PES UNIVERSITY

EC Campus, Bengaluru

Department of Computer Science & Engineering

COMPUTER NETWORKS - UE21CS252B

5th Semester – E Section

ASSIGNMENT – 1

Title of the Project

VOICE CHAT APP

Submitted to:

Dr. Geetha D

Associate Professor

SRN:PES2UG21CS295

Submitted By:

Name : Monish D

SRN: PES2UG21CS303

Name: Madhava bhave

SRN: PES2UG21CS265

Name: MV Sanjay

Table of Contents

1. Abstract and Scope of the Project

Abstract:
The voice chat app project using socket programming aims to develop a real-time communication platform where users can communicate with each other using voice over the internet. The application will use socket programming, which is a low-level networking technology that allows for real-time communication over a network. The project will involve the development of a server-side application that can manage the connections between multiple clients, and a client-side application that can connect to the server and facilitate voice communication with other clients.

Scope:
The scope of the voice chat app project using socket programming includes the following features:

- Users will be able to see a list of other users who are online and available for communication.
- Users will be able to initiate voice chat sessions with other users.
- The server-side application will be able to handle multiple clients simultaneously.
- The application will be designed with security in mind to prevent unauthorized access and ensure the privacy of users' conversations.
- The application will have a user-friendly interface that is easy to navigate and use.
- The application will be scalable and able to handle a large number of users simultaneously.

Overall, the voice chat app project using socket programming will provide a robust platform for real-time voice communication over the internet, enabling users to connect and communicate with each other from anywhere in the world.

2. Project Description

(Detailed explanation about project implementation and details of functions defined)

The client sends audio data to the server, and the server plays back the audio on the default output device. The project can be broken down into the following steps:

3. Create a client and a server using the socket module.
4. Initialize a PyAudio object to capture and play back audio.
5. Set up the audio format and parameters, such as the sampling rate, number of channels, and chunk size.
6. In the client code, continuously read audio data from the input device and send it to the server.
7. In the server code, continuously receive audio data from the client and play it back on the output device.
8. Close the connection and socket when the communication is complete.

Functions Defined:
The code snippets define several functions that are used to set up the client and server connections, initialize the PyAudio object, and read or write audio data.

9. socket.socket() - This function creates a new socket object using the specified address family (AF_INET for IPv4), and socket type (SOCK_STREAM for TCP). It takes no arguments.
10. socket.bind() - This function binds a socket to a specific host and port. It takes a tuple containing the host and port as arguments.
11. socket.listen() - This function puts the socket in listening mode, waiting for incoming connections. It takes the maximum number of queued connections as an argument.
12. socket.accept() - This function accepts an incoming connection request and returns a new socket object for communicating with the client, and the client's address as a tuple.
13. socket.connect() - This function initiates a connection to a remote host and port. It takes a tuple containing the host and port as arguments.
14. socket.sendall() - This function sends data to the remote host. It takes the data as an argument.
15. socket.recv() - This function receives data from the remote host. It takes the maximum number of bytes to receive as an argument.
16. pyaudio.PyAudio() - This function initializes a new PyAudio object.
17. PyAudio.open() - This function opens an audio stream using the specified audio format and parameters, and returns a stream object. It takes several arguments such as format, channels, rate, input/output, etc.
18. Stream.read() - This function reads audio data from the input stream, and returns a bytes object.
19. Stream.write() - This function writes audio data to the output stream.

It takes the data as an argument.

20. Socket.close() - This function closes the socket and releases the resources used by it.

21. Stream.stop_stream() - This function stops the audio stream.

22. Stream.close() - This function closes the audio stream and releases the resources used by it.

Software Requirements

(Description about Programming Languages, API's, Methods, etc.,)

Programming Languages:
The code snippets are written in Python, which is a high-level programming language that is widely used for developing a wide range of applications, including web applications, scientific computing, data analysis, artificial intelligence, and more.
APIs:
The code uses two APIs, namely the socket and PyAudio APIs.

Socket API: The socket API provides a low-level interface for network communication. It allows developers to create socket objects and use them to establish connections, send and receive data, and perform other network-related tasks.

PyAudio API: The PyAudio API provides an interface for capturing and playing back audio data. It allows developers to open audio streams, set up the audio format and parameters, and read or write audio data.

Source Code

Server:

```
import socket
import pyaudio

CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 5000))
server_socket.listen(1)
print('Server listening on port 5000...')

connection, address = server_socket.accept()
print(f'Client connected from {address[0]}:{address[1]}')

audio = pyaudio.PyAudio()

stream = audio.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=RATE,
                    output=True,
                    frames_per_buffer=CHUNK)

while True:
    data = connection.recv(CHUNK)
    stream.write(data)

connection.close()
server_socket.close()
```

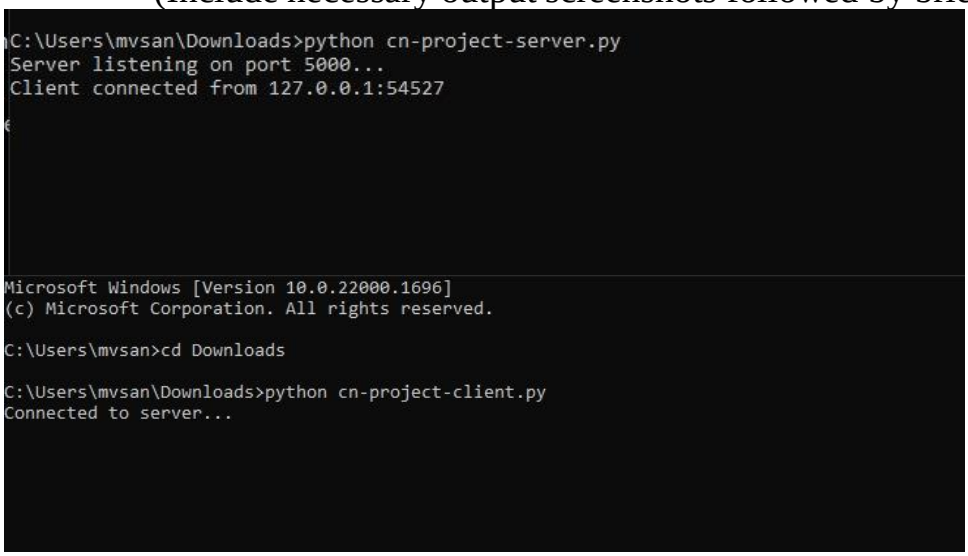
Client:

```
import socket
import pyaudio

CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 5000))
print('Connected to server...')
```

```
audio = pyaudio.PyAudio()

stream = audio.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=RATE,
                    input=True,
                    frames_per_buffer=CHUNK)

while True:
    data = stream.read(CHUNK)
    client_socket.sendall(data)

client_socket.close()
```

Sample Output

(Include necessary output screenshots followed by brief description)



Conclusion

In conclusion, the project implemented a simple client-server voice chat application using Python programming language, socket and PyAudio APIs. The client and server codes were provided in separate code snippets, and they were designed to establish a connection between a client and a server, and to allow them to communicate by sending and receiving audio data.

The project made use of various methods from the socket and PyAudio APIs to create and manage sockets, to establish connections, and to capture and play audio data. The project also used some constants to specify the audio format, number of channels, and sampling rate.

Overall, the project demonstrated the basic principles of network programming and audio processing using Python and the socket and PyAudio APIs.

23. References