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**  
**«Строительная компания»**

Выполнили: ст. гр. ТКИ-441

Земсков Н. А.

Моисеенко М. А.

Проверил: доц., к.т.н.

Васильева М. А.

Москва 2024

**Оглавление**

[Цель курсового проекта …………………………………...3](#_Toc184207231)

[Задание на курсовой проект………………………………3](#_Toc184207232)

[UML-Диаграмма…………………………………………...4](#_Toc184207233)

[Код программы………………………………………….....5](#_Toc184207234)

[Прохождение тестов……………………………………..15](#_Toc184207235)

[Настройка миграции……………………………………..28](#_Toc184207236)

[ER-диаграмма………………………………………….....33](#_Toc184207237)

[Список литературы………………………………………34](#_Toc184207238)

# 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 создаётся для информационного хранения информации о сеансах в кинотеатре. БД должна содержать информацию о фильмах, сеансах, о залах для показа, билетах и зрителях.

# UML-Диаграмма

Изображение выглядит как текст, диаграмма, Параллельный, снимок экрана

Автоматически созданное описание

Рисунок 1 – UML - диаграмма

# Код программы

**Customer.cs**

// <copyright file="Order.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** System.Collections.Generic;

**namespace** **Domain**

{

**///** **<summary>**

**///** Класс, представляющий клиента.

**///** **</summary>**

**public** **class** **Customer**

{

**///** **<summary>**

**///** Уникальный идентификатор клиента.

**///** **</summary>**

**public** Guid Id { **get**; **private** **set**; }

**///** **<summary>**

**///** Имя клиента.

**///** **</summary>**

**public** **string** Name { **get**; **private** **set**; }

**///** **<summary>**

**///** Контактная информация клиента.

**///** **</summary>**

**public** **string**? ContactInfo { **get**; **private** **set**; }

**///** **<summary>**

**///** Список заказов, связанных с клиентом.

**///** **</summary>**

**public** ICollection<Order> Orders { **get**; **private** **set**; } = **new** List<Order>();

**///** **<summary>**

**///** Конструктор для создания клиента с обязательным именем.

**///** **</summary>**

**///** **<param name="name">**Имя клиента.**</param>**

**///** **<param name="contactInfo">**Контактная информация клиента.**</param>**

**public** **Customer**(**string** name, **string**? contactInfo = null)

{

**if** (**string**.IsNullOrWhiteSpace(name))

**throw** **new** ArgumentNullException(**nameof**(name), "Имя клиента не может быть пустым.");

Id = Guid.NewGuid();

Name = name.Trim();

ContactInfo = contactInfo?.Trim();

}

**///** **<summary>**

**///** Устанавливает новую контактную информацию для клиента.

**///** **</summary>**

**///** **<param name="contactInfo">**Новая контактная информация.**</param>**

**public** **void** **UpdateContactInfo**(**string**? contactInfo)

{

ContactInfo = contactInfo?.Trim();

}

**///** **<summary>**

**///** Переопределение метода ToString для удобного отображения клиента.

**///** **</summary>**

**///** **<returns>**Имя и контактная информация клиента.**</returns>**

**public** **override** **string** **ToString**()

{

**return** $"{Name} ({ContactInfo ?? "Контактная информация отсутствует"})";

}

**///** **<summary>**

**///** Переопределение метода Equals для сравнения клиентов.

**///** **</summary>**

**///** **<param name="obj">**Объект для сравнения.**</param>**

**///** **<returns>**True, если клиенты равны.**</returns>**

**public** **override** **bool** **Equals**(**object**? obj)

{

**if** (obj **is** not Customer other)

**return** false;

**return** Id == other.Id &&

Name == other.Name &&

ContactInfo == other.ContactInfo;

}

**///** **<summary>**

**///** Переопределение метода GetHashCode.

**///** **</summary>**

**///** **<returns>**Хэш-код клиента.**</returns>**

**public** **override** **int** **GetHashCode**()

{

**return** HashCode.Combine(Id, Name, ContactInfo);

}

}

}

**Order.cs**

// <copyright file="OrderConfiguration.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** System.Collections.Generic;

**namespace** **Domain**

{

**///** **<summary>**

**///** Класс, представляющий заказ.

**///** **</summary>**

**public** **class** **Order**

{

**///** **<summary>**

**///** Уникальный идентификатор заказа.

**///** **</summary>**

**public** Guid OrderId { **get**; **private** **set**; }

**///** **<summary>**

**///** Идентификатор услуги, связанной с заказом.

**///** **</summary>**

**public** Guid ServiceId { **get**; **private** **set**; }

**///** **<summary>**

**///** Навигационное свойство для услуги.

**///** **</summary>**

**public** Service Service { **get**; **private** **set**; } = null!; // Навигационное свойство

**///** **<summary>**

**///** Идентификатор клиента, сделавшего заказ.

**///** **</summary>**

**public** Guid CustomerId { **get**; **private** **set**; }

**///** **<summary>**

**///** Навигационное свойство для клиента.

**///** **</summary>**

**public** Customer Customer { **get**; **private** **set**; } = null!; // Навигационное свойство

**///** **<summary>**

**///** Идентификатор команды, назначенной на выполнение заказа.

**///** **</summary>**

**public** Guid? TeamId { **get**; **private** **set**; }

**///** **<summary>**

**///** Навигационное свойство для команды.

**///** **</summary>**

**public** Team? Team { **get**; **private** **set**; } // Навигационное свойство (может быть null)

**///** **<summary>**

**///** Дата заказа.

**///** **</summary>**

**public** DateTime OrderDate { **get**; **private** **set**; }

**///** **<summary>**

**///** Указывает, является ли проект кастомным.

**///** **</summary>**

**public** **bool** CustomProject { **get**; **private** **set**; }

**///** **<summary>**

**///** Общая стоимость заказа.

**///** **</summary>**

**public** **decimal** TotalPrice { **get**; **private** **set**; }

**///** **<summary>**

**///** Конструктор для создания нового заказа.

**///** **</summary>**

**///** **<param name="serviceId">**Идентификатор услуги.**</param>**

**///** **<param name="customerId">**Идентификатор клиента.**</param>**

**///** **<param name="orderDate">**Дата заказа.**</param>**

**///** **<param name="totalPrice">**Общая стоимость заказа.**</param>**

**///** **<param name="customProject">**Является ли проект кастомным.**</param>**

**///** **<param name="teamId">**Идентификатор команды (опционально).**</param>**

**public** **Order**(Guid serviceId, Guid customerId, DateTime orderDate, **decimal** totalPrice, **bool** customProject = false, Guid? teamId = null)

