Assignment2 design document

수업코드 12175

2018079116 정진성

Contents

Summary

Assignment2는 가구 카탈로그를 제공하는 작은 웹 서버를 만드는 것입니다. HTTP 프로토콜을 사용하여 클라이언트와 서버가 상호작용하고 쿠키를 사용하여 Statefull을 구현합니다. 이때, 이전 Assignment1과 달리 자바의 Http 및 Cookie 클래스의 이용없이 TCP Socket을 이용하여 네트워크 통신을 구현합니다.

Specification

- Java 17 version

- java.net.Socket package
- java.net.ServerSocket package

Design and Implementation

전체적인 작동 구조에 대한 코드입니다.

```
public class Server {
    public static void main(String[] args) throws IOException {
        // 8080 포트 서버 생성
        ServerSocket serverSocket = new ServerSocket(8080);
        System.out.println("Listening on port: " +
serverSocket.getLocalPort());

        while (true){
            // 클라이언트 요청 대기
            Socket clientSocket = serverSocket.accept();

            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            // 요청에 따른 클라이언트 소켓 생성
            Client client  = new Client(clientSocket, bufferedReader);

            // 클라이언트 요청 파싱해서 요청에 따른 responese 메시지 만들기
            ResponseDto responseDto = requestHandler(client.requestParse());

            // 응답 전송
            sendResponse(client, responseDto);

            bufferedReader.close();

        }
    }
...
```

Java.net 클래스의 ServerSocket 을 이용하여 8080 포트의 서버를 생성하고 while 문의 serverSocket.accept() 메서드로 클라이언트의 요청을 기다립니다.

클라이언트의 요청이 발생하면 직접 생성한 클래스인 Client 클래스에 clientsocket 객체의 주소를 가지는 client 객체를 생성합니다.

Client 클래스의 requestParse() 메서드를 통해 요청 메시지를 파싱하고, requestHandler 를 통해 요청에 맞는 응답을 구성하여 sendResponse 메서드를 통해 보냅니다.

Client 클래스는 request 메시지를 파싱하여 RequestDto 로 반환하도록 requestParse 메서드를 지원합니다.

```
public class Client {
    private final Socket socket;
    private final BufferedReader bufferedReader;

    public Client (Socket socket, BufferedReader bufferedReader) {
        this.socket = socket;
        this.bufferedReader = bufferedReader;
    }

    public Socket getSocket() {
        return socket;
    }

    // method, url, version 반환하고 다른 것들 Map 으로 파싱해서 반환
    public RequestDto requestParse() throws IOException {

...

        return new RequestDto(method, URL, version, body, headerMap);
    }

}
```

```java
// method, url, version 반환하고 다른 것들 Map 으로 파싱해서 반환
    public RequestDto requestParse() throws IOException {
        String method = "";
        String URL = "";
        String version = "";
        Map<String, String> headerMap = new HashMap<>();
        String body = "";
        StringBuilder bodyBuilder = new StringBuilder();


        // socket inputstream 연결
        String line;
        boolean isBody = false;

        while ((line = bufferedReader.readLine()) != null
&& !line.isEmpty()) {
            if (method.isEmpty()) {
                // 첫 번째 줄
                String[] requestLine = line.split(" ");
                method = requestLine[0];
                URL = requestLine[1];
                version = requestLine[2];
            } else {
                // 헤더 Map 으로 저장
                String[] header = line.split(": ", 2);
                if (header.length == 2) {
                    headerMap.put(header[0], header[1]);
                }
            }
            // 빈 줄을 만나면 body
            if (line.trim().isEmpty()) {
                isBody = true;
            } else if (isBody) {
                bodyBuilder.append(line).append("\n");
            }
        }

        body = bodyBuilder.toString().trim();

        return new RequestDto(method, URL, version, body, headerMap);
    }
```

client socket의 inputstream을 연결하여 요청 메시지를 line으로 확인하며 method, 요청 URL, HTTP 프로토콜의 version을 저장하였습니다. 또한 header를 key: value의 Map으로 저장하여 필요한 헤더의 내용을 빠르게 찾을 수 있도록 하였습니다. header이후 빈줄 다음 body가 나오게 되는데 body는 어떤 형식인지 아직 알지 못하므로 이후 처리하기 위해 String으로 저장하였습니다. 이렇게 parsing한 내용을 바탕으로 RequestDto를 구성하여 요청을 처리하기 편하도록 하였습니다.

```java
public class RequestDto {
    private String method;
    private String URL;
    private String version;
    private String body;
    private Map<String, String> header;

    public RequestDto(String method, String URL, String version, String
body, Map<String, String> header) {
        this.method = method;
        this.URL = URL;
        this.version = version;
        this.body = body;
        this.header = header;
    }

    public String getMethod() {
        return method;
    }

    public String getURL() {
        return URL;
    }

    public Map<String, String> getHeader() {
        return header;
    }

    public String getVersion() {
        return version;
    }
}
```

requestDto는 method, URL, version, body, header를 가지고 있습니다.

```
private static ResponseDto requestHandler(RequestDto requestDto) throws
IOException {
        String basePath = "./src/resources/";
        String path = requestDto.getURL();
        Map<String, String> header = new HashMap<>();
        String hello;
        boolean isCookie = false;
        byte[] body = null;
        // 요청 확인
        if (requestDto.getMethod().equals("GET")) {

...

        }
        // 잘못된 요청
        header.put("Content-Type", "text/html");
        return new ResponseDto(404, "Not Found", header, "<h1>404 Not
Found</h1>".getBytes(), requestDto.getVersion());
    }
```

server의 requestHandler는 client가 처리하여 반환한 requestDto를 인자로 받아 요청에 따른 responseDto를 반환합니다. 우선 과제 내용은 GET method만이 활용되므로 GET 요청 이외의 method 및 잘못된 요청은 404 Not Found를 반환하도록 상태 코드와 상태 메시지를 담은 ResponseDto를 반환합니다. 또한 쿠키 처리를 위해 requestDto의 header를 확인하여 Cookie 로직을 수행합니다.

쿠키 처리는 다음과 같이 이루어집니다.

```
// 요청 확인
        if (requestDto.getMethod().equals("GET")) {

            // 쿠키 확인
            Map<String, String> requestHeader = requestDto.getHeader();
            Set<String> keys = requestHeader.keySet();
            if (keys.contains("Cookie") &&
requestHeader.get("Cookie").split("=")[0].equals("StudentNumber")) {
                hello = "Returning user, welcome " +
requestHeader.get("Cookie").split("=")[1];
                isCookie = true;
            } else {
                hello = "New user requested page, cookie will be set";
                isCookie = false;
            }
```

Header의 key 중 "Cookie"가 있는지 확인하고 있다면 쿠키의 내용까지 확인하여 isCookie = true, 없다면 false로 설정한 후 각각에 맞는 로그를 출력하기 위한 String을 구성합니다. 만약 isCookie가 false라면 response의 header에 "Set-Cookie"를 추가하여 Cookie를 설정합니다.

과제 명세의 Request는 index page, detail page, json, image 요청의 4가지로 이루어집니다.

(1) Index page

```
// index page 요청
            if (path.equals("/")) {
                path = basePath + "html/index.html";
                System.out.println("Index page requested" + "\n" + hello);
                if ((body = getFileContent(path)) != null) {
                    header.put("Content-Type", "text/html");
                    header.put("Content-Length",
String.valueOf(body.length));
                    header.put("Date", new Date().toString());
                    if (!isCookie) header.put("Set-
Cookie","StudentNumber=2018079116");

                    return new ResponseDto(200, "OK", header, body,
requestDto.getVersion());
                }
            }
```

요청 URL이 root 페이지라면 index.html이 저장되어 있는 path를 지정하고, 로그를 출력합니다. 이때 hello는 쿠키의 설정 여부에 따라 달라지는 쿠키 로그입니다. Body에 getFileContent 메서드를 활용해 html 파일을 담고 파일에 맞는 header를 추가합니다. 쿠키 처리 또한 isCookie가 false일 경우에 Set-Cookie 헤더를 추가하여 설정합니다. 로직이 수행되면 ResponseDto객체에 상태코드, 상태메시지, 헤더, body, httpversion을 담아 반환합니다.

```
// 파일 읽기
    public static byte[] getFileContent(String path) throws IOException{
        File file = new File(path);

        FileInputStream fileInputStream = new FileInputStream(file);
        byte[] fileBytes = new byte[(int) file.length()];
        fileInputStream.read(fileBytes);
        fileInputStream.close();

        return fileBytes;
    }
```

getFileContent는 path에 있는 파일을 읽어 Byte배열로 반환하는 함수입니다.

(2) Detail page

```
// detail page 요청
            else if(path.equals("/chair") || path.equals("/table") ||
path.equals("/closet")) {
                String[] detail = path.split("/");
                System.out.println( detail[detail.length-1] + " page
requested"+ "\n" + hello);
                path = basePath + "html/detail.html";
                if ((body = getFileContent(path)) != null) {
                    header.put("Content-Type", "text/html");
                    header.put("Content-Length",
String.valueOf(body.length));
                    header.put("Date", new Date().toString());
                    if (!isCookie) header.put("Set-
Cookie","StudentNumber=2018079116");

                    return new ResponseDto(200, "OK", header, body,
requestDto.getVersion());
                }
            }
```

Detail page는 유저가 사진을 클릭할 경우에 요청되는 URL로 매핑하였습니다. Index page와 동일한 로직으로 getFileContent를 이용해 file을 읽고 쿠키를 처리하여 ResponseDto를 반환합니다.

(3) Furniture.json

```
// furniture.json 요청
            else if (path.equals("/furniture.json")) {
                System.out.println("furniture.json requested");
                path  = basePath + "data/furniture.json";
                if ((body = getFileContent(path)) != null) {
                    header.put("Content-Type", "application/json");
                    header.put("Content-Length",
String.valueOf(body.length));
                    header.put("Date", new Date().toString());
                    return new ResponseDto(200, "OK", header, body,
requestDto.getVersion());
                }
            }
```

Furniture.json 요청은 detail page 요청과 함께 발생하는 요청입니다. Detail.html에서 js를 이용해 furniture.json을 동적으로 요청하기 때문입니다. Page 요청이 아니기 때문에 cookie 처리는 하지 않는 점 빼고는 이전 요청들과 동일한 로직으로 수행됩니다.

(4) Image

```
// image 요청
            else if (path.equals("/chair.png") || path.equals("/table.png")
|| path.equals("/closet.png")) {
                String[] detail = path.split("/");
                String image = detail[detail.length-1];
                System.out.println( image + " requested");
                path = basePath + "data/" + image;
                if ((body = getFileContent(path)) != null) {
                    header.put("Content-Type", "image/png");
                    header.put("Content-Length",
String.valueOf(body.length));
                    header.put("Date", new Date().toString());
                    return new ResponseDto(200, "OK", header, body,
requestDto.getVersion());
                }
            }
```

Image 요청 또한 furniture.json 요청과 마찬가지로 detail.html에서 js를 이용해 동적으로 요청하는 부분입니다. 로직은 동일합니다.

requestHandler 메서드를 통해 반환받은 responseDto 클래스입니다.

```
public class ResponseDto {
    private int statusCode;
    private String statusMessage;
    private Map<String, String> header;
    private byte[] body;
    private String version;

    public ResponseDto(int statusCode, String statusMessage, Map<String,
String> header, byte[] body, String version) {
        this.statusCode = statusCode;
        this.statusMessage = statusMessage;
        this.header = header;
        this.body = body;
        this.version = version;
    }

    public int getStatusCode() {
        return statusCode;
    }

    public String getStatusMessage() {
        return statusMessage;
    }

    public Map<String, String> getHeader() {
        return header;
    }

    public byte[] getBody() {
        return body;
    }

    public String getVersion() {
        return version;
    }

}
```

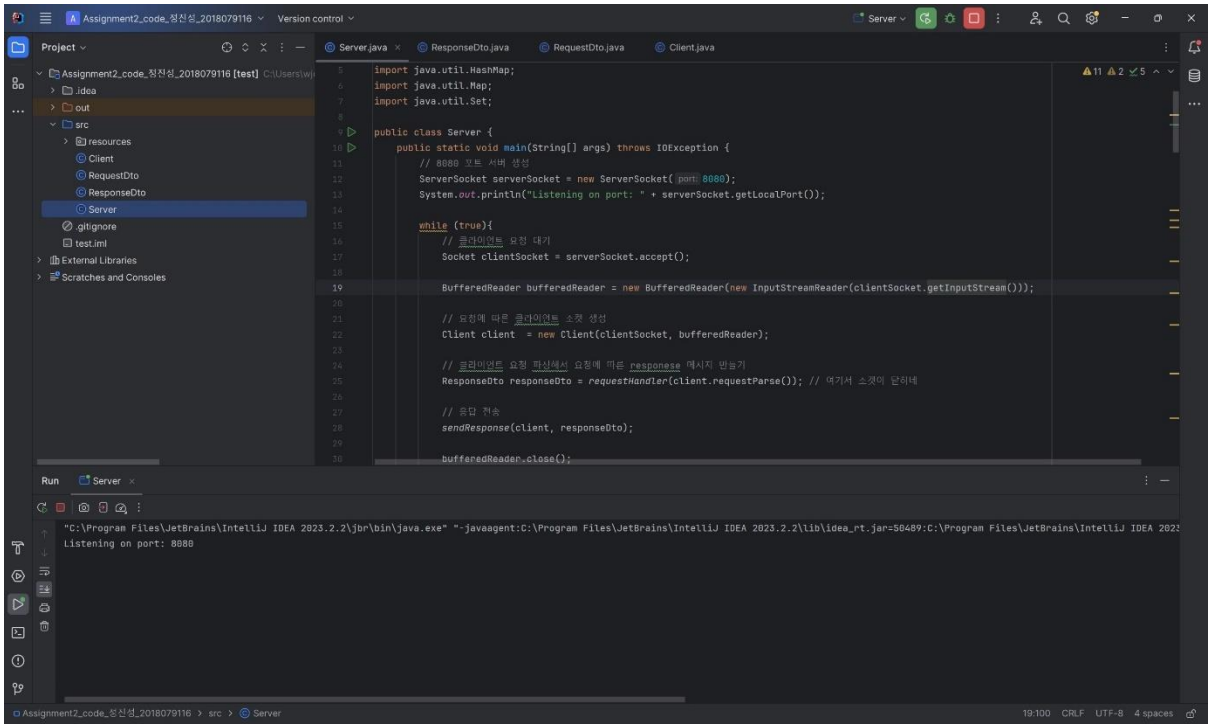
필드에는 StatusCode, Message, 헤더, body, http 프로토콜의 version을 저장하도록 했습니다.

만들어진 responseDto 는 sendResponse 메서드를 통해 client 소켓으로 보내집니다.

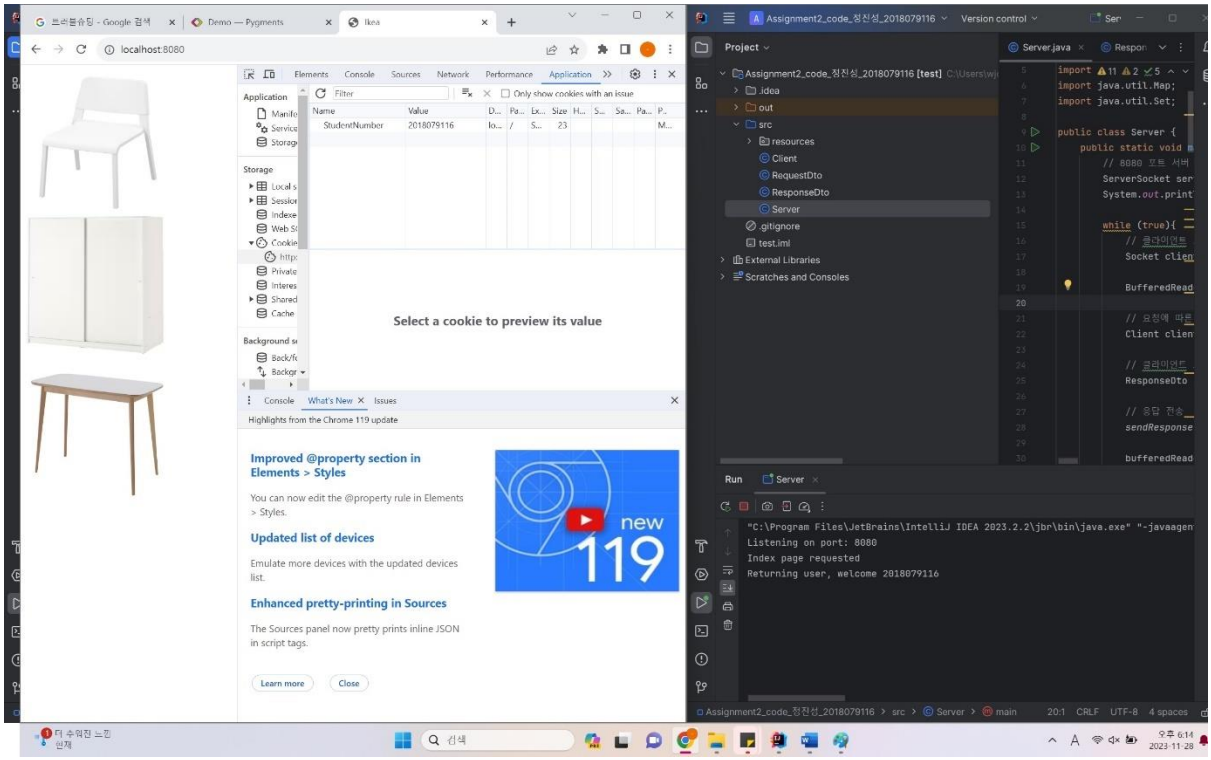
```
// 응답 전송
    public static void sendResponse(Client client, ResponseDto responseDto)
throws IOException{
        Socket socket = client.getSocket();
        OutputStream outputStream = socket.getOutputStream();

        // HTTP 응답 헤더 작성
        String response = responseDto.getVersion() + " " +
responseDto.getStatusCode() + " " + responseDto.getStatusMessage() + "\r\n";
        Map<String, String> header = responseDto.getHeader();
        Set<String> keys = header.keySet();

        for (String key : keys)
            response += key+": " + header.get(key) + "\r\n";
        response += "\r\n"; // 헤더와 바디를 나누는 빈 줄 추가

        outputStream.write(response.getBytes());
        outputStream.flush();
        outputStream.write(responseDto.getBody());
        outputStream.flush();

        outputStream.close();
        socket.close();
    }
```

Client 클래스의 socket을 받아와서 outputstream을 연결합니다. 이후 응답헤더를 http 프로토콜에 맞게 구성합니다. Version과 status code, status message를 포함하고, header 또한 프로토콜의 형식에 맞게 추가합니다. Response를 outputstream을 이용해 write하고 body 또한 write 한 후 소켓을 닫아주면 응답 전송이 완료됩니다.

Testing

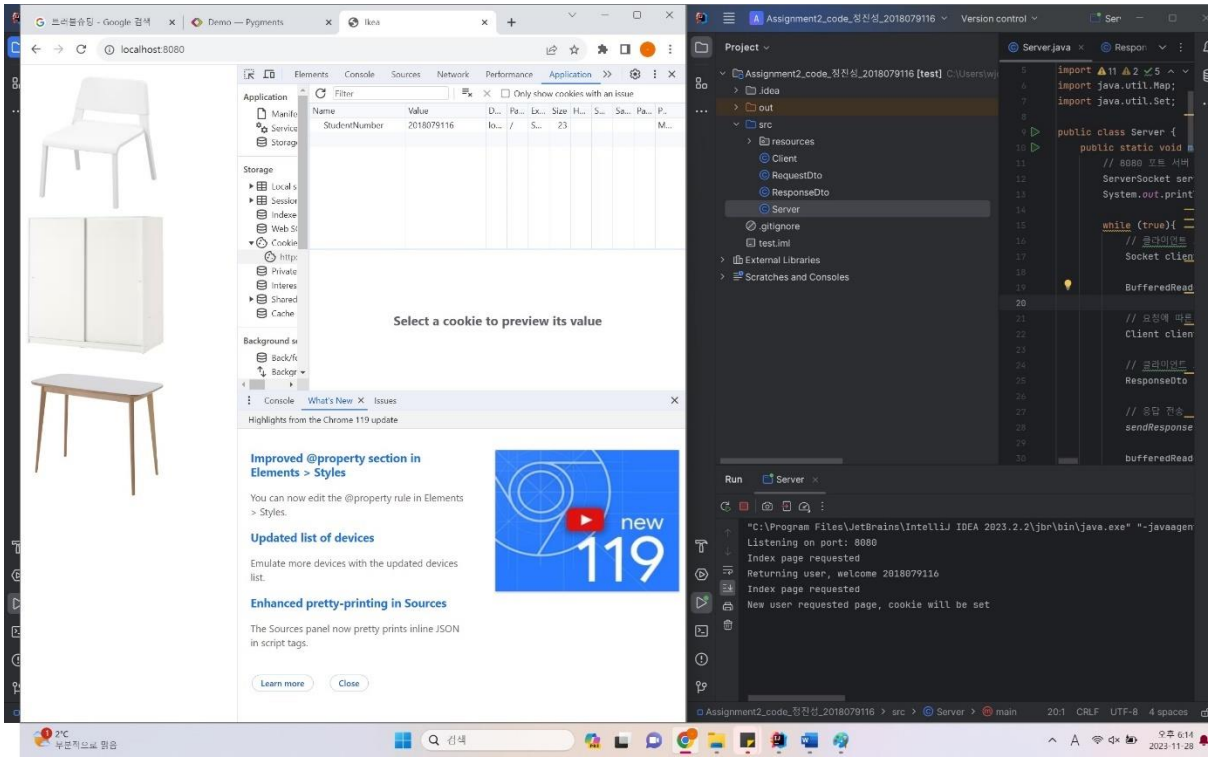


서버를 실행하기 위해 인텔리제이를 이용해 실행하였습니다. Server를 실행하면 8080 포트에 서버를 생성하고, 그에 따른 로그를 출력합니다.

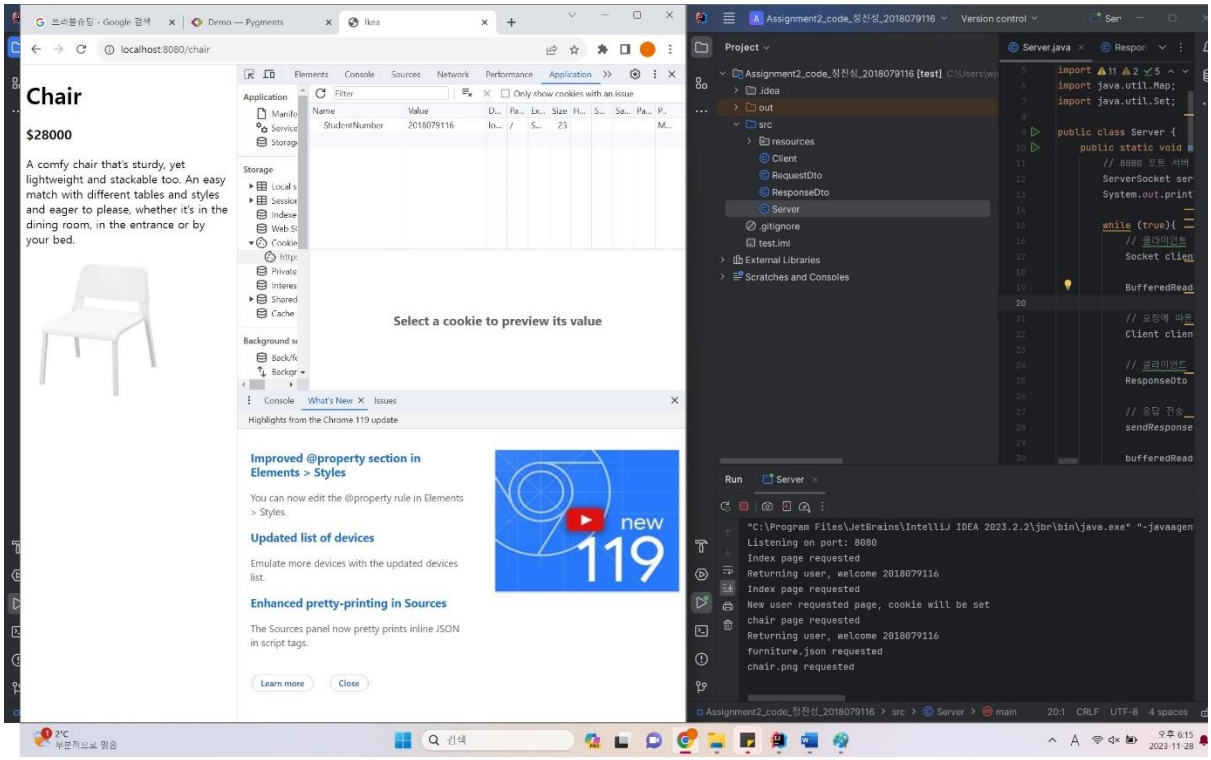


Index 페이지에 접근 시 그에 따른 page requested 로그를 출력합니다. 이 때는 쿠키가 설정되어

있어서 "Returnning user, welcome 학번"이 출력됩니다.

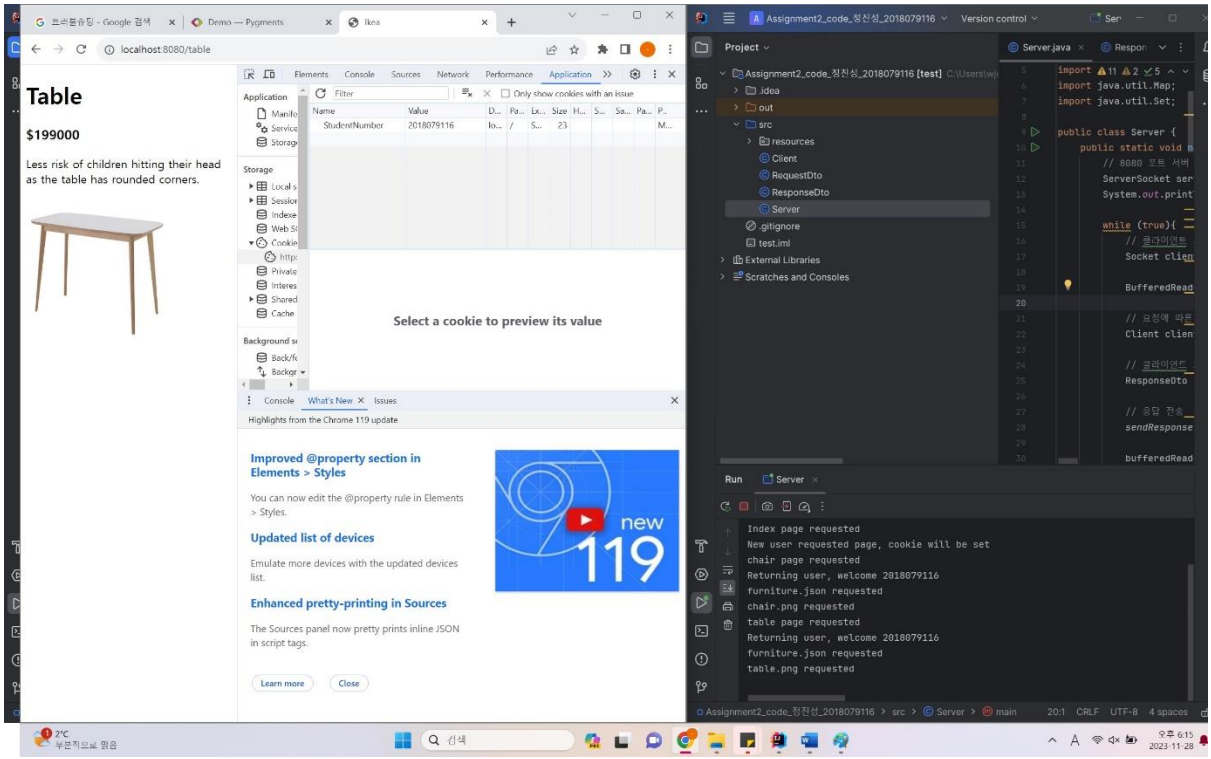


쿠키를 지운 후 새로고침해보면 "New user requested page, cookie wile be set" 문구가 출력되고 새로운 쿠키가 설정되어있음을 확인할 수 있습니다.

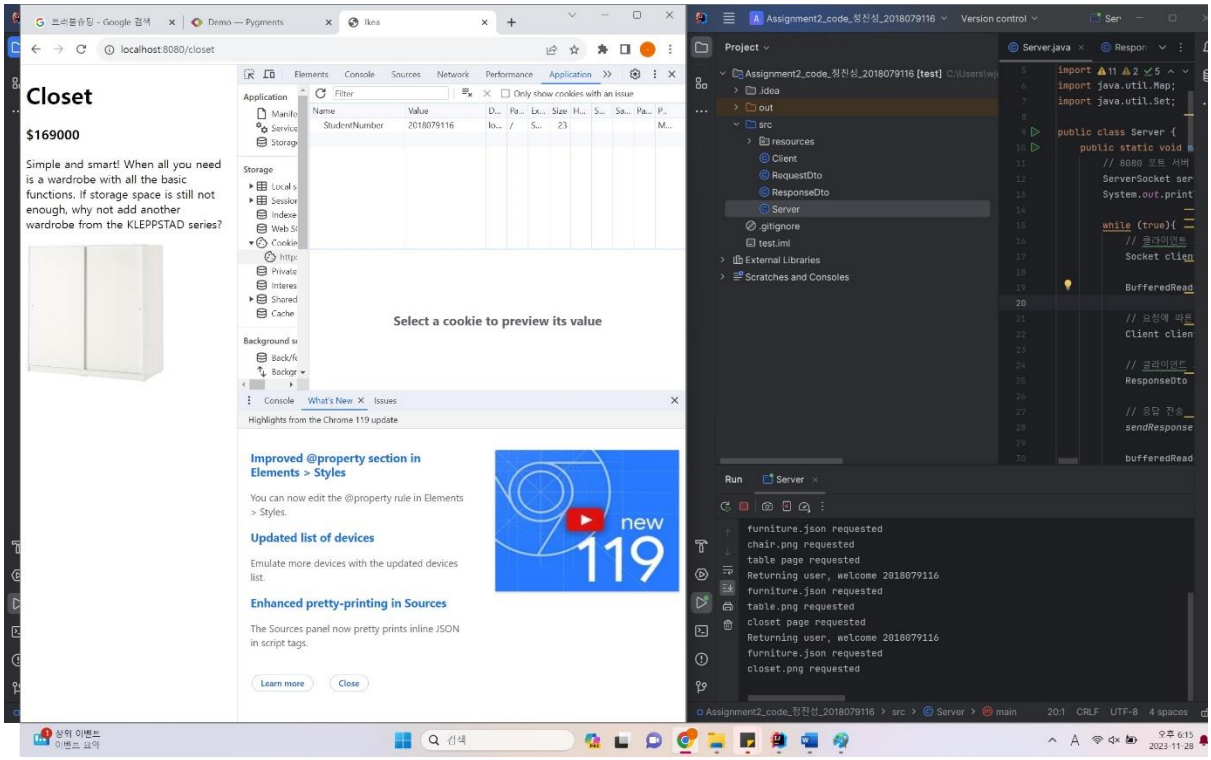


Detail page 중 chair page를 접근하면 그에 따른 로그가 출력되고, 쿠키처리 및 detail.html의 요

청으로 인한 furniture.json, chair.png가 요청됨을 확인할 수 있습니다.

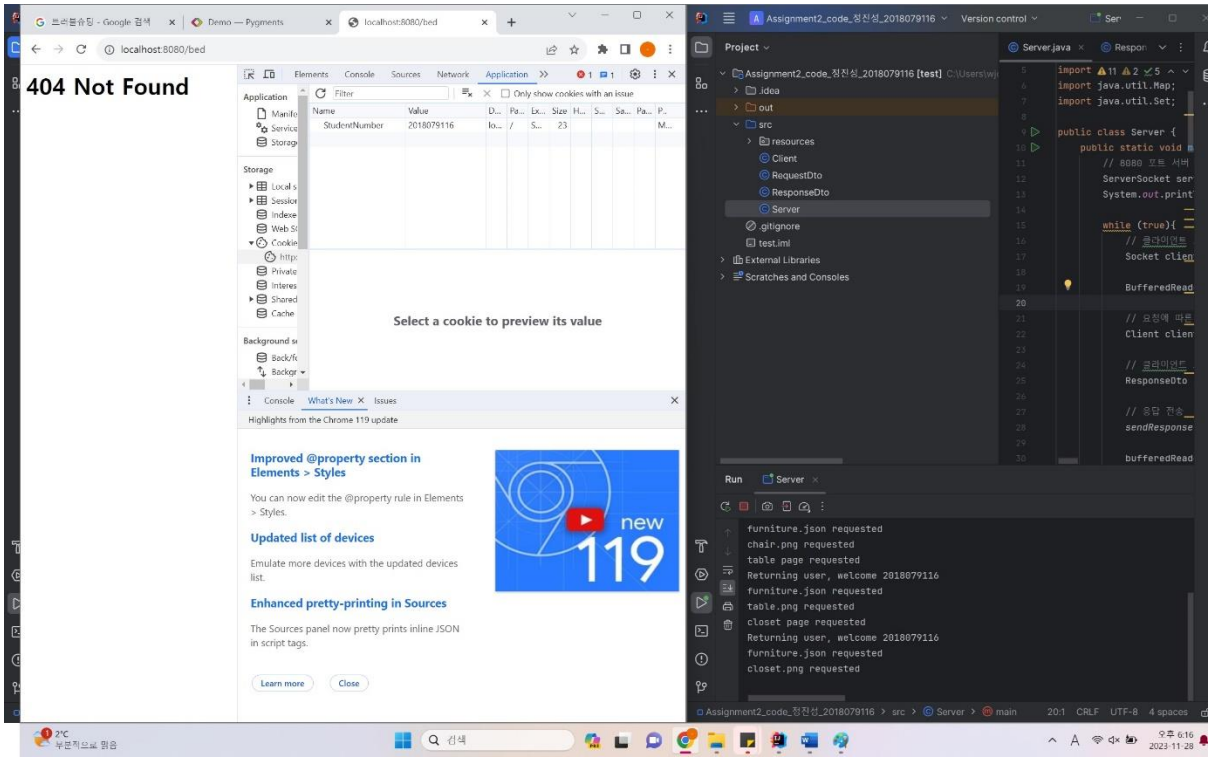


Detail page 중 table page를 접근하면 그에 따른 로그가 출력되고, 쿠키처리 및 detail.html의 요청으로 인한 furniture.json, table.png가 요청됨을 확인할 수 있습니다.

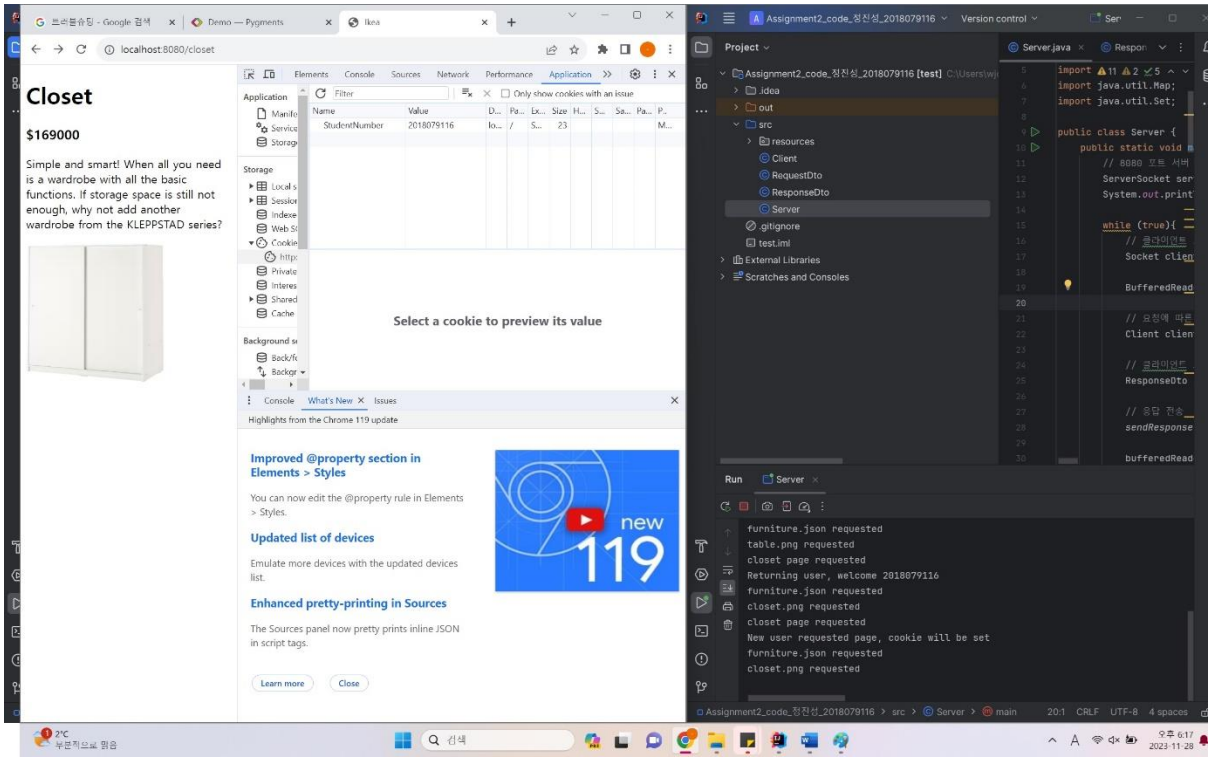


Detail page 중 closet page를 접근하면 그에 따른 로그가 출력되고, 쿠키처리 및 detail.html의 요

청으로 인한 furniture.json, closet.png가 요청됨을 확인할 수 있습니다.



만약 의도되지 않은 bed와 같은 요청이 들어오면 404 Not Found를 반환하는 것을 확인할 수 있습니다.



Detail page에서의 쿠키의 작동을 테스트하기 위해 closet 페이지에서 쿠키를 삭제하고 새로고침

해보면, 쿠키가 다시 설정되는 것을 확인할 수 있습니다.

Troubleshooting

1) SocketException

Client socket과 Server socket의 역할을 분리하기 위해 client 클래스를 만들어 request를 파싱하는 로직을 client 클래스에서 수행했습니다. 하지만 Client에서 inputstream을 열고 요청을 받아 파싱한 후 서버에서 다시 outputstream을 열 경우 socket이 이미 닫혀 outputstream에 접근할 수 없는 socketException이 발생했습니다.

이는 client 클래스에서 inputstream을 사용 후 닫았을 때 socket 또한 닫혀버렸기 때문입니다. 따라서 이를 해결하기 위해 Server 클래스에서 inputStream을 열고 이 inputStream 객체를 client에 전달해주는 방식으로 해결했고, Server에서 socket 처리 후 inputStream 및 outputStream, Socket을 모두 close 하는 방식으로 해결했습니다.

```
public class Server {
    public static void main(String[] args) throws IOException {
        // 8080 포트 서버 생성
        ServerSocket serverSocket = new ServerSocket(8080);
        System.out.println("Listening on port: " +
serverSocket.getLocalPort());

        while (true){
            // 클라이언트 요청 대기
            Socket clientSocket = serverSocket.accept();

            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            // 요청에 따른 클라이언트 소켓 생성
            Client client  = new Client(clientSocket, bufferedReader);

            // 클라이언트 요청 파싱해서 요청에 따른 responese 메시지 만들기
            ResponseDto responseDto = requestHandler(client.requestParse());

            // 응답 전송
            sendResponse(client, responseDto);

            bufferedReader.close();

        }
    }
```

```java
public static void sendResponse(Client client, ResponseDto responseDto)
throws IOException{
        Socket socket = client.getSocket();
        OutputStream outputStream = socket.getOutputStream();

        // HTTP 응답 헤더 작성
        String response = responseDto.getVersion() + " " +
responseDto.getStatusCode() + " " + responseDto.getStatusMessage() + "\r\n";
        Map<String, String> header = responseDto.getHeader();
        Set<String> keys = header.keySet();

        for (String key : keys)
            response += key+": " + header.get(key) + "\r\n";
        response += "\r\n"; // 헤더와 바디를 나누는 빈 줄 추가

        outputStream.write(response.getBytes());
        outputStream.flush();
        outputStream.write(responseDto.getBody());
        outputStream.flush();

        outputStream.close();
        socket.close();
    }
```