**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине: «Программирование сетевых приложений»

на тему: **«**Микро-сервисная архитектура**»**

Выполнил: студент гр. ИТИ-41

Ковшаров Г. Ю.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2025

**Цель работы:** освоить методов построения приложений, основанных на микро-сервисной архитектуре. Научиться разбивать задачу на сервисы и выстраивать с ними взаимодействие.

**Задание:**

Разработать сетевое приложение согласно варианту. Необходимо реализовать *CRUD* операции для минимума трех сущностей и агрегатные вычисления минимум для пяти показателей (самый прибыльный месяц, самый популярный музыкальный альбом, длина маршрута, пройдённого водителем за период времени и т.п.) в соответствующей предметной области, **использовать микро-сервисы** в построении архитектуры системы. Для взаимодействия сервисов **использовать TCP или UDP сокеты**.

Таблица 1 – Вариант задания

|  |  |
| --- | --- |
| **Вариант** | **Предметная область** |
| 13 | Видеотека |

**Ход работы**

Для решения поставленной задачи была спроектирована система, состоящая из нескольких независимых, слабосвязанных сервисов, каждый из которых отвечает за свой узкий бизнес-контекст. Такой подход позволил изолировать данные, упростить разработку и обеспечить возможность независимого масштабирования и развертывания компонентов.

Система была разделена на следующие микросервисы:

1. **Сервис Меню (PizzaService):** отвечает исключительно за управление каталогом продукции. Он владеет данными о пиццах и предоставляет полный набор *CRUD*-операций для сущности *Pizza*. Сервис использует собственную базу данных *pizzeria.db* (*SQLite*) и не зависит от других компонентов системы.
2. **Сервис Заказов (*OrderService*):** является ядром бизнес-логики. Он управляет жизненным циклом заказов, реализуя *CRUD*-операции для сущностей *Order* и *OrderItem*. При создании или изменении заказа данный сервис выступает в роли клиента и обращается к *MenuService* для получения актуальной информации о пиццах (цена, название), демонстрируя прямое межсервисное взаимодействие.
3. **Сервис Отчетов (*ReportingService*):** предназначен для выполнения аналитических и агрегатных вычислений. Этот сервис не имеет собственного хранилища данных, а вместо этого агрегирует информацию, получаемую от других сервисов (преимущественно от *OrderService*). Он реализует вычисление пяти ключевых показателей: общая выручка за период, самая популярная и самая прибыльная пицца, средний чек и количество заказов по статусам.
4. ***API* Шлюз (*API Gateway*):** выступает в роли единой точки входа для всей системы. Внешний клиент (*WPF*-приложение) взаимодействует только со шлюзом, который, в свою очередь, маршрутизирует запросы к соответствующим внутренним микросервисам. Шлюз скрывает внутреннюю топологию сети, упрощает клиентскую логику и служит центральным местом для потенциальной реализации сквозных функций, таких как аутентификация или логирование.

Взаимодействие между всеми компонентами системы, включая связь «клиент-шлюз» и «шлюз-сервис», реализовано с использованием *TCP*-сокетов. Для организации обмена данными был разработан простой текстовый протокол прикладного уровня, имитирующий *REST* поверх *HTTP*. Запросы содержат стартовую строку (метод и путь), заголовки (включая *Content*-*Length* для запросов с телом) и тело в формате *JSON*. Каждый микросервис, включая шлюз, реализован как отдельное консольное приложение, прослушивающее свой выделенный *TCP*-порт. В качестве идентификаторов для всех ключевых сущностей (*Pizza*, *Order*) были использованы глобально уникальные идентификаторы (*GUID*), что является стандартной практикой для распределенных систем для предотвращения конфликтов *ID*.

На рисунке 1 представлен вывод из консольного приложения   
*API Gateway.*

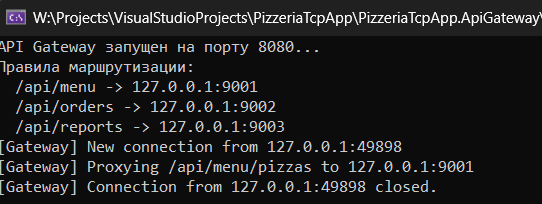


Рисунок 1 – Вывод из консольного приложения *API Gateway*

В качестве клиента было разработано десктопное приложение на технологии WPF. Приложение имеет многовкладочный интерфейс, позволяющий пользователю выполнять все необходимые операции:

* **Создание заказа:** выбор пицц из динамически загружаемого меню и формирование корзины.
* **Управление меню:** просмотр, добавление, редактирование и удаление пицц из каталога.
* **Управление заказами:** просмотр списка всех заказов, а также редактирование деталей существующего заказа, включая изменение его состава (добавление/удаление позиций) и статуса.
* **Просмотр отчетов:** формирование и отображение аналитических данных от *ReportingService*.

