Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по домашней работе № 3

Название: HTTP-сервер на языке C#

Дисциплина: Разработка приложений на языке C#

Студент	ИУ6-73Б	_____	К.А.Логачев
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		_____	А.М. Минитаева
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

Цель работы: сделать HTTP сервер с функцией калькулятора на языке C #.

Задание: На основе ДЗ 2 сделать новый запрос на сервер. В поле ввода HTML файла вводится числовое выражение и нажимается кнопка «посчитать». Java Script делает AJAX запрос на сервер. C# программа из первого домашнего задания просчитывает результат и отдает клиенту результат в виде JSON структуры, которая потом выводится пользователю.

Ход работы

Описание работы программы:

1. Html-страничка про нажатию кнопки отправляет запрос с введенным в input примером
2. Сервер принимает запрос, парсит его
3. Создается созданный в ДЗ1 класс Калькулятора, он считает результат
4. Результат отправляется клиенту в формате JSON

Рисунок 1 – Схема работы алгоритма

Код программы интерфейса приведен ниже:

```
<!DOCTYPE html>
<html lang="ru">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
      Калькулятор
    </title>
    <script>
      function calculateExpression() {
        let expression = document.getElementById('expression').value;

        // Отправляем AJAX-запрос на сервер
        let xhr = new XMLHttpRequest();
        xhr.open('POST', 'http://localhost:49314/calculate', true);
        xhr.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
        xhr.onreadystatechange = function() {
          if (xhr.readyState === 4 && xhr.status === 200) {
            let result = JSON.parse(xhr.responseText);
```

```
                document.getElementById('result').innerHTML = 'Результат: ' +
result.Result;
              }
            };

            xhr.send(JSON.stringify(expression));
          }
        </script>
      </head>

      <body>
        <h1>
          Калькулятор
        </h1>
        <label for="expression">
          Введите арифметическое выражение:
        </label>
        <input type="text" id="expression" />
        <button onclick="calculateExpression()">
          Посчитать
        </button>
        <p id="result">
        </p>
      </body>

    </html>
```

Код программы для вычислений приведен ниже:

```
using System;
using System.Collections.Generic;

namespace AjaxWebServer {
  public class Calculator {
    // Метод для вычисления выражения по методу Бауэра-Земельзона
    public int Calculate(string expression) {
      // Шаг 1: Преобразование выражения в обратную польскую запись (ОПЗ)
      string rpn = ConvertToRPN(expression);

      // Шаг 2: Вычисление значения выражения, представленного в ОПЗ
      return EvaluateRPN(rpn);
    }

    // Преобразование инфиксного выражения в обратную польскую запись (ОПЗ)
    private string ConvertToRPN(string expression) {
      // Стек для хранения операторов
      Stack<char> stack = new Stack<char>();
      // Строка для результата в ОПЗ
      string output = "";
      // Переменная для накопления многоразрядных чисел
      string number = "";

      // Проход по каждому символу выражения
      foreach (char token in expression) {
        // Если символ — цифра (накапливаем число)
        if (char.IsDigit(token)) {
          number += token;
        } else {
          // Если число накоплено, добавляем его в результат и сбрасываем
          // накопление
          if (number != "") {
            output += number + " ";
            number = "";
          }
```

```
        // Обработка скобок и операторов
        if (token == '(') {
          stack.Push(token);  // Открывающая скобка помещается в стек
        } else if (token == ')') {
          // Закрывающая скобка: извлекаем операторы до открывающей скобки
          while (stack.Count > 0 && stack.Peek() != '(') {
            output += stack.Pop() + " ";
          }
          stack.Pop();  // Убираем открывающую скобку
        } else if (IsOperator(token)) {
          // Обработка операторов: поддержка приоритетов операций
          while (stack.Count > 0 &&
                 GetPrecedence(token) <= GetPrecedence(stack.Peek())) {
            output += stack.Pop() + " ";
          }
          stack.Push(token);  // Добавляем текущий оператор в стек
        }
      }
    }

    // Если осталось накопленное число, добавляем его в результат
    if (number != "") {
      output += number + " ";
    }

    // Извлекаем все оставшиеся операторы из стека
    while (stack.Count > 0) {
      output += stack.Pop() + " ";
    }

    // Возвращаем окончательный результат в виде строки ОПЗ
    return output.Trim();
  }

  // Вычисление выражения, представленного в ОПЗ
  private int EvaluateRPN(string rpn) {
    // Стек для операндов
    Stack<int> stack = new Stack<int>();
    // Разделяем строку ОПЗ на токены (операнды и операторы)
    string[] tokens = rpn.Split(' ');

    // Обрабатываем каждый токен
    foreach (string token in tokens) {
      // Если токен — число, помещаем его в стек
      if (int.TryParse(token, out int number)) {
        stack.Push(number);
      }
      // Если токен — оператор, выполняем операцию
      else if (IsOperator(token[0])) {
        // Извлекаем два операнда
        int operand2 = stack.Pop();
        int operand1 = stack.Pop();

        // Выполняем операцию и результат кладём обратно в стек
        int result = PerformOperation(operand1, operand2, token[0]);
        stack.Push(result);
      }
    }

    // В стеке остаётся окончательный результат
    return stack.Pop();
```

```
    }

    // Проверка, является ли символ оператором
    private bool IsOperator(char c) {
      return c == '+' || c == '-' || c == '*' || c == '/';
    }

    // Определение приоритета операций (чем выше значение, тем выше приоритет)
    private int GetPrecedence(char op) {
      switch (op) {
        case '+':
        case '-':
          return 1;  // Низкий приоритет
        case '*':
        case '/':
          return 2;  // Высокий приоритет
        default:
          return 0;  // Неизвестный оператор
      }
    }

    // Выполнение арифметической операции
    private int PerformOperation(int operand1, int operand2, char op) {
      switch (op) {
        case '+':
          return operand1 + operand2;  // Сложение
        case '-':
          return operand1 - operand2;  // Вычитание
        case '*':
          return operand1 * operand2;  // Умножение
        case '/':
          return operand1 / operand2;  // Деление
        default:
          throw new ArgumentException("Неверный оператор");
      }
    }
  }
}
```

Код программы сервера приведен ниже:

```
using System;
using System.IO;
using System.Net;
using System.Text.Json;
using System.Threading.Tasks;

namespace AjaxWebServer {
  public class Server {
    private HttpListener listener;
    private const int PORT = 49314;
    private Calculator calculator;

    public Server() {
      listener = new HttpListener();
      listener.Prefixes.Add($"http://localhost:{PORT}/");
      calculator = new Calculator();
    }

    public void Start() {
      listener.Start();
      Console.WriteLine(
          $"Сервер запущен на порту {PORT}. Ожидание запросов...");

      while (true) {
        var context = listener.GetContext();
```

```
        Task.Run(() => HandleRequest(context));
      }
    }

    private async Task HandleRequest(HttpListenerContext context) {
      string url = context.Request.Url.AbsolutePath;

      if (url == "/calculate" && context.Request.HttpMethod == "POST") {
        using (StreamReader reader =
                    new StreamReader(context.Request.InputStream)) {
          // Читаем выражение как строку JSON
          string jsonExpression = await reader.ReadToEndAsync();

          // Десериализуем JSON, чтобы извлечь строку выражения
          string expression =
              JsonSerializer.Deserialize<string>(jsonExpression);

          // Выполняем вычисление через Calculator
          int result = calculator.Calculate(expression);

          // Создаем объект для ответа
          var responseJson = new { Expression = expression, Result = result };

          // Сериализуем объект ответа в JSON
          string jsonResponse = JsonSerializer.Serialize(responseJson);
          context.Response.ContentType = "application/json";
          await context.Response.OutputStream.WriteAsync(
              System.Text.Encoding.UTF8.GetBytes(jsonResponse));
          context.Response.Close();
        }
      } else if (url == "/") {
        // Отправка HTML страницы
        string htmlPage = File.ReadAllText("index.html");
        byte[] htmlBytes = System.Text.Encoding.UTF8.GetBytes(htmlPage);
        context.Response.ContentType = "text/html";
        await context.Response.OutputStream.WriteAsync(htmlBytes, 0,
                                                       htmlBytes.Length);
        context.Response.Close();
      }
    }
  }
}
```

Код программы приведен ниже:

```
using System;

namespace AjaxWebServer {
  class Program {
    static void Main(string[] args) {
      Server server = new Server();
      server.Start();
    }
  }
}
```

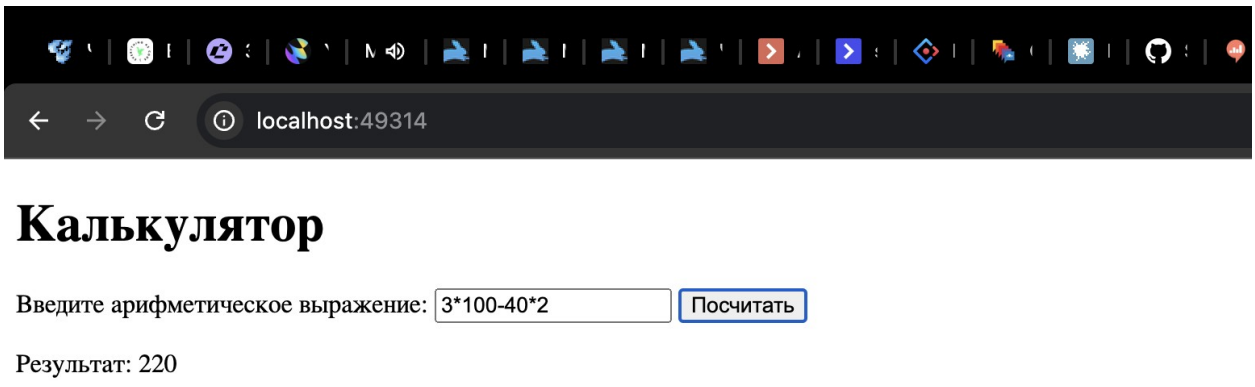

Рисунок 2 – Результат работы программы

Вывод: в ходе данной лабораторной работы была получена практика в написании http-серверов на языке C#.