HTTP Test (with Python)

20203104 소프트웨어학부 유동현

☐ 목적

- TCP 기반의 소켓(Socket) 통신을 활용하여 HTTP의 동작원리를 이해하고, WireShark(패킷 분석 툴)를 활용하여 직접 구현한 HTTP 패킷을 캡쳐 가능하도록 구현하는 데 있다.

☐ 목표

1. TCP 기반의 Server, Client 간의 소켓(Socket) 통신 구현

2. Client에서 HTTP Protocol의 GET/PUT/HEAD/POST 형식에 맞는 Request를 Server에 보내고, Response를 받는다

3. Server에서 HTTP Protocol의 Client로부터의 Request에 대한 Response를 보낸다

4. WireShark를 활용하여 Resposne와 Request를 캡쳐한다.

5. 처음에는 Local 환경에서 진행, 이후 2대 이상의 PC를 활용하여 Client와 Server를 분리하여 실행한다.

☐ 실행 방법

Server

```
  $ git clone
https://github.com/DongHyeonYu/ComputerNetwork_test_HTTP.git
  $ cd /ComputerNetwork_test_HTTP.git
  $ python3 HTTP_Server.py
```

Client

```
  $ git clone
https://github.com/DongHyeonYu/ComputerNetwork_test_HTTP.git
  $ cd /ComputerNetwork_test_HTTP/HTTP/dist/

  HTTP_Client.py 코드 상단의 serverName(IP Address)/serverPort 조정
  (Local환경에서 실행 시 조정 불필요)

  [방법1]
  $ pip install PyQt5
  $ pip install pyinstaller

  $ cd ../
  $ pyinstaller --onefile HTTP_Test.py

  /dist 내부의 HTTP_Test.exe 실행

  [방법2]
  $ python3 HTTP_Client.py
```

□ 유의사항

- 반드시 Server Code(HTTP_Server.py)를 먼저 실행 시킨 후, Client 코드를 실행 시켜야 테스트가 가능하다.

□ 개발환경

Client

Windows10 / Python 3.12

- MacOS에서는 PyQt5 라이브러리 사용불가능으로 [방법2]로 진행

Server

MacOS / Python 3.12

□ Test Case 설계

# CASE 1

```
 Request :
   Method : GET
   Path : /
Response : 200 OK
```

# CASE 2

```
 Request :
   Method : GET
   Path : /NotFoundError
Response : 404 Not Found
```

# CASE 3_4

```
 Request :
   Method : POST
   Path : /
Response : 100 Continue & 200 OK
```

# CASE 5

```
 Request :
   Method : POST
   Path : /
Response : 100 Continue & 400 Bad Request
```

# CASE 6

```
 Request :
   Method : POST
   Path : /NotFoundError
Response : 404 Not Found
```

# CASE 7

```
 Request :
   Method : HEAD
   Path : /
Response : 200 OK
```

# CASE 8

```
 Request :
   Method : HEAD
   Path : /NotFoundError
Response : 404 Not Found
```

# CASE 9

```
Request :
  Method : PUT
  Path : /test.jpeg
Response : 100 Continue & 200 OK
```

# CASE 10

```
Request :
  Method : PUT
  Path : /400_Bad_Request.png
Response : 400 Bad Reqeust
```

□ 구현 목표

내용	적용 여부	비고
TCP기반의 소켓 프로그래밍 작성	O	HTTP_Client.py HTTP_Server.py
GET/POST/HEAD/PUT Request구현	O	HTTP_Client.py
HTTP Request에 대한 Response구현	O	HTTP_Server.py

□ 추가 구현 사항

내용	적용 여부	비고
PyQt 활용 간단한 GUI구현	O	HTTP_Test.py HTTP_Test.exe
간단한 웹페이지 활용 Response, Request 확인	X	

- 기존의 Console에서 Response, Request Message 확인하는데 있어 많은 Test Case로 인하여 가독성이 떨어지는 문제 발생
- 초기 웹페이지를 활용한 Response, Request 확인을 목표로 설정하였으나, Python코드를 HTML코드 내에서 실행(pyscript) 하는데 있어 성능상의 문제(속도, 무한로딩) 발생

□ TCP 프로그래밍

Server

```
from socket import *

serverPort = 8080
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((‘’, serverPort))
serverSocket.listen(1)

…

While True:
    connectionSocket, addr = serverSocket.accept()
    try:
        message = connectionSocket.recv(1024).decode()
        …
            response = response_line + headers + response_body
            connectionSocket.send(response.encode())
        …
…
connectionSocket.close()
```

- 기존 Python socket 라이브러리 활용
- Port 8080으로 Socket을 생성한 뒤, listen으로 Client의 Message 기다림
- while 무한 루프를 통해 사용자의 Message를 수신한 뒤, response를 생성하여 connectionSocket.send( )를 활용하여 Response 전송

Client

```
from socket import *

serverName = ‘192.168.0.22’
serverPort = 8080

def CASE1():
    clientSocekt = socket(AF_INET, SOCK_STREAM)
    clientSocekt.connect((serverName, serverPort))

    …
    response = clientSocket.recv(4096).decode()
    print(“From Server :”)
    print(response)
    clientSocekt.close()
    return request, response
```

- 기존 Python socket 라이브러리 활용
- serverName은 server code가 실행 되고 있는 PC의 내부 IP Address
- 각 CASE를 실행할 때마다 socket( ) 메소드로 소켓을 생성한 뒤, socket.connect( )로 server의 소켓과 연결하고 socket.send( )로 Request 메시지를 보냄
- 메시지를 보내고 socket.recv( ) 메소드로 서버의 Response를 기다리고, Response가 도착하면 화면에 출력

□ HTTP Protocol

Server

```
…
lines, request_line, method, path = request_parser(message)

    if method == “GET”:
        GET(path)

    elif method == “HEAD”:
        HEAD(path)

    elif method == “POST”:
        POST(connectionSocket, message, path)

    elif method == “PUT”:
        PUT(connectionSocekt, message, path)

    else:
        another_request()
```

- TCP 소켓 프로그래밍 코드에서 받은 Request 메시지를 request_parser( ) 함수로 파싱 한 뒤, 각 method에 맞는 함수로 분기

```
def request_parser(message):
    try:
        lines = message.split(“\r\n”)
        request_line = lines[0].split()
        method = request_line[0]
        path = request_line[1]
    …
    return lines, request_line, method, path
```

```
def GET(path):
    if path == “/”:
        response_line = “HTTP/1.1 200 OK\r\n”
        header = “Content-Type: text/html\r\n\r\n”
        body = “<html><script>alert(“Hello!”);</script></html>
        response = response_line + header + body
        connectionSocket.send(response.encode())
    else:
        response_line = “HTTP/1.1 404 Not Found\r\n”
        header = “Content-Type: text/html\r\n\r\n”
        body = “<html><body><h1>404 Not Found</h1></body></html>
        response = response_line + header + body
        connectionSocekt.send (response.encode())
    return

def POST(connectionSocket, message, path):
    if “Expect: 100-Continue” in message and path == “/”:
        response_line = “HTTP/1.1 100 Continue\r\n\r\n”
        connectionSocket.send(response_line.endcode())

        body=connectionSocket.recv(1024).decode().lstrip().rstrip()
.upper()
        if 0<len(body)<=30:
            final_response_line = “HTTP/1.1 200 OK\r\n”
            header = “Content-Type: text/html\r\n\r\n”
            response_body=
f“<html><body><h1>{body}</h1></body></html>”
            response = final_response_line + header + response_body
            connectionSocket.send(response.encode())
        else:
            final_response_line = “HTTP/1.1 400 Bad Request\r\n”
            header = “Content-Type: text/html\r\n\r\n”
            response_body = f“<html><body><h1>400 Bad
Request</h1></body></html>”
            response = final_response_line + header + response_body
            connectionSocket.send(response.encode())
    else:
        response_line = “HTTP/1.1 404 Not Found\r\n”
        header = “Content-Type: text/html\r\n\r\n”
        response_body = “<html><body><h1>404 Not 
Found</h1></body></html>”
        response = response_line + header + response_body
        connectionSocket.send(response.encode())

def HEAD(path):
    …(GET 동일)

def PUT(connectionSocket, message, path):
    …(POST 동일)
```

Client

```
def CASE1():
    …
      request_line = f“GET / HTTP/1.1\r\n”
      headers = (f“Host: {serverName}:{serverPort}\r\n”
                 f“User-Agent: Custom/1.0\r\n”
                 f“Connection: close\r\n\r\n”)
      request = request_line + headers
      clientSocekt.send(request.endcode())
    …

  def CASE2():
      …
      request_line = f“GET /NotFoundError HTTP/1.1\r\n”
      headers = (f“Host: {serverName}:{serverPort}\r\n”
                 f“User-Agent: Custom/1.0\r\n”
                 f“Connection: close\r\n\r\n”)
      …

  def CASE3_4():
      …
      message = “Hello World”
      request_line = f“POST / HTTP/1.1\r\n”
      headers = (f“Host: {serverName}:{serverPort}\r\n”
                 f“Expect: 100-Continue\r\n”
                 f“User-Agent: Custom/1.0\r\n”
                 f“Content-Length: {len(message)}\r\n”
                 f“Content-Type: text/plain\r\n\r\n”)
      …
      response = clientSocket.recv(1024).decode()
      if “100 Continue” in response:
          clientSocket.send((message + “\r\n”).endcode())
          final_response = clientSocekt.recv(1024).decode()
      …

  def CASE5():
      …
      message = “ ”
      request_line = f“POST / HTTP/1.1\r\n”
      headers = (f“Host: {serverName}:{serverPort}\r\n”
                 f“Expect: 100-Continue\r\n”
                 f“User-Agent: Custom/1.0\r\n”
                 f“Content-Length: {len(message)}\r\n”
                 f“Content-Type: text/plain\r\n\r\n”)
      …
      response = clientSocket.recv(1024).decode()

      if “100 Continue” in response:
          clientSocket.send((message + “\r\n”).endcode())
          final_response = clientSocekt.recv(1024).decode()
      …

  def CASE6():
      …
      message = “Hello World”
      request_line = f“POST /NotFoundError HTTP/1.1\r\n”
      headers = (f“Host: {serverName}:{serverPort}\r\n”
                 f“Expect: 100-Continue\r\n”
                 f“User-Agent: Custom/1.0\r\n”
                 f“Content-Length: {len(message)}\r\n”
                 f“Content-Type: text/plain\r\n\r\n”)
      …
      response = clientSocket.recv(1024).decode()

      if “100 Continue” in response:
          clientSocket.send((message + “\r\n”).endcode())
          final_response = clientSocekt.recv(1024).decode()
      …

  def CASE7():
      …
      request_line = f“HEAD / HTTP/1.1\r\n”
      headers = (f“Host: {serverName}:{serverPort}\r\n”
                 f“User-Agent: Custom/1.0\r\n”
                 f“Connection: close\r\n\r\n”)
      …

  def CASE8():
      …
      request_line = f“HEAD /NotFoundError HTTP/1.1\r\n”
      headers = (f“Host: {serverName}:{serverPort}\r\n”
                 f“User-Agent: Custom/1.0\r\n”
                 f“Connection: close\r\n\r\n”)
      …

  def CASE9():
      …
      filename = “test.jpeg”
      content_type = “image/jpeg”
      …
      request_line = (f“PUT /{filename} HTTP/1.1\r\n”
                      f“Host: {serverName}:{serverPort}\r\n”
                      f“Expect: 100-Continue\r\n”
                      f“Content-Type: {content_type}\r\n\r\n”
                     f“Content-Length: {len(file_content)\r\n}”
      …
      response = clientSocekt.recv(1024).decode()
      if “100 Continue in response:
          clientSocket.send(file_content)
      final_response = clientSocket.recv(1024).decode()
      …

  def CASE10():
      …
      filename = “400_Bad_Request.png”
      content_type = “image/png”
      …
      request_line = (f“PUT /{filename} HTTP/1.1\r\n”
                      f“Host: {serverName}:{serverPort}\r\n”
                      f“Expect: 100-Continue\r\n”
                      f“Content-Type: {content_type}\r\n\r\n”
                     f“Content-Length: {len(file_content)\r\n}”
      …
      response = clientSocekt.recv(1024).decode()
      if “100 Continue in response:
          clientSocket.send(file_content)
      final_response = clientSocket.recv(1024).decode()
      …
```

□ CASE 설명

# CASE1

- 기본적인 GET Method, Route path 로 요청
- Server Code에서 200 OK 응답

# CASE2

- GET Method, 의도적으로 존재하지 않는 path(/NotFoundError)로 Request를 보냄,
- Server Code에서 else문을 통해, Route path가 아닌 모든 path에 대하여 404 Not Found Error를 발생하도록 처리

# CASE3_4

- POST Method, Route path로 요청, Expect: 100-Continue 를 통해 서버에서 100응답을 받은 이후 100 Continue(CASE3)이면 body인 “Hello World” 메시지를 서버에 전송
- 서버에서 200 OK(CASE4) 응답과 함께 body의 모든 문자가 대문자로 변경된 응답을 받음

# CASE5

- POST Method, 100 응답을 받은 이후, body를 전송할 때 의도적으로 빈 문자열을 전송
- Server Code에서 빈 문자열을 전송받을 경우, 400 Bad Request가 발생하도록 처리

# CASE6

- POST Method, “hello world” 메시지를 의도적으로 존재하지 않는 path(/NotFoundError)로 전송
- Server Code에서 Routh path(/)가 아닌 모든 경로에 대해서 404 Not Found Error 가 발생하도록 처리

# CASE7

- HEAD Method, Route path로 요청
- Server Code에서 200 OK응답

# CASE8

- HAED Method, 의도적으로 존재하지 않는 경로 (/NotFoundError)로 요청
- Server Code에서 Route path(/)가 아닌 모든 경로에 대하여 404 Not Found Error가 발생하도록 처리

# CASE9

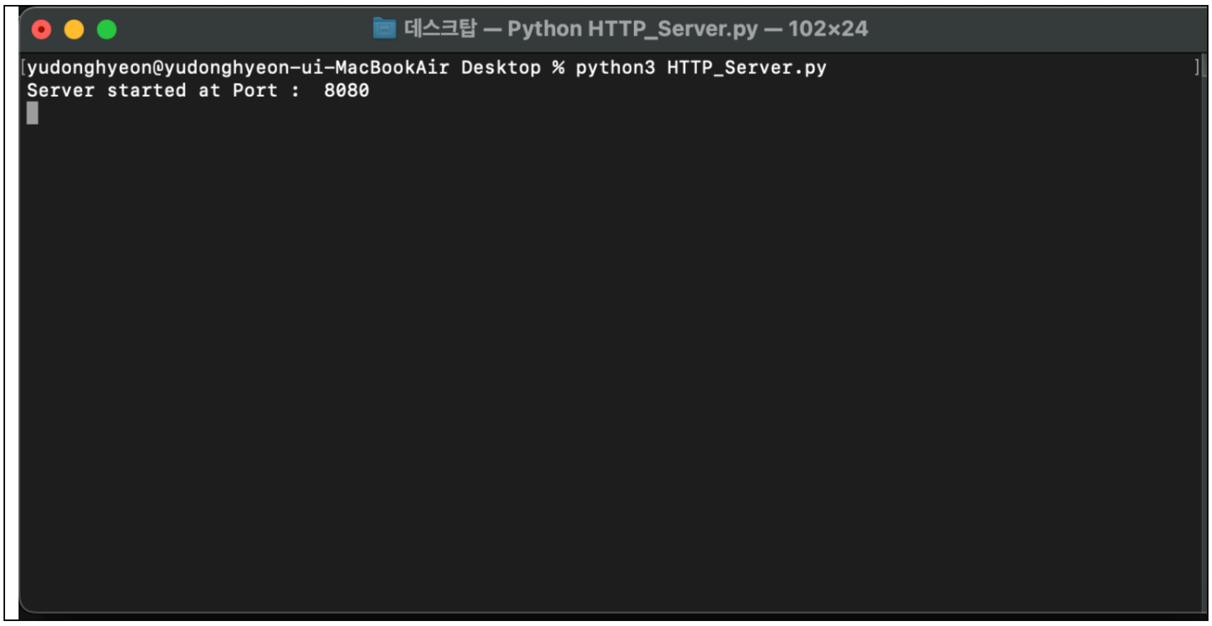
- PUT Method, test.jpeg(Image)전송
- Expect: 100-Continue를 통해 100 응답을 받은 이후 Image File을 open함수를 통해 bytes code로 변환한 후 서버에 전송
- Server Code에서 파일을 저장이후 200 OK 응답

# CASE10

- PUT Method, 400_Bad_Request.png(Image)전송
- Server Code에서 받을 수 있는 파일의 최대크기를 8192(byte)로 제한, 이를 초과하는 파일의 경우 400 Bad Request가 발생하도록 처리

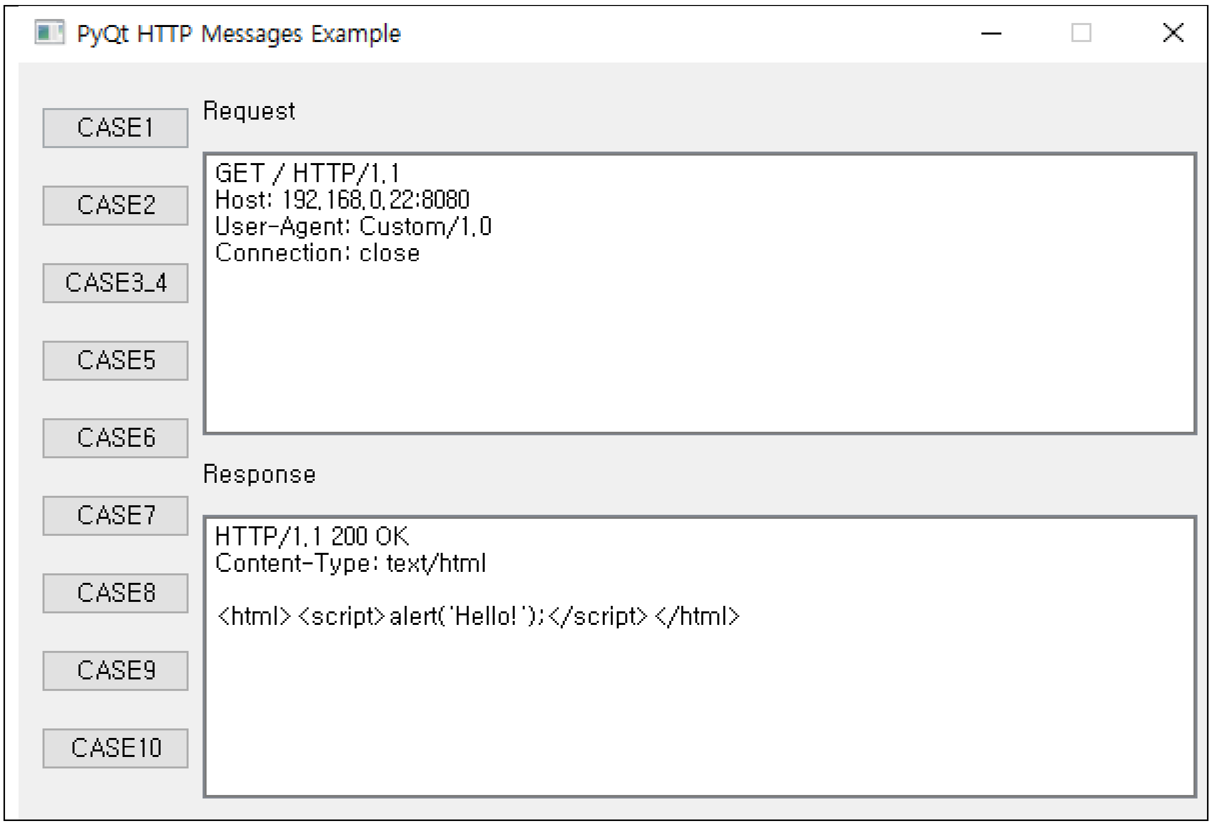
☐ 실행 환경

Server (MacOS, Terminal)



```
데스크탑 — Python HTTP_Server.py — 102×24
[yudonghyeon@yudonghyeon-ui-MacBookAir Desktop % python3 HTTP_Server.py
Server started at Port :  8080
```

Client (Windows10, PyQt GUI)

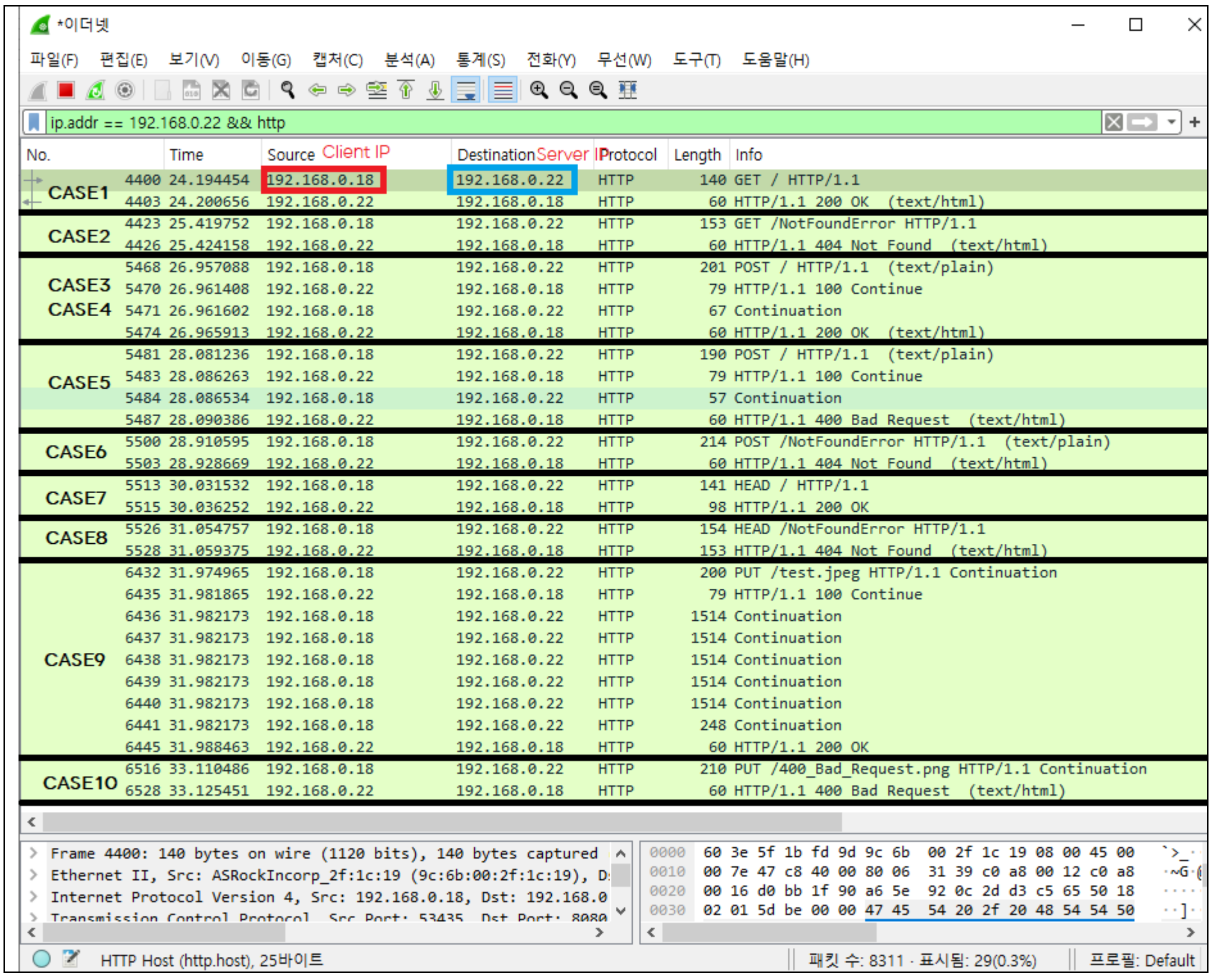


```
GET / HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<html> <script> alert('Hello!'); </script> </html>
```

□ WireShark 캡처 내용


*이더넷
파일(F) 편집(E) 보기(V) 이동(G) 캡처(C) 분석(A) 통계(S) 전화(Y) 무선(W) 도구(T) 도움말(H)
Client IP
Server IP

```
ip.addr == 192.168.0.22 && http
```

	No.	Time	Source	Destination	Protocol	Length	Info
CASE1	4400	24.194454	192.168.0.18	192.168.0.22	HTTP	140	GET / HTTP/1.1
	4403	24.200656	192.168.0.22	192.168.0.18	HTTP	60	HTTP/1.1 200 OK (text/html)
CASE2	4423	25.419752	192.168.0.18	192.168.0.22	HTTP	153	GET /NotFoundError HTTP/1.1
	4426	25.424158	192.168.0.22	192.168.0.18	HTTP	60	HTTP/1.1 404 Not Found (text/html)
CASE3 CASE4	5468	26.957088	192.168.0.18	192.168.0.22	HTTP	201	POST / HTTP/1.1 (text/plain)
	5470	26.961408	192.168.0.22	192.168.0.18	HTTP	79	HTTP/1.1 100 Continue
	5471	26.961602	192.168.0.18	192.168.0.22	HTTP	67	Continuation
	5474	26.965913	192.168.0.22	192.168.0.18	HTTP	60	HTTP/1.1 200 OK (text/html)
CASE5	5481	28.081236	192.168.0.18	192.168.0.22	HTTP	190	POST / HTTP/1.1 (text/plain)
	5483	28.086263	192.168.0.22	192.168.0.18	HTTP	79	HTTP/1.1 100 Continue
	5484	28.086534	192.168.0.18	192.168.0.22	HTTP	57	Continuation
	5487	28.090386	192.168.0.22	192.168.0.18	HTTP	60	HTTP/1.1 400 Bad Request (text/html)
CASE6	5500	28.910595	192.168.0.18	192.168.0.22	HTTP	214	POST /NotFoundError HTTP/1.1 (text/plain)
	5503	28.928669	192.168.0.22	192.168.0.18	HTTP	60	HTTP/1.1 404 Not Found (text/html)
CASE7	5513	30.031532	192.168.0.18	192.168.0.22	HTTP	141	HEAD / HTTP/1.1
	5515	30.036252	192.168.0.22	192.168.0.18	HTTP	98	HTTP/1.1 200 OK
CASE8	5526	31.054757	192.168.0.18	192.168.0.22	HTTP	154	HEAD /NotFoundError HTTP/1.1
	5528	31.059375	192.168.0.22	192.168.0.18	HTTP	153	HTTP/1.1 404 Not Found (text/html)
CASE9	6432	31.974965	192.168.0.18	192.168.0.22	HTTP	200	PUT /test.jpeg HTTP/1.1 Continuation
	6435	31.981865	192.168.0.22	192.168.0.18	HTTP	79	HTTP/1.1 100 Continue
	6436	31.982173	192.168.0.18	192.168.0.22	HTTP	1514	Continuation
	6437	31.982173	192.168.0.18	192.168.0.22	HTTP	1514	Continuation
	6438	31.982173	192.168.0.18	192.168.0.22	HTTP	1514	Continuation
	6439	31.982173	192.168.0.18	192.168.0.22	HTTP	1514	Continuation
	6440	31.982173	192.168.0.18	192.168.0.22	HTTP	1514	Continuation
	6441	31.982173	192.168.0.18	192.168.0.22	HTTP	248	Continuation
	6445	31.988463	192.168.0.22	192.168.0.18	HTTP	60	HTTP/1.1 200 OK
CASE10	6516	33.110486	192.168.0.18	192.168.0.22	HTTP	210	PUT /400_Bad_Request.png HTTP/1.1 Continuation
	6528	33.125451	192.168.0.22	192.168.0.18	HTTP	60	HTTP/1.1 400 Bad Request (text/html)

```
> Frame 4400: 140 bytes on wire (1120 bits), 140 bytes captured
> Ethernet II, Src: ASRockIncorp_2f:1c:19 (9c:6b:00:2f:1c:19), D:
> Internet Protocol Version 4, Src: 192.168.0.18, Dst: 192.168.0
> Transmission Control Protocol, Src Port: 53435, Dst Port: 8080
```

```
0000  60 3e 5f 1b fd 9d 9c 6b  00 2f 1c 19 08 00 45 00   `>_·
0010  00 7e 47 c8 40 00 80 06  31 39 c0 a8 00 12 c0 a8   ·~G·
0020  00 16 d0 bb 1f 90 a6 5e  92 0c 2d d3 c5 65 50 18   ····
0030  02 01 5d be 00 00 47 45  54 20 2f 20 48 54 54 50   ··]·
```

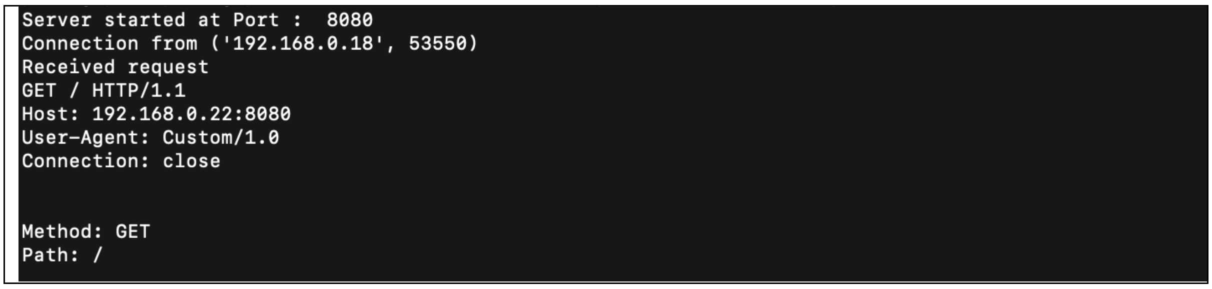
HTTP Host (http.host), 25바이트 | 패킷 수: 8311 · 표시됨: 29(0.3%) | 프로필: Default

- 캡처 필터 ip.addr == 192.168.0.22(Server PC 내부 IP Address)
- http 프로토콜 캡처

□ 실행 화면

# CASE1

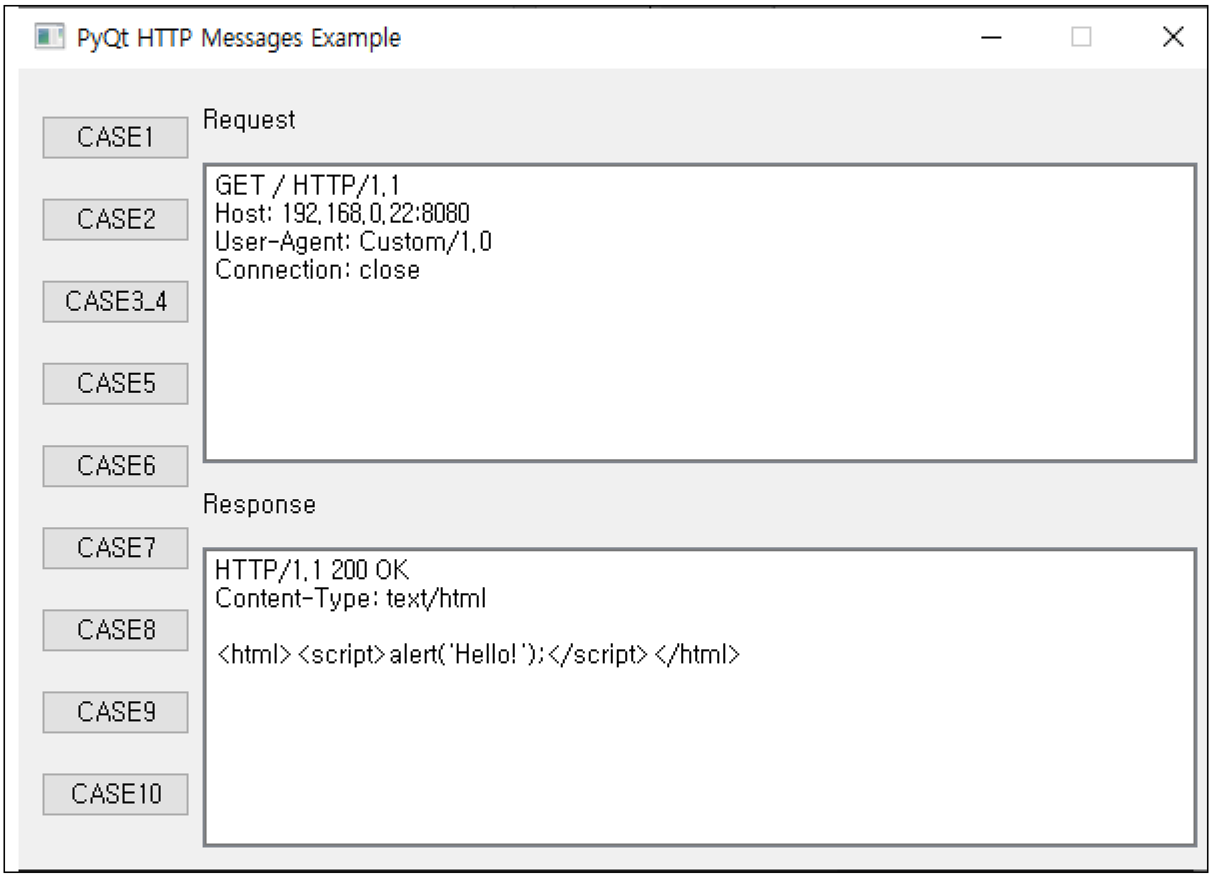
Server



```
Server started at Port :  8080
Connection from ('192.168.0.18', 53550)
Received request
GET / HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close


Method: GET
Path: /
```

Client



Request

```
GET / HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close
```

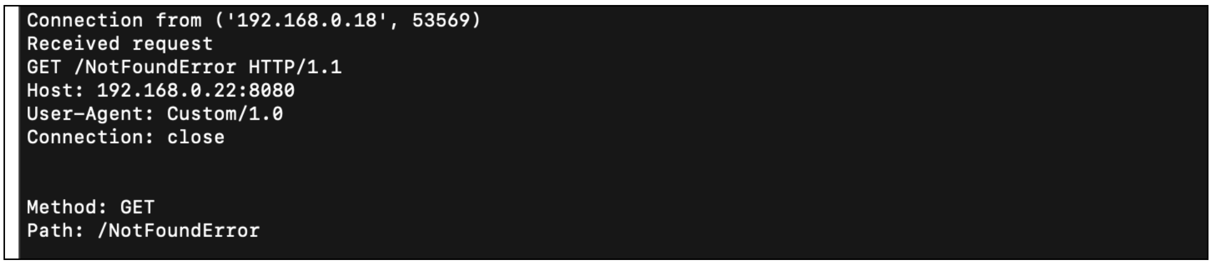
Response

```
HTTP/1.1 200 OK
Content-Type: text/html

<html> <script> alert('Hello!'); </script> </html>
```

CASE2

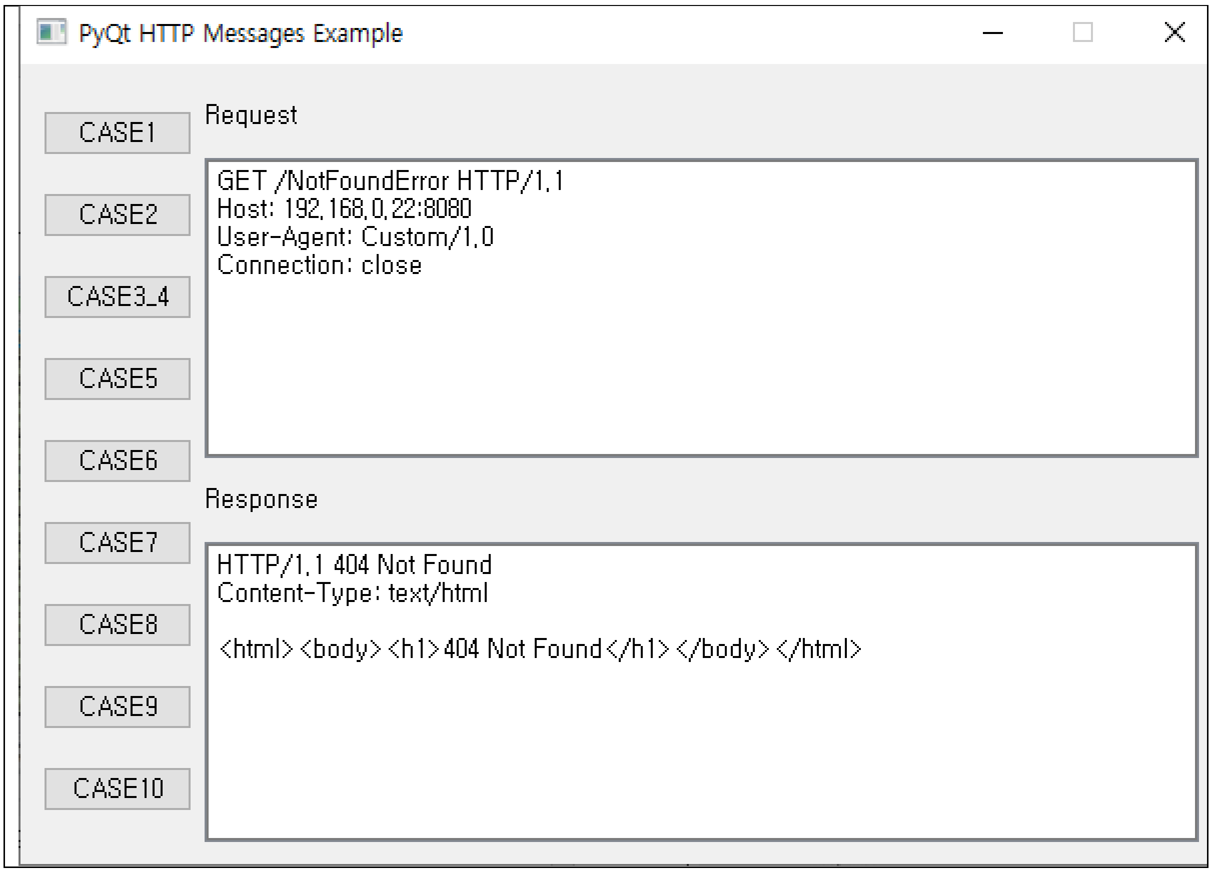
Server



```
Connection from ('192.168.0.18', 53569)
Received request
GET /NotFoundError HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close


Method: GET
Path: /NotFoundError
```

Client



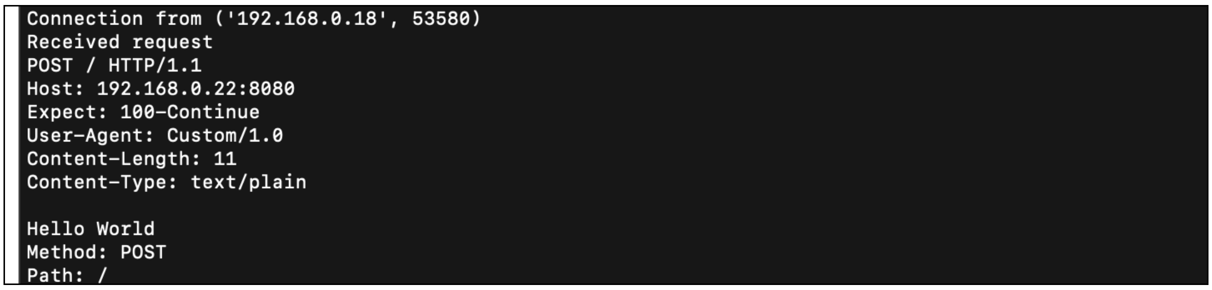
```
GET /NotFoundError HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close
```

```
HTTP/1.1 404 Not Found
Content-Type: text/html

<html> <body> <h1> 404 Not Found </h1> </body> </html>
```

CASE3_4

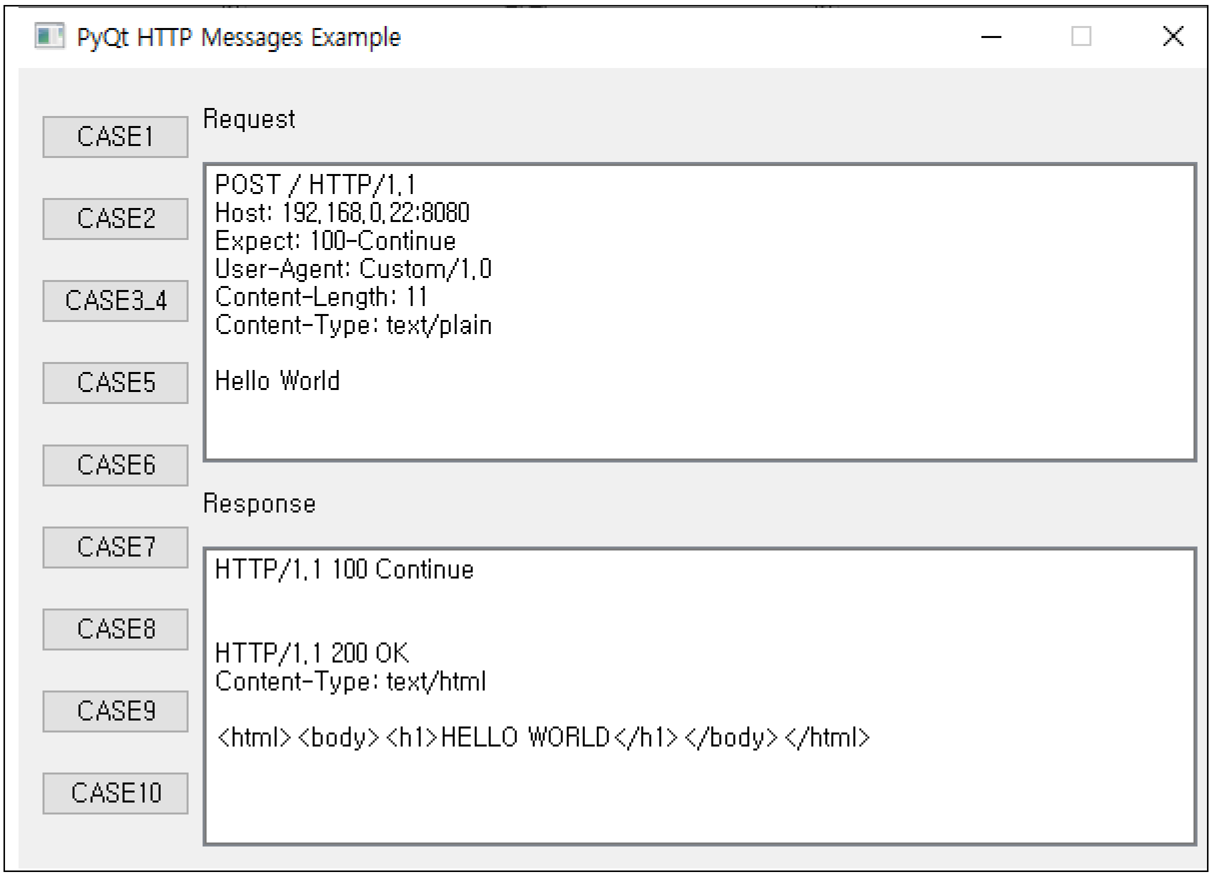
Server



```
Connection from ('192.168.0.18', 53580)
Received request
POST / HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Length: 11
Content-Type: text/plain

Hello World
Method: POST
Path: /
```

Client



Request

```
POST / HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Length: 11
Content-Type: text/plain

Hello World
```

Response

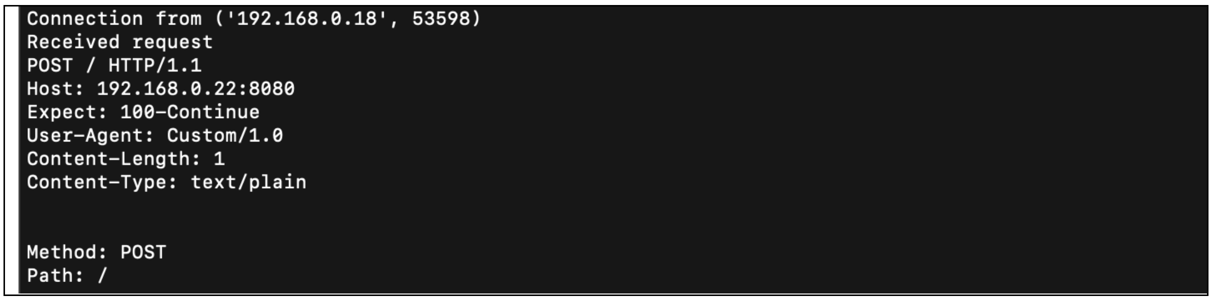
```
HTTP/1.1 100 Continue


HTTP/1.1 200 OK
Content-Type: text/html

<html><body><h1>HELLO WORLD</h1></body></html>
```

# CASE5

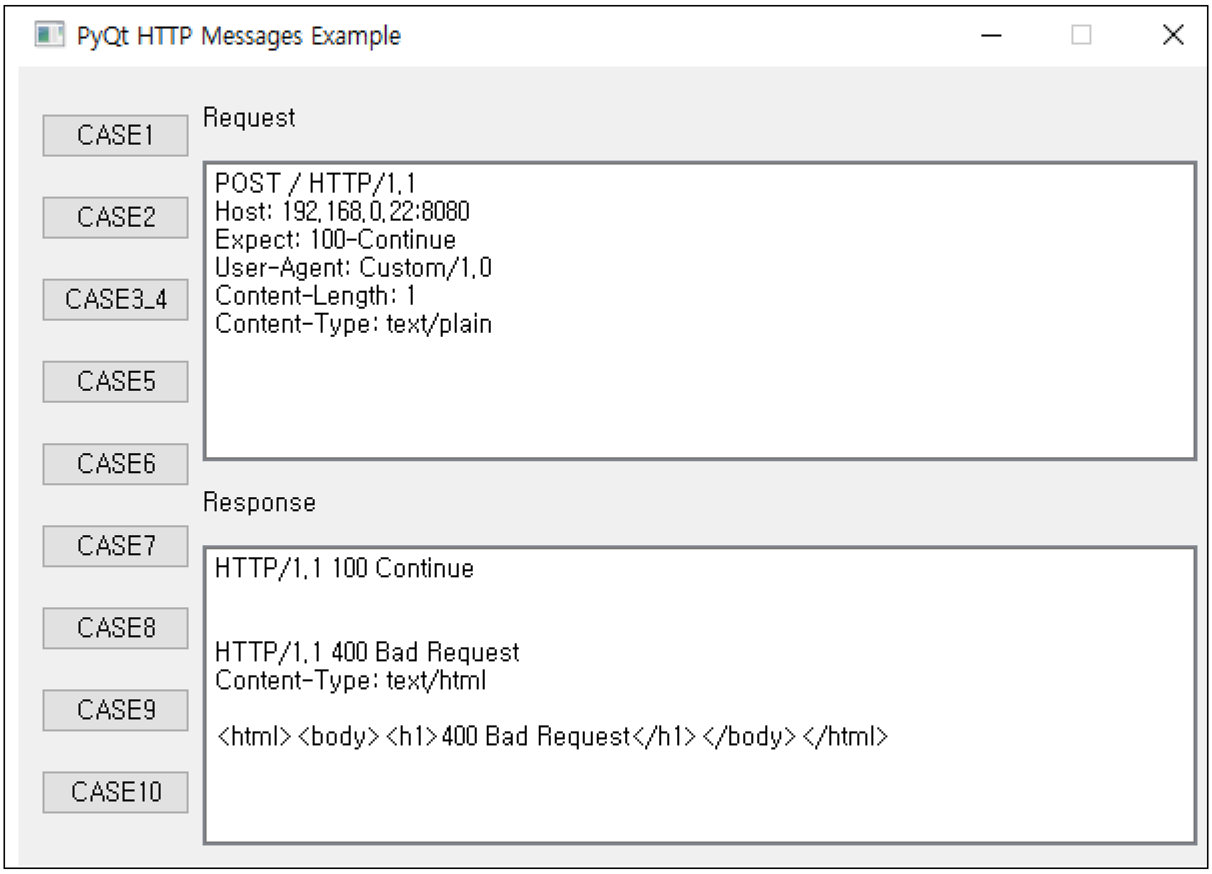
Server



```
Connection from ('192.168.0.18', 53598)
Received request
POST / HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Length: 1
Content-Type: text/plain


Method: POST
Path: /
```

Client



```
POST / HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Length: 1
Content-Type: text/plain
```

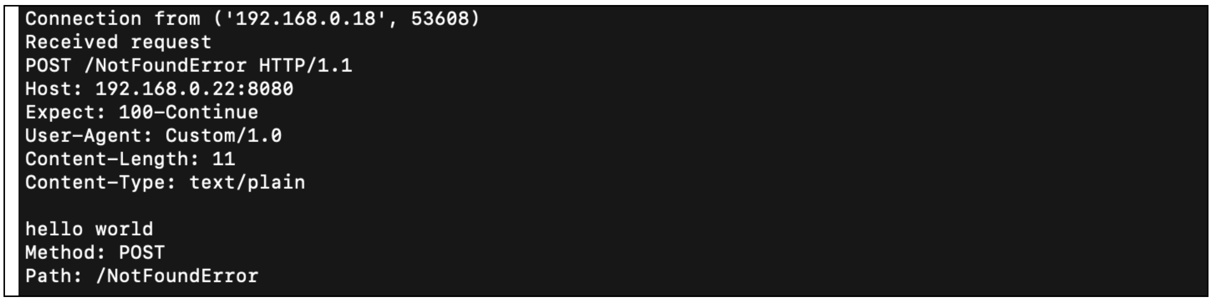
```
HTTP/1.1 100 Continue


HTTP/1.1 400 Bad Request
Content-Type: text/html

<html><body><h1>400 Bad Request</h1></body></html>
```

CASE6

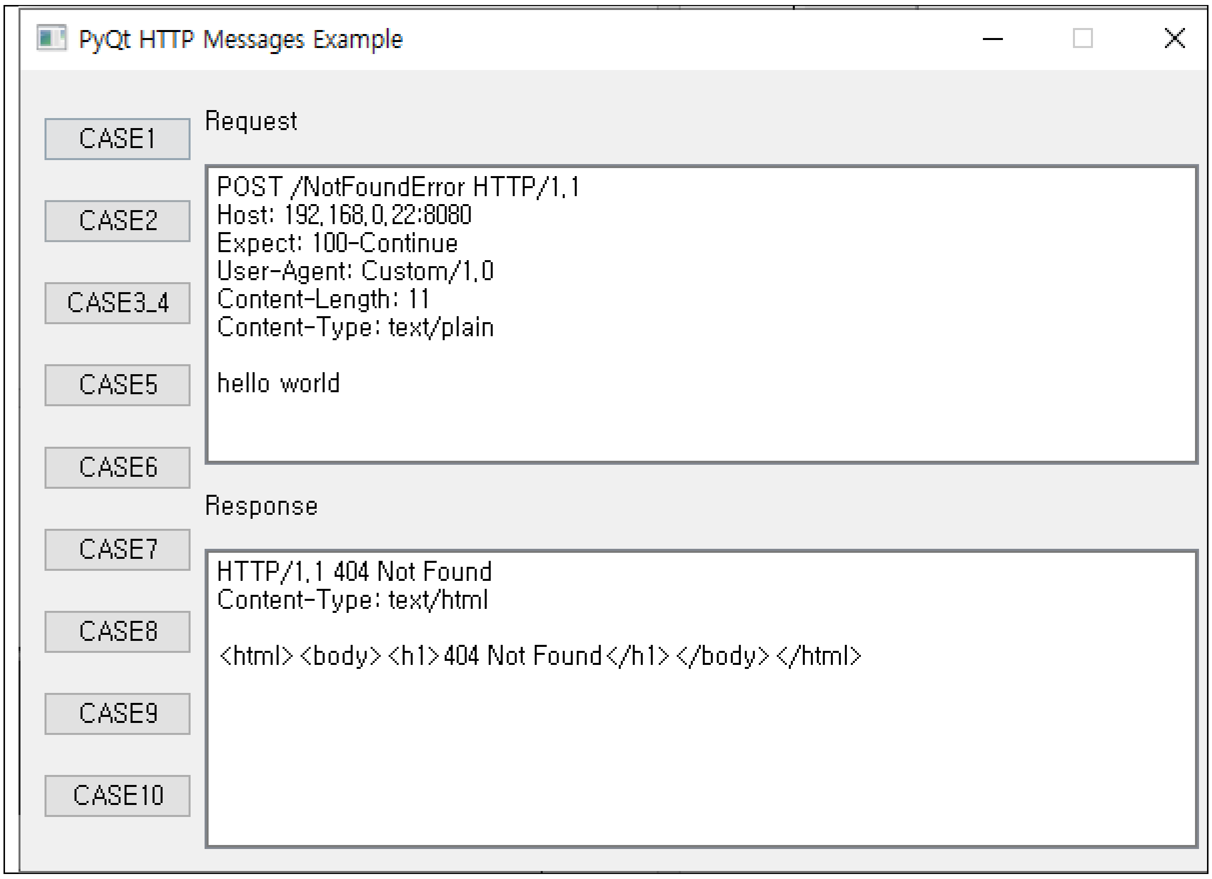
Server



```
Connection from ('192.168.0.18', 53608)
Received request
POST /NotFoundError HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Length: 11
Content-Type: text/plain

hello world
Method: POST
Path: /NotFoundError
```

Client



Request

```
POST /NotFoundError HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Length: 11
Content-Type: text/plain

hello world
```

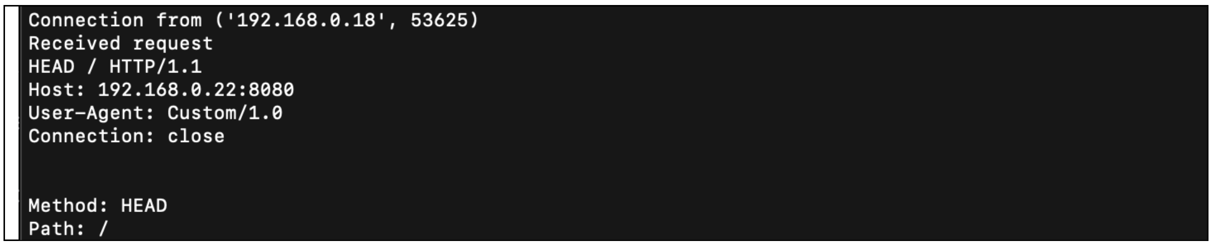
Response

```
HTTP/1.1 404 Not Found
Content-Type: text/html

<html><body><h1>404 Not Found</h1></body></html>
```

CASE7

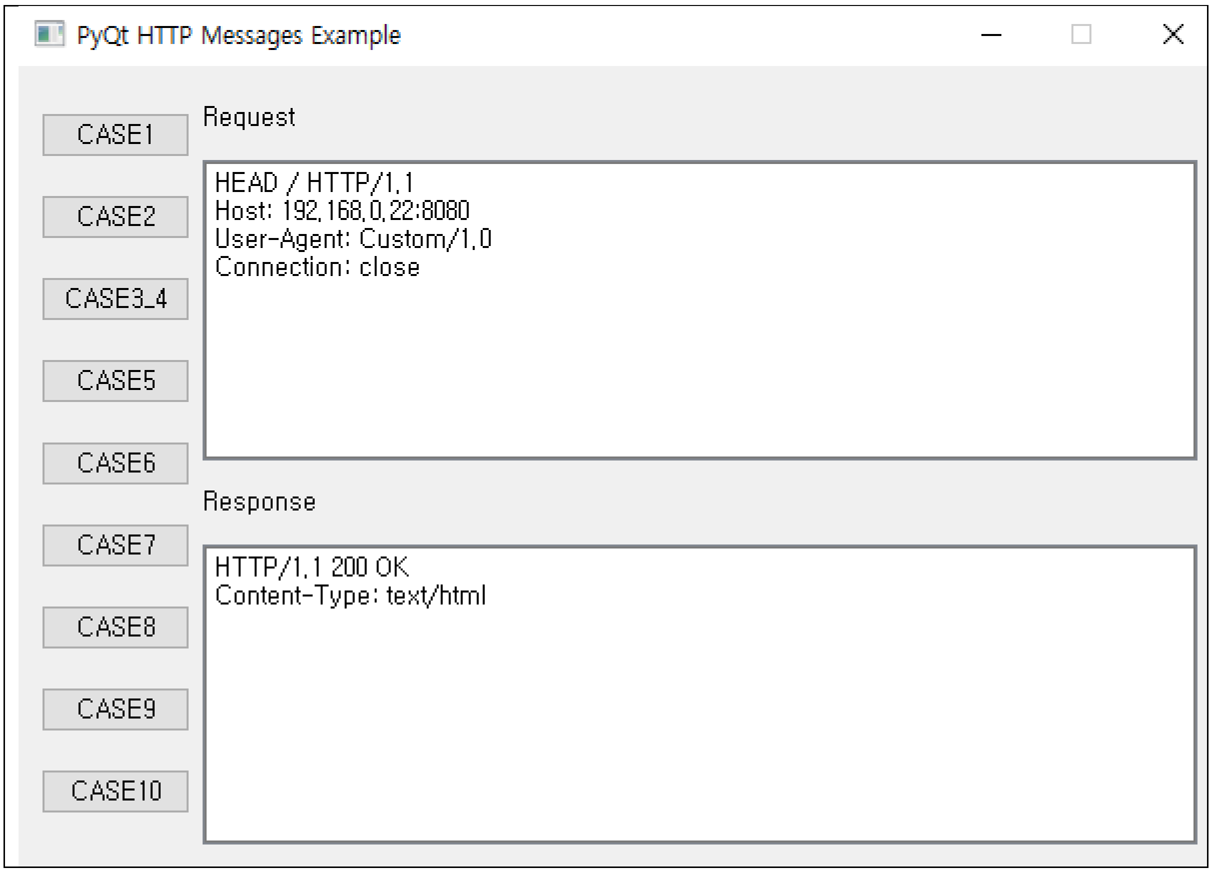
Server



```
Connection from ('192.168.0.18', 53625)
Received request
HEAD / HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close


Method: HEAD
Path: /
```

Client



Request

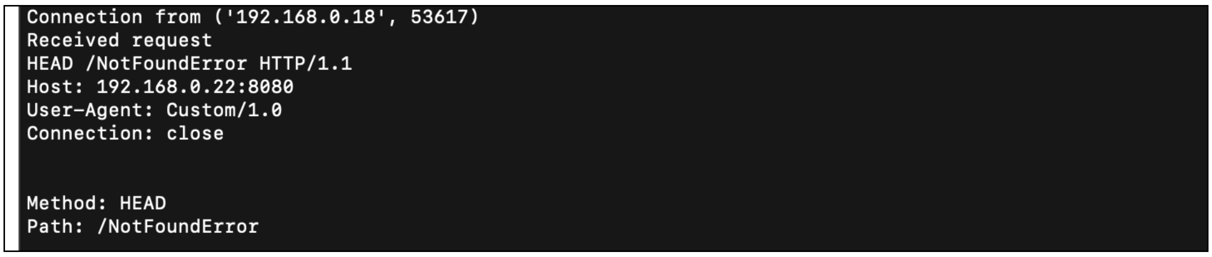
```
HEAD / HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/html
```

# CASE8

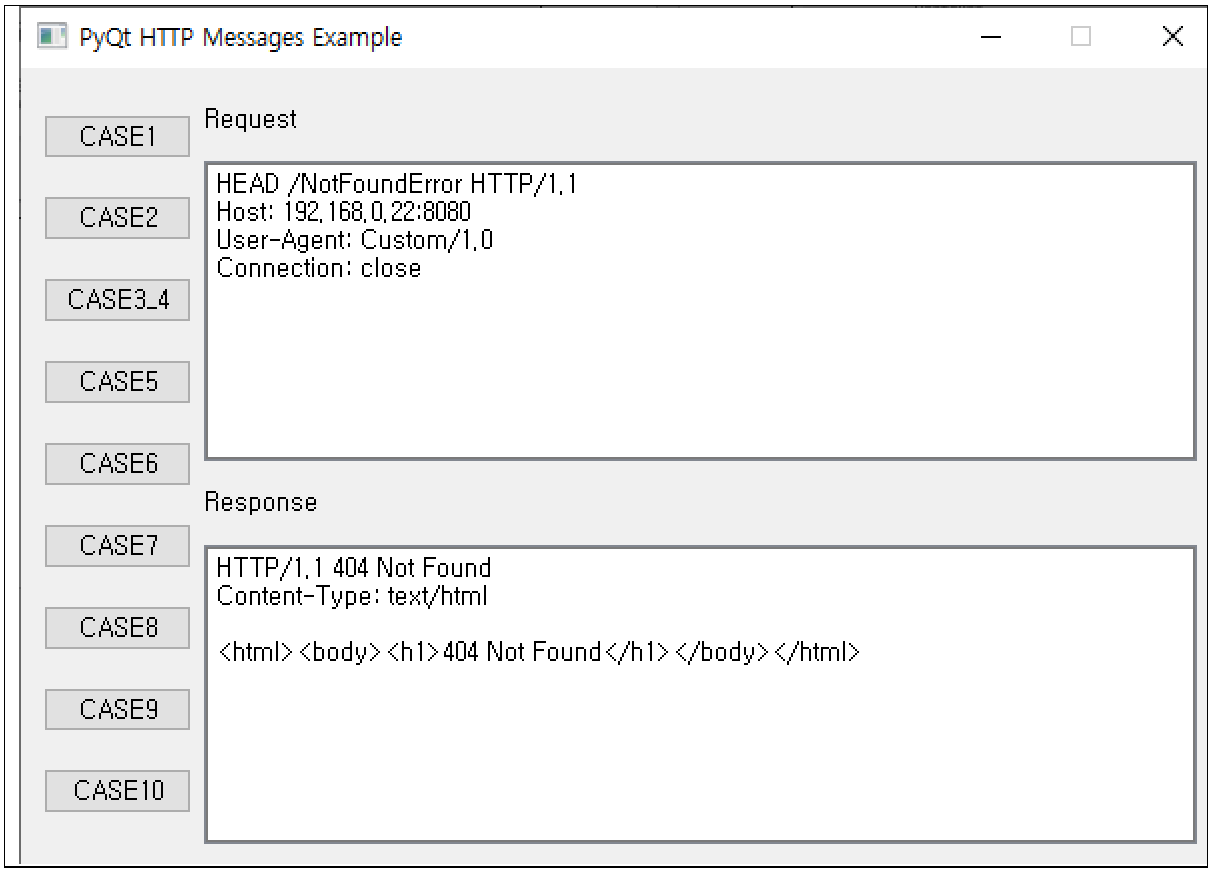
Server



```
Connection from ('192.168.0.18', 53617)
Received request
HEAD /NotFoundError HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close


Method: HEAD
Path: /NotFoundError
```

Client



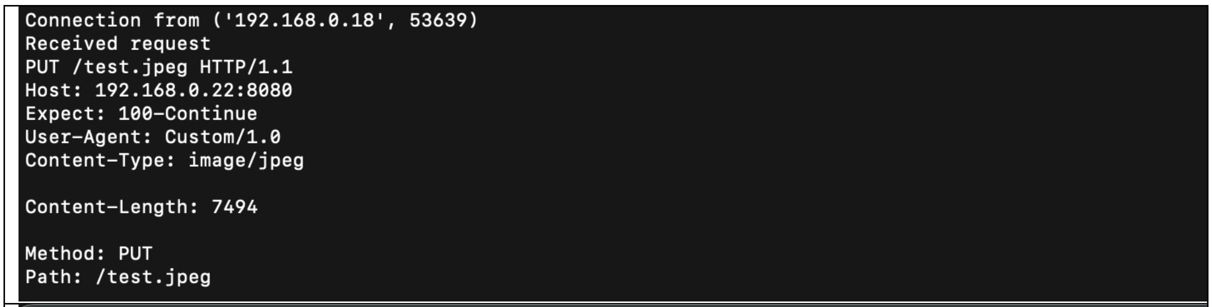
```
HEAD /NotFoundError HTTP/1.1
Host: 192.168.0.22:8080
User-Agent: Custom/1.0
Connection: close
```

```
HTTP/1.1 404 Not Found
Content-Type: text/html

<html><body><h1>404 Not Found</h1></body></html>
```

CASE9

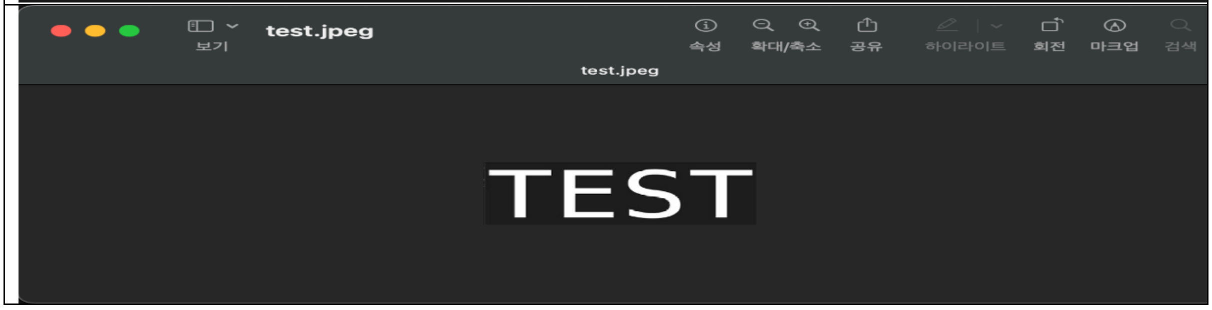
Server



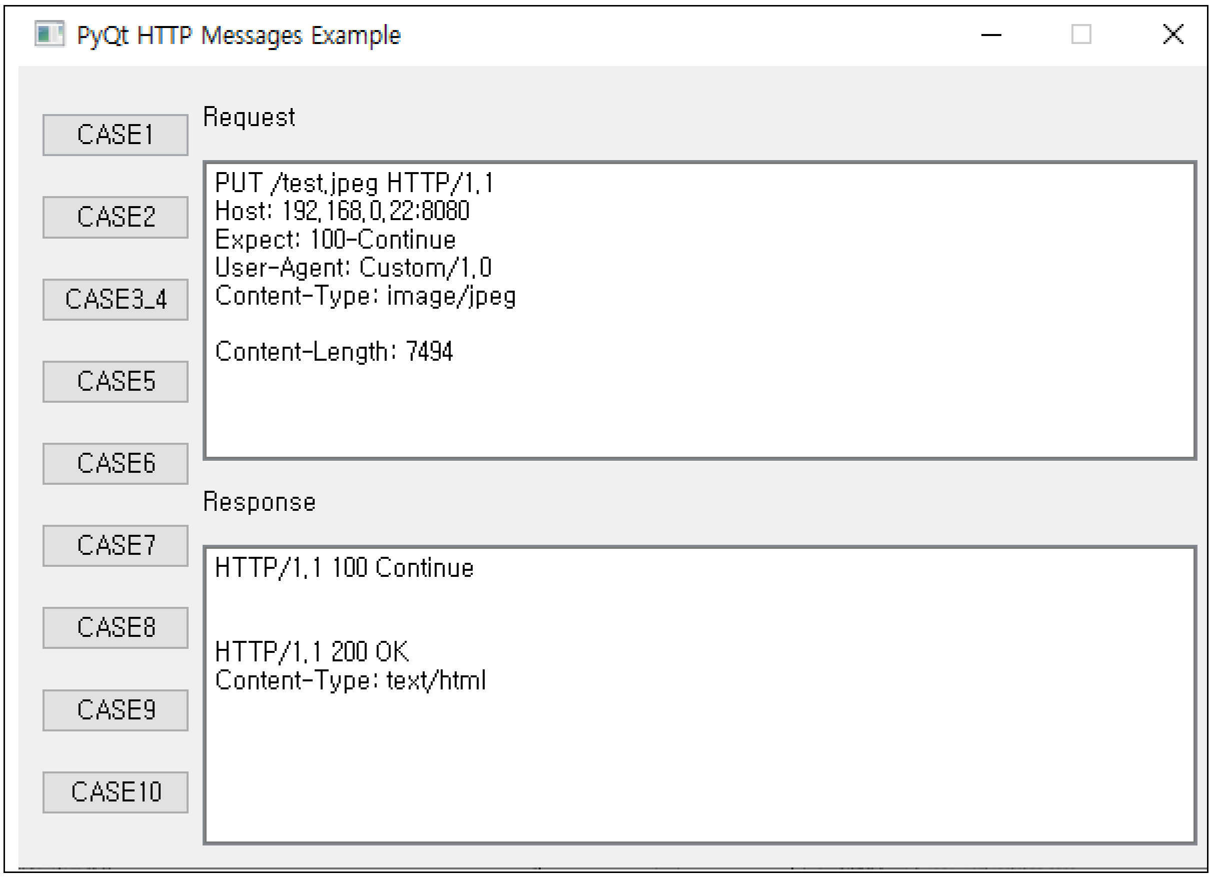
```
Connection from ('192.168.0.18', 53639)
Received request
PUT /test.jpeg HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Type: image/jpeg

Content-Length: 7494

Method: PUT
Path: /test.jpeg
```



Client



Request

```
PUT /test.jpeg HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Type: image/jpeg

Content-Length: 7494
```

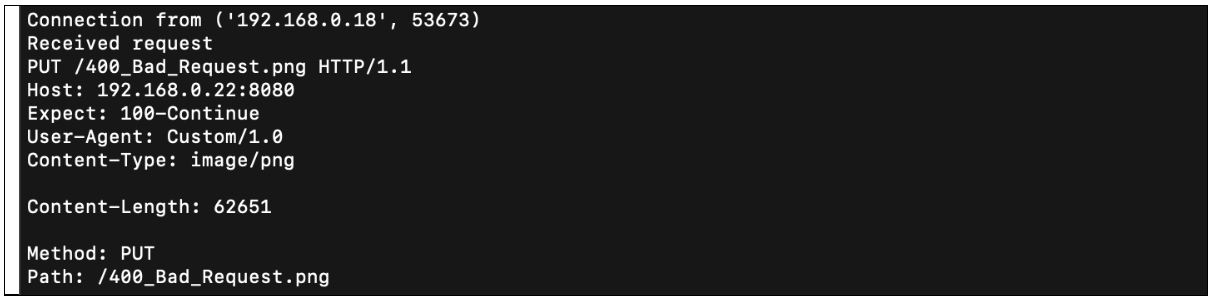
Response

```
HTTP/1.1 100 Continue


HTTP/1.1 200 OK
Content-Type: text/html
```

CASE10

Server

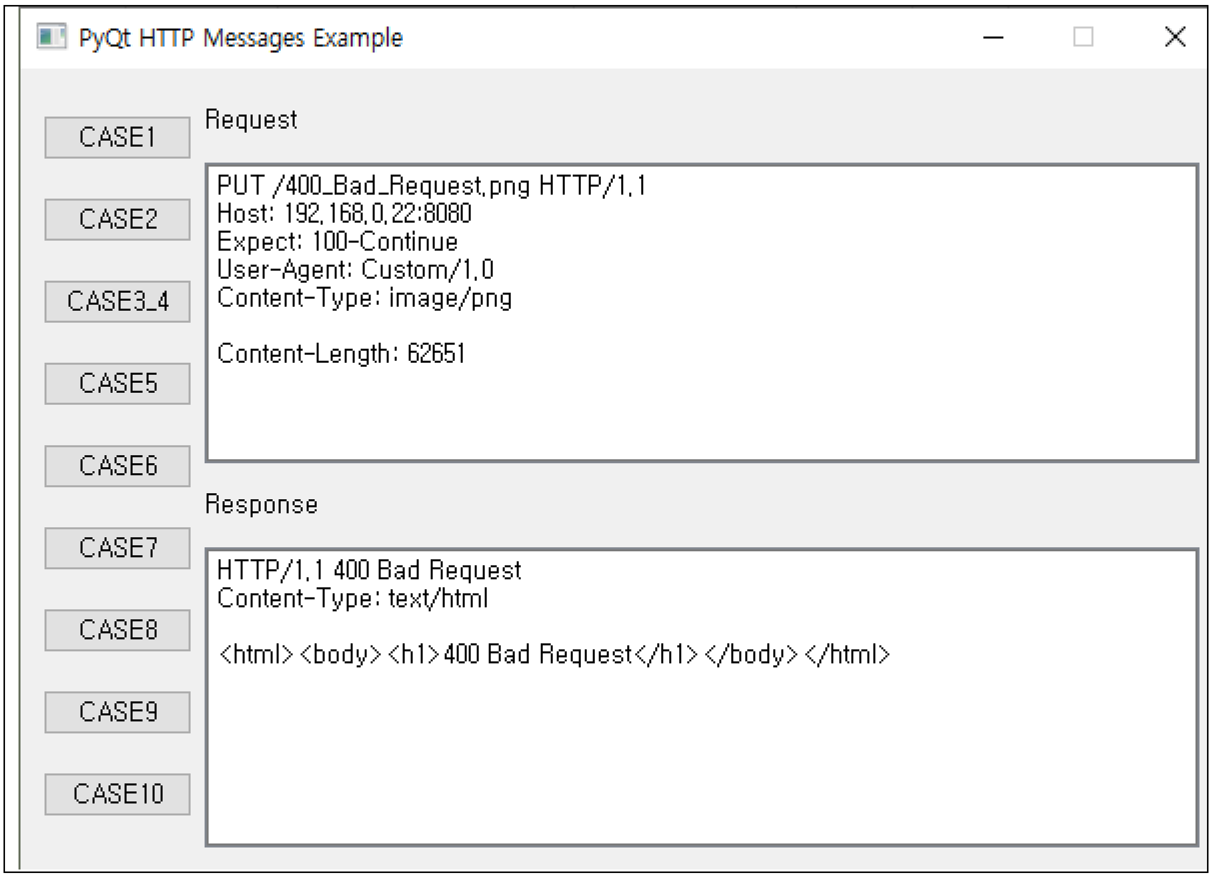


```
Connection from ('192.168.0.18', 53673)
Received request
PUT /400_Bad_Request.png HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Type: image/png

Content-Length: 62651

Method: PUT
Path: /400_Bad_Request.png
```

Client



Request

```
PUT /400_Bad_Request.png HTTP/1.1
Host: 192.168.0.22:8080
Expect: 100-Continue
User-Agent: Custom/1.0
Content-Type: image/png

Content-Length: 62651
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type: text/html

<html><body><h1>400 Bad Request</h1></body></html>
```

□ 참고문헌

HTTP RFC7231

https://datatracker.ietf.org/doc/html/rfc7231

Python Socket 프로그래밍

https://mcc96.tistory.com/58

Python PyQt5 라이브러리 사용

https://dev-guardy.tistory.com/41

HTTP Status Code

https://dev-cho.tistory.com/78