Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 2

по курсу «Программные технологии проектирования ПО ВС»

на тему «Продолжение разработки Web API. Подключение БД, реализация методов»

Вариант 2

Выполнили

студенты группы 22ВВП1:

Зубриянова А.А.

Кондратьева В.И.

Приняли:

Патунин Д.В.

Деев М.В.

Пенза 2025

**Цель работы**

Разработать класс ProductService с интерфейсом IProductService, подключить текстовый файл в качестве БД, реализовать методы класса ProductService, используя мьютексы для записи изменений.

**Лабораторное задание**

1. С помощью внедрения зависимостей связать разработанный класс ProductService с интерфейсом IProductService (builder.Service.AddSingleton).
2. В конструктор контроллера ProductController передать интерфейс IProductService и присвоить его внутренней переменной.
3. Усовершенствовать класс ProductService. Конструктор класса принимает на вход параметр IConfiguration. Присвоить его внутренней переменной. Конфигурация по умолчанию считывает из файла appsettings.json. Внутри файла объявить поле "DataBaseFilePath" : "database.txt".
4. Внутри ProductService объявить приватный метод InitFromFile, который из файла читает сериализованный список продуктов. Путь до файла запрашивать из конфигурации.
5. Десериализовать считанный файл в словарь ключ – значение. Данный словарь будет служить копией базы данных в памяти.
6. Явно реализовать методы класса ProductService. Все методы связанные с изменением состояния базы данных должны записывать изменения как в оперативную память, так и на диск. Использовать мьютекс для записи изменений.
7. Для сохранения изменений на диск реализовать приватный метод SaveToFile внутри класса ProductService.

**Ход работы**

1. С помощью внедрения зависимостей связали разработанный класс ProductService с интерфейсом IProductService (builder.Service.AddSingleton).

builder.Services.AddSingleton<IProductService, ProductService>();

1. В конструктор контроллера ProductController передали интерфейс IProductService и присвоили его внутренней переменной.

public class ProductsController : ControllerBase

{

private IProductService ProductServ;

public ProductsController(IProductService newProductServ)

{

ProductServ = newProductServ;

}

1. Усовершенствовали класс ProductService. Конструктор класса принимает на вход параметр IConfiguration. Присвоили его внутренней переменной. Конфигурация по умолчанию считывает из файла appsettings.json. Внутри файла объявили поле "DataBaseFilePath".

private readonly IConfiguration \_config;

private readonly string \_databaseFilePath;

public ProductService(IConfiguration config)

{

\_config = config;

\_products = new Dictionary<Guid, Product>();

\_databaseFilePath = \_config["DataBaseFilePath"];

Console.WriteLine($"Путь к базе данных: {\_databaseFilePath}");

InitFromFile(); // Загружаем данные из файла

}

файл appsettings.json:

{

"DataBaseFilePath": "d:\\Уник\\3 курс\\5 семестр\\net\\лабы\\.NET\\database.txt",

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

1. Внутри ProductService объявили приватный метод InitFromFile, который из файла читает сериализованный список продуктов. Путь до файла запрашиваем из конфигурации. Десериализовали считанный файл в словарь ключ – значение. Данный словарь будет служить копией базы данных в памяти.

private readonly Dictionary<Guid, Product> \_products;

private void InitFromFile()

{

if (!File.Exists(\_databaseFilePath))

{

Console.WriteLine("Файл базы данных не найден");

return;

}

string json = File.ReadAllText(\_databaseFilePath);

if (string.IsNullOrWhiteSpace(json))

{

Console.WriteLine("Файл базы данных пуст, инициализируем его.");

File.WriteAllText(\_databaseFilePath, "{}");

json = "{}";

}

var loadedProducts = JsonSerializer.Deserialize<Dictionary<Guid, Product>>(json);

if (loadedProducts != null)

{

\_products.Clear();

foreach (var i in loadedProducts)

{

\_products[i.Key] = i.Value;

}

}

Console.WriteLine($"Загружено {\_products.Count} продуктов из файла.");

}

1. Явно реализовали методы класса ProductService. Все методы связанные с изменением состояния базы данных записывают изменения как в оперативную память, так и на диск. Использовали мьютекс для записи изменений.

public Product Add(string defin, string name, double price, string image)

{

\_mutex.WaitOne();

Console.WriteLine("using add");

if (string.IsNullOrEmpty(defin) || string.IsNullOrEmpty(name) ||

price < 0 || string.IsNullOrEmpty(image))

{

return null;

}

var product = new Product()

{

Id = Guid.NewGuid(),

Definition = defin,

Name = name,

Price = price,

Image = image

};

if (\_products.TryAdd(product.Id, product))

{

SaveToFile();

Console.WriteLine("GOOD add");

Console.WriteLine("id продукта: " + product.Id);

Console.WriteLine("Добавление: " + product.Definition + " " + product.Name + " " + product.Price + " " + product.Image);

\_mutex.ReleaseMutex();

return product;

}

Console.WriteLine("BAD add");

\_mutex.ReleaseMutex();

return null;

}

public void Remove(Guid id)

{

\_mutex.WaitOne();

Console.WriteLine("using remove");

if (\_products.ContainsKey(id))

{

var product = \_products[id];

if (\_products.Remove(id))

{

SaveToFile();

Console.WriteLine("GOOD remove");

\_mutex.ReleaseMutex();

//return product;

return;

}

}

Console.WriteLine("BAD remove");

\_mutex.ReleaseMutex();

//return;

//return product;

}

public Product Edit(Guid id, string defin, string name, double price, string image)

{

\_mutex.WaitOne();

Console.WriteLine("using edit");

if (\_products.ContainsKey(id))

{

var product = \_products[id];

if (defin != "-")

{

product.Definition = defin;

}

if (name != "-")

{

product.Name = name;

}

if (price > 0)

{

product.Price = price;

}

if (image != "-")

{

product.Image = image;

}

//(!string.IsNullOrEmpty(image))

SaveToFile();

Console.WriteLine("GOOD edit");

Console.WriteLine("Редактирование: ");

Console.WriteLine(product.Definition);

Console.WriteLine(product.Name);

Console.WriteLine(product.Price);

Console.WriteLine(product.Image);

\_mutex.ReleaseMutex();

return product;

}

Console.WriteLine("BAD edit");

\_mutex.ReleaseMutex();

return null;

}

public Product Search(Guid id)

{

\_mutex.WaitOne();

Console.WriteLine("using search");

foreach (var product in \_products.Values)

{

if (product.Id == id)

{

Console.WriteLine("GOOD search");

Console.WriteLine("Поиск: " + product.Definition + " " + product.Name + " " + product.Price + " " + product.Image);

\_mutex.ReleaseMutex();

return product;

}

}

Console.WriteLine("BAD search");

\_mutex.ReleaseMutex();

return null;

}

1. Для сохранения изменений на диск реализовать приватный метод SaveToFile внутри класса ProductService.

private void SaveToFile()

{

string json = JsonSerializer.Serialize(\_products, new JsonSerializerOptions { WriteIndented = true });

File.WriteAllText(\_databaseFilePath, json);

Console.WriteLine("База данных обновлена.");

