**ДОКУМЕНТАЦИЯ К ПРОГРАММЕ «SPORT SHOP»**

В данном проекте реализуется спортивный магазин в приложении WPF. Данная программа реализована с помощью паттерна MVVM.

Model-View-ViewModel (MVVM) – это программный архитектурный шаблон, который облегчает разделение разработки графического пользовательского интерфейса (представления) – будь то с помощью языка разметки или GUI-кода - от разработки бизнес-логики или внутренней логики (модели), чтобы вид не зависит от какой-либо конкретной платформы модели.

Реализация данной архитектуры основывается на организации проекта. Чтобы подробнее разобраться в программе, необходимо понять, как устроена данная модель и проект.

База данных написана с помощью программы PostgreSQL. Данная реализация выбрана из-за удобства использования. Для базы данных используется также Entity Framework Core.

Entity Framework (EF) Core — это простая, кроссплатформенная и расширяемая версия популярной технологии доступа к данным Entity Framework с открытым исходным кодом.

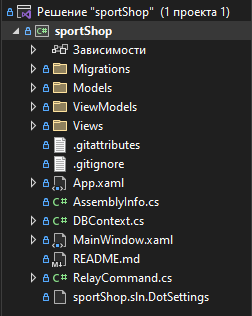
EF Core может использоваться в качестве объектно-реляционного модуля сопоставления (O/RM), который:

Позволяет разработчикам .NET работать с базой данных с помощью объектов .NET.

Устраняет необходимость в большей части кода для доступа к данным, который обычно приходится писать.

**Разбор архитектуры проекта**

Ниже приведены файлы проекта:



Папки Models, ViewModels, Views являются организацией архитектуры MVVM. Папка Migration – является организацией кода для компилирования базы данных из модели в SQL. Также в данном проекте присутствует файлы AssemblyInfo.cs, DBContext.cs, RelayCommand.cs. В DBContext описана модель базы данных. AssemblyInfo.cs поставляет компилятору необходимую информацию для создания библиотеки. RealyCommand описывает логику команд, используемую в программе. App.xaml и sportShop.sln. DotSettings – системные файлы, которые создаются вместе с проектом. MainWindow.xaml – главное окно программы. Файлы .gitignore, .gitattributes и README.md – файлы для контроля версий git, их мы рассматривать не будем.

1. Папка Model:

В данной папке содержится файл Model.cs. При использовании архитектуры MVVM (Model-View-ViewModel) содержание файла "model" включает в себя данные и бизнес-логику приложения. Модель представляет собой объекты данных, которые используются в приложении, а также методы для их обработки. Она отвечает за управление данными и их бизнес-логикой, независимо от пользовательского интерфейса.

Примерное содержание файла "model" в архитектуре MVVM:

* Определение структуры данных и их свойств;
* Методы для получения, обновления и обработки данных;
* Логика, связанная с бизнес-процессами приложения.

Ниже приведён рисунок с кодом файла Model.cs:

using System.Collections.Generic;

using System.ComponentModel;

namespace sportShop.Models;

public class Client

{

public int Id { get; set; }

public required string Name { get; set; }

public required int Age { get; set; }

public required string Login { get; set; }

public required string Password { get; set; }

public ICollection<Product> Products { get; set; } = new List<Product>();

}

public class Manager

{

public int Id { get; set; }

public required string Login { get; set; }

public required string Password { get; set; }

}

public class Administrator

{

public int Id { get; set; }

public required string Login { get; set; }

public required string Password { get; set; }

}

sealed public class Product : INotifyPropertyChanged

{

public int Id { get; set; }

public required string Name { get; set; }

public required int ProductTypeId { get; set; }

public required ProductTypes ProductType { get; set; }

public required int ProductCount { get; set; }

private double \_price;

public required double Price

{

get => \_price;

set

{

\_price = value;

DiscountedPrice = GetDiscountedPrice();

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Price)));

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(DiscountedPrice)));

}

}

private int \_sale;

public required int Sale

{

get => \_sale;

set

{

\_sale = value switch

{

< 0 => 0,

> 100 => 100,

\_ => value

};

DiscountedPrice = GetDiscountedPrice();

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Sale)));

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(DiscountedPrice)));

}

}

public double DiscountedPrice { get; private set; }

public double GetDiscountedPrice()

{

var discount = (double) \_sale / 100;

var discountedPrice = \_price - \_price \* discount;

return discountedPrice;

}

public required int FabricId { get; set; }

public required Fabric Fabric { get; set; }

public event PropertyChangedEventHandler? PropertyChanged;

}

public class Fabric

{

public int Id { get; set; }

public required string Name { get; set; }

public ICollection<Product> Products { get; set; } = new List<Product>();

}

public class ProductTypes

{

public int Id { get; set; }

public required string Name { get; set; }

}

Все классы в данном файле представляют собой логику базы данных, а методы в них необходимы для внутренней логики. Здесь содержится классы, которые обозначают роли доступа (Client, Administrator, Manager), также классы для продукта. Суть каждой роли будет рассмотрена ниже.

