A

COURSE PROJECT REPORT

On

SUMMARIZING AUDIO FILES IN PYTHON

Submitted to the Faculty of Engineering and Technology

B. Tech - V Semester

By

Pittala Sivamani

B21CS162

Under the Guidance of
Sri. S. Naga Raju
Associate Professor

Department of Computer Science and Engineering

Kakatiya Institute of Technology & Science

(An Autonomous Institute under Kakatiya University)

Warangal – Telangana

2022-2023

NAAC - 'A' Grade Institute (CGPA : 3.21) ISO 9001:2015 NIRF - 2020 Rank Band : 201-250

KAKATIYA INSTITUTE OF TECHNOLOGY & SCIENCE

Opp : Yerragattu Gutta, Hasanparthy (Mandal), WARANGAL - 506 015, Telangana, INDIA.

काकतीय प्रौद्योगिकी एवं विज्ञान संस्थान, वरंगल - ५०६ ०१५ तेलंगाना, भारत

కాకతీయ సాంకేతిక విజ్ఞాన శాస్త్ర విద్యాలయం, వరంగల్ - ౫౦౬ ౦౧౫ తెలంగాణ, భారతదేశము

Estd-1980 KITSW

(An Autonomous Institute under Kakatiya University, Warangal)

(Approved by AICTE, New Delhi; Recognised by UGC under 2(f) & 12(B); Sponsored by EKASILA EDUCATION SOCIETY)

website: www.kitsw.ac.in E-mail: principal@kitsw.ac.in ✆ : +91 9392055211, +91 7382564888

C E R T I F I C A T E

This is to certify that **Sivamani Pittala** bearing Roll No. **B21CS162** of the V Semester B.Tech. ComputerScience and Engineering has satisfactorily completed the Course Project dissertation work entitled, **“SUMMARIZING AUDIO FILES IN PYTHON”,** in partial fulfilment of the requirements of B. Tech Degree during this academic year 2023-2024.

Supervisor
Dr. P. NIRANJAN
Professor

Coordinator
Sri. S. NAGA RAJU
Associate Professor

Convenor
Sri. S. NAGA RAJU
Associate Professor

Head of the Department
Dr. P. NIRANJAN
Professor

ACKNOWLEDGEMENT

I extend my sincere and heartfelt thanks to our esteemed counselor/guide **Prof. P. NIRANJAN,**for his exemplary guidance, monitoring and consistent encouragement throughout the course at crucialjunctures and for showing us the right way.

It is a great privilege for me to here by deep sense of gratitude towards Seminar Coordinator **Sri. S. NAGA RAJU, Associate Professor** who guided and permitting me in successful completion of my work on time.

I am grateful to respected Mini-Project Convenor **Sri. S. NAGA RAJU, Associate Professor** for guiding and permitting me to utilize all the necessary facilities of the Institute.

I would like to extend thanks to **Dr. P. NIRANJAN, Professor & Head Department of CSE** for allowing me to use the facilities available.

I express my immense delight to **Prof. P. NIRANJAN, Dean, Research & Development** for his kind cooperation and elderly suggestions through out of this project work. I am very much thankfulfor issuing similarity report through Turnitin Anti Plagiarism Software.

I take this opportunity to express my sincere and heartfelt gratitude to honourable Principal **Prof. K. ASHOKA REDDY** of our institute for granting me permission to do this the Seminar.

I am very happy in expressing my sincere thanks to faculty & staff, Department of Computer Science and Engineering, friends and family for the support and encouragement that they have given us during the Seminar.

P. Sivamani

B21cs162

ABSTRACT

The "Summarizing Audio Files in Python" project utilizes Python and libraries like Flask, MoviePy, SpeechRecognition, and Summarizer to efficiently process audio content, covering extraction, transcription, speech recognition, and text summarization. MoviePy aids in extracting audio from videos, SpeechRecognition converts audio to text, and Summarizer generates meaningful summaries. The project extends to real-time scenarios, allowing users to record and summarize spoken words through microphones. Despite facing challenges in audio processing and real-time transcription, the system successfully navigates these complexities. The report concludes by highlighting achievements, showcasing sample outputs, and discussing future enhancements for refinement. The project addresses the need for audio information extraction and summarization, offering a versatile solution with core features and real-time capabilities. The abstract provides a concise overview of objectives, features, outcomes, and potential future developments.

TABLE OF CONTENTS

SUMMARIZING AUDIO FILES IN PYTHON

1. INTRODUCTION

1.1 INTRODUCTION

In today's digital age, the vast amount of audio content available presents both opportunities and challenges. While audio files contain valuable information, efficiently extracting and summarizing this data can be daunting, requiring advanced processing techniques and tools. The "Summarizing Audio Files in Python" project emerges as a response to this need, leveraging the power of Python and various libraries to streamline the extraction and summarization process.

With the proliferation of multimedia platforms and the increasing popularity of podcasts, lectures, and interviews, the demand for effective audio summarization tools has surged. These tools are essential for researchers, journalists, educators, and professionals across diverse fields who seek to glean insights from audio content swiftly and accurately.

The project aims to tackle the complexities associated with audio processing, encompassing tasks such as audio extraction, transcription, speech recognition, and text summarization. By harnessing the capabilities of libraries like Flask, MoviePy, SpeechRecognition, and Summarizer, the project offers a comprehensive solution for processing audio files from various sources, including videos and real-time microphone recordings.

Throughout this report, we delve into the intricacies of the project, addressing key challenges such as audio extraction compatibility, real-time transcription accuracy, and text summarization coherence. We discuss the software and hardware requirements, provide a step-by-step guide for setting up the project environment, and elucidate the core functionalities of the implemented system.

Furthermore, the report outlines the project's significance in facilitating efficient information extraction from audio sources and its potential applications across different domains. By providing users with a versatile tool for summarizing audio content, the project aims to enhance productivity, facilitate research, and foster innovation in fields reliant on audio data analysis.

In summary, the "Summarizing Audio Files in Python" project represents a concerted effort to bridge the gap between vast amounts of audio content and actionable insights. Through meticulous implementation, integration of cutting-edge technologies, and a user-centric approach, the project endeavors to empower users with the tools they need to unlock the wealth of knowledge embedded within audio files.

1.2 OBJECTIVES

The objectives of this report can be listed as

- Develop a robust Python-based system capable of efficiently extracting audio content from various sources, including video files and real-time microphone recordings, to facilitate subsequent processing and analysis.
- Implement advanced audio processing techniques, including transcription, speech recognition, and text summarization, leveraging libraries such as Flask, MoviePy, SpeechRecognition, and Summarizer, to generate concise and meaningful summaries of audio content, addressing challenges such as compatibility, accuracy, and coherence.

2. UNDERSTANDING THE PROBLEM STATEMENT

2.1 PROBLEM STATEMENT

The "Summarizing Audio Files in Python" project addresses the challenge of efficiently extracting and summarizing information from audio sources. With a focus on video files and real-time microphone recordings, the project encounters several key problems:

1. **Audio Extraction:** Extracting audio from video files poses a challenge due to different formats and codecs. Ensuring compatibility and accurate extraction is crucial for subsequent processing.

2. **Real-time Transcription:** Achieving accurate and real-time transcription of spoken words from microphone recordings requires overcoming latency issues and ensuring the system can adapt to diverse speaking styles and accents.

3. **Speech Recognition Accuracy:** The accuracy of converting audio to text, a critical step in the summarization process, is influenced by variations in pronunciation, background noise, and language nuances.

4. **Text Summarization:** Generating concise and meaningful summaries from transcribed text is a complex natural language processing task. Ensuring the summaries capture essential information while remaining coherent and contextually accurate is challenging.

5. **Microphone Recording Stability:** Real-time recording from microphones demands stability and robust error handling to account for potential disruptions, ensuring a seamless user experience.

6. **Integration Complexity:** Integrating multiple libraries (Flask, MoviePy, SpeechRecognition, Summarizer) and managing their interactions adds a layer of complexity, requiring careful synchronization and error handling.

2.2 UTILISED ENVIRONMENTS AND TOOLS

Utilized Environments and Tools:

1. Python: The primary programming language used for developing the project, providing a versatile and efficient platform for audio processing and integration with various libraries.

2. Flask: A lightweight web framework for Python, utilized to create the web interface for user interaction, enabling seamless access to the audio summarization functionalities.

3. MoviePy: A Python library for video editing tasks, employed for extracting audio from video files, enabling efficient handling of multimedia content.

4. SpeechRecognition: A Python library for converting audio to text, facilitating accurate transcription of spoken words from audio sources such as video files and microphone recordings.

5. Summarizer: A Python library for text summarization, utilized to generate concise and meaningful summaries of transcribed audio content, enhancing accessibility and usability.

6. Librosa: A Python package for music and audio analysis, potentially utilized for background noise reduction and additional audio processing tasks.

7. PyAudio: A Python library for audio input and output, used for microphone-related functionalities, including real-time recording and playback.

8. NumPy and SciPy: Scientific computing libraries for Python, potentially used for various audio processing tasks, such as signal processing and data manipulation.

9. FFmpeg: A multimedia framework for handling video, audio, and other multimedia files, utilized for video/audio processing tasks, including extracting audio from video files and handling different formats.

10. Subprocess Module: A module in the Python standard library used for interacting with the system shell to execute external commands, facilitating integration with tools such as FFmpeg.

11. Wave Module: A module in the Python standard library used for reading and writing WAV audio files, ensuring compatibility and efficient handling of audio data.

12. YouTube_dl: A Python library for downloading videos from YouTube, potentially used for accessing and summarizing audio content from online sources.

These environments and tools collectively enable the development of a comprehensive audio summarization system, empowering users to extract, transcribe, and summarize audio content efficiently and accurately.

CUSTOMER SEGMENTATION BENEFITS:

Customer Segmentation Benefits in Short:

1. Targeted Marketing: Reach the right customers with personalized campaigns.
2. Improved Retention:Foster stronger relationships by addressing specific needs.
3. Efficient Resource Use: Allocate resources effectively for maximum impact.
4. Innovation Guidance: Develop products and services that align with market demands.
5. Competitive Edge: Stand out by offering tailored solutions and experiences.

