به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی برق

گزارشکار تمرین سری اول درس چند رسانه ای

استاد:

دکتر شریفیان

دانشجویان:

مهدی صفری

کسری حسنی

محمد نصیری

فروردین ۱۴۰۳

نحوه انجام مکالمه

برای این قسمت از برنامه یک UI طراحی شد که در آن webcam به صورت آنلاین نمایش داده می شود و با زدن دکمه های موجود در UI می توان از تصویر webcam یک فریم capture کرد و آن را نمایش داد و همچنین می توان یک صدا را به مدت ۳۰ ثانیه ضبط کرد و بعدا آن را گوش کرد. در ضمن این فایل ها دخیره میشوند تا ارسال شوند.

ایده و الگوریتم طراحی شده

در این قسمت از ۳ تا کلاس استفاده شده است که اولی برای ضبط webcam و دومی برای ضبط صدا و سومی برای نشان دادن UI است که تحت یک اپلیکیشن اجرا می شوند. که در قسمت توضیح کد به آن عمیق تر می پردازیم.

توضیح برنامه نوشته شده

اضافه کردن کتابخانه های مورد نیاز

```
import sys
import cv2
import socket
import pickle
import struct
import pyaudio
import wave
import wave
from PyQt6.QtCore import QThread, pyqtSignal, Qt
from PyQt6.QtGui import QImage, QPixmap, QFont
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel,
QPushButton, QMessageBox
```

استایل ها برای دکمه های UI

```
################## strings for styles ############################
str1 = 'background-color:green'
str2 = 'background-color:red'
str3 = 'background-color:yellow'
str4 = 'background-color:blue'
```

کلاس webcame

```
################## Initializing Webcam ############################
class WebcamThread(QThread):
    frame_ready = pyqtSignal(object)

    def __init__(self, parent=None):
        super(WebcamThread, self).__init__(parent)
        self.webcam = cv2.VideoCapture(0)

    def run(self):
        while True:
            ret, frame = self.webcam.read()
            if ret:
                frame_resized = cv2.resize(frame, (200, 200))  # Resize the frame
                self.frame_ready.emit(frame_resized)
```

کد بالا یک کلاس به نام WebcamThread را تعریف می‌کند که یک نخسته وبکم را با استفاده از کتابخانه OpenCV اجرا می‌کند. هر بار که یک فریم از وبکم خوانده می‌شود، فریم را به اندازه 200*200 پیکسل تغییر می‌دهد و سیگنال frame_ready را به استفاده‌کننده‌ها ارسال می‌کند.

کلاس WebcamThread: این کلاس زیر کلاس QThread است و وظیفه گرفتن فریم ها از وب کم را بر عهده دارد.

سیگنال frame_ready: این سیگنال از نوع pyqtSignal زمانی که یک فریم آماده است منتشر می شود.

متد __init__: سازنده شی WebcamThread را مقداردهی اولیه می کند و وب کم را با استفاده از cv2.VideoCapture(۰) باز می کند.

روش run: این متد زمانی اجرا می شود که thread شروع به اجرا کند. به طور مداوم فریم ها را از وب کم می خواند، با استفاده از cv2.resize اندازه آنها را به ۲۰۰*۲۰۰ پیکسل تغییر می دهد، و سیگنال frame_ready را با داده های فریم تغییر اندازه منتشر می کند.

```
################## Initializing Record ############################
class AudioRecorder(QThread):
    def __init__(self, parent=None):
        super(AudioRecorder, self).__init__(parent)
        self.CHUNK = 1024
        self.FORMAT = pyaudio.paInt16
        self.CHANNELS = 1
        self.RATE = 44100
        self.RECORD_SECONDS = 30  # Adjust the recording duration as needed
        self.frames = []

    def run(self):
        audio = pyaudio.PyAudio()
        stream = audio.open(format=self.FORMAT,
                            channels=self.CHANNELS,
                            rate=self.RATE,
                            input=True,
                            frames_per_buffer=self.CHUNK)

        print("Recording...")
        for i in range(0, int(self.RATE / self.CHUNK * self.RECORD_SECONDS)):
            data = stream.read(self.CHUNK)
            self.frames.append(data)

        print("Finished recording")

        stream.stop_stream()
        stream.close()
        audio.terminate()


    def get_audio_data(self):
        return b''.join(self.frames)
```

کد بالا یک کلاس AudioRecorder را تعریف می کند و وظیفه ضبط صدا را بر عهده دارد. در اینجا به تفکیک کد را توضیح می دهم:

کلاس AudioRecorder: این کلاس عملکرد ضبط صدا را انجام می دهد.

روش __init__: سازنده پارامترهای ضبط صدا مانند اندازه قطعه، قالب، تعداد کانال ها، سرعت نمونه برداری، مدت زمان ضبط و یک لیست خالی را برای ذخیره فریم های صوتی مقداردهی اولیه می کند.

روش run: این متد زمانی اجرا می شود که thread شروع به اجرا کند. از کتابخانه pyaudio برای باز کردن یک فایل صوتی استفاده می‌کند، داده‌های صوتی را در تکه‌هایی به اندازه CHUNK برای مدت RECORD_SECONDS ضبط می‌کند و داده‌های ضبط‌شده را به فهرست فریم‌ها اضافه می‌کند.

متد get_audio_data: این روش داده های صوتی ضبط شده را با پیوستن فریم های صوتی ضبط شده در لیست فریم ها برمی گرداند.

به طور کلی، این قطعه کد یک رشته برای ضبط داده های صوتی در زمان واقعی تنظیم می کند. با استفاده از روش get_audio_data می توان به داده های صوتی ضبط شده دسترسی داشت.

تنظیم UI:

```
class Window(QWidget):
    def __init__(self):
        super(Window, self).__init__()

        self.webcam_thread = WebcamThread()
        self.webcam_thread.frame_ready.connect(self.update_webcam_frame)
        self.webcam_thread.start()

        self.webcam_label = QLabel()
        self.webcam_label.setFixedSize(640, 360)

        self.capture_button = QPushButton("Capture")
        self.capture_button.setFont(QFont("Times New Roman",20))
        self.capture_button.setStyleSheet(str1)
        self.capture_button.clicked.connect(self.capture_photo)

        self.show_button = QPushButton("Show")
        self.show_button.setFont(QFont("Times New Roman",20))
        self.show_button.setStyleSheet(str2)
        self.show_button.clicked.connect(self.show_photo)


        self.record_button = QPushButton("Record")
        self.record_button.setFont(QFont("Times New Roman",20))
        self.record_button.setStyleSheet(str3)
        self.record_button.clicked.connect(self.record_audio)

        self.play_button = QPushButton("Play")
        self.play_button.setFont(QFont("Times New Roman",20))
```

```
        self.play_button.setStyleSheet(str4)
        self.play_button.clicked.connect(self.play_audio)

        layout = QVBoxLayout()
        layout.addWidget(self.webcam_label)
        layout.addWidget(self.capture_button)
        layout.addWidget(self.show_button)
        layout.addWidget(self.record_button)
        layout.addWidget(self.play_button)

        self.setLayout(layout)

        self.captured_photo = None  # Variable to store the captured photo
        self.audio_recorder = AudioRecorder()
################## functions to take action ############################
    def update_webcam_frame(self, frame):
        h, w, _ = frame.shape
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        qimg = QImage(rgb_frame.data, w, h, QImage.Format.Format_RGB888)
        qimg_scaled = qimg.scaled(self.webcam_label.size(),
Qt.AspectRatioMode.KeepAspectRatio)
        self.webcam_label.setPixmap(QPixmap.fromImage(qimg_scaled))

    def capture_photo(self):
        self.captured_photo = self.webcam_thread.webcam.read()[1]  # Capture
photo from webcam
    def show_photo(self):
        if self.captured_photo is not None:
            # Display the captured photo in a separate window
            cv2.imshow("Captured Photo", self.captured_photo)
            cv2.imwrite("image.png", self.captured_photo)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
        else:
            QMessageBox.warning(self, "Warning", "No photo captured yet.")
    def record_audio(self):
        self.audio_recorder.start()
        audio_data = self.audio_recorder.get_audio_data()
        if audio_data:
            print("saving...")
            with wave.open('voice.wav', 'wb') as f:
                f.setnchannels(1)
                f.setsampwidth(2)
                f.setframerate(44100)
                f.writeframes(audio_data)
```

```
    def play_audio(self):
        audio_data = self.audio_recorder.get_audio_data()
        if audio_data:
            print("saving...")
            with wave.open('voice.wav', 'wb') as f:
                f.setnchannels(1)
                f.setsampwidth(2)
                f.setframerate(44100)
                f.writeframes(audio_data)
            audio = pyaudio.PyAudio()
            stream = audio.open(format=audio.get_format_from_width(2),
                                channels=1,
                                rate=44100,
                                output=True)
            stream.write(audio_data)
            stream.stop_stream()
            stream.close()
            audio.terminate()

        else:
            print("No audio data recorded yet")
    # Function to send data over a network connection
```

کد بالا یک کلاس Window را تعریف می کند که یک پنجره رابط کاربری گرافیکی با نمایشگر وب کم و دکمه هایی برای گرفتن عکس، نمایش عکس، ضبط صدا و پخش صدا تنظیم می کند. در اینجا به تفکیک کد را توضیح می دهم:

روش __init__: سازنده پنجره رابط کاربری گرافیکی با نمایشگر وب‌کم، دکمه‌های ضبط، نمایش، ضبط و پخش صدا را مقداردهی می‌کند. همچنین نمونه هایی از WebcamThread و AudioRecorder را ایجاد می کند.

روش update_webcam_frame: پنجره نمایش وب کم را با فریم فعلی که توسط وب کم گرفته شده به روز می کند.

روش capture_photo: عکسی را از وب کم می گیرد و آن را در متغیر captured_photo ذخیره می کند.

روش show_photo: عکس گرفته شده را در یک پنجره جداگانه نمایش می دهد و آن را به عنوان فایل تصویری ذخیره می کند.

روش record_audio: ضبط صدا را آغاز می کند، صدای ضبط شده را به عنوان یک فایل WAV ذخیره می کند و شروع به ذخیره داده های صوتی می کند.

روش play_audio: صدای ضبط شده را پخش می کند، آن را به عنوان یک فایل WAV ذخیره می کند و صدا را با استفاده از PyAudio پخش می کند.

به طور کلی، این قطعه کد یک پنجره رابط کاربری گرافیکی با نمایش وب کم و عملکردی برای گرفتن عکس، نمایش عکس، ضبط صدا و پخش صدا تنظیم می کند. همچنین عکس گرفته شده و صدای ضبط شده را به ترتیب به صورت تصویر و فایل های WAV ذخیره می کند.

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Window()
    window.show()
    sys.exit(app.exec())
```

کد بالا یک ساختار معمولی است که در برنامه های PyQt برای ایجاد و اجرای پنجره اصلی برنامه استفاده می شود. در اینجا خلاصه ای از آنچه هر بخش انجام می دهد آورده شده است:

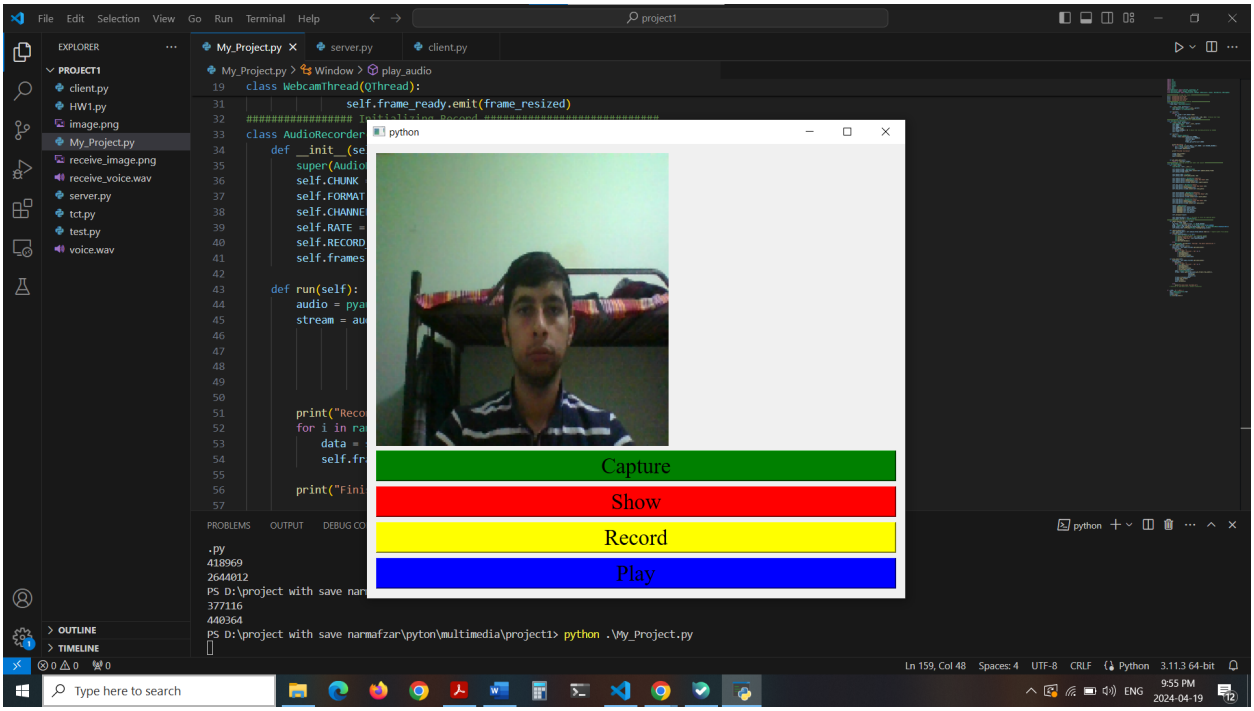
if __name__ == "__main__":: این شرط بررسی می کند که آیا اسکریپت به عنوان برنامه اصلی اجرا می شود یا خیر.

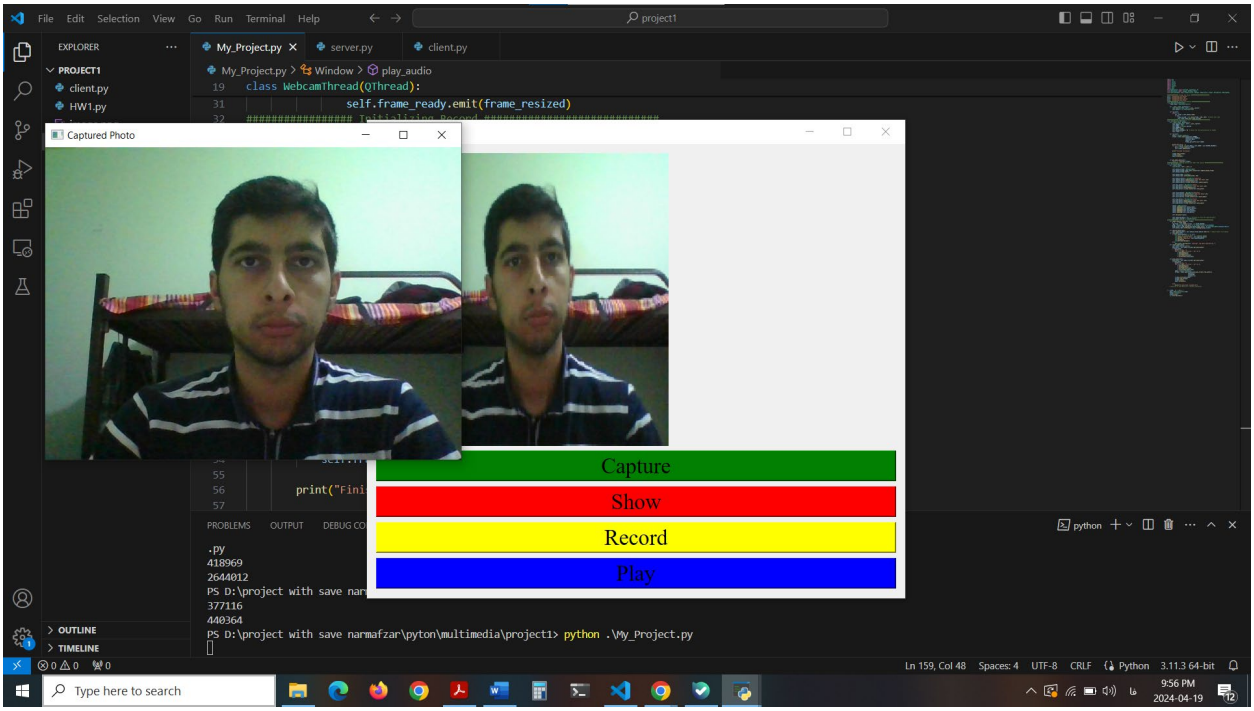
app = QApplication(sys.argv): این خط نمونه ای از کلاس QApplication ایجاد می کند

window = Window(): این خط نمونه ای از کلاس Window را ایجاد می کند که نمایانگر پنجره اصلی برنامه است.

window.show(): این خط پنجره اصلی برنامه را روی صفحه نمایش می دهد.

خروجی سیستم:


python
Capture
Show
Record
Play
PS D:\project with save narmafzar\pyton\multimedia\project1> python .\My_Project.py


Captured Photo
python
Capture
Show
Record
Play
PS D:\project with save narmafzar\pyton\multimedia\project1> python .\My_Project.py

نحوه انتقال فایل ها

ایده و الگوریتم طراحی شده:

در اینجا دو قطعه کد داریم که یکی به نام server و دیگری به نام client است که با توجه به اینکه کدام یک از لبتاب ها وظیفه server با client داشته باشند آن کد را اجرا میکنند و ارتباط از طریق آدرس IP برقرار میشود.

توضیح برنامه نوشته شده:

```
import socket
import cv2

server_ip = '192.168.234.68'
server_port_image = 7000
server_port_voice = 5000

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port_image))
server_socket.listen()
print("Server: Server is Running")


def read_data(cs: socket, filename):
    data_list = b''
    while data:=cs.recv(1024):
        data_list += data
    with open(filename, "wb") as file:
        file.write(data_list)
    print(len(data_list))

client_socket, client_info = server_socket.accept()
read_data(client_socket, "receive_image.png")
print(f"Server: Accepted new Connection on {client_info}")
server_socket.close()

#receive voice
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port_voice))
server_socket.listen()
```

```
print("Server: Server is Running")

client_socket, client_info = server_socket.accept()
read_data(client_socket, "receive_voice.wav")
print(f"Server: Accepted new Connection on {client_info}")
server_socket.close()
```

قطعه کد بالا سروری را راه اندازی می کند که به اتصالات ورودی گوش می دهد تا تصویر و داده های صوتی را از طریق پورت های جداگانه دریافت کند. در اینجا به تفکیک کد را توضیح می دهم:

راه اندازی سرور تصویر:

سرور به آدرس IP '192.168.234.68' و پورت ۷۰۰۰ برای دریافت داده های تصویر متصل است.

به اتصالات ورودی گوش می دهد و پیامی را چاپ می کند که نشان می دهد سرور در حال اجرا است.

هنگامی که یک کلاینت متصل می شود، سرور اتصال را می پذیرد، داده های تصویر را می خواند و آن را به عنوان "receive_image.png" ذخیره می کند.

سپس سرور طول داده های تصویر دریافتی را چاپ می کند و اتصال را می بندد.

راه اندازی سرور صوتی:

سرور دیگری بر روی همان آدرس IP اما در پورت ۵۰۰۰ برای دریافت داده های صوتی راه اندازی شده است.

مشابه سرور تصویر، به اتصالات ورودی گوش می دهد و پیامی را چاپ می کند که نشان می دهد سرور در حال اجرا است.

هنگامی که یک کلاینت متصل می شود، سرور اتصال را می پذیرد، داده های صوتی را می خواند و آن را به عنوان "receive_voice.wav" ذخیره می کند.

سپس سرور طول داده های صوتی دریافتی را چاپ می کند و اتصال را می بندد.

تابع read_data:

این تابع داده ها را از سوکت سرویس گیرنده در تکه های ۱۰۲۴ بایتی می خواند و در فایلی که با نام فایل مشخص شده است می نویسد.

به طور کلی، این قطعه کد سروری را برای دریافت داده های تصویری و صوتی از کلاینت ها از طریق پورت های جداگانه تنظیم می کند و داده های دریافتی را در فایل ها ذخیره می کند.

```
import socket

server_ip = '192.168.234.68'
server_port_image = 7000
server_port_voice = 5000

cs = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cs.connect((server_ip, server_port_image))

with open("image.png", 'rb') as file:
    data1 = file.read()
print(cs.send(data1))
cs.close()

# send voice
cs = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cs.connect((server_ip, server_port_voice))

with open("voice.wav", 'rb') as file:
    data2 = file.read()
print(cs.send(data2))
cs.close()
```

قطعه کد بالا برای ارسال تصویر و داده های صوتی، به سروری در آدرس IP: 192.168.234.68 در پورت های مختلف (۷۰۰۰ برای داده های تصویری و ۵۰۰۰ برای داده های صوتی) اتصال برقرار می کند. در اینجا به تفکیک کد را توضیح می دهم:

ارسال اطلاعات تصویر:

یک سوکت cs ایجاد شده و به IP سرور و پورت ۷۰۰۰ برای ارسال داده های تصویر متصل می شود.

کد محتوای فایل "image.png" را در حالت باینری می خواند و آن را در data1 ذخیره می کند.

اندازه داده های ارسال شده به سرور با استفاده از cs.send(data1) چاپ می شود.

سپس اتصال بسته می شود.

ارسال داده های صوتی

یک سوکت cs دیگر ایجاد شده و به IP سرور و پورت ۵۰۰۰ برای ارسال داده های صوتی متصل می شود.

