

TCP 기반 소켓프로그래밍

소프트웨어학부 20223103 양나래

목차

1. 과제 내용
 2. 개발 환경
 3. HttpClient.java 코드 설명
 4. HttpServer.java 코드 설명
 5. 실행 결과
-
1. 과제 내용
 - A. 소켓 통신을 활용하여 Server, Client 프로그램 작성
 - B. TCP 기반 소켓프로그래밍 작성 후 Client에서는 HTTP 프로토콜의 GET/HEAD/POST/PUT/DELETE Request를 요청하여 Server에서는 Client의 Request에 따라 응답 메시지를 구성하여 Response하도록 구현
 2. 개발 환경
 - A. 사용 언어: JAVA
 - B. IDE: Visual Studio Code
 - C. 개발 운영체제: Mac OS

3. HttpClient.java 코드 설명

```
String hostname = "localhost";
int port = 7070;
```

- hostname과 port를 지정해서 접속할 서버 정보 설정

```
try (Socket socket = new Socket(hostname, port)) {
    // 하나의 소켓에서 지속 연결을 이용하기 위해 PrintWriter와 BufferedReader 생성
    PrintWriter writer = new PrintWriter(socket.getOutputStream(), autoFlush:true);
    BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
}
```

- 서버와 소켓 연결을 맺음
- 요청 전송용으로 PrintWriter, 응답 수신용으로 BufferedReader 준비

```
// POST /users: 사용자 추가하기 -> 성공
sendPostRequest(writer, reader, hostname, name:"bae", address:"Seoul", tel:"010-0101-0101", age:"10");
System.out.println();
sendPostRequest(writer, reader, hostname, name:"lee", address:"Busan", tel:"010-1212-1212", age:"21");
System.out.println();
sendPostRequest(writer, reader, hostname, name:"kang", address:"Seoul", tel:"010-2323-2323", age:"22");
System.out.println();

// GET /users: 전체 사용자 조회하기 -> 성공
sendGetAllRequest(writer, reader, hostname);
System.out.println();

// GET /users?key=value: 조건으로 검색하기 -> 성공
sendGetRequestWithKeyValue(writer, reader, hostname, key:"name", value:"kim");
System.out.println();
sendGetRequestWithKeyValue(writer, reader, hostname, key:"address", value:"Jeju");
System.out.println();
sendGetRequestWithKeyValue(writer, reader, hostname, key:"tel", value:"010-2222-2222");
System.out.println();
sendGetRequestWithKeyValue(writer, reader, hostname, key:"age", value:"22");
System.out.println();

// HEAD /users: 전체 사용자 조회에 대한 HEAD 요청하기 -> 성공
sendHeadRequest(writer, reader, hostname, path:"/users");
System.out.println();

// HEAD /users?key=value: 조건 검색에 대한 HEAD 요청하기 -> 성공
sendHeadRequest(writer, reader, hostname, path:"/users?name=kim");
System.out.println();
```

```

// PUT /users: id에 해당되는 유저 정보 수정하는 PUT 요청하기 -> 성공
sendPutRequest(writer, reader, hostname, id:"98e59ac7-c8ec-4f9f-9556-7a9de227e289", name:"yang", address:"Seoul", tel:"010-9999-9999", age:"23");
System.out.println();

// PUT /users: 존재하지 않는 id로 유저 정보를 수정하려고 할 때 -> 404 Not Found
sendPutRequest(writer, reader, hostname, id:"abcdefg", name:"hwang", address:"Gangwon", tel:"010-1234-5678", age:"100");
System.out.println();

// DELETE /users/{id}: id에 해당되는 유저를 삭제하는 DELETE 요청하기 -> 성공
sendDeleteRequest(writer, reader, hostname, id:"314b5ba8-6e32-4b5b-adef-a7a715e0ae0f");
System.out.println();

// GET /users: 사용자 수정, 삭제가 잘 되었는지 확인하기 위한 전체 사용자 조회 -> 성공
send GetAllRequest(writer, reader, hostname);
System.out.println();

// 잘못된 GET 쿼리: 빈 쿼리 -> 400 Bad Request
sendCustomRequest(writer, reader, hostname, requestLine:"GET /users? HTTP/1.1");
System.out.println();

// 지원되지 않는 POST 경로: POST /invalid-> 400 Bad Request
sendCustomRequest(writer, reader, hostname, requestLine:"POST /invalid HTTP/1.1");
System.out.println();

// 지원되지 않는 메서드: PATCH /users -> 405 Method Not Allowed
sendCustomRequest(writer, reader, hostname, requestLine:"PATCH /users HTTP/1.1");
System.out.println();

System.out.println(x:"요청 전송 완료");

```

- POST, GET, HEAD, PUT, DELETE 등의 메소드를 정의된 함수 호출을 통해 전송, 응답받은 결과를 출력
- 세 번의 sendPostRequest로 세 명의 유저를 추가
- send GetAllRequest로 전체 유저 조회
- sendGetRequestWithValue로 key에 해당하는 값이 value인 유저를 조회 -> 총 네 번의 조회
- sendHeadRequest를 통해 HEAD /users 요청으로 전체 유저를 조회한 헤더만 응답 받기
- sendHeadRequest를 통해 HEAD /users?name=kim 요청으로 name이 kim인 유저를 조회한 헤더만 응답 받기
- sendPutRequest에서 수정하고 싶은 유저의 id로 해당 유저의 정보 수정하기
- sendPutRequest 요청할 때 전달한 id에 해당하는 유저가 없으면 404 Not Found를 응답 받음
- sendDeleteRequest로 DELETE /users/{id} 요청으로 id에 해당하는 유저 삭제
- 수정한 결과를 확인하기 위해 send GetAllRequest로 전체 유저 다시 한번 조회
- sendCustomRequest를 통해 GET /users? 요청을 보냄. 쿼리 파라미터가 비어있기 때문에 400 Bad Request 응답을 받음
- sendCustomRequest를 통해 POST /invalid 요청을 보냄. 지원되지 않는 경로로 요청을 보냈기 때문에 400 Bad Request 응답을 받음

- `sendCustomRequest`를 통해 PATCH /users 요청을 보냄. 지원되지 않는 메소드 요청을 보냈기 때문에 405 Method Not Allowed 응답을 받음

```
// USER 생성 POST 요청을 보내는 함수
public static void sendPostRequest(PrintWriter writer, BufferedReader reader, String hostname, String name, String address, String tel, String age) throws IOException{
    String id = UUID.randomUUID().toString();

    // JSON 형식 문자열 생성
    String body = String.format(
        format "{\"id\":\"%s\", \"name\":\"%s\", \"address\":\"%s\", \"tel\":\"%s\", \"age\":\"%s\"}",
        id, name, address, tel, age
    );

    // POST 요청 작성
    writer.println(x:"POST /users HTTP/1.1");
    writer.println("Host: " + hostname);
    writer.println(x:"Content-Type: application/json");
    writer.println("Content-Length: " + body.length());
    writer.println(x:"Connection: keep-alive");
    writer.println(); // 헤더와 본문 사이의 빈 줄
    writer.print(body);
    writer.flush();

    // RESPONSE 읽기
    System.out.println(x:[POST /users HTTP/1.1] 응답: ");
    readResponse(reader);
}
```

- `randomUUID()`를 통해 유저의 id 생성
- json 형태로 요청할 body 생성
- HTTP/1.1 POST 요청 라인과 Host, Content-Type, Content-Length, Connection 정보를 가진 헤더 작성
- 빈 줄로 헤더 끝을 표시한 후 body 추가
- `writer.flush()`로 전송
- `readResponse` 함수로 응답받은 내용 출력

```
// ALL USERS를 GET하는 요청을 보내는 함수
public static void sendGetAllRequest(PrintWriter writer, BufferedReader reader, String hostname) throws IOException{
    // GET 요청 작성
    writer.println(x:"GET /users HTTP/1.1");
    writer.println("Host: " + hostname);
    writer.println(x:"Connection: keep-alive");
    writer.println();

    // RESPONSE 읽기
    System.out.println(x:[GET /users HTTP/1.1] 응답: ");
    readResponse(reader);
}
```

- 전체 유저 목록 조회 요청을 보내는 함수
- Host, Connection 정보를 가진 헤더
- 본문 없이 헤더만 전송
- 빈 줄로 헤더 끝을 표시
- `readResponse` 함수로 응답받은 내용 출력

```

// Key와 Value를 통해 조건에 맞는 USER를 GET하는 요청을 보내는 함수
public static void sendGetRequestWithKeyValue(PrintWriter writer, BufferedReader reader, String hostname, String key, String value) throws IOException{
    // GET 요청 작성
    String requestLine = String.format(format:"GET /users?%s=%s HTTP/1.1", key, value);
    writer.println(requestLine);
    writer.println("Host: " + hostname);
    writer.println(x:"Connection: keep-alive");
    writer.println(); // 헤더와 본문 사이의 빈 줄

    // RESPONSE 읽기
    System.out.println("[ " + requestLine + " ] 응답: ");
    readResponse(reader);
}

```

- 특정한 조건의 유저를 요청하는 함수
- 검색할 조건으로 쿼리 파라미터 작성
- Host, Connection 정보를 가진 헤더
- 빈 줄로 헤더 끝을 표시
- readResponse 함수로 응답받은 내용 출력

```

// 임의의 HTTP 요청을 보내는 함수 (그 외의 케이스 테스트용)
public static void sendCustomRequest(PrintWriter writer, BufferedReader reader, String hostname, String requestLine) throws IOException{
    // HTTP 요청 작성
    writer.println(requestLine);
    writer.println("Host: " + hostname);
    writer.println(x:"Connection: keep-alive");
    writer.println();

    // RESPONSE 읽기
    System.out.println("[ " + requestLine + " ] 응답: ");
    readResponse(reader);
}

```

- 다양한 응답을 위한 HTTP 요청에 쓰이는 함수
- requestLine을 통해 임의의 요청 라인 사용
- Host, Connection 정보를 가진 헤더
- 빈 줄로 헤더 끝을 표시
- readResponse 함수로 응답받은 내용 출력

```

// HEAD 요청을 보내는 함수
public static void sendHeadRequest(PrintWriter writer, BufferedReader reader, String hostname, String path) throws IOException{
    // HEAD 요청 작성
    writer.println("HEAD " + path + " HTTP/1.1");
    writer.println("Host: " + hostname);
    writer.println(x:"Connection: keep-alive");
    writer.println(); // 헤더와 본문 사이의 빈 줄

    // RESPONSE 읽기
    System.out.println("[HEAD " + path + " HTTP/1.1] 응답: ");
    readHeadResponse(reader);
}

```

- HEAD 요청을 보내는 함수
- Host, Connection 정보를 가진 헤더
- readHeadResponse 함수로 응답받은 내용 출력
- 빈 줄로 헤더 끝을 표시
-

```

// PUT 요청을 보내는 함수
public static void sendPutRequest(PrintWriter writer, BufferedReader reader, String hostname, String id, String name, String address, String tel, String age) throws IOException{
    // 수정할 사용자 정보를 포함하는 JSON 문자열 생성
    // id를 포함해야 합니다.
    String body = String.format(
        "{'id': \"%s\", \"name\": \"%s\", \"address\": \"%s\", \"tel\": \"%s\", \"age\": \"%s\"}",
        id, name, address, tel, age
    );

    // PUT 요청 작성
    writer.println("PUT /users HTTP/1.1");
    writer.println("Host: " + hostname);
    writer.println("Content-Type: application/json");
    writer.println("Content-Length: " + body.length());
    writer.println("Connection: keep-alive");
    writer.println(); // 헤더와 본문 사이의 빈 줄
    writer.print(body);
    writer.flush();

    // RESPONSE 읽기
    System.out.println("[PUT /users HTTP/1.1] 응답: ");
    readResponse(reader);
}

```

- PUT 요청을 보내는 함수
- 수정할 사용자 id를 포함해서 수정할 내용을 json 형식으로 body 작성
- Host, Content-Type, Content-Length, Connection 정보를 가진 헤더 작성
- 빈 줄로 헤더 끝을 표시한 후 body 추가
- writer.flush()로 전송
- readResponse 함수로 응답받은 내용 출력

```

// DELETE 요청을 보내는 함수
public static void sendDeleteRequest(PrintWriter writer, BufferedReader reader, String hostname, String id) throws IOException {
    String path = "/users/" + id;

    // DELETE 요청 작성
    writer.println("DELETE " + path + " HTTP/1.1");
    writer.println("Host: " + hostname);
    writer.println("Connection: keep-alive");
    writer.println(); // 빈 줄

    //RESPONSE 읽기
    System.out.println("[DELETE " + path + " HTTP/1.1] 응답:");
    readResponse(reader);
}

```

- DELETE 요청을 보내는 함수
- /users/{id} 로 path에 삭제 원하는 id 전달
- Host, Content-Type, Connection 정보를 가진 헤더 작성
- 빈 줄로 헤더 끝을 표시

```

// RESPONSE 읽는 함수
public static void readResponse(BufferedReader reader) throws IOException {
    // 상태 라인과 헤더 읽기
    String statusLine = reader.readLine();
    if (statusLine == null) return;
    System.out.println(statusLine);

    String line;
    int contentLength = 0;
    while ((line = reader.readLine()) != null && !line.isEmpty()) {
        System.out.println(line);
        if (line.toLowerCase().startsWith(prefix:"content-length:")) {
            try {
                contentLength = Integer.parseInt(line.split(regex":")[1].trim());
            } catch(NumberFormatException e) {
                contentLength = 0;
            }
        }
    }

    System.out.println();

    // 빈 줄 후 본문 읽기 : exactly contentLength 바이트 읽기
    char[] bodyChars = new char[contentLength];
    int totalRead = 0;
    while(totalRead < contentLength) {
        int read = reader.read(bodyChars, totalRead, contentLength - totalRead);
        if (read == -1) break;
        totalRead += read;
    }
    String responseBody = new String(bodyChars, offset:0, totalRead);
    System.out.println(responseBody);
}

```

- Response 읽는 함수
- 상태라인을 읽음
- 파싱을 통해 헤더를 읽음
- 헤더에서 content-length를 얻음
- 헤더는 빈 줄로 끝나기 때문에 빈 줄이 나오기 전까지 읽음
- 빈 줄 후 body를 헤더의 content-length의 크기만큼 읽음

```

// HEAD RESPONSE 읽는 함수
public static void readHeadResponse(BufferedReader reader) throws IOException {
    // 상태 라인 읽기
    String statusLine = reader.readLine();
    if (statusLine == null) return;
    System.out.println(statusLine);

    // 헤더 읽기
    String line;
    while ((line = reader.readLine()) != null && !line.isEmpty()) {
        System.out.println(line);
    }
}

```

- 본문이 없는 HEAD 요청의 Response를 읽는 함수
- 헤더만 읽음
- 헤더는 빈 줄로 끝나기 때문에 빈 줄이 나오기 전까지 읽음

4. HttpServer.java 코드 설명

```
private static final int PORT = 7070;
```

- 실행할 PORT 설정

```

try (ServerSocket serverSocket = new ServerSocket(PORT)) {
    System.out.println(PORT + "번 포트에서 서버 실행되는 중");

    while (true) {
        Socket socket = serverSocket.accept();
        new Thread(() -> handleClient(socket)).start();
    }
} catch (IOException e) {
    e.printStackTrace();
}

```

- 서버를 시작하고 연결 대기 시작
- serverSocket 생성
- 무한 루프에서 클라이언트 연결 수락 -> 스레드로 처리

```

private static void handleClient(Socket socket) {
    try {
        BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter writer = new PrintWriter(socket.getOutputStream(), autoFlush:true)
    } {
        // 지속 연결
        while (true) {
            // Request 첫 줄 읽기
            String request = br.readLine();
            if (request == null) break;
            if (request.isEmpty()) continue;

            System.out.println("클라이언트 요청: " + request);
            String[] part = request.split(regex:" ");
            String method = part[0];
            String path = part[1];

            int contentLength = 0;
            String line;

            // 헤더 읽기
            while ((line = br.readLine()) != null && !line.isEmpty()) {
                if (line.toLowerCase().startsWith(prefix:"content-length:")) {
                    try {
                        contentLength = Integer.parseInt(line.split(regex:" ")[1].trim());
                    } catch (NumberFormatException e) {
                        contentLength = 0;
                    }
                }
            }
        }
    }
}

```

- 클라이언트의 요청이 올 때마다 Request의 첫 줄 읽어서 method와 path 파악
- 헤더는 빈 줄로 끝나기 때문에 빈 줄이 나오기 전까지 읽음
- 헤더에서 content-length를 얻음

```

// GET USERS BY KEY AND VALUE REQUEST
if (method.equals(anObject:"GET") && path.startsWith(prefix:"/users?")) {
|   handleGetUsersByKeyValue(writer, path);
}

// GET ALL USERS REQUEST
else if (method.equals(anObject:"GET") && path.equals(anObject:"/users")) {
|   handleGetAllUsers(writer);
}

// GET (BAD REQUEST)
else if (method.equals(anObject:"GET")) {
|   sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"지원하지 않는 요청입니다.");
}

// HEAD 쿼리 파라미터가 포함된 경우 REQUEST
else if (method.equals(anObject:"HEAD") && path.startsWith(prefix:"/users?")) {
|   handleHeadUsersByKeyValue(writer, path);
}

// HEAD 전체 사용자 조회 REQUEST
else if (method.equals(anObject:"HEAD") && path.equals(anObject:"/users")) {
|   handleHeadAllUsers(writer);
}

// POST (BAD REQUEST)
else if (method.equals(anObject:"POST") && !path.equals(anObject:"/users")) {
|   sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"지원하지 않는 요청입니다.");
}

// POST USER REQUEST
else if (method.equals(anObject:"POST") && path.equals(anObject:"/users")) {
|   handlePostUser(br, writer, contentLength);
}

// PUT 사용자 정보 수정 REQUEST
else if (method.equals(anObject:"PUT") && path.equals(anObject:"/users")) {
|   handlePutUser(br, writer, contentLength);
}

// DELETE USER REQUEST
else if (method.equals(anObject:"DELETE") && path.startsWith(prefix:"/users/")) {
|   String id = path.substring("/users/.length());
|   handleDeleteUser(writer, id);
}

// NOT ALLOWED METHOD
else {
|   sendHttpResponse(writer, statusCode:405, statusText:"Method Not Allowed", body:"지원되지 않는 메서드입니다.");
}

```

- 요청에서 얻은 method와 path를 통해 알맞은 handle 함수를 호출, 응답을 보냄
- 사용되지 않는 메서드 요청이 왔을 때는 바로 sendHttpResponse 함수를 호출

```
// HTTP 응답 처리 함수 (본문 전송)
private static void sendHttpServletResponse(PrintWriter writer, int statusCode, String statusText, String body) {
    String date = DateTimeFormatter.RFC_1123_DATE_TIME.format(ZonedDateTime.now(ZoneId.of("GMT")));
    writer.println("HTTP/1.1 " + statusCode + " " + statusText);
    writer.println("Date: " + date);
    writer.println("Content-Type: text/plain");
    writer.println("Content-Length: " + body.length());
    writer.println(); // 헤더와 본문 사이 빈 줄
    writer.println(body);
}
```

- 본문도 함께 전송하는 메소드들에서 사용하는 응답 전송 함수
- 상태줄, Date, Content-Type, Content-Length 정보를 가진 헤더
- 헤더와 본문 사이에 빈 줄 추가
- 응답 바디 추가

```
// HEAD 응답 처리 함수 (본문은 보내지 않고 헤더만 전송)
private static void sendHttpHeadResponse(PrintWriter writer, int statusCode, String statusText, String body) {
    String date = DateTimeFormatter.RFC_1123_DATE_TIME.format(ZonedDateTime.now(ZoneId.of("GMT")));
    writer.println("HTTP/1.1 " + statusCode + " " + statusText);
    writer.println("Date: " + date);
    writer.println("Content-Type: text/plain");
    writer.println("Content-Length: " + body.length());
    writer.println(); // 헤더와 본문 사이 빈 줄
    // HEAD 방식은 본문을 전송하지 않음
}
```

- 본문을 전송하지 않는 HEAD 요청의 응답을 보내는 함수
- Date, Content-Type, Content-Length 정보를 가진 헤더
- 빈 줄로 헤더 끝을 나타냄

```

// GET /users?key=value 요청 처리 함수
private static void handleGetUsersByKeyValue(PrintWriter writer, String path) {
    // ex) /users?name=kim
    String query = path.substring(path.indexOf(str:"?") + 1).trim();

    // 쿼리 파라미터가 없을 때 400 Bad Request
    if(query.isEmpty()){
        sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"쿼리 파라미터가 없습니다.");
        return;
    }

    // "="를 기준으로 최대 2부분만 분리 (ex) name=kim
    String[] param = query.split(regex: "=", limit:2);

    // 잘못된 쿼리 파라미터일 때 400 Bad Request
    if (param.length != 2 || param[0].trim().isEmpty() || param[1].trim().isEmpty()) {
        sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"잘못된 쿼리 파라미터입니다.");
        return;
    }

    String key = param[0].trim();
    String value = param[1].trim();

    File file = new File(pathname:"users.txt");

    // file이 존재하지 않을 때 404 Not Found
    if (!file.exists()) {
        sendHttpResponse(writer, statusCode:404, statusText:"Not Found", body:"users.txt 파일을 찾을 수 없습니다.");
        return;
    }

    StringBuilder matchedUsers = new StringBuilder();
    try (BufferedReader fileReader = new BufferedReader(new FileReader(file))) {
        String fileLine;
        while ((fileLine = fileReader.readLine()) != null) {
            // 단순 문자열 포함 검사: "key": "value" 형태로 검색
            if (fileLine.contains(key + ":" + value) && fileLine.contains(value + "\n")) {
                matchedUsers.append(fileLine).append(str:"\\n");
            }
        }
    } catch (IOException e) {
        sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일 처리 중 오류가 발생했습니다.");
        return;
    }

    // 해당하는 유저가 없을 때 404 Not Found
    if (matchedUsers.length() == 0) {
        sendHttpResponse(writer, statusCode:404, statusText:"Not Found", body:"일치하는 사용자를 찾을 수 없습니다.");
        return;
    } else {
        sendHttpResponse(writer, statusCode:200, statusText:"OK", matchedUsers.toString());
        return;
    }
}

```

- 요청 쿼리 파라미터에서 쿼리를 읽어냄
- 만약 쿼리가 없다면 400 응답을 보냄
- 잘못된 쿼리 파라미터일 때는 400 응답을 보냄
- 쿼리에서 검색할 파라미터를 리스트로 저장
- File이 존재하지 않을 때 404 응답을 보냄
- key와 value를 통해 users.txt에서 찾아냄
- 중간에 에러가 발생하면 500 응답을 보냄
- 검색 조건에 맞는 유저가 없을 때 404 응답을 보냄

- 검색 조건에 맞는 유저가 있으면 그 유저들을 보냄

```
// GET /users 전체 조회 요청 처리 함수
private static void handleGetAllUsers(PrintWriter writer) {
    File file = new File(pathname:"users.txt");
    if (file.exists() && file.isFile()) {
        if (file.canRead()) {
            try {
                String content = Files.readString(file.toPath());
                sendHttpResponse(writer, statusCode:200, statusText:"OK", content);
                return;
            } catch (IOException e) {
                sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일을 읽는 중에 오류가 발생했습니다.");
                return;
            }
        } else {
            sendHttpResponse(writer, statusCode:403, statusText:"Forbidden", body:"users.txt 파일에 접근할 권한이 없습니다.");
            return;
        }
    } else {
        // file이 존재하지 않을 때 404 Not Found
        sendHttpResponse(writer, statusCode:404, statusText:"Not Found", body:"users.txt 파일을 찾을 수 없습니다.");
        return;
    }
}
```

- 전체 사용자를 조회하는 함수
- 중간에 에러가 발생하면 500 응답을 보냄
- 파일을 읽을 수 없으면 403 응답을 보냄
- 파일이 존재하지 않으면 404 응답을 보냄
- 파일을 읽으면 파일의 내용을 보냄

```
// POST /users 요청 처리 함수 (사용자 추가)
private static void handlePostUser(BufferedReader br, PrintWriter writer, int contentLength) {
    // 요청 본문이 비어있을 때 400 Bad Request
    if (contentLength <= 0) {
        sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"요청 본문이 비어 있습니다.");
        return;
    }

    char[] bodyChars = new char[contentLength];
    try {
        int readChars = br.read(bodyChars, off:0, contentLength);
        // 요청 본문 읽은 것과 contentLength가 다를 때 400 Bad Request
        if (readChars != contentLength) {
            sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"요청 본문을 완전히 읽지 못했습니다.");
            return;
        }

        String body = new String(bodyChars);
        try {
            Files.write(Paths.get(first:"users.txt"), (body + "\n").getBytes(),
                StandardOpenOption.CREATE, StandardOpenOption.APPEND);
            sendHttpResponse(writer, statusCode:201, statusText:"Created", body:"사용자 정보가 저장되었습니다.");
            return;
        } catch (IOException e) {
            sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일 저장 중 오류가 발생했습니다.");
            return;
        }
    } catch (IOException e) {
        sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"요청 본문을 읽는 중 오류가 발생했습니다.");
        return;
    }
}
```

- 사용자를 추가하는 함수

- 요청 본문이 비어있으면 400 응답을 보냄
- 요청 본문의 길이와 헤더의 Content-Length와 다를 때 400 응답 보냄
- 요청 본문을 users.txt 파일에 저장함
- 성공적으로 저장되면 200 응답을 보냄
- 중간에 에러가 발생하면 500 응답을 보냄

```

// HEAD /users?key=value 요청 처리 함수 (헤더만 전송)
private static void handleHeadUsersByKeyValue(PrintWriter writer, String path) {
    String query = path.substring(path.indexOf(str:"?") + 1).trim();

    // 쿼리 파라미터가 없을 때 400 Bad Request
    if(query.isEmpty()){
        sendHttpHeadResponse(writer, statusCode:400, statusText:"Bad Request", body:"쿼리 파라미터가 없습니다.");
        return;
    }

    String[] param = query.split(regex:"=", limit:2);

    // 잘못된 쿼리 파라미터일 때 400 Bad Request
    if (param.length != 2 || param[0].trim().isEmpty() || param[1].trim().isEmpty()) {
        sendHttpHeadResponse(writer, statusCode:400, statusText:"Bad Request", body:"잘못된 쿼리 파라미터입니다.");
        return;
    }

    String key = param[0].trim();
    String value = param[1].trim();

    File file = new File(pathname:"users.txt");

    //file이 존재하지 않을 때 404 Not Found
    if (!file.exists()) {
        sendHttpHeadResponse(writer, statusCode:404, statusText:"Not Found", body:"");
        return;
    }

    StringBuilder matchedUsers = new StringBuilder();
    try (BufferedReader fileReader = new BufferedReader(new FileReader(file))) {
        String fileLine;
        while ((fileLine = fileReader.readLine()) != null) {
            if (fileLine.contains(key + ":" + value)) {
                matchedUsers.append(fileLine).append(str:"\n");
            }
        }
    } catch (IOException e) {
        sendHttpHeadResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"");
        return;
    }

    //해당하는 유저가 없을 때 404 Not Found
    if (matchedUsers.length() == 0) {
        sendHttpHeadResponse(writer, statusCode:404, statusText:"Not Found", body:"");
        return;
    } else {
        sendHttpHeadResponse(writer, statusCode:200, statusText:"OK", matchedUsers.toString());
        return;
    }
}

```

- 조건에 맞는 유저를 조회하는 GET 함수와 동일하게 동작
- 쿼리 파라미터가 없으면 400 응답을 보냄
- 쿼리 파라미터가 잘못됐으면 400 응답을 보냄
- 쿼리 파라미터에서 key와 value를 얻음
- 파일이 존재하지 않으면 404 응답을 보냄
- 중간에 에러가 발생하면 500 응답을 보냄
- 조건에 해당하는 유저가 없으면 404 응답을 보냄
- 조건에 해당하는 유저를 sendHttpHeadResponse 함수를 통해 보냄. 이때 해당하는 유저를 보내는 이유는 헤더에 응답의 길이는 포함하기 때문

```
// HEAD /users 전체 조회 처리 함수 (헤더만 전송)
private static void handleHeadAllUsers(PrintWriter writer) {
    File file = new File(pathname:"users.txt");
    if (file.exists() && file.isFile()) {
        if (file.canRead()) {
            try {
                String content = Files.readString(file.toPath());
                sendHttpHeadResponse(writer, statusCode:200, statusText:"OK", content);
                return;
            } catch (IOException e) {
                sendHttpHeadResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일을 읽는 중에 오류가 발생했습니다.");
                return;
            }
        } else {
            sendHttpHeadResponse(writer, statusCode:403, statusText:"Forbidden", body:"users.txt 파일에 접근할 권한이 없습니다.");
            return;
        }
    } else {
        // file이 존재하지 않을 때 404 Not Found
        sendHttpHeadResponse(writer, statusCode:404, statusText:"Not Found", body:"users.txt 파일을 찾을 수 없습니다.");
        return;
    }
}
```

- 전체 유저를 조회하는 GET 함수와 동일하게 동작
- 파일에 접근하지 못하면 403 응답을 보냄
- 파일이 존재하지 않으면 404 응답을 보냄
- 파일을 정상적으로 읽으면 해당 내용을 sendHttpHeadResponse 함수를 통해 보냄. 헤더에 응답의 길이를 포함하기 때문에 content도 함수에 전달

```
// PUT /user 요청 처리 함수
private static void handlePutUser(BufferedReader br, PrintWriter writer, int contentLength) {
    // 요청 본문이 비어있을 때 400 Bad Request
    if (contentLength <= 0) {
        sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"요청 본문이 비어 있습니다.");
        return;
    }

    char[] bodyChars = new char[contentLength];
    try {
        int readChars = br.read(bodyChars, off:0, contentLength);
        // 요청 본문 읽은 것과 contentLength가 다를 때 400 Bad Request
        if (readChars != contentLength) {
            sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"요청 본문을 완전히 읽지 못했습니다.");
            return;
        }
    } catch (IOException e) {
        sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"요청 본문을 읽는 중 오류 발생");
        return;
    }
    String body = new String(bodyChars);

    // JSON 본문에서 id 추출 ("id":"값" 형태)
    String id = extractValueFromJson(body, key:"id");
    // id가 없을 때 400 Bad Request
    if (id == null || id.isEmpty()) {
        sendHttpResponse(writer, statusCode:400, statusText:"Bad Request", body:"요청 본문에 id가 없습니다.");
        return;
    }

    File file = new File(pathname:"users.txt");
    // file이 존재하지 않을 때 404 Not Found
    if (!file.exists()) {
        sendHttpResponse(writer, statusCode:404, statusText:"Not Found", body:"users.txt 파일을 찾을 수 없습니다.");
        return;
    }
}
```

```

// 파일에서 id가 일치하는 줄은 수정할 정보를 모으고, 나머지는 원래 정보를 모은다
StringBuilder fileContent = new StringBuilder();
boolean userFound = false;
try (BufferedReader fileReader = new BufferedReader(new FileReader(file))) {
    String fileLine;
    while ((fileLine = fileReader.readLine()) != null) {
        // "id":값 형태로 id를 검사
        if (fileLine.contains("\"id\":\"" + id + "\"")) {
            fileContent.append(body).append(str:"\n");
            userFound = true; // 해당하는 줄을 수정된 정보로 대체
        } else {
            fileContent.append(fileLine).append(str:"\n");
        }
    }
} catch (IOException e) {
    sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일 읽기 중 오류 발생");
    return;
}

// 해당 id의 유저가 없을 때 404 Not Found
if (!userFound) {
    sendHttpResponse(writer, statusCode:404, statusText:"Not Found", body:"해당 id의 사용자를 찾을 수 없습니다.");
    return;
}

// 전체 파일 내용을 업데이트 (덮어쓰기)
try {
    Files.write(Paths.get(first:"users.txt"), fileContent.toString().getBytes(),
    StandardOpenOption.TRUNCATE_EXISTING, StandardOpenOption.CREATE);
    sendHttpResponse(writer, statusCode:200, statusText:"OK", body:"사용자 정보가 수정되었습니다.");
    return;
} catch (IOException e) {
    sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일 저장 중 오류 발생");
    return;
}
}

```

- 기존의 사용자를 수정하는 함수
- 요청 본문이 비어있으면 400 응답을 보냄
- 읽은 요청 본문의 길이와 Content-Length가 다르면 500 응답을 보냄
- body에서 extractValueFromJson함수를 통해 id에 해당하는 값을 읽어옴
- 요청 본문에 id가 없으면 400 응답을 보냄
- 파일이 존재하지 않으면 404 응답을 보냄
- 중간에 에러가 발생하면 500 응답을 보냄
- 파일에서 id에 해당하는 줄을 요청 받은 body로 바꿈
- 해당하는 유저가 없으면 404 응답을 보냄
- users.txt 파일을 새롭게 변경된 내용으로 덮어씀

```

// DELETE /users/{id} 요청 처리 함수
private static void handleDeleteUser(PrintWriter writer, String id) {
    File file = new File(pathname:"users.txt");
    // users.txt 파일이 존재하지 않을 때 404 Not Found
    if (!file.exists()) {
        sendHttpResponse(writer, statusCode:404, statusText:"Not Found", body:"users.txt 파일을 찾을 수 없습니다.");
        return;
    }

    StringBuilder fileContent = new StringBuilder();
    boolean userFound = false;

    // 파일에서 id가 일치하는 줄은 생략하고, 나머지를 StringBuilder에 모은다
    try (BufferedReader fileReader = new BufferedReader(new FileReader(file))) {
        String fileLine;
        while ((fileLine = fileReader.readLine()) != null) {
            if (fileLine.contains("\"id\":\"" + id + "\"")) {
                userFound = true; // 해당하는 줄은 생략
            } else {
                fileContent.append(fileLine).append(str:"\n");
            }
        }
    } catch (IOException e) {
        sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일 읽기 중 오류 발생");
        return;
    }

    // 해당 id의 사용자가 없으면 404 Not Found
    if (!userFound) {
        sendHttpResponse(writer, statusCode:404, statusText:"Not Found", body:"해당 id의 사용자를 찾을 수 없습니다.");
        return;
    }

    // 전체 파일 내용을 업데이트 (덮어쓰기)
    try {
        Files.write(Paths.get(first:"users.txt"), fileContent.toString().getBytes(),
        StandardOpenOption.TRUNCATE_EXISTING, StandardOpenOption.CREATE);
        sendHttpResponse(writer, statusCode:200, statusText:"OK", body:"사용자 정보가 삭제되었습니다.");
        return;
    } catch (IOException e) {
        sendHttpResponse(writer, statusCode:500, statusText:"Internal Server Error", body:"파일 저장 중 오류 발생");
        return;
    }
}

```

- id에 해당하는 유저를 삭제하는 함수
- 파일이 존재하지 않으면 404 응답을 보냄
- 파일에서 id에 해당하는 줄은 생략하고 StringBuilder에 모음
- 중간에 에러가 발생하면 500 응답을 보냄
- 해당하는 유저가 없으면 404 응답을 보냄
- users.txt 파일을 해당 id의 유저가 생략된 내용으로 덮어씀

```

// JSON 문자열에서 특정 key의 값 추출 함수
private static String extractValueFromJson(String json, String key) {
    String search = "\"" + key + "\":\""; 
    int start = json.indexOf(search);
    if (start == -1) return null;
    start += search.length();
    int end = json.indexOf("\"", start);
    if (end == -1) return null;
    return json.substring(start, end);
}

```

- json을 파싱하는 함수
- 요청받은 key 값에 해당하는 값을 json에서 찾는 함수

5. 실행 결과

전체적인 흐름:

기존에 유저가 들어있는 users.txt 파일에 새로운 유저를 추가

- > 전체 유저 조회
- > 조건에 맞는 유저 조회
- > 전체 유저 조회 HEAD 요청
- > 조건에 맞는 유저 조회 HEAD 요청
- > id에 해당하는 유저 수정
- > id에 해당하는 유저 삭제
- > 변경사항을 확인하기 위한 전체 유저 조회

```

{"id":"98e59ac7-c8ec-4f9f-9556-7a9de227e289","name":"hwang","address":"Seoul","tel":"010-1234-1234","age":"99"}
{"id":"c2ec2f90-3ca6-4ea4-85dd-b7e7a73c82fc","name":"lee","address":"Busan","tel":"010-1111-1111","age":"21"}
{"id":"314b5ba8-6e32-4b5b-adef-a7a715e0ae0f","name":"park","address":"Jeju","tel":"010-2222-2222","age":"22"}
{"id":"bc375aee-f72d-417e-b51d-a95b3b1e2140","name":"kim","address":"Jeju","tel":"010-3333-3333","age":"22"}

```

- 실행 전 users.txt

7070번 포트에서 서버 실행되는 중
 클라이언트 요청 : POST /users HTTP/1.1
 클라이언트 요청 : POST /users HTTP/1.1
 클라이언트 요청 : POST /users HTTP/1.1

```
[POST /users HTTP/1.1] 응답 :  
HTTP/1.1 201 Created  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 16
```

사용자 정보가 저장되었습니다.

```
[POST /users HTTP/1.1] 응답 :
```

```
HTTP/1.1 201 Created  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 16
```

사용자 정보가 저장되었습니다.

```
[POST /users HTTP/1.1] 응답 :
```

```
HTTP/1.1 201 Created  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 16
```

사용자 정보가 저장되었습니다.

- 세 명의 유저를 성공적으로 추가

```
클라이언트 요청 : GET /users HTTP/1.1
```

```
[GET /users HTTP/1.1] 응답 :
```

```
HTTP/1.1 200 OK  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 772
```

```
{"id":"98e59ac7-c8ec-4f9f-9556-7a9de227e289","name":"hwang","address":"Seoul","tel":"010-1234-1234","age":"99"}  
{"id":"c2ec2f90-3ca6-4ea4-85dd-b7e7a73c82fc","name":"lee","address":"Busan","tel":"010-1111-1111","age":"21"}  
{"id":"314b5ba8-6e32-4b5b-adef-a7a715e0ae0f","name":"park","address":"Jeju","tel":"010-2222-2222","age":"22"}  
{"id":"bc375aee-f72d-417e-b51d-a95b3b1e2140","name":"kim","address":"Jeju","tel":"010-3333-3333","age":"22"}  
{"id":"cdab4c20-6d1d-42f8-bc21-8bf4aecc43fc","name":"bae","address":"Seoul","tel":"010-0101-0101","age":"10"}  
{"id":"6f9fd19e-97d9-4929-8880-5a692c731063","name":"lee","address":"Busan","tel":"010-1212-1212","age":"21"}  
{"id":"7d857ed2-77ac-4935-b52d-8ca46d77c3b7","name":"kang","address":"Seoul","tel":"010-2323-2323","age":"22"}
```

- 전체 유저를 성공적으로 조회

```
클라이언트 요청 : GET /users?name=kim HTTP/1.1
```

```
[GET /users?name=kim HTTP/1.1] 응답 :
```

```
HTTP/1.1 200 OK  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 109
```

```
{"id":"bc375aee-f72d-417e-b51d-a95b3b1e2140","name":"kim","address":"Jeju","tel":"010-3333-3333","age":"22"}
```

- Name이 kim인 유저를 성공적으로 조회

클라이언트 요청 : GET /users?address=Jeju HTTP/1.1

[GET /users?address=Jeju HTTP/1.1] 응답 :

```
HTTP/1.1 200 OK
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 219

{"id":"314b5ba8-6e32-4b5b-adef-a7a715e0ae0f","name":"park","address":"Jeju","tel":"010-2222-2222","age":"22"}
{"id":"bc375aee-f72d-417e-b51d-a95b3b1e2140","name":"kim","address":"Jeju","tel":"010-3333-3333","age":"22"}
```

- Address가 Jeju인 유저를 성공적으로 조회

클라이언트 요청 : GET /users?tel=010-2222-2222 HTTP/1.1

[GET /users?tel=010-2222-2222 HTTP/1.1] 응답 :

```
HTTP/1.1 200 OK
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 110

{"id":"314b5ba8-6e32-4b5b-adef-a7a715e0ae0f","name":"park","address":"Jeju","tel":"010-2222-2222","age":"22"}
```

- Tel이 010-2222-2222인 유저를 성공적으로 조회

클라이언트 요청 : GET /users?age=22 HTTP/1.1

[GET /users?age=22 HTTP/1.1] 응답 :

```
HTTP/1.1 200 OK
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 330

{"id":"314b5ba8-6e32-4b5b-adef-a7a715e0ae0f","name":"park","address":"Jeju","tel":"010-2222-2222","age":"22"}
{"id":"bc375aee-f72d-417e-b51d-a95b3b1e2140","name":"kim","address":"Jeju","tel":"010-3333-3333","age":"22"}
{"id":"7d857ed2-77ac-4935-b52d-8ca46d77c3b7","name":"kang","address":"Seoul","tel":"010-2323-2323","age":"22"}
```

- Age가 22인 유저들을 성공적으로 조회

클라이언트 요청 : HEAD /users HTTP/1.1

[HEAD /users HTTP/1.1] 응답 :

```
HTTP/1.1 200 OK
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 772
```

- 전체 유저를 조회의 HEAD 요청 (헤더만 응답 받음)

클라이언트 요청 : HEAD /users?name=kim HTTP/1.1

[HEAD /users?name=kim HTTP/1.1] 응답 :

```
HTTP/1.1 200 OK
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 109
```

- Name이 kim인 유저 조회의 HEAD 요청 (헤더만 응답 받음)

클라이언트 요청 : PUT /users HTTP/1.1

```
[PUT /users HTTP/1.1] 응답 :  
HTTP/1.1 200 OK  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 16
```

사용자 정보가 수정되었습니다.

- 해당 id를 가진 유저의 정보를 성공적으로 수정

클라이언트 요청 : PUT /users HTTP/1.1

```
[PUT /users HTTP/1.1] 응답 :
```

```
HTTP/1.1 404 Not Found  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 22
```

해당 id의 사용자를 찾을 수 없습니다.

- 해당 id를 가진 유저가 존재하지 않는 PUT 요청인 경우 404 Not Found를 응답 받음

클라이언트 요청 : DELETE /users/314b5ba8-6e32-4b5b-edef-a7a715e0ae0f HTTP/1.1

```
[DELETE /users/314b5ba8-6e32-4b5b-edef-a7a715e0ae0f HTTP/1.1] 응답 :
```

```
HTTP/1.1 200 OK  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 16
```

사용자 정보가 삭제되었습니다.

- 해당 id를 가진 유저를 성공적으로 삭제

클라이언트 요청 : GET /users HTTP/1.1

```
[GET /users HTTP/1.1] 응답 :
```

```
HTTP/1.1 200 OK  
Date: Wed, 30 Apr 2025 03:50:51 GMT  
Content-Type: text/plain  
Content-Length: 661
```

```
{"id":"98e59ac7-c8ec-4f9f-9556-7a9de227e289","name":"yang","address":"Seoul","tel":"010-9999-9999","age":23}  
{"id":"c2ec2f90-3ca6-4ea4-85dd-b7e7a73c82fc","name":"lee","address":"Busan","tel":"010-1111-1111","age":21}  
{"id":"bc375aee-f72d-417e-b51d-a95b3b1e2140","name":"kim","address":"Jeju","tel":"010-3333-3333","age":22}  
{"id":"cdab4c20-6d1d-42f8-bc21-8bf4aec43fc","name":"bae","address":"Seoul","tel":"010-0101-0101","age":10}  
{"id":"6f9fd19e-97d9-4929-8880-5a692c731063","name":"lee","address":"Busan","tel":"010-1212-1212","age":21}  
{"id":"7d857ed2-77ac-4935-b52d-8ca46d77c3b7","name":"kang","address":"Seoul","tel":"010-2323-2323","age":22}
```

- 유저 정보 수정이 성공적인지 확인하기 위한 전체 유저 조회

클라이언트 요청 : GET /users? HTTP/1.1

[GET /users? HTTP/1.1] 응답 :

HTTP/1.1 400 Bad Request
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 14

쿼리 파라미터가 없습니다.

- 특정한 조건의 유저를 조회하는 요청에서 쿼리 파라미터가 비어있는 경우 400 Bad Request를 응답 받음

클라이언트 요청 : POST /invalid HTTP/1.1

[POST /invalid HTTP/1.1] 응답 :

HTTP/1.1 400 Bad Request
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 14

지원하지 않는 요청입니다.

- 존재하지 않는 경로의 POST 요청을 보낼 경우 400 Bad Request를 응답 받음

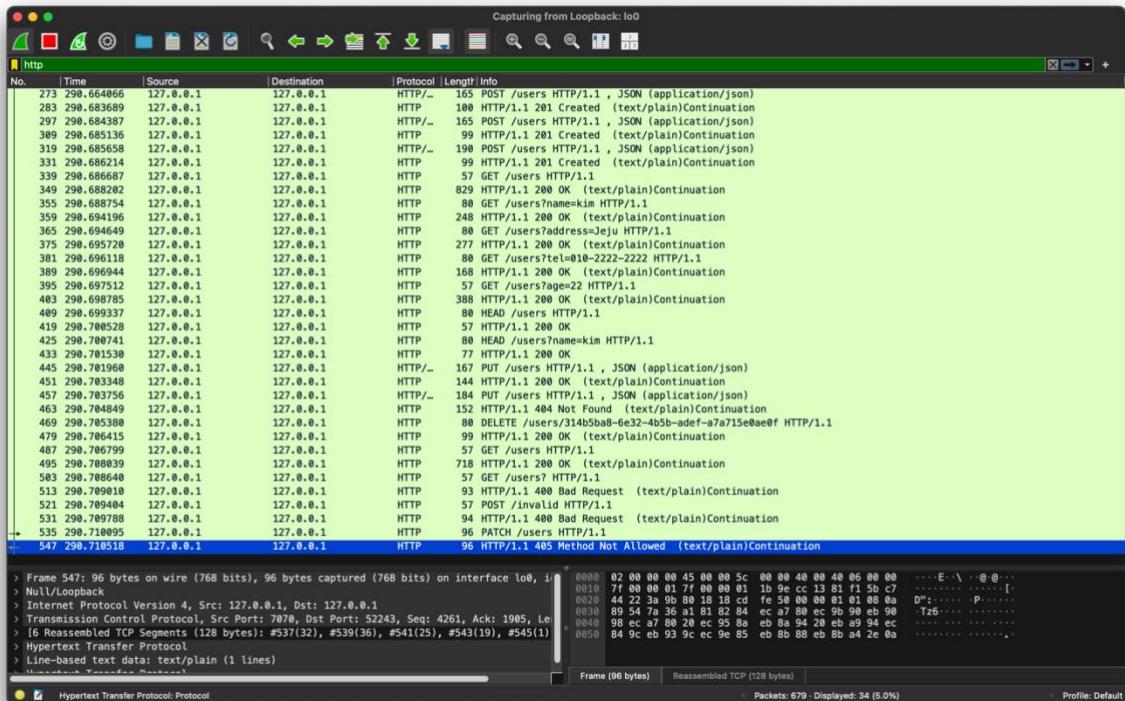
클라이언트 요청 : PATCH /users HTTP/1.1

[PATCH /users HTTP/1.1] 응답 :

HTTP/1.1 405 Method Not Allowed
Date: Wed, 30 Apr 2025 03:50:51 GMT
Content-Type: text/plain
Content-Length: 15

지원되지 않는 메서드입니다.

- 지원하지 않는 메서드인 PATCH 요청을 보냈을 때 405 Method Not Allowed를 응답 받음



- 와이어샤크 화면

```
["id":"98e59ac7-c8ec-4f9f-9556-7a9de227e289", "name":"yang", "address":"Seoul", "tel":"010-9999-9999", "age":23}
 {"id":"c2ec2f90-3ca6-4ea4-85dd-b7e7a73c82fc", "name":"lee", "address":"Busan", "tel":"010-1111-1111", "age":21}
 {"id":"bc375aee-f72d-417e-b51d-a95b3b1e2140", "name":"kim", "address":"Jeju", "tel":"010-3333-3333", "age":22}
 {"id":"cdab4c20-6d1d-42f8-bc21-8bf4aec43fc", "name":"bae", "address":"Seoul", "tel":"010-0101-0101", "age":10}
 {"id":"6f9fd19e-97d9-4929-8880-5a692c731063", "name":"lee", "address":"Busan", "tel":"010-1212-1212", "age":21}
 {"id":"7d857ed2-77ac-4935-b52d-8ca46d77c3b7", "name":"kang", "address":"Seoul", "tel":"010-2323-2323", "age":22}
```

- 실행 후 users.txt

- 유저 추가, 수정, 삭제가 성공적으로 된 것을 확인할 수 있음