Вся сетевая логика на клиенте инкапсулирована в отдельную библиотеку, что делает код *UI* чистым и сфокусированным на представлении. Все сетевые вызовы выполняются асинхронно, обеспечивая отзывчивость пользовательского интерфейса.

На рисунках 2-5 продемонстрирован пользовательский интерфейс приложения.

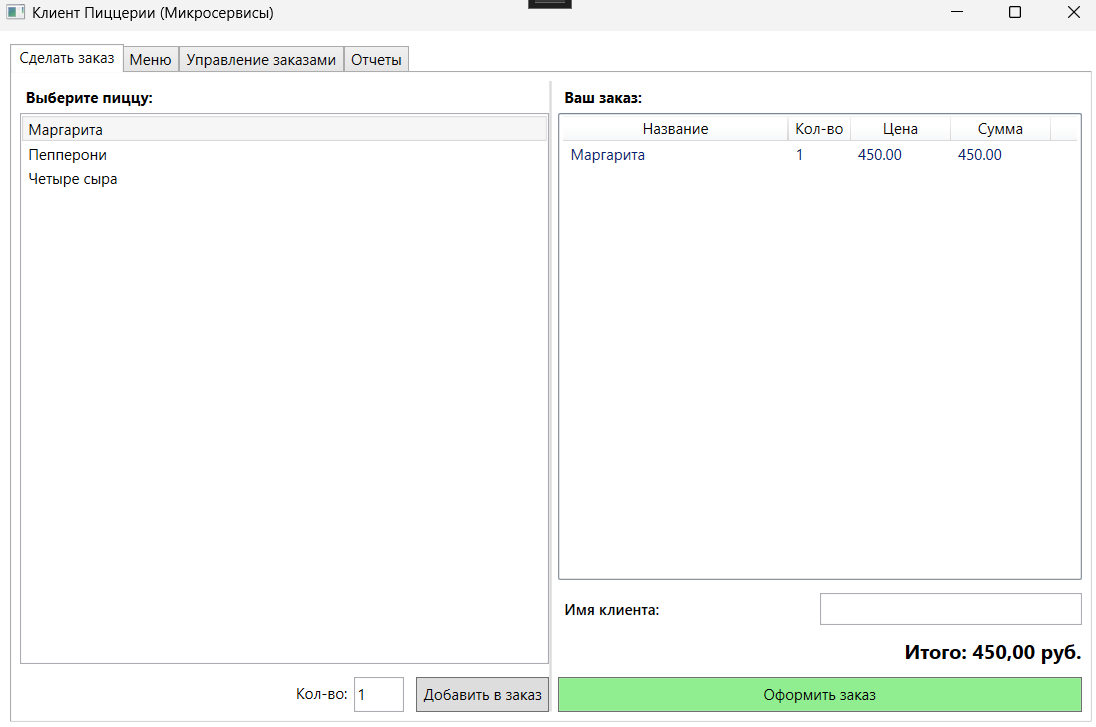


Рисунок 2 – Скриншот *UI* на вкладке «Сделать заказ» *WPF* приложения

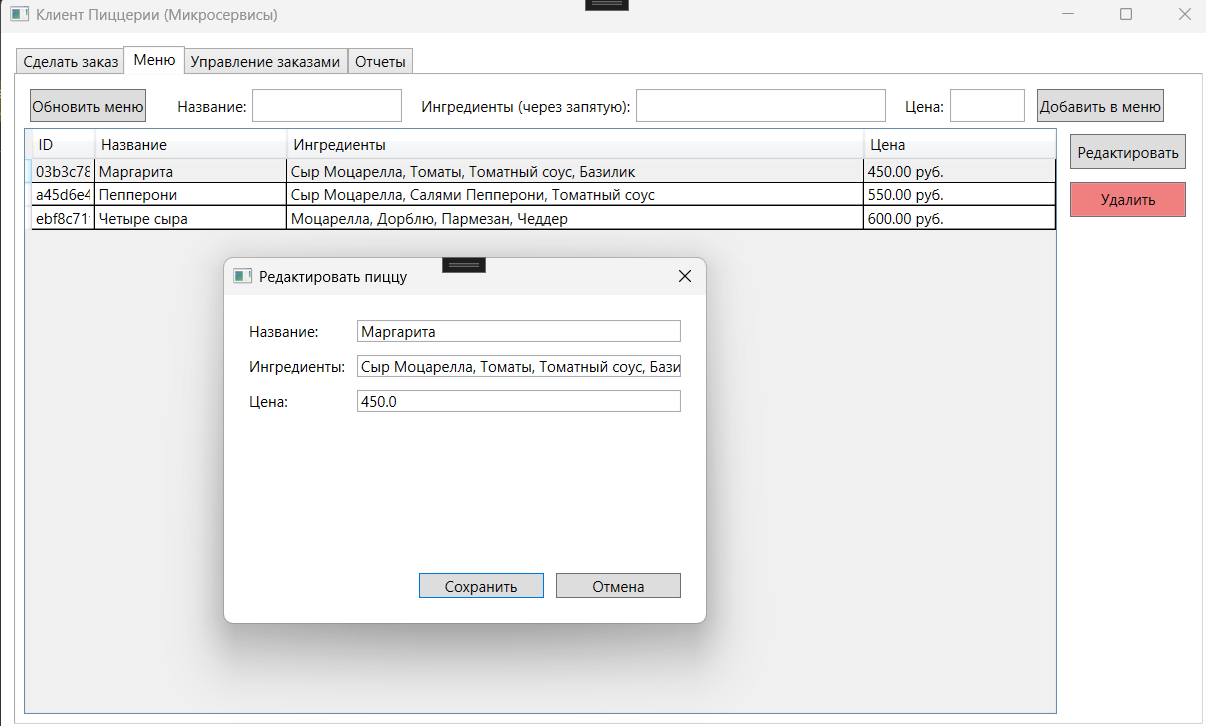