3. METHODOLOGY

Folder Creation:

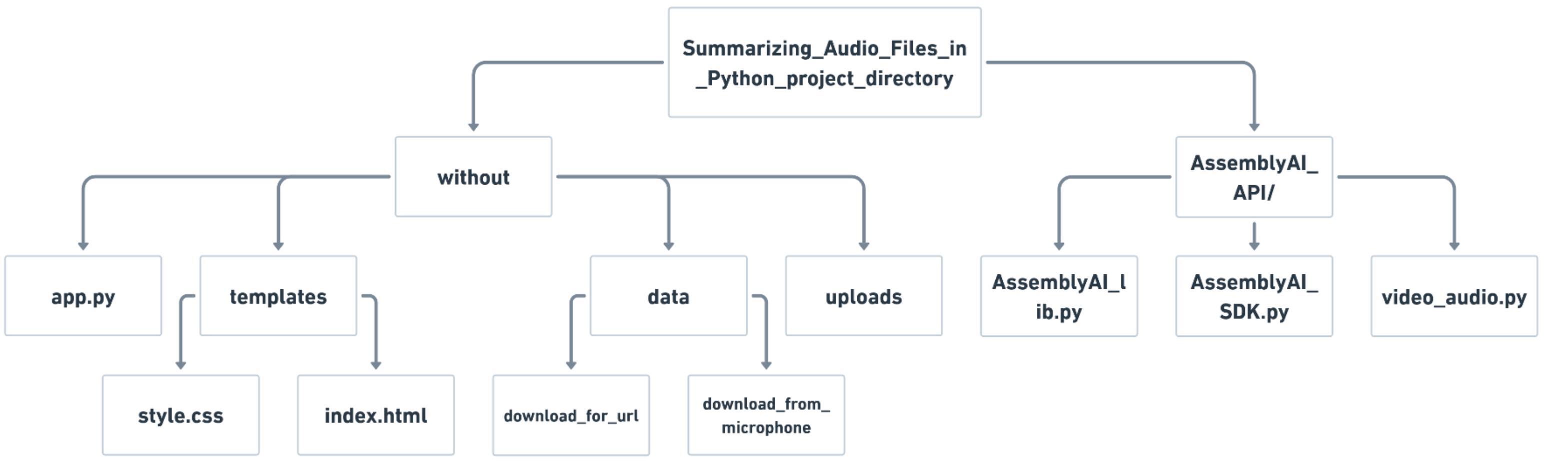
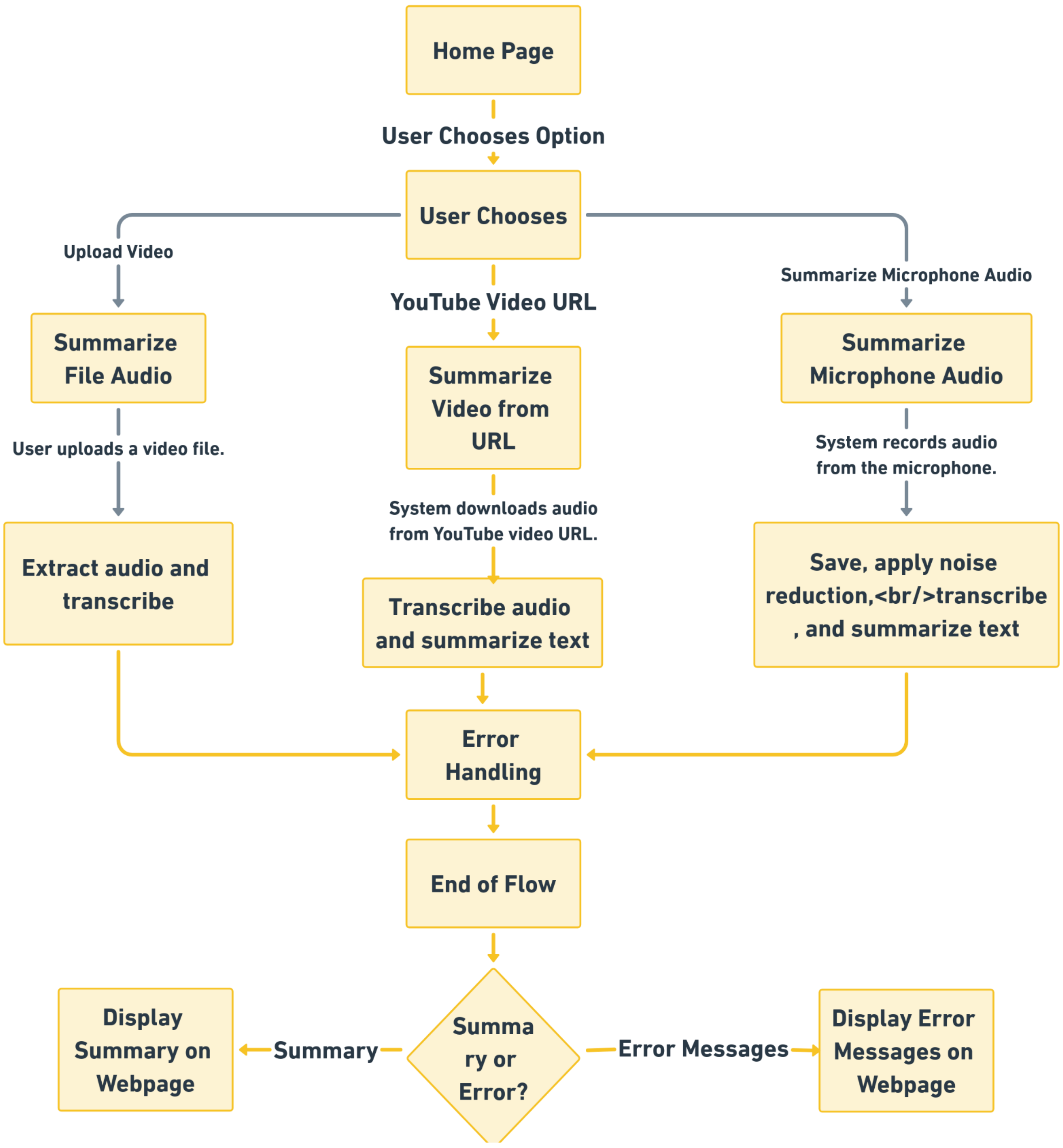


Figure 1: Directory Structure for Summarizing Audio Files in Python Project.

Flow Chart

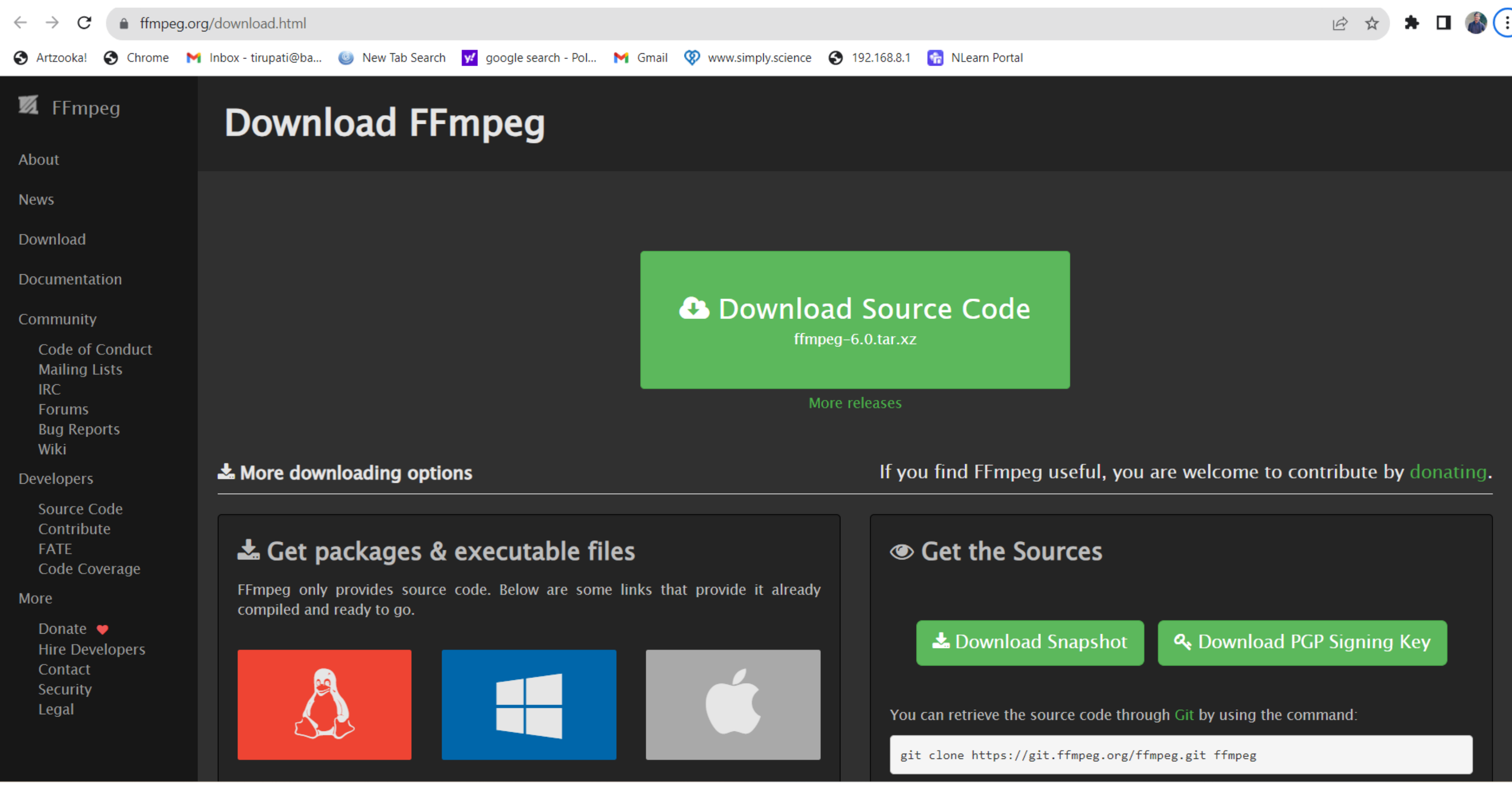


Home Page
User Chooses Option
User Chooses
Upload Video
YouTube Video URL
Summarize Microphone Audio
Summarize File Audio
Summarize Video from URL
Summarize Microphone Audio
User uploads a video file.
System downloads audio from YouTube video URL.
System records audio from the microphone.
Extract audio and transcribe
Transcribe audio and summarize text
Save, apply noise reduction,<br/>transcribe , and summarize text
Error Handling
End of Flow
Summary or Error?
Summary
Error Messages
Display Summary on Webpage
Display Error Messages on Webpage

Figure 2: Audio Summarizer Processing Flow

Step-By-Step process to get Download FFmpeg:

Step-1: open https://ffmpeg.org/download.html#build-windows in browser



Download FFmpeg

More downloading options

If you find FFmpeg useful, you are welcome to contribute by donating.