1. Папка ViewModel

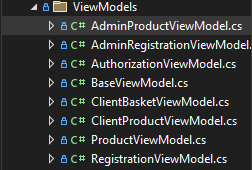
Содержание папки "viewmodel" включает в себя логику представления данных и взаимодействие между моделью (Model) и пользовательским интерфейсом (View). ViewModel представляет данные и методы, необходимые для отображения информации на пользовательском интерфейсе и обработки пользовательских действий.

Примерное содержание файла "ViewModel" в архитектуре MVVM:

* Логика, связанная с отображением данных на пользовательском интерфейсе;
* Методы для обработки пользовательских действий и взаимодействия с моделью;
* Связывание данных между моделью и представлением;
* Управление состоянием представления и обновлением данных.

Таким образом, файл "ViewModel" в архитектуре MVVM содержит логику, необходимую для связи между моделью и пользовательским интерфейсом, обеспечивая разделение бизнес-логики и отображения данных.

Содержание папки ViewModel:



В первых двух файлах содержится логика для взаимодействия интерфейса и кода при использовании роли «Admin», третий файл – содержит базовое взаимодействие интерфейса, дальше идёт логика для «Client», потом логика для самого продукта.

1. Папка View

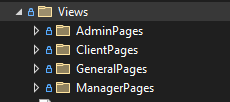
"View" содержит представление данных, то есть пользовательский интерфейс приложения. В этом файле определяется, как данные из ViewModel будут отображаться для пользователя, а также как пользовательские действия будут передаваться обратно в ViewModel для обработки.

Примерное содержание файла "view" в архитектуре MVVM:

* Определение пользовательского интерфейса с использованием соответствующих элементов (например, кнопки, поля ввода, списки и т. д.);
* Привязка данных из ViewModel к элементам пользовательского интерфейса;
* Логика отображения данных и реагирования на пользовательские действия;
* Отсутствие прямой бизнес-логики, так как она должна быть вынесена в ViewModel.

Таким образом, файл "view" в архитектуре MVVM содержит описание пользовательского интерфейса и его взаимодействие с ViewModel для отображения данных и обработки пользовательских действий.

В данной папке содержатся файлы для каждой из роли:



Папка же GeneralPages – содержит ахитиктуру базовой страницы.

1. Враимодействие с базой данных.

Для базы данных используется файлы папки Migration, которые компилируют модель базы данных в sql код. Сама же база данных подключается в файле DBContext.cs:

using Microsoft.EntityFrameworkCore;

using sportShop.Models;

#pragma warning disable CS8618 // Non-nullable field must contain a non-null value when exiting constructor. Consider declaring as nullable.

namespace sportShop;

/// <summary>

/// подключение базы данных

/// </summary>

public class DbContext : Microsoft.EntityFrameworkCore.DbContext

{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>

optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=postgres;Username=postgres;Password=Boec\_UFC1123");

public DbSet<Product> Products { get; set; }

public DbSet<Fabric> Fabrics { get; set; }

public DbSet<Manager> Managers { get; set; }

public DbSet<Administrator> Administrators { get; set; }

public DbSet<Client> Clients { get; set; }

public DbSet<ProductTypes> ProductTypes { get; set; }

}

В данном файле с помощью дерективы using sportShop.Model подключается файл Model.cs для наследования модели базы данных. Вызывается она в свою очередь через код ниже:

public DbSet<Product> Products { get; set; }

public DbSet<Fabric> Fabrics { get; set; }

public DbSet<Manager> Managers { get; set; }

public DbSet<Administrator> Administrators { get; set; }

public DbSet<Client> Clients { get; set; }

public DbSet<ProductTypes> ProductTypes { get; set; }

Подключается база данных через данную строчку:

optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=postgres;Username=postgres;Password=Boec\_UFC1123");

В данной строчке содержится хост подключения, порт, название базы данных, и пароль подключения.