-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**  
**«Автомастерская»**

Выполнил: ст. гр. ТКИ-441

Чевычалов А. В.

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……………………………………..12](#_Toc184207235)

[Настройка миграции……………………………………..21](#_Toc184207236)

[ER-диаграмма………………………………………….....24](#_Toc184207237)

[Список литературы………………………………………25](#_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

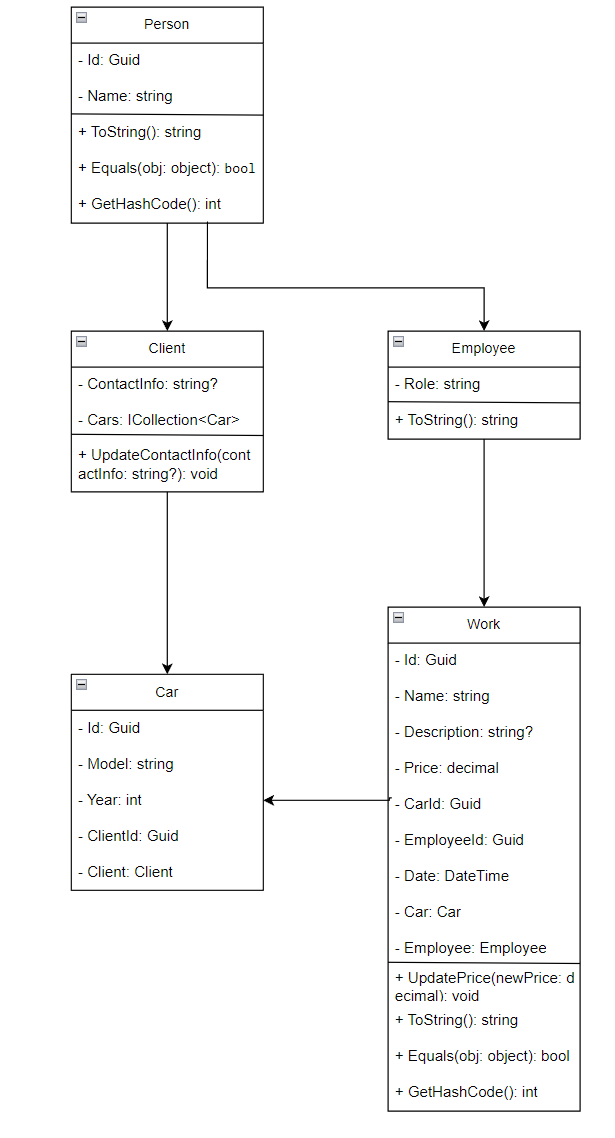


Рисунок 1 – UML - диаграмма

# Код программы

**Car.cs**

// <copyright file="Car.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace Carshop

{

/// <summary>

/// Класс, представляющий автомобиль.

/// </summary>

public class Car

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Car"/>.

/// Конструктор для создания нового автомобиля.

/// </summary>

/// <param name="mark">Марка автомобиля.</param>

/// <param name="model">Модель автомобиля.</param>

/// <param name="year">Год выпуска автомобиля.</param>

/// <param name="clientId">Идентификатор клиента.</param>

public Car(string mark, string model, int year, Guid clientId)

{

if (string.IsNullOrWhiteSpace(mark))

{

throw new ArgumentNullException(nameof(mark), "Марка автомобиля не может быть пустой.");

}

if (string.IsNullOrWhiteSpace(model))

{

throw new ArgumentNullException(nameof(model), "Модель автомобиля не может быть пустой.");

}

if (year < 1900 || year > DateTime.Now.Year + 1)

{

throw new ArgumentOutOfRangeException(nameof(year), "Неверный год выпуска автомобиля.");

}

this.Id = Guid.NewGuid();

this.Mark = mark.Trim();

this.Model = model.Trim();

this.Year = year;

this.ClientId = clientId;

}

/// <summary>

/// Уникальный идентификатор автомобиля.

/// </summary>

public Guid Id { get; private set; }

/// <summary>

/// Марка автомобиля.

