Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»
Институт новых материалов и технологий
Кафедра «Теплофизика и информатика в металлургии»

Управление контейнерами с помощью Docker Compose и использование готовых образов

ОТЧЕТ
по практической работе № 2
по дисциплине «Основы методологии Development Operation»

Вариант №9

Направление 09.03.02 «Информационные системы и технологии» (уровень бакалавриата)
Образовательная программа
09.03.02/33.02 «Информационные системы и технологии» (СУОС)

Студент
группы НМТ-413901 Я.В.Крашенинников

Преподаватель:
профессор, д.т.н. В.В.Лавров

Екатеринбург
2024

Оглавление

Практическая работа «Управление контейнерами с помощью Docker Compose и использование готовых образов»

1.1 Цель работы

Разработать тестовое приложение «Калькулятор» на платформе .NET Core, включающее функционал для хранения всех выполненных операций в базе данных MariaDB и настроить взаимодействие с этой базой данных через Entity Framework. В процессе требуется создать и настроить многоконтейнерное приложение с использованием Docker для упрощения развёртывания и обеспечения независимости окружения. Итоговый проект должен быть размещён в системе контроля версий GitHub, база данных — доступна для администрирования с помощью программы HeidiSQL.

1.2 Ход проведения работы

Для начала установил все необходимые пакеты с указанными версиями в методичке:

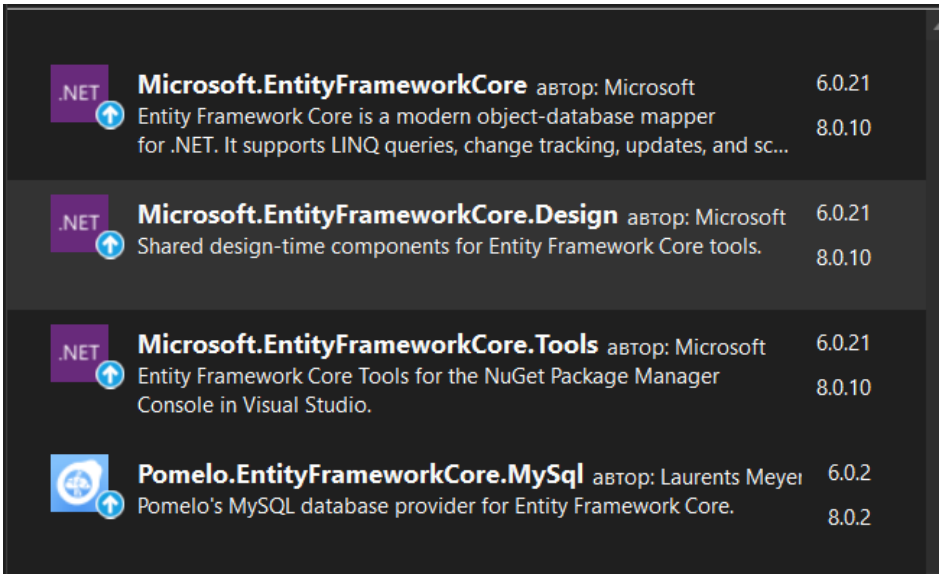

Рисунок 1 – Установка пакетов NuGet

1.2.1 Разработка тестового приложения Visual Studio .NET Core

Для начала я добавил все недостающие части кода, создал папку Data, где описал все переменные и настроил подключение к БД:

