Gebze Technical University

Term Project Report

CSE476 | Mobile Communication Networks

Abdurrahman Bulut

1901042258

Assignment 1: Web Server

In this assignment, a simple Web Server is developed in Python that is capable of processing only one request. This Web Server creates a connection socket when contacted by a client (browser).

It receives the HTTP request from this connection.

It parses the request to determine the specific file being requested.

It gets the requested file from the server's file system.

It creates an HTTP response message consisting of the requested file preceded by header lines.

It sends the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in the server, It returns a "404 Not Found" error message.

Code Explanation

Server TCP port is selected as 6789 and it creates a socket. It binds to TCP port with using bind() function. After that, socket goes into listening mode with using listen() function.

```
# Prepare a server socket
port = 6789

# Fill in start
serverSocket.bind(('', port))

# put the socket into listening mode
serverSocket.listen(1)
# Fill in end
```

It confirm to be ready to serve. With accept() function, the connection is established.

```
# Establish the connection
print("Ready to serve...")
# Fill in start
connectionSocket, addr = serverSocket.accept()
# Fill in end
```

It gets file name and opens that file with open() function. After that, it reads file with using f.read() function. And it close file with using f.close().

```
# Fill in start
message = connectionSocket.recv(1024)
# Fill in end

filename = message.split()[1]
f = open(filename[1:])

# Fill in start
outputdata = f.read()
f.close()
# Fill in end
```

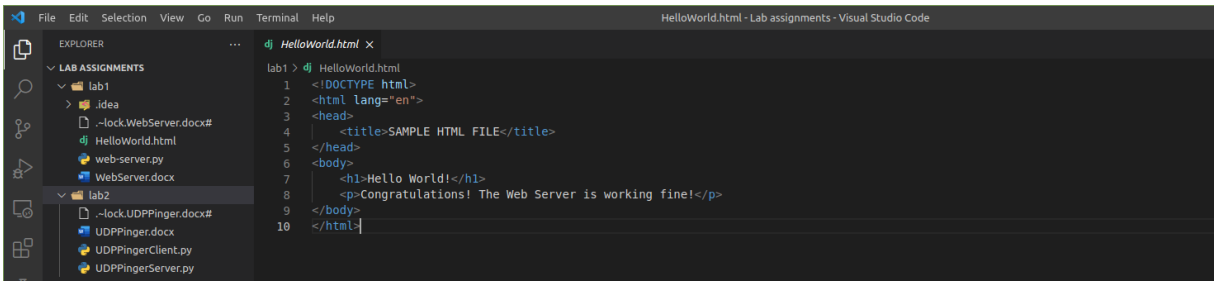
I send one HTTP header line into socket with using send() function. Information is encrypted with encode() function. Read file is send to the client with using send() function. I close socket for client.

```
# Fill in start
# Send one HTTP header line into socket
info = "HTTP/1.1 200 OK\r\n\r\n "
print(info)
connectionSocket.send(info.encode())
# Fill in end

# Send the content of the requested file to the client
for i in range(0, len(outputdata)):
    connectionSocket.send(outputdata[i].encode())
connectionSocket.close()
```

If the file is not found, 404 error will be send to the client and printed with an html page. This messages are sent with using send() function. Server socket is closed with “connectionSocket.close()” statement.

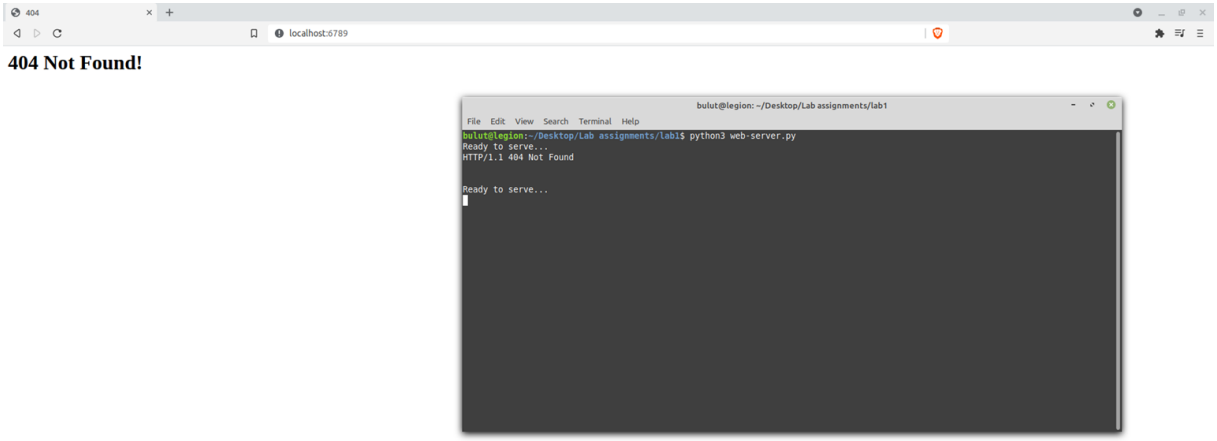
```
except IOError:

    # Send response message for file not found
    # Fill in start
    info = "HTTP/1.1 404 Not Found\r\n\n"
    print(info)
    connectionSocket.send(info.encode())

    info = "<html><head><title> 404 </title></head><body><h1>404 Not Found!</h1></body></html>\r\n"
    connectionSocket.send(info.encode())
    # Fill in end

    # Close client socket
    # Fill in start
    connectionSocket.close()
    # Fill in end
```

This is my basic Html code.

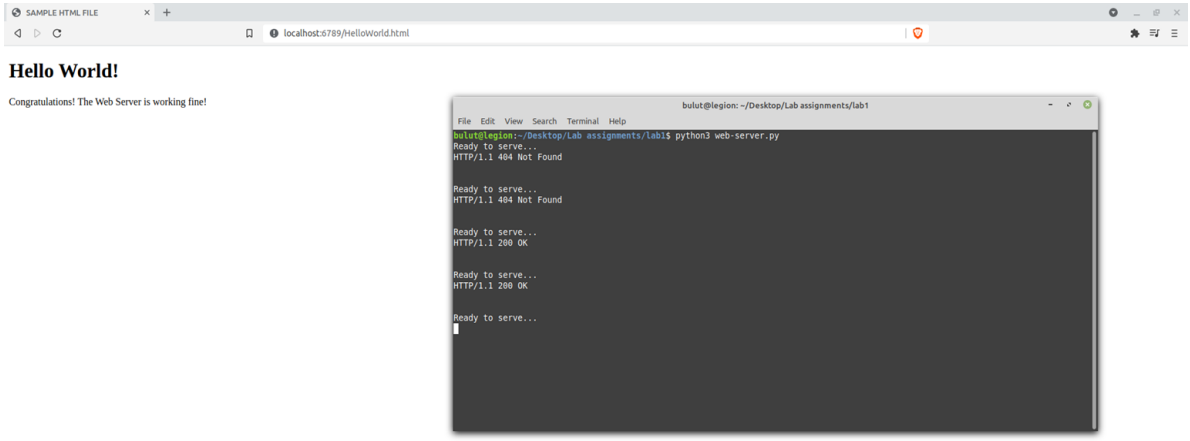


TESTS

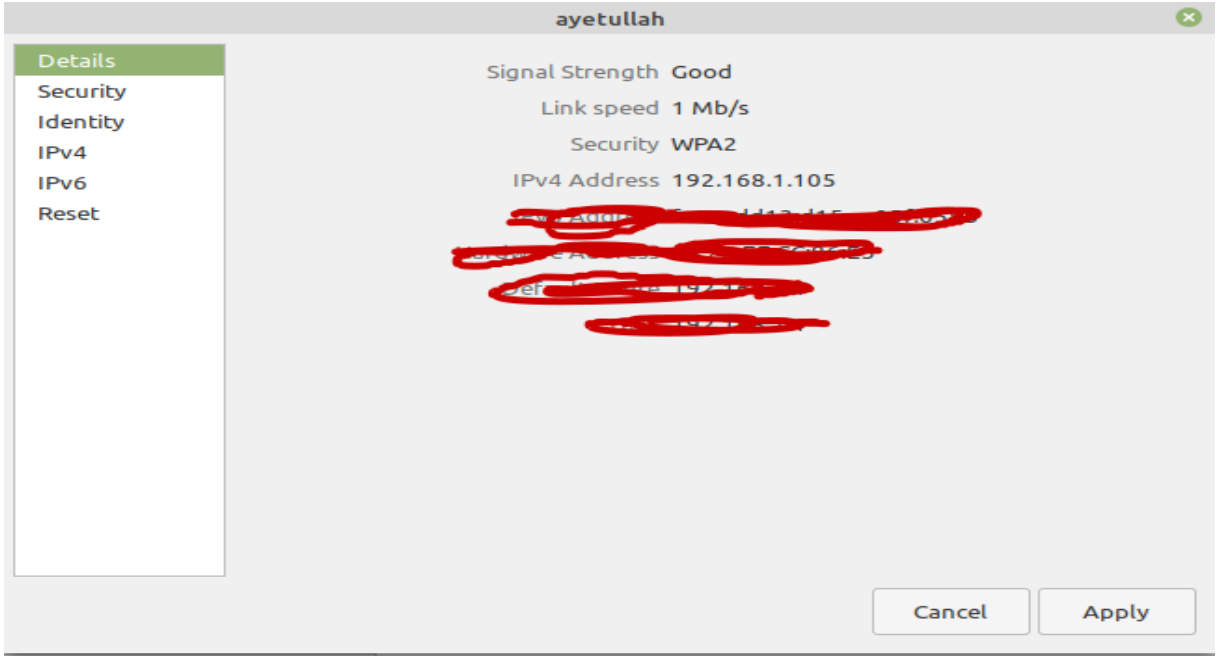
If file is not found, it will give 404 error.



If file is found, it will show the file content and print “OK” message.



My server ip is 192.168.1.105



I tested with my mobile device. If the file is not found, it will print 404 error message. If it finds the file, it prints content of file.

Assignment 2 : UDP Pinger

In this part, a client program is implemented. The client sends 10 pings to the server which is given. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. The client waits up to one second for a reply; if no reply is received within one second, the client program assumes that the packet was lost during transmission across the network. This client program sends the ping message using UDP (connectionless) and it prints the response message frome server, if any. It calculates and prints the round trip time (RTT), in seconds, of each packet, if server responses. Otherwise, it prints "Request timed out".

Code Explanation

Server Code | Already given

UDPPingerClient.py UDPPingerServer.py

lab2 > UDPPingerServer.py > ...

```python
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *

# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Generate random number in the range of 0 to 10
    rand = random.randint(0, 10)
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)
    # Capitalize the message from the client
    message = message.upper()
    # If rand is less is than 4, we consider the packet lost and do not respond
    if rand < 4:
        continue
    # Otherwise, the server responds
    serverSocket.sendto(message, address)

```

Client code:

Time module will give current time. Socket module is for communication between client and server. I will create an UDP socket and set timeout. By looking at the timeout, the client will wait up one second for reply. Sequence_number is for count client ping number. It will be in a loop for 10 times.

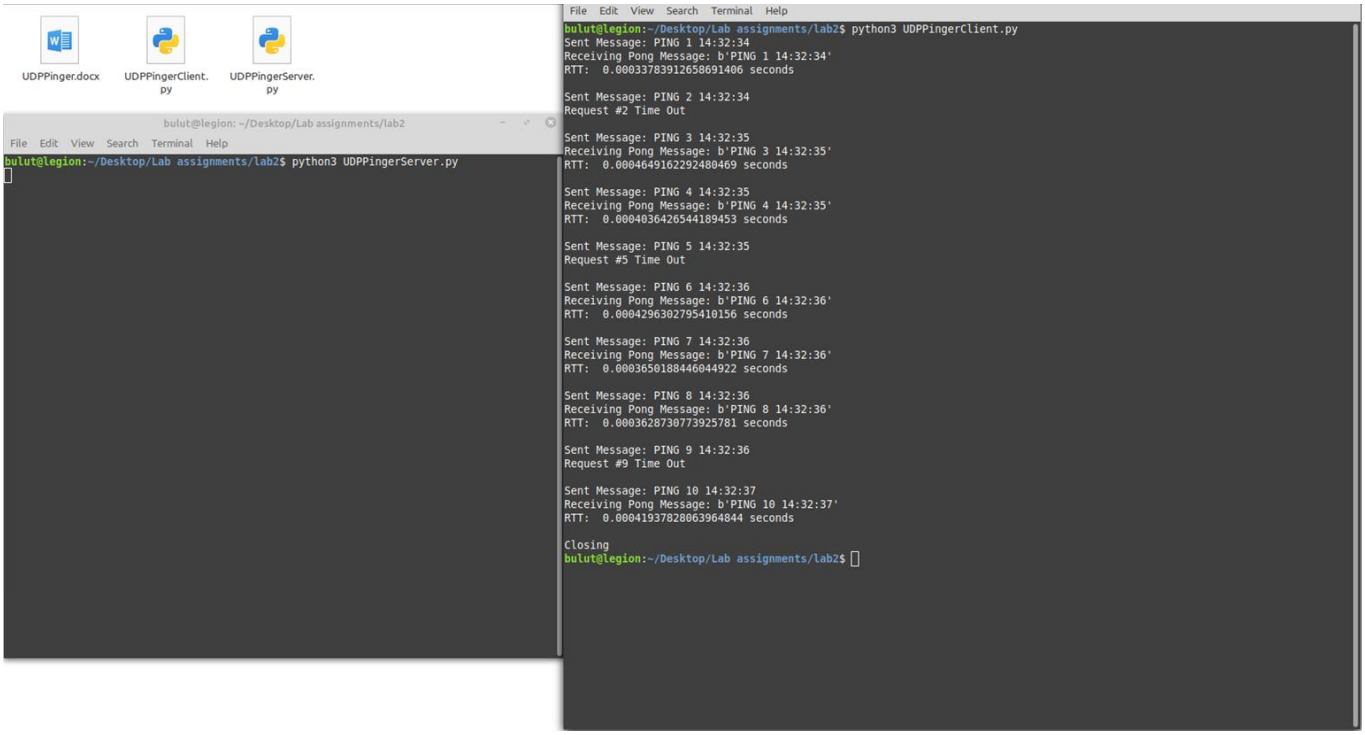
```
lab2 > UDPPingerClient.py > ...
from socket import *
from time import *

clientSocket = socket(AF_INET,SOCK_DGRAM) #create the socket.
clientSocket.settimeout(1) #set the timeout at 1 second.
sequence_number = 1 #ping counter.
```

Time() func gives us the current time. I send the ping message via localhost and also print to the terminal. After that, the program gets message frome server and calculate finished time. It calculates RTT and prints. On timeout situation, it prints timeout information and goes loop start point. In the end , it closes the socket.

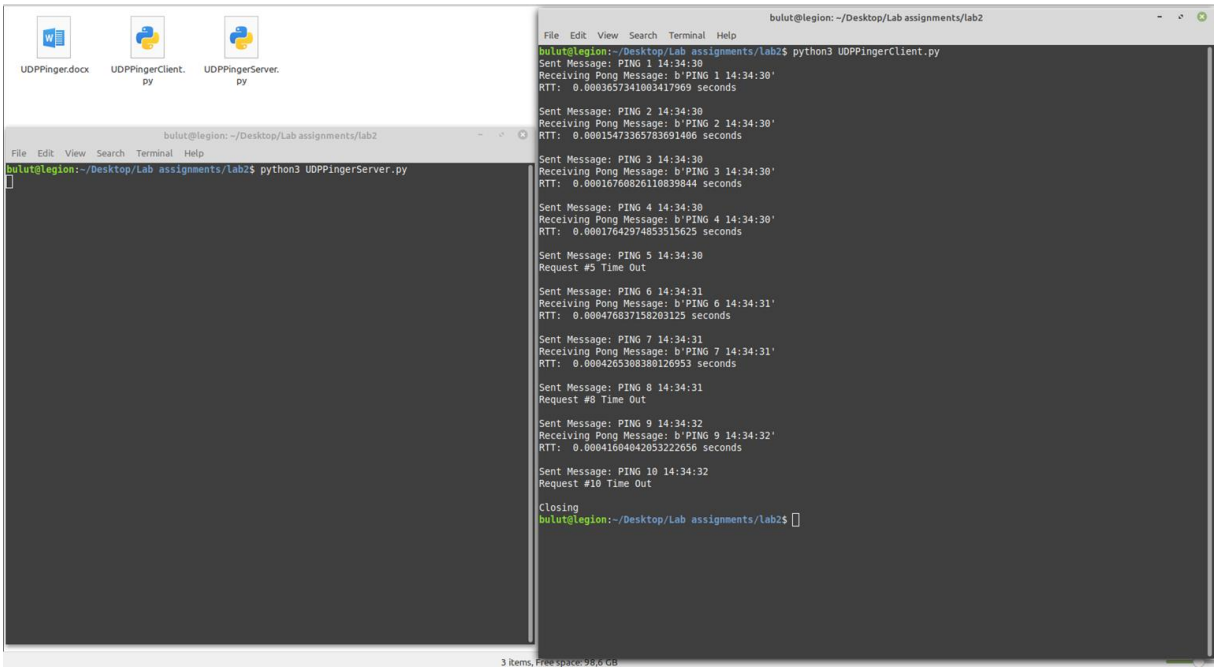
```
UDPPingerClient.py ×   UDPPingerServer.py
lab2 > UDPPingerClient.py > ...
from socket import *
from time import *

clientSocket = socket(AF_INET,SOCK_DGRAM) #create the socket.
clientSocket.settimeout(1) #set the timeout at 1 second.
sequence_number = 1 #ping counter.

while sequence_number<=10:

    startTime = time() # keep current time.

    try:
        message = "PING " + str(sequence_number) + " " + str(strftime("%H:%M:%S"))
        clientSocket.sendto(message.encode(), ('Localhost', 12000)) #send ping message via localhost.
        print("Sent Message: " + message)

        data, address = clientSocket.recvfrom(1024) #recieving message from server.
        responseTime = time()   #finish time.
        rtt=responseTime-startTime # the round trip time (RTT)

        print("Receiving Pong Message:",data)
        print("RTT: ", str(rtt) + " seconds\n")

    except timeout:
        print("Request #" + str(sequence_number) + " Time Out\n")

    sequence_number += 1

print("Closing")
clientSocket.close()
```

Tests

test-1: (3 time out – packet loss)



test-2: (3 time out – packet loss)



Assignment 3 : Mail Client

A mail client is developed which sends email to any recipient. This client establishes a TCP connection with Google mail server, dialogues with the mail server using the SMTP protocol, sends an email message to a recipient via the mail server, and finally closes the TCP connection with the mail server.

I chose Google server for mail server.

Google Gmail SMTP server: smtp.gmail.com

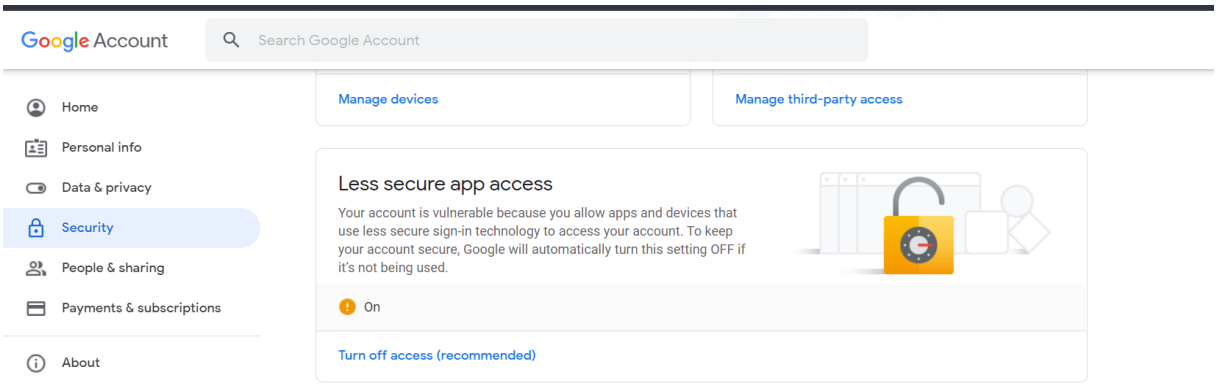
Gmail SMTP port: 587(TLS)

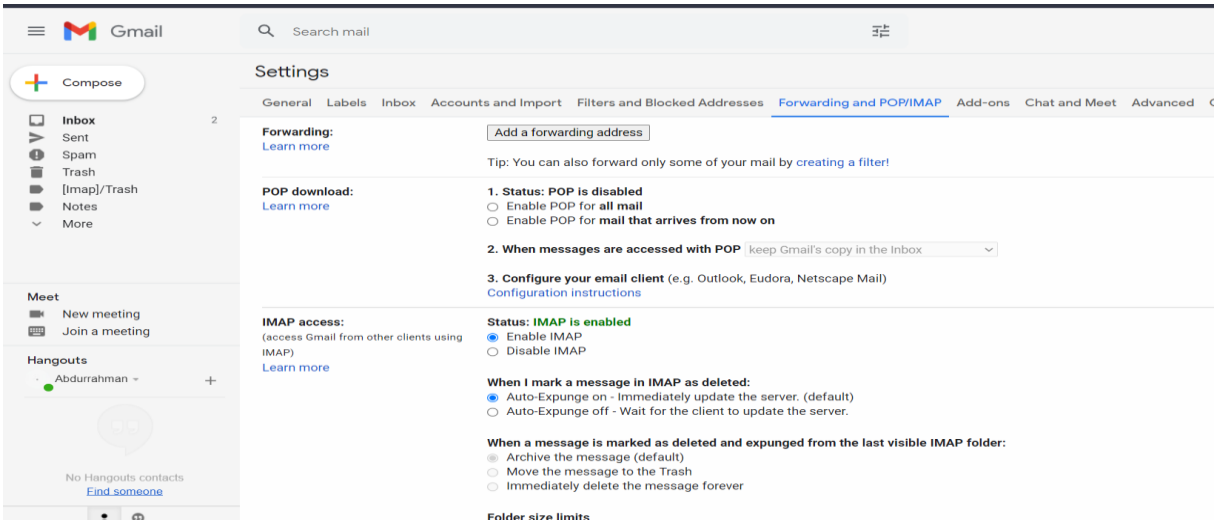
Transport Layer Security(TLS) and Secure Sockets Layer(SSL) is added for authentication and security reasons.

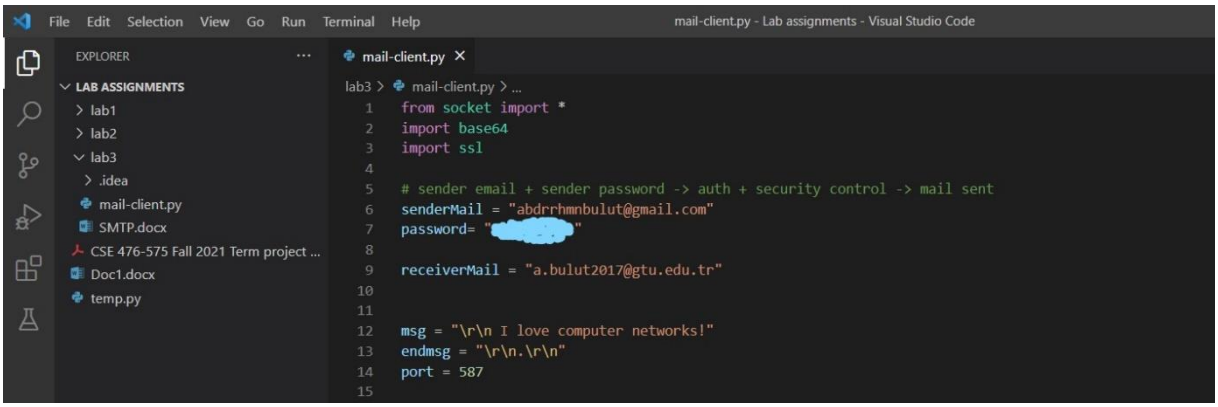
TESTS

Socket module is needed for server and client communication. Base64 module is used for encoding mail address and password. Ssl module is for connecting google gmail server. Gmail SMTP port: 587(TLS). Msg is a message that will be sent to reveiver mail address.

Gmail can create some problems for security when sending mail. There may be 2 solutions. One of them is to enable IMAP access. To do this, go Gmail all settings page. Click “Forwarding and POP/IMAP” tab and enable IMAP. The Other one is to let less secure app access. To do this, go Google account settings and security part. Enable Less secure app access. I used my Google account as sender email and my school email as receiver email and I used Python 2.7.18 version.







Google Gmail SMTP server: smtp.gmail.com with port 587. I created a socket called clientSocket and established a TCP connection with mailserver between mail server and client. Line 33 send HELO command and them prints server response message. If no response is received, the error message will be printed.

```
16
17  # Choose a mail server (I choose Google mail server) and call mailserver
18  mailserver = ("smtp.gmail.com", port)
19
20  # Create socket called clientSocket and establish a TCP connection with mailserver
21  #Fill in start
22  clientSocket = socket(AF_INET, SOCK_STREAM)
23  clientSocket.connect(mailserver)
24  #Fill in end
25
26  recv = clientSocket.recv(1024)
27  print (recv)
28  if recv[:3] != '220':
29      print ('220 reply not received from server.')
30
31  # Send HELO command and print server response.
32  heloCommand = 'HELO Alice\r\n'
33  clientSocket.send(heloCommand)
34  recv1 = clientSocket.recv(1024)
35  print (recv1)
36  if recv1[:3] != '250':
37      print ('250 reply not received from server.' )
38
```

In this part I open a TLS for Gmail mail server. It is needed for security reasons. It encrypes TLS message which is “starttls” and sends to the mail server via send() function. At line 49, It make a SSL socket for security reasons.

```
38
39
40  #TLS for Google mail server
41  starttls = 'starttls\r\n'
42  clientSocket.send((starttls).encode())
43  recv2=clientSocket.recv(1024)
44  print (recv2)
45  if recv2[:3] != '220':
46      print ('220 reply not received from server.')
47
48
49  clientSocket = ssl.wrap_socket(clientSocket)
```

Email address and password are encrypted with encode() function. After that, they are encrypted with b64encode() function as one piece. Finally, auth message is created and sent to the mail server.

```
# AUTH with base64 encoded senderMail and password
base64_str = ("\x00"+senderMail+"\x00"+password).encode()
base64_str = base64.b64encode(base64_str)
authMsg = "AUTH PLAIN ".encode()+base64_str+"\r\n".encode()
clientSocket.send(authMsg)
recv_auth = clientSocket.recv(1024)
print(recv_auth.decode())
if recv1[:3] != '250':
    print('250 reply not received from server.')
```

This part has sender and receiver informations. This part encryptes terms and sends to the mail server .

```
# Send MAIL FROM command and print server response.
# Fill in start
mailFrom = "MAIL FROM: <"+senderMail+"> \r\n"
clientSocket.send(mailFrom.encode())
recv6 =clientSocket.recv(1024)
print (recv6)
if recv6[:3] != '250':
    print ('250 reply not received from server.')
# Fill in end

# Fill in start
# Send RCPT TO command and print server response.
rcptTo = "RCPT TO: <"+receiverMail+"> \r\n"
clientSocket.send(rcptTo.encode())
recv7 = clientSocket.recv(1024)
print (recv7)
if recv7[:3] != '250':
    print ('250 reply not received from server.')
# Fill in end

# Fill in start
```

DATA part declares mail content. In message part, Subject can be defined like in picture. If we put newline in Subject message, the part after newline will be the content.

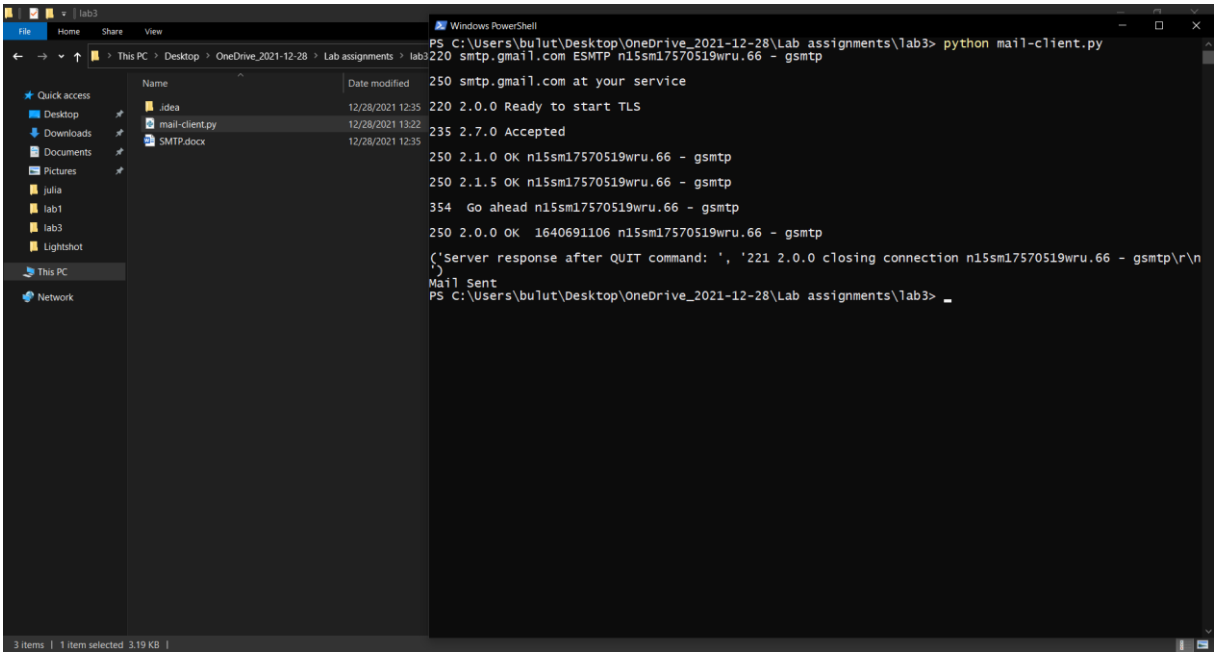
```
# Fill in start
# Send DATA command and print server response.
data = "DATA\r\n"
clientSocket.send(data.encode())
recv8 = clientSocket.recv(1024).decode()
print (recv8)
if recv8[:3] != '354':
    print ('354 reply not received from server.')
# Fill in end


# Fill in start
# Send message data.
message = 'SUBJECT: SMTP Mail Client Testing \nContent \r\n'
clientSocket.send(message.encode())
clientSocket.send(msg.encode())
# Fill in end
```

In this part, the message ends with single period. Quit command is sent and server response is received. At the end of code, socket is closed.

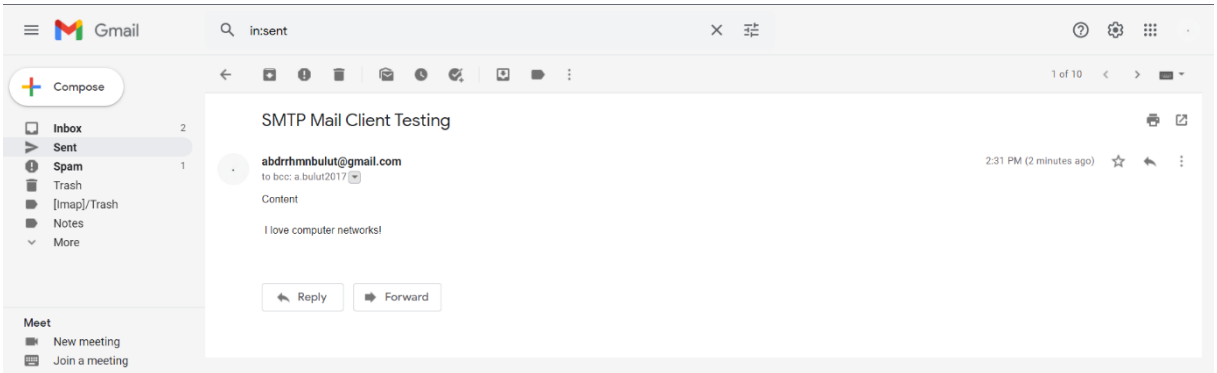
```
# Fill in start
#message ends with single period.
clientSocket.send(endmsg.encode())
recv9 = clientSocket.recv(1024)
print (recv9.decode())
if recv9[:3] != '250':
    print ('250 reply not received from server.')
# Fill in end


# Fill in start
# Send QUIT command and get server response.
clientSocket.send("QUIT\r\n".encode())
recv10=clientSocket.recv(1024)
print ('Server response after QUIT command: ', recv10)
if recv10[:3] != '221':
    print ('221 reply not received from server.')

print("Mail Sent")
clientSocket.close() #close socket.
# Fill in end
```

TESTS

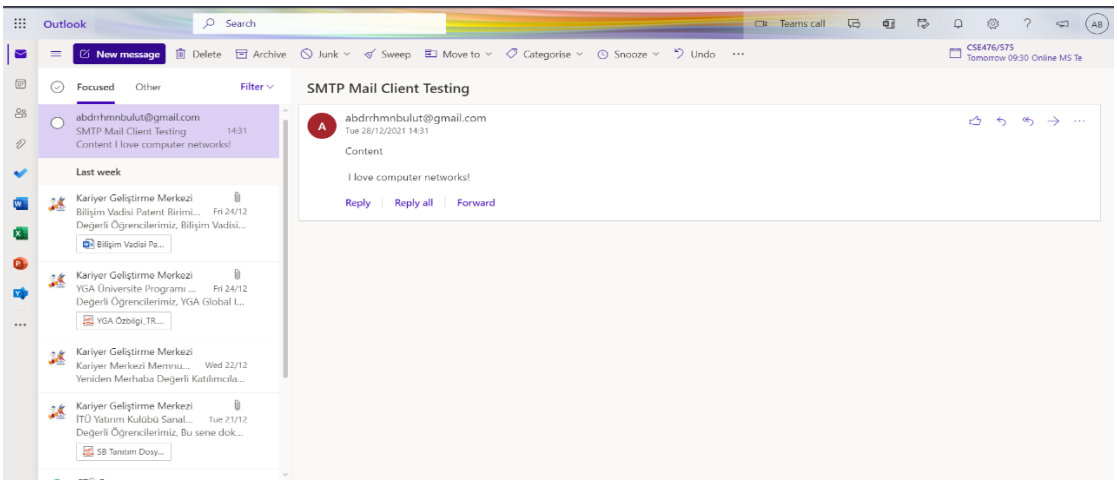
Code work-1 | Powershell | gmail -> gtu mail



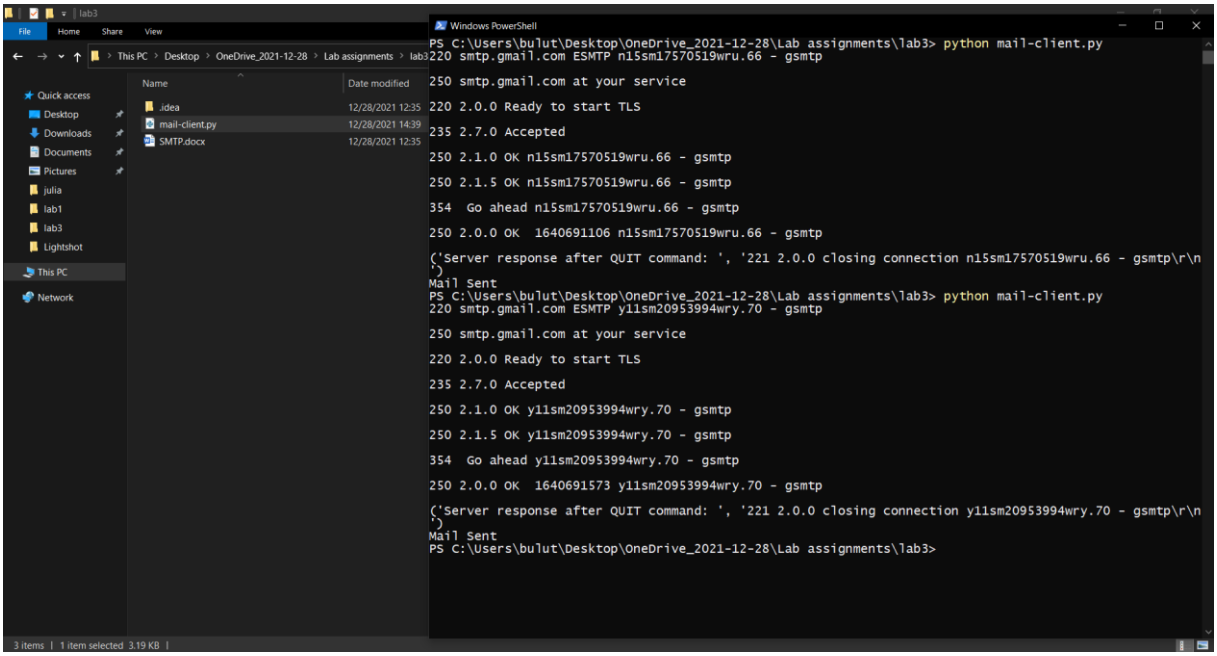
Sender : abdrrhmnbulut@gmail.com



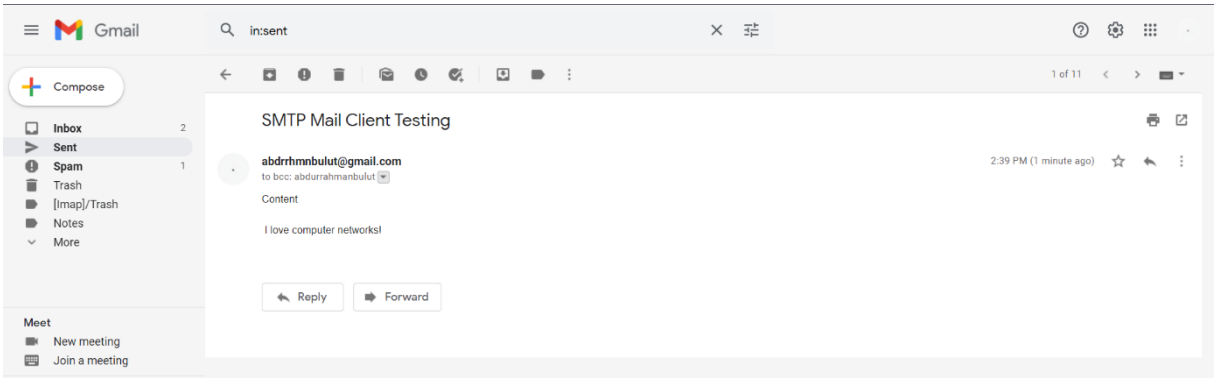
Receiver : a.bulut2017@gtu.edu.tr



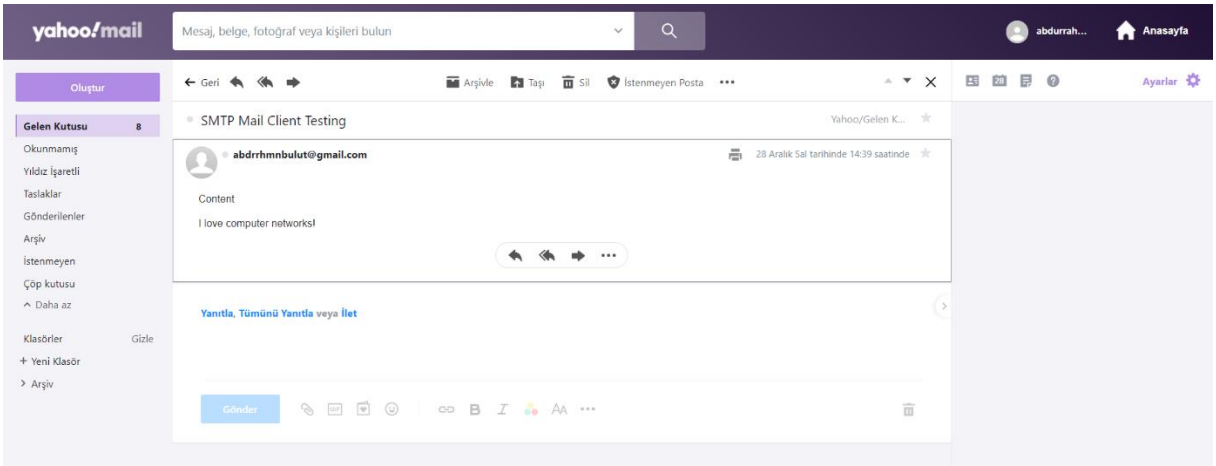
Code work-2 | Powershell | gmail -> yahoo mail



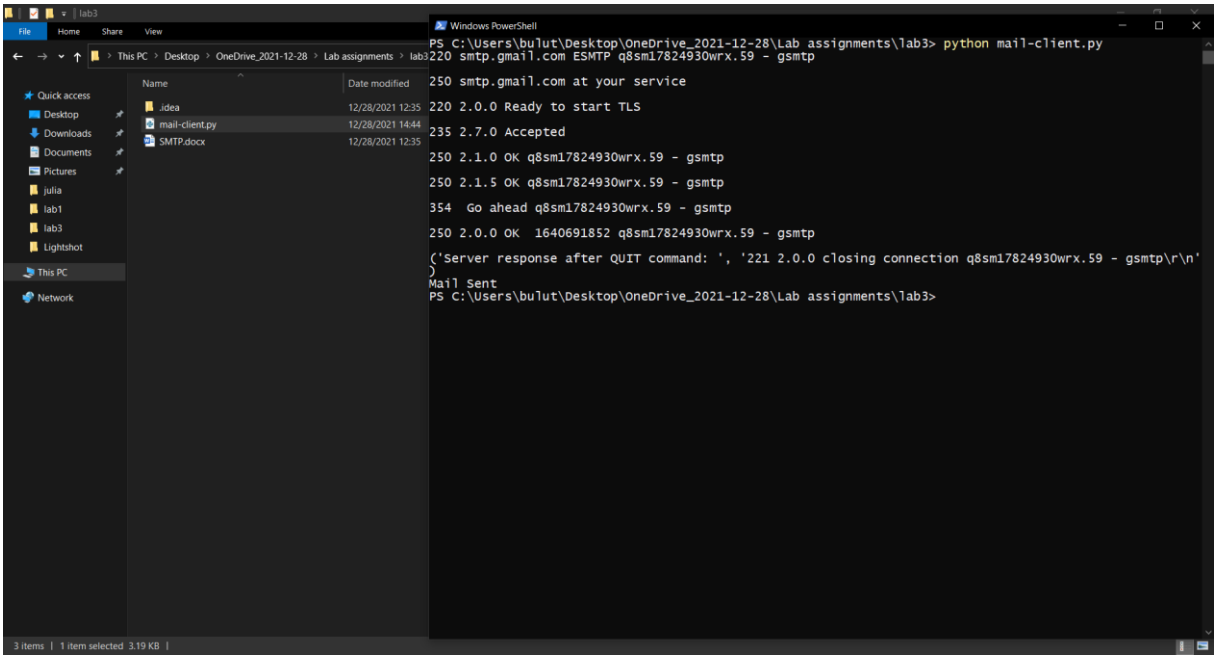
Sender : abdrrhmnbulut@gmail.com



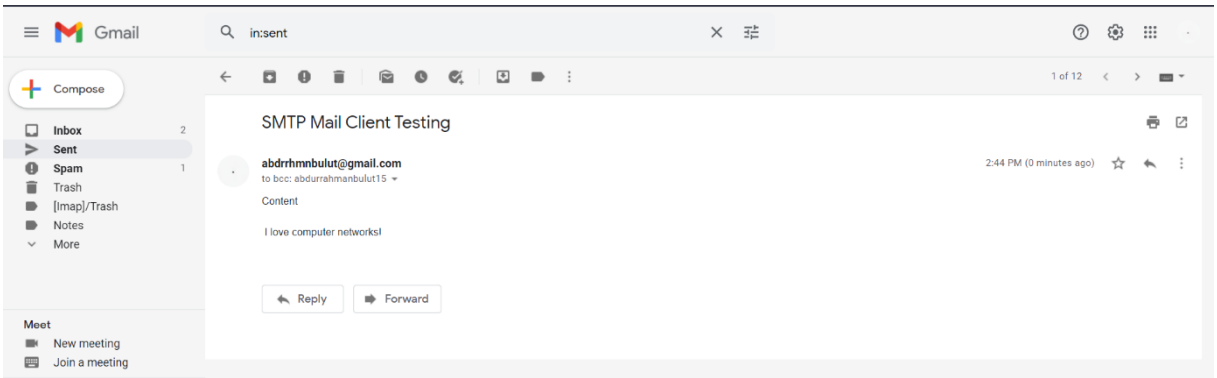
Receiver : abdurrahmanbulut@yahoo.com



Code work-3 | Powershell | gmail -> gmail



Sender : abdrrhmnbulut@gmail.com



Receiver : abdurrahmanbulut15@gmail.com