/// </summary>

public string Mark { get; private set; }

/// <summary>

/// Модель автомобиля.

/// </summary>

public string Model { get; private set; }

/// <summary>

/// Год выпуска автомобиля.

/// </summary>

public int Year { get; private set; }

/// <summary>

/// Идентификатор клиента, которому принадлежит автомобиль.

/// </summary>

public Guid ClientId { get; private set; }

/// <summary>

/// Навигационное свойство для клиента.

/// </summary>

public Client Client { get; private set; }

/// <summary>

/// Переопределение метода ToString для удобного отображения автомобиля.

/// </summary>

/// <returns>Описание автомобиля в виде строки.</returns>

public override string ToString()

{

return $"{this.Mark} {this.Model} ({this.Year})";

}

/// <summary>

/// Переопределение метода Equals для сравнения автомобилей.

/// </summary>

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

/// <returns>True, если автомобили равны.</returns>

public override bool Equals(object? obj)

{

if (obj is not Car other)

{

return false;

}

return this.Id == other.Id &&

this.Mark == other.Mark &&

this.Model == other.Model &&

this.Year == other.Year &&

this.ClientId == other.ClientId;

}

/// <summary>

/// Переопределение метода GetHashCode.

/// </summary>

/// <returns>Хэш-код автомобиля.</returns>

public override int GetHashCode()

{

return HashCode.Combine(this.Id, this.Mark, this.Model, this.Year, this.ClientId);

}

}

}

**Client.cs**

namespace Cinema

// <copyright file="Client.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace Carshop

{

/// <summary>

/// Класс, представляющий клиента.

/// </summary>

public class Client : Person

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Client"/>.

/// Конструктор для создания клиента с обязательным именем.

/// </summary>

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

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

public Client(string name, string? contactInfo = null)

: base(name)

{

this.ContactInfo = contactInfo?.Trim();

}

/// <summary>

/// Контактная информация клиента.

/// </summary>

public string? ContactInfo { get; private set; }

/// <summary>

/// Список автомобилей, принадлежащих клиенту.

/// </summary>

public ICollection<Car> Cars { get; private set; } = new List<Car>();

/// <summary>

/// Устанавливает новую контактную информацию для клиента.

/// </summary>

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

public void UpdateContactInfo(string? contactInfo)

{

this.ContactInfo = contactInfo?.Trim();

}

/// <summary>

/// Переопределение метода ToString для удобного отображения клиента.

/// </summary>

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

public override string ToString()

{

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

}

}

}

**Employee.cs**

// <copyright file="Employee.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace Carshop

{

/// <summary>

/// Класс, представляющий работника.

/// </summary>

public class Employee : Person

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Employee"/>.

/// Конструктор для создания нового работника.

/// </summary>

/// <param name="name">Имя работника.</param>

/// <param name="role">Роль работника.</param>

public Employee(string name, string role)

: base(name)

{

if (string.IsNullOrWhiteSpace(role))

{

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

}

this.Role = role.Trim();

}

/// <summary>

/// Роль работника (например, "Механик" или "Мастер").

/// </summary>

public string Role { get; private set; }

/// <summary>

/// Переопределение метода ToString для удобного отображения работника.

/// </summary>

/// <returns>Описание работника в виде строки.</returns>

public override string ToString()

{

return $"{this.Name} - {this.Role}";

}

}

}

**Person.cs**

// <copyright file="Person.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace Carshop

{

/// <summary>

/// Абстрактный класс, представляющий человека.

/// </summary>

public abstract class Person

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Person"/>.

/// Конструктор для создания нового человека с обязательным именем.

/// </summary>

/// <param name="name">Имя человека.</param>

protected Person(string name)

{

if (string.IsNullOrWhiteSpace(name))

{

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

}

this.Id = Guid.NewGuid();

this.Name = name.Trim();

}

/// <summary>

/// Уникальный идентификатор человека.

/// </summary>

public Guid Id { get; protected set; }

/// <summary>

/// Имя человека.