Get packages & executable files

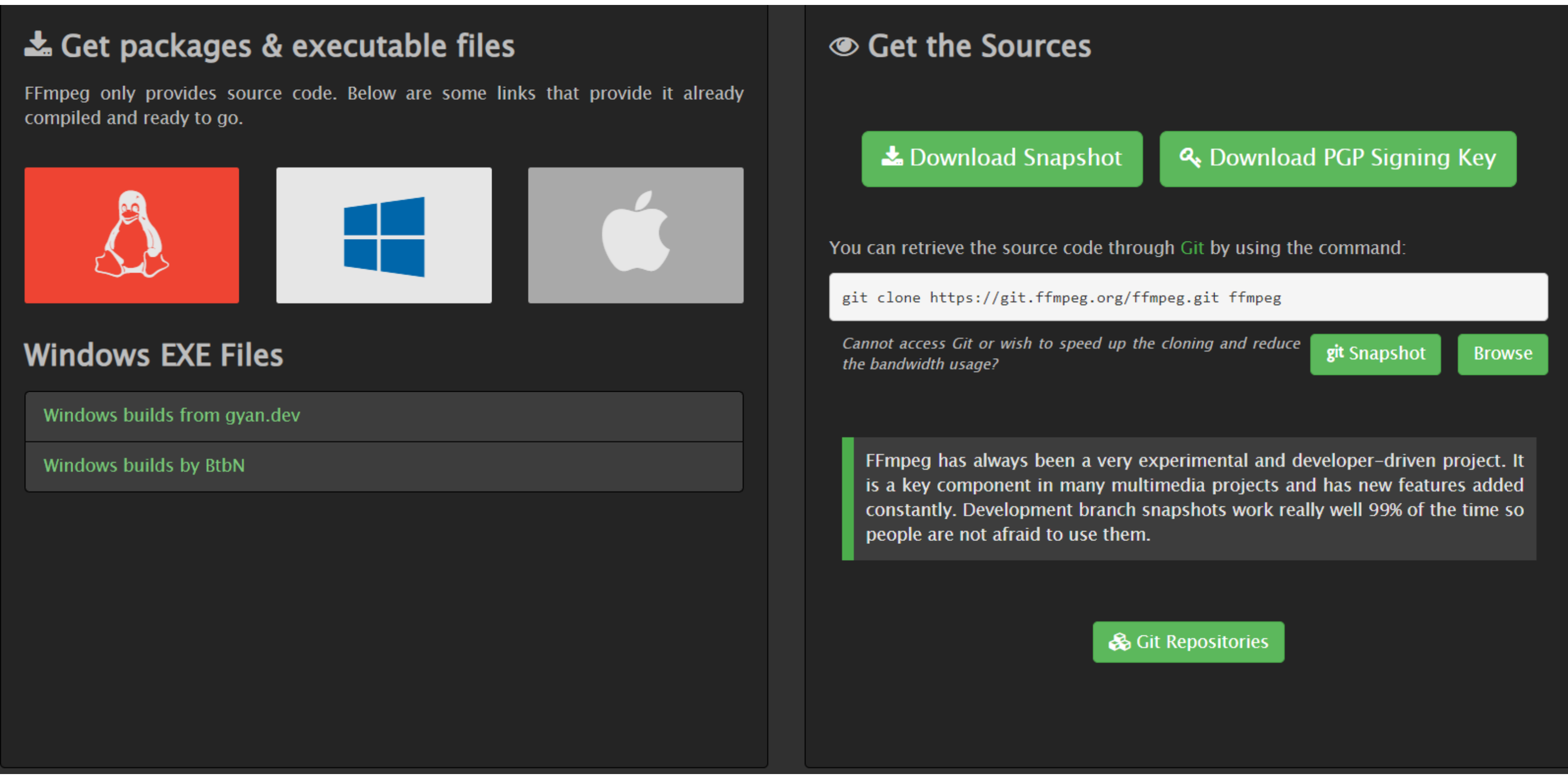
FFmpeg only provides source code. Below are some links that provide it already compiled and ready to go.

Get the Sources

You can retrieve the source code through Git by using the command:

```
git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg
```

Step-2: Select your operation system (for me it is window)



Get packages & executable files

FFmpeg only provides source code. Below are some links that provide it already compiled and ready to go.

Windows EXE Files

Windows builds from gyan.dev

Windows builds by BtbN

Get the Sources

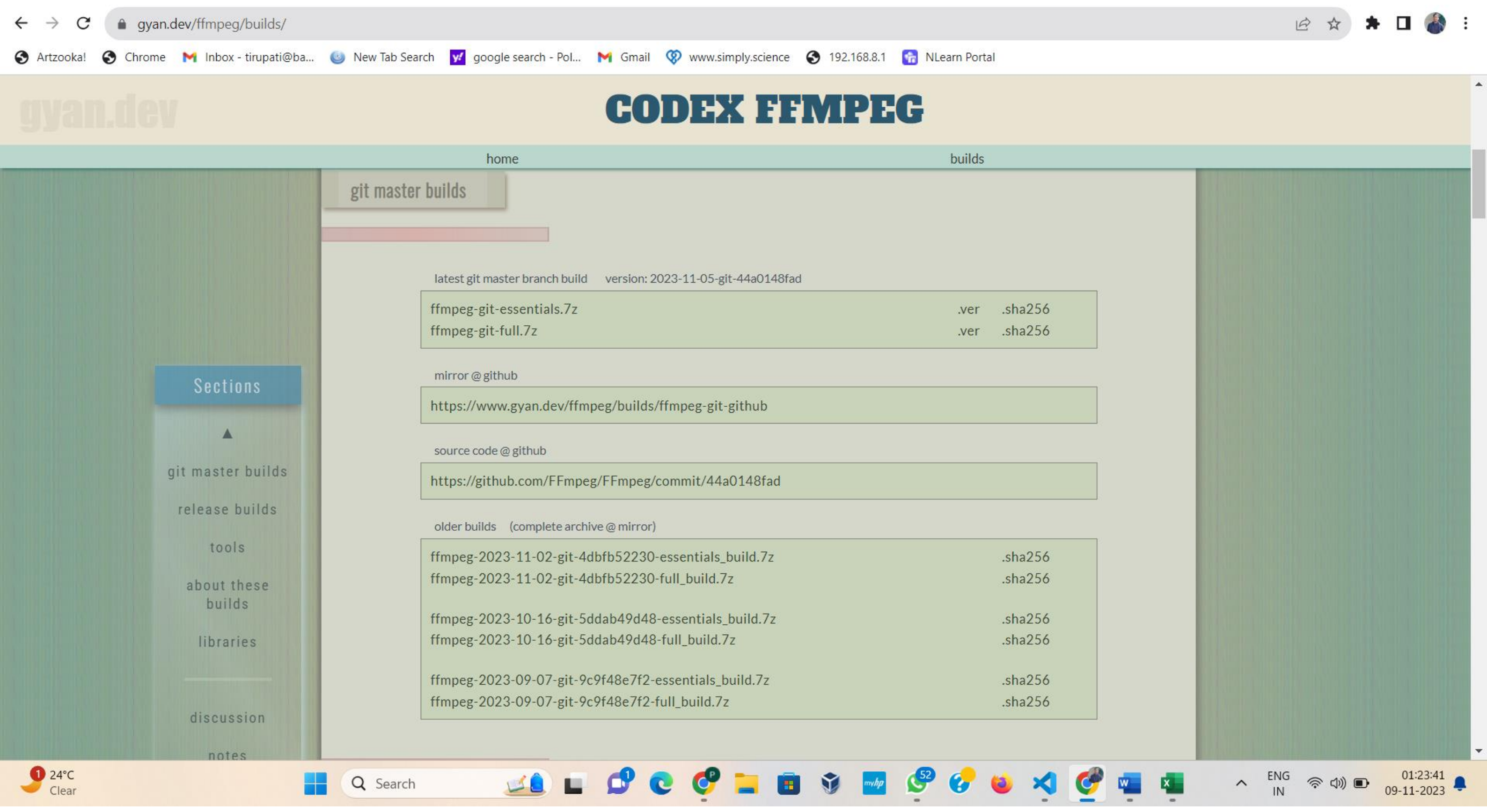
You can retrieve the source code through Git by using the command:

```
git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg
```

Cannot access Git or wish to speed up the cloning and reduce the bandwidth usage?

FFmpeg has always been a very experimental and developer-driven project. It is a key component in many multimedia projects and has new features added constantly. Development branch snapshots work really well 99% of the time so people are not afraid to use them.

Step-3: click on windows builds from gyan.dev (first option)



gyan.dev/ffmpeg/builds/

CODEX FFMPEG

home builds

git master builds

latest git master branch build version: 2023-11-05-git-44a0148fad

```
ffmpeg-git-essentials.7z                                   .ver   .sha256
ffmpeg-git-full.7z                                         .ver   .sha256
```

mirror @ github

https://www.gyan.dev/ffmpeg/builds/ffmpeg-git-github

source code @ github

https://github.com/FFmpeg/FFmpeg/commit/44a0148fad

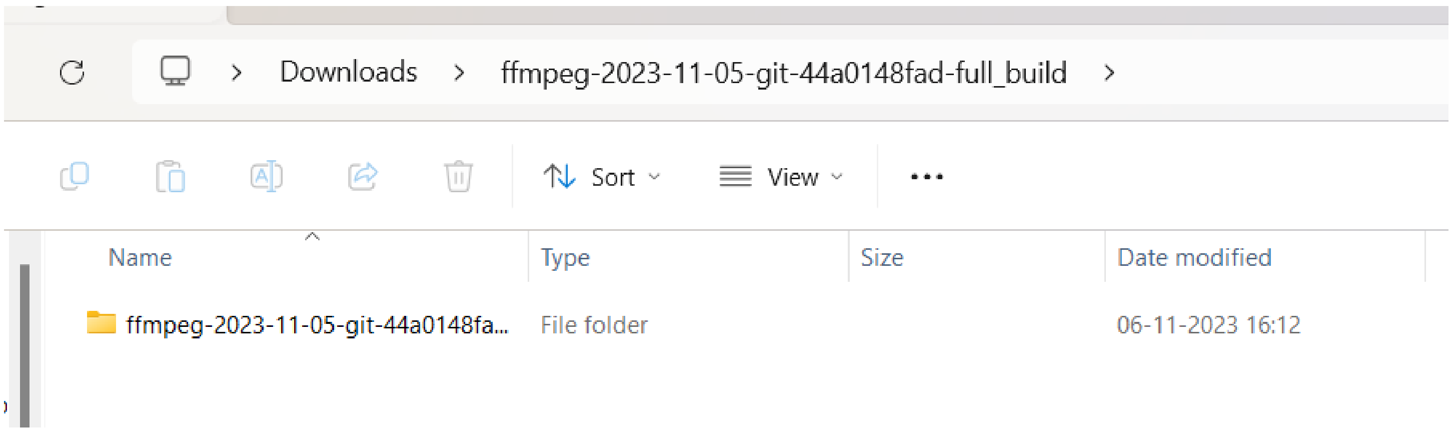
older builds (complete archive @ mirror)

```
ffmpeg-2023-11-02-git-4dbfb52230-essentials_build.7z        .sha256
ffmpeg-2023-11-02-git-4dbfb52230-full_build.7z              .sha256

ffmpeg-2023-10-16-git-5ddab49d48-essentials_build.7z        .sha256
ffmpeg-2023-10-16-git-5ddab49d48-full_build.7z              .sha256

ffmpeg-2023-09-07-git-9c9f48e7f2-essentials_build.7z        .sha256
ffmpeg-2023-09-07-git-9c9f48e7f2-full_build.7z              .sha256
```

