



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 домашнему заданию № 2

Название: Разработка HTTP сервера

Дисциплина: Разработка приложений на языке C#

Студент ИУ6-71Б
(Группа)

И.В. Шлюжас
(Подпись, дата) 14.12.2023
(И.О. Фамилия)

Преподаватель

А.М. Минитаева
(Подпись, дата) 14.12.2023
(И.О. Фамилия)

Вариант 14.

Часть 1. HTTP сервер, отдающий статичную html страницу

Введение.

Веб-сервер (web server) – это программа, которая обрабатывает запросы на доступ к веб-страницам и возвращает клиенту (браузеру) запрошенную информацию. Веб-сервер может обслуживать несколько клиентов одновременно, обрабатывая запросы параллельно.

HTTP сервер является разновидностью веб-сервера, который работает с протоколом HTTP (HyperText Transfer Protocol). HTTP – это протокол прикладного уровня для передачи гипертекста (веб-страниц, представляющих собой текст с разметкой HTML) в сети Интернет.

Протокол HTTP использует клиент-серверную модель, где клиент отправляет запрос на сервер, а сервер возвращает ответ. Формат запроса и ответа определен протоколом.

Каждый запрос содержит метод, URL, версию протокола, различные заголовки и данные (необязательно). Методы запроса в HTTP включают GET, POST, PUT и DELETE. Например, GET-запрос используется для получения информации от сервера, а POST-запрос для отправки данных на сервер для дальнейшей обработки.

Каждый ответ содержит версию протокола, код статуса, заголовки и данные (необязательно). Код статуса определяет результат выполнения запроса (например, 200 означает "OK", а 404 означает "Not Found").

Клиент и сервер могут обмениваться более чем одним запросом и ответом в рамках одной сессии, для уменьшения нагрузки на сервер и сеть.

Таким образом, работа HTTP-протокола предполагает передачу данных между клиентом и сервером в формате запрос-ответ, используя определенные методы, заголовки и статус-коды, устанавливаемые протоколом.

Существует несколько вариантов реализации сервера: «Thread per Request» и «Pool Thread», каждая из которых имеет свои плюсы и минусы.

Цель работы – развитие навыков программирования на языке C# и создание простого HTTP сервера, который может обрабатывать запросы на отдачу статичных html страниц.

Задание:

1. Разработать алгоритм для обработки запросов на получение статичной html страницы по протоколу TCP/IP.
2. Реализовать HTTP сервер с использованием класса TcpListener, который будет принимать запросы на получение статичной html страницы и отдавать нужную страницу в ответ.
3. Создать статичную HTML страницу для тестирования сервера.
4. Протестировать работу сервера с использованием различных HTTP клиентов (например, браузеров или утилит типа curl).

В соответствии с вариантом ($14 \% 2 = 0$) необходимо реализовать сервер с применением «Pool Thread».

Ход работы

Разрабатываемая программа будет работать по следующему принципу:

- 1) Создание TcpListener с указанием IP и Порта;
- 2) Запуск TcpListener и обработка ошибки;
- 3) Обработка клиентов в соответствии с «Pool Thread»;
- 4) Парсинг пришедшего запроса;
- 5) Создание тела ответа согласно пришедшему запросу;
- 6) Добавление в тело запроса статичной HTML страницы;
- 7) Передача клиенту;
- 8) Закрытие соединения с клиентом.

Схемы алгоритмов методов класса, реализующего работу сервера с применением «Pool Thread» представлены на рисунках 1-3.



Рисунок 1 – Схема алгоритма метода создания сервера с применением «Pool Thread»

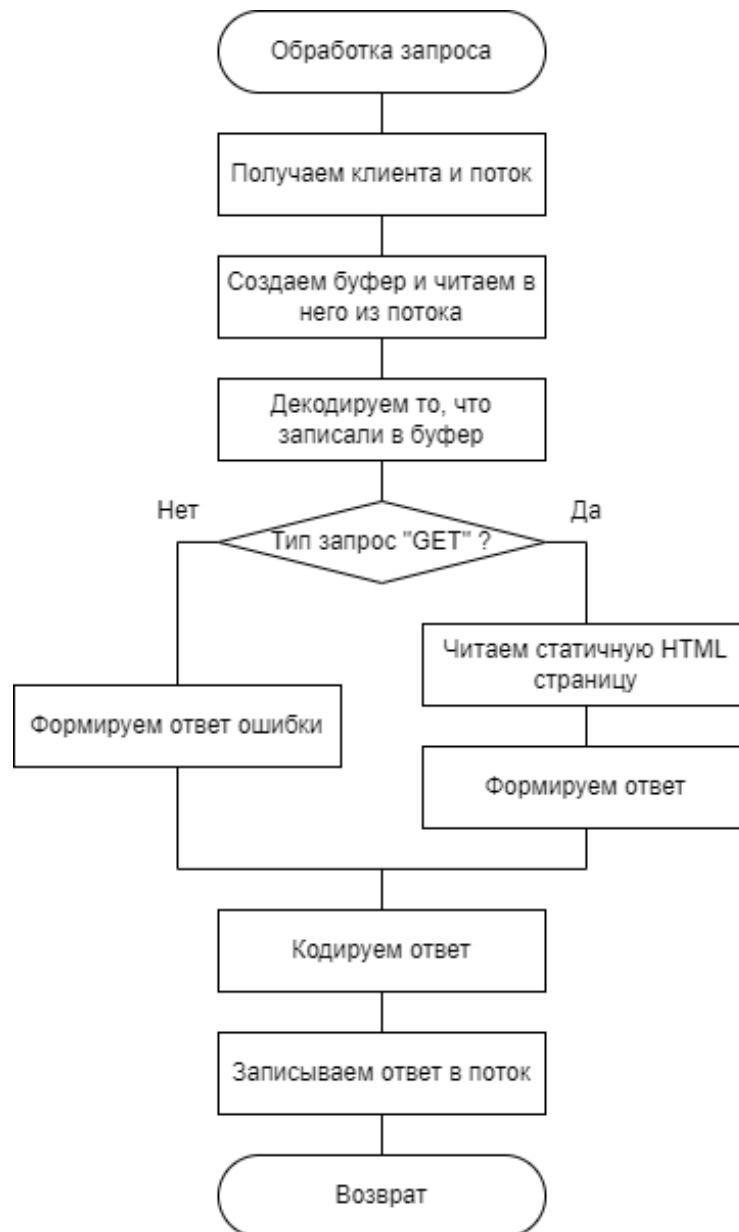


Рисунок 2 – Схема алгоритма метода обработки запроса и ответа на него



Рисунок 3 – Схема алгоритма метода остановки сервера

На рисунке 4 представлена схема алгоритма основной программы.

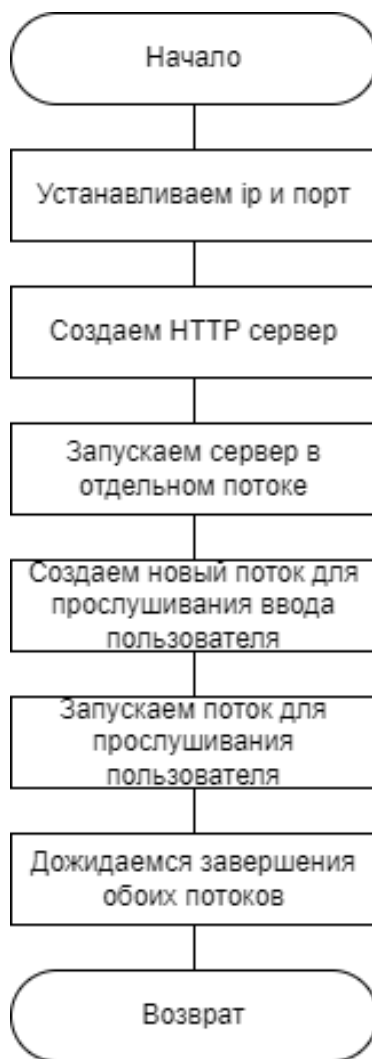


Рисунок 4 – Схема алгоритма основной программы

Исходя из полученных схем алгоритмов разработаем код программы.

Текст основной программы представлен в листинге 1.

Листинг 1 – Код основной программы

```
using System;
using System.Threading;

namespace Dz2._1
{
    internal static class Program
    {
        public static void Main(string[] args)
        {
            const string ipAddress = "127.0.0.1";
            const int port = 49214;

            var server = new HttpServer(ipAddress, port);

            // Start the server on a separate thread
            var serverThread = new Thread(server.Start);
```

```

        serverThread.Start();

        // Start a thread to handle user input for stopping the
server
        var userInputThread = new Thread(() =>
        {
            Console.WriteLine("Press any key to stop the
server...");
            Console.ReadKey();
            server.Stop();
        });
        userInputThread.Start();

        // Wait for both threads to complete
        serverThread.Join();
        userInputThread.Join();
    }
}

```

Код, реализующий класс для работы с HTTP сервером показан в листинге 2.

Листинг 2 – Код HTTP сервера

```

using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace Dz2._1
{
    public class HttpServer
    {
        private readonly TcpListener _listener;
        private bool _isRunning = true;
        private bool _isStopping = false;

        private const string HtmlFilePath = "../..//page.html";

        public HttpServer(string ipAddress, int port)
        {
            _listener = new TcpListener(IPAddress.Parse(ipAddress),
port);
            _listener.Start();
            Console.WriteLine($"Server started at
{ipAddress}:{port}");
        }

        public void Start()
        {
            while (_isRunning)
            {
                try

```

```

        {
            var client = _listener.AcceptTcpClient();
            ThreadPool.QueueUserWorkItem(ProcessClient,
client);
        }
        catch (SocketException)
        {
            // SocketException occurs when stopping the
listener
            // Handle it gracefully or just break the loop
            break;
        }
    }

    private static void ProcessClient(object obj)
    {
        using (var client = (TcpClient)obj)
        {
            using (var stream = client.GetStream())
            {
                var requestBuffer = new byte[4096];
                var bytesRead = stream.Read(requestBuffer, 0,
requestBuffer.Length);
                var request =
Encoding.UTF8.GetString(requestBuffer, 0, bytesRead);

                if (request.StartsWith("GET"))
                {
                    var responseHtml =
File.ReadAllText(HtmlFilePath);
                    var response = $"HTTP/1.1 200 OK\r\nContent-
Type: text/html\r\nContent-Length:
{responseHtml.Length}\r\n\r\n{responseHtml}";
                    var responseBytes =
Encoding.UTF8.GetBytes(response);
                    stream.Write(responseBytes, 0,
responseBytes.Length);
                }
                else
                {
                    const string ErrorResponse = "HTTP/1.1 400
Bad Request\r\n\r\nInvalid Request";
                    var ErrorResponseBytes =
Encoding.UTF8.GetBytes(ErrorResponse);
                    stream.Write(ErrorResponseBytes, 0,
ErrorResponseBytes.Length);
                }
            }
        }
    }

    public void Stop()
    {
        if (_isStopping) return;

```

```

        _isStopping = true;
        _isRunning = false;
        _listener.Stop();

        Console.WriteLine("Server stopped.");
    }

    ~HttpServer()
    {
        Stop();
    }
}

```

Код отдаваем статичной HTML странички продемонстрирован в листинге 3.

Листинг 3 – Код статичной HTML страницы

```

<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Sample HTML Page</title>
    </head>

    <body>
        <h1>Sample HTML Page</h1>
        <form action="/submit" method="post">
            <label for="inputField">Input:</label>
            <input type="text" id="inputField" name="inputField">
            <button type="submit">Submit</button>
        </form>
    </body>

</html>

```

Результат работы программы показан на рисунке 5.

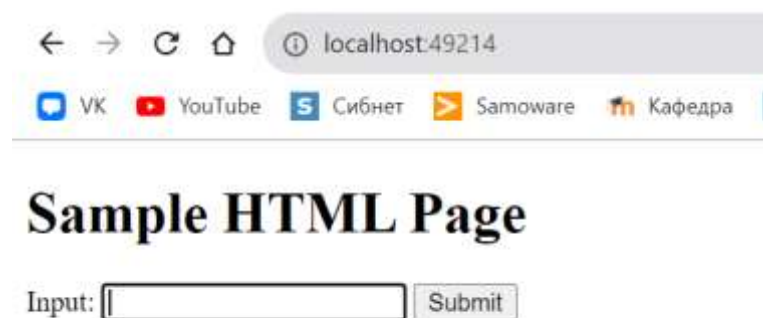


Рисунок 5 – Результат работы HTTP сервера

Вывод: в рамках первой части домашнего задания я ознакомился с основами TCP-подключения, протоколом TCP/IP, номера портов и методами создание сервера на языке программирования C#, разработал веб-сервер с применением «Pool Thread», отдающий статичную HTML страницу.

Часть 2. Бэкенд с JSON форматом ответа на запрос

Введение.

Формат JSON (JavaScript Object Notation) – это легковесный формат обмена данными, который основан на языке JavaScript. Он представляет собой текстовый формат, который используется для передачи данных между приложениями.

Структура данных JSON представляет собой коллекцию пар "ключ-значение", где ключом может быть строка, а значением - любой из следующих типов данных: строка, число, логический тип, массив, объект или null. Данные в формате JSON могут быть легко интерпретированы и созданы как на стороне сервера, так и на стороне клиента.

Концепция AJAX (Asynchronous JavaScript and XML) – это технология, которая позволяет отправлять асинхронные запросы на сервер и обновлять часть страницы без необходимости перезагрузки всей страницы. AJAX запросы обычно отправляются с помощью JavaScript-кода.

Запросы AJAX могут использоваться для получения данных с сервера, отправки данных на сервер, обновления содержимого страницы и других задач.

Чтобы отправить AJAX запрос с html страницы, можно использовать JavaScript объект XMLHttpRequest.

Сериализация данных – это процесс преобразования объекта или структуры данных в последовательность байтов или в другой формат, который может быть сохранен в файле, передан по сети или сохранен в другом месте. Такой формат может быть использован для восстановления данных в исходном виде в будущем.

Сериализация – это важный инструмент для работы с данными в современных приложениях, который позволяет эффективно сохранять, передавать и восстанавливать информацию в различных форматах.

Работу с форматом JSON и его сериализацию в языке C# можно производить с помощью .NET библиотеки Newtonsoft.Json или с помощью System.Text.Json.

Цель работы – развитие навыков программирования на языке C# и создание простого HTTP сервера, который может обрабатывать запросы на отдачу статичных html страниц, а также способен обрабатывать AJAX запрос и возвращать ответ в формате JSON.

Задание: на основе части 1 домашнего задания №2 сделать обработку AJAX запроса веб-сервером. В поле ввода html страницы вводится арифметическое выражение и нажимается кнопка «Посчитать». Код JavaScript отправляет AJAX запрос на сервер. Программа на языке C# из домашнего задания №1 выполняет разбор строки с арифметическим выражением и вычисляет его результат. Далее веб-сервер отдает клиенту (в браузер на html страницу) результат в виде JSON структуры, результат вычислений из которой потом выводится пользователю внутри веб-страницы.

В соответствии с вариантом ($14 \% 2 = 0$) сериализацию JSON необходимо реализовать с помощью библиотеки System.Text.Json.

Ход работы

Разрабатываемая программа будет работать по следующему принципу:

- 1) Создать фрагмент кода на JavaScript, который будет отправлять и обрабатывать запрос;
- 2) Реализовать обработку AJAX запроса, отделять её от запроса на получение страницы;
- 3) Полученные данные отправить в подпрограмму для расчета арифметического выражения;
- 4) Отправить JSON структуру с результатом вычислений пользователю на веб-страницу.

Схемы алгоритмов методов класса, реализующего работу сервера с применением «Pool Thread» и обработкой AJAX запроса представлены на рисунках 6-8.



Рисунок 6 – Схема алгоритма метода создания сервера с применением «Pool Thread»

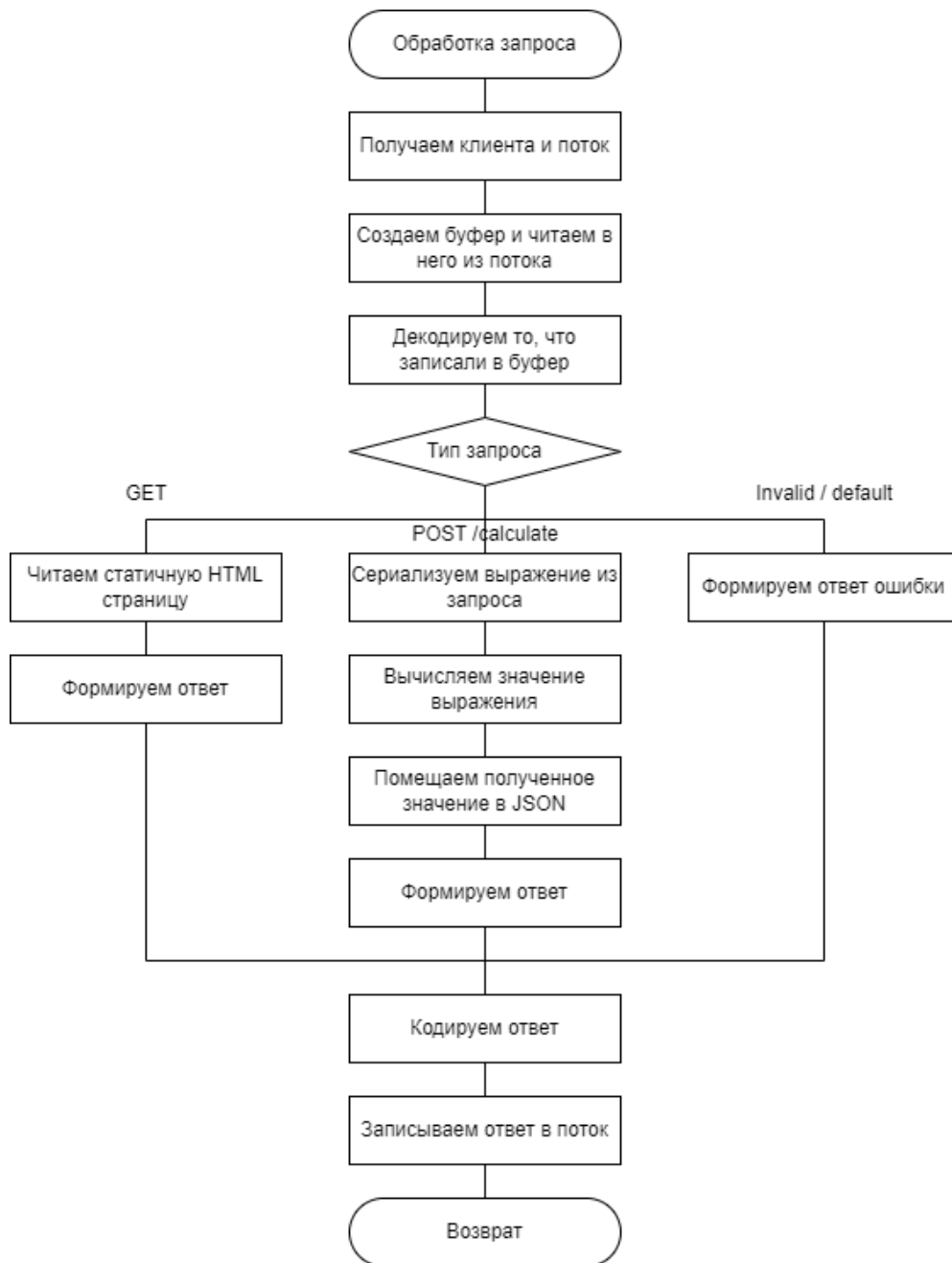


Рисунок 7 – Схема алгоритма метода обработки запроса и ответа на него

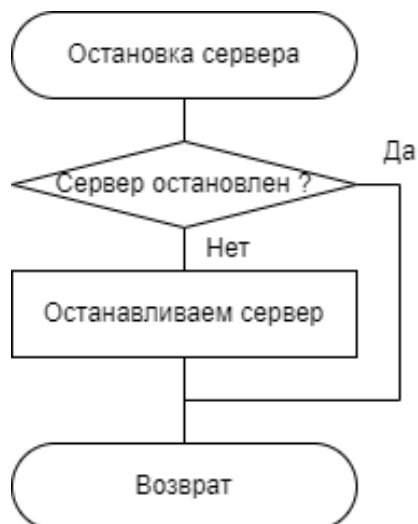


Рисунок 8 – Схема алгоритма метода остановки сервера

На рисунке 9 представлена схема алгоритма основной программы.

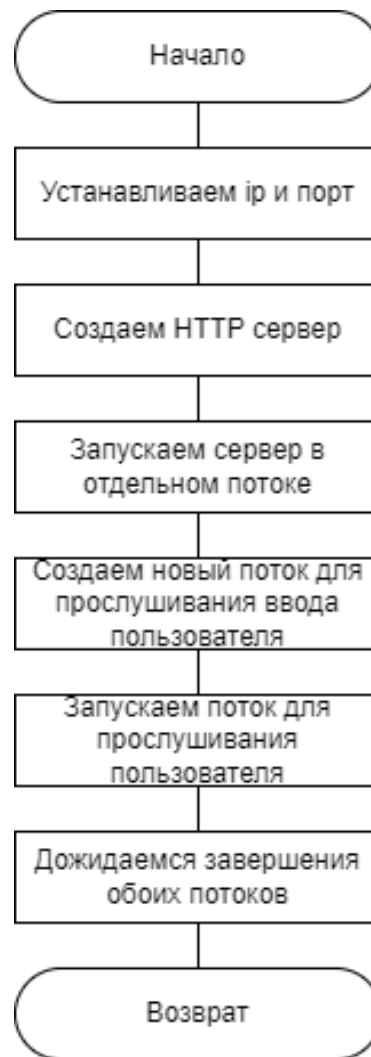


Рисунок 9 – Схема алгоритма основной программы

Исходя из полученных схем алгоритмов разработаем код программы.

Текст основной программы представлен в листинге 4.

Листинг 4 – Код основной программы

```
using System;
using System.Threading;

namespace Dz2._1
{
    internal static class Program
    {
        public static void Main(string[] args)
        {
            const string ipAddress = "127.0.0.1";
            const int port = 49214;

            var server = new HttpServer(ipAddress, port);

            // Start the server on a separate thread
            var serverThread = new Thread(server.Start);
```

```

        serverThread.Start();

        // Start a thread to handle user input for stopping the
server
        var userInputThread = new Thread(() =>
        {
            Console.WriteLine("Press any key to stop the
server...");
            Console.ReadKey();
            server.Stop();
        });
        userInputThread.Start();

        // Wait for both threads to complete
        serverThread.Join();
        userInputThread.Join();
    }
}

```

Код, реализующий класс для работы с HTTP сервером показан в листинге 5.

Листинг 5 – Код HTTP сервера

```

using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Text.Json;

using Calc = Calculator.Calculator;

namespace Dz2._2
{
    public class HttpServer
    {
        private readonly TcpListener _listener;
        private bool _isRunning = true;
        private bool _isStopping = false;

        private const string HtmlFilePath =
"../../../../../calculate.html";

        private enum RequestType
        {
            Invalid,
            Get,
            PostCalculate
        }

        private static RequestType GetRequestType(string request)
        {
            if (request.StartsWith("GET")) return RequestType.Get;
            if (request.StartsWith("POST /calculate")) return
RequestType.PostCalculate;

```

```

        return RequestType.Invalid;
    }

    public HttpServer(string ipAddress, int port)
    {
        _listener = new TcpListener(IPAddress.Parse(ipAddress),
port);
        _listener.Start();
        Console.WriteLine($"Server started at
{ipAddress}:{port}");
    }

    public void Start()
    {
        while (_isRunning)
        {
            try
            {
                var client = _listener.AcceptTcpClient();
                ThreadPool.QueueUserWorkItem(ProcessClient!,
client);
            }
            catch (SocketException)
            {
                break;
            }
        }
    }

    private static void ProcessClient(object obj)
    {
        using var client = (TcpClient)obj;
        using var stream = client.GetStream();

        var requestBuffer = new byte[4096];
        var bytesRead = stream.Read(requestBuffer, 0,
requestBuffer.Length);
        var request = Encoding.UTF8.GetString(requestBuffer, 0,
bytesRead);

        string response;
        switch (GetRequestType(request))
        {
            case RequestType.Get:
                var responseHtml =
File.ReadAllText(HtmlFilePath);

                response = $"HTTP/1.1 200 OK\r\nContent-Type:
text/html\r\nContent-Length:
{responseHtml.Length}\r\n\r\n{responseHtml}";
                break;

            case RequestType.PostCalculate:
                var expression =

```

```

GetExpressionFromRequest(request);

        string result;
        try
        {
            result =
Calc.EvaluateExpression(expression).ToString();
        }
        catch
        {
            result = "Invalid expression!";
        }

        var jsonResponse = new { result };
        var jsonString =
JsonSerializer.Serialize(jsonResponse);

        response = $"HTTP/1.1 200 OK\r\nContent-Type:
application/json\r\nContent-Length:
{jsonString.Length}\r\n\r\n{jsonString}";
        break;

        case RequestType.Invalid:
        default:
            response = "HTTP/1.1 400 Bad
Request\r\n\r\nInvalid Request";
            break;
    }

    var responseBytes = Encoding.UTF8.GetBytes(response);
    stream.Write(responseBytes, 0, responseBytes.Length);
}

private static string GetExpressionFromRequest(string
request)
{
    var requestBody = request[request.IndexOf('{')..];
    dynamic? requestData =
JsonSerializer.Deserialize<Dictionary<string, string>>(requestBody);
    return requestData?["expression"] ?? "Invalid
expression!";
}

public void Stop()
{
    if (_isStopping) return;

    _isStopping = true;
    _isRunning = false;
    _listener.Stop();

    Console.WriteLine("Server stopped.");
}

```

```

        ~HttpServer()
        {
            Stop();
        }
    }
}

```

Код отдаваем статичной HTML странички продемонстрирован в листинге 6.

Листинг 6 – Код статичной HTML страницы

```

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Arithmetic Calculator</title>
    </head>

    <body>
        <h1>Arithmetic Calculator</h1>
        <label for="expressionInput">Enter Arithmetic
Expression:</label>
        <input type="text" id="expressionInput"
name="expressionInput">
        <button onclick="calculateExpression()">Calculate</button>
        <div id="result"></div>

        <script>
            function calculateExpression() {
                const expression =
document.getElementById("expressionInput").value;

                // Send AJAX request
                const xhr = new XMLHttpRequest();
                xhr.open("POST", "/calculate", true);
                xhr.setRequestHeader("Content-Type",
"application/json");
                xhr.onreadystatechange = function () {
                    if (xhr.readyState === 4 && xhr.status === 200)
                {
                    const response =
JSON.parse(xhr.responseText);
                    document.getElementById("result").innerText
= "Result: " + response.result;
                }
                };
                xhr.send(JSON.stringify({ expression: expression
}));
            }
        </script>
    </body>
</html>

```

Для выполнения вычисления был взят код калькулятора выражений из домашнего задания №1.

Результат работы программы показан на рисунках 10-12.

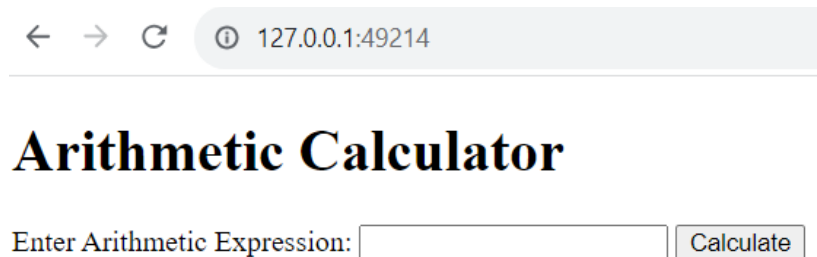


Рисунок 10 – Результат работы HTTP сервера

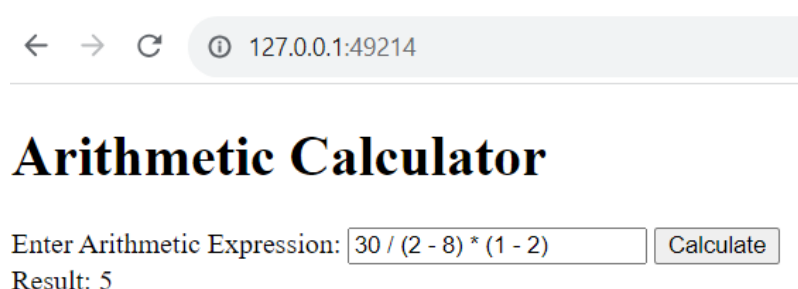


Рисунок 11 – Результат работы HTTP сервера с вычислением значения

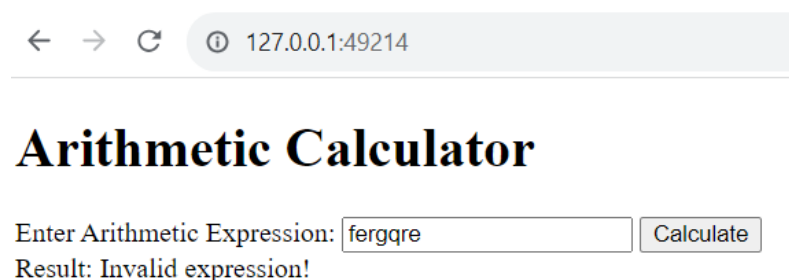


Рисунок 12 – Результат работы HTTP сервера с обработкой ошибки
На рисунке 13 представлены заголовки запроса и ответа.

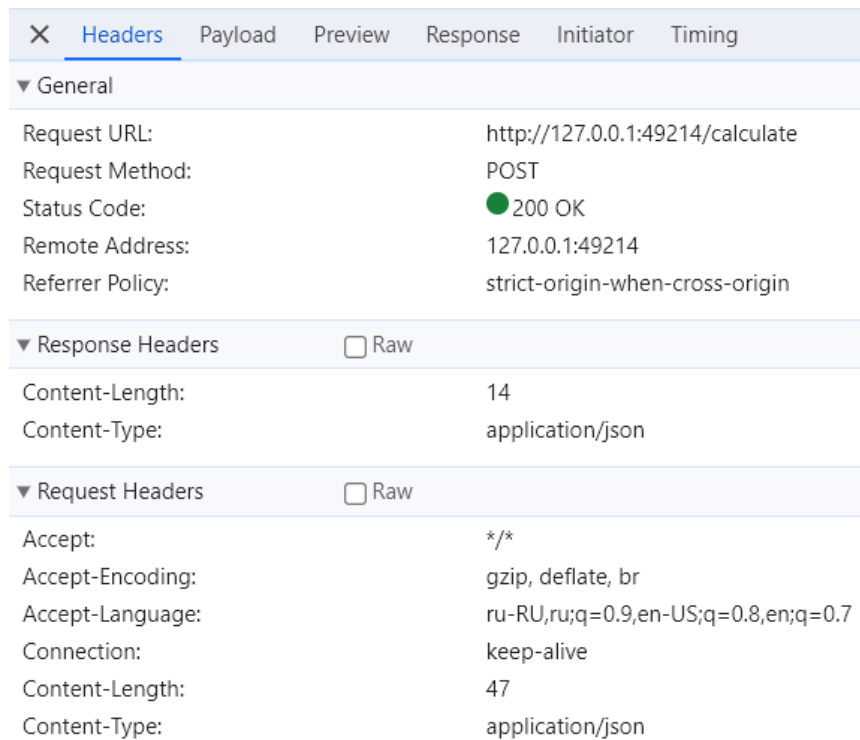


Рисунок 13 – Заголовки HTTP запроса

На рисунке 14 показаны исходные данные (выражение, введенное в поле ввода), на рисунке 15 представлен ответ в JSON формате от сервера, на запрос с рисунков 12 и 13.

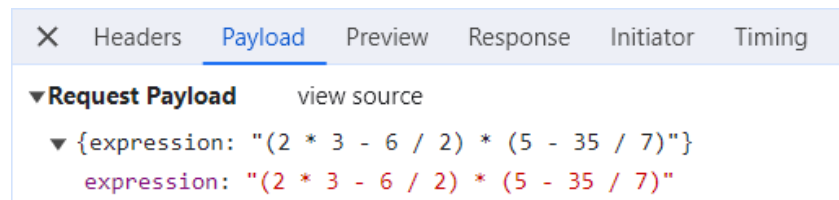


Рисунок 14 – Тело отправляемого запроса

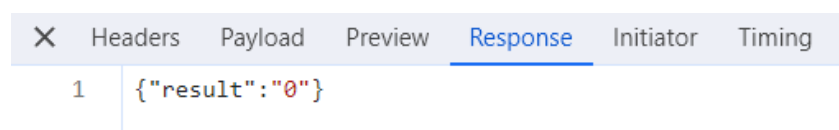


Рисунок 15 – Тело ответа на запрос

Вывод: в рамках второй части домашнего задания я создал фрагмент кода на JavaScript, который отправляет и обрабатывает AJAX запрос, научился обрабатывать POST запросы на сервере и отправлять ответ в формате JSON.