/// </summary>

public string Name { get; protected set; }

/// <summary>

/// Переопределение метода ToString для удобного отображения человека.

/// </summary>

/// <returns>Имя человека.</returns>

public override string ToString()

{

return this.Name;

}

/// <summary>

/// Переопределение метода Equals для сравнения людей.

/// </summary>

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

/// <returns>True, если люди равны.</returns>

public override bool Equals(object? obj)

{

if (obj is not Person other)

{

return false;

}

return this.Id == other.Id &&

this.Name == other.Name;

}

/// <summary>

/// Переопределение метода GetHashCode.

/// </summary>

/// <returns>Хэш-код человека.</returns>

public override int GetHashCode()

{

return HashCode.Combine(this.Id, this.Name);

}

}

}

**Work.cs**

// <copyright file="Work.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace Carshop

{

/// <summary>

/// Класс, представляющий работу (услугу) в автосервисе.

/// </summary>

public class Work

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Work"/>.

/// Конструктор для создания новой работы.

/// </summary>

/// <param name="name">Название работы.</param>

/// <param name="description">Описание работы.</param>

/// <param name="price">Стоимость работы.</param>

/// <param name="carId">Идентификатор автомобиля.</param>

/// <param name="employeeId">Идентификатор работника.</param>

/// <param name="date">Дата выполнения работы.</param>

public Work(string name, string? description, decimal price, Guid carId, Guid employeeId, DateTime date)

{

if (string.IsNullOrWhiteSpace(name))

{

throw new ArgumentNullException(nameof(name), "Название работы не может быть пустым.");

}

if (price < 0)

{

throw new ArgumentOutOfRangeException(nameof(price), "Стоимость работы не может быть отрицательной.");

}

this.Id = Guid.NewGuid();

this.Name = name.Trim();

this.Description = description?.Trim();

this.Price = price;

this.CarId = carId;

this.EmployeeId = employeeId;

this.Date = date;

}

/// <summary>

/// Уникальный идентификатор работы.

/// </summary>

public Guid Id { get; private set; }

/// <summary>

/// Название работы.

/// </summary>

public string Name { get; private set; }

/// <summary>

/// Описание работы.

/// </summary>

public string? Description { get; private set; }

/// <summary>

/// Стоимость работы.

/// </summary>

public decimal Price { get; private set; }

/// <summary>

/// Идентификатор автомобиля, к которому относится работа.

/// </summary>

public Guid CarId { get; private set; }

/// <summary>

/// Навигационное свойство для автомобиля.

/// </summary>

public Car Car { get; private set; }

/// <summary>

/// Идентификатор работника, выполняющего работу.

/// </summary>

public Guid EmployeeId { get; private set; }

/// <summary>

/// Навигационное свойство для работника.

/// </summary>

public Employee Employee { get; private set; }

/// <summary>

/// Дата выполнения работы.

/// </summary>

public DateTime Date { get; private set; }

/// <summary>

/// Обновляет стоимость работы.

/// </summary>

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

public void UpdatePrice(decimal newPrice)

{

if (newPrice < 0)

{

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

}

this.Price = newPrice;

}

/// <summary>

/// Переопределение метода ToString для удобного отображения работы.

/// </summary>

/// <returns>Описание работы в виде строки.</returns>

public override string ToString()

{

return $"{this.Name} ({this.Date:yyyy-MM-dd}) - {this.Price:C}";

}

/// <summary>

/// Переопределение метода Equals для сравнения работ.

/// </summary>

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

/// <returns>True, если работы равны.</returns>

public override bool Equals(object? obj)

{

if (obj is not Work other)

{

return false;

}

return this.Id == other.Id &&

this.Name == other.Name &&

this.Description == other.Description &&

this.Price == other.Price &&

this.CarId == other.CarId &&

this.EmployeeId == other.EmployeeId &&

this.Date == other.Date;

}

/// <summary>

/// Переопределение метода GetHashCode.

