
        : BaseRepositoryTests<RealtyRepository, Realty>
    {
        [SetUp]
        public void SetUp()
        {
            _ = this.DataContext.Database.EnsureCreated();
        }

        [TearDown]
        public void TearDown()
        {
            _ = this.DataContext.Database.EnsureDeleted();
        }

        [Test]
        public void Create_ValidData_Success()
        {
            // arrange
            var realtyType = new RealtyType("House");
            var realty = new Realty(realtyType, 100.0, "123 Main St", 200000.0m);

            // act
            _ = this.Repository.Create(realty);

            // assert
            var result = this.DataContext.Find<Realty>(realty.Id);
            Assert.That(result, Is.EqualTo(realty));
        }

        [Test]
        public void Delete_ValidData_Success()
        {
            // arrange
            var realtyType = new RealtyType("House");
            var realty = new Realty(realtyType, 100.0, "123 Main St", 200000.0m);
            _ = this.DataContext.Add(realty);
            _ = this.DataContext.SaveChanges();

            // act
            _ = this.Repository.Delete(realty);

            // assert
            var result = this.DataContext.Find<Realty>(realty.Id);
            Assert.That(result, Is.Null);
        }
    }
}
```

RealtyTypeRepositoryTest.cs

```
// <copyright file="RealtyTypeRepositoryTests.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

namespace Repository.Tests
{
    using System;
    using System.Linq;
    using Domain;
    using NUnit.Framework;

    /// <summary>
    /// Ìîäóëüíûå òåñòû äëÿ <see cref="RealtyTypeRepository"/>.
    /// </summary>
    [TestFixture]
    internal sealed class RealtyTypeRepositoryTests
        : BaseRepositoryTests<RealtyTypeRepository, RealtyType>
    {
        [SetUp]
        public void SetUp()
        {
            _ = this.DataContext.Database.EnsureCreated();
        }

        [TearDown]
        public void TearDown()
        {
            _ = this.DataContext.Database.EnsureDeleted();
        }

        [Test]
        public void Create_ValidData_Success()
        {
            // arrange
            var realtyType = new RealtyType("House");

            // act
            _ = this.Repository.Create(realtyType);

            // assert
            var result = this.DataContext.Find<RealtyType>(realtyType.Id);
            Assert.That(result, Is.EqualTo(realtyType));
        }

        [Test]
        public void Delete_ValidData_Success()
        {
            // arrange
            var realtyType = new RealtyType("House");
            _ = this.DataContext.Add(realtyType);
            _ = this.DataContext.SaveChanges();

            // act
            _ = this.Repository.Delete(realtyType);

            // assert
            var result = this.DataContext.Find<RealtyType>(realtyType.Id);
            Assert.That(result, Is.Null);
        }
    }
}
```



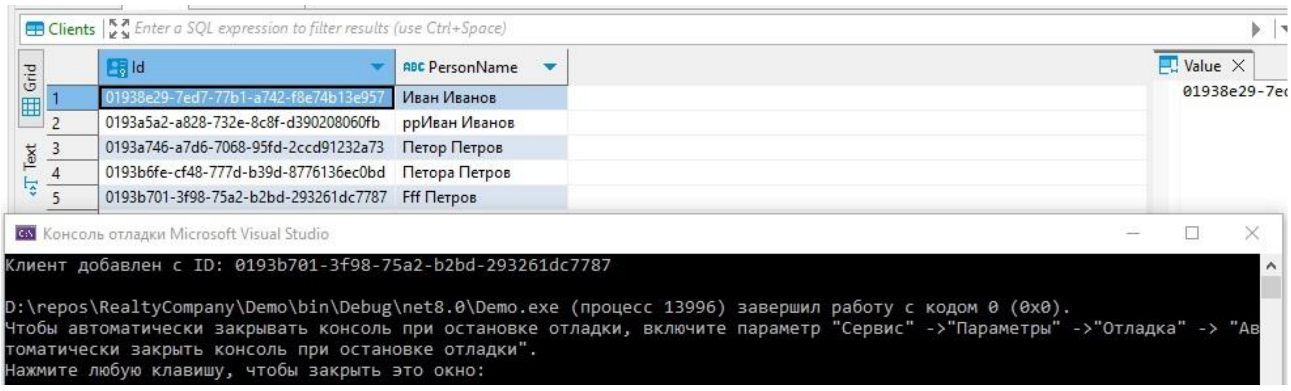
```
◢ ✓ Repository.Tests (11)                 873 мс
  ◢ ✓ Repository.Tests (11)               873 мс
    ▷ ✓ ApplicationRepositoryTests (3)    820 мс
    ▷ ✓ ClientRepositoryTests (2)          15 мс
    ▷ ✓ RealtorRepositoryTests (2)         27 мс
    ▷ ✓ RealtyRepositoryTests (2)           6 мс
```

Рисунок 5 – Результаты выполнения тестов

ТОЧКА ВХОДА В ПРОГРАММУ

Program.cs

```
// <copyright file="Program.cs" company="Realty">
// Copyright (c) Realty. All rights reserved.
// </copyright>

using DataAccessLayer;
using Domain;
using Repository;

namespace Demo
{
    using System;

    class Program
    {
        static void Main(string[] args)
        {
            // Создаём контекст данных
            using (var context = new DataContext())
            {
                // Создаём репозиторий для клиентов
                var clientRepository = new ClientRepository(context);

                // Создаём нового клиента
                var client = new Client("Петора Петров");

                // Добавляем клиента в базу через репозиторий
                clientRepository.Create(client);

                Console.WriteLine($"Клиент добавлен с ID: {client.Id}");
            }
        }
    }
}
```


Clients
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Value
01938e29-7ed
Консоль отладки Microsoft Visual Studio

	Id	PersonName
1	01938e29-7ed7-77b1-a742-f8e74b13e957	Иван Иванов
2	0193a5a2-a828-732e-8c8f-d390208060fb	ррИван Иванов
3	0193a746-a7d6-7068-95fd-2ccd91232a73	Петор Петров
4	0193b6fe-cf48-777d-b39d-8776136ec0bd	Петора Петров
5	0193b701-3f98-75a2-b2bd-293261dc7787	Fff Петров

```
Клиент добавлен с ID: 0193b701-3f98-75a2-b2bd-293261dc7787

D:\repos\RealtyCompany\Demo\bin\Debug\net8.0\Demo.exe (процесс 13996) завершил работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 6 – Результат выполнения запроса

СПИСОК ЛИТЕРАТУРЫ

1. Васильева М.А., Хобта Д.О., Фильтрация набора данных. Рекомендации по выполнению работы и перечень типовых заданий: Учебно-методическое пособие. Издание второе, исправленное и дополненное–М.:РУТ(МИИТ). 2023.–105с.
2. Васильева М.А., Меркулов Д.А. Группировка и обобщение данных. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 46–с.
3. Васильева М.А., Ракинцев Н.А. Соединение данных из множества таблиц. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 63–с.
4. Балакина Е.П., Васильева М.А., Филипченко К.М. Информационное обеспечение систем управления. Методические указания к курсовому проектированию. Учебно-методическое пособие. Издание второе, исправленное и дополненное, 2023.102–с.
5. SQLAlchemy [Электронный ресурс] // SQLAlchemy [сайт]. URL: https://www.sqlalchemy.org/ (дата обращения 24.10.2023).
6. PostgreSQL [Электронный ресурс] // PostgreSQL [сайт]. — URL: https://www.postgresql.org/ (дата обращения 24.10.2023).