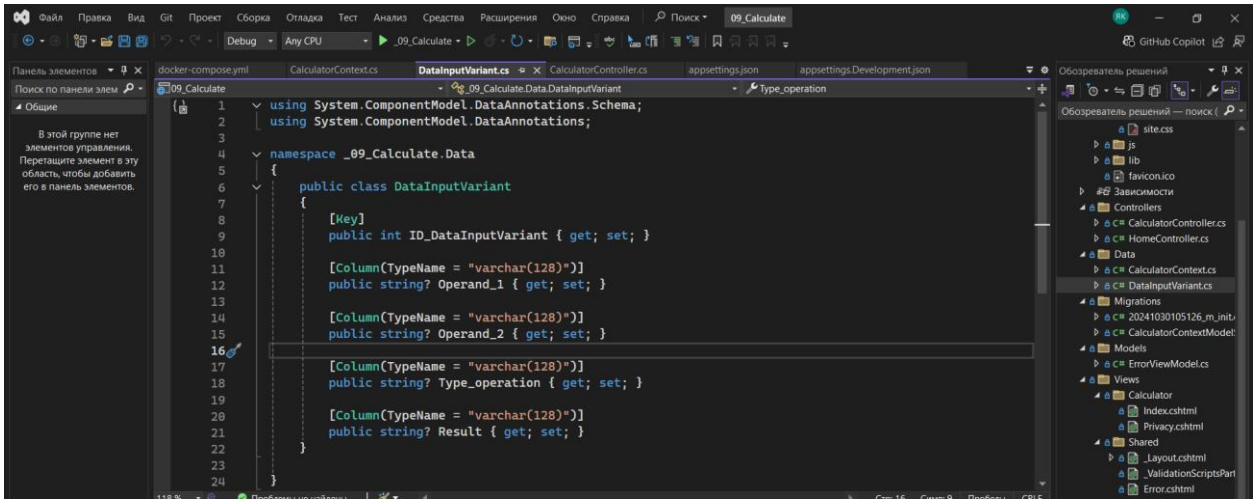

Рисунок 2 – созданная папка Data

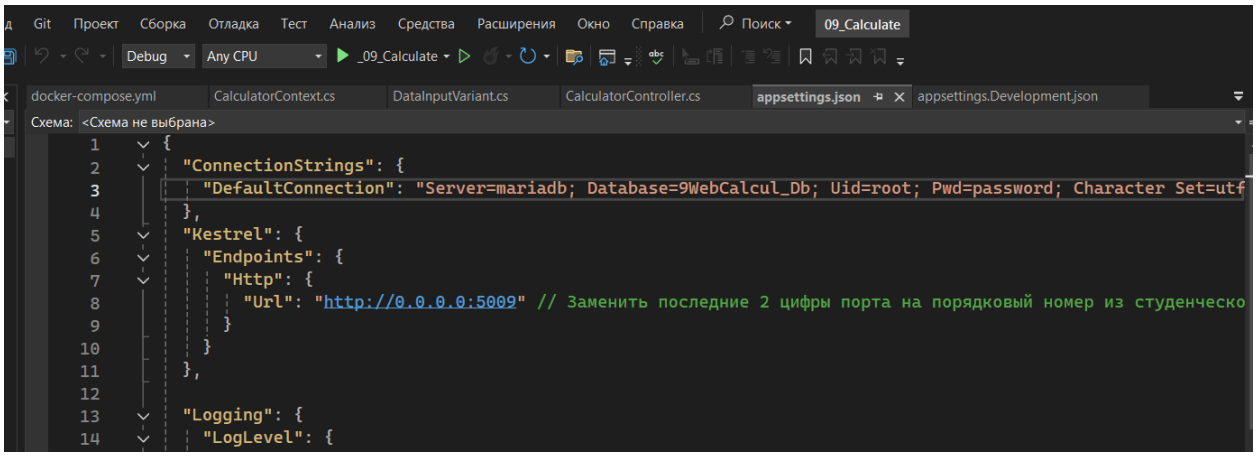

Рисунок 3 – Описание ConnectionStrings в appsetting.json

На этом моменте главным было правильно указать названия в DefaultConnection.

