Socket Programming Project 2024

Raneem Alarian

Old Dominion University

CYSE-250: Cybersecurity Programming and Networking

Hind AlDabagh

April 18, 2024

Problem Statement:

Developing reliable and secure pharmacy kiosk self-services is imperative for enhancing patient experience and efficiency. Waiting in long pick-up lines at the pharmacy only to discover that prescriptions aren't ready is not only ineffective but frustrating for patients. As a licensed pharmacy technician, I frequently hear patients complain about this issue. My colleagues and I have often envisioned the convenience of a self-service kiosk within the pharmacy. Here, patients can securely input and verify their information, ensuring compliance with HIPAA regulations, to promptly verify the status of their prescriptions before joining the pick-up line. This innovation not only streamlines pharmacy operations but also saves valuable time for patients. To address this need, I've developed a software program enabling patients to interact with the kiosk and check their prescription status before proceeding to the pick-up line.

Scenarios:

1. The patient inputs their information correctly. The patient is found in the database with no duplicate profiles. If they have one or more medications ready for pick-up, the server outputs the first three letters of the name of the medications that are ready. If the patient has no medications ready at the time, the server outputs there are no medications found. Connection ends.
2. The patient inputs their information correctly. If there is more than one patient with the same name and date of birth found in the database, the server will ask the patient to verify their home address. If the home address is not found or does not match, the connection ends.
3. The patient inputs their information correctly. The patient is not found in the database. The server outputs that the patient is not found. The connection ends.
4. The patient inputs their first and last name correctly, but the date of birth does not match what is found in the database. The patient has to re-verify their date of birth. If the patient inputs the correct date of birth upon re-verification and has medications ready for pick-up, the server will output the first three letters of the name of the medications ready and the connection ends. If the patient does not enter the correct date of birth upon re-verification, the connection ends.

Hardware and Software:

The hardware used to implement this program is a MacBook Air laptop. Below is an overview of the hardware.

Model Name:	MacBook Air
Model Identifier:	MacBookAir7,2
Processor Name:	Dual-Core Intel Core i5
Processor Speed:	1.8 GHz
Number of Processors:	1
Total Number of Cores:	2
L2 Cache (per Core):	256 KB
L3 Cache:	3 MB
Hyper-Threading Technology:	Enabled
Memory:	8 GB
System Firmware Version:	486.0.0.0.0
OS Loader Version:	540.120.3~37
SMC Version (system):	2.27f2
Serial Number (system):	FVFTN1RMJ1WK
Hardware UUID:	23FF8E3F-3457-5B81-BB53-15F638AA97C2
Provisioning UDID:	23FF8E3F-3457-5B81-BB53-15F638AA97C2

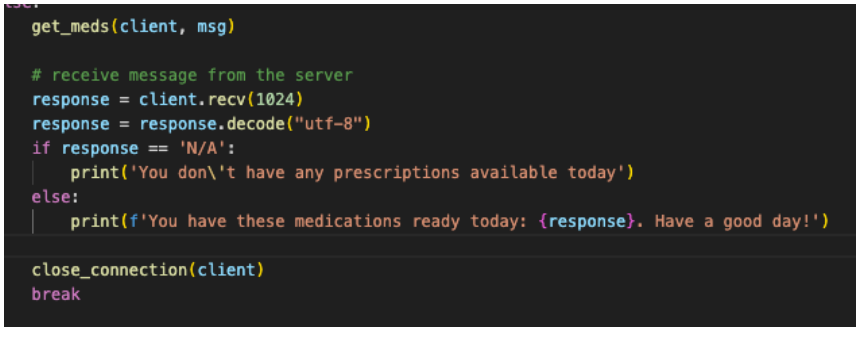
Below is an overview of the system software.

System Version:	macOS 12.7.2 (21G1974)	
	Kernel Version:	Darwin 21.6.0
	Boot Volume:	Macintosh HD
	Boot Mode:	Normal
	Computer Name:	Raneem's MacBook Air
	User Name:	Raneem Alarian (raneemalarian)
	Secure Virtual Memory:	Enabled
	System Integrity Protection:	Enabled
	Time since boot:	99 days 20:50

The software used to build this program is Visual Studio Code, a source code editor that runs on a desktop. It can be used for Windows, macOS, and Linux. VS Code has an ecosystem of extensions or other languages and runtimes such as Java, Python, PHP, C++, C#, Go., and .NET.

Results:

Scenario 1- Patient is found in the database with no duplicate records:



```
get_meds(client, msg)

# receive message from the server
response = client.recv(1024)
response = response.decode("utf-8")
if response == 'N/A':
    print('You don\'t have any prescriptions available today')
else:
    print(f'You have these medications ready today: {response}. Have a good day!')

close_connection(client)
break
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

MacBook-Air:~ raneemalarian$ /usr/local/bin/python3 /Users/raneema
First and Last Name Please: Amy Lee
Can I please have your DOB? 4/3/1997
4/3/1997
You have these medications ready today: PHE****. Have a good day!
Connection to server closed
MacBook-Air:~ raneemalarian$
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PO

MacBook-Air:~ raneemalarian$ /usr/local/bin/python3
First and Last Name Please: Harold Johnson
Can I please have your DOB? 6/9/1945
6/9/1945
You don't have any prescriptions available today
Connection to server closed
MacBook-Air:~ raneemalarian$
```

Scenario 2- Duplicate name and date of birth found in the database:

```python
elif response.count('/') > 2:
    wanted_info = 'Address'
    msg = input('Can I please have your address?')
    msg = f'{wanted_info} {msg}'
    client.send(msg.encode("utf-8")[:1024])
    # receive message from the server
    response = client.recv(1024)
    response = response.decode("utf-8")

    if response == 'N/A':
        print('Sorry you are not on record here. Have a good day.')
        close_connection(client)
        break
```

If address is verified correctly:

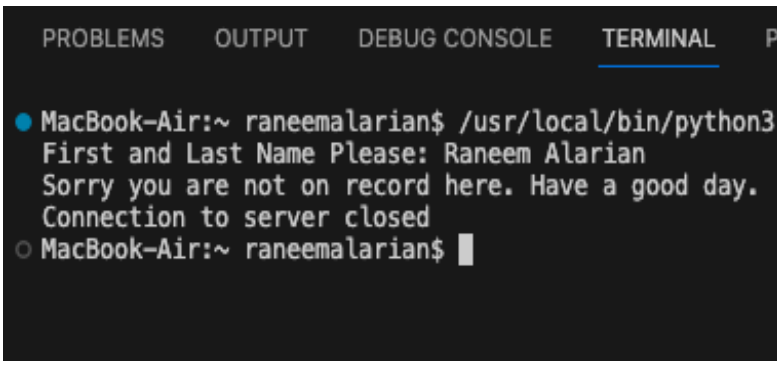
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PO

MacBook-Air:~ raneemalarian$ /usr/local/bin/python3
First and Last Name Please: John Doe
Can I please have your DOB? 5/17/1980
5/17/1980
5/17/1980
Can I please have your address?5567 Apple Farm Blvd.
You have these medications ready today: ART***
LOW***. Have a good day!
Connection to server closed
MacBook-Air:~ raneemalarian$
```

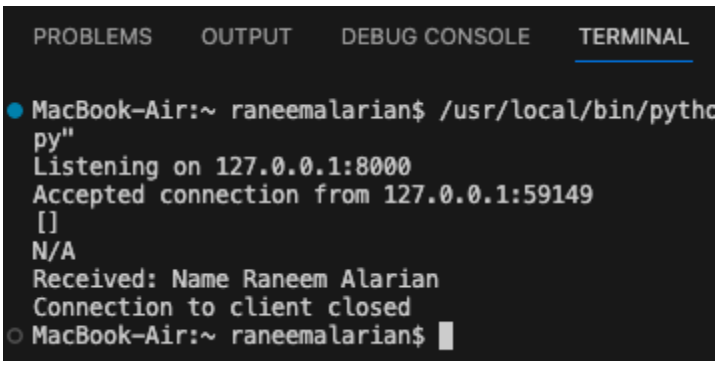
If address does not match what is on record:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  P

MacBook-Air:~ raneemalarian$ /usr/local/bin/python3
First and Last Name Please: John Doe
Can I please have your DOB? 5/17/1980
Can I please have your address?677 Ave Rd
Sorry you are not on record here. Have a good day.
Connection to server closed
MacBook-Air:~ raneemalarian$
```

Scenario 3- Patient is not found in the database:

```
while True:
    # input message and send it to the server
    wanted_info = 'Name'
    msg = input("First and Last Name Please: ")
    msg = f'{wanted_info} {msg}'
    client.send(msg.encode("utf-8")[:1024])

    # receive message from the server
    response = client.recv(1024)
    response = response.decode("utf-8")

    # if server sent us "closed" in the payload, we break out of the loop and close our socket
    if response.lower() == "closed":
        break

    if response == 'N/A':
        print('Sorry you are not on record here. Have a good day.')
        close_connection(client)
        break
```

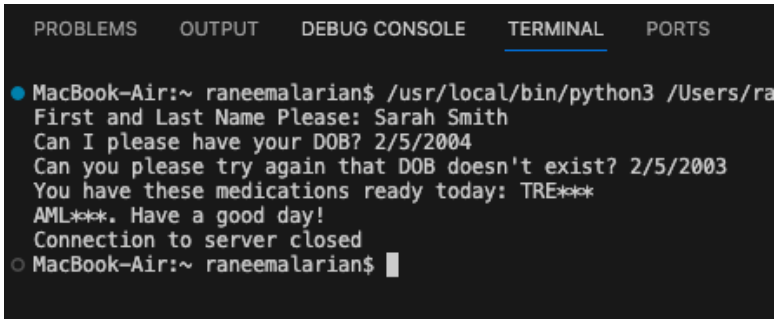




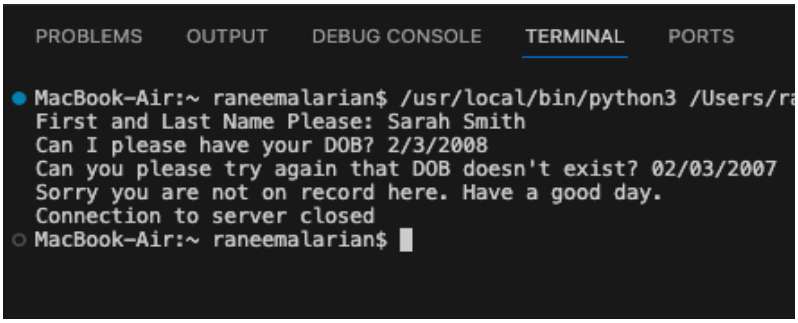
Scenario 4- Patient inputs the wrong date of birth:

```
else:
    wanted_info = 'DOB'
    msg = input("Can I please have your DOB? ")
    msg = f'{wanted_info} {msg}'
    client.send(msg.encode("utf-8")[:1024])
    response = client.recv(1024)
    response = response.decode("utf-8")

    if response == 'N/A':
        msg = input("Can you please try again that DOB doesn't exist? ")
        msg = f'{wanted_info} {msg}'
        client.send(msg.encode("utf-8")[:1024])
        response = client.recv(1024)
        response = response.decode("utf-8")

        if response == 'N/A':
            print('Sorry you are not on record here. Have a good day.')
            close_connection(client)
            break
        else:
            get_meds(client, msg)
```

If DOB is re-verified correctly:



If DOB is not re-verified correctly:



Discussions:

The client-server program has been successfully implemented, with the server actively listening for incoming connections and passing all scenario tests. When the client accurately verifies all patient information, the server promptly responds with the prescription pick-up status before terminating the connection. If the provided first and last name is not found in the database, the connection ends without further inquiry.

In cases where the client provides partially correct information, such as the correct name but an incorrect date of birth, the server initiates a re-verification process for the date of birth. If the date of birth is verified correctly, the prescription pick-up status is provided, and the connection is terminated. However, if the date of birth is not re-verified correctly, no prescription information is disclosed, and the connection is terminated. Furthermore, if the server identifies multiple patients with the same name and date of birth in the database, the client is required to verify their home address.

Overall, the scenario tests validate the program's functionality, confirming its ability to effectively process patient information, verify prescriptions, and terminate connections appropriately, thereby meeting the desired performance criteria for this project.

Appendix A

```
import socket
import json

dict = {
  "person1" : {
     "Name" : "Sarah Smith",
     "DOB" : '2/5/2003',
     "Address" : '321 Python Way',
     "Medication" : ["TRE***", "AML***"]
  },
  "person2" : {
     "Name" : "John Doe",
     "DOB" : '5/17/1980',
     "Address" : '3564 Beverly Hill Rd.',
     "Medication" : []
  },
  "person3" : {
     "Name" : "Kimberly Shaw",
     "DOB" : '9/10/1970',
     "Address" : '1245 Washington Tree Terrace',
     "Medication" : ["DOX****"]
  },
  "person4" : {
     "Name" : "Amy Lee",
     "DOB" : '4/3/1997',
     "Address" : '943 Beach Front Ave.',
     "Medication" : ["PHE****"]
```

```
  },
  "person5" : {
      "Name" : "John Doe",
      "DOB" : '5/17/1980',
      "Address" : '5567 Apple Farm Blvd.',
      "Medication" : ['ART***', 'LOW***']
  },
  "person6" : {
      "Name" : "Amy Lee",
      "DOB" : '12/13/2000',
      "Address" : '6324 Sunnydale Drive',
      "Medication" : ["COL****", 'NIM****']
  },
  "person7" : {
      "Name" : "Harold Johnson",
      "DOB" : '6/9/1945',
      "Address" : '7845 Senior Citizen Ave.',
      "Medication" : []
  },
}
def run_server():
  # create a socket object
  server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

  server_ip = "127.0.0.1"
  port = 8000

  # bind the socket to a specific address and port
```

```
server.bind((server_ip, port))
# listen for incoming connections
server.listen(0)
print(f"Listening on {server_ip}:{port}")

# accept incoming connections
client_socket, client_address = server.accept()
print(f"Accepted connection from {client_address[0]}:{client_address[1]}")

# receive data from the client
while True:
    request = client_socket.recv(1024)
    request = request.decode("utf-8") # convert bytes to string

    # if we receive "close" from the client, then we break
    # out of the loop and close the conneciton
    if request.lower() == "close":
        # send response to the client which acknowledges that the
        # connection should be closed and break out of the loop
        client_socket.send("closed".encode("utf-8"))
        break

    if request[:4] == 'Name':
        name_list = [names['Name'] for names in dict.values() if names['Name'] ==
request[5:]]
        print(name_list)

        if len(name_list) == 0:
```

```
            response = 'N/A'
            print(response)
            client_socket.send(response.encode('utf-8'))
        else:
            # response = "accepted".encode("utf-8") # convert string to bytes
            response = '\n'.join(name_list)
            # convert and send accept response to the client
            client_socket.send(response.encode('utf-8'))

    elif request[:3] == 'DOB':
        dob_list = [dob['DOB'] for dob in dict.values() if dob['DOB'] == request[4:]]
        print(dob_list)

        if len(dob_list) == 0:
            response = 'N/A'
            print(response)
            client_socket.send(response.encode('utf-8'))
        else:
            # response = "accepted".encode("utf-8") # convert string to bytes
            response = '\n'.join(dob_list)
            # convert and send accept response to the client
            client_socket.send(response.encode('utf-8'))

    elif request[:7] == 'Address':
        address_list = [address['Address'] for address in dict.values() if
address['Address'] == request[8:]]
        print(address_list)
```

```
        if len(address_list) == 0:
            response = 'N/A'
            print(response)
            client_socket.send(response.encode('utf-8'))
        else:
            # response = "accepted".encode("utf-8") # convert string to bytes
            response = '\n'.join(address_list)
            # convert and send accept response to the client
            client_socket.send(response.encode('utf-8'))

    else:
        medication_available = []
        if request[4:11] == 'Address':
            dictionary_key = [key for key, value in dict.items() if value.get('Address') ==
request[12:]]
            print(dictionary_key)
            print(dict[dictionary_key[0]]['Medication'])
            medication_available.append(dict[dictionary_key[0]]['Medication'])

            if len(medication_available[0]) == 0:
                response = 'N/A'
                print(response)
                client_socket.send(response.encode('utf-8'))
            else:
                # response = "accepted".encode("utf-8") # convert string to bytes
                response = '\n'.join(medication_available[0])
                # convert and send accept response to the client
                client_socket.send(response.encode('utf-8'))
```

```
        else:
            medication_available = []
            dictionary_key = [key for key, value in dict.items() if value.get('DOB') ==
request[8:]]
            print(dictionary_key)
            print(dict[dictionary_key[0]]['Medication'])
            medication_available.append(dict[dictionary_key[0]]['Medication'])

            if len(medication_available[0]) == 0:
                response = 'N/A'
                print(response)
                client_socket.send(response.encode('utf-8'))
            else:
                # response = "accepted".encode("utf-8") # convert string to bytes
                response = '\n'.join(medication_available[0])
                # convert and send accept response to the client
                client_socket.send(response.encode('utf-8'))

    print(f"Received: {request}")

  # close connection socket with the client
  client_socket.close()
  print("Connection to client closed")
  # close server socket
  server.close()

run_server()
```

Appendix B

```
import socket

def close_connection(client):
    msg = 'close'
    client.send(msg.encode("utf-8")[:1024])

def get_meds(client, msg):
    meds = 'Meds'
    msg = f'{meds}{msg}'
    client.send(msg.encode("utf-8")[:1024])

def run_client():
    # create a socket object
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    server_ip = "127.0.0.1"  # replace with the server's IP address
    server_port = 8000  # replace with the server's port number
    # establish connection with server
    client.connect((server_ip, server_port))

    while True:
        # input message and send it to the server
        wanted_info = 'Name'
        msg = input("First and Last Name Please: ")
        msg = f'{wanted_info} {msg}'
        client.send(msg.encode("utf-8")[:1024])
```

```
    # receive message from the server
    response = client.recv(1024)
    response = response.decode("utf-8")

    # if server sent us "closed" in the payload, we break out of the loop and close our
socket
    if response.lower() == "closed":
        break

    if response == 'N/A':
        print('Sorry you are not on record here. Have a good day.')
        close_connection(client)
        break

    else:
        wanted_info = 'DOB'
        msg = input("Can I please have your DOB? ")
        msg = f'{wanted_info} {msg}'
        client.send(msg.encode("utf-8")[:1024])
        response = client.recv(1024)
        response = response.decode("utf-8")

        if response == 'N/A':
            msg = input("Can you please try again that DOB doesn't exist? ")
            msg = f'{wanted_info} {msg}'
            client.send(msg.encode("utf-8")[:1024])
            response = client.recv(1024)
            response = response.decode("utf-8")
```

```
            if response == 'N/A':
                print('Sorry you are not on record here. Have a good day.')
                close_connection(client)
                break
            else:
                get_meds(client, msg)

                # receive message from the server
                response = client.recv(1024)
                response = response.decode("utf-8")
                if response == 'N/A':
                    print('You don\'t have any prescriptions available today')
                else:
                    print(f'You have these medications ready today: {response}. Have a good
day!')

                close_connection(client)
                break

        elif response.count('/') > 2:
            wanted_info = 'Address'
            msg = input('Can I please have your address?')
            msg = f'{wanted_info} {msg}'
            client.send(msg.encode("utf-8")[:1024])
            # receive message from the server
            response = client.recv(1024)
            response = response.decode("utf-8")
```

```
            if response == 'N/A':
                print('Sorry you are not on record here. Have a good day.')
                close_connection(client)
                break

            else:
                get_meds(client, msg)

                # receive message from the server
                response = client.recv(1024)
                response = response.decode("utf-8")

                if response == 'N/A':
                    print('You don\'t have any prescriptions available today')
                else:
                    print(f'You have these medications ready today: {response}. Have a good
day!')

                close_connection(client)
                break
        else:
            get_meds(client, msg)

            # receive message from the server
            response = client.recv(1024)
            response = response.decode("utf-8")
            if response == 'N/A':
```

```
                print('You don\'t have any prescriptions available today')
            else:
                print(f'You have these medications ready today: {response}. Have a good
day!')

            close_connection(client)
            break

    # close client socket (connection to the server)
    client.close()
    print("Connection to server closed")

run_client()
```