

به نام خدا تکلیف اول کامپیوتری

دانیال علی عظیمی ۹۷۲۴۳۰۸۳

حسنا بشیریان ۹۷۲۴۳۱۰۲

● وب سرور ساده

```
8 class Request():
9     def __init__(self, method, path, version) -> None:
10         self.method = method
11         self.version = version
12         self.path = path
13         self.headers = {}
14         self.body = ""
15
```

در مرحله اول برای راحتی در پیاده سازی دو تا کلاس تعریف کردیم. اولین این کلاس ها request است که دارای مواردی همچون path، method و ... است. همه ی این ها از request دریافت شده و ذخیره می شود.

```
C:\WINDOWS\py.exe
listening
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 197

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Table template</title>
  </head>
  <body>
    <h1>Table template</h1>
    <h2>this is a test file</h2>
  </body>
</html>

listening
_
```

وقتی که یک request داخل browser بزنیم که وقتی که سرور ران می شود یک listening به ما نشان می دهد و منتظر است که یک درخواست به آن زده شود. پس از آن که درخواست زده شد request را می خواند و یک response برای آن می سازد. در تصویر بالا body ساخته شده را چاپ کردیم.

```
class Response():
    def __init__(self, request) -> None:
        self.request = request
        self.headers = {}
        self.body = ""
```

دومین کلاس تعریف شده برای راحتی در استفاده است که شامل سه فیلد request و headers و body است و یک تابع دارد که request را براساس این پارامتر ها می سازد.

```
def getResponse(self, conn) -> str:
    if self.request.path == "/":
        self.request.path = "/index.html"
    self.request.path = '.' + self.request.path

    resp = ""
    try:
        content_type, _ = mimetypes.guess_type(self.request.path)
        file = open(self.request.path)

        self.body = file.read()

        statusLine = "HTTP/1.1" + " " + \
            "200" + " " + "OK" + "\r\n"
        headerlines = f"Content-Type: {content_type}\r\n"
        headerlines += f"Content-Length: {len(self.body.encode('utf-8'))}\r\n"

        resp = statusLine + headerlines + "\r\n" + self.body + "\r\n"

    except Exception as e:
        self.body = "<h1>FILE NOT FOUND</h1>"
        statusLine = self.request.version + " " + "404" + " " + "NOT FOUND" + "\r\n"
        headerlines = "Content-Type: text/html\r\n"
        headerlines += f"Content-Length: {18}\r\n"
        resp = statusLine + headerlines + "\r\n" + self.body

    return resp
```

این تابع getResponse ما است که response ما را بر اساس فرمت پروتکل های http می سازد. اول path را بررسی می کنیم که برای در این سوال یک فایل index.html است سپس header های content type و content length را تعیین می کنیم که برای content type از کتابخانه خود پایتون استفاده کردیم به نام mimetypes. برای به دست آوردن content length، ما string خوانده شده از فایل را تبدیل به یک string بایت می کنیم و طول آن را می خوانیم این به ما تعداد بایت را می دهد. سپس statusLine که جواب

ما است را می سازیم و به **headerline** های مان **append** می کنیم و بعد **body** فایل مان را به انتهای آن **append** می کنیم. اگر **exception** رخ داد ما فرض می کنیم که این **exception** به خاطر نبود فایل است که پیام مناسب مربوط به آن را برمی گردانیم.

```
def readLine(socket) -> str:
    line = b""
    while True:
        data = socket.recv(1) # read one byte from recv
        if(data == b"\n"):
            line += data
            break
        line += data
    return line
```

کار تابع **readLine** این است که یک سوکت دریافت می کند و از سوکت یک بایت یک بایت میخواند تا **n/** ببیند چون آخرین چیزی که در هر خط داریم **n/** است. پس تا انتهای خط می خواند و دیتا را برمی گرداند.

```
# get request first line and return request object
def getRequestLine(socket) -> Request:
    try:
        requestLine = str(readLine(socket)) # read request line
        requestLine = requestLine.split[" "]
        requestLine[2] = requestLine[2].replace("\\r\\n", "")
        return Request(requestLine[0], requestLine[1], requestLine[2])
    except:
        print('err')
```

با استفاده از تابع **readLine** اول خط **requestLine** که خط اولین خط **http request** مان است را می خوانیم و آن را پارس می کنیم که این بر اساس اول متد آن است که در این جا کاری با آن نداریم بعد از آن **path** است که آن را استخراج می کنیم و پس از آن ورژن **http** را هم استخراج می کنیم. اگر **exception** رخ داد دیگر ادامه نمی دهیم.

```
def getRequestHeaders(socket, request) -> Request: # extract header values from request
    headers = {}
    while True:
        headerLine = readLine(socket)
        if headerLine == b'\r\n': # check if headers are finished ?
            request.headers = headers
            return request
        headerLine = str(headerLine)
        headerName = headerLine.split(":")[0]
        headerValues = headerLine.split(':')[1:]
        headers[headerName] = headerValues
```

تابع `getRequestHeaders` با همان تابع `readLine` تا زمانی که سوکت مان `header` هایش تمام نشده باشد یعنی تا وقتی که با `r/n/` رو به رو نشده `header` ها را می خواند و آن ها را ذخیره می کند و به ما برمی گرداند یعنی در خط آخر به آبجکت `request` آن را اضافه می کند.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as serverSocket: # same as try catch
    # define socket for server
    serverSocket.bind((SERVER_NAME, SERVER_PORT))
    while True:
        print('listening') # log
        serverSocket.listen() # listen for tcp connection
        conn, addr = serverSocket.accept() # accept connection

        request = getRequestLine(conn)
        request = getRequestHeaders(conn, request)
        try:
            response = Response(request)
            response_string = response.getResponse(conn)
            print(response_string)
            conn.sendall(bytes(response_string, 'ascii')) # send response
            # conn.sendfile(file)
        except Exception as e:
            conn.sendall(RESPONSE_BAD_REQUEST)
            print(e)
        finally:
            conn.close()
```

در این قسمت ما اول یک سوکت می سازیم از جنس `tcp` به عنوان سرور سوکت آن سوکت را `bind` می کنیم به اسم سرور در این جا `localhost` مان است و `http` که پورت ۸۰ است سپس تا زمانی که مشکلی رخ دهد سرور ما روی پورت ۸۰ `listen` می کند و منتظر یک `connection` است که به آن وصل شود. وقتی `connection accept` شد و وصل شد آدرس و `conn` را می گیریم سپس از آن سوکت `getRequestLine` و `getRequestHeaders` می کنیم و `request` ها را دریافت می کنیم و `response` ها را می سازیم. `Response` ها را هم از `response.getResponse` می سازیم و بعد با استفاده از تابع `sendall` ما `response` را از طریق `tcp` می فرستیم و اگر `exception` رخ داد ما با `bad request` پاسخ می دهیم. در آخر هم `connection` را می بندیم و دوباره `listen` می کنیم و این کار را تا بی نهایت ادامه می دهیم.

● ارسال کننده ایمیل

```
def handShake(conn):  
    #clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    clientSocket.connect(mailServer)  
    recv = clientSocket.recv(1024)  
    return recv
```

در این جا handshake یک connection (سوکت) می گیرد و با استفاده از سوکت به سرور وصل می شود. از mailServer یک اسم و پورت داریم که ما به پورت ۵۸۷ وصل شدیم که می توانستیم به پورت ۲۵ هم می توانستیم وصل شویم ولی چون با google کار می کنیم پورت ۲۵ یک سری مشکلات داشت و با پورت ۵۸۷ مشکلی نداشت.

```
SMTP_SERVER = "smtp.gmail.com"  
SENDER_EMAIL = "dannyhosna@gmail.com"  
RECEIVER_EMAIL = "dazza1379@gmail.com"  
SENDER_PASS = "123456net"
```

تصویر بالا یک سری متغیر است که ما تعریف کردیم که در آینده از آن ها استفاده می کنیم.

```
def sendHelo(conn):  
    sendHelo = f"HELO dan \r\n"  
    conn.send(sendHelo.encode("utf-8"))  
    recv = clientSocket.recv(1024)  
    return recv
```

در این بخش ما hello اولیه را به سرور می فرستیم و منتظر آن هستیم که یک جواب به ما بدهد که بعد بررسی کنیم که جواب ۲۵۰ (accepted) است یا خیر.

```
# send authentication
# sendAuthPlain(clientSocket)
# start ttls
clientSocket.send(b"STARTTLS \r\n")
recv = clientSocket.recv(1024)
recv = recv.decode()
print(recv)
```

برای این که ما بتوانیم با سرور های گوگل ارتباط برقرار کنیم اول می بایست STARTTLS را فعال می کردیم که به بحث امنیتی آن مربوط می شود. ما در ادامه سوکت مان را از جنس ssl می سازیم که امن باشد.

```
# wrap ssl
secureClientSocket = ssl.wrap_socket(
    clientSocket, ssl_version=ssl.PROTOCOL_SSLv23)
secureClientSocket.send(b"AUTH login \r\n")
recv = secureClientSocket.recv(1024)
recv = recv.decode("utf-8")
recv = recv.split(" ")
print(base64.b64decode(recv[1]))
```

تصویر بالا یک wrapper ssl است که از یک پروتکلی استفاده می کند. ما سوکت مان را به این wrapper می دهیم و داخل ssl ما wrap می کنیم و از این به بعد از این سوکت امن استفاده می کنیم.

```
def sendAuthLOGIN(conn):

    # send user name
    username = base64.b64encode(
        SENDER_EMAIL.encode()) + "\r\n".encode()
    conn.send(username)

    recv = conn.recv(1024)
    recv = recv.decode()
    recv = recv.split(" ")
    print(base64.b64decode(recv[1]))

    # send password
    password = base64.b64encode(
        SENDER_PASS.encode()) + "\r\n".encode()
    conn.send(password)
    recv = conn.recv(1024)
    recv = recv.decode("ascii")
    print(recv)
```

در این جا با سوکت جدیدی که ساختیم دستور `authlogin` را به سرور می فرستیم. سرور یک جواب به ما می دهد که از ما درخواست می کند `username` را وارد کنیم سپس از ما پسورد را درخواست می کند.

```
220 smtp.gmail.com ESMTP t4sm4698330wmi.48 - gsmt  
250 smtp.gmail.com at your service  
220 2.0.0 Ready to start TLS
```

این تصور سه تا درخواستی که فرستادیم را نشان می دهد.

```
b'Username: '  
b'Password: '  
235 2.7.0 Accepted
```

پس از فرستادن `auth login` ما جواب های سرور را چاپ کردیم
اول درخواست `username` داده میشود سپس درخواست `password`
و اگر مورد تایید باشند این دو جواب `accepted` برگردانده میشود.

پس از این باید `recipient` را مشخص کنیم که با دستور `RCP` مشخص میشود :

```
def sendRCP(conn):  
    rcp = f"RCPT TO: <{RECEIVER_EMAIL}>\r\n".encode()  
    conn.send(rcp)  
    recv = conn.recv(1024)  
    return recv
```

و از سرور پیام تایید را دریافت میکنیم

```
250 2.1.5 OK t4sm4698330wmi.48 - gsmt
```

سپس باید data خود را یا content email را به سرور ارسال کنیم

```
def sendData(conn):
    conn.send("DATA\r\n".encode())
    recv = conn.recv(1024)
    print(recv.decode())

    # send data
    conn.send("Do you like ketchup?\r\n".encode())
    conn.send("i know i do\r\n".encode())
    conn.send(".\r\n".encode())
    recv = conn.recv(1024)
    print(recv.decode())
```

که ما 3 خط ارسال میکنیم و با یک خط با نقطه بخش دیتا را به پایان میرسانیم و منتظر پیام سرور میمانیم.

```
354 Go ahead t4sm4698330wmi.48 - gsmtip
250 2.0.0 OK 1638531900 t4sm4698330wmi.48 - gsmtip
```

در خط اول تایید آمادگی برای دریافت DATA
در خط بعدی تایید دریافت شدن Data


پس از این در آخر ارتباط خود را با سرور به پایان میرسانیم :

```
def sendQuit(conn):
    conn.send(b"QUIT\r\n")
    recv = conn.recv(1024)
    print(recv.decode())
```

```
221 2.0.0 closing connection t4sm4698330wmi.48 - gsmtip
```


و پیام متناظر را از سرور دریافت میکنیم.

در پایان میتوانیم inbox خود را چک کنیم و چنین پیامی را ببینیم :

(no subject)  Inbox x



dannyhosna@gmail.com

to ▼

Do you like ketchup?

i know i do