Рисунок 3 – Скриншот *UI* на вкладке «Меню» *WPF* приложения

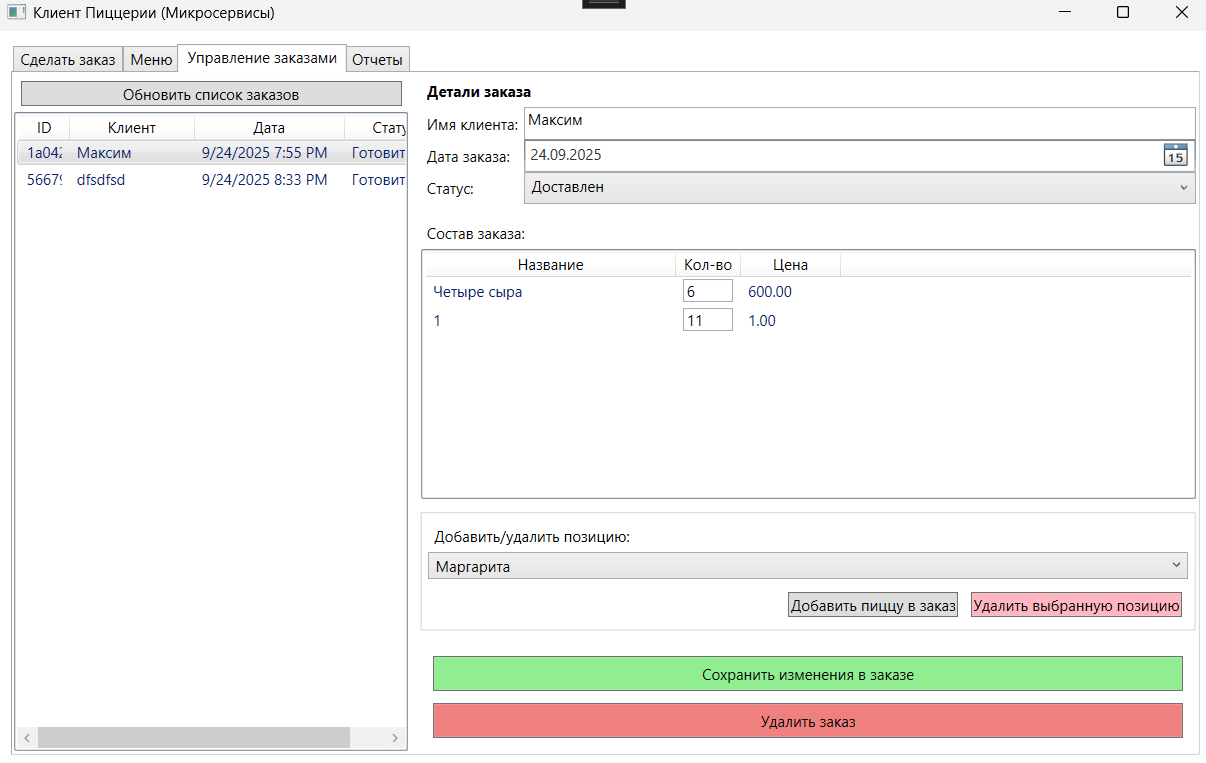


Рисунок 4 – Скриншот *UI* на вкладке «Управление заказами»   
*WPF* приложения

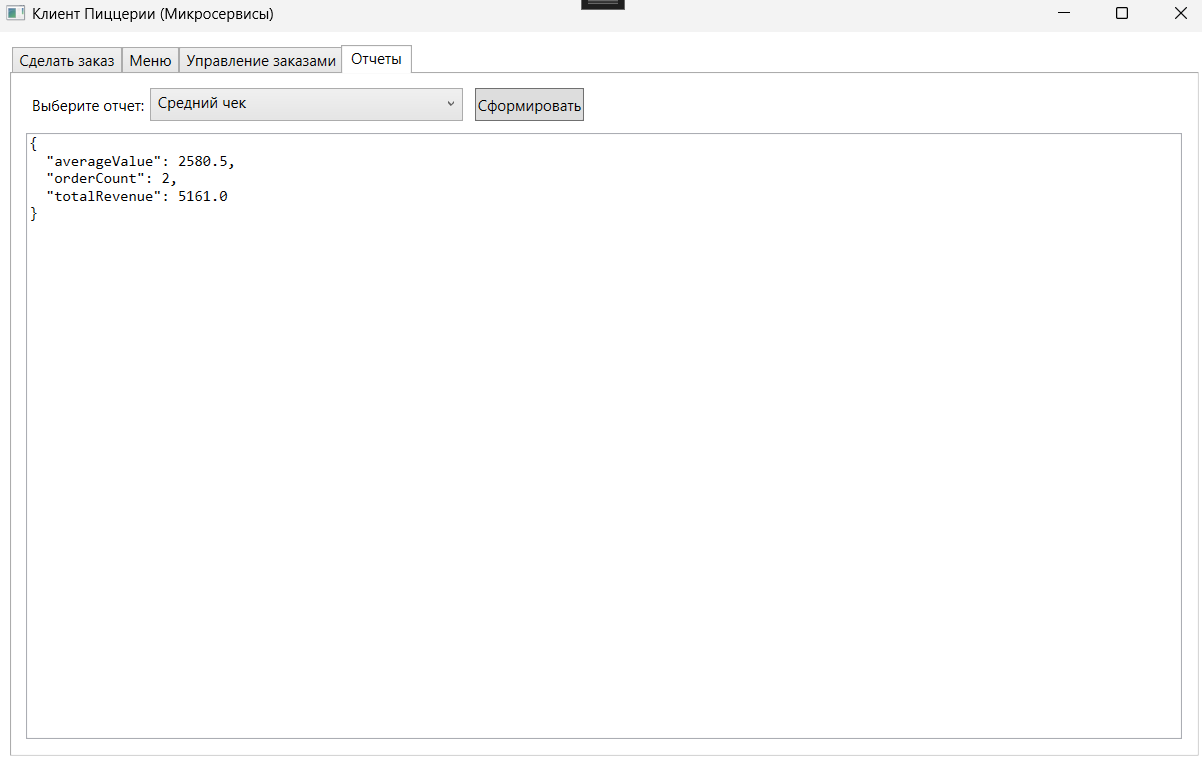


Рисунок 5 – Скриншот *UI* на вкладке «Отчеты» *WPF* приложения

В ходе работы была успешно создана распределенная система, полностью соответствующая требованиям задания. Реализованная микросервисная архитектура продемонстрировала свои преимущества: четкое разделение ответственности, изоляцию данных и возможность независимой работы над компонентами. Все *CRUD*-операции для трех сущностей функционируют корректно, а сервис отчетов предоставляет требуемые агрегатные вычисления на основе данных из других сервисов. Система стабильно работает, данные персистентно сохраняются в базах данных сервисов, а взаимодействие через *API* шлюз происходит прозрачно для конечного пользователя.