/// </summary>

/// <returns>Хэш-код работы.</returns>

public override int GetHashCode()

{

return HashCode.Combine(this.Id, this.Name, this.Description, this.Price, this.CarId, this.EmployeeId, this.Date);

}

}

}

# Прохождение тестов

**CarTests.cs**

// <copyright file="CarTests.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace CarshopTest

{

using System;

using Carshop;

using NUnit.Framework;

/// <summary>

/// Тесты для класса <see cref="Car"/>.

/// </summary>

[TestFixture]

public sealed class CarTests

{

[Test]

public void Ctor\_ValidData\_Success()

{

// Arrange

var clientId = Guid.NewGuid();

// Act

var car = new Car("Toyota", "Corolla", 2020, clientId);

// Assert

Assert.Multiple(() =>

{

Assert.That(car.Mark, Is.EqualTo("Toyota"));

Assert.That(car.Model, Is.EqualTo("Corolla"));

Assert.That(car.Year, Is.EqualTo(2020));

Assert.That(car.ClientId, Is.EqualTo(clientId));

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

});

}

[Test]

public void Ctor\_NullMake\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Car(null!, "Corolla", 2020, Guid.NewGuid()));

}

[Test]

public void Ctor\_EmptyMake\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Car(string.Empty, "Corolla", 2020, Guid.NewGuid()));

}

[Test]

public void Ctor\_NullModel\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Car("Toyota", null!, 2020, Guid.NewGuid()));

}

[Test]

public void Ctor\_EmptyModel\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Car("Toyota", string.Empty, 2020, Guid.NewGuid()));

}

[Test]

public void Ctor\_InvalidYear\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() => \_ = new Car("Toyota", "Corolla", 1899, Guid.NewGuid()));

Assert.Throws<ArgumentOutOfRangeException>(() => \_ = new Car("Toyota", "Corolla", DateTime.Now.Year + 2, Guid.NewGuid()));

}

[Test]

public void ToString\_ValidData\_ReturnsExpected()

{

// Arrange

var car = new Car("Toyota", "Corolla", 2020, Guid.NewGuid());

// Act

var result = car.ToString();

// Assert

Assert.That(result, Is.EqualTo("Toyota Corolla (2020)"));

}

[Test]

public void Equals\_SameData\_ReturnsFalse()

{

// Arrange

var car1 = new Car("Toyota", "Corolla", 2020, Guid.NewGuid());

var car2 = new Car("Toyota", "Corolla", 2020, Guid.NewGuid());

// Act & Assert

Assert.That(car1, Is.Not.EqualTo(car2), "Cars with the same data but different GUIDs should not be equal.");

}

[Test]

public void GetHashCode\_DifferentCars\_ReturnsDifferentHashCodes()

{

// Arrange

var car1 = new Car("Toyota", "Corolla", 2020, Guid.NewGuid());

var car2 = new Car("Honda", "Civic", 2021, Guid.NewGuid());

// Act

var hash1 = car1.GetHashCode();

var hash2 = car2.GetHashCode();

// Assert

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

}

}

}

**ClientTests.cs**

// <copyright file="ClientTests.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace CarshopTest

{

using System;

using Carshop;

using NUnit.Framework;

/// <summary>

/// Тесты для класса <see cref="Client"/>.

/// </summary>

[TestFixture]

public sealed class ClientTests

{

[Test]

public void Ctor\_ValidData\_Success()

{

// Arrange & Act

var client = new Client("Иван Иванов", "ivanov@example.com");

// Assert

Assert.Multiple(() =>

{

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

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

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

});

}

[Test]

public void Ctor\_NullName\_ThrowsException()

{

// Arrange, Act & Assert

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

}

[Test]

public void Ctor\_EmptyName\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Client(string.Empty));

}

[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 client = new Client(name, contact);

// Act

var actual = client.ToString();

// Assert

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

}

[Test]

public void UpdateContactInfo\_ValidData\_Success()

{

// Arrange

var client = new Client("Иван Иванов", "ivanov@example.com");

// Act

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

// Assert

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

}

[Test]

public void UpdateContactInfo\_NullData\_SetsNull()

{

// Arrange

var client = new Client("Иван Иванов", "ivanov@example.com");