Также необходимо настроить appsetting.Devol-nt.json, чтобы успешно подключиться на нужном порте и IP к БД:

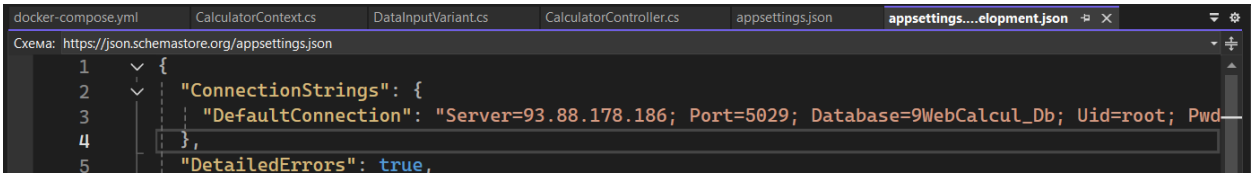

Рисунок 4 - Описание ConnectionStrings в appsetting.Devolopment

1.2.2 Создание конфигурационного файла docker-compose.yml

```
version: '3.7'

services:
  mariadb:
    container_name: web-mariadb_09
    hostname: mariadb
    image: mariadb:10.5
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: password
      MARIADB_DATABASE: 9WebCalcul_Db    # Имя базы данных, указанное в настройках подключения
      MARIADB_USER: root                 # Имя пользователя, указанное в настройках подключения
      MARIADB_PASSWORD: password         # Пароль, указанный в настройках подключения
    ports:
      - "5029:3306"                      # Используем стандартный порт MariaDB 3306

  web:
    container_name: web-app-09calculate
    hostname: app
    build: ./
    ports:
      - "5009:5009"                      # Порт, использующийся для веб-приложения
    depends_on:
      - mariadb
```

Тут:

mariadb: Создаёт контейнер для базы данных MariaDB версии 10.5 с именем web-mariadb_09. Указываются параметры подключения: имя базы данных, пользователя и пароль, а также прокидывается порт 5029 на хосте к стандартному порту 3306 в контейнере.

web: Создает контейнер для веб-приложений с именем web-app-09calculate, компилируя приложение из текущего каталога (./). Портируется 5009 в контейнер и перекидывается на 5009 хост. Контейнер зависит от данных базы данных mariadb, поэтому база данных запускается при запуске веб-приложений.

1.2.3 Размещение приложения в системе контроля версий GitHub

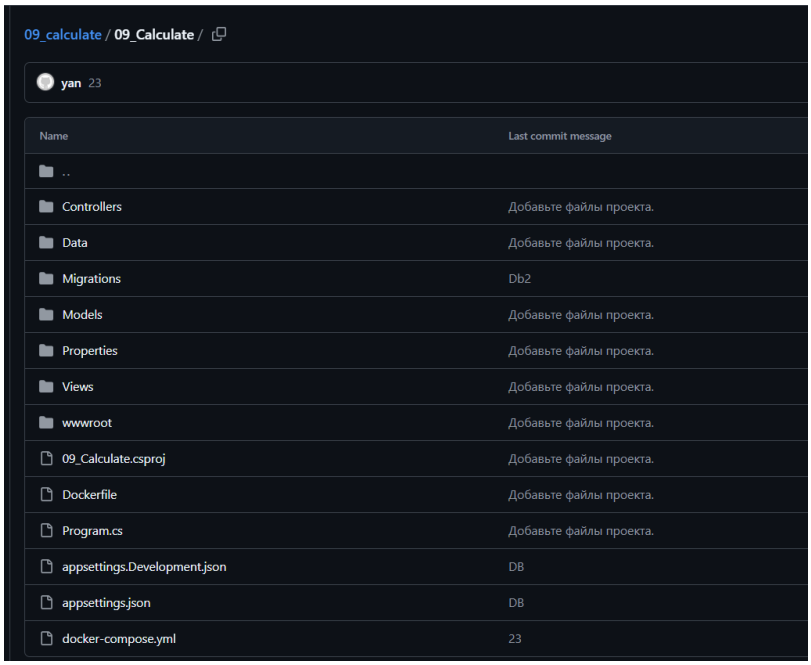

Рисунок 5 – Размещенный файл на GitHub в директории проекта

Ссылка - https://github.com/krosshiik/09_calculate.git

1.2.4 Сборка образов с использованием команды docker compose

В процессе выполнения я столкнулся с проблемой – при выполнении docker compose у меня дублировались контейнеры на :5009 порте, в следствии чего калькулятор запускался, но не было связи с БД. Для удаления контейнера из первой практической нужно было ввести «*docker ps -a*», скопировать ID старого контейнера, остановить его «*docker stop "ID"*» и удалить «*docker rm "ID"*».

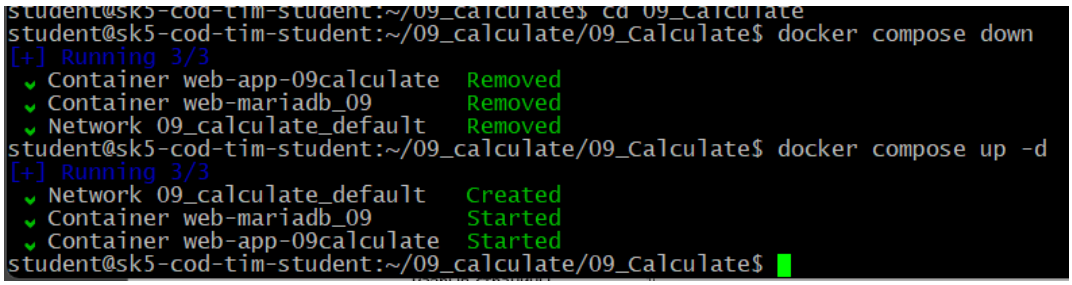

Рисунок 6 – docker compose up -d, запускаются все контейнеры

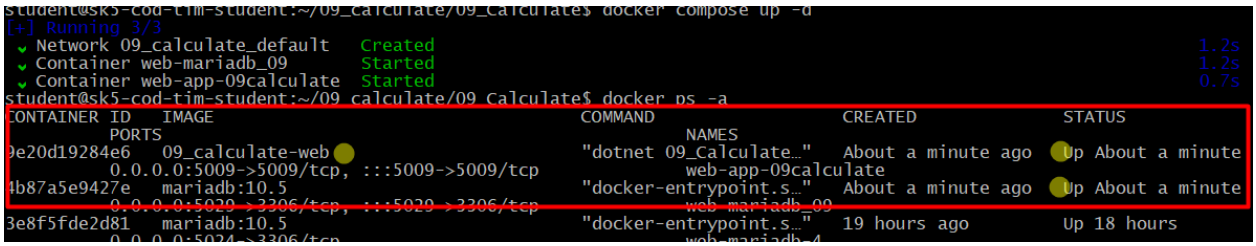

Рисунок 7 – Если всё в статусе UP, значит будет работать

После запуска/перезапуска compose требуется обновить БД в visual studio.