**Вывод:** в процессе выполнения лабораторной работы были освоены ключевые принципы и методы построения приложений на основе микросервисной архитектуры. Получен практический опыт декомпозиции сложной задачи на независимые сервисы, организации их сетевого взаимодействия с помощью *TCP*-сокетов и реализации единой точки входа через *API* шлюз. Созданное приложение является наглядным примером гибкой и масштабируемой системы, построенной по современным архитектурным стандартам.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

MainWindow.cs  
using System.Windows;

using System.Windows.Controls;

using PizzeriaTcpApp.Wpf.Models;

using TcpRestNetworking;

namespace PizzeriaTcpApp.Wpf;

public partial class MainWindow : Window

{

private readonly RestOverTcpClient \_client = new ("127.0.0.1", 8888);

private Pizza? \_selectedPizza;

public MainWindow()

{

InitializeComponent();

Loaded += MainWindow\_Loaded;

}

private async void MainWindow\_Loaded(object sender, RoutedEventArgs e)

{

await LoadPizzasAsync();

}

private void PizzasListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

\_selectedPizza = PizzasListBox.SelectedItem as Pizza;

PopulateDetails();

}

#region CRUD Operations

private async void AddButton\_Click(object sender, RoutedEventArgs e)

{

try

{

if (!ValidateInput()) return;

var newPizza = new Pizza();

MapFormToPizza(newPizza);

await \_client.PostAsync("/pizzas", newPizza);

MessageBox.Show("Новая пицца успешно добавлена!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

ClearForm();

await LoadPizzasAsync();

}

catch (Exception ex)

{

ShowError($"Ошибка при добавлении: {ex.Message}");

}

}

private async void UpdateButton\_Click(object sender, RoutedEventArgs e)

{

if (\_selectedPizza == null)

{

MessageBox.Show("Сначала выберите пиццу для обновления.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

try

{

if (!ValidateInput()) return;

MapFormToPizza(\_selectedPizza);

await \_client.PutAsync($"/pizzas/{\_selectedPizza.Id}", \_selectedPizza);

MessageBox.Show("Данные пиццы успешно обновлены!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

await LoadPizzasAsync();

}

catch (Exception ex)

{

ShowError($"Ошибка при обновлении: {ex.Message}");

}

}

private async void DeleteButton\_Click(object sender, RoutedEventArgs e)

{

if (\_selectedPizza == null)

{

MessageBox.Show("Сначала выберите пиццу для удаления.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

var result = MessageBox.Show($"Вы уверены, что хотите удалить '{\_selectedPizza.Name}'?",

"Подтверждение удаления", MessageBoxButton.YesNo, MessageBoxImage.Question);

if (result == MessageBoxResult.Yes)

{

try

{

await \_client.DeleteAsync($"/pizzas/{\_selectedPizza.Id}");

MessageBox.Show("Пицца успешно удалена.", "Успех");

await LoadPizzasAsync();

ClearForm();

}

catch (Exception ex)

{

ShowError($"Ошибка при удалении: {ex.Message}");

}

}

}

#endregion

#region Ingredient Management

private void AddIngredientButton\_Click(object sender, RoutedEventArgs e)

{

var newIngredient = NewIngredientTextBox.Text.Trim();

if (!string.IsNullOrEmpty(newIngredient))

{

var ingredients = (IngredientsListBox.ItemsSource as IEnumerable<string> ?? []).ToList();

ingredients.Add(newIngredient);

IngredientsListBox.ItemsSource = ingredients;

NewIngredientTextBox.Clear();

}

}

private void RemoveIngredientButton\_Click(object sender, RoutedEventArgs e)

{

if (IngredientsListBox.SelectedItem is string selectedIngredient)

{

var ingredients = (IngredientsListBox.ItemsSource as IEnumerable<string>).ToList();

ingredients.Remove(selectedIngredient);

IngredientsListBox.ItemsSource = ingredients;

}

}

#endregion

#region UI Helper Methods

private async Task LoadPizzasAsync()

{

try

{

var pizzas = await \_client.GetAsync<IEnumerable<Pizza>>("/pizzas");

var selectedId = \_selectedPizza?.Id;

PizzasListBox.ItemsSource = pizzas;

if (selectedId != null)

{

PizzasListBox.SelectedItem = pizzas?.FirstOrDefault(p => p.Id == selectedId);

}

}

catch (Exception ex)

{

ShowError($"Не удалось загрузить список пицц. Убедитесь, что сервер запущен.\n\nОшибка: {ex.Message}");

}

}

private void PopulateDetails()

{

if (\_selectedPizza != null)

{

DetailsPanel.IsEnabled = true;

IdTextBox.Text = \_selectedPizza.Id.ToString();

NameTextBox.Text = \_selectedPizza.Name;

PriceTextBox.Text = \_selectedPizza.Price.ToString("F2");