// Act

client.UpdateContactInfo(null);

// Assert

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

}

[Test]

public void Equals\_SameData\_ReturnsFalse()

{

// Arrange

var client1 = new Client("Иван Иванов", "ivanov@example.com");

var client2 = new Client("Иван Иванов", "ivanov@example.com");

// Act & Assert

Assert.That(client1, Is.Not.EqualTo(client2), "Clients with the same data but different GUIDs should not be equal.");

}

[Test]

public void GetHashCode\_DifferentClients\_ReturnsDifferentHashCodes()

{

// Arrange

var client1 = new Client("Иван Иванов", "ivanov@example.com");

var client2 = new Client("Мария Смирнова", "maria@example.com");

// Act

var hash1 = client1.GetHashCode();

var hash2 = client2.GetHashCode();

// Assert

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

}

}

}

**EmploeeTests.cs**

// <copyright file="EmployeeTests.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace CarshopTest

{

using System;

using Carshop;

using NUnit.Framework;

/// <summary>

/// Тесты для класса <see cref="Employee"/>.

/// </summary>

[TestFixture]

public sealed class EmployeeTests

{

[Test]

public void Ctor\_ValidData\_Success()

{

// Arrange & Act

var employee = new Employee("Алексей Павлов", "Механик");

// Assert

Assert.Multiple(() =>

{

Assert.That(employee.Name, Is.EqualTo("Алексей Павлов"));

Assert.That(employee.Role, Is.EqualTo("Механик"));

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

});

}

[Test]

public void Ctor\_NullName\_ThrowsException()

{

// Arrange, Act & Assert

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

}

[Test]

public void Ctor\_EmptyName\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Employee(string.Empty, "Механик"));

}

[Test]

public void Ctor\_NullRole\_ThrowsException()

{

// Arrange, Act & Assert

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

}

[Test]

public void Ctor\_EmptyRole\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Employee("Алексей Павлов", string.Empty));

}

[Test]

public void ToString\_ValidData\_ReturnsExpected()

{

// Arrange

var employee = new Employee("Алексей Павлов", "Механик");

// Act

var result = employee.ToString();

// Assert

Assert.That(result, Is.EqualTo("Алексей Павлов - Механик"));

}

[Test]

public void Equals\_SameData\_ReturnsFalse()

{

// Arrange

var employee1 = new Employee("Алексей Павлов", "Механик");

var employee2 = new Employee("Алексей Павлов", "Механик");

// Act & Assert

Assert.That(employee1.Equals(employee2), Is.False, "Employees with the same data but different GUIDs should not be equal.");

}

[Test]

public void GetHashCode\_DifferentEmployees\_ReturnsDifferentHashCodes()

{

// Arrange

var employee1 = new Employee("Алексей Павлов", "Механик");

var employee2 = new Employee("Мария Смирнова", "Мастер");

// Act

var hash1 = employee1.GetHashCode();

var hash2 = employee2.GetHashCode();

// Assert

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

}

}

}

**WorkTests.cs**

// <copyright file="WorkTests.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace CarshopTest

{

using System;

using Carshop;

using NUnit.Framework;

/// <summary>

/// Тесты для класса <see cref="ServiceJob"/>.

/// </summary>

[TestFixture]

public sealed class WorkTests

{

[Test]

public void Ctor\_ValidData\_Success()

{

// Arrange

var carId = Guid.NewGuid();

var employeeId = Guid.NewGuid();

var date = DateTime.Now;

// Act

var work = new Work("Замена масла", "Описание работы", 1500m, carId, employeeId, date);

// Assert

Assert.Multiple(() =>

{

Assert.That(work.Name, Is.EqualTo("Замена масла"));

Assert.That(work.Description, Is.EqualTo("Описание работы"));

Assert.That(work.Price, Is.EqualTo(1500m));

Assert.That(work.CarId, Is.EqualTo(carId));

Assert.That(work.EmployeeId, Is.EqualTo(employeeId));

Assert.That(work.Date, Is.EqualTo(date));

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

});

