ISTANBUL TECHNICAL UNIVERSITY
Faculty of Computer Science and Informatics

IMPLEMENTING A CHAT PROGRAM

INTERNSHIP PROGRAM REPORT

MUSTAFA CAN ÇALIŞKAN
150200097

SUMMER / 2023

Istanbul Technical University

Faculty of Computer Science and Informatics

INTERNSHIP REPORT

Academic Year:	2022/2023
Internship Term:	☒ Summer ☐ Spring ☐ Fall

Student Information

Name Surname:	MUSTAFA CAN ÇALIŞKAN
Student ID:	150200097
Department:	Computer Engineering
Program:	100% English
E-Mail:	caliskanmu20@itu.edu.tr
Mobile Phone:	+90 (533) 192 8068
Pursuing a Double Major?	☐ Yes (Faculty/Department of DM: _____) ☒ No
In the Graduation Term?	☐ Yes ☒ No
Taking a class at Summer School?	☐ Yes (Number of Courses:_____) ☒ No

Institution Information

Company Name:	Aselsan Konya Silah Sistemleri A.Ş.
Department:	Software Design
Web Address:	https://www.aselsankonya.com/en
PostalAddress:	Aşağıpınarbaşı Mah. Fatih Sultan Mehmet Cad. No:2 Selçuklu/Konya

Authorized Person Information

Department:	SOFTWARE DESIGN
Title:	LEAD OF SOFTWARE DESIGN
Name Surname:	SELÇUK ÇALLI
Corporate E-Mail:	ik@aselsankonya.com.tr
Corporate Phone:	0332 224 06 00

Internship Work Information

Internship Location:	☒ Turkey ☐ Abroad
Internship Start Date:	01.08.2023
Internship End Date:	28.08.2023
Number of Days Worked:	20
During your internship, did you have insurance?	☒ Yes, I was insured by İTÜ. ☐ Yes, I was insured by institution. ☐ No, I did my internship abroad. ☒ No.

Table of Contents

1 INFORMATION ABOUT THE INSTITUTION

Aselsan Konya Silah Sistemleri Inc., which was established on December 17, 2020, with a 51% ownership by Aselsan and 49% ownership by Konya Savunma Sanayi Inc. [1], does not limit its activities solely to the production of firearms and weapon systems. Aselsan Konya also engages in research, design, development, and engineering activities. It carries out the entire process from the production of firearms, weapon systems, and defense industry products to testing, assembly, and integration. Additionally, it manages processes such as sales, marketing, import, and export of these products while providing training, maintenance, and after sales services. Within this framework, it also participates in commercial and industrial activities, with approximately 300 employees in various roles, including design, quality control, procurement, and production at Aselsan Konya.

2 INTRODUCTION

The first week was devoted to occupational safety training, basic information security training, campus introduction, and orientation. At the beginning of the second week, due to my prior knowledge of Python, PySide, and PyQt, my mentor asked me to implement a simple messaging software with message and file transfer capabilities and a graphical user interface. In the subsequent weeks, I implemented the program with feedback from my mentor. At the beginning, because I was familiar with the Linux environment, using certain tools on Windows initially posed some difficulties for me. However, I eventually got used to it.

For privacy reasons, I will not be able to provide outputs and codes from certain sections of the project; instead, I will provide sample codes and outputs.

3 DESCRIPTION AND ANALYSIS OF THE INTERNSHIP PROJECT

3.1 Back-end Implementation

3.1.1 General Program Structure

To make the program's functions more comprehensible, facilitate testing and maintenance, and enhance its modularity, I divided it into four modules, adhering to the MVC (Model - View - Controller) [2] design pattern and including the main module. The MVC design pattern encompasses the model, which regulates the program's underlying processes, the view, which includes the designs related to the program's user interface, and the controller classes that manage background operations based on the user's interactions with the user interface, ensuring coordination between the two classes.

For testing purposes, I added the Python code representing the server-side as a separate module. Additionally, for testing, I included only two clients in the main module, but I structured it in a way that allows for the possibility of increasing the number of clients and implementing routing between them as needed in the future. Connection diagram and UML (Unified Modeling Language) diagram of the program can be seen in Figure 1 and Figure 2 respectively.

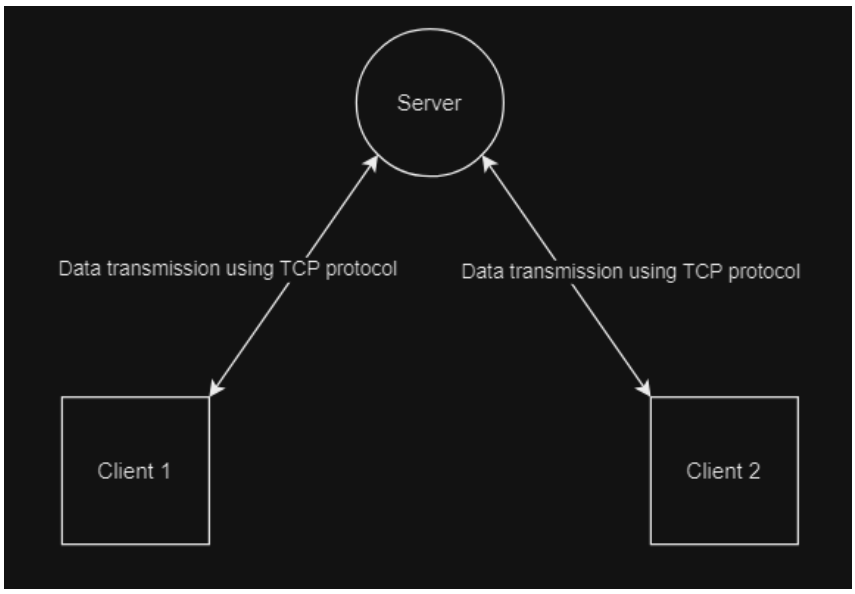

Figure 1: Communication Diagram

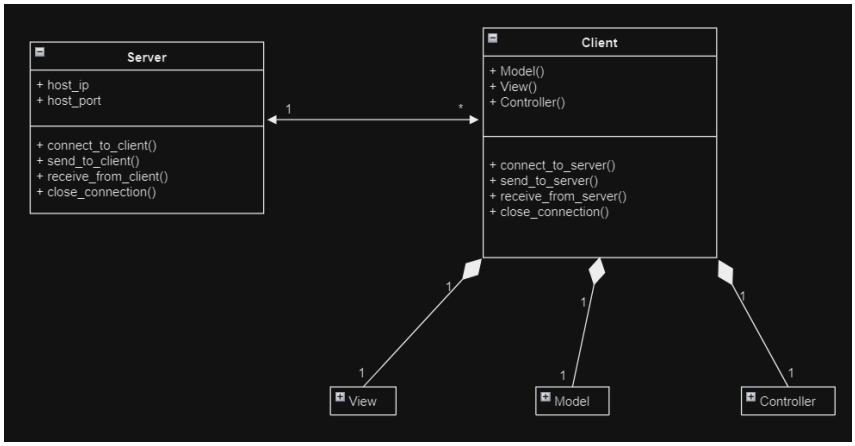

Figure 2: UML Diagram

3.1.2 Preliminary Research

Firstly, I researched various programming approaches for the implementation of the program and gathered information about socket programming.

Through my research, I have discovered that socket programming serves as a foundational technique for establishing communication across computer networks. Sockets essentially act as interfaces, enabling the exchange of data between devices. Python provides some libraries allowing the creation of these communication endpoints and supporting various protocols.

As a result, I have decided that I need to determine the protocol I will use and identify the libraries in Python associated with this protocol.

3.1.3 Choosing Communication Protocol

I conducted various research to determine the protocol for data transmission between two users (clients). As a result of my research, I acquired information about TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) protocols. I observed that there are several fundamental differences [3] between them:

- TCP is a connection-oriented protocol. It establishes a connection before communication and utilizes a series of control mechanisms such as handshaking processes to ensure the reliability of the transmitted data. UDP, on the other hand, is a connectionless protocol. It sends and receives data directly, without any connection establishment or authentication process.
- In TCP, data is transmitted after it has been acknowledged by the receiver, and in cases of loss or faulty transmission, it is retransmitted. Therefore, it ensures reliable and error-free communication. UDP, on the other hand, provides fast data transmission but carries the possibility of data loss or non-sequential delivery.
- The additional control mechanisms inherent in the TCP protocol can occasionally introduce delays. Therefore, the UDP protocol, lacking these additional control mechanisms, is particularly utilized in applications that require low latency, such as games and audio/video transmission.
- The additional control mechanisms inherent in the TCP protocol can sometimes lead to delays. Therefore, the UDP protocol, which lacks these additional control mechanisms, is commonly employed in applications, especially in those requiring low latency, such as games, audio, and video transmission.

Due to these differences (Also can be seen in Figure 3), I observed that the TCP protocol is more suitable for message and file transfer, and I decided to use it in my project.

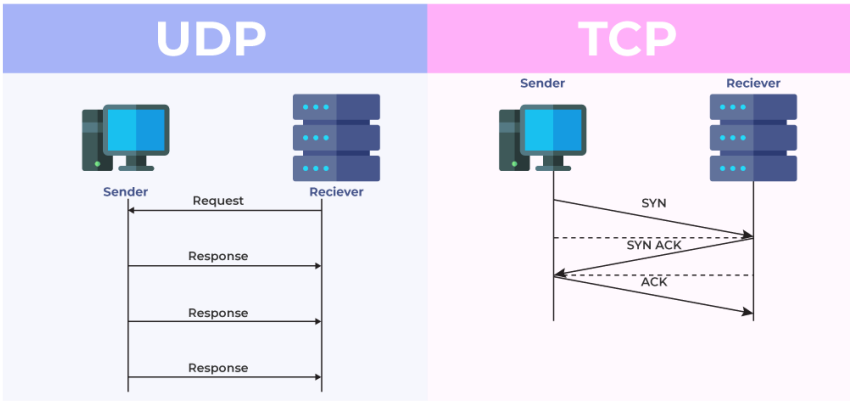

Figure 3: TCP vs UDP Communication [3]

3.1.4 Python Socket Library

I decided to utilize the socket module in Python, which supports the TCP protocol and is employed for low-level network communication. Below are some code snippets and functions that serve as examples of the fundamental methods I used for the client and server sides (connection establishment, connection waiting, data transmission and reception, and connection termination).

The codes for the client side are as follows:

```
# Clientside
import socket
server_ip = "localhost" # Server ip and port
server_port = 12345
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_ip, server_port))
message = "Hello, ITU BBF!" # Send a message to server
client_socket.send(message.encode())
reply = client_socket.recv(1024) # Get a reply (maximum 1024 byte)
print("Reply:", reply.decode())
client_socket.close() # Close connection
```

The codes for the server side are as follows:

```
# Serverside
server_ip = "localhost" # Server ip and port
server_port = 12345
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.bind((HOST, PORT))
    server_socket.listen()
    print(f"Server listening on {HOST}:{PORT}")
    conn, addr = server_socket.accept() # Accept client connection
data = conn.recv(1024) # Getting message
print(f"Received from client: {data.decode()}")
reply = "Hello, client!" # Send a reply
conn.send(reply.encode())
conn.close() # Close connection
```

3.1.5 Communication Problems and Multi-threading

Initially, I thought that a simple infinite loop would suffice for the server side to continuously listen to two (or more) clients simultaneously:

```
while True:
    message_1 = client_1_connection.recv(1024) # Maximum 1024 byte
    message_2 = client_2_connection.recv(1024)
```

Later on, I realized that the ".recv()" method, which is part of the socket module and used to receive data from the remote end, puts the program into a waiting state until any amount of data is received from the remote side. For example, I noticed that the second client's messages were not being listened to by the server until the first client sent any message. Additionally, I observed that the program would enter a waiting state again based on the size of the transferred file and the transmission speed, causing the other client not to be listened to.

To address this issue, I decided to create separate threads to manage the message waiting methods on both the server and client sides using the threading module, which handles multi-threading processes in Python. An example of the multi-threading solution I developed for the server side is provided below. The concept of multi-threading programming is also illustrated in Figure 4.

```
import threading
def handle_client(client_socket):
    while True:
        data = client_socket.recv(1024)
        if not data: # In case it encounters an error
            break
    client_socket.close()
# Creating threads
thread_1 = threading.Thread(target=handle_client, args(client_socket_1, ))
thread_2 = threading.Thread(target=handle_client, args(client_socket_2, ))
thread_1.start() # Starting threads
thread_2.start() # Starting threads
# all clients can send and receive messages at the same time.
```

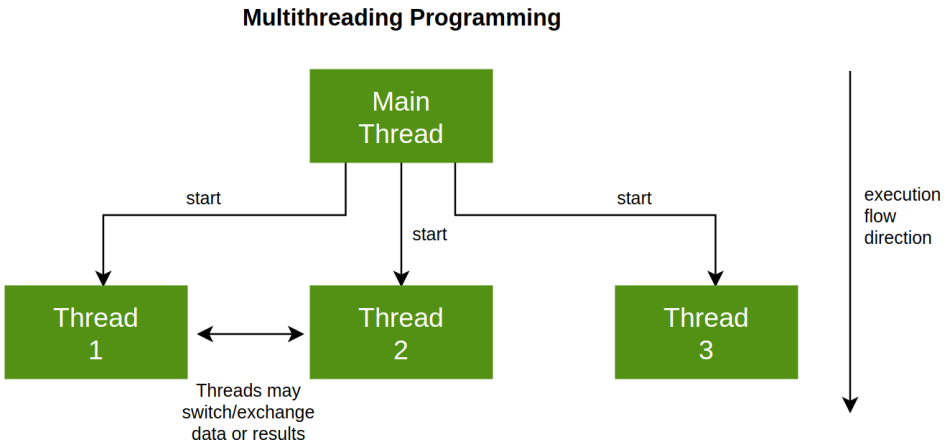

Figure 4: Multi-threading Illustration [8]

3.1.6 File Transfer

The fundamental challenge in file transfer revolved around how to differentiate between message and file transfers when they share the same connection. Alternatively, if they were to be carried out over different connections, it focused on mitigating the performance loss that these separate connections might introduce.

When I attempted to perform file and message transfers over different connections, I noticed that each client would use two threads, and on the server side, there would be twice as many threads as the number of clients. Considering that my computer's processor had four cores, I understood that this situation could lead to "overhead"[4] issues. I tried various ways to manage threads but couldn't overcome the performance loss.

When I attempted to perform file transfer over the same connection by reading the file in binary and adding a signature to indicate whether it was a message or a file before transmission, I encountered various encoding errors. Subsequently, I realized that there was actually no need for such a signature, as certain file types (png, jpg, pdf, etc.) inherently have specific signatures at the beginning of their contents.

```
file_signatures = {
    b'\x89PNG\r\n\x1a\n': 'png',
    b'\xff\xd8\xff\xe0': 'jpg',
    b'%PDF': 'pdf',
}

for x in file_signatures.keys():
    if received_message[:len(x)] == x:
        print(f"This is a/an {file_signatures[x]} file.")
```

Since not all file types contained the mentioned signatures (e.g., Excel and Word files), I utilized the Magic [5] library, which accurately predicts data types from the content of the files.

```
import magic
    message_mime = magic.from_buffer(received_message, mime=True)
    # Ex: application/pdf
    print(f"This is a/an {message_mime.split("/")[1]} file.")
```

3.2 Front-end Implementation

3.2.1 GUI Design

For the front-end design, I chose the PySide library, which is a Python binding library for the Qt [6] C++ library. It allows the use of Qt's powerful graphical features with Python. The fundamental approach of the library involves creating new classes that inherit predefined classes from the library, such as QMainWindow and QWidgets, and using attributes and methods from the parent library with new arguments. I cannot provide the actual interface due to privacy reasons, but below are code snippets representing the components of a sample interface. Figure 5 shows the appearance of the sample chat interface.

Firstly, below, the necessary libraries were imported, and a ChatProgram class inheriting the QMainWindow class was created.

```
from PySide6.QtWidgets import *
class ChatProgram(QMainWindow):
    def __init__(self):
        super().__init__() # To initialize parent QMainWindow object
        self.init_ui()
```

Afterwards, the properties of the window (size, position, placement of buttons inside it, etc.) were adjusted using methods provided by the library.

```
def init_ui(self):
        self.setWindowTitle('Chat Program') # Window title
        self.setGeometry(100, 100, 400, 400) # To adjust window size
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        layout = QVBoxLayout() # To create a layout
        self.chat_history = QTextEdit()
        layout.addWidget(self.chat_history)
        self.user_input = QLineEdit()
        layout.addWidget(self.user_input)
        send_button = QPushButton('Send')
```

```
        send_button.clicked.connect(self.send_message)
        layout.addWidget(send_button)
        central_widget.setLayout(layout)
```

Finally, the send_message method was written to add the sent messages to the created chat box.

```
def send_message(self):
    user_message = self.user_input.text()
    if user_message:
        self.chat_history.append('You: ' + user_message)
        self.user_input.clear() # To clear taken message from text input
```

3.2.2 Multi-threading Problems with PySide

In order to display the received response from the server as a message on the screen, I needed to create a new QLabel object from the PySide library on the client side and add it to the QFrame object in the chat screen. However, I encountered an error indicating that the ".recv()" method, responsible for listening to incoming messages, needed to run in the same thread as the program's interface in order to deliver the message [7].

In order to address such situations, I opted to utilize the QThread object available in the PySide library. The ".recv()" method on the client side operates within a QThread object, and it sends the received message to an external method as an argument using PySide's signal-slot mechanism, which is included within an event. An example code snippet related to this process is provided below.

```
class ReceiveThread(QThread):
    message_received = Signal(str)
    def run(self):
        while True:
            data = client_socket.recv(1024)
            if not data: # Error handling (disconnections etc.)
                break
            message = data.decode("utf-8") # To decode byte to string data type
             # Received string will be sent by emit method
            self.message_received.emit(message)
thread_obj = ReceiveThread()
thread_obj.start() # start method belongs to parent QThread
thread_obj.message_received.connect(add_message_to_frame) # Add message to frame
def add_message_to_frame(message):
    # method continues using message
```

4 CONCLUSIONS

As a result, during my internship, I refreshed my knowledge in PySide and GUI development while enhancing my practical experience. Additionally, I gained in-depth knowledge of multi-threading and acquired valuable experience in communication protocols such as TCP/UDP. And ultimately, by refreshing my knowledge and utilizing the information from my research, I have implemented a chat program with a graphical user interface using the TCP protocol and socket programming. This journey not only improved my technical skills

but also enhanced my problem-solving abilities and eagerness to learn. With the guidance and support of my mentors, I was able to make the most of this experience.

In addition to my technical development, I gained significant personal growth during this period. Meeting engineers working on important research projects has been a great contribution to my professional network. Getting to know the culture of Aselsan Konya and gaining insights into the defense industry provided me with a valuable opportunity to observe R&D activities up close.

5 REFERENCES

[1] Aselsan Konya, https://www.aselsankonya.com/en/who-we-are

[2] Design Patterns - MVC Pattern, https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

[3] Differences between TCP and UDP, https://www.geeksforgeeks.org/differences-between-tcp-and-udp/

[4] Why Too Many Threads Hurts Performance, and What to do About It, https://www.codeguru.com/cplusplus/why-too-many-threads-hurts-performance-and-what-to-do-about-it/

[5] Python-magic, https://pypi.org/project/python-magic/

[6] Qt, https://www.qt.io/

[7] Qt Documentation, Threads and QObjects, https://doc.qt.io/archives/qt-4.8/threads-qobject.html

[8] The Difference Between Asynchronous and Multi-Threading, https://www.baeldung.com/cs/async-vs-multi-threading

6 APPENDIX

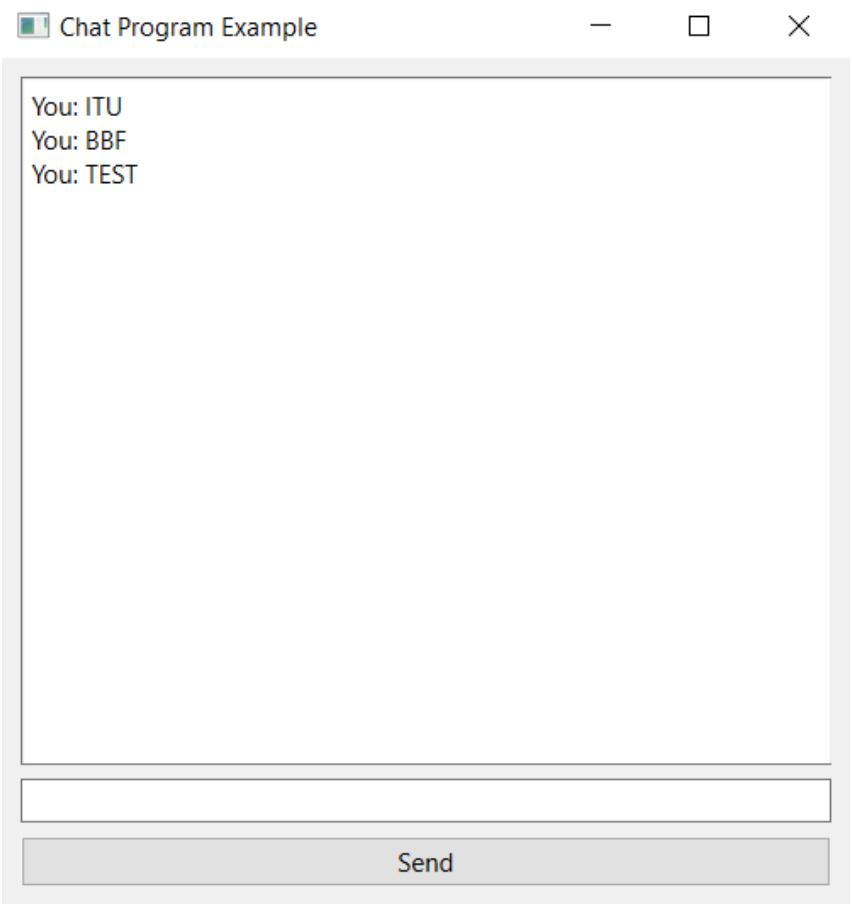

Figure 5: Sample Interface of Chatbot Program