{

**if** (totalPrice < 0)

**throw** **new** ArgumentOutOfRangeException(**nameof**(totalPrice), "Стоимость заказа не может быть отрицательной.");

ServiceId = serviceId;

CustomerId = customerId;

OrderDate = orderDate;

TotalPrice = totalPrice;

CustomProject = customProject;

TeamId = teamId;

OrderId = Guid.NewGuid();

}

**///** **<summary>**

**///** Обновляет информацию о команде, назначенной на выполнение заказа.

**///** **</summary>**

**///** **<param name="teamId">**Новый идентификатор команды.**</param>**

**public** **void** **AssignTeam**(Guid? teamId)

{

TeamId = teamId;

}

**///** **<summary>**

**///** Устанавливает новую стоимость заказа.

**///** **</summary>**

**///** **<param name="newPrice">**Новая стоимость.**</param>**

**public** **void** **UpdateTotalPrice**(**decimal** newPrice)

{

**if** (newPrice < 0)

**throw** **new** ArgumentOutOfRangeException(**nameof**(newPrice), "Стоимость заказа не может быть отрицательной.");

TotalPrice = newPrice;

}

**///** **<summary>**

**///** Переопределение метода ToString для удобного отображения заказа.

**///** **</summary>**

**///** **<returns>**Описание заказа в виде строки.**</returns>**

**public** **override** **string** **ToString**()

{

**return** $"Заказ {OrderId} | Услуга: {ServiceId} | Клиент: {CustomerId} | " +

$"Дата: {OrderDate:yyyy-MM-dd} | Кастомный проект: {CustomProject} | " +

$"Стоимость: {TotalPrice:C}";

}

**///** **<summary>**

**///** Переопределение метода Equals для сравнения заказов.

**///** **</summary>**

**///** **<param name="obj">**Объект для сравнения.**</param>**

**///** **<returns>**True, если заказы равны.**</returns>**

**public** **override** **bool** **Equals**(**object**? obj)

{

**if** (obj **is** not Order other)

**return** false;

**return** OrderId == other.OrderId &&

ServiceId == other.ServiceId &&

CustomerId == other.CustomerId &&

TeamId == other.TeamId &&

OrderDate == other.OrderDate &&

CustomProject == other.CustomProject &&

TotalPrice == other.TotalPrice;

}

**///** **<summary>**

**///** Переопределение метода GetHashCode.

**///** **</summary>**

**///** **<returns>**Хэш-код заказа.**</returns>**

**public** **override** **int** **GetHashCode**()

{

**return** HashCode.Combine(OrderId, ServiceId, CustomerId, TeamId, OrderDate, CustomProject, TotalPrice);

}

}

}

**Service.cs**

// <copyright file="Service.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** System.Collections.Generic;

**namespace** **Domain**

{

**///** **<summary>**

**///** Класс, представляющий услугу.

**///** **</summary>**

**public** **class** **Service**

{

**///** **<summary>**

**///** Уникальный идентификатор услуги.

**///** **</summary>**

**public** Guid ServiceId { **get**; **private** **set**; }

**///** **<summary>**

**///** Название услуги.

**///** **</summary>**

**public** **string** ServiceName { **get**; **private** **set**; }

**///** **<summary>**

**///** Цена услуги.

**///** **</summary>**

**public** **decimal** Price { **get**; **private** **set**; }

**///** **<summary>**

**///** Описание услуги.

**///** **</summary>**

**public** **string**? Description { **get**; **private** **set**; }

**///** **<summary>**

**///** Коэффициент для расчета кастомной стоимости.

**///** **</summary>**

**public** **decimal** CustomPriceMultiplier { **get**; **private** **set**; }

**///** **<summary>**

**///** Заказы, связанные с данной услугой.

**///** **</summary>**

**public** ICollection<Order> Orders { **get**; **private** **set**; } = **new** List<Order>();

**///** **<summary>**

**///** Конструктор для создания новой услуги.

**///** **</summary>**

**///** **<param name="serviceName">**Название услуги.**</param>**

**///** **<param name="price">**Цена услуги.**</param>**

**///** **<param name="description">**Описание услуги.**</param>**

**///** **<param name="customPriceMultiplier">**Коэффициент для кастомной стоимости.**</param>**

**public** **Service**(**string** serviceName, **decimal** price, **string**? description = null, **decimal** customPriceMultiplier = 1.0m)

{

**if** (**string**.IsNullOrWhiteSpace(serviceName))

**throw** **new** ArgumentNullException(**nameof**(serviceName), "Название услуги не может быть пустым.");

**if** (price < 0)

**throw** **new** ArgumentOutOfRangeException(**nameof**(price), "Цена услуги не может быть отрицательной.");

**if** (customPriceMultiplier <= 0)

**throw** **new** ArgumentOutOfRangeException(**nameof**(customPriceMultiplier), "Коэффициент кастомной стоимости должен быть больше нуля.");