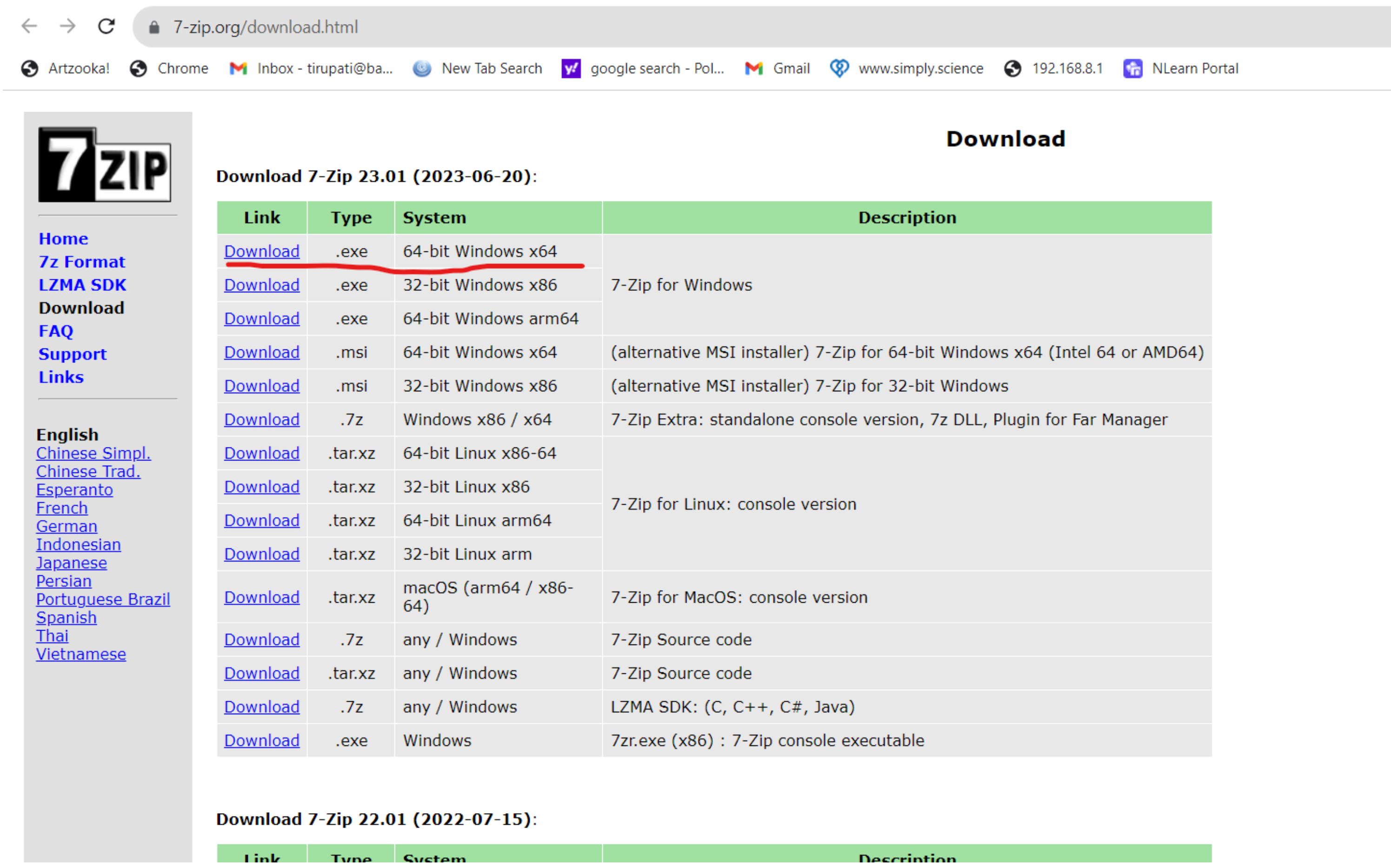
step-4: you will be redirected to the new page click ffmpeg-git-full.7z then you can the file is downloaded in your directory



Downloads > ffmpeg-2023-11-05-git-44a0148fad-full_build >

Name	Type	Size	Date modified
ffmpeg-2023-11-05-git-44a0148fa...	File folder		06-11-2023 16:12

Step-5: download 7-Zip: https://7-zip.org/download.html



Download

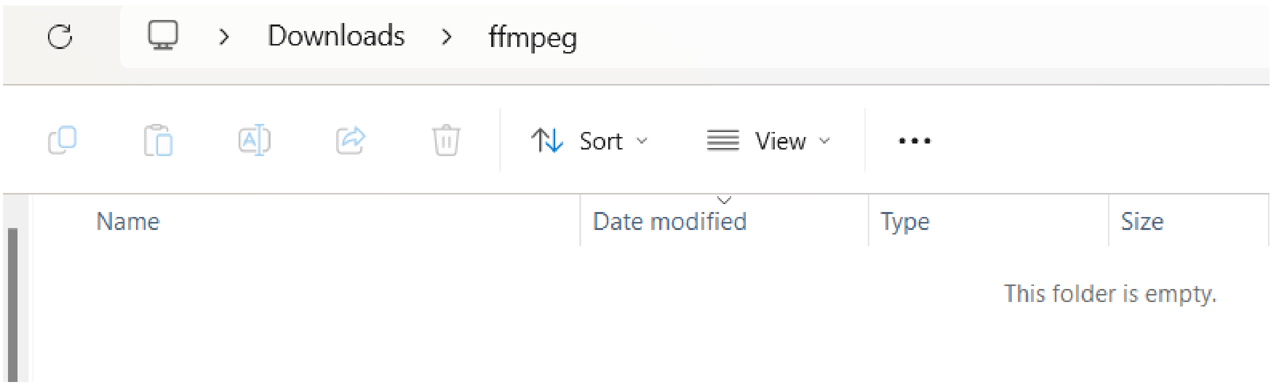
Download 7-Zip 23.01 (2023-06-20):

Link	Type	System	Description
Download	.exe	64-bit Windows x64	7-Zip for Windows
Download	.exe	32-bit Windows x86	
Download	.exe	64-bit Windows arm64	
Download	.msi	64-bit Windows x64	(alternative MSI installer) 7-Zip for 64-bit Windows x64 (Intel 64 or AMD64)
Download	.msi	32-bit Windows x86	(alternative MSI installer) 7-Zip for 32-bit Windows
Download	.7z	Windows x86 / x64	7-Zip Extra: standalone console version, 7z DLL, Plugin for Far Manager
Download	.tar.xz	64-bit Linux x86-64	7-Zip for Linux: console version
Download	.tar.xz	32-bit Linux x86	
Download	.tar.xz	64-bit Linux arm64	
Download	.tar.xz	32-bit Linux arm	
Download	.tar.xz	macOS (arm64 / x86-64)	7-Zip for MacOS: console version
Download	.7z	any / Windows	7-Zip Source code
Download	.tar.xz	any / Windows	7-Zip Source code
Download	.7z	any / Windows	LZMA SDK: (C, C++, C#, Java)
Download	.exe	Windows	7zr.exe (x86) : 7-Zip console executable

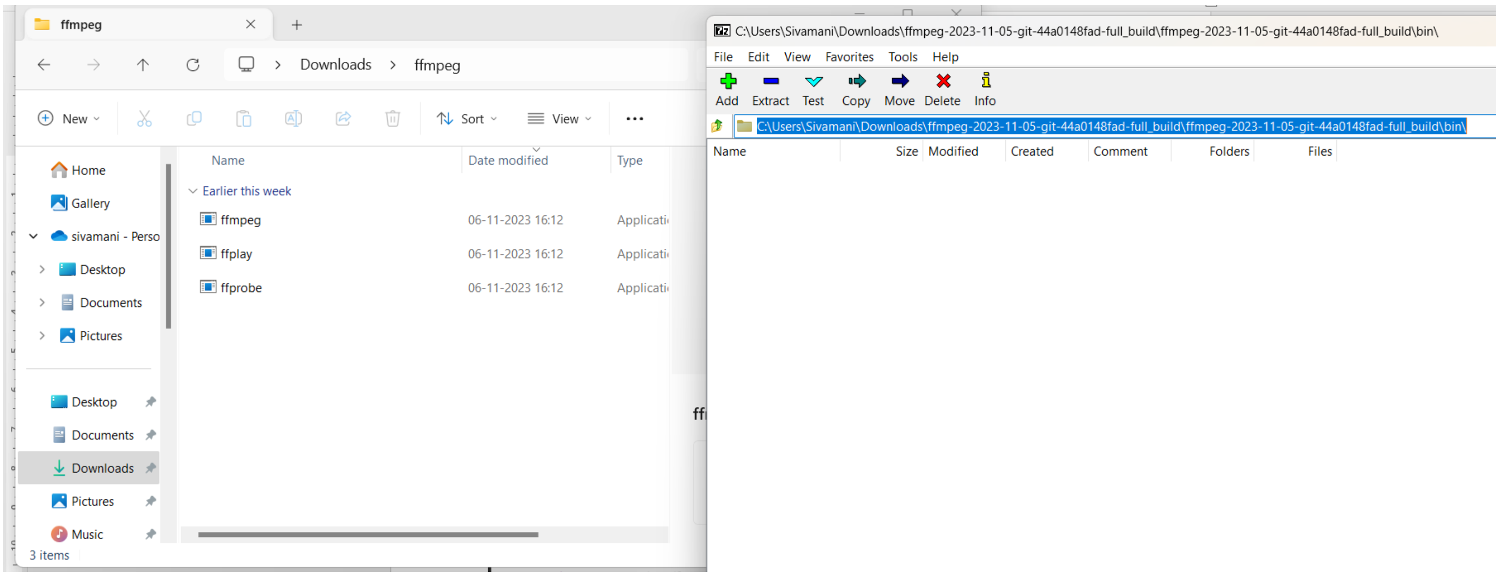
Download 7-Zip 22.01 (2022-07-15):

Link	Type	System	Description

Step-6: After installing create a new folder new with name ffmpeg



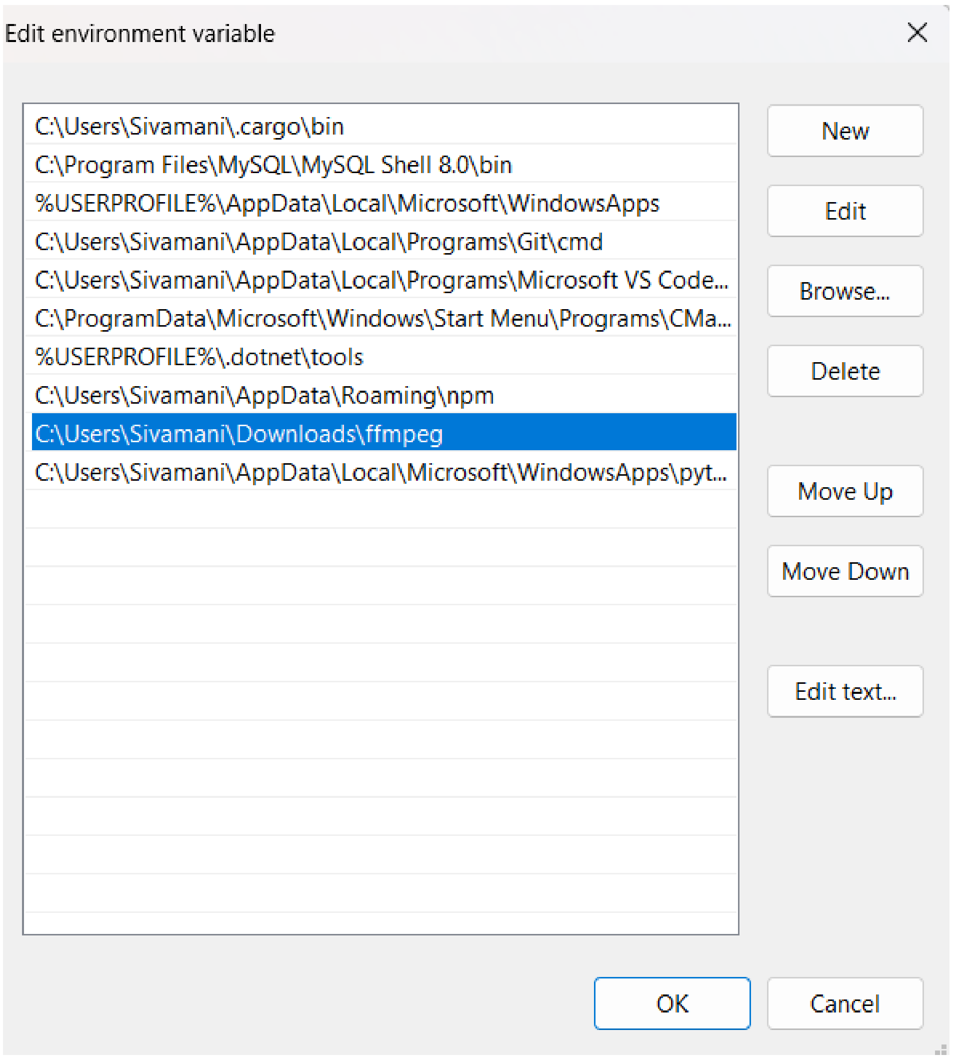
Step-7: Open the 7-zip and go this dir C:\Users\Sivamani\Downloads\ffmpeg-2023-11-05-git-44a0148fad-full_build\ffmpeg-2023-11-05-git-44a0148fad-full_build\bin\ copy and paste the files into the newly create folder



Name	Date modified	Type
ffmpeg	06-11-2023 16:12	Applicati
ffplay	06-11-2023 16:12	Applicati
ffprobe	06-11-2023 16:12	Applicati

C:\Users\Sivamani\Downloads\ffmpeg-2023-11-05-git-44a0148fad-full_build\ffmpeg-2023-11-05-git-44a0148fad-full_build\bin\

Step-8: Add the path of the newly create folder to the environment variables



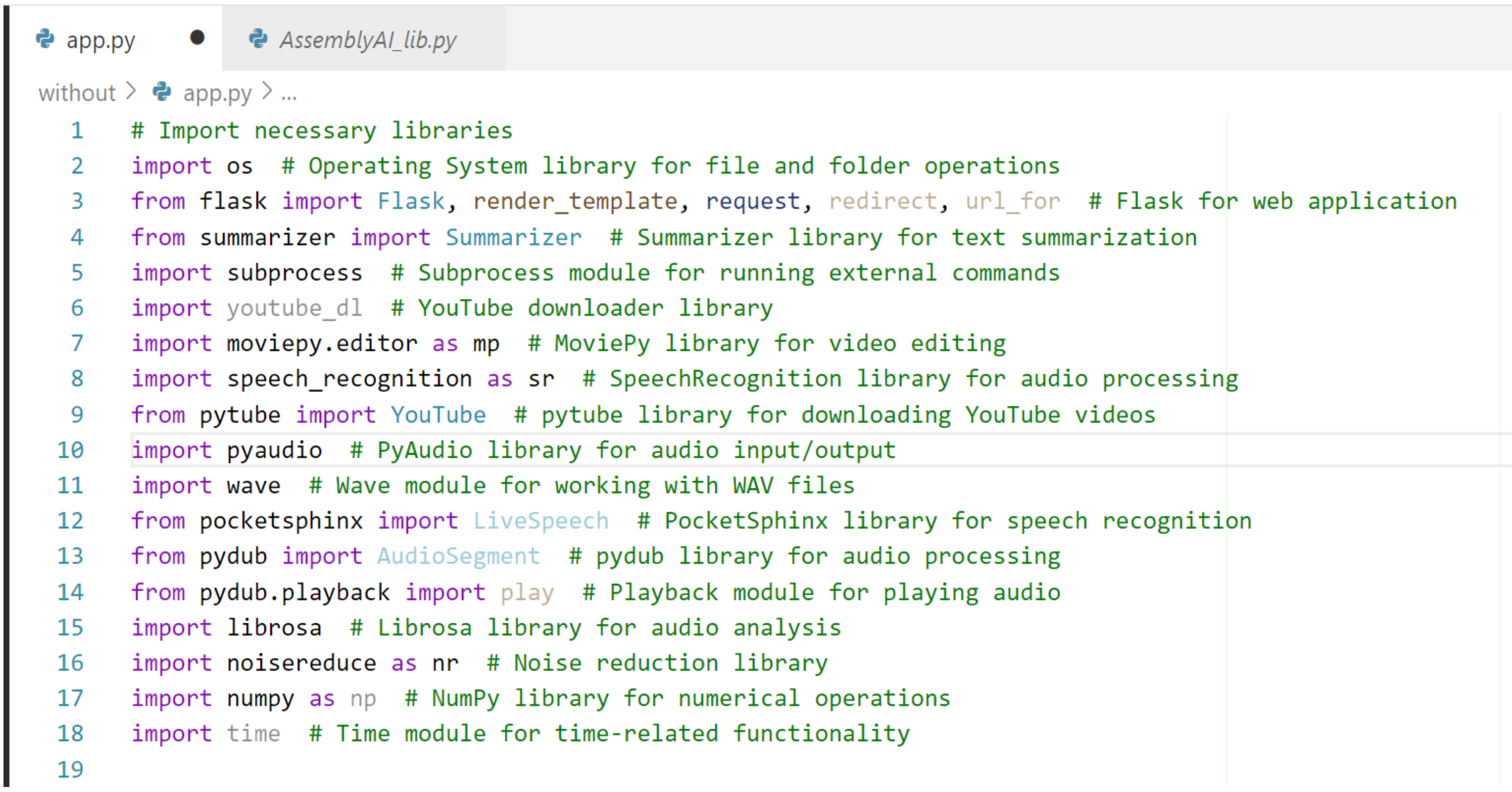
```
C:\Users\Sivamani\.cargo\bin
C:\Program Files\MySQL\MySQL Shell 8.0\bin
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Sivamani\AppData\Local\Programs\Git\cmd
C:\Users\Sivamani\AppData\Local\Programs\Microsoft VS Code...
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\CMa...
%USERPROFILE%\.dotnet\tools
C:\Users\Sivamani\AppData\Roaming\npm
C:\Users\Sivamani\Downloads\ffmpeg
C:\Users\Sivamani\AppData\Local\Microsoft\WindowsApps\pyt...
```

To test ffmpeg is running you type ffmpeg -version in cmd

+

4. IMPLEMENTATION AND RESULTS

Web Application (app.py):



```python
# Import necessary libraries
import os  # Operating System library for file and folder operations
from flask import Flask, render_template, request, redirect, url_for  # Flask for web application
from summarizer import Summarizer  # Summarizer library for text summarization
import subprocess  # Subprocess module for running external commands
import youtube_dl  # YouTube downloader library
import moviepy.editor as mp  # MoviePy library for video editing
import speech_recognition as sr  # SpeechRecognition library for audio processing
from pytube import YouTube  # pytube library for downloading YouTube videos
import pyaudio  # PyAudio library for audio input/output
import wave  # Wave module for working with WAV files
from pocketsphinx import LiveSpeech  # PocketSphinx library for speech recognition
from pydub import AudioSegment  # pydub library for audio processing
from pydub.playback import play  # Playback module for playing audio
import librosa  # Librosa library for audio analysis
import noisereduce as nr  # Noise reduction library
import numpy as np  # NumPy library for numerical operations
import time  # Time module for time-related functionality
```

1. os (Operating System):

- File Operations: Manages files and directories, allowing the creation, deletion, and checking of file existence.
- Path Handling: Facilitates constructing and manipulating file paths for cross-platform compatibility.
- Directory Creation: Creates directories for organizing files and ensuring a structured project layout.

2. Flask:

- Web Framework: Powers the web application, defining routes for different functionalities.
- HTTP Handling: Manages HTTP requests and responses for communication between the server and client.
- Template Rendering: Integrates HTML templates to render dynamic content on web pages.
- Web Development: Simplifies the creation of web applications with a flexible and modular design.

3. Summarizer:

- Natural Language Processing: Implements advanced NLP techniques for extracting key information from text.
- Abstractive Summarization: Generates concise summaries that capture the essence of the input text.
- Pretrained Models: Utilizes pre-trained models to understand and summarize content effectively.
- Text Complexity Handling: Adapts to various text structures, making it suitable for diverse content types.

4. subprocess:

- External Command Execution: Interacts with the system shell to execute external commands or programs.
- Process Management: Enables the running of ffmpeg commands for audio and video processing.
- Error Handling: Captures and handles errors or output generated during external command execution.
- Shell Commands: Executes system commands to perform tasks outside the Python environment.

5. youtube_dl:

- YouTube Video Download: Downloads videos from YouTube based on user-provided URLs.
- Video Format Options: Allows specifying video and audio format preferences during the download.
- Metadata Retrieval: Extracts metadata information about the video, such as title and duration.
- Download Progress Tracking: Provides options for tracking the progress of video downloads.

6. moviepy.editor:

- Video Editing: Edits videos by extracting audio, cutting clips, and performing various transformations.
- Audio Extraction: Extracts audio content from video files for further processing.
- Format Conversion: Converts video files to different formats and handles various video codecs.
- Timeline-Based Editing: Utilizes a timeline model for intuitive video editing operations.

7. speech_recognition:

- Speech Recognition: Transcribes spoken words from audio files or microphone input.
- Multiple Recognition Engines: Supports various speech recognition engines, including Google Web Speech API.
- Audio Source Handling: Manages different audio sources, such as files, microphones, or audio streams.
- Language Specification: Allows specifying the language of the spoken content for accurate transcription.

8. pytube:

- YouTube Video Download: Downloads YouTube videos and audio streams for offline access.
- Quality Control: Provides options to choose video quality, resolution, and audio format.
- Playlist Handling: Supports downloading entire playlists or individual videos from YouTube.
- Video Information: Retrieves details about YouTube videos, such as title, author, and duration.

9. pyaudio:

- Audio Recording: Captures audio from the microphone or other input sources in real-time.
- Sampling Parameters: Allows customization of audio sampling parameters, including sample rate and format.
- Stream Management: Manages audio streams for recording and playback.
- Cross-Platform Support: Provides a consistent interface for audio input across different operating systems.

10. wave:

- WAV File Handling: Reads and writes WAV audio files, a widely used format for uncompressed audio.
- Audio Metadata: Manages metadata information such as number of channels, sample width, and frame rate.

- Frame-Level Access: Allows access to individual audio frames for processing and analysis.
- Compatibility: Ensures compatibility with audio processing libraries that support the WAV format.

11. pocketsphinx:

- Speech Recognition Engine: Implements a lightweight and efficient speech recognition engine.
- Offline Speech Recognition: Enables speech recognition without the need for an internet connection.
- Continuous Speech Recognition: Supports continuous listening for processing longer speech segments.
- Customization: Allows customization of language models for improved recognition accuracy.

12. pydub:

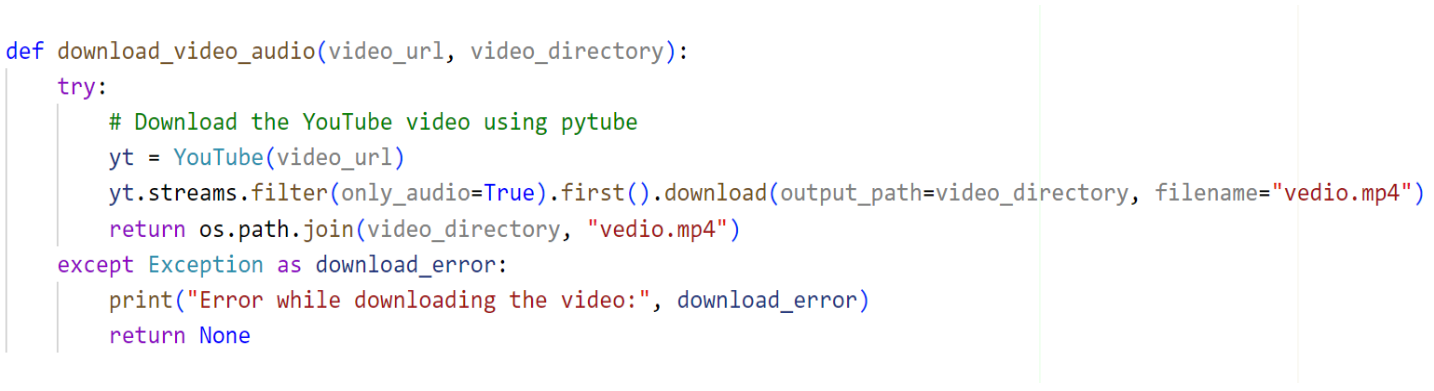
- Audio Manipulation: Provides a high-level interface for manipulating audio data.
- Format Conversion: Converts between different audio formats, allowing seamless integration with other libraries.
- Audio Playback: Facilitates playing audio directly within the Python environment.
- Effects and Filters: Applies various effects and filters to modify audio characteristics.

13. librosa:

- Audio Analysis: Analyzes audio data for tasks such as feature extraction and visualization.
- Noise Reduction: Applies algorithms for reducing noise in audio signals.
- Time-Frequency Representation: Generates spectrograms and other time-frequency representations.
- Compatibility: Integrates seamlessly with other audio processing libraries.

14. noisereduce:

- Noise Reduction: Implements algorithms for reducing noise in audio signals.
- Adaptive Filtering: Adapts to varying noise conditions for effective noise reduction.
- Signal Processing: Utilizes signal processing techniques to enhance audio quality.
- Real-Time Applications: Applicable to real-time audio processing scenarios.



```python
def download_video_audio(video_url, video_directory):
    try:
        # Download the YouTube video using pytube
        yt = YouTube(video_url)
        yt.streams.filter(only_audio=True).first().download(output_path=video_directory, filename="vedio.mp4")
        return os.path.join(video_directory, "vedio.mp4")
    except Exception as download_error:
        print("Error while downloading the video:", download_error)
        return None
```

This function downloads the audio from a YouTube video using the pytube library.

1. Input:

- video_url: The URL of the YouTube video.
- video_directory: The directory where the downloaded video will be saved.

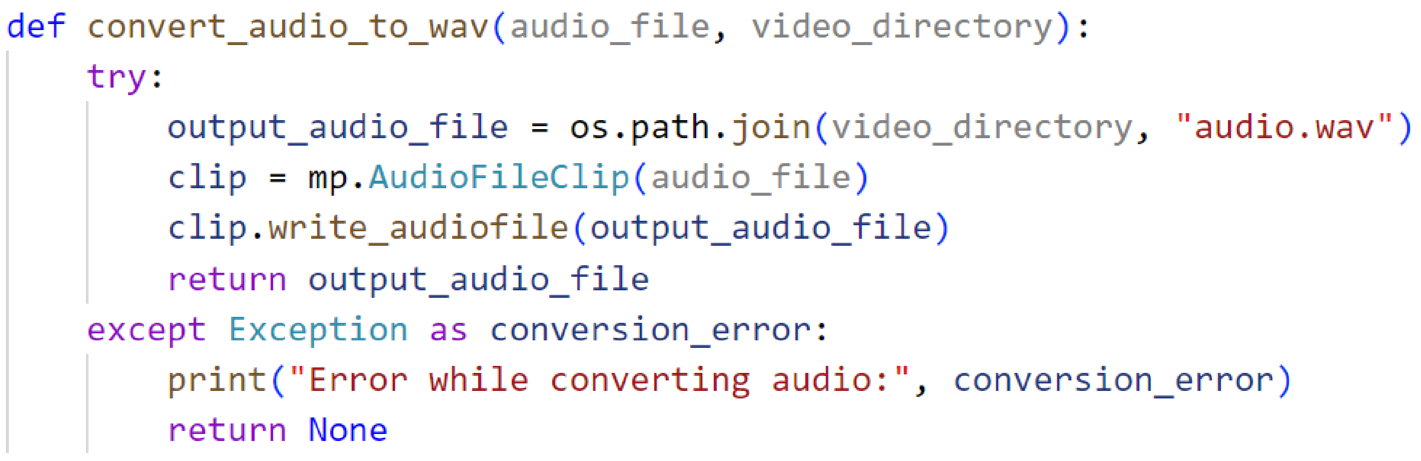
2. Process:

- It creates a YouTube object from the provided video URL using the pytube library.
- Filters the available streams to include only the audio streams (only_audio=True).
- Chooses the first audio stream (assuming it's the best quality) and downloads it.
- The downloaded audio is saved as "vedio.mp4" in the specified video_directory.

3. Output:

- Returns the file path of the downloaded audio ("vedio.mp4") if successful.
- If an error occurs during the download, it prints an error message and returns None.

In essence, this function encapsulates the process of fetching the audio content from a YouTube video, making it ready for further processing in the audio summarization pipeline.



```python
def convert_audio_to_wav(audio_file, video_directory):
    try:
        output_audio_file = os.path.join(video_directory, "audio.wav")
        clip = mp.AudioFileClip(audio_file)
        clip.write_audiofile(output_audio_file)
        return output_audio_file
    except Exception as conversion_error:
        print("Error while converting audio:", conversion_error)
        return None
```

This function converts an audio file to the WAV format using the moviepy library.

1. Input:

- audio_file: The path to the input audio file that needs to be converted.
- video_directory: The directory where the converted audio file will be saved.

2. Process:

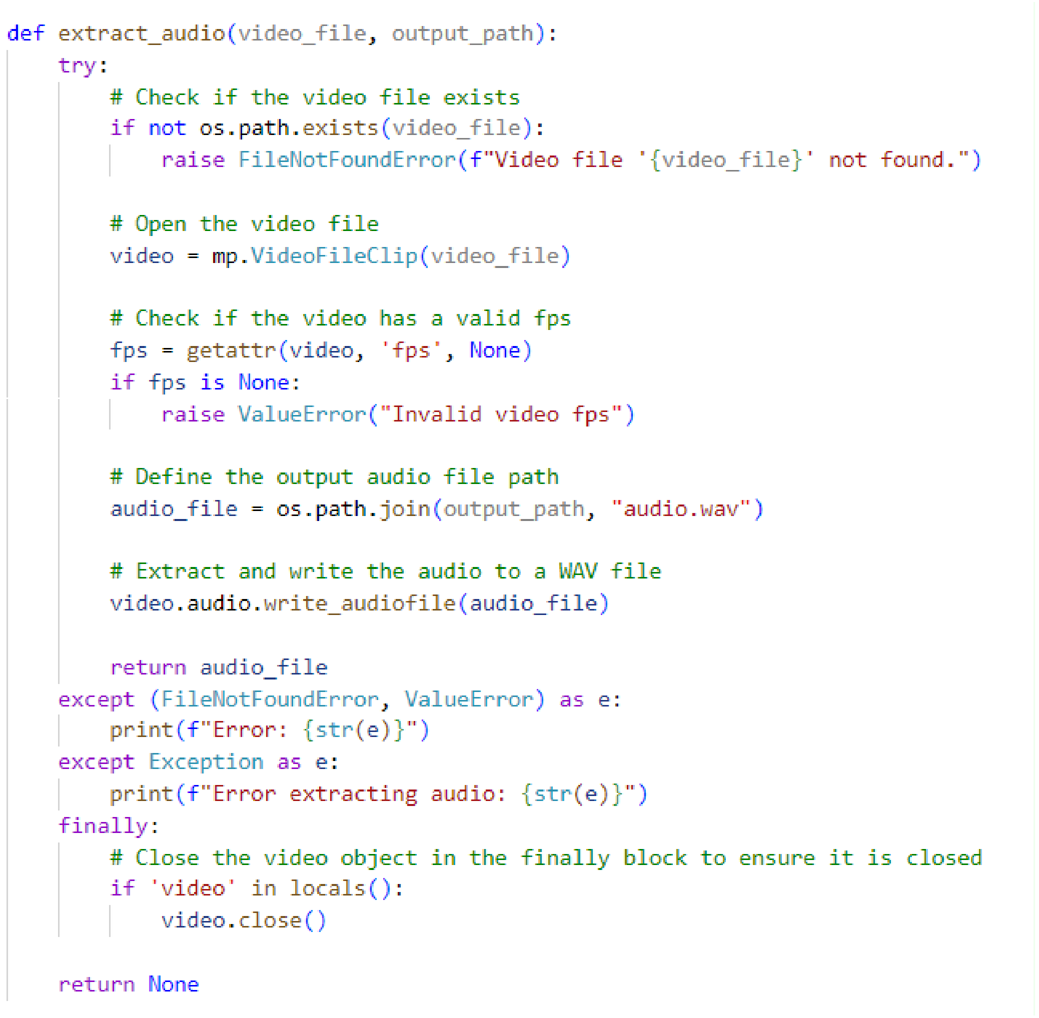
- It defines the output path for the converted audio file as "audio.wav" in the specified video_directory.
- Utilizes moviepy's AudioFileClip to create an audio clip from the input audio file.

- Writes the audio clip to the WAV format file specified in the output path.

3. Output:

- Returns the file path of the converted audio ("audio.wav") if successful.
- If an error occurs during the conversion, it prints an error message and returns None.

This function is part of the pipeline to ensure uniformity in the audio file format for further processing, specifically preparing the audio content for transcription and summarization.



```python
def extract_audio(video_file, output_path):
    try:
        # Check if the video file exists
        if not os.path.exists(video_file):
            raise FileNotFoundError(f"Video file '{video_file}' not found.")

        # Open the video file
        video = mp.VideoFileClip(video_file)

        # Check if the video has a valid fps
        fps = getattr(video, 'fps', None)
        if fps is None:
            raise ValueError("Invalid video fps")

        # Define the output audio file path
        audio_file = os.path.join(output_path, "audio.wav")

        # Extract and write the audio to a WAV file
        video.audio.write_audiofile(audio_file)

        return audio_file
    except (FileNotFoundError, ValueError) as e:
        print(f"Error: {str(e)}")
    except Exception as e:
        print(f"Error extracting audio: {str(e)}")
    finally:
        # Close the video object in the finally block to ensure it is closed
        if 'video' in locals():
            video.close()

    return None
```

21

This function extracts the audio from a video file using the moviepy library.

1. Input:

- video_file: The path to the input video file from which audio will be extracted.
- output_path: The directory where the extracted audio file will be saved.

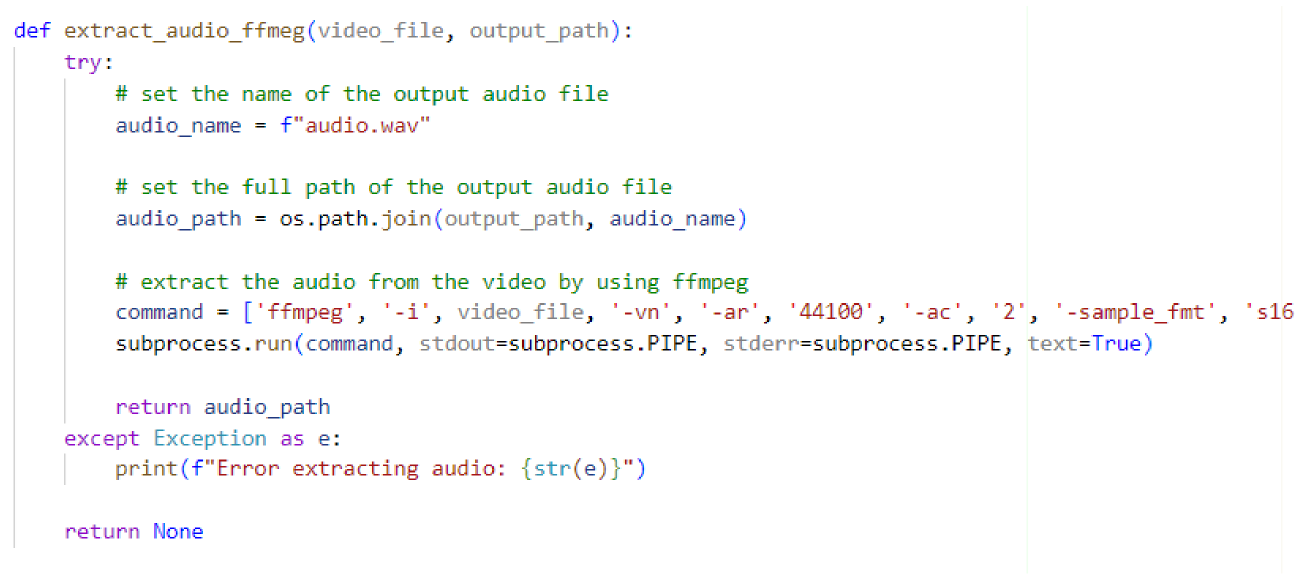
2. Process:

- Checks if the input video file exists; if not, raises a FileNotFoundError.
- Opens the video file using moviepy and checks if it has a valid frames per second (fps) attribute. If not, raises a ValueError.
- Defines the output path for the extracted audio file as "audio.wav" in the specified output_path.
- Extracts and writes the audio from the video to a WAV file using moviepy's audio writing functionality.

3. Output:

- Returns the file path of the extracted audio ("audio.wav") if successful.
- If an error occurs during the extraction, it prints an error message and returns None.

This function is a crucial step in the audio processing pipeline, providing the audio content needed for transcription and subsequent summarization. It ensures that the audio is extracted uniformly from various video sources for consistent further processing.



```python
def extract_audio_ffmeg(video_file, output_path):
    try:
        # set the name of the output audio file
        audio_name = f"audio.wav"

        # set the full path of the output audio file
        audio_path = os.path.join(output_path, audio_name)

        # extract the audio from the video by using ffmpeg
        command = ['ffmpeg', '-i', video_file, '-vn', '-ar', '44100', '-ac', '2', '-sample_fmt', 's16
        subprocess.run(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

        return audio_path
    except Exception as e:
        print(f"Error extracting audio: {str(e)}")

    return None
```

This function extracts audio from a video file using the FFmpeg command-line tool.

1. Input:

- video_file: The path to the input video file from which audio will be extracted.
- output_path: The directory where the extracted audio file will be saved.

2. Process:

- Defines the name of the output audio file as "audio.wav."
- Constructs the full path of the output audio file by joining the specified output_path and the predefined audio name.
- Uses the FFmpeg command-line tool to execute a command for audio extraction. The command includes:
  - ➢ -i: Input file (the video file).
  - ➢ -vn: Disable video recording.
  - ➢ -ar 44100: Set audio sample rate to 44100 Hz.
  - ➢ -ac 2: Set audio channels to stereo.
  - ➢ -sample_fmt s16: Set the audio sample format to 16-bit.
- The output audio file path.

3. Output:

- Returns the file path of the extracted audio ("audio.wav") if the extraction is successful.
- If an error occurs during the extraction, it prints an error message and returns None.

This function provides an alternative method for extracting audio, utilizing FFmpeg's capabilities. It is a robust approach to handling various video formats and extracting high-quality audio for further processing.

```python
def transcribe_audio(audio_file):
    if not os.path.exists(audio_file):
        print(f"Audio file '{audio_file}' does not exist.")
        return None

    # Check the format of the audio file and convert it to WAV if needed
    audio_format = audio_file.split('.')[-1]
    if audio_format != 'wav':
        try:
            # Convert the audio file to WAV format
            output_audio_file = audio_file.replace(audio_format, 'wav')
            audio = mp.AudioFileClip(audio_file)
            audio.write_audiofile(output_audio_file)  # Use 'pcm_s16le' codec for WAV format
            audio.close()
            audio_file = output_audio_file
        except Exception as e:
            print(f"Error converting audio to WAV format: {str(e)}")
            return None

    try:
        recognizer = sr.Recognizer()
        with sr.AudioFile(audio_file) as source:
            audio = recognizer.record(source)
        return recognizer.recognize_google(audio, language="en-US")
    except Exception as e:
        print(f"Error transcribing audio: {str(e)}")
        return None
```

This function transcribes the content of an audio file into text using Google's speech recognition service.

1. Input:

- audio_file: The path to the input audio file that needs to be transcribed.

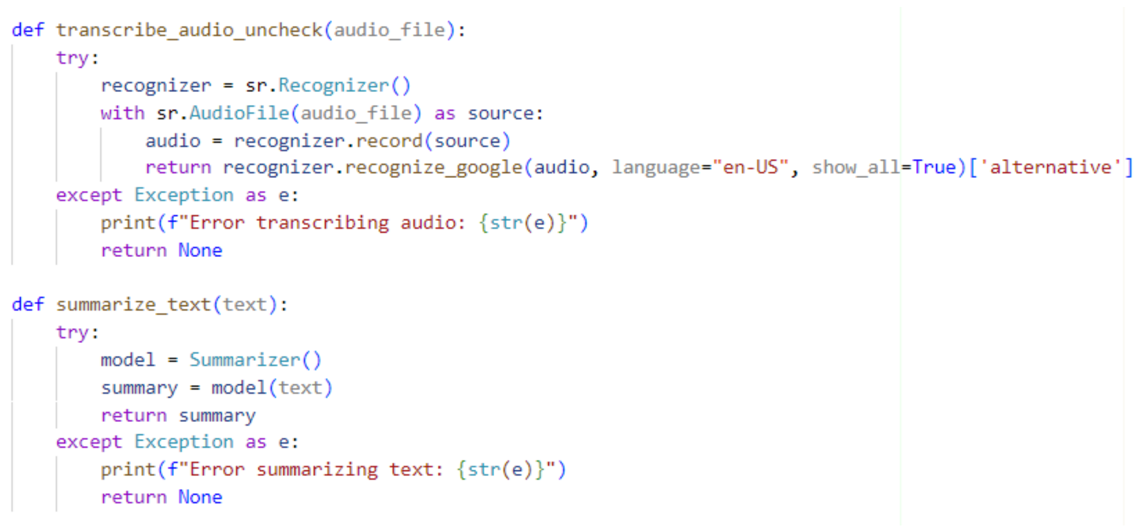
2. Process:

- Checks if the specified audio file exists. If not, it prints an error message and returns None.
- Checks the format of the audio file, and if it's not in WAV format, it attempts to convert it.
- Creates an output audio file with a ".wav" extension using MoviePy library.
- Uses the SpeechRecognition library (recognizer) to transcribe the audio.
- Reads the audio file using sr.AudioFile.
- Records the audio from the file.
- Sends the recorded audio to Google's speech recognition service using recognizer.recognize_google.
- Specifies the language as "en-US" (English, United States).

3. Output:

- Returns the transcribed text if the process is successful.
- If there are errors during the transcription, it prints an error message and returns None.

This function ensures that the input audio file is in the required WAV format for accurate transcription. It leverages the SpeechRecognition library for interfacing with Google's speech recognition service, providing a straightforward way to convert spoken words into text.



```
def transcribe_audio_uncheck(audio_file):
    try:
        recognizer = sr.Recognizer()
        with sr.AudioFile(audio_file) as source:
            audio = recognizer.record(source)
            return recognizer.recognize_google(audio, language="en-US", show_all=True)['alternative'][0]['transcript
    except Exception as e:
        print(f"Error transcribing audio: {str(e)}")
        return None

def summarize_text(text):
    try:
        model = Summarizer()
        summary = model(text)
        return summary
    except Exception as e:
        print(f"Error summarizing text: {str(e)}")
        return None
```

transcribe_audio_uncheck Function

This function performs an unchecked transcription of the content of an audio file into text using Google's speech recognition service. It returns the transcribed text without checking for errors.

1. Input:

- audio_file: The path to the input audio file that needs to be transcribed.

2. Process:

- Uses the SpeechRecognition library (recognizer) to transcribe the audio.
- Reads the audio file using sr.AudioFile.
- Records the audio from the file.
- Sends the recorded audio to Google's speech recognition service using recognizer.recognize_google.
- Specifies the language as "en-US" (English, United States).
- Uses show_all=True to retrieve alternative transcriptions.
- Extracts the first alternative's transcript, converts it to lowercase, and removes leading/trailing whitespaces.

3. Output:

- Returns the transcribed text if the process is successful.
- If there are errors during the transcription, it prints an error message and returns None.

summarize_text Function

This function generates a summary of the provided text using the Summarizer model.

1. Input:

- text: The input text that needs to be summarized.

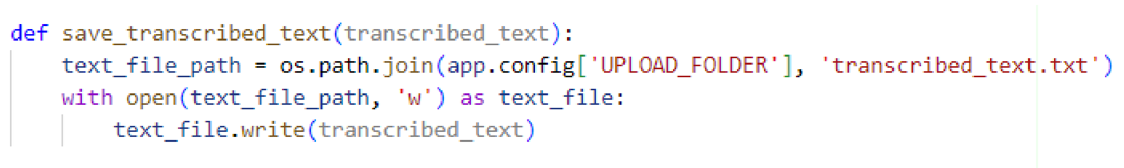
2. Process:

- Uses the Summarizer library (model) to generate a summary of the input text.

3. Output:

- Returns the generated summary if the process is successful.
- If there are errors during the summarization, it prints an error message and returns None.

These functions contribute to the core functionality of summarizing spoken words into concise text summaries. The first function focuses on transcription, while the second one handles the summarization process.



```python
def save_transcribed_text(transcribed_text):
    text_file_path = os.path.join(app.config['UPLOAD_FOLDER'], 'transcribed_text.txt')
    with open(text_file_path, 'w') as text_file:
        text_file.write(transcribed_text)
```

This function saves the transcribed text to a text file.

1. Input:

- transcribed_text: The text that needs to be saved.

2. Process:

- Constructs the full path to the text file using the app.config['UPLOAD_FOLDER'] (upload folder defined in the Flask app).
- Opens the text file in write mode ('w').
- Writes the transcribed text to the file.

3. Output:

- Saves the transcribed text to a text file with the name 'transcribed_text.txt' in the specified upload folder.

This function is crucial for persisting the transcribed text, providing a record of the content that has been transcribed during the application's operation.

```python
# Summarize microphone audio
def summarize_microphone_audio():
    try:
        # Record audio from the microphone
        recognizer = sr.Recognizer()
        with sr.Microphone() as source:
            print("Say something...")
            audio = recognizer.listen(source)

        # Transcribe the recorded audio
        audio_text = recognizer.recognize_google(audio)
        if audio_text:
            summary = summarize_text(audio_text)
            if summary:
                return summary
            else:
                return "Summary generation failed."
        else:
            return "No speech detected."
    except sr.RequestError as e:
        return f"Could not request results: {str(e)}"
    except sr.UnknownValueError:
        return "No speech detected."
```

This function captures audio from the microphone, transcribes the spoken words using Google's Speech Recognition, and then generates a summary of the transcribed text using the summarize_text function.

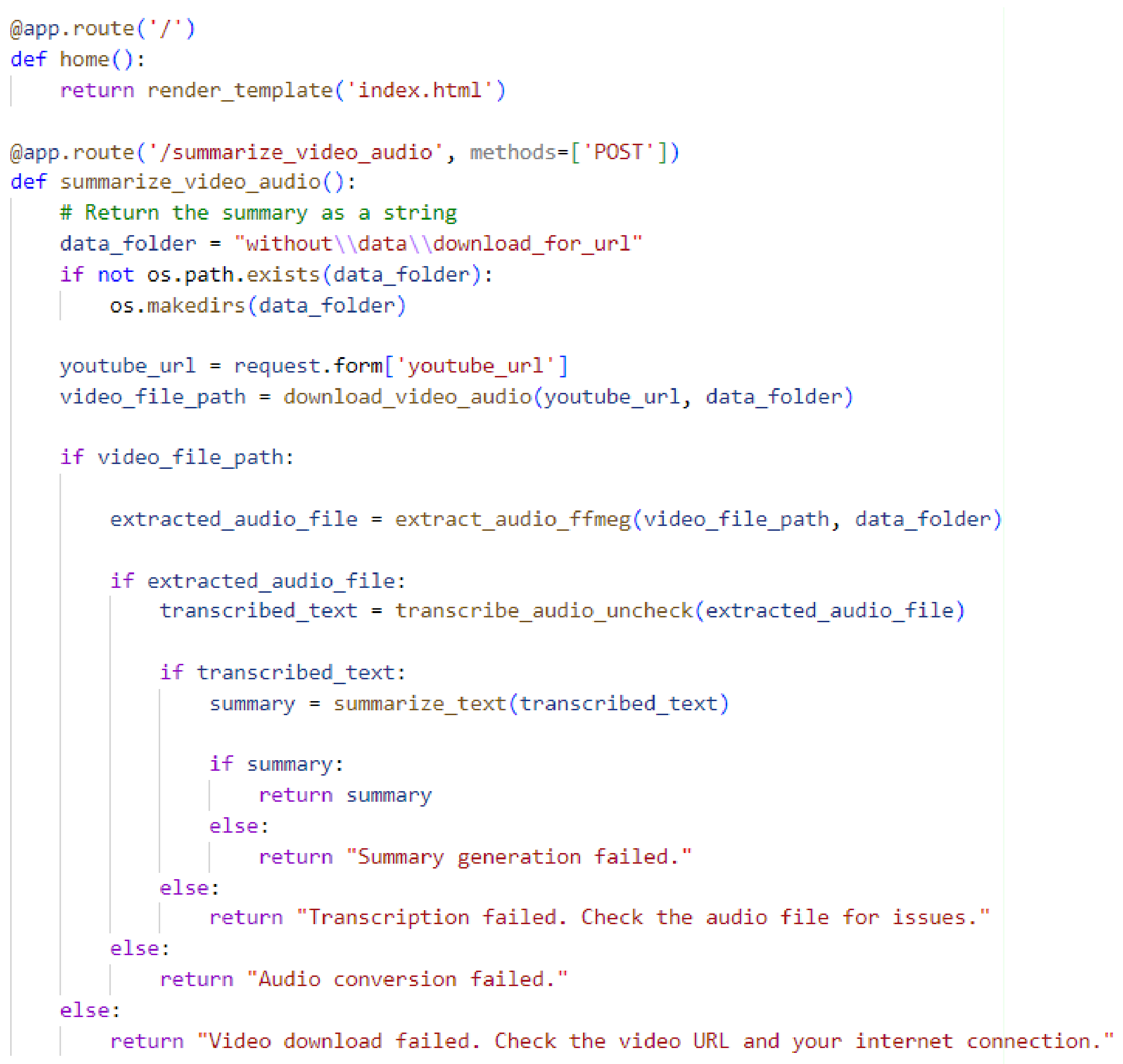
1. Process:

- Initializes a Recognizer object from the speech_recognition library.
- Opens the microphone as a source for audio input using the with sr.Microphone() as source block.
- Prints a prompt to the user to say something.
- Listens to the audio input and records it.
- Uses Google's Speech Recognition (recognizer.recognize_google(audio)) to convert the recorded audio to text (audio_text).
- If audio text is detected:
- Generates a summary of the transcribed text using the summarize_text function.
- Returns the summary if successful.
- If summary generation fails, returns an appropriate message.
- If no speech is detected, returns a corresponding message.

2. Output:

- Summary of the transcribed audio if successful.
- Messages indicating failure in case of no speech detected or summary generation failure.

This function provides real-time summarization of spoken words from the microphone, making the application interactive and user-friendly.



```python
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/summarize_video_audio', methods=['POST'])
def summarize_video_audio():
    # Return the summary as a string
    data_folder = "without\\data\\download_for_url"
    if not os.path.exists(data_folder):
        os.makedirs(data_folder)

    youtube_url = request.form['youtube_url']
    video_file_path = download_video_audio(youtube_url, data_folder)

    if video_file_path:

        extracted_audio_file = extract_audio_ffmeg(video_file_path, data_folder)

        if extracted_audio_file:
            transcribed_text = transcribe_audio_uncheck(extracted_audio_file)

            if transcribed_text:
                summary = summarize_text(transcribed_text)

                if summary:
                    return summary
                else:
                    return "Summary generation failed."
            else:
                return "Transcription failed. Check the audio file for issues."
        else:
            return "Audio conversion failed."
    else:
        return "Video download failed. Check the video URL and your internet connection."
```

Flask Routes and summarize_video_audio:

1. Route:

