**AMITY UNIVERSITY KOLKATA**

****

**MINOR PROJECT**

**On**

**TOPIC: IMAGE PROCESSING USING SOCKET PROGRAMMING**

**Submitted to Amity School of Engineering and Technology**

**Submitted By: Guided By:**

**Name: Ms. Aratrika Ray Mr. Dhritiman Mukkherjee**

**Department: B.TECH (CSE)**

**Semester: 7**

**Section: B**

**Enrolment No.: A91005219084**

**2022**

**DECLARATION**

We hereby declare that the project entitled "Image Processing using Socket Programming", which is being submitted as Minor Project of 7th semester in Computer Science & Engineering to AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY, Kolkata is an authentic record of our genuine work done under the guidance of Prof. Dhritiman Mukherjee, Dept. of Computer Science & Engineering, AUK.

Date: 12/12/2022 Place: Kolkata

**ACKNOWLEDGEMENT**

I would like to express my special thanks of gratitude to my faculty guide Dr. Moumita Basu as well as my Mentor Mr. Dhritiman Mukherjee who gave me the golden opportunity to do this wonderful project on the topic ,”Image Processing Using Socket Programming” . It has been a genuine pleasure for me to work on such a significant topic.

Thank You

Name: Aratrika Ray

Enrolment Number: A91005219084

Date: 12.12.2022

Place: AUK

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
|  | **TITLE** |  |
|  | **ABSTRACT** | **iii** |
|  | **LIST OF FIGURES** | **vi** |
| **1.0** | **INTRODUCTION** | **1** |
|  | 1.1 Socket Programming | 1 |
|  | 1.2 Client Server Architecture | 1 |
|  | 1.3 Image Processing | 1 |
| **2.0** | **METHODOLOGY** | **3** |
|  | 2.1 Overview | 3 |
|  | 2.2 Modules and Library | 3 |
| **3.0** | **EXPERIMENTATION** | **4** |
|  | 3.1 Procedures | 4 |
|  | 3.2 Code Snippet | 4 |
| **4.0** | **RESULT AND DISCUSSION** | **5** |
| **5.0** | **CONCLUSION** | **6** |
| **6.0** | **FUTURE SCOPE** | **7** |
|  | **REFERENCES** | **8** |

**ABSTRACT**

The minor project on the topic "Image Processing Using Socket Programming" deals with constructing a client-server architecture in which we send an image from the client to the server for processing. The server returns the image to the client after performing the necessary processing.

We developed our project in Microsoft Visual Studio IDE using Python programming language, including important modules like Socket, Threading, io, and PIL.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No**. |
| Figure 1.1 | Client Server Architecture | 1 |
| Figure 1.2 | Threading | 2 |
| Figure 1.3 | Socket Flowchart | 3 |
| Figure 2.1 | Network( Original Image) |  |
| Figure 2.2 | Processed Server Image |  |
| Figure 2.3 | Processed Client Image |  |

**INTRODUCTION**

* 1. **SOCKET PROGRAMMING**

Socket programming is a way of connecting two nodes on a network to communicate with each other.  The other socket (node) reaches out to the first socket to establish a connection while the first socket listens on a certain port at an IP. While the client reaches out to the server, the server creates the listener socket.

They really form the basis of online browsing. There is a server and a client, to put it simply.

Importing the socket library and creating a basic socket are the first steps in socket programming.

**1.2 CLIENT SERVER ARCHITECTURE**

A system that hosts, transmits, and maintains the majority of the resources and services that the client requests is known as a client-server architecture. This approach, also known as the networking computing model or client server network, involves the transmission of all requests and services across a network. A network application defined as client-server architecture, sometimes known as a client-server model, divides workloads and duties between clients and servers that are housed on the same system or are connected via a computer network.

Client-server architecture often consists of laptops, PCs, or other devices belonging to many users that are linked to a central server through the Internet or another network. When the server receives a request for data from the client, it complies and transmits the requested data packets back to the user.

**1.3 IMAGE PROCESSING**

Image processing is a technique for applying various procedures to an image in order to improve it or extract some relevant information from it. It is a kind of signal processing where the input is an image and the output may either be another image or features or characteristics related to that image. Image processing is one of the technologies that is currently expanding quickly. It is a major subject of research in both the engineering and computer science fields.

Pillow is built above PIL (Python Image Library). One of the key Python image processing modules is PIL. The PIL module, however, does not support Python 3 and has not been supported since 2011.

**METHODOLOGY**

**2.1 OVERVIEW**

**Step 1:** We create a client.py file and import socket library. We then specify the server address which serves as a tuple argument to establish a connection between the client and the server.

**Step 2:** We now open the network.jpg which is to be sent by this client to the server for image processing. The functions required to be performed on this image being sent are crop, rotate (by 90 degrees) and greyscale.

**Step 3:** We read this opened image and then send it to the server.

**Step 4:** We create a server.py file and import socket, threading, io and PIL libraries and corresponding PIL functions.