}

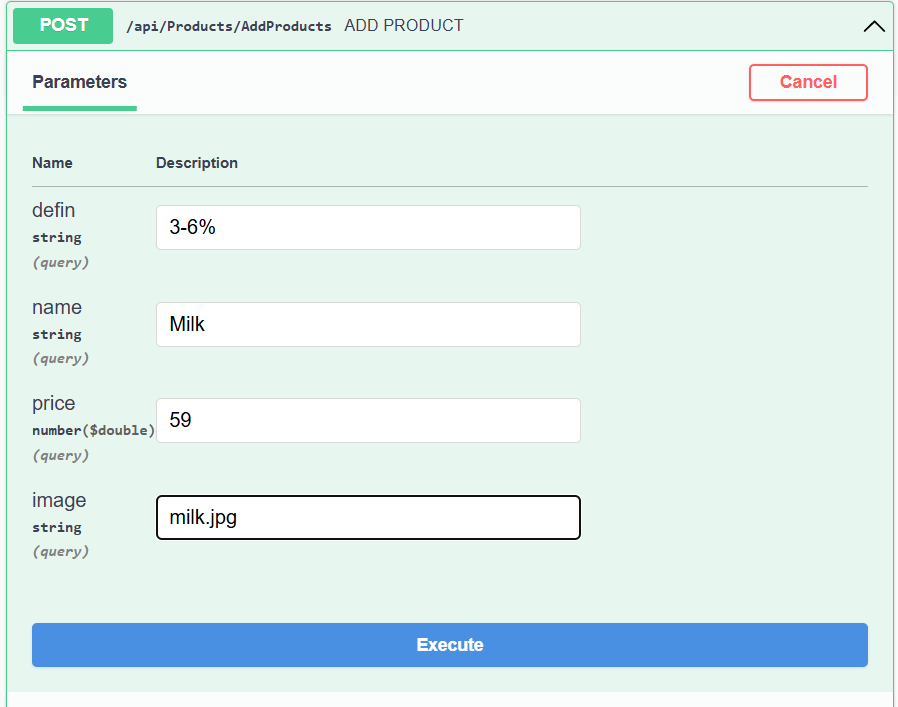
****

Рисунок 1 – Заполнили данные для добавления продукта

Выводим в консоль для проверки информацию о всех совершенных действиях:

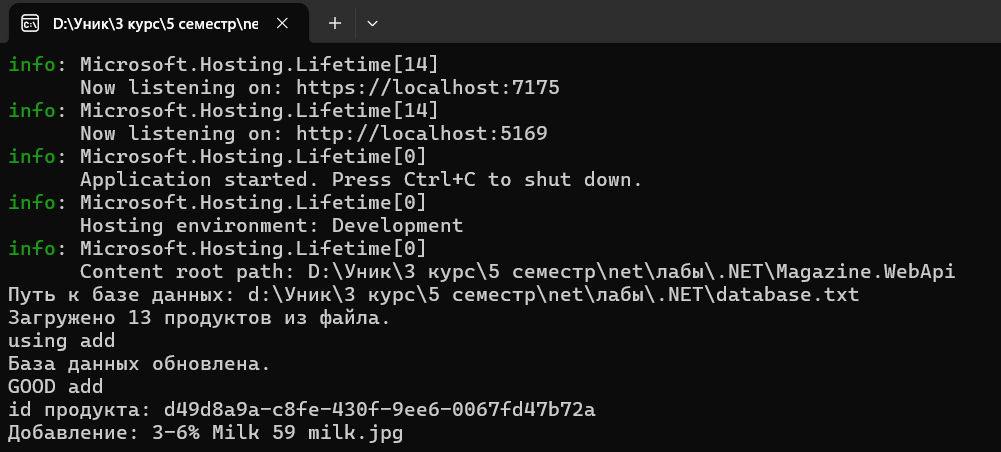
****

Рисунок 2 – Продукт успешно создался

Просмотрели изменения в БД. Продукт успешно добавлен:

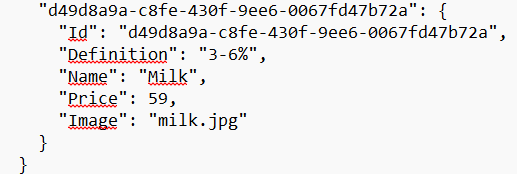
****

Рисунок 3 – database.txt

**Вывод**

В ходе выполнения лабораторной работы была продолжена разработка Web API. Был разработан класс ProductService с интерфейсом IProductService, подключен текстовый файл в качестве БД, явно реализованы методы класса ProductService, используя мьютексы для записи изменений. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок.