Gebze Technical University

Computer Engineering

CSE476 - Mobile Communication Networks

Term Project - Final Report

Student:
Nevra GÜRSES
161044071

1 PART 1 - Web Server

1.1 Problem Definition

In this part, we will develop a simple Web Server in Python that is capable of processing only one request.Our Web server will do below operations:

- Socket when contacted by a client that is browser.
- Receive the HTTP request via this connection.
- Parse the request to determine the specific file being requested.
- Get the requested file from the server's file system.
- Create an HTTP response message consisting of the requested file preceded by header lines.
- Send the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in your server, our server should return a "404 Not Found" error message.

1.2 Operations Made for This Part:

For this part, I did given all operations with using skeleton code. I completed the code, run my server, and then tested my server by sending requests from browsers running on different hosts.

If I explain the operations for doing web server with given requirements:

- Firstly, I import socket module.

- Then, I prepare a server socket with binding localhost or different hosts and port number 8080.This socket is prepared as listening socket.

- After, in while loop, I establish the connection.

- Then, I receive message from client and parse this message for getting file name.

- After that, I open file and read the contents.

- Then, I send one HTTP header line into socket.

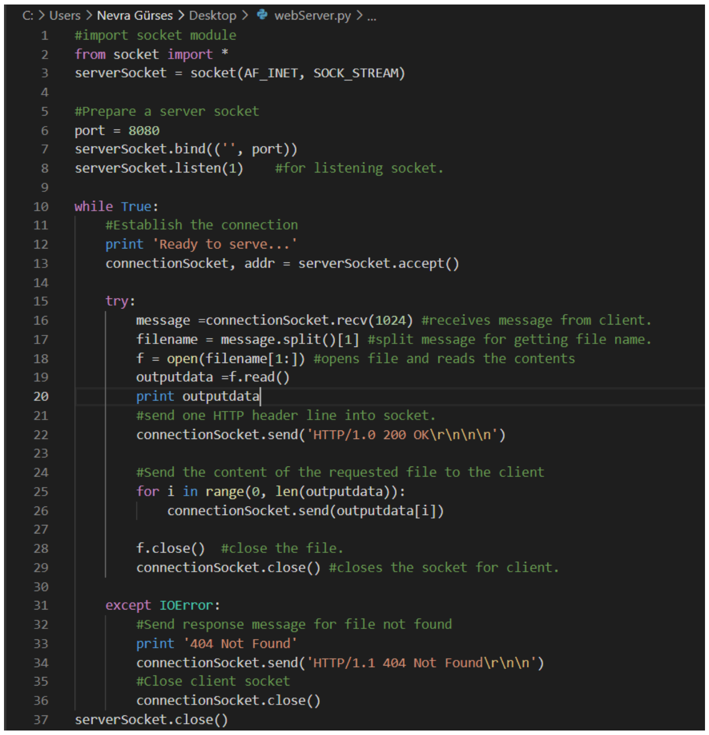
- After that, I send the content of the requested file to the client.

- Then, I close file and socket for client.

- If file not found, in IOError exception, I send message about 404 error not found.

- Last, I close server socket.

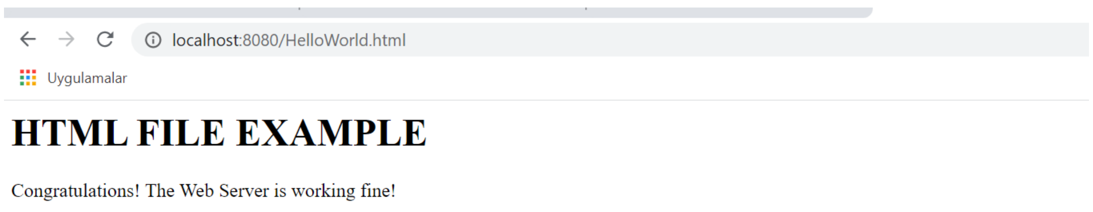
1.3 Code of Web Server



```python
#import socket module
from socket import *
serverSocket = socket(AF_INET, SOCK_STREAM)

#Prepare a server socket
port = 8080
serverSocket.bind(('', port))
serverSocket.listen(1)    #for listening socket.

while True:
    #Establish the connection
    print 'Ready to serve...'
    connectionSocket, addr = serverSocket.accept()

    try:
        message =connectionSocket.recv(1024) #receives message from client.
        filename = message.split()[1] #split message for getting file name.
        f = open(filename[1:]) #opens file and reads the contents
        outputdata =f.read()
        print outputdata
        #send one HTTP header line into socket.
        connectionSocket.send('HTTP/1.0 200 OK\r\n\n\n')

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

        f.close()  #close the file.
        connectionSocket.close() #closes the socket for client.

    except IOError:
        #Send response message for file not found
        print '404 Not Found'
        connectionSocket.send('HTTP/1.1 404 Not Found\r\n\n')
        #Close client socket
        connectionSocket.close()
serverSocket.close()
```

1.4 Outputs

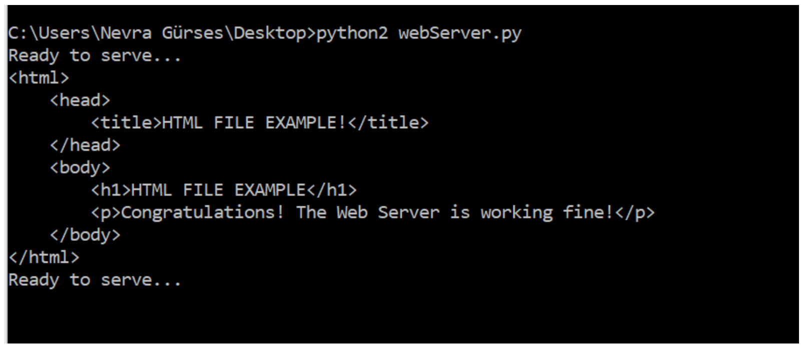
OUTPUT IF HTML FILE EXIST:



HTML FILE EXAMPLE

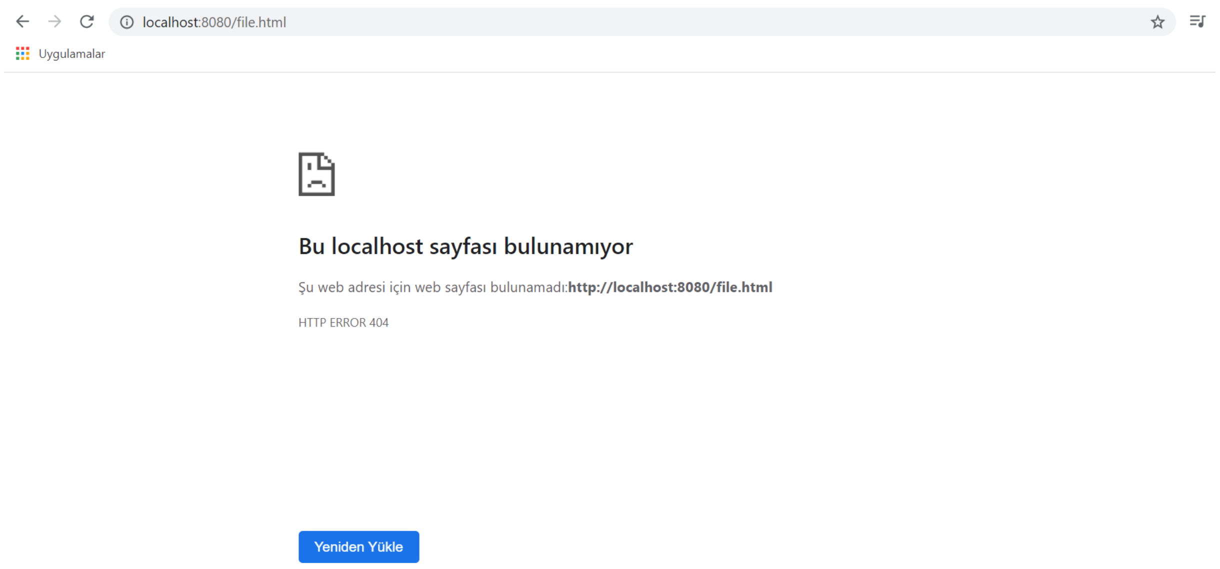
Congratulations! The Web Server is working fine!