1.2.5 Установка программы HeidiSQL для сопровождения базы данных в СУБД MariaDB

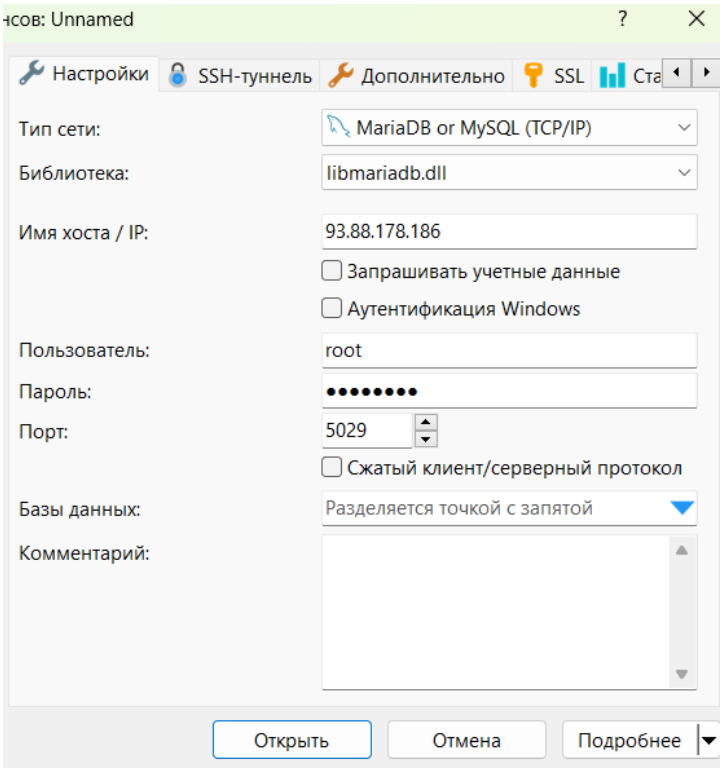

Рисунок 8 – Подключение к БД по указанным параметрам

HeidiSQL была установлена стандартно, при установке пользователь – root, пароль – password. Порт указан 5029 (номер варианта+20), IP сервера, эти данные взяты из кофигурации *appsetting*

1.2.6 Настройка миграции базы данных в MariaDB

Вызвал консоль диспетчера пакетов и выполнил команду: *add-migration m_init*

В проекте создана папка Migrations с двумя файлами.

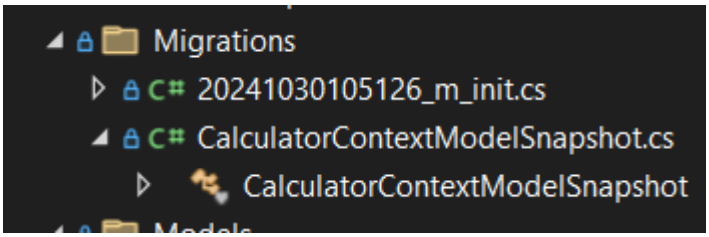

Рисунок 9 – Миграция БД

Далее выполняю обновление базы данных с помощью команды: *update-database*. Обновлять БД нужно каждый раз перед новой сессией (как я понял)

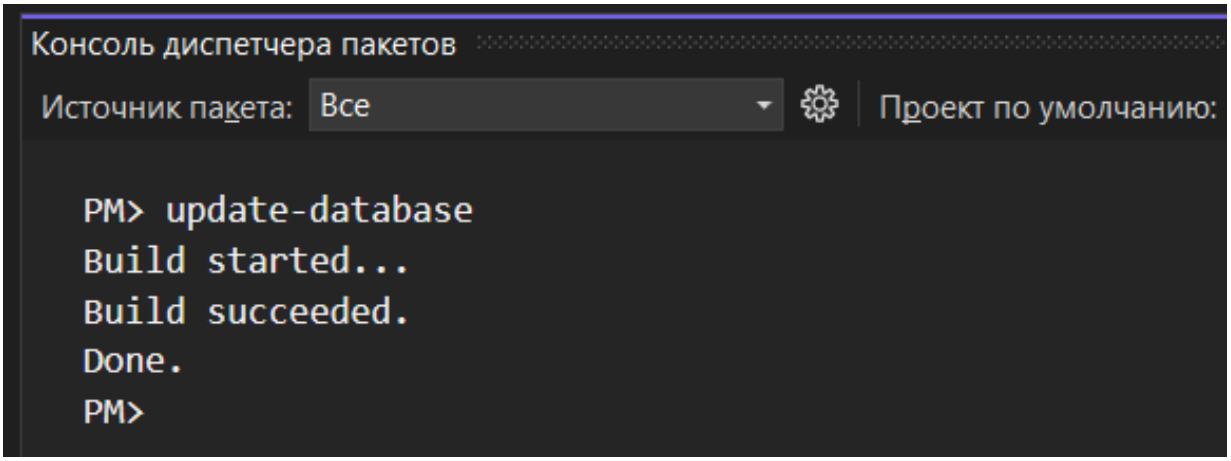

Рисунок 10 – Done означает что у нас все поучилось и данные будут улетать в БД

1.2.7 Демонстрация работы приложения с базой данных в СУБД MariaDB

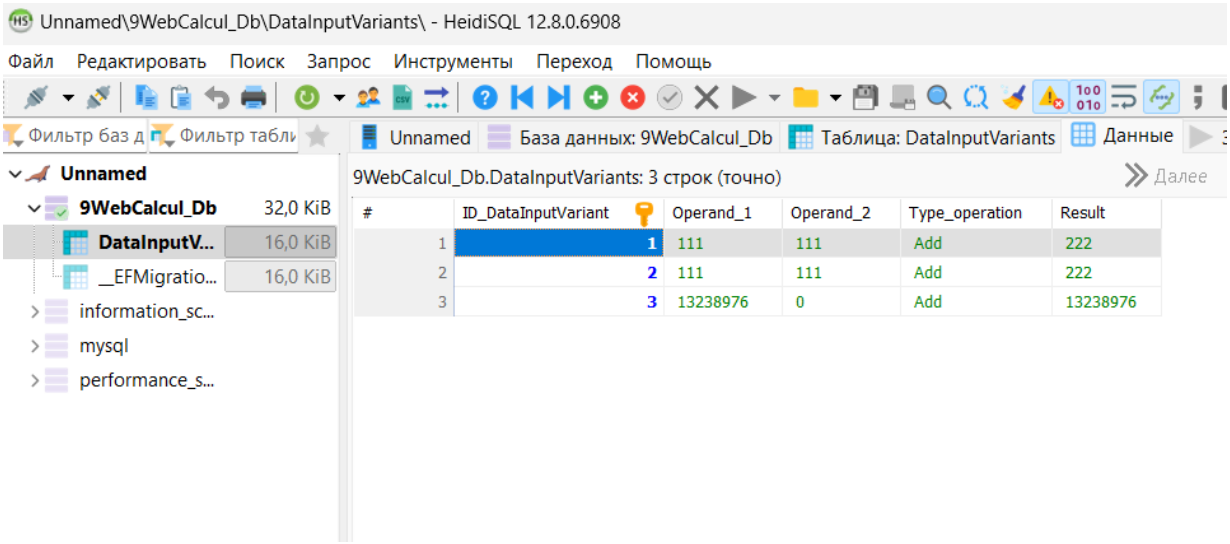

Рисунок 11 – Моя БД с сохраненным данными на сервере и локалхосте

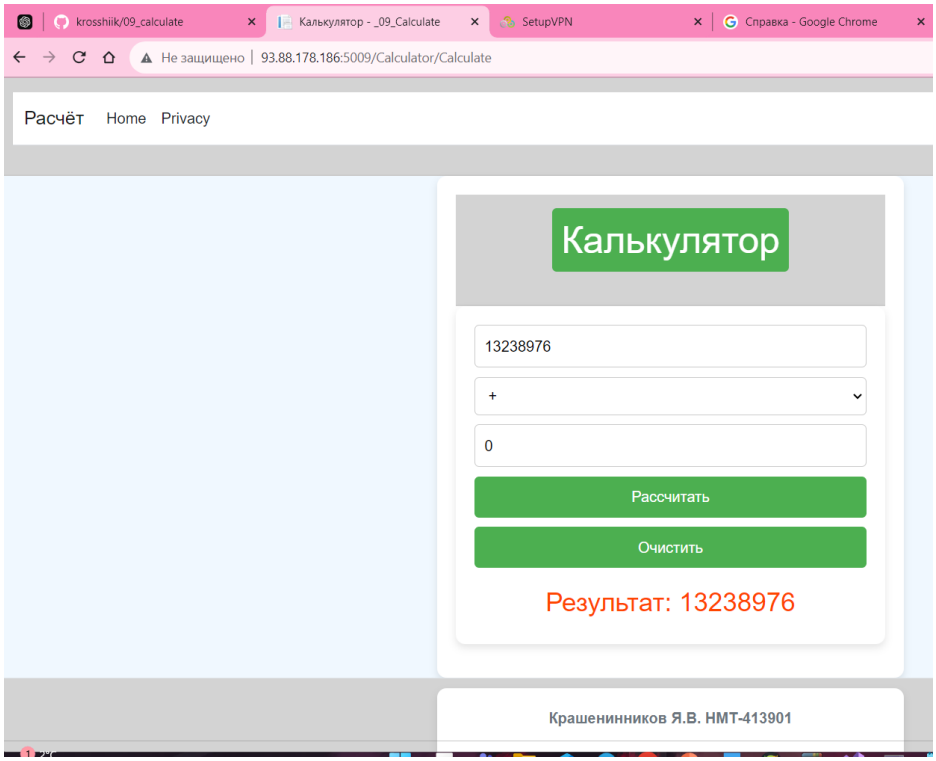

Рисунок 12 – Расчет на сервере

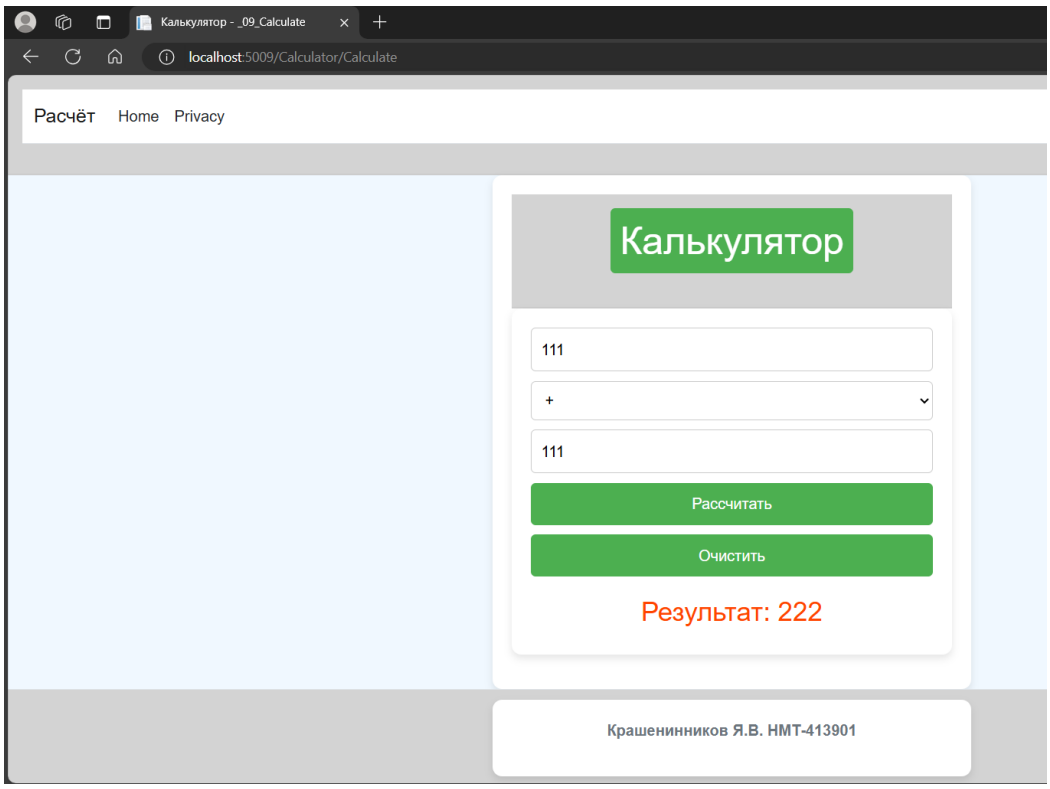

Рисунок 13 – Расчёт на localhost

1.3 Выводы

В ходе практической работы был разработан тестовый проект «Калькулятор», расширенный функционал для учета всех операций в объектах данных MariaDB. С помощью Entity Framework были созданы приложения и успешно реализована настройка взаимодействия приложения с базой данных, что обеспечивает автоматическое сохранение вычислений. Настройка многоконтейнерного окружения через Docker упростила развертывание и настройку компонентов изоляции, использование GitHub упростило контрольную версию и резервное хранение проекта. Итоговая проверка подтверждает корректность работы приложения и базы данных, а также успешную интеграцию всех необходимых сервисов.

Приложение А Листинг программного кода приложения «Калькулятор» (контроллер)

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using _09_Calculate.Data;
namespace _09_Calculate.Controllers
{
    public enum Operation { Add, Subtract, Multiply, Divide }
    public class CalculatorController : Controller
    {
        private CalculatorContext _context;
        public CalculatorController(CalculatorContext context)
        {
            _context = context;
        }
        [HttpGet]
        public IActionResult Index()
        {
            return View();
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Calculate(double num1, double num2, Operation operation)
        {
            double result = 0;
            string errorMessage = null;
            try
            {
                switch (operation)
                {
                    case Operation.Add:
                        result = num1 + num2;
                        break;
                    case Operation.Subtract:
                        result = num1 - num2;
                        break;
```

```
            case Operation.Multiply:
                result = num1 * num2;
                break;
            case Operation.Divide:
                if (num2 != 0)
                {
                    result = num1 / num2;
                }
                else
                {
                    errorMessage = "Ошибка: деление на ноль невозможно.";
                }
                break;
        }
    }
    catch (Exception ex)
    {
        errorMessage = "Произошла ошибка: " + ex.Message;
    }
    ViewBag.Result = result;
    ViewBag.Num1 = num1;
    ViewBag.Num2 = num2;
    ViewBag.Operation = operation.ToString();
    ViewBag.ErrorMessage = errorMessage;

    // Создаем экземпляр класса DataInputVariant и заполняем его
    var dataInputVariant = new DataInputVariant
    {
        Operand_1 = num1.ToString(),
        Operand_2 = num2.ToString(),
        Type_operation = operation.ToString(),
        Result = result.ToString()
    };
    _context.DataInputVariants.Add(dataInputVariant);
    _context.SaveChanges();
    return View("Index");
}
```

Приложение Б Листинг программного кода приложения «Калькулятор» (appsettings.json)

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=mariadb; Database=9WebCalcul_Db; Uid=root; Pwd=password; Character Set=utf8; ConvertZeroDatetime=True;"
  },
  "Kestrel": {
    "Endpoints": {
      "Http": {
        "Url": "http://0.0.0.0:5009" // Заменить последние 2 цифры порта на порядковый номер из студенческого журнала. Например, порт 5012 соответствует номеру 12
      }
    }
  },

  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Приложение В Листинг программного кода приложения «Калькулятор» (appsettings.Development.json)

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=93.88.178.186; Port=5029; Database=9WebCalcul_Db; Uid=root; Pwd=password; Character Set=utf8; ConvertZeroDatetime=True;"
  },
  "DetailedErrors": true,
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  }
}
```

Приложение Г Листинг программного кода приложения «Калькулятор» (Program.cs)

```
using _09_Calculate.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

string mariadbCS = builder.Configuration.GetConnectionString("DefaultConnection");

builder.Services.AddDbContext<CalculatorContext>(options =>
{
    options.UseMySql(mariadbCS, new MySqlServerVersion(new Version(10, 5, 15)));
});

builder.Services.AddRazorPages();

var app = builder.Build();


// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Calculator}/{action=Index}/{id?}");

app.Run();
```