ServiceId = Guid.NewGuid();

ServiceName = serviceName.Trim();

Price = price;

Description = description?.Trim();

CustomPriceMultiplier = customPriceMultiplier;

}

**///** **<summary>**

**///** Обновляет цену услуги.

**///** **</summary>**

**///** **<param name="newPrice">**Новая цена.**</param>**

**public** **void** **UpdatePrice**(**decimal** newPrice)

{

**if** (newPrice < 0)

**throw** **new** ArgumentOutOfRangeException(**nameof**(newPrice), "Цена услуги не может быть отрицательной.");

Price = newPrice;

}

**///** **<summary>**

**///** Обновляет коэффициент кастомной стоимости.

**///** **</summary>**

**///** **<param name="newMultiplier">**Новый коэффициент.**</param>**

**public** **void** **UpdateCustomPriceMultiplier**(**decimal** newMultiplier)

{

**if** (newMultiplier <= 0)

**throw** **new** ArgumentOutOfRangeException(**nameof**(newMultiplier), "Коэффициент кастомной стоимости должен быть больше нуля.");

CustomPriceMultiplier = newMultiplier;

}

**///** **<summary>**

**///** Рассчитывает кастомную стоимость услуги.

**///** **</summary>**

**///** **<returns>**Кастомная стоимость услуги.**</returns>**

**public** **decimal** **CalculateCustomPrice**()

{

**return** Price \* CustomPriceMultiplier;

}

**///** **<summary>**

**///** Переопределение метода ToString для удобного отображения услуги.

**///** **</summary>**

**///** **<returns>**Описание услуги в виде строки.**</returns>**

**public** **override** **string** **ToString**()

{

**return** $"{ServiceName}: {Price:C} (Кастомная стоимость: {CalculateCustomPrice():C})";

}

**///** **<summary>**

**///** Переопределение метода Equals для сравнения услуг.

**///** **</summary>**

**///** **<param name="obj">**Объект для сравнения.**</param>**

**///** **<returns>**True, если услуги равны.**</returns>**

**public** **override** **bool** **Equals**(**object**? obj)

{

**if** (obj **is** not Service other)

**return** false;

**return** ServiceId == other.ServiceId &&

ServiceName == other.ServiceName &&

Price == other.Price &&

Description == other.Description &&

CustomPriceMultiplier == other.CustomPriceMultiplier;

}

**///** **<summary>**

**///** Переопределение метода GetHashCode.

**///** **</summary>**

**///** **<returns>**Хэш-код услуги.**</returns>**

**public** **override** **int** **GetHashCode**()

{

**return** HashCode.Combine(ServiceId, ServiceName, Price, Description, CustomPriceMultiplier);

}

}

}

**Team.cs**

// <copyright file="Team.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** System.Collections.Generic;

**namespace** **Domain**

{

**///** **<summary>**

**///** Класс, представляющий бригаду.

**///** **</summary>**

**public** **class** **Team**

{

**///** **<summary>**

**///** Уникальный идентификатор бригады.

**///** **</summary>**

**public** Guid Id { **get**; **private** **set**; }

**///** **<summary>**

**///** Название бригады.

**///** **</summary>**

**public** **string** TeamName { **get**; **private** **set**; }

**///** **<summary>**

**///** Указывает, активна ли бригада.

**///** **</summary>**

**public** **bool** IsActive { **get**; **private** **set**; }

**///** **<summary>**

**///** Рабочие, принадлежащие бригаде.

**///** **</summary>**

**public** ICollection<Worker> Workers { **get**; **private** **set**; } = **new** List<Worker>();

**///** **<summary>**

**///** Заказы, назначенные бригаде.

**///** **</summary>**

**public** ICollection<Order> Orders { **get**; **private** **set**; } = **new** List<Order>();

**///** **<summary>**

**///** Конструктор для создания новой бригады.

**///** **</summary>**

**///** **<param name="teamName">**Название бригады.**</param>**

**///** **<param name="isActive">**Статус активности бригады.**</param>**

**public** **Team**(**string** teamName, **bool** isActive = true)

{

**if** (**string**.IsNullOrWhiteSpace(teamName))

**throw** **new** ArgumentNullException(**nameof**(teamName), "Название бригады не может быть пустым.");

Id = Guid.NewGuid();

TeamName = teamName.Trim();

IsActive = isActive;

}

**///** **<summary>**

**///** Устанавливает статус активности бригады.

**///** **</summary>**

**///** **<param name="isActive">**Новый статус активности.**</param>**

**public** **void** **SetActiveStatus**(**bool** isActive)

{

IsActive = isActive;

}

**///** **<summary>**

**///** Переопределение метода ToString для удобного отображения бригады.

**///** **</summary>**

**///** **<returns>**Описание бригады в виде строки.**</returns>**

**public** **override** **string** **ToString**()

{

**return** $"{TeamName} ({(IsActive ? "Активна" : "Не активна")})";

}

**///** **<summary>**

**///** Переопределение метода Equals для сравнения бригад.

**///** **</summary>**

**///** **<param name="obj">**Объект для сравнения.**</param>**

**///** **<returns>**True, если бригады равны.**</returns>**

**public** **override** **bool** **Equals**(**object**? obj)

{

**if** (obj **is** not Team other)

**return** false;

**return** Id == other.Id &&

TeamName == other.TeamName &&

IsActive == other.IsActive;

}

**///** **<summary>**

**///** Переопределение метода GetHashCode.

**///** **</summary>**

**///** **<returns>**Хэш-код бригады.**</returns>**

**public** **override** **int** **GetHashCode**()

{

**return** HashCode.Combine(Id, TeamName, IsActive);

}

}

}

**Worker.cs**

// <copyright file="Worker.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**namespace** **Domain**