}

[Test]

public void Ctor\_NullName\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Work(null!, "Описание работы", 1500m, Guid.NewGuid(), Guid.NewGuid(), DateTime.Now));

}

[Test]

public void Ctor\_EmptyName\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentNullException>(() => \_ = new Work(string.Empty, "Описание работы", 1500m, Guid.NewGuid(), Guid.NewGuid(), DateTime.Now));

}

[Test]

public void Ctor\_NegativePrice\_ThrowsException()

{

// Arrange, Act & Assert

Assert.Throws<ArgumentOutOfRangeException>(() => \_ = new Work("Замена масла", "Описание работы", -100m, Guid.NewGuid(), Guid.NewGuid(), DateTime.Now));

}

[Test]

public void UpdatePrice\_ValidPrice\_Success()

{

// Arrange

var serviceJob = new Work("Замена масла", "Описание работы", 1500m, Guid.NewGuid(), Guid.NewGuid(), DateTime.Now);

// Act

serviceJob.UpdatePrice(2000m);

// Assert

Assert.That(serviceJob.Price, Is.EqualTo(2000m));

}

[Test]

public void UpdatePrice\_NegativePrice\_ThrowsException()

{

// Arrange

var serviceJob = new Work("Замена масла", "Описание работы", 1500m, Guid.NewGuid(), Guid.NewGuid(), DateTime.Now);

// Act & Assert

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

}

[Test]

public void Equals\_SameData\_ReturnsFalse()

{

// Arrange

var carId = Guid.NewGuid();

var employeeId = Guid.NewGuid();

var date = DateTime.Now;

var serviceJob1 = new Work("Замена масла", "Описание работы", 1500m, carId, employeeId, date);

var serviceJob2 = new Work("Замена масла", "Описание работы", 1500m, carId, employeeId, date);

// Act & Assert

Assert.That(serviceJob1.Equals(serviceJob2), Is.False, "ServiceJobs with the same data but different GUIDs should not be equal.");

}

[Test]

public void GetHashCode\_DifferentServiceJobs\_ReturnsDifferentHashCodes()

{

// Arrange

var carId = Guid.NewGuid();

var employeeId = Guid.NewGuid();

var date = DateTime.Now;

var serviceJob1 = new Work("Замена масла", "Описание работы", 1500m, carId, employeeId, date);

var serviceJob2 = new Work("Ремонт двигателя", "Описание работы", 5000m, carId, employeeId, date);

// Act

var hash1 = serviceJob1.GetHashCode();

var hash2 = serviceJob2.GetHashCode();

// Assert

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

}

}

}

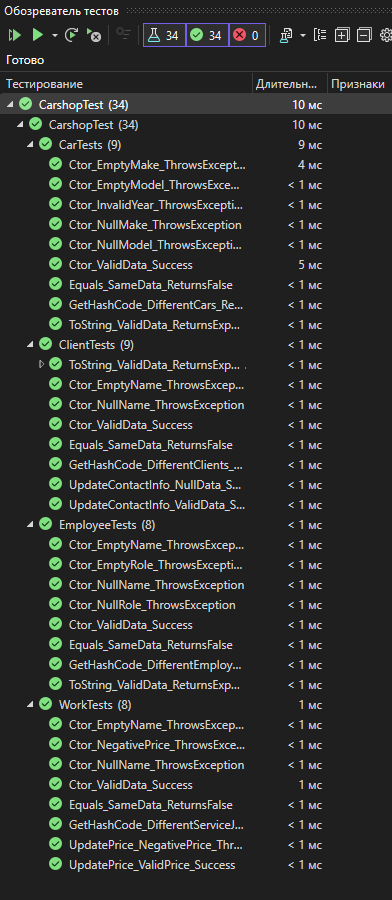


Рисунок 2 - Результат прохождения тестов

# Настройка миграции

**CarConfiguration.cs**

// <copyright file="CarConfiguration.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Configurations

{

using Carshop;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

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

/// </summary>

public sealed class CarConfiguration : IEntityTypeConfiguration<Car>

{

/// <inheritdoc/>

public void Configure(EntityTypeBuilder<Car> builder)