**Step 5:** Here we bind with the ADDR tuple containing the address and then we define two methods which handle\_client() and start() respectively.

**Step 6:** In the start() we listen to new connections using listen() and then we accept a new client connection using accept().We create and start a new thread and set handle\_client as the target and this basically passes the new connection to the handle\_client() method. We also print the number of active connections within this method.

**Step 7:** In the handle\_client() we print the address of the active connection. We receive and accept the data packets sent by the client. While we are receiving the data packets we write it in a file. After the writing and accepting has completed, we send a marker indicating the completion.

**Step 8:** We now perform the image processing operations such as crop, rotate and grayscale using crop(), rotate() and grayscale() functions present in the Image and ImageOps modules of the PIL library.

**Step 9:** After the completion of the processing we save the processed image and then send it back to the client in the form of data packets and once the sending has been completed we send an image\_completed marker indicating the completion.

**Step 10:** We now close all the existing connections.

**2.2 MODULES AND LIBRARY**

1. import socket:

Importing the socket library is the first step to using a socket object in your programme. It comes pre-installed; there is no need to install it using a package management. A two-way communication channel between two network-running programmes has a socket as its endpoint.

1. import thread:

Import thread is used to create a new thread by importing the threading module, as shown.

A threading module is made up of a Thread class, which is instantiated to create a Python thread.

1. import PIL:

PIL stands for Python Imaging Library, and it's the original library that enabled Python to deal with images.

The Python Imaging Library adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

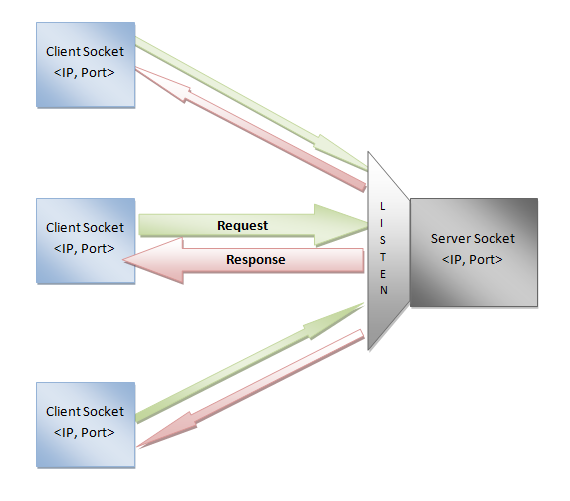
1. from PIL import Image

The Python Imaging Library, or PIL, gives the Python interpreter the ability to modify images. A class with the same name is provided by the Image module and is used to represent a PIL image. The module also contains a variety of manufacturing functions, including tools for generating new images and loading images from files.

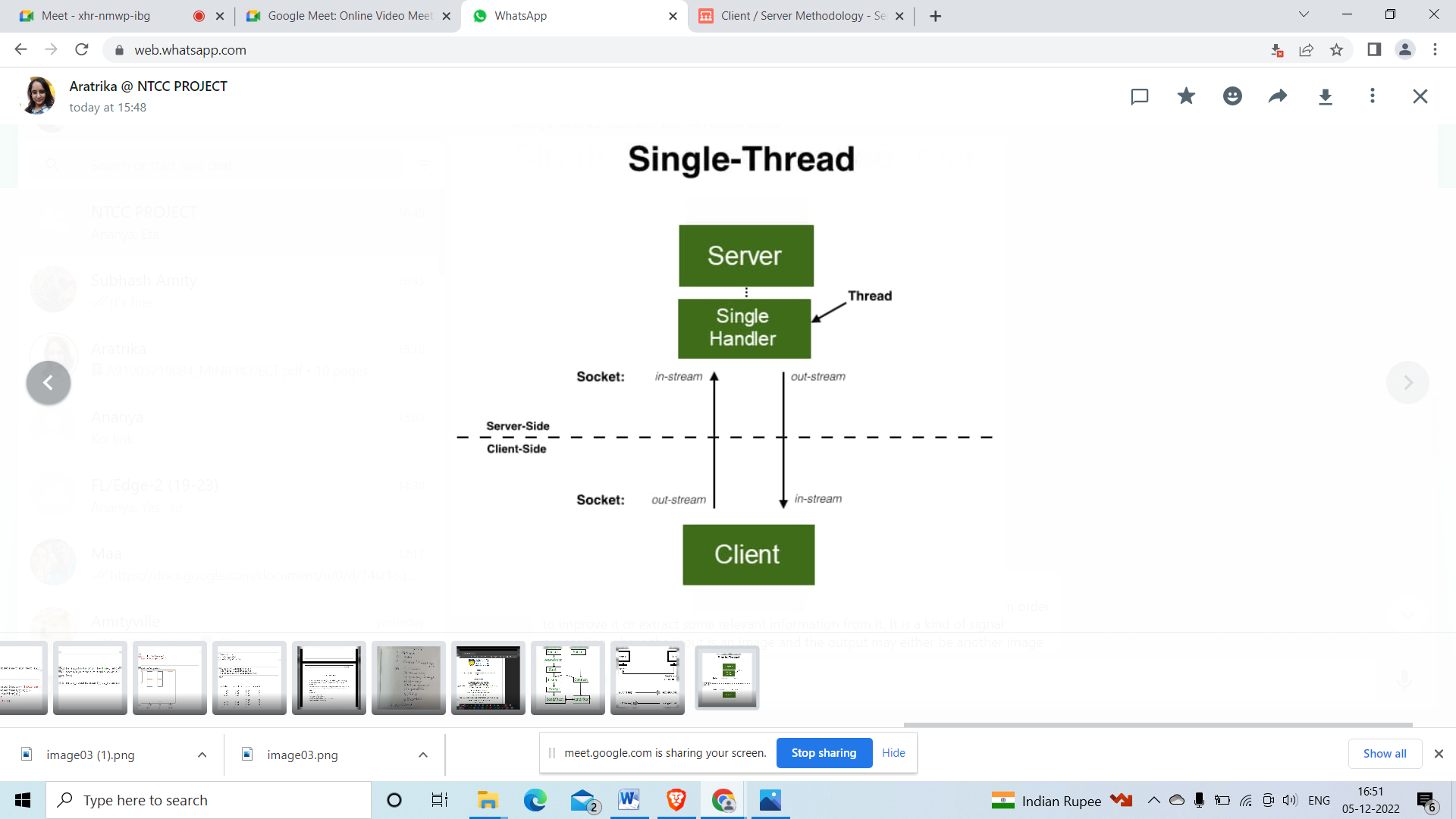
1. ImageOps:

The Python Imaging Library, or PIL, gives the Python interpreter the ability to modify images. There are several "ready-made" image processing operations in the ImageOps module. Most operators in this module, which is relatively experimental, only function on L and RGBimages.

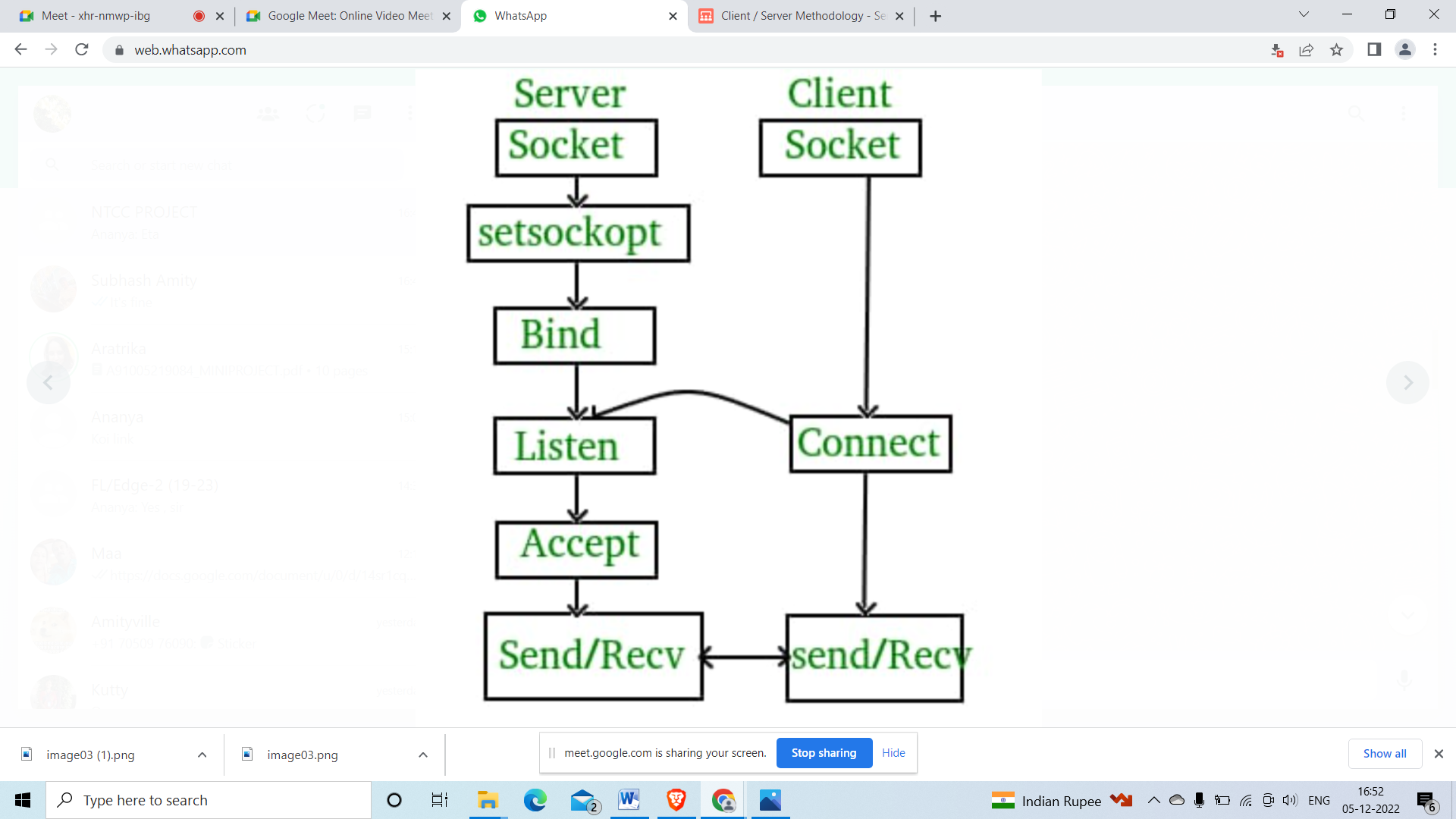
|  |
| --- |
| **1.1 CLIENT SERVER ARCHITECTURE** |

****

|  |
| --- |
| **1.2 THREADING** |



|  |
| --- |
| **1.3 SOCKET FLOWCHART** |



**EXPERIMENTATION**

**3.1 PROCEDURE**

We will be using Microsoft Visual Studio to execute our project using Python programming language.

We create a server.py file for the server socket and a client.py file for the client socket and save the apple.jpg inside the same folder in the IDE.

**Server**

**Steps involved in server socket:**

**Step 1:** In the server.py file, we import socket , threading and PIL libraries and then define **HEADER,PORT** as well as **FORMAT.**

Code snippet:

**HEADER = 64**

**PORT = 5050**

**FORMAT = 'utf-8'**

**Step 2:** We store the system IP address in **SERVER** variable using **socket.gethostbyname(socket.gethostname())** where **gethostbyname()** is a function in the socket module that returns the IP address by accepting the hostname argument.

**Step 3:** We create an **ADDR** constant where we pass the **SERVER** and **PORT** as arguments in the form of tuple.

**Step 4:** We create a server using socket Mojo and socket class. Within the socket class we pass two arguments socket.**AF\_1NET** for IP and **socket.SOCK\_STREAM** for TCP. Thus, we create an IP and TCP socket naming it server.

**Step 5:** We now use **bind()** function present inside the server module to bind the server socket with a specific address present in the **ADDR** tuple.

**Step 6:** We create a **start()** method for server starting and listening. We use the **listen()** function the server module to listen for new connections. For a new connection, using **server.accept()** we store the socket object in client\_socket and the client address (port number and ip address) in client\_address.

**Step 7:** We now pass the new connection to **handle\_client()** method using thread. Thus we create a new thread using **thread = threading.Thread(target=handle\_client, args=(client\_socket,client\_address))** and here we set the target as **handle\_client()** method with **client\_socket** and **client\_address** passed as arguments in the **Thread()** function. After creating the thread, we start it using **thread.start()**. We now print the number of active connections in the terminal.

**Step 8:** Outside of the **handle\_client()** and **start()** method we call the **start()** method before which we print a server is starting message in the terminal.

**Step 9:** Inside the **handle\_client()** method we print client\_address for the new connection. We save the file received from the client using file=**io.BytesIO().** We now accept data packets using **image\_chunk**= **client\_socket.recv**(2048) where **recv()** function accepts the data from the client and 2048 is the buffer size.

**Step 10:** Under a while function using **image\_chunk** as the condition we write the **image\_chunk** in our file using **file.write(image\_chunk).** We then use **image\_chunk= client\_socket.recv(**2048) to receive new data from the client. Now here if **image\_chunk** is equal to the bytes of image completed using **image\_chunk==b"%IMAGE\_COMPLETED%"** then we break. It means that the entire data is received and we no longer require to wait.

**Step 11:** We now open the received image and perform image processing techniques such as crop, rotation and grayscale using PIL library. After this we save the image.

Code snippet:

**image=Image.open(file)**

**width, height = image.size**

**im = image.rotate(90, PIL.Image.Resampling.NEAREST, expand = 1)**

**im1=ImageOps.grayscale(im)**

**im2=im1.crop((0,0,width//2,height//2))**

**im2.save("server\_image.jpeg",format='JPEG')**

**Step 12:** We open the saved image and then read it using **image\_data=files.read(2048)** where **read()** function includes 2048 as the buffer size meaning we read 2048 bytes of data at a time. Under a while function using image\_data as the condition we use **client\_socket.send(image\_data)** where we send our processed data back to the client and then we read the **image\_data** in our files using **files.read(image\_data)**.

**Step 13:** We close the file and send the marker **b'%IMAGE\_COMPLETED**%' as an indication that the image has been completely processed. After this, we end the connection with the current client.

Code snippet:

**files.close()**

**print("Connection closed")**

**client\_socket.send(b'%IMAGE\_COMPLETED%')**

**client\_socket.close()**

**Client**

**Steps involved in client socket:**

**Step 1:** In the client.py file, we import socket and then define HEADER,PORT as well as FORMAT. We even specify the server ip address using SERVER = **"192.168.1.5".**

**Code snippet:**

**HEADER = 64**

**PORT = 5050**

**FORMAT = 'utf-8'**

**Step 2:** We create an **ADDR** constant where we pass the **SERVER and PORT** as arguments in the form of tuple.

**Step 3:** We create a server using socket Mojo and socket class. Within the socket class we pass two arguments socket.**AF\_1NET for IP** and **socket.SOCK\_STREAM** for TCP. Thus, we create an IP and TCP socket naming it client.

**Step 4:** We now use the **connect()** function to connect the client with the server. We pass **ADDR** tuple as an argument which contains the server address.

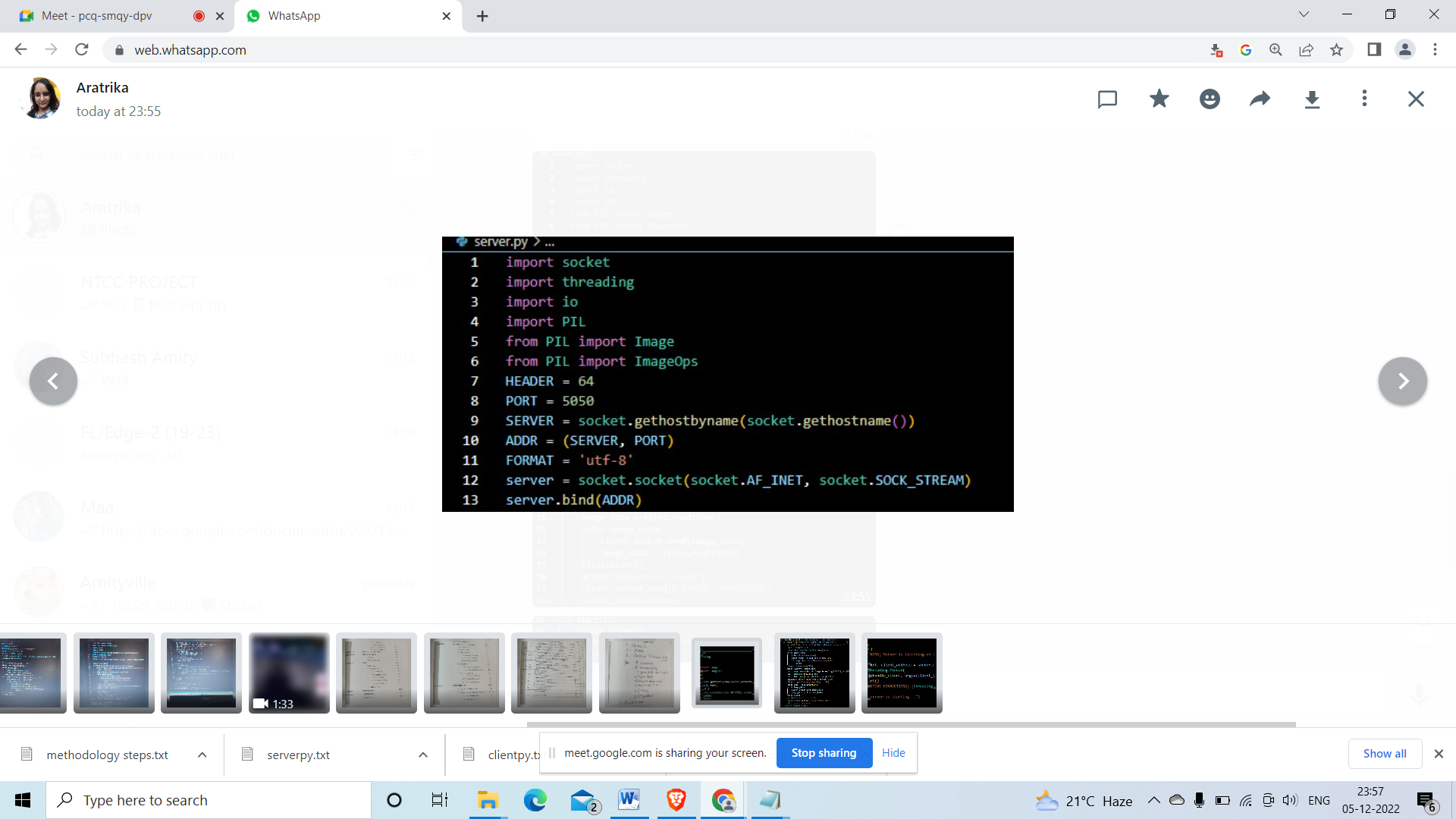
**Step 5:** We now open the network.jpg file to be sent by this client to the server using **file = open('apple.jpg', 'rb')**. Under a while function using **image\_data** as the condition we **use client.send(image\_data)** where we send our data to the server and then we read the **image\_data** in our files using **files.read(image\_data).** Here client is the client socket object.

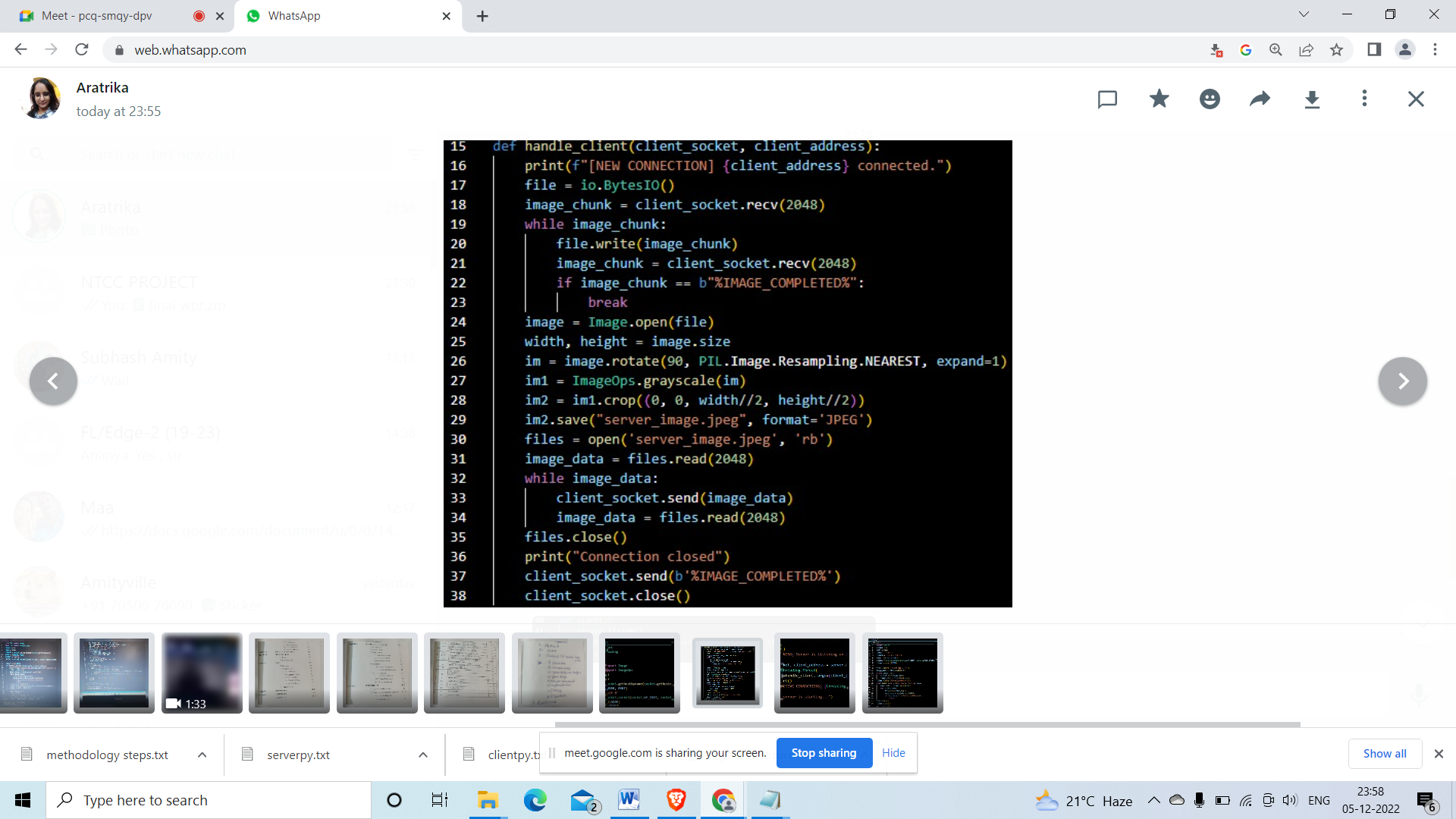
**Step 6:** We send the marker **b'%IMAGE\_COMPLETED%**' as an indication that the image has been completely read and sent to the server.

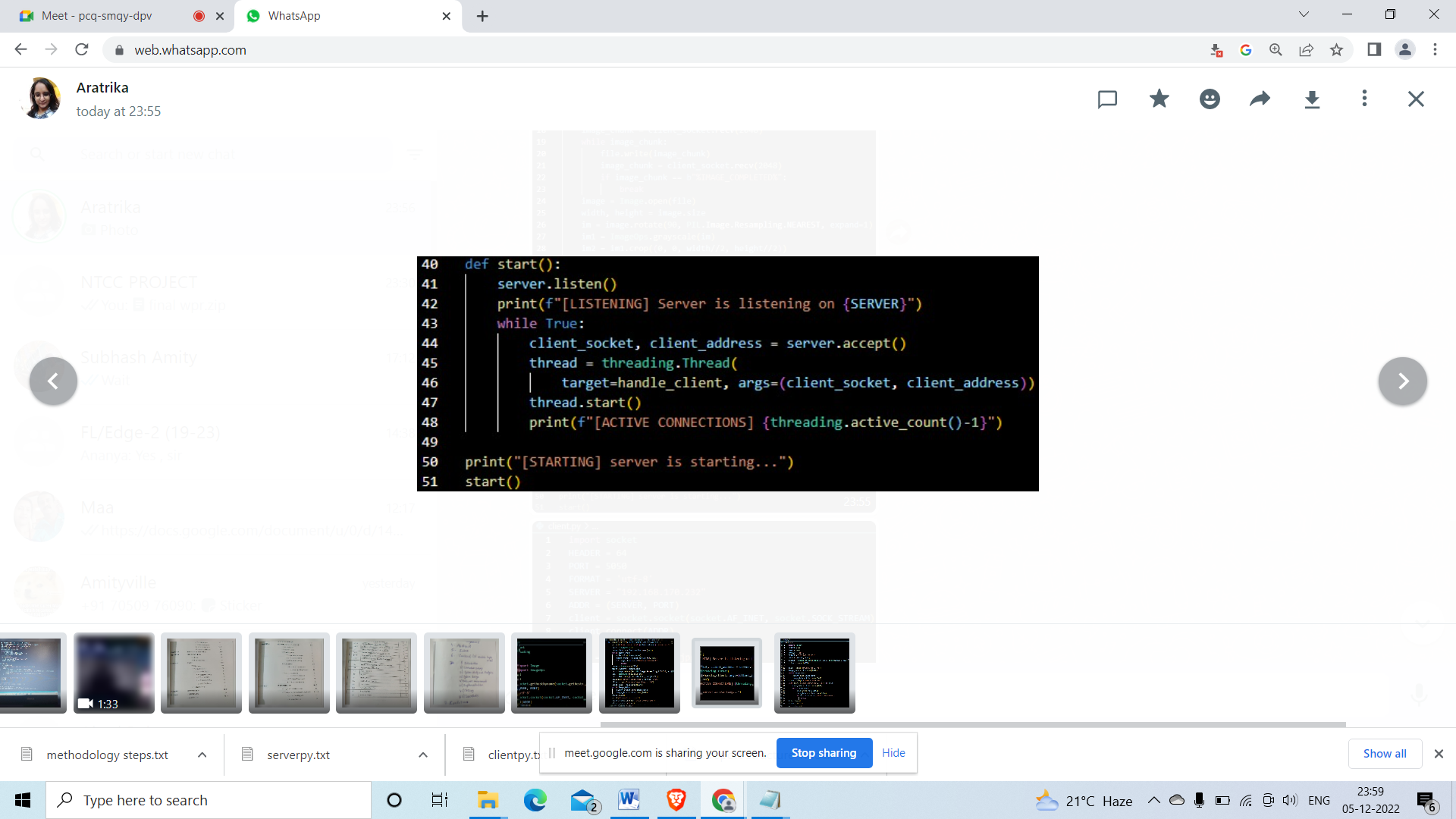
**Step 7:** We open the received processed image as file like with **open('client\_edit.jpeg','wb')** as file. We now receive bytes of processed data from the server using **recv\_data=client.recv(2048**). Under a while function using **recv\_data** as the condition we write the **recv\_data** in our file using **file.write(recv\_data)**. We then use recv\_data= client.recv(2048) to receive new data from the server. Now here if **recv\_data** is equal to the bytes of image completed using **recv\_data==b"%IMAGE\_COMPLETED%"** then we break. It means that the entire data is received and we no longer require to wait.

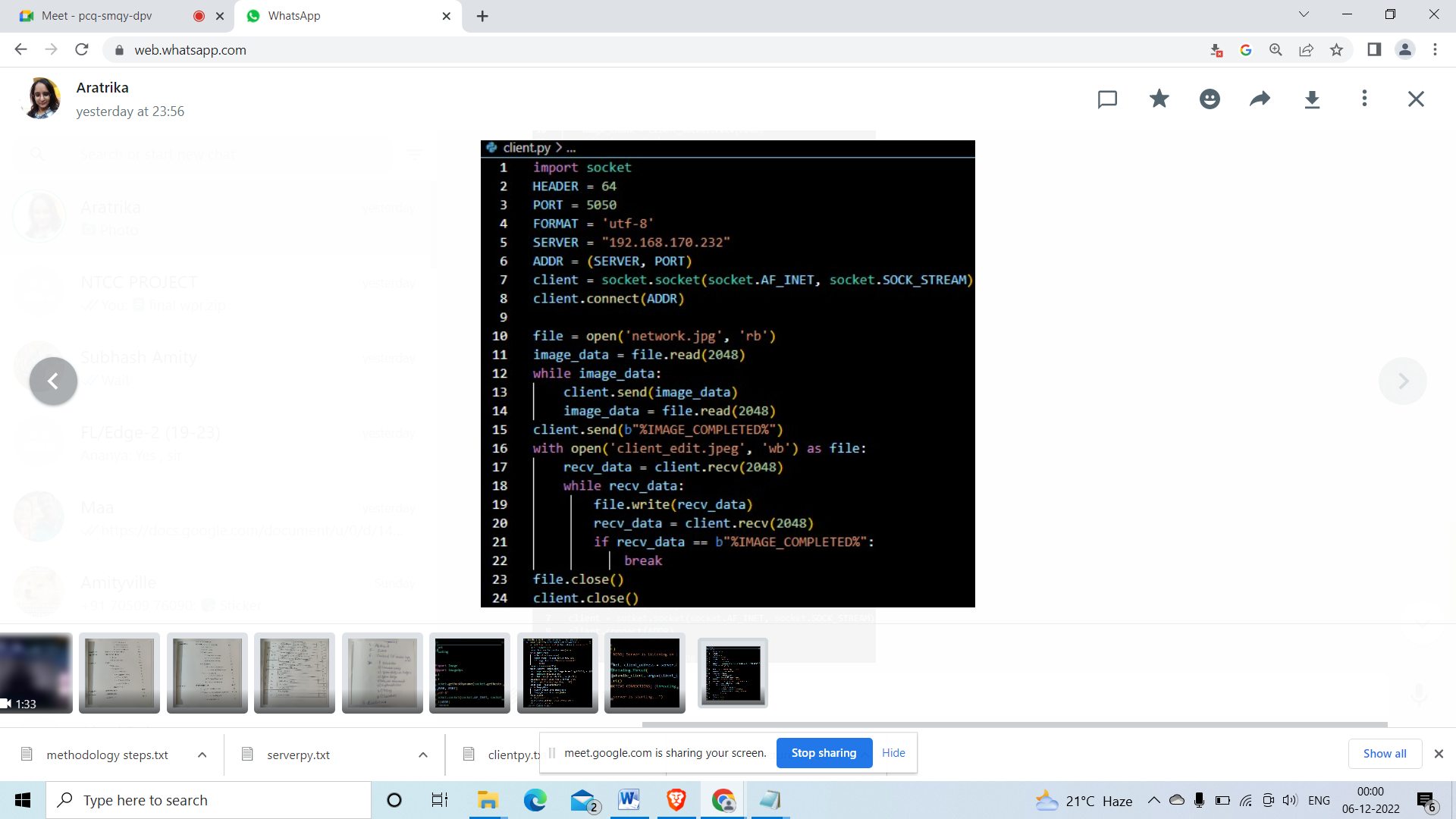
**Step 9:**We close the file and we end the connection with the server.

**3.2 CODE SNIPPET**









**RESULT AND DISCUSSION**

**ORIGINAL IMAGE PROCESSED IMAGE**

**Server\_image.jpeg**

**network.jpg**





We have learned about numerous image processing methods, such as cropping, rotating, and grayscaling using PIL, from the minor project on the topic of "Image processing using socket programming."

To develop a client-server architecture, we have honed our socket programming skills. Because of this project, we gained more knowledge about socket programming and image processing in general.

**Client\_image.jpeg**



The original picture (network.jpg) is initially sent from the client to the server. The processed image is transferred to the client and saved there as client image.jpg. After processing theoriginal image it is saved as server\_image.jpg on the server side.

We have learned about numerous image processing methods, such as cropping, rotating, and grayscaling using PIL, from the minor project on the topic of "Image processing using socket programming."

To develop a client-server architecture, we have honed our socket programming skills. Because of this project, we gained more knowledge about socket programming and image processing in general.

**CONCLUSION**

From the minor project on the topic "Image processing using socket programming" we have gathered knowledge regarding various image processing techniques such as cropping, rotating and grayscaling using PIL.

We have gained expertise with socket programming to create a client-server architecture. We learned more about socket programming and image processing overall because to this project.

**FUTURE SCOPE**

Due to its incorporation into numerous autonomous applications, image processing is becoming increasingly popular in the tech sector. Nearly every discipline, including education, research in the sciences and medicine, traffic management, and government services, finds use for this field. Given the abundance of available picture data and the fact that an image is a matrix of numbers, deep learning is the method of choice for processing images since it makes it simple to transmit input to many layers of neurons for the extraction of key properties. If this system is integrated into an application, these features aid in determining the course of action to be performed. You learn more as you try with more diverse image processing projects.

Real-time analytics, instant chat, binary streaming, and document collaboration are all possible with socket programming. As everyone is aware, the popular messaging service WhatsApp Messenger also employs the socket programming idea for free messaging and chats. Socket programming is used by various apps, including Grofers, Karigar, OLA, and Uber, in addition to messaging applications, to send and receive data from clients and servers. The idea behind Uber is for users to send requests for taxis, which drivers can accept or deny at the other end. Users of these apps send requests for any service or product, and a server responds with the appropriate services and goods within a specific time frame.

Client server-based apps may become more popular as the impact of the client-server model on improving online business grows. Client-server based applications play a big part in how people connect with online commercial enterprises dispersed throughout the internet. Through these popular websites, millions of consumers engage in daily interaction related to their individual interests.

**REFERENCES**

<https://www.sciencedirect.com/science/article/abs/pii/S0262885602000902>

<https://www.spiedigitallibrary.org/conference-proceedings-of-spie/2187/0000/Software-implementation-of-MPEG-II-video-encoding-using-socket-programming/10.1117/12.174957.short>

https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-018-0383-6