{

**///** **<summary>**

**///** Класс, представляющий рабочего.

**///** **</summary>**

**public** **class** **Worker**

{

**///** **<summary>**

**///** Уникальный идентификатор рабочего.

**///** **</summary>**

**public** Guid WorkerId { **get**; **private** **set**; }

**///** **<summary>**

**///** Имя рабочего.

**///** **</summary>**

**public** **string** WorkerName { **get**; **private** **set**; }

**///** **<summary>**

**///** Роль рабочего (например, "Worker" или "Master").

**///** **</summary>**

**public** **string** Role { **get**; **private** **set**; }

**///** **<summary>**

**///** Идентификатор команды, в которой работает рабочий.

**///** **</summary>**

**public** Guid? TeamId { **get**; **private** **set**; }

**///** **<summary>**

**///** Навигационное свойство для команды, к которой принадлежит рабочий.

**///** **</summary>**

**public** Team? Team { **get**; **private** **set**; }

**///** **<summary>**

**///** Конструктор для создания нового рабочего.

**///** **</summary>**

**///** **<param name="workerName">**Имя рабочего.**</param>**

**///** **<param name="role">**Роль рабочего.**</param>**

**///** **<param name="teamId">**Идентификатор команды (опционально).**</param>**

**public** **Worker**(**string** workerName, **string** role, Guid? teamId = null)

{

**if** (**string**.IsNullOrWhiteSpace(workerName))

**throw** **new** ArgumentNullException(**nameof**(workerName), "Имя рабочего не может быть пустым.");

**if** (**string**.IsNullOrWhiteSpace(role))

**throw** **new** ArgumentNullException(**nameof**(role), "Роль рабочего не может быть пустой.");

WorkerId = Guid.NewGuid();

WorkerName = workerName.Trim();

Role = role.Trim();

TeamId = teamId;

}

**///** **<summary>**

**///** Назначает рабочего в команду.

**///** **</summary>**

**///** **<param name="teamId">**Идентификатор команды.**</param>**

**public** **void** **AssignToTeam**(Guid teamId)

{

TeamId = teamId;

}

**///** **<summary>**

**///** Убирает рабочего из команды.

**///** **</summary>**

**public** **void** **RemoveFromTeam**()

{

TeamId = null;

}

**///** **<summary>**

**///** Переопределение метода ToString для удобного отображения рабочего.

**///** **</summary>**

**///** **<returns>**Описание рабочего в виде строки.**</returns>**

**public** **override** **string** **ToString**()

{

**return** $"{WorkerName} - {Role} {(TeamId.HasValue ? $"(Команда: {TeamId})" : "(Без команды)")}";

}

**///** **<summary>**

**///** Переопределение метода Equals для сравнения рабочих.

**///** **</summary>**

**///** **<param name="obj">**Объект для сравнения.**</param>**

**///** **<returns>**True, если рабочие равны.**</returns>**

**public** **override** **bool** **Equals**(**object**? obj)

{

**if** (obj **is** not Worker other)

**return** false;

**return** WorkerId == other.WorkerId &&

WorkerName == other.WorkerName &&

Role == other.Role &&

TeamId == other.TeamId;

}

**///** **<summary>**

**///** Переопределение метода GetHashCode.

**///** **</summary>**

**///** **<returns>**Хэш-код рабочего.**</returns>**

**public** **override** **int** **GetHashCode**()

{

**return** HashCode.Combine(WorkerId, WorkerName, Role, TeamId);

}

}

}

# Прохождение тестов

**CustomerTests.cs**

// <copyright file="CustomerTests.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** Domain;

**using** NUnit.Framework;

**namespace** **DomainTest**

{

**using** System;

**using** System.Collections.Generic;

**using** Domain;

**using** NUnit.Framework;

**///** **<summary>**

**///** Тесты для класса **<see cref="Customer"/>**.

**///** **</summary>**

[TestFixture]

**public** **sealed** **class** **CustomerTests**

{

[Test]

**public** **void** **Ctor\_ValidData\_Success**()

{

// Arrange & Act

**var** customer = **new** Customer("Иван Иванов", "ivanov@example.com");

// Assert

Assert.Multiple(() =>

{

Assert.That(customer.Name, Is.EqualTo("Иван Иванов"));

Assert.That(customer.ContactInfo, Is.EqualTo("ivanov@example.com"));

Assert.That(customer.Id, Is.Not.EqualTo(Guid.Empty));

});

}

[Test]

**public** **void** **Ctor\_NullName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Customer(null!));

}

[Test]

**public** **void** **Ctor\_EmptyName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Customer(""));

}

[TestCase("Иван Иванов", null, "Иван Иванов (Контактная информация отсутствует)")]

[TestCase("Мария Смирнова", "maria@example.com", "Мария Смирнова (maria@example.com)")]

**public** **void** **ToString\_ValidData\_ReturnsExpected**(**string** name, **string**? contact, **string** expected)

{

// Arrange

**var** customer = **new** Customer(name, contact);

// Act

**var** actual = customer.ToString();

// Assert

Assert.That(actual, Is.EqualTo(expected));

}

[Test]

**public** **void** **UpdateContactInfo\_ValidData\_Success**()

{

// Arrange

**var** customer = **new** Customer("Иван Иванов", "ivanov@example.com");

// Act

customer.UpdateContactInfo("new\_email@example.com");

// Assert

Assert.That(customer.ContactInfo, Is.EqualTo("new\_email@example.com"));

}

[Test]

**public** **void** **UpdateContactInfo\_NullData\_SetsNull**()

{

// Arrange

**var** customer = **new** Customer("Иван Иванов", "ivanov@example.com");

// Act

customer.UpdateContactInfo(null);

// Assert

Assert.That(customer.ContactInfo, Is.Null);

}

[Test]

**public** **void** **Equals\_SameData\_ReturnsTrue**()

{

// Arrange

**var** customer1 = **new** Customer("Иван Иванов", "ivanov@example.com");

**var** customer2 = **new** Customer("Иван Иванов", "ivanov@example.com");

// Act & Assert

Assert.That(customer1.Equals(customer2), Is.False, "Guid is unique, objects should not be equal");

}

[Test]

**public** **void** **GetHashCode\_DifferentCustomers\_ReturnsDifferentHashCodes**()

{

// Arrange

**var** customer1 = **new** Customer("Иван Иванов", "ivanov@example.com");

**var** customer2 = **new** Customer("Мария Смирнова", "maria@example.com");

// Act

**var** hash1 = customer1.GetHashCode();

**var** hash2 = customer2.GetHashCode();

// Assert

Assert.That(hash1, Is.Not.EqualTo(hash2));

}

}

}

**OrderTests.cs**

// <copyright file="OrderTests.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** Domain;

**using** NUnit.Framework;

**namespace** **DomainTest**

{

**using** System;

**using** System.Collections.Generic;

**using** Domain;

**using** NUnit.Framework;

**///** **<summary>**

**///** Тесты для класса **<see cref="Order"/>**.

**///** **</summary>**

[TestFixture]

**public** **sealed** **class** **OrderTests**

{

[Test]

**public** **void** **Ctor\_ValidData\_Success**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** teamId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

// Act

**var** order = **new** Order(serviceId, customerId, orderDate, 150000m, true, teamId);

// Assert

Assert.Multiple(() =>

{

Assert.That(order.ServiceId, Is.EqualTo(serviceId));

Assert.That(order.CustomerId, Is.EqualTo(customerId));

Assert.That(order.TeamId, Is.EqualTo(teamId));

Assert.That(order.OrderDate, Is.EqualTo(orderDate));

Assert.That(order.TotalPrice, Is.EqualTo(150000m));

Assert.That(order.CustomProject, Is.True);

Assert.That(order.OrderId, Is.Not.EqualTo(Guid.Empty));

});

}

[Test]

**public** **void** **Ctor\_NegativePrice\_ThrowsException**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

// Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() =>

\_ = **new** Order(serviceId, customerId, orderDate, -100m));

}

[Test]

**public** **void** **AssignTeam\_ValidTeamId\_Success**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

**var** order = **new** Order(serviceId, customerId, orderDate, 150000m);

**var** teamId = Guid.NewGuid();

// Act

order.AssignTeam(teamId);

// Assert

Assert.That(order.TeamId, Is.EqualTo(teamId));

}

[Test]

**public** **void** **AssignTeam\_NullTeam\_RemovesTeam**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

**var** teamId = Guid.NewGuid();

**var** order = **new** Order(serviceId, customerId, orderDate, 150000m, true, teamId);

// Act

order.AssignTeam(null);

// Assert

Assert.That(order.TeamId, Is.Null);

}

[Test]

**public** **void** **UpdateTotalPrice\_ValidPrice\_Success**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

**var** order = **new** Order(serviceId, customerId, orderDate, 150000m);

// Act

order.UpdateTotalPrice(200000m);

// Assert

Assert.That(order.TotalPrice, Is.EqualTo(200000m));

}

[Test]

**public** **void** **UpdateTotalPrice\_NegativePrice\_ThrowsException**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

**var** order = **new** Order(serviceId, customerId, orderDate, 150000m);

// Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() => order.UpdateTotalPrice(-100m));

}

[Test]

**public** **void** **ToString\_ValidData\_ReturnsExpected**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = **new** DateTime(2024, 11, 1);

**var** order = **new** Order(serviceId, customerId, orderDate, 150000m, true);

// Act

**var** result = order.ToString();

// Assert

StringAssert.Contains("150 000,00", result); // Проверка стоимости

StringAssert.Contains("2024-11-01", result); // Проверка даты

}

[Test]

**public** **void** **Equals\_SameData\_ReturnsFalse**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

**var** order1 = **new** Order(serviceId, customerId, orderDate, 150000m);

**var** order2 = **new** Order(serviceId, customerId, orderDate, 150000m);

// Act & Assert

Assert.That(order1.Equals(order2), Is.False, "Orders with the same data but different GUIDs should not be equal.");

}

[Test]

**public** **void** **GetHashCode\_DifferentOrders\_ReturnsDifferentHashCodes**()

{

// Arrange

**var** serviceId = Guid.NewGuid();

**var** customerId = Guid.NewGuid();

**var** orderDate = DateTime.Now;

**var** order1 = **new** Order(serviceId, customerId, orderDate, 150000m);

**var** order2 = **new** Order(serviceId, customerId, orderDate, 150000m);

// Act

**var** hash1 = order1.GetHashCode();

**var** hash2 = order2.GetHashCode();

// Assert

Assert.That(hash1, Is.Not.EqualTo(hash2));

}

}

}

**ServiceTests.cs**

// <copyright file="ServiceTests.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** Domain;

**using** NUnit.Framework;

**namespace** **DomainTest**

{

**using** System;

**using** System.Collections.Generic;

**using** Domain;

**using** NUnit.Framework;

**///** **<summary>**

**///** Тесты для класса **<see cref="Service"/>**.

**///** **</summary>**

[TestFixture]

**public** **sealed** **class** **ServiceTests**

{

[Test]

**public** **void** **Ctor\_ValidData\_Success**()

{

// Arrange & Act

**var** service = **new** Service("Строительство дома", 1000000m, "Описание услуги", 1.2m);

// Assert

Assert.Multiple(() =>

{

Assert.That(service.ServiceName, Is.EqualTo("Строительство дома"));

Assert.That(service.Price, Is.EqualTo(1000000m));

Assert.That(service.Description, Is.EqualTo("Описание услуги"));

Assert.That(service.CustomPriceMultiplier, Is.EqualTo(1.2m));

Assert.That(service.ServiceId, Is.Not.EqualTo(Guid.Empty));

});

}

[Test]

**public** **void** **Ctor\_NullServiceName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Service(null!, 1000000m));

}

[Test]

**public** **void** **Ctor\_EmptyServiceName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Service("", 1000000m));

}

[Test]

**public** **void** **Ctor\_NegativePrice\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() => \_ = **new** Service("Услуга", -500m));

}

[Test]

**public** **void** **Ctor\_NonPositiveMultiplier\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() => \_ = **new** Service("Услуга", 100000m, null, 0m));

}

[Test]

**public** **void** **UpdatePrice\_ValidPrice\_Success**()

{

// Arrange

**var** service = **new** Service("Строительство дома", 1000000m);

// Act

service.UpdatePrice(1200000m);

// Assert

Assert.That(service.Price, Is.EqualTo(1200000m));

}

[Test]

**public** **void** **UpdatePrice\_NegativePrice\_ThrowsException**()

{

// Arrange

**var** service = **new** Service("Строительство дома", 1000000m);

// Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() => service.UpdatePrice(-100m));

}

[Test]

**public** **void** **UpdateCustomPriceMultiplier\_ValidMultiplier\_Success**()

{

// Arrange

**var** service = **new** Service("Строительство дома", 1000000m, null, 1.1m);

// Act

service.UpdateCustomPriceMultiplier(1.3m);

// Assert

Assert.That(service.CustomPriceMultiplier, Is.EqualTo(1.3m));

}

[Test]

**public** **void** **UpdateCustomPriceMultiplier\_NonPositiveMultiplier\_ThrowsException**()

{

// Arrange

**var** service = **new** Service("Строительство дома", 1000000m);

// Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() => service.UpdateCustomPriceMultiplier(0m));

}

[Test]

**public** **void** **CalculateCustomPrice\_ValidData\_ReturnsExpectedPrice**()

{

// Arrange

**var** service = **new** Service("Строительство дома", 1000000m, null, 1.2m);

// Act

**var** customPrice = service.CalculateCustomPrice();

// Assert

Assert.That(customPrice, Is.EqualTo(1200000m));

}

[Test]

**public** **void** **ToString\_ValidData\_ReturnsExpected**()

{

// Arrange

**var** service = **new** Service("Строительство дома", 1000000m, "Описание", 1.2m);

// Act

**var** result = service.ToString();

// Assert

StringAssert.Contains("Строительство дома", result); // Проверка имени услуги

StringAssert.Contains("1 200 000,00", result); // Проверка кастомной стоимости

}

[Test]

**public** **void** **Equals\_SameData\_ReturnsFalse**()

{

// Arrange

**var** service1 = **new** Service("Строительство дома", 1000000m);

**var** service2 = **new** Service("Строительство дома", 1000000m);

// Act & Assert

Assert.That(service1.Equals(service2), Is.False, "Services with the same data but different GUIDs should not be equal.");

}

[Test]

**public** **void** **GetHashCode\_DifferentServices\_ReturnsDifferentHashCodes**()

{

// Arrange

**var** service1 = **new** Service("Строительство дома", 1000000m);

**var** service2 = **new** Service("Ремонт крыши", 500000m);

// Act

**var** hash1 = service1.GetHashCode();

**var** hash2 = service2.GetHashCode();

// Assert

Assert.That(hash1, Is.Not.EqualTo(hash2));

}

}

}

**TeamTests.cs**

// <copyright file="TeamTests.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** Domain;

**using** NUnit.Framework;

**namespace** **DomainTest**

{

**using** System;

**using** System.Collections.Generic;

**using** Domain;

**using** NUnit.Framework;

**///** **<summary>**

**///** Тесты для класса **<see cref="Team"/>**.

**///** **</summary>**

[TestFixture]

**public** **sealed** **class** **TeamTests**

{

[Test]

**public** **void** **Ctor\_ValidData\_Success**()

{

// Arrange & Act

**var** team = **new** Team("Бригада №1", true);

// Assert

Assert.Multiple(() =>

{

Assert.That(team.TeamName, Is.EqualTo("Бригада №1"));

Assert.That(team.IsActive, Is.True);

Assert.That(team.Id, Is.Not.EqualTo(Guid.Empty));

});

}

[Test]

**public** **void** **Ctor\_DefaultIsActive\_Success**()

{

// Arrange & Act

**var** team = **new** Team("Бригада №2");

// Assert

Assert.That(team.IsActive, Is.True);

}

[Test]

**public** **void** **Ctor\_NullTeamName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Team(null!));

}

[Test]

**public** **void** **Ctor\_EmptyTeamName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Team(""));

}

[Test]

**public** **void** **SetActiveStatus\_SetToFalse\_Success**()

{

// Arrange

**var** team = **new** Team("Бригада №1", true);

// Act

team.SetActiveStatus(false);

// Assert

Assert.That(team.IsActive, Is.False);

}

[Test]

**public** **void** **SetActiveStatus\_SetToTrue\_Success**()

{

// Arrange

**var** team = **new** Team("Бригада №2", false);

// Act

team.SetActiveStatus(true);

// Assert

Assert.That(team.IsActive, Is.True);

}

[Test]

**public** **void** **Equals\_SameData\_ReturnsFalse**()

{

// Arrange

**var** team1 = **new** Team("Бригада №1");

**var** team2 = **new** Team("Бригада №1");

// Act & Assert

Assert.That(team1.Equals(team2), Is.False, "Teams with the same name but different GUIDs should not be equal.");

}

[Test]

**public** **void** **GetHashCode\_DifferentTeams\_ReturnsDifferentHashCodes**()

{

// Arrange

**var** team1 = **new** Team("Бригада №1");

**var** team2 = **new** Team("Бригада №2");

// Act

**var** hash1 = team1.GetHashCode();

**var** hash2 = team2.GetHashCode();

// Assert

Assert.That(hash1, Is.Not.EqualTo(hash2));

}

}

}

**WorkerTests.cs**

// <copyright file="WorkerTests.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**using** System;

**using** Domain;

**using** NUnit.Framework;

**namespace** **DomainTest**

{

**using** System;

**using** System.Collections.Generic;

**using** Domain;

**using** NUnit.Framework;

**///** **<summary>**

**///** Тесты для класса **<see cref="Worker"/>**.

**///** **</summary>**

[TestFixture]

**public** **sealed** **class** **WorkerTests**

{

[Test]

**public** **void** **Ctor\_ValidData\_Success**()

{

// Arrange & Act

**var** worker = **new** Worker("Алексей Павлов", "Worker");

// Assert

Assert.Multiple(() =>

{

Assert.That(worker.WorkerName, Is.EqualTo("Алексей Павлов"));

Assert.That(worker.Role, Is.EqualTo("Worker"));

Assert.That(worker.TeamId, Is.Null);

Assert.That(worker.WorkerId, Is.Not.EqualTo(Guid.Empty));

});

}

[Test]

**public** **void** **Ctor\_WithTeamId\_Success**()

{

// Arrange

**var** teamId = Guid.NewGuid();

// Act

**var** worker = **new** Worker("Алексей Павлов", "Worker", teamId);

// Assert

Assert.That(worker.TeamId, Is.EqualTo(teamId));

}

[Test]

**public** **void** **Ctor\_NullWorkerName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Worker(null!, "Worker"));

}

[Test]

**public** **void** **Ctor\_EmptyWorkerName\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Worker("", "Worker"));

}

[Test]

**public** **void** **Ctor\_NullRole\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Worker("Алексей Павлов", null!));

}

[Test]

**public** **void** **Ctor\_EmptyRole\_ThrowsException**()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = **new** Worker("Алексей Павлов", ""));

}

[Test]

**public** **void** **AssignToTeam\_ValidTeamId\_Success**()

{

// Arrange

**var** worker = **new** Worker("Алексей Павлов", "Worker");

**var** teamId = Guid.NewGuid();

// Act

worker.AssignToTeam(teamId);

// Assert

Assert.That(worker.TeamId, Is.EqualTo(teamId));

}

[Test]

**public** **void** **RemoveFromTeam\_RemovesTeamAssignment**()

{

// Arrange

**var** teamId = Guid.NewGuid();

**var** worker = **new** Worker("Алексей Павлов", "Worker", teamId);

// Act

worker.RemoveFromTeam();

// Assert

Assert.That(worker.TeamId, Is.Null);

}

[Test]

**public** **void** **ToString\_WithoutTeam\_ReturnsExpected**()

{

// Arrange

**var** worker = **new** Worker("Алексей Павлов", "Worker");

// Act

**var** result = worker.ToString();

// Assert

Assert.That(result, Is.EqualTo("Алексей Павлов - Worker (Без команды)"));

}

[Test]

**public** **void** **ToString\_WithTeam\_ReturnsExpected**()

{

// Arrange

**var** teamId = Guid.NewGuid();

**var** worker = **new** Worker("Алексей Павлов", "Worker", teamId);

// Act

**var** result = worker.ToString();

// Assert

StringAssert.Contains("Алексей Павлов - Worker", result);

StringAssert.Contains(teamId.ToString(), result);

}

[Test]

**public** **void** **Equals\_SameData\_ReturnsFalse**()

{

// Arrange

**var** worker1 = **new** Worker("Алексей Павлов", "Worker");

**var** worker2 = **new** Worker("Алексей Павлов", "Worker");

// Act & Assert

Assert.That(worker1.Equals(worker2), Is.False, "Workers with the same data but different GUIDs should not be equal.");

}

[Test]

**public** **void** **GetHashCode\_DifferentWorkers\_ReturnsDifferentHashCodes**()

{

// Arrange

**var** worker1 = **new** Worker("Алексей Павлов", "Worker");

**var** worker2 = **new** Worker("Мария Смирнова", "Master");

// Act

**var** hash1 = worker1.GetHashCode();

**var** hash2 = worker2.GetHashCode();

// Assert

Assert.That(hash1, Is.Not.EqualTo(hash2));

}

}

}

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2 - Результат прохождения тестов

# Настройка миграции

**CustomerConfiguration.cs**

// <copyright file="CustomerConfiguration.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**namespace** DataAccess.Configurations

{

**using** Domain;

**using** Microsoft.EntityFrameworkCore;

**using** Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

/// Конфигурация правил отображения сущности (<see cref="Customer"/>) в таблицу БД.

/// </summary>

**public** sealed **class** CustomerConfiguration : IEntityTypeConfiguration<Customer>

{

/// <inheritdoc/>

**public** **void** **Configure**(EntityTypeBuilder<Customer> builder)

{

// Установка первичного ключа

\_ = builder.HasKey(customer => customer.Id);

// Конфигурация свойства Name

\_ = builder.Property(customer => customer.Name)

.HasMaxLength(200)

.IsRequired();

// Конфигурация свойства ContactInfo

\_ = builder.Property(customer => customer.ContactInfo)

.HasMaxLength(300)

.IsRequired(false);

// Опционально: настройка индекса по имени клиента

\_ = builder.HasIndex(customer => customer.Name)

.IsUnique(false); // Уникальность не требуется, может быть несколько клиентов с одинаковым именем

}

}

}

**OrderConfiguration.cs**

// <copyright file="OrderConfiguration.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

namespace DataAccess.Configurations

{

using Domain;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

/// Конфигурация правил отображения сущности (<see cref="Order"/>) в таблицу БД.

/// </summary>

**public** sealed class OrderConfiguration : IEntityTypeConfiguration<**Order**>

{

/// <inheritdoc/>

**public** void Configure(EntityTypeBuilder<**Order**> builder)

{

// Установка первичного ключа

\_ = builder.HasKey(**order** => **order**.OrderId);

// Конфигурация свойства OrderDate

\_ = builder.Property(**order** => **order**.OrderDate)

.IsRequired();

// Конфигурация свойства TotalPrice

\_ = builder.Property(**order** => **order**.TotalPrice)

.HasColumnType("decimal(18,2)") // Для финансовых значений

.IsRequired();

// Конфигурация свойства CustomProject

\_ = builder.Property(**order** => **order**.CustomProject)

.IsRequired();

// Настройка связи с Service

\_ = builder.HasOne(**order** => **order**.Service)

.WithMany()

.HasForeignKey(**order** => **order**.ServiceId)

.OnDelete(DeleteBehavior.Cascade); // Удаление заказа при удалении услуги

// Настройка связи с Customer

\_ = builder.HasOne(**order** => **order**.Customer)

.WithMany(customer => customer.Orders)

.HasForeignKey(**order** => **order**.CustomerId)

.OnDelete(DeleteBehavior.Cascade); // Удаление заказа при удалении клиента

// Настройка связи с Team (может быть null)

\_ = builder.HasOne(**order** => **order**.Team)

.WithMany(team => team.Orders)

.HasForeignKey(**order** => **order**.TeamId)

.OnDelete(DeleteBehavior.SetNull); // Установка null, если команда удалена

}

}

}

**ServiceConfiguration.cs**

// <copyright file="ServiceConfiguration.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

namespace DataAccess.Configurations

{

using Domain;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <**summary**>

/// Конфигурация правил отображения сущности (<**see** cref="Service"/>) в таблицу БД.

/// </**summary**>

public sealed **class** **ServiceConfiguration** : **IEntityTypeConfiguration**<**Service**>

{

/// <**inheritdoc**/>

public **void** Configure(EntityTypeBuilder<Service> builder)

{

// Установка первичного ключа

\_ = builder.HasKey(service => service.ServiceId);

// Конфигурация свойства ServiceName

\_ = builder.Property(service => service.ServiceName)

.HasMaxLength(200)

.IsRequired();

// Конфигурация свойства Price

\_ = builder.Property(service => service.Price)

.HasColumnType("decimal(18,2)") // Для финансовых значений

.IsRequired();

// Конфигурация свойства Description

\_ = builder.Property(service => service.Description)

.HasMaxLength(500)

.IsRequired(**false**);

// Конфигурация свойства CustomPriceMultiplier

\_ = builder.Property(service => service.CustomPriceMultiplier)

.HasColumnType("decimal(5,2)") // Тип для коэффициента

.IsRequired()

.HasDefaultValue(1.0m);

// Опционально: Добавление индекса на название услуги

\_ = builder.HasIndex(service => service.ServiceName)

.IsUnique();

}

}

}

**TeamConfiguration.cs**

// <copyright file="TeamConfiguration.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

**namespace** DataAccess.Configurations

{

**using** Domain;

**using** Microsoft.EntityFrameworkCore;

**using** Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

/// Конфигурация правил отображения сущности (<see cref="Team"/>) в таблицу БД.

/// </summary>

**public** sealed **class** TeamConfiguration : IEntityTypeConfiguration<Team>

{

/// <inheritdoc/>

**public** **void** **Configure**(EntityTypeBuilder<Team> builder)

{

// Установка первичного ключа

\_ = builder.HasKey(team => team.Id);

// Конфигурация свойства TeamName

\_ = builder.Property(team => team.TeamName)

.HasMaxLength(200)

.IsRequired();

// Конфигурация свойства IsActive

\_ = builder.Property(team => team.IsActive)

.IsRequired()

.HasDefaultValue(true); // Значение по умолчанию — активна

// Опционально: Добавление индекса на имя команды

\_ = builder.HasIndex(team => team.TeamName)

.IsUnique(); // Уникальные имена команд

}

}

}

**WorkerConfiguration.cs**

// <copyright file="WorkerConfiguration.cs" company="Земсков Н.А и Моисеенко М.А">

// Copyright (c) Земсков Н.А и Моисеенко М.А. All rights reserved.

// </copyright>

namespace DataAccess.Configurations

{

using Domain;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <**summary**>

/// Конфигурация правил отображения сущности (<**see** cref="Worker"/>) в таблицу БД.

/// </**summary**>

public sealed **class** **WorkerConfiguration** : **IEntityTypeConfiguration**<**Worker**>

{

/// <**inheritdoc**/>

public **void** Configure(EntityTypeBuilder<Worker> builder)

{

// Установка первичного ключа

\_ = builder.HasKey(worker => worker.WorkerId);

// Конфигурация свойства WorkerName

\_ = builder.Property(worker => worker.WorkerName)

.HasMaxLength(200)

.IsRequired();

// Конфигурация свойства Role

\_ = builder.Property(worker => worker.Role)

.HasMaxLength(100)

.IsRequired();

// Связь с командой (Team)

\_ = builder.HasOne(worker => worker.Team)

.WithMany(team => team.Workers)

.HasForeignKey(worker => worker.TeamId)

.OnDelete(DeleteBehavior.SetNull); // При удалении команды TeamId станет NULL

// Опционально: Индекс на WorkerName и Role

\_ = builder.HasIndex(worker => **new** { worker.WorkerName, worker.Role })

.IsUnique(); // Уникальная пара "Имя + Роль"

}

}

}

# ER-диаграмма

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 3 – ER-диаграмма

# 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).