IngredientsListBox.ItemsSource = new List<string>(\_selectedPizza.Ingredients);

}

else

{

ClearForm();

}

}

private void MapFormToPizza(Pizza pizza)

{

pizza.Name = NameTextBox.Text;

pizza.Price = decimal.Parse(PriceTextBox.Text);

pizza.Ingredients = [.. (IngredientsListBox.ItemsSource as IEnumerable<string> ?? [])];

}

private void ClearForm()

{

\_selectedPizza = null;

PizzasListBox.SelectedItem = null;

IdTextBox.Clear();

NameTextBox.Clear();

PriceTextBox.Clear();

IngredientsListBox.ItemsSource = null;

NewIngredientTextBox.Clear();

DetailsPanel.IsEnabled = false;

}

private bool ValidateInput()

{

if (string.IsNullOrWhiteSpace(NameTextBox.Text))

{

MessageBox.Show("Поле 'Название' не может быть пустым.", "Ошибка валидации", MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

if (!decimal.TryParse(PriceTextBox.Text, out \_))

{

MessageBox.Show("Поле 'Цена' должно быть числом.", "Ошибка валидации", MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

return true;

}

private void ShowError(string message)

{

MessageBox.Show(message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

private async void RefreshButton\_Click(object sender, RoutedEventArgs e)

{

await LoadPizzasAsync();

}

private void ClearButton\_Click(object sender, RoutedEventArgs e)

{

PizzasListBox.SelectedItem = null;

DetailsPanel.IsEnabled = true;

IdTextBox.Text = "(новый)";

}

#endregion

}

MainWindow.xaml

<Window x:Class="PizzeriaTcpApp.Wpf.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:PizzeriaTcpApp.Wpf"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid Margin="15">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

</Grid.ColumnDefinitions>

<DockPanel Grid.Column="0">

<Label Content="Меню пиццерии" DockPanel.Dock="Top" FontWeight="Bold"/>

<Button Content="Обновить список" DockPanel.Dock="Bottom" Click="RefreshButton\_Click" Margin="0,10,0,0"/>

<ListBox x:Name="PizzasListBox"

SelectionChanged="PizzasListBox\_SelectionChanged"

DisplayMemberPath="Name"/>

</DockPanel>

<Border Grid.Column="1" Margin="20,0,0,0" BorderBrush="Gainsboro" BorderThickness="1" Padding="10">

<StackPanel x:Name="DetailsPanel" IsEnabled="False">

<Label Content="Детали пиццы" FontWeight="Bold"/>

<Label Content="ID"/>

<TextBox x:Name="IdTextBox" IsReadOnly="True" Background="#eee"/>

<Label Content="Название"/>

<TextBox x:Name="NameTextBox"/>

<Label Content="Ингредиенты"/>

<ListBox x:Name="IngredientsListBox" Height="100" Margin="0,0,0,5"/>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<TextBox x:Name="NewIngredientTextBox" Grid.Column="0" VerticalContentAlignment="Center"/>

<Button x:Name="AddIngredientButton" Grid.Column="1" Content="Добавить" Margin="5,0,0,0" Click="AddIngredientButton\_Click"/>

<Button x:Name="RemoveIngredientButton" Grid.Column="2" Content="Удалить" Margin="5,0,0,0" Click="RemoveIngredientButton\_Click"/>

</Grid>

<Label Content="Цена"/>

<TextBox x:Name="PriceTextBox"/>

<WrapPanel Margin="0,25,0,0" HorizontalAlignment="Right">

<Button Content="Добавить новую" Click="AddButton\_Click" Width="120" Margin="5"/>

<Button Content="Сохранить изменения" Click="UpdateButton\_Click" Width="140" Margin="5"/>

<Button Content="Удалить" Click="DeleteButton\_Click" Width="80" Margin="5" Background="#FFE08282"/>

<Button Content="Очистить" Click="ClearButton\_Click" Width="80" Margin="5"/>

</WrapPanel>

</StackPanel>

</Border>

</Grid>

</Window>

RestOverTcpClient.cs

using System.Net.Sockets;

using System.Text;

using System.Text.Json;

using TcpRestNetworking.Exceptions;

namespace TcpRestNetworking;

public class RestOverTcpClient

{

private readonly string \_host;

private readonly int \_port;

private readonly JsonSerializerOptions \_jsonOptions;

public RestOverTcpClient(string host, int port)

{

\_host = host;

\_port = port;

\_jsonOptions = new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true

};

}

public Task<T?> GetAsync<T>(string path)

{

return SendRequestAsync<T>("GET", path);