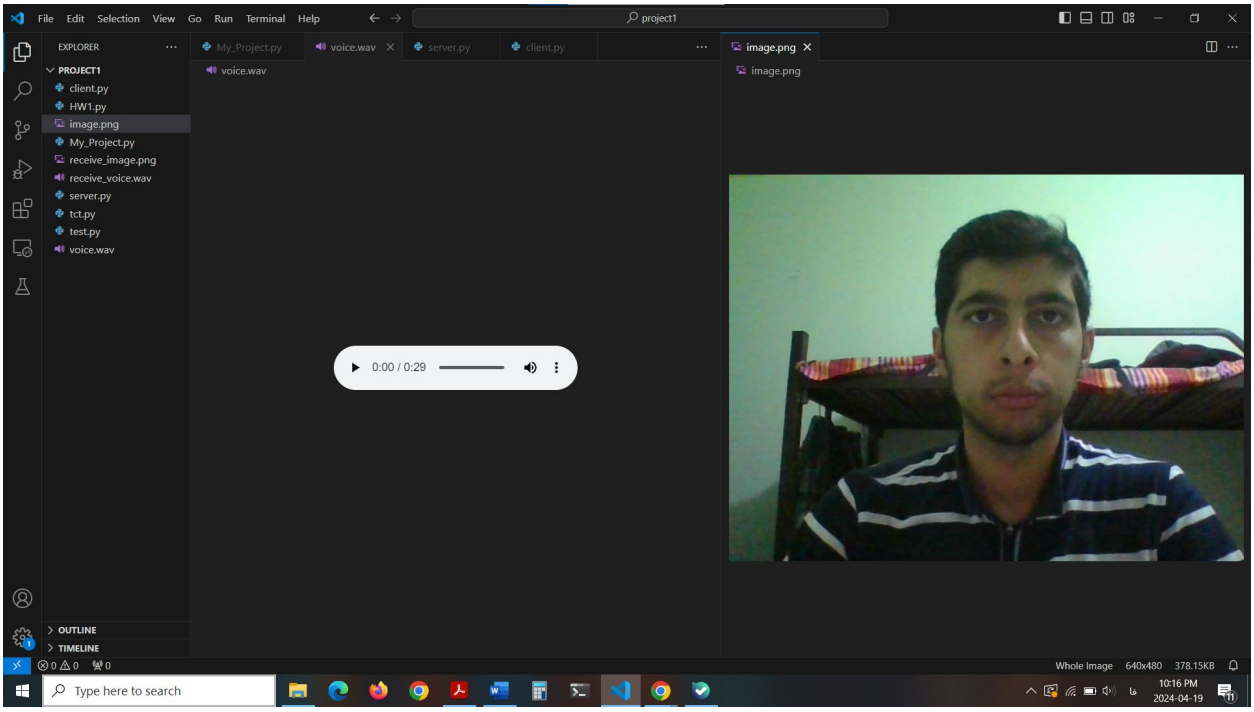
کد محتوای فایل "voice.wav" را در حالت باینری می خواند و آن را در data2 ذخیره می کند.

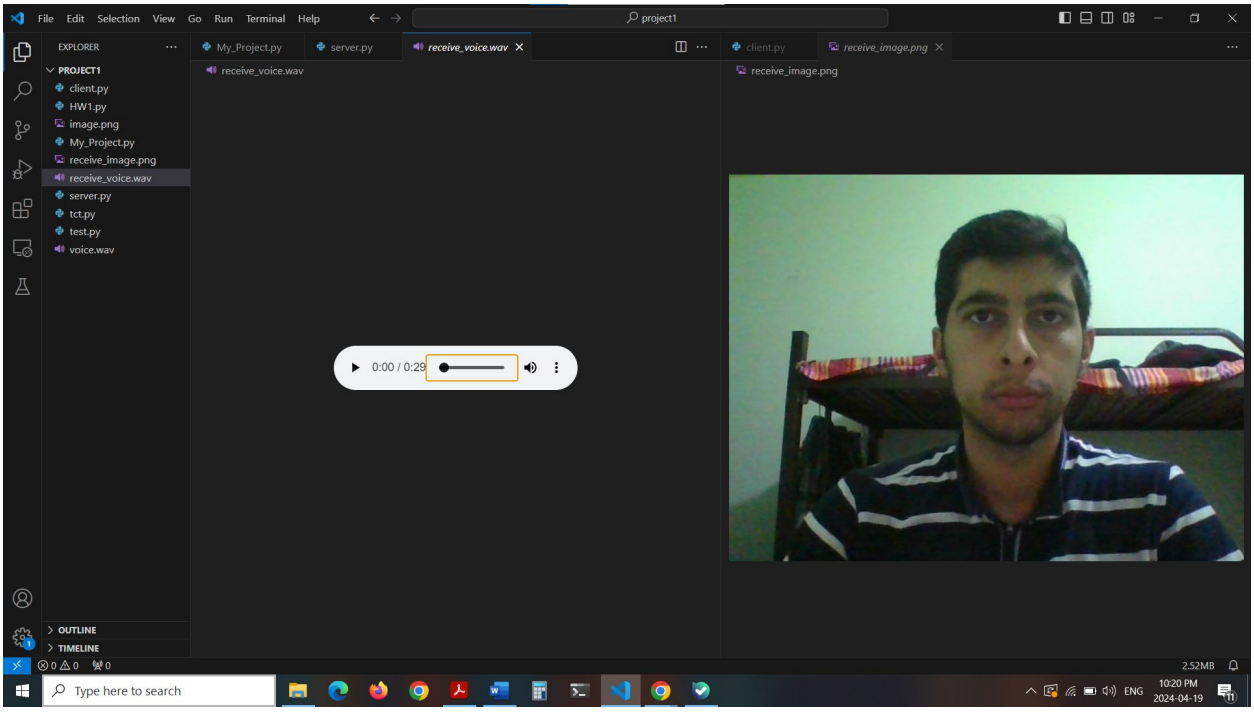
اندازه داده های ارسال شده به سرور با استفاده از cs.send(data2) چاپ می شود.

سپس اتصال بسته می شود.

به طور کلی، این قطعه کد اتصالات را به سرور برقرار می کند و داده های تصویری و صوتی را با استفاده از سوکت ها از طریق اتصالات جداگانه ارسال می کند.

خروجی کد ها


File Edit Selection View Go Run Terminal Help
project1
EXPLORER
PROJECT1
client.py
HW1.py
image.png
My_Project.py
receive_image.png
receive_voice.wav
server.py
tct.py
test.py
voice.wav
My_Project.py
voice.wav
server.py
client.py
image.png
image.png
0:00 / 0:29
OUTLINE
TIMELINE
Whole Image 640x480 378.15KB
Type here to search
10:16 PM
2024-04-19


File Edit Selection View Go Run Terminal Help
project1
EXPLORER
PROJECT1
client.py
HW1.py
image.png
My_Project.py
receive_image.png
receive_voice.wav
server.py
tct.py
test.py
voice.wav
My_Project.py
server.py
receive_voice.wav
client.py
receive_image.png
receive_voice.wav
receive_image.png
0:00 / 0:29
OUTLINE
TIMELINE
2.52MB
Type here to search
ENG
10:20 PM
2024-04-19