{

// Установка первичного ключа

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

// Конфигурация свойства Make

\_ = builder.Property(car => car.Mark)

.HasMaxLength(100)

.IsRequired();

// Конфигурация свойства Model

\_ = builder.Property(car => car.Model)

.HasMaxLength(100)

.IsRequired();

// Конфигурация свойства Year

\_ = builder.Property(car => car.Year)

.IsRequired();

// Настройка связи с Client

\_ = builder.HasOne(car => car.Client)

.WithMany(client => client.Cars)

.HasForeignKey(car => car.ClientId)

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

}

}

}

**DirectorConfiguration.cs**

// <copyright file="ClientConfiguration.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Configurations

{

using Carshop;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

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

/// </summary>

public sealed class ClientConfiguration : IEntityTypeConfiguration<Client>

{

/// <inheritdoc/>

public void Configure(EntityTypeBuilder<Client> builder)

{

// Установка первичного ключа

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

// Конфигурация свойства Name

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

.HasMaxLength(200)

.IsRequired();

// Конфигурация свойства ContactInfo

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

.HasMaxLength(300)

.IsRequired(false);

// Опционально: настройка индекса по имени клиента

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

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

}

}

}

**FilmConfiguration.cs**

// <copyright file="EmployeeConfiguration.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Configurations

{

using Carshop;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

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

/// </summary>

public sealed class EmployeeConfiguration : IEntityTypeConfiguration<Employee>

{

/// <inheritdoc/>

public void Configure(EntityTypeBuilder<Employee> builder)

{

// Установка первичного ключа

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

// Конфигурация свойства Name

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

.HasMaxLength(200)

.IsRequired();

// Конфигурация свойства Role

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

.HasMaxLength(100)

.IsRequired();

// Опционально: Добавление индекса на имя и роль

\_ = builder.HasIndex(employee => new { employee.Name, employee.Role })

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

}

}

}

**HallConfiguration.cs**

// <copyright file="WorkConfiguration.cs" company="Чевычалов А.В.">

// Copyright (c) Чевычалов А.В.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Configurations

{

using Carshop;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

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

/// </summary>

public sealed class WorkConfiguration : IEntityTypeConfiguration<Work>

{

/// <inheritdoc/>

public void Configure(EntityTypeBuilder<Work> builder)

{

// Установка первичного ключа

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

// Конфигурация свойства Name

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

.HasMaxLength(200)

.IsRequired();

// Конфигурация свойства Description

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

.HasMaxLength(500)

.IsRequired(false);

// Конфигурация свойства Price

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

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

.IsRequired();

// Конфигурация свойства Date

\_ = builder.Property(work => work.Date)

.IsRequired();

// Настройка связи с Car

\_ = builder.HasOne(work => work.Car)

.WithMany()

.HasForeignKey(work => work.CarId)

.OnDelete(DeleteBehavior.Cascade); // Удаление работы при удалении автомобиля

// Настройка связи с Employee

\_ = builder.HasOne(work => work.Employee)

.WithMany()

.HasForeignKey(work => work.EmployeeId)

.OnDelete(DeleteBehavior.Cascade); // Удаление работы при удалении работника

}

}

}

# ER-диаграмма

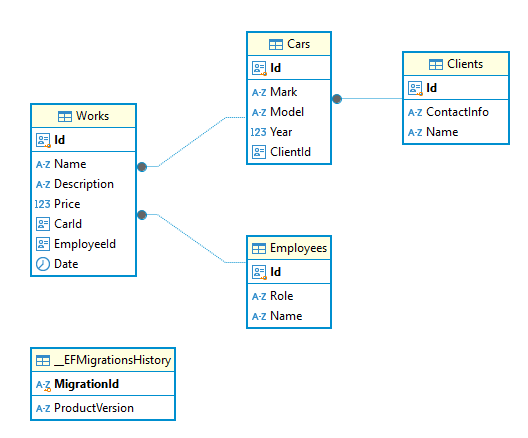


Рисунок 3 – ER-диаграмма

# 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).