}

public Task<T?> PostAsync<T>(string path, T body)

{

return SendRequestAsync<T>("POST", path, body);

}

public Task<T?> PutAsync<T>(string path, T body)

{

return SendRequestAsync<T>("PUT", path, body);

}

public Task DeleteAsync(string path)

{

return SendRequestAsync<object>("DELETE", path);

}

private async Task<T?> SendRequestAsync<T>(string method, string path, object? bodyContent = null)

{

var requestBuilder = new StringBuilder();

requestBuilder.AppendLine($"{method} {path}");

byte[]? bodyBytes = null;

if (bodyContent != null)

{

var jsonBody = JsonSerializer.Serialize(bodyContent);

bodyBytes = Encoding.UTF8.GetBytes(jsonBody);

requestBuilder.AppendLine("Content-Type: application/json");

requestBuilder.AppendLine($"Content-Length: {jsonBody.Length}");

}

requestBuilder.AppendLine();

using var client = new TcpClient();

await client.ConnectAsync(\_host, \_port);

await using var stream = client.GetStream();

var headerBytes = Encoding.UTF8.GetBytes(requestBuilder.ToString());

await stream.WriteAsync(headerBytes);

if (bodyBytes != null)

{

await stream.WriteAsync(bodyBytes);

}

using var reader = new StreamReader(stream, Encoding.UTF8);

var rawResponse = await reader.ReadToEndAsync();

return ParseResponse<T>(rawResponse);

}

private T? ParseResponse<T>(string rawResponse)

{

if (string.IsNullOrWhiteSpace(rawResponse))

{

throw new ApiException("Response from server is empty.");

}

var separator = new[] { "\r\n\r\n" };

var parts = rawResponse.Split(separator, 2, System.StringSplitOptions.None);

var headersPart = parts[0];

var bodyPart = parts.Length > 1 ? parts[1] : string.Empty;

var statusLine = headersPart.Split('\r', '\n')[0];

if (statusLine.StartsWith("200") || statusLine.StartsWith("201"))

{

if (string.IsNullOrWhiteSpace(bodyPart))

{

throw new ApiException($"Server returned succesfull status '{statusLine}', but body of response is empty.");

}

return JsonSerializer.Deserialize<T>(bodyPart, \_jsonOptions);

}

if (statusLine.StartsWith("204"))

{

return default;

}

throw new ApiException($"Server returned error: {statusLine}");

}

}

ServerHelper.cs

using System.Net.Sockets;

using System.Text;

using System.Text.Json;

using PizzeriaTcpApp.PizzaService.Application.Services.Interfaces;

using PizzeriaTcpApp.PizzaService.Application.Services.Implementations;

using PizzeriaTcpApp.PizzaService.Domain.Models;

namespace PizzeriaTcpApp.PizzaService.Presentation.Helpers;

public class ServerHelper

{

private static readonly IPizzasService \_pizzaService = new PizzasService();

public static void HandleClient(TcpClient client)

{

Console.WriteLine("New connection!");

using var stream = client.GetStream();

using var reader = new StreamReader(stream, Encoding.UTF8);

using var writer = new StreamWriter(stream, Encoding.UTF8) { AutoFlush = true };

try

{

// 1. Читаем стартовую строку

var requestLine = reader.ReadLine();

if (string.IsNullOrEmpty(requestLine)) return;

Console.WriteLine(requestLine);

var parts = requestLine.Split(' ');

var method = parts[0];

var path = parts[1];

// 2. Читаем все заголовки

var headers = new Dictionary<string, string>();

string line;

while (!string.IsNullOrEmpty(line = reader.ReadLine()))

{

var headerParts = line.Split(':', 2);

if (headerParts.Length == 2)

{

headers[headerParts[0].Trim()] = headerParts[1].Trim();

}

}

// 3. Читаем тело, если есть Content-Length

string body = null;

if (headers.TryGetValue("Content-Length", out var contentLengthValue))

{

if (int.TryParse(contentLengthValue, out var contentLength))

{

var buffer = new char[contentLength];

reader.ReadBlock(buffer, 0, contentLength);

body = new string(buffer);

}

}

string response;

if (method == "GET" && path == "/pizzas")

{

var allPizzas = \_pizzaService.GetAllPizzas();

response = CreateResponse("200 OK", allPizzas);

}

else if (method == "GET" && path.StartsWith("/pizzas/"))

{

var id = Guid.Parse(path.Split('/')[2]);

var pizza = \_pizzaService.GetPizzaById(id);

response = pizza != null ? CreateResponse("200 OK", pizza) : CreateResponse("404 Not Found");