CONSOLE OUTPUT IF FILE EXIST:



```
C:\Users\Nevra Gürses\Desktop>python2 webServer.py
Ready to serve...
<html>
    <head>
        <title>HTML FILE EXAMPLE!</title>
    </head>
    <body>
        <h1>HTML FILE EXAMPLE</h1>
        <p>Congratulations! The Web Server is working fine!</p>
    </body>
</html>
Ready to serve...
```

OUTPUT IF HTML FILE DOES NOT EXIST:


localhost:8080/file.html

Uygulamalar

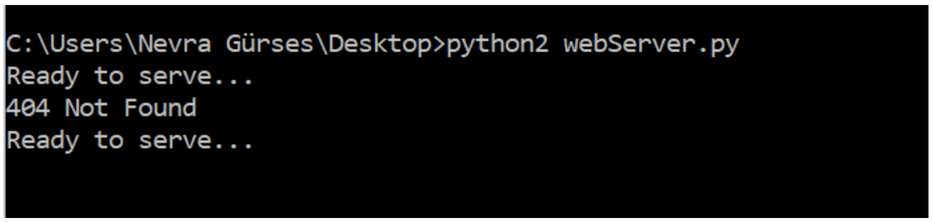
Bu localhost sayfası bulunamıyor

Şu web adresi için web sayfası bulunamadı:**http://localhost:8080/file.html**

HTTP ERROR 404

Yeniden Yükle

CONSOLE OUTPUT IF FILE DOES NOT EXIST:


```
C:\Users\Nevra Gürses\Desktop>python2 webServer.py
Ready to serve...
404 Not Found
Ready to serve...
```

2 PART 2 - UDP Pinger

2.1 Problem Definition

In this part, we will write a client ping program in Python. Our client will send a simple ping message to a server, receive a corresponding pong message back from the server, and determine the delay between when the client sent the ping message and received the pong message. This delay is called the Round Trip Time (RTT). The functionality provided by the client and server is similar to the functionality provided by standard ping program available in modern operating systems. However, standard ping programs use the Internet Control Message Protocol (ICMP).Here we will create a nonstandard UDP-based ping program. Our ping program is to send 10 ping messages to the target server over UDP. For each message, our client is to determine and print the RTT when the corresponding pong message is returned. Because UDP is an unreliable protocol, a packet sent by the client or server may be lost. For this reason, the client cannot wait indefinitely for a reply to a ping message. We should have the client wait up to one second for a reply from the server; if no reply is received, the client should assume that the packet was lost and print a message accordingly.

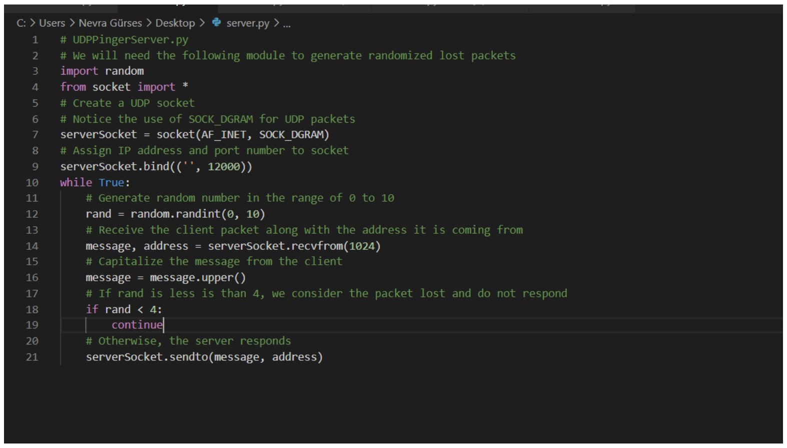
2.2 Operations Made For This Part:

For this part, I examined the server code.Then I wrote client code corresponding for server code and made all necessary operations.

In client code,client send ping message that is a line include sequence number and time when client send ping message for server with using UDP connectionless protocol in loop from 1 to 10.In this loop receive response ping message from server.This response message is uppercase letter of sending ping message.But UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons.For this reason, the client cannot wait indefinitely for a reply to a ping message. So,client wait up to one second for a reply; if no reply is received within one second, my client program assume that the packet was lost during transmission across the network and prints request timed out message on screen. Below,I add server and client code picture,my output pictures in next page.

2.3 Server and Client Code

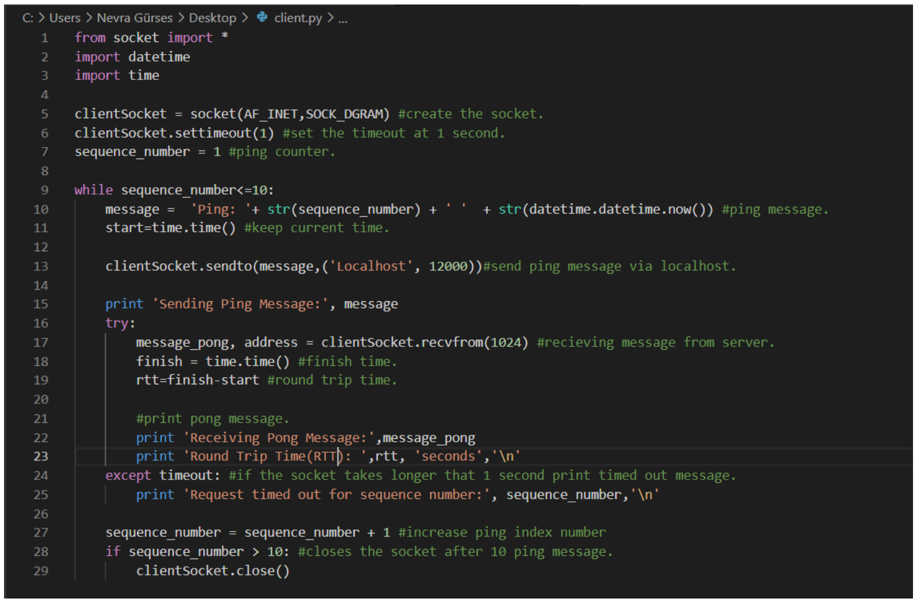
SERVER CODE THAT IS GIVEN FOR US:



C: > Users > Nevra Gürses > Desktop > server.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 THAT I WROTE:

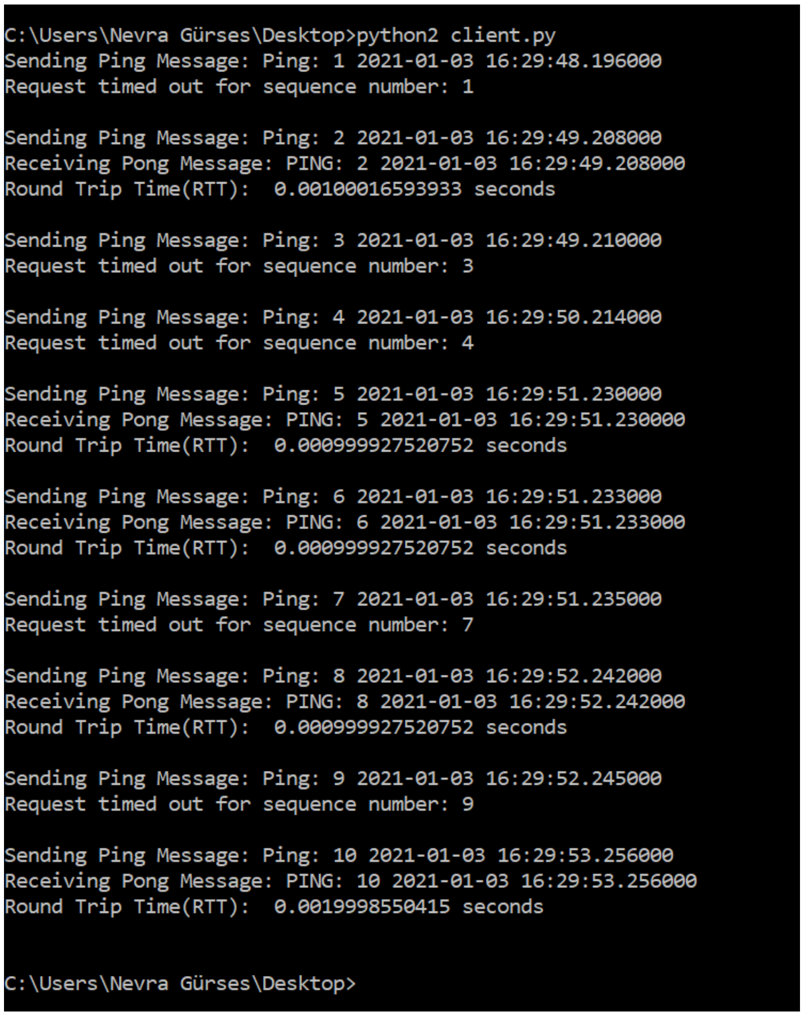


C: > Users > Nevra Gürses > Desktop > client.py > ...

```python
from socket import *
import datetime
import time

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:
    message =  'Ping: '+ str(sequence_number) + ' '  + str(datetime.datetime.now()) #ping message.
    start=time.time() #keep current time.

    clientSocket.sendto(message,('Localhost', 12000))#send ping message via localhost.

    print 'Sending Ping Message:', message
    try:
        message_pong, address = clientSocket.recvfrom(1024) #recieving message from server.
        finish = time.time() #finish time.
        rtt=finish-start #round trip time.

        #print pong message.
        print 'Receiving Pong Message:',message_pong
        print 'Round Trip Time(RTT): ',rtt, 'seconds','\n'
    except timeout: #if the socket takes longer that 1 second print timed out message.
        print 'Request timed out for sequence number:', sequence_number,'\n'

    sequence_number = sequence_number + 1 #increase ping index number
    if sequence_number > 10: #closes the socket after 10 ping message.
        clientSocket.close()
```

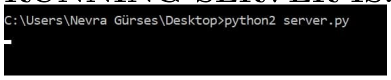
2.4 Output

CLIENT IS:



```
C:\Users\Nevra Gürses\Desktop>python2 client.py
Sending Ping Message: Ping: 1 2021-01-03 16:29:48.196000
Request timed out for sequence number: 1

Sending Ping Message: Ping: 2 2021-01-03 16:29:49.208000
Receiving Pong Message: PING: 2 2021-01-03 16:29:49.208000
Round Trip Time(RTT):  0.00100016593933 seconds

Sending Ping Message: Ping: 3 2021-01-03 16:29:49.210000
Request timed out for sequence number: 3

Sending Ping Message: Ping: 4 2021-01-03 16:29:50.214000
Request timed out for sequence number: 4

Sending Ping Message: Ping: 5 2021-01-03 16:29:51.230000
Receiving Pong Message: PING: 5 2021-01-03 16:29:51.230000
Round Trip Time(RTT):  0.000999927520752 seconds

Sending Ping Message: Ping: 6 2021-01-03 16:29:51.233000
Receiving Pong Message: PING: 6 2021-01-03 16:29:51.233000
Round Trip Time(RTT):  0.000999927520752 seconds

Sending Ping Message: Ping: 7 2021-01-03 16:29:51.235000
Request timed out for sequence number: 7

Sending Ping Message: Ping: 8 2021-01-03 16:29:52.242000
Receiving Pong Message: PING: 8 2021-01-03 16:29:52.242000
Round Trip Time(RTT):  0.000999927520752 seconds

Sending Ping Message: Ping: 9 2021-01-03 16:29:52.245000
Request timed out for sequence number: 9

Sending Ping Message: Ping: 10 2021-01-03 16:29:53.256000
Receiving Pong Message: PING: 10 2021-01-03 16:29:53.256000
Round Trip Time(RTT):  0.0019998550415 seconds


C:\Users\Nevra Gürses\Desktop>
```

RUNNİNG SERVER IS:



```
C:\Users\Nevra Gürses\Desktop>python2 server.py
```

3 PART 3 - Mail Client

3.1 Problem Definition

For this part we will create a simple mail client that sends email to any recipient. Our client will need to establish a TCP connection with a mail server (e.g., a Google mail server), dialogue with the mail server using the SMTP protocol, send an email message to a recipient (e.g., our friend) via the mail server, and finally close the TCP connection with the mail server.

3.2 Operations Made For This Part:

For this part, I create mail client with using SMTP protocol and also using Google Mail Server. For being able to use google mail server,I add a Transport Layer Security (TLS) and Secure Sockets Layer (SSL) for authentication and security reasons, before send MAIL FROM command. While making mail client, I follow below operations:

- Firstly, I choose mail server that is Google Mail Server and I call it with port 587.

- Then, I create client socket and establish a TCP connection with mail server.

- After, I send HELO Command and print server response.

- Then, I open TLS for Google mail server and print server response.

- After that, I create SSL for authentication and security reasons.

- Then, I send MAIL FROM command with using sender mail address and print server response.

-Then, I send RCPT TO command with using receiver mail address and print server response.

- Then, I send DATA command and print server response.

- After that, I send message data.

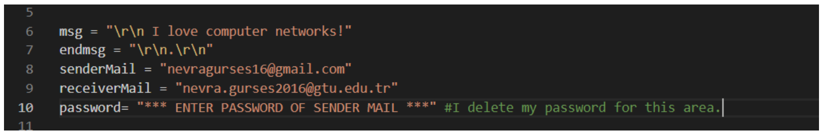
- After that, message ends with single period.

- Then, I send QUIT command and get server response.

- Last, I close socket.

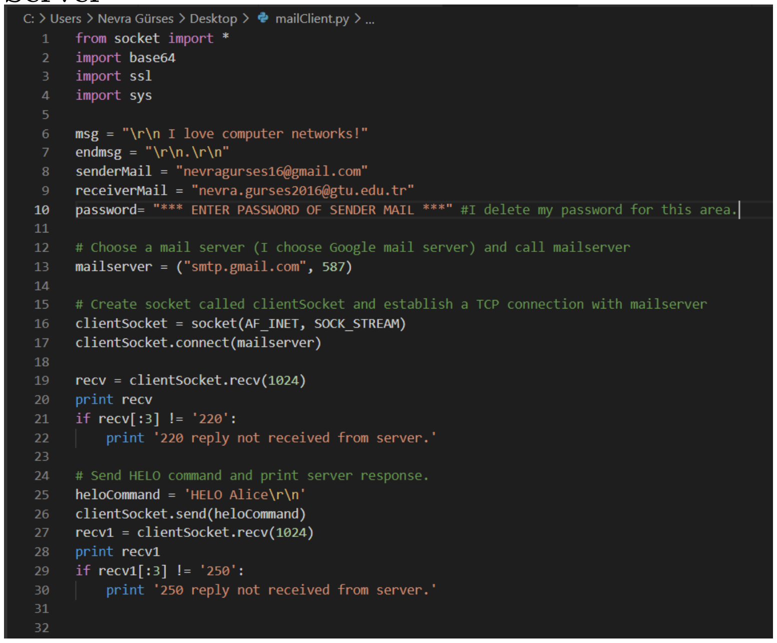
3.3 Mail Server Code

Before I show all code, This part is important.Sender Mail address should be Google e-mail address and password should be password of that address. I delete actual password for security while taking screenshot.



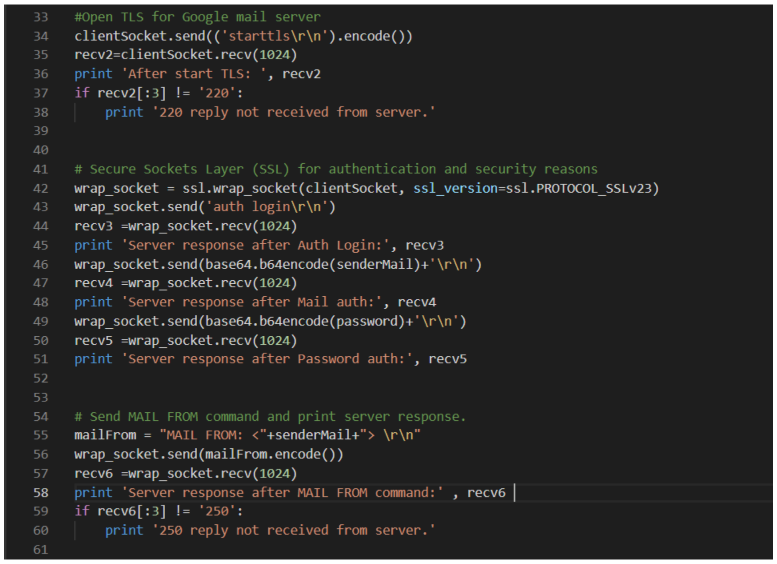
```
msg = "\r\n I love computer networks!"
endmsg = "\r\n.\r\n"
senderMail = "nevragurses16@gmail.com"
receiverMail = "nevra.gurses2016@gtu.edu.tr"
password= "*** ENTER PASSWORD OF SENDER MAIL ***" #I delete my password for this area.
```

Mail Server Code with Using SMTP Protocol and Google Mail Server

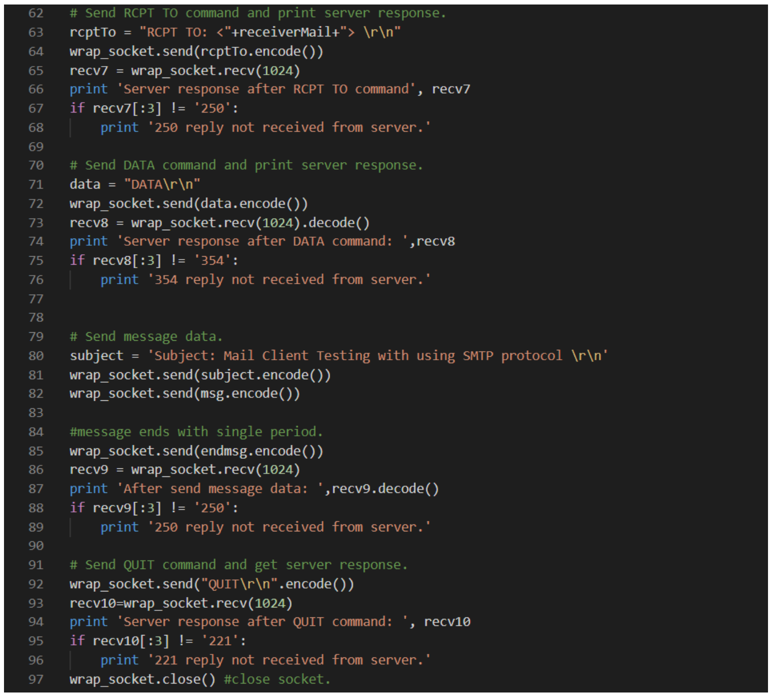


C: > Users > Nevra Gürses > Desktop > mailClient.py > ...

```
from socket import *
import base64
import ssl
import sys

msg = "\r\n I love computer networks!"
endmsg = "\r\n.\r\n"
senderMail = "nevragurses16@gmail.com"
receiverMail = "nevra.gurses2016@gtu.edu.tr"
password= "*** ENTER PASSWORD OF SENDER MAIL ***" #I delete my password for this area.

# Choose a mail server (I choose Google mail server) and call mailserver
mailserver = ("smtp.gmail.com", 587)

# Create socket called clientSocket and establish a TCP connection with mailserver
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect(mailserver)

recv = clientSocket.recv(1024)
print recv
if recv[:3] != '220':
    print '220 reply not received from server.'

# Send HELO command and print server response.
heloCommand = 'HELO Alice\r\n'
clientSocket.send(heloCommand)
recv1 = clientSocket.recv(1024)
print recv1
if recv1[:3] != '250':
    print '250 reply not received from server.'


```

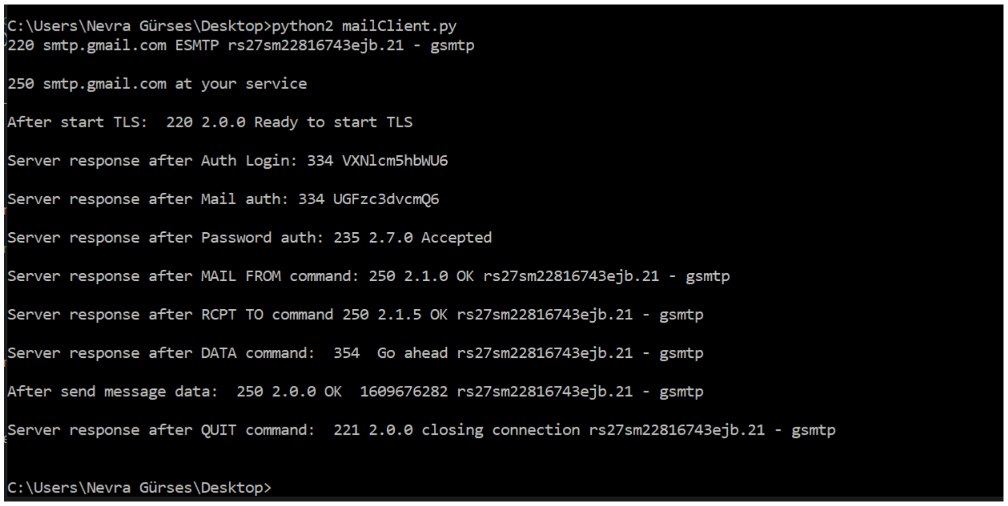


```python
#Open TLS for Google mail server
clientSocket.send(('starttls\r\n').encode())
recv2=clientSocket.recv(1024)
print 'After start TLS: ', recv2
if recv2[:3] != '220':
    print '220 reply not received from server.'


# Secure Sockets Layer (SSL) for authentication and security reasons
wrap_socket = ssl.wrap_socket(clientSocket, ssl_version=ssl.PROTOCOL_SSLv23)
wrap_socket.send('auth login\r\n')
recv3 =wrap_socket.recv(1024)
print 'Server response after Auth Login:', recv3
wrap_socket.send(base64.b64encode(senderMail)+'\r\n')
recv4 =wrap_socket.recv(1024)
print 'Server response after Mail auth:', recv4
wrap_socket.send(base64.b64encode(password)+'\r\n')
recv5 =wrap_socket.recv(1024)
print 'Server response after Password auth:', recv5


# Send MAIL FROM command and print server response.
mailFrom = "MAIL FROM: <"+senderMail+"> \r\n"
wrap_socket.send(mailFrom.encode())
recv6 =wrap_socket.recv(1024)
print 'Server response after MAIL FROM command:' , recv6
if recv6[:3] != '250':
    print '250 reply not received from server.'

```



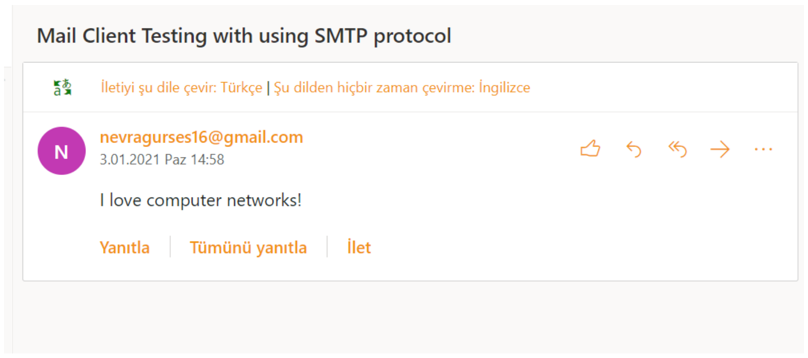
```python
# Send RCPT TO command and print server response.
rcptTo = "RCPT TO: <"+receiverMail+"> \r\n"
wrap_socket.send(rcptTo.encode())
recv7 = wrap_socket.recv(1024)
print 'Server response after RCPT TO command', recv7
if recv7[:3] != '250':
    print '250 reply not received from server.'

# Send DATA command and print server response.
data = "DATA\r\n"
wrap_socket.send(data.encode())
recv8 = wrap_socket.recv(1024).decode()
print 'Server response after DATA command: ',recv8
if recv8[:3] != '354':
    print '354 reply not received from server.'


# Send message data.
subject = 'Subject: Mail Client Testing with using SMTP protocol \r\n'
wrap_socket.send(subject.encode())
wrap_socket.send(msg.encode())

#message ends with single period.
wrap_socket.send(endmsg.encode())
recv9 = wrap_socket.recv(1024)
print 'After send message data: ',recv9.decode()
if recv9[:3] != '250':
    print '250 reply not received from server.'

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

3.4 Console Output



```
C:\Users\Nevra Gürses\Desktop>python2 mailClient.py
220 smtp.gmail.com ESMTP rs27sm22816743ejb.21 - gsmtp

250 smtp.gmail.com at your service

After start TLS:  220 2.0.0 Ready to start TLS

Server response after Auth Login: 334 VXNlcm5hbWU6

Server response after Mail auth: 334 UGFzc3dvcmQ6

Server response after Password auth: 235 2.7.0 Accepted

Server response after MAIL FROM command: 250 2.1.0 OK rs27sm22816743ejb.21 - gsmtp

Server response after RCPT TO command 250 2.1.5 OK rs27sm22816743ejb.21 - gsmtp

Server response after DATA command:  354  Go ahead rs27sm22816743ejb.21 - gsmtp

After send message data:  250 2.0.0 OK  1609676282 rs27sm22816743ejb.21 - gsmtp

Server response after QUIT command:  221 2.0.0 closing connection rs27sm22816743ejb.21 - gsmtp


C:\Users\Nevra Gürses\Desktop>
```

3.5 Mail Output (Mail Sended from Given Sender Mail Address to Given Receiver Mail Address



Mail Client Testing with using SMTP protocol

İletiyi şu dile çevir: Türkçe | Şu dilden hiçbir zaman çevirme: İngilizce

nevragurses16@gmail.com
3.01.2021 Paz 14:58

I love computer networks!

NOTE : I made Optional Exercises-1 for this part.I use Google Mail Server and I use TLS and SSL protocol for being able to use Google Mail Server.