МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №8

по дисциплине: Компьютерные сети
тема: «Программирование протокола HTTP»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
Рубцов Константин Анатольевич

Белгород 2025 г.

Лабораторная работа №8
Программирование протокола HTTP

Цель работы: изучить протокол HTTP и составить программу согласно заданию.

Краткие теоретические сведения

HTTP (Hyper Text Transfer Protocol – протокол передачи гипертекста) – протокол прикладного уровня стека протоколов TCP/IP, предназначенный для передачи данных по сети с использованием транспортного протокола TCP. Текущая версия протокола HTTP v1.1, его спецификация приводится в документе RFC 2616.

Протокол HTTP может использоваться также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP или XML-RPC.

Основой HTTP является технология «клиент-сервер». HTTP-клиенты отсылают HTTP-запросы, которые содержат метод, обозначающий потребность клиента. Также такие запросы содержат универсальный идентификатор ресурса, указывающий на желаемый ресурс. Обычно такими ресурсами являются хранящиеся на сервере файлы. По умолчанию HTTP-запросы передаются на порт 80. HTTP-сервер отсылает коды состояния, сообщая, успешно ли выполнен HTTP-запрос или же нет.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т.д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

Унифицированный идентификатор ресурса представляет собой сочетание унифицированного указателя ресурса (Uniform Resource Locator, URL) и унифицированного имени ресурса (Uniform Resource Name, URN). Например:

- URI = http://iitus.bstu.ru/to_schoolleaver/230400
- URL = http://iitus.bstu.ru/
- URN = to_schoolleaver/230400

Метод протокола HTTP – это команда, передаваемая HTTP-клиентом HTTP-серверу. В табл. 8.1. перечислены некоторые методы, определенные в протоколе HTTP v1.1. Полный список методов HTTP v1.1. содержится в документе RFC 2616.

Таблица 8.1

Основные методы HTTP v1.1

Метод	Назначение
GET	Используется для запроса содержимого ресурса, на который указывает URI, содержащийся в запросе
HEAD	Используется для извлечения метаданных или проверки наличия ресурса, на который указывает URI, содержащийся в запросе
POST	Применяется для передачи данных заданному ресурсу. Данный метод предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого
PUT	Применяется для передачи данных заданному ресурсу. Данный метод предполагает, что передаваемое клиентом содержимое соответствует находящемуся по данному URI ресурсу
OPTIONS	Используется для определения возможностей HTTP-сервера или параметров соединения для конкретного ресурса

DELETE	Применяется для удаления ресурса, на который указывает URI
PATCH	Применяется для частичного обновления по данному URI ресурсу

Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру. Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (NotImplemented). Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (MethodNotAllowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Код состояния HTTP представляет собой целое число из трех цифр. Первая цифра указывает на класс состояния:

- информационные сообщения (1XX)
- успешное выполнение (2XX)
- переадресация (3XX)
- ошибка клиента (4XX)
- ошибка сервера (5XX)

Полный список статусов HTTP v1.1. содержится в документе RFC 2616. Примеры:

- 201 Webpage Created
- 403 Access allowed only for registered users

Каждое HTTP-сообщение состоит из трех частей, которые передаются в следующем порядке:

1. Стартовая строка – определяет тип сообщения
2. Заголовки – характеризуют тело сообщения, параметры передачи и прочие сведения
3. Тело сообщения – непосредственно данные сообщения.

Стартовые строки HTTP-сообщения различаются для запроса и ответа. Стартовая строка HTTP-запроса имеет следующий формат: Метод URI HTTP/Версия, где метод - название запроса, URI определяет путь к запрашиваемому документу, версия - пара разделённых точкой арабских цифр. Стартовая строка HTTP-ответа имеет следующий формат: HTTP/Версия КодСостояния Пояснение.

Заголовок HTTP представляет собой строку в HTTP-сообщении, содержащую разделенную двоеточием пару параметр-значение. Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой. Все заголовки разделяются на четыре основных группы:

1. Основные заголовки (General Headers) – должны включаться в любое сообщение клиента и сервера.
2. Заголовки запроса (Request Headers) – используются только в запросах клиента.
3. Заголовки ответа (Response Headers) – только для ответов от сервера.
4. Заголовки сущности (Entity Headers) – сопровождают каждую сущность сообщения.

Полный список заголовков HTTP v1.1. содержится в документе RFC 2616. Примеры заголовков:

Content-Type: text/plain; charset=windows-1251

Content-Language: ru

Тело HTTP-сообщения, если оно присутствует, используется для передачи данных, связанных с запросом или ответом.

Чтобы понять, как работает протокол HTTP, рассмотрим пример получения HTML-страницы с HTTP-сервера.

HTTP-запрос:

```
GET /to_schoolleaver/230400 HTTP/1.1
Host: iitus.bstu.ru
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0) Accept: text/html
Accept-Language: ru
Connection: close
```

HTTP-ответ:

```
HTTP/1.1 200 OK
Date: Fri, 16 Dec 2012 13:45:00 GMT
Server: Apache Last-Modified: Wed, 11 Feb 2009 11:20:59 GMT
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close

(далее следует запрошенная HTML-страница)
```

Протокол HTML позволяет достаточно легко создавать клиентские приложения. Возможности протокола можно расширить благодаря внедрению своих собственных заголовков, с помощью которых можно получить необходимую функциональность при решении нетривиальной задачи. При этом сохраняется совместимость с другими клиентами и серверами: они будут просто игнорировать неизвестные им заголовки.

Протокол HTTP устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.

При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт заголовок «Content-Type: тип/подтип», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле (при этом один и тот же файл в зависимости от аргументов запроса и своих собственных соображений может порождать ответы разных типов — в простейшем случае картинки в разных форматах).

Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

Перечисленные особенности HTTP позволили создавать поисковые машины (первой из которых стала AltaVista, созданная фирмой DEC), форумы и Internet-магазины. Это коммерциализировало Интернет, появились компании, основным полем деятельности которых стало предоставление доступа в Интернет (провайдеры) и создание сайтов.

Используемые функции

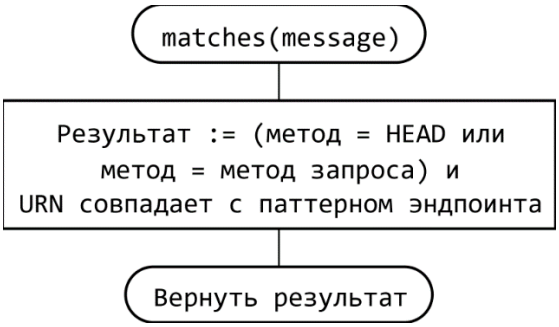
- Перевод сокета в состояние “прослушивания” (для TCP) осуществляется функцией listen (SOCKET s, int backlog), где s – дескриптор сокета; backlog – максимальный

размер очереди входящих сообщений на соединение. Используется сервером, чтобы информировать ОС, что он ожидает (“слушает”) запросы связи на данном сокете. Без этой функции всякое требование связи с сокетом будет отвергнуто.

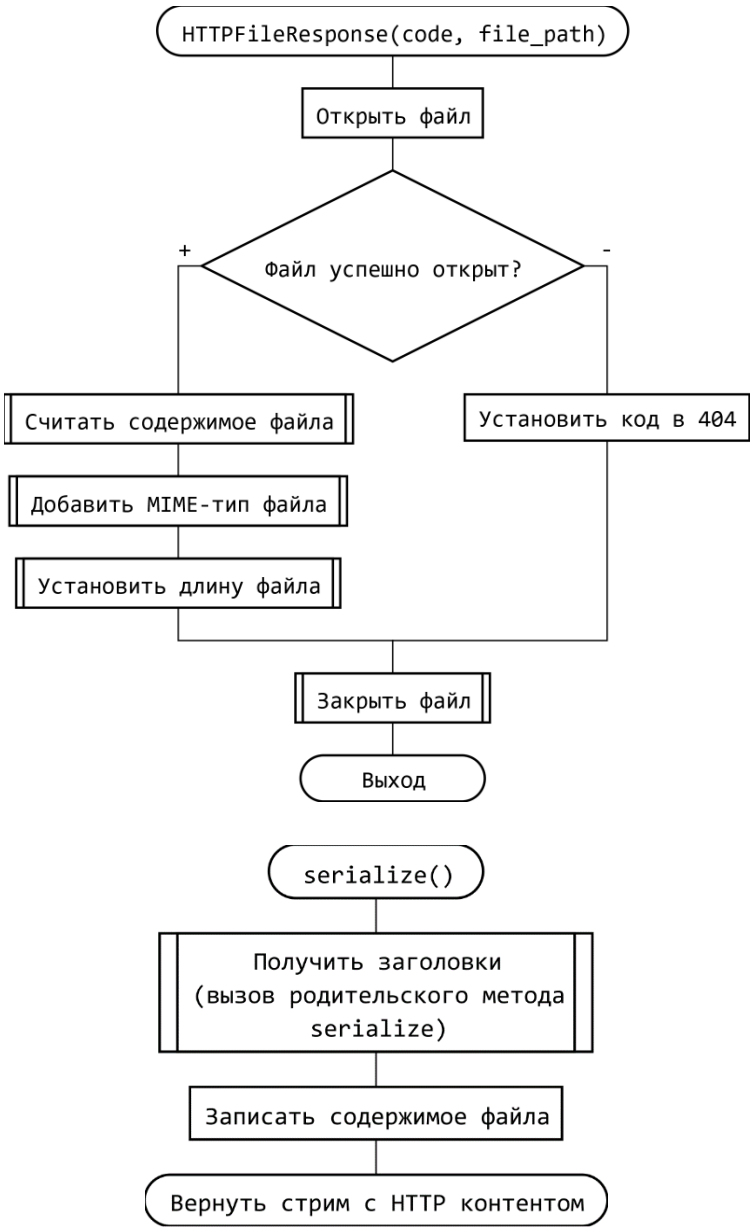
- Функция connect (SOCKET s, const struct sockaddr FAR* name, int namelen) нужна для соединения с сокетом, находящимся в состоянии “прослушивания” (для TCP). Она ипользуется процессомклиентом для установления связи с сервером. В случае успешного установления соединения connect возвращает 0, иначе SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Функция accept (SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen) служит для подтверждения запроса на соединение (для TCP). Функция используется для принятия связи на сокет. Сокет должен быть уже слушающим в момент вызова функции. Если сервер устанавливает связь с клиентом, то данная функция возвращает новый сокет-дескриптор, через который и производит общение клиента с сервером. Пока устанавливается связь клиента с сервером, функция блокирует другие запросы связи с данным сервером, а после установления связи “прослушивание” запросов возобновляется.
- В случае автоматического распределения адресов и портов узнать какой адрес и порт присвоен сокету можно при помощи функции getsockname (SOCKET s, struct sockaddr FAR* name, int FAR* namelen). Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Для передачи данных по протоколу UDP используется функция sendto (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen). Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Для передачи данных по протоколу TCP используется функция send (SOCKET s, const char FAR * buf, int len, int flags), где s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Для приема данных по протоколу UDP используется функция recvfrom (SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen). Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Для приема данных по протоколу TCP используется функция recv (SOCKET s, char FAR* buf, int len, int flags). Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError

Разработка программы. Блок-схемы программы.

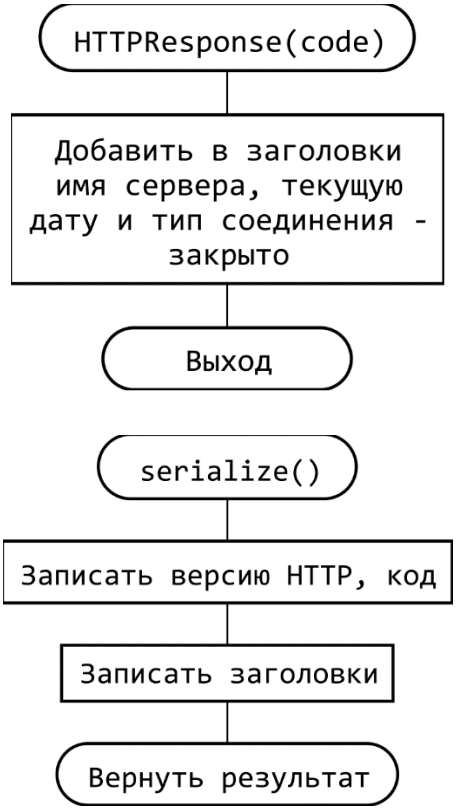
HTTPEndpoint



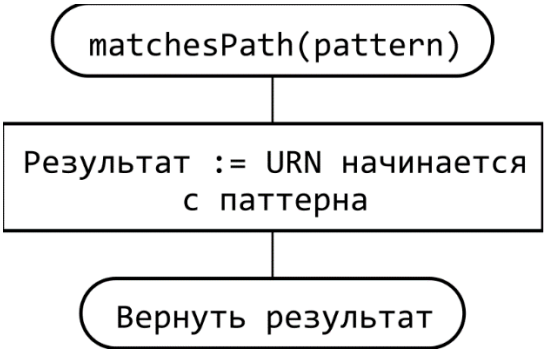
HTTPFileResponse

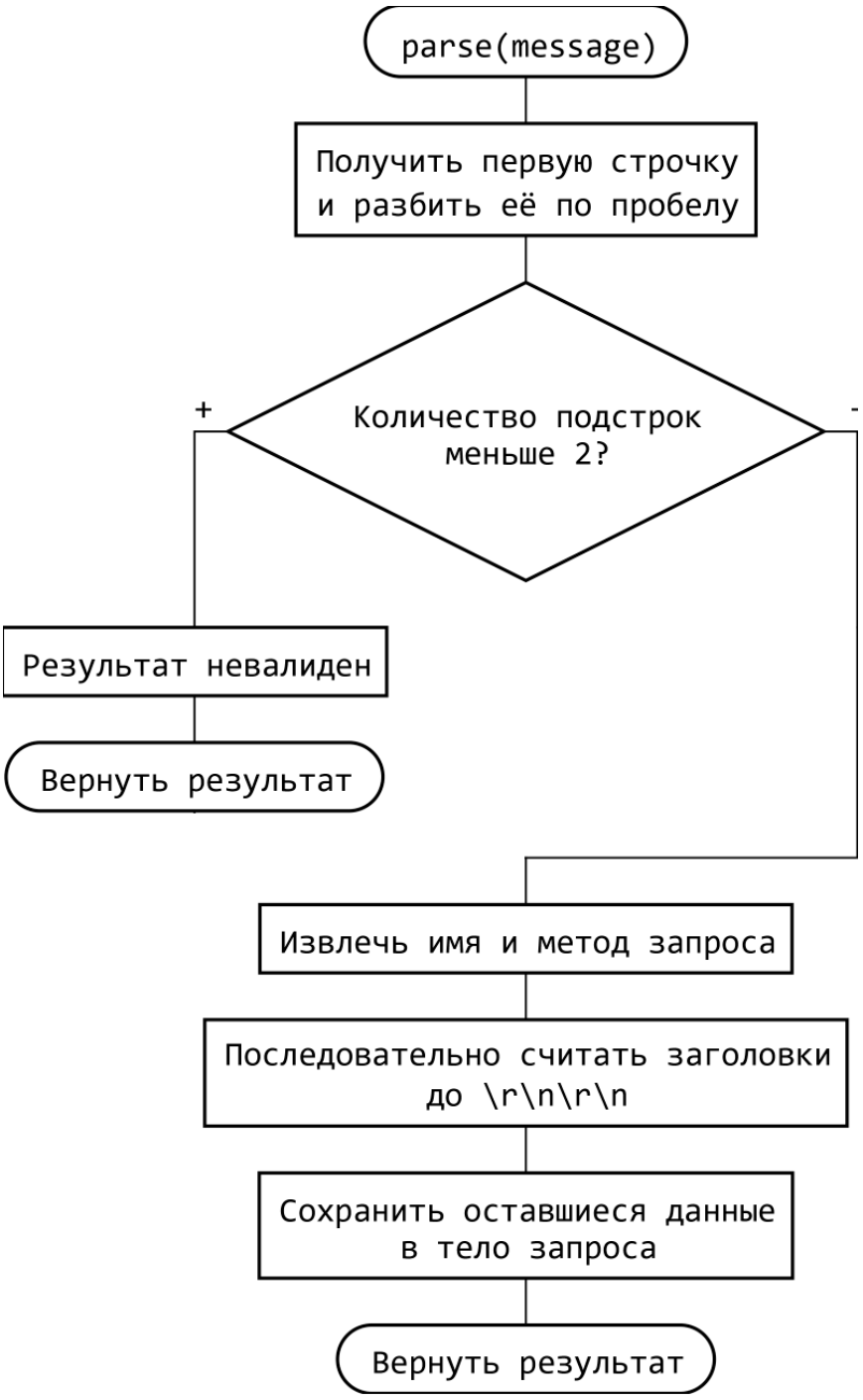


HTTPResponse

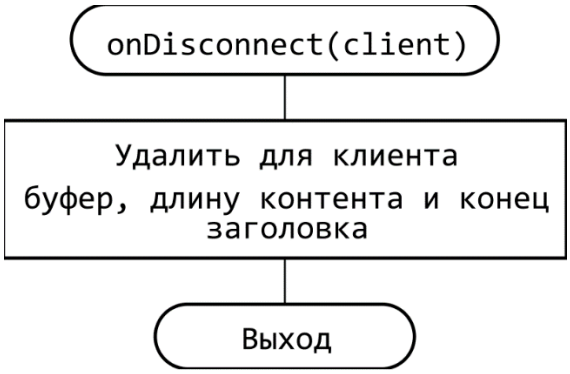


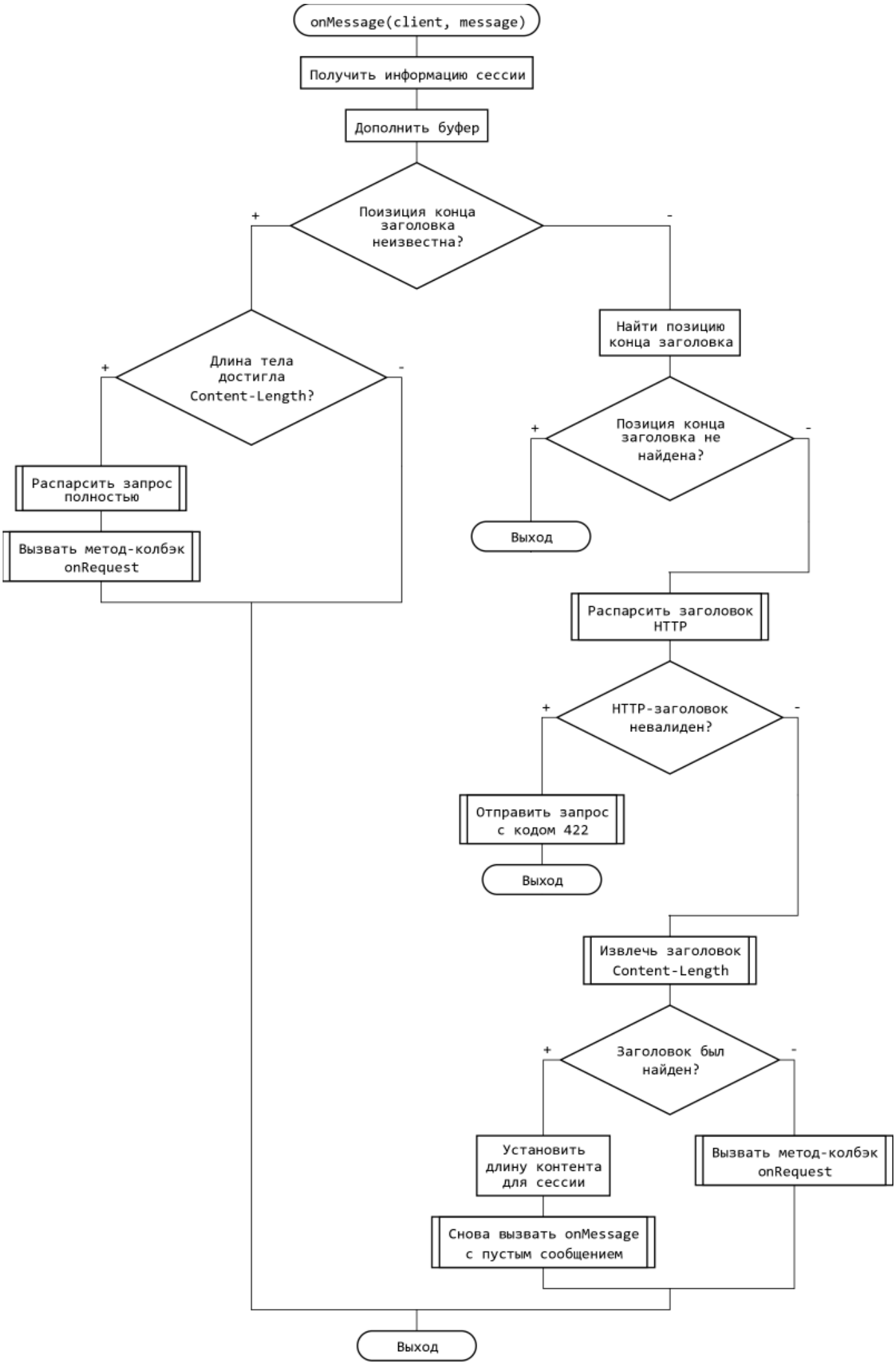
HTTPRequest

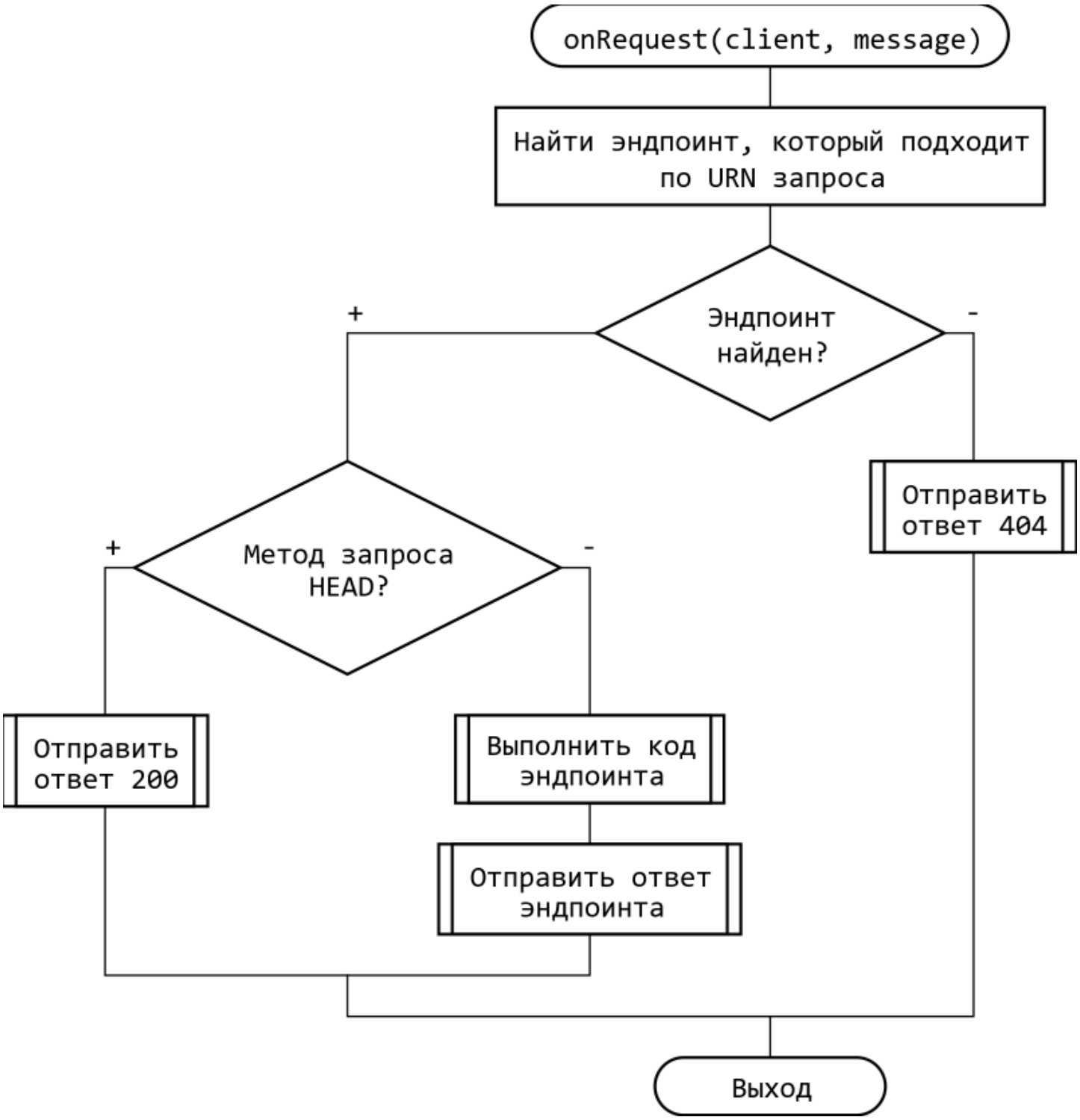


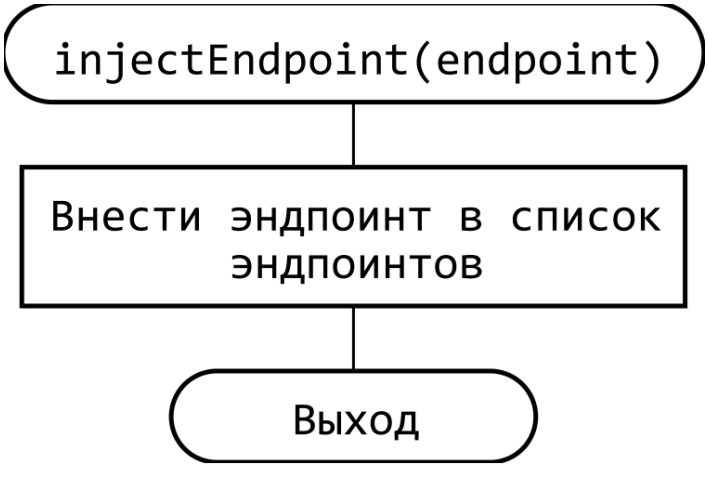

parse(message)
Получить первую строчку и разбить её по пробелу
Количество подстрок меньше 2?
+
-
Результат невалиден
Вернуть результат
Извлечь имя и метод запроса
Последовательно считать заголовки до \r\n\r\n
Сохранить оставшиеся данные в тело запроса
Вернуть результат

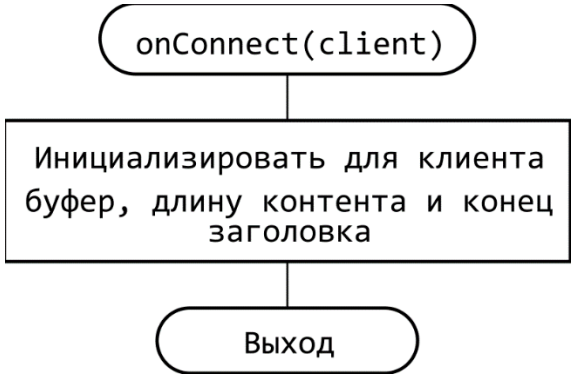
HTTPServer

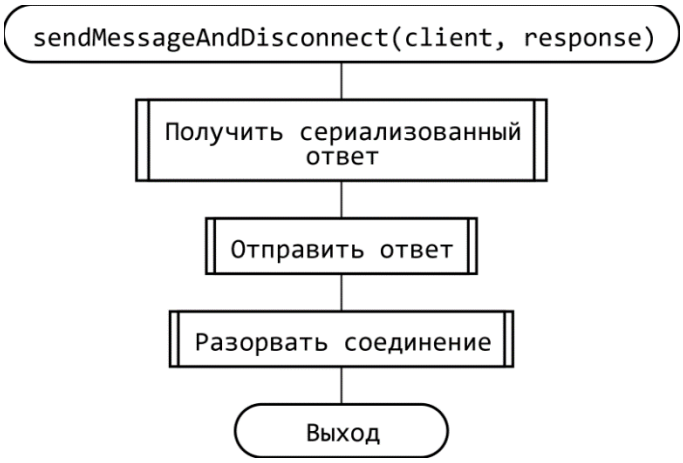

onDisconnect(client)
Удалить для клиента буфер, длину контента и конец заголовка
Выход


onMessage(client, message)
Получить информацию сессии
Дополнить буфер
Поизиция конца заголовка неизвестна?
+
-
Длина тела достигла Content-Length?
+
-
Распарсить запрос полностью
Вызвать метод-колбэк onRequest
Найти позицию конца заголовка
Позиция конца заголовка не найдена?
+
-
Выход
Распарсить заголовок HTTP
HTTP-заголовок невалиден?
+
-
Отправить запрос с кодом 422
Выход
Извлечь заголовок Content-Length
Заголовок был найден?
+
-
Установить длину контента для сессии
Снова вызвать onMessage с пустым сообщением
Вызвать метод-колбэк onRequest
Выход

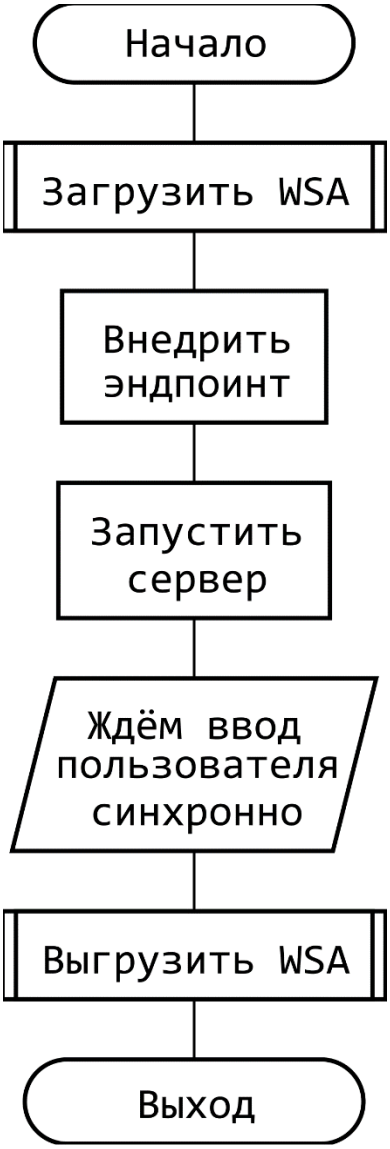

onRequest(client, message)
Найти эндпоинт, который подходит по URN запроса
Эндпоинт найден?
+
-
Отправить ответ 404
Метод запроса HEAD?
+
-
Отправить ответ 200
Выполнить код эндпоинта
Отправить ответ эндпоинта
Выход


injectEndpoint(endpoint)
Внести эндпоинт в список эндпоинтов
Выход


onConnect(client)
Инициализировать для клиента буфер, длину контента и конец заголовка
Выход


sendMessageAndDisconnect(client, response)
Получить сериализованный ответ
Отправить ответ
Разорвать соединение
Выход

Основная программа



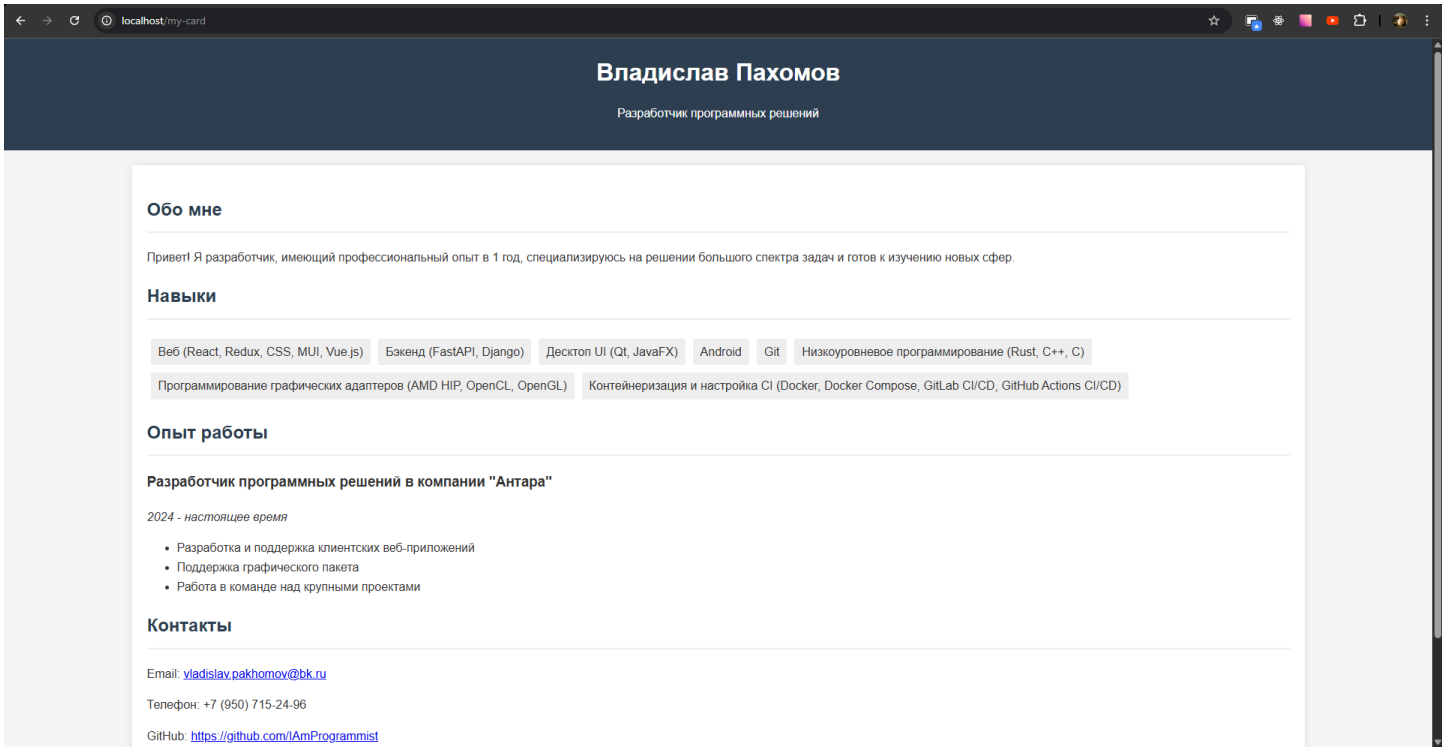
Анализ функционирования программ

Разработанная программа позволяет создать HTTP-сервер, добавлять свои ресурсы (под разные URN). В качестве примера будем выдавать HTML-страницу с визиткой автора лабораторной работы.

Полученная программа базируется на решении лабораторной работы №4 (класс TCPServer), поэтому это решение поддерживает преимущества, которые были представлены в программе, например многопоточность.

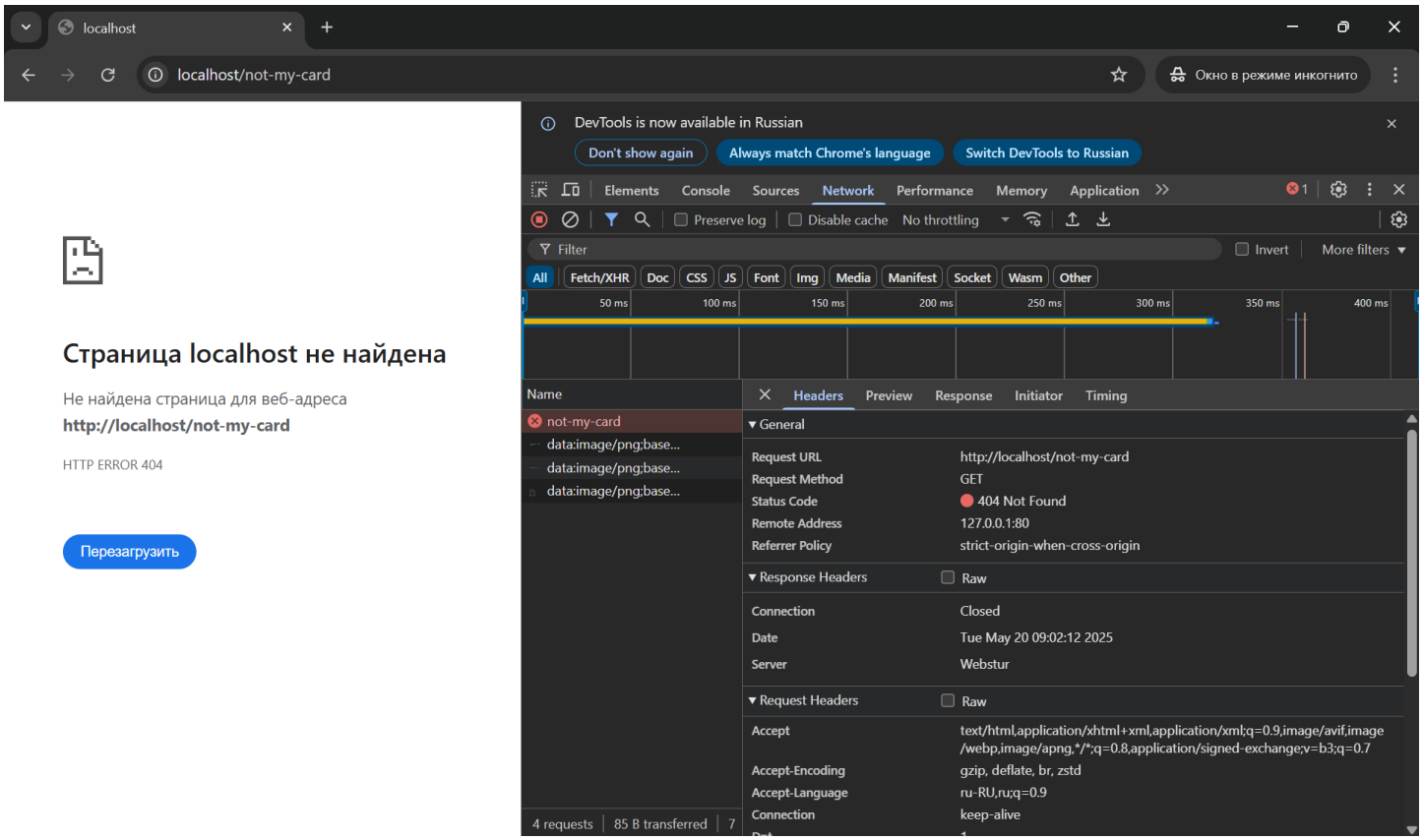
Попробуем получить эту страницу при помощи Google Chrome, указав в качестве URL локальный адрес, а в качестве URN – ресурс, который указан был при инициализации эндпоинта (/my-card).

Результат выполнения запроса:



Аналогичный результат можно получить, если подключиться к серверу с мобильного устройства в той же сети что и сервер

Если же результат не обнаружен, сервер возвращает ответ с кодом 404



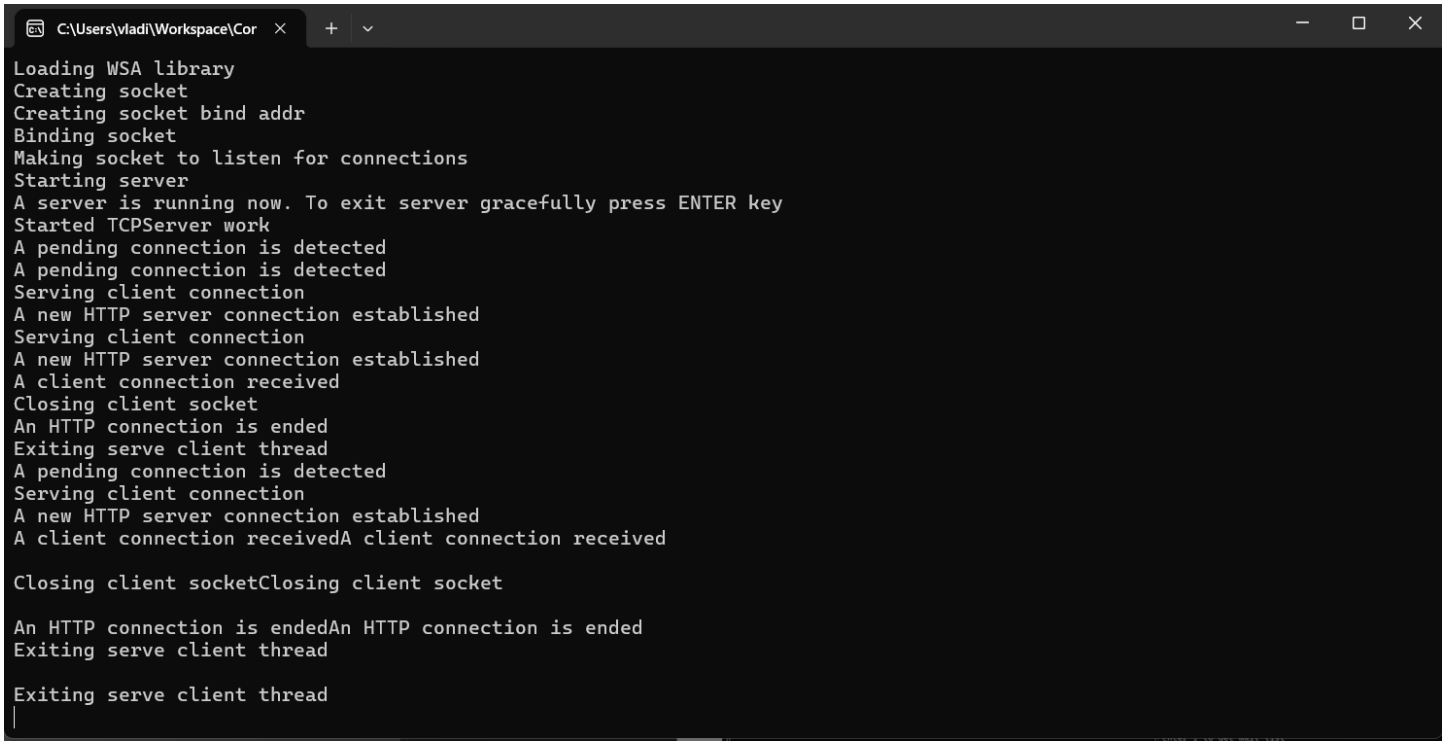
Важно отметить, что сайт-визитка содержится в едином файле, для создания полноценного веб-сервера в приложении необходимо автоматизировать эндпоинты, который раздают файлы и папки со статическими данными, а остальные эндпоинты использовать для редактирования состояния.

Можно также сервер «научить» использовать path-переменные из URN и многое другоею Полученная программа позволяет развернуть простой веб-сервер.

Вывод: в ходе лабораторной изучили протокол HTTP и составить программу согласно заданию.

Текст программ. Скриншоты программ.

Ссылка на репозиторий с кодом: https://github.com/IAmProgrammist/comp_net/tree/lab8




```cpp
#pragma once

#include <webstur/utils.h>
#include <webstur/ip/tcp/http/httprequest.h>
#include <webstur/ip/tcp/http/httpresponse.h>
#include <webstur/ip/tcp/http/httpmethod.h>

// Класс, описывающий эндпоинт. Задаёт паттерн пути а также метод
class DLLEXPORT HTTPEndpoint {
private:
    HTTPMethod method;
    std::string path;
public:
    // Возвращает true, если путь message соответствует паттерну эндпоинта
    bool matches(const HTTPRequest& message);

    // Конструктор-копия
    HTTPEndpoint(HTTPEndpoint& endpoint);

    // Конструктор, задающий паттерн и метод
    HTTPEndpoint(std::string path, HTTPMethod method);

    // Метод для обработки запроса.
    //
    // Внимание! Возвращаемый ответ должен быть
    // в heap (используйте new), память освободится автоматически
    // после отправки ответа.
    virtual HTTPResponse* process(const HTTPRequest& request) = 0;
protected:
};
```

```cpp
#pragma once

// Методы для HTTP сервера
enum HTTPMethod {
    GET,
    POST,
    PATCH,
    PUT,
```

```
    DEL,
    HEAD,
};
```

```
#pragma once
#include <map>
#include <webstur/utils.h>
#include <webstur/ip/tcp/http/httpmethod.h>

#define custom_min(a,b)            (((a) < (b)) ? (a) : (b))
#define custom_max(a,b)            (((a) > (b)) ? (a) : (b))

class DLLEXPORT HTTPRequest {
public:
    // Возвращает true, если запрос удалось корректно распознать
    bool isValid();

    // Конструирует HTTPRequest на основе полученного сообщения
    static HTTPRequest parse(const std::string& message);

    // Возвращает true, если путь в сообщении совпадает с паттерном
    bool matchesPath(const std::string& pattern) const;

    // Возвращает заголовки запроса
    const std::map<std::string, std::string>& getHeaders() const;

    // Возвращает тело запроса
    const std::string& getBody() const;

    // Возвращает метод
    HTTPMethod getMethod() const;
private:
    bool is_valid = true;
    HTTPMethod method;
    std::string query;
    std::map<std::string, std::string> headers;
    std::string body;

    HTTPRequest();
};
```

```
#pragma once

#include <map>
#include <sstream>
#include <string>
#include <webstur/utils.h>

#define HTTP_VERSION "HTTP/1.1"
#define SERVER_NAME "Webstur"

class DLLEXPORT HTTPResponse {
public:
    std::map<std::string, std::string> headers;
    int code;

    HTTPResponse(int code);
    virtual ~HTTPResponse();
    // Возвращает стрим для ответа
    virtual std::stringstream serialize() const;
};

class DLLEXPORT HTTPFileResponse : public HTTPResponse {
public:
```

```
    std::string file_path;
    std::string contents;

    HTTPFileResponse(int code, std::string file_path);
    virtual ~HTTPFileResponse();

    // Возвращает стрим для ответа
    std::stringstream serialize()  const;
};
```

```
// Простой HTTP-сервер

#pragma once

#include <vector>

#include <webstur/ip/tcp/http/httpendpoint.h>
#include <webstur/ip/tcp/tcpserver.h>

#define HTTP_DEFAULT_SERVER_PORT 80

class DLLEXPORT HTTPServer : public TCPServer {
private:
    std::vector<HTTPEndpoint*> endpoints;
    // Буфер накопления ответов с сервера
    std::map<SOCKET, std::string> buffers;
    std::map<SOCKET, std::size_t> content_lengths;
    std::map<SOCKET, std::size_t> header_end_poses;
public:
    HTTPServer(int port = HTTP_DEFAULT_SERVER_PORT);

    // Включает в список эндпоинтов новый эндпоинт
    void injectEndpoint(HTTPEndpoint& endpoint);
    // Метод, вызываемый при установлении соединения с клиентом
    void onConnect(SOCKET client);
    // Метод, вызываемый при разрыве соединения с клиентом
    // Переданный сокет уже не является действительным,
    // однако передаётся для отладочной информации
    void onDisconnect(SOCKET client);
    // Метод, вызываемый при разрыве общения с клиентом
    void onMessage(SOCKET client, const std::vector<char>& message);
    // Метод, вызываемый при получении валидного запроса.
    // выполняет роутинг запроса в нужный эндпоинт
    void onRequest(SOCKET client, HTTPRequest& request);
    // Метод, отправляющий сообщение и разрывающий соединение
    void sendMessageAndDisconnect(SOCKET client, const HTTPResponse& response);
};
```

```
#include "pch.h"
#include <webstur/ip/tcp/http/httprequest.h>
#include <webstur/ip/tcp/http/httpendpoint.h>

bool HTTPEndpoint::matches(const HTTPRequest& message) {
    // Если метод - HEAD или метод эндпоинта с запросом совпадают а также
    // расположение запроса совпадает с заданным
    return (message.getMethod() == HEAD || message.getMethod() == this->method)
        && message.matchesPath(this->path);
}

HTTPEndpoint::HTTPEndpoint(HTTPEndpoint& endpoint) {
    this->path = endpoint.path;
    this->method = endpoint.method;
}
```

```
HTTPEndpoint::HTTPEndpoint(std::string path, HTTPMethod method) : path(path), method(method) {};
```

```
#include "pch.h"

#include <webstur/ip/tcp/http/httprequest.h>

HTTPRequest::HTTPRequest() {}

HTTPRequest HTTPRequest::parse(const std::string& message) {
    HTTPRequest result;

    bool first_line_to_be_extracted = true;
    // Начало строчки
    size_t previous_break_line_pos = 0;
    // Конец строчки
    auto break_line_pos = message.find("\r\n");

    // Получить метод и запрос
    {
        // Разбить первую строчку
        auto splitted = split(message.substr(0, break_line_pos), " ");

        // В первой строчке должны быть как минимум два признака: метод и путь к методу
        if (splitted.size() < 2) {
            result.is_valid = false;
            return result;
        }

        // Сохраним имя метода и запрос
        auto method_name = splitted[0];
        if (method_name == "GET")
            result.method = HTTPMethod::GET;
        else if (method_name == "HEAD")
            result.method = HTTPMethod::HEAD;
        else if (method_name == "POST")
            result.method = HTTPMethod::POST;
        else if (method_name == "PATCH")
            result.method = HTTPMethod::PATCH;
        else if (method_name == "PUT")
            result.method = HTTPMethod::PUT;
        else if (method_name == "DELETE")
            result.method = HTTPMethod::DEL;
        else {
            result.is_valid = false;
            return result;
        }

        result.query = splitted[1];
    }

    auto message_size = message.size();

    // Считываем заголовки, до первого \r\n\r\n
    while (true) {
        if (break_line_pos == std::string::npos) {
            result.is_valid = false;
            return result;
        }
        previous_break_line_pos = break_line_pos + 2;
        break_line_pos = message.find("\r\n", previous_break_line_pos);

        if (break_line_pos - previous_break_line_pos <= 2) {
            break;
        }
```

```
        auto slice = std::string_view(message.c_str() + previous_break_line_pos,
            custom_min(break_line_pos, message_size) - previous_break_line_pos);
        auto space_index = slice.find(": ");
        auto header_name = slice.substr(0, space_index);
        auto header_val = slice.substr(space_index + 2);
        result.headers.insert(std::pair{ header_name, header_val });
    }

    // Остальное сохраняем в body
    result.body = message.substr(break_line_pos + 2);
    result.is_valid = true;

    return result;
}

bool HTTPRequest::matchesPath(const std::string& pattern) const {
    // TODO: сделать продвинутый распознаватель по паттерну, поддерживающий
    // PATH-переменные

    // Запрос должен начинаться с паттерна
    return this->query.starts_with(pattern);
}

const std::map<std::string, std::string>& HTTPRequest::getHeaders() const {
    return this->headers;
}

bool HTTPRequest::isValid() {
    return this->is_valid;
}

const std::string& HTTPRequest::getBody() const {
    return this->body;
}

HTTPMethod HTTPRequest::getMethod() const {
    return this->method;
}
```

```
#include "pch.h"

#include <iomanip>
#include <ctime>
#include <urlmon.h>
#include <filesystem>
#include <fstream>
#include <webstur/ip/tcp/http/httpresponse.h>

HTTPResponse::HTTPResponse(int code): code(code) {
    time_t     now = time(0);
    struct tm  tstruct;
    char       buf[80];
    localtime_s(&tstruct, &now);

    //strftime(buf, sizeof(buf), "%a, %d %b %Y %H:%M:%S", &tstruct);
    strftime(buf, sizeof(buf), "%c", &tstruct);

    // Сохранить в заголовки сервер, дату и тип соединения
    this->headers.insert({ "Server", SERVER_NAME });
    this->headers.insert({ "Date", buf });
    this->headers.insert({ "Connection", "Closed" });
}

std::stringstream HTTPResponse::serialize() const {
    // Записать версию HTTP, код
```

```
    std::stringstream output;
    output << HTTP_VERSION << " " << this->code << "\r\n";
    // Записать заголовки
    for (auto& header : this->headers) {
        output << header.first << ": " << header.second << "\r\n";
    }
    output << "\r\n";
    return output;
}

HTTPResponse::~HTTPResponse() {

}

HTTPFileResponse::HTTPFileResponse(int code, std::string file_path) : HTTPResponse(code),
file_path(file_path) {
    // Если файл существует
    std::ifstream in(this->file_path);
    if (in.is_open()) {
        // Считать содержимое файла, добавить MIME-тип и размер
        this->contents = std::string((std::istreambuf_iterator<char>(in)),
            std::istreambuf_iterator<char>());
        in.close();
        this->headers.insert({ "Content-Length",
            std::to_string(this->contents.size()) });
        this->headers.insert({ "Content-Type", mimeTypeFromString(file_path) });
    }
    else {
        // Установить код в 404 - файла не существует
        this->code = 404;
    }

    // Закрыть файл
    in.close();
}

std::stringstream HTTPFileResponse::serialize() const {
    // Дозаписать в ответ содержимое файла
    auto response = HTTPResponse::serialize();
    response << this->contents;

    return response;
}

HTTPFileResponse::~HTTPFileResponse() {
}
```

```
#include "pch.h"

#include <iostream>
#include <webstur/ip/tcp/http/httprequest.h>
#include <webstur/ip/tcp/http/httpserver.h>

HTTPServer::HTTPServer(int port) : TCPServer(port) {};

void HTTPServer::injectEndpoint(HTTPEndpoint& endpoint) {
    // Внести эндпоинт в список эндпоинтов
    this->endpoints.push_back(&endpoint);
}

void HTTPServer::onConnect(SOCKET client) {
    std::clog << "A new HTTP server connection established" << std::endl;

    // Внести временную информацию для сессии: буферы, длину контента, позицию конца заголовка
    this->buffers.insert(std::pair{client, ""});
```

```
    this->content_lengths.insert({ client, std::string::npos });
    this->header_end_poses.insert({ client, std::string::npos });
}

void HTTPServer::onDisconnect(SOCKET client) {
    std::clog << "An HTTP connection is ended" << std::endl;

    // Удалить временную информацию для сессии
    this->buffers.erase(this->buffers.find(client));
    this->content_lengths.erase(this->content_lengths.find(client));
    this->header_end_poses.erase(this->header_end_poses.find(client));
}

void HTTPServer::onMessage(SOCKET client, const std::vector<char>& message) {
    // Получить временную информацию для сессии
    std::string& buffer = this->buffers.find(client)->second;
    std::size_t& content_length = this->content_lengths.find(client)->second;
    std::size_t& header_end_pos = this->header_end_poses.find(client)->second;

    // Дополнить буфер
    buffer += std::string(message.begin(), message.end());

    // Если позиция конца заголовка ещё неизвестна
    if (header_end_pos == std::string::npos) {
        // Попытаться найти позицию конца заголовка
        header_end_pos = buffer.find("\r\n\r\n");

        // Если позиции не найдено, выйти из метода
        if (header_end_pos == std::string::npos)
            return;

        // Распарсить заголовок
        auto opt_parsed_header = HTTPRequest::parse(buffer.substr(0, header_end_pos + 4));

        // Если заголовок невалиден
        if (!opt_parsed_header.isValid()) {
            // Отправить запрос с кодом 422 и выйти
            HTTPResponse response_422 = HTTPResponse(422);
            sendMessageAndDisconnect(client, response_422);
            return;
        }

        // Постараться получить длину запроса
        auto headers = opt_parsed_header.getHeaders();
        auto content_length_it = headers.find("Content-Length");

        // Если длина запроса найдена
        if (content_length_it != headers.end()) {
            // Обновить длину заголовка для сессии, перезапустить метод onMessage
            // для обработки с установленным content_length на случай,
            // если клиент пришлёт тело с заголовками в одном сообщении
            content_length = std::atoi(content_length_it->second.c_str());
            std::vector<char> empty_string;
            this->onMessage(client, empty_string);
        }
        else {
            // Вызвать метод-колбек для запроса: если не установлена длина,
            // то тело запроса игнорируется
            onRequest(client, opt_parsed_header);
        }
    }
    else {
        // Если длина тела достигла Content-Length
        if (buffer.size() - header_end_pos + 4 >= content_length) {
            // Распарсить запрос
            auto request = HTTPRequest::parse(buffer);
```

```
            // Вызвать метод-колбек
            onRequest(client, request);
        }
    }
}

void HTTPServer::onRequest(SOCKET client, HTTPRequest& message) {
    HTTPEndpoint* endpoint = nullptr;

    // Найти эндпоинт, для которого удовлетворяет путь запроса
    for (auto& p_endpoint : this->endpoints) {
        if (p_endpoint->matches(message)) {
            endpoint = p_endpoint;
            break;
        }
    }

    // Если запрос не найден, возвращаем 404
    if (endpoint == nullptr) {
        HTTPResponse response_404 = HTTPResponse(404);
        sendMessageAndDisconnect(client, response_404);
        return;
    }

    // Если метод HEAD, возвращаем 200
    if (message.getMethod() == HEAD) {
        HTTPResponse response_200 = HTTPResponse(200);
        sendMessageAndDisconnect(client, response_200);
        return;
    }

    // Получить ответ от эндпоинта и отправить его
    auto response = endpoint->process(message);
    sendMessageAndDisconnect(client, *response);
    delete response;
}

void HTTPServer::sendMessageAndDisconnect(SOCKET client, const HTTPResponse& response) {
    // Сериализовать ответ
    auto response_stream = response.serialize();
    // Отправить ответ
    this->sendMessage(client, response_stream);
    // Разорвать соединение
    this->disconnect(client);
}
```

```
#pragma once

#include <webstur/ip/tcp/http/httpendpoint.h>

class CardEndpoint : public HTTPEndpoint {
public:
    CardEndpoint();

    HTTPResponse* process(const HTTPRequest& request);
};
```

```
#include "cardendpoint.h"

// Сконструировать эндпоинт по пути /my-card
CardEndpoint::CardEndpoint() : HTTPEndpoint("/my-card", GET) {};

HTTPResponse* CardEndpoint::process(const HTTPRequest& request) {
    // Выдать файл-визитку
```

```
    HTTPResponse* answer = new HTTPFileResponse(200, ".\\assets\\index.html");

    return answer;
}
```

```
#include <iostream>
#include <algorithm>
#include <ws2tcpip.h>
#include <webstur/ip/tcp/http/httpserver.h>
#include "endpoints/cardendpoint.h"

int main() {
    try {
        // Загрузить библиотеки WSA
        IServer::init();

        // Инициализировать сервер и эндпоинты
        CardEndpoint endpoint;
        auto server = HTTPServer();
        server.injectEndpoint(endpoint);

        // Запустить сервер
        server.start();

        std::cout << "A server is running now. To exit server gracefully press ENTER key" <<
std::endl;
        // При любом вводе пользователя приостановить выполнение программы
        std::cin.ignore();
    }
    catch (const std::runtime_error& error) {
        std::cerr << "Failed while running server. Caused by: '" << error.what() << "'" <<
std::endl;

        return -1;
    }

    // Выгрузка библиотеки WSA
    IServer::detach();

    return 0;
}
```

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Моя визитка</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            line-height: 1.6;
            margin: 0;
            padding: 0;
            color: #333;
            background-color: #f5f5f5;
        }
        header {
            background-color: #2c3e50;
            color: white;
            padding: 20px 0;
            text-align: center;
        }
        .container {
```

```
            width: 80%;
            margin: 20px auto;
            background: white;
            padding: 20px;
            box-shadow: 0 0 10px rgba(0,0,0,0.1);
        }
        h1 {
            margin: 0;
        }
        h2 {
            color: #2c3e50;
            border-bottom: 2px solid #eee;
            padding-bottom: 10px;
        }
        .contact-info {
            margin: 20px 0;
        }
        .skills {
            display: flex;
            flex-wrap: wrap;
        }
        .skill {
            background: #eee;
            padding: 5px 10px;
            margin: 5px;
            border-radius: 3px;
        }
        footer {
            text-align: center;
            padding: 20px;
            background: #2c3e50;
            color: white;
        }
    </style>
</head>
<body>
    <header>
        <h1>Владислав Пахомов</h1>
        <p>Разработчик программных решений</p>
    </header>

    <div class="container">
        <section>
            <h2>Обо мне</h2>
            <p>Привет! Я разработчик, имеющий профессиональный опыт в 1 год, специализируюсь на
решении большого спектра задач и готов к изучению новых сфер.</p>
        </section>

        <section>
            <h2>Навыки</h2>
            <div class="skills">
                <span class="skill">Веб (React, Redux, CSS, MUI, Vue.js)</span>
                <span class="skill">Бэкенд (FastAPI, Django)</span>
                <span class="skill">Десктоп UI (Qt, JavaFX)</span>
                <span class="skill">Android</span>
                <span class="skill">Git</span>
                <span class="skill">Низкоуровневое программирование (Rust, C++,
C)</span>                        <span class="skill">Программирование графических адаптеров (AMD
HIP, OpenCL, OpenGL)</span>
        <span class="skill">Контейнеризация и настройка CI (Docker, Docker Compose, GitLab CI/CD,
GitHub Actions CI/CD)</span>
            </div>
        </section>

        <section>
            <h2>Опыт работы</h2>
            <h3>Разработчик программных решений в компании "Антара"</h3>
```

```
            <p><em>2024 - настоящее время</em></p>
            <ul>
                <li>Разработка и поддержка клиентских веб-приложений</li>
                <li>Поддержка графического пакета</li>
        <li>Работа в команде над крупными проектами</li>
            </ul>

        </section>

        <section class="contact-info">
            <h2>Контакты</h2>
            <p>Email: <a href="mailto:vladislav.pakhomov@bk.ru">vladislav.pakhomov@bk.ru</a></p>
            <p>Телефон: +7 (950) 715-24-96</p>
            <p>GitHub: <a
href="https://github.com/IAmProgrammist">https://github.com/IAmProgrammist</a></p>
        </section>
    </div>

    <footer>
        <p>&copy; 2025 Владислав Пахомов. Have fun.</p>
    </footer>
</body>
</html>
```