}

else if (method == "POST" && path == "/pizzas")

{

var pizzaDto = JsonSerializer.Deserialize<Pizza>(body);

var createdPizza = \_pizzaService.CreatePizza(pizzaDto);

Console.WriteLine($"Добавлена пицца: {createdPizza.Name}");

response = CreateResponse("201 Created", createdPizza);

}

else if (method == "PUT" && path.StartsWith("/pizzas/"))

{

var id = Guid.Parse(path.Split('/')[2]);

var pizzaDto = JsonSerializer.Deserialize<Pizza>(body);

var updatedPizza = \_pizzaService.UpdatePizza(id, pizzaDto);

if (updatedPizza != null)

{

Console.WriteLine($"Обновлена пицца ID: {id}");

response = CreateResponse("200 OK", updatedPizza);

}

else

{

response = CreateResponse("404 Not Found");

}

}

else if (method == "DELETE" && path.StartsWith("/pizzas/"))

{

var id = Guid.Parse(path.Split('/')[2]);

var success = \_pizzaService.DeletePizza(id);

if (success)

{

Console.WriteLine($"Удалена пицца ID: {id}");

response = CreateResponse("204 No Content");

}

else

{

response = CreateResponse("404 Not Found");

}

}

else

{

response = CreateResponse("400 Bad Request");

}

writer.Write(response);

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

var errorResponse = CreateResponse("500 Internal Server Error");

writer.Write(errorResponse);

}

finally

{

client.Close();

Console.WriteLine("Соединение закрыто.");

}

}

public static string CreateResponse<T>(string status, T body)

{

var jsonBody = JsonSerializer.Serialize(body);

return $"{status}\r\nContent-Type: application/json\r\n\r\n{jsonBody}";

}

public static string CreateResponse(string status)

{

return $"{status}\r\n\r\n";

}

}

PizzasService.cs

using Microsoft.EntityFrameworkCore;

using PizzeriaTcpApp.PizzaService.Application.Services.Interfaces;

using PizzeriaTcpApp.PizzaService.Domain.Models;

using PizzeriaTcpApp.PizzaService.Infrastructure.Persistence.Common;

namespace PizzeriaTcpApp.PizzaService.Application.Services.Implementations;

public class PizzasService : IPizzasService

{

private readonly List<Pizza> \_pizzas = [];

public PizzasService()

{

using var context = new AppDbContext();

context.Database.Migrate();

if (!context.Pizzas.Any())

{

context.Pizzas.AddRange(

new Pizza { Name = "Маргарита", Ingredients = ["Сыр Моцарелла", "Томаты", "Томатный соус", "Базилик"], Price = 450 },

new Pizza { Name = "Четыре сыра", Ingredients = ["Моцарелла", "Дорблю", "Пармезан", "Чеддер"], Price = 600 },

new Pizza { Name = "Пепперони", Ingredients = ["Сыр Моцарелла", "Салями Пепперони", "Томатный соус"], Price = 550 }

);

context.SaveChanges();

}

}

public Pizza? CreatePizza(Pizza? newPizza)

{

ArgumentNullException.ThrowIfNull(newPizza);

newPizza.Id = Guid.NewGuid();

using var context = new AppDbContext();

context.Pizzas.Add(newPizza);

context.SaveChanges();

return newPizza;

}

public bool DeletePizza(Guid id)

{

using var context = new AppDbContext();

var pizzaToRemove = context.Pizzas.FirstOrDefault(p => p.Id == id);

if (pizzaToRemove == null)

{

return false;

}

context.Pizzas.Remove(pizzaToRemove);

context.SaveChanges();

return true;

}

public IEnumerable<Pizza> GetAllPizzas()

{

using var context = new AppDbContext();

return [.. context.Pizzas.AsNoTracking()];

}

public Pizza? GetPizzaById(Guid id)

{

using var context = new AppDbContext();

return context.Pizzas.AsNoTracking().FirstOrDefault(p => p.Id == id);

}

public Pizza? UpdatePizza(Guid id, Pizza? updatedPizza)

{

using var context = new AppDbContext();

var existingPizza = context.Pizzas.FirstOrDefault(p => p.Id == id);

if (existingPizza == null)

{

return null;

}

existingPizza.Name = updatedPizza?.Name ?? "";

existingPizza.Ingredients = updatedPizza?.Ingredients ?? [];

existingPizza.Price = updatedPizza?.Price ?? 0;

context.SaveChanges();

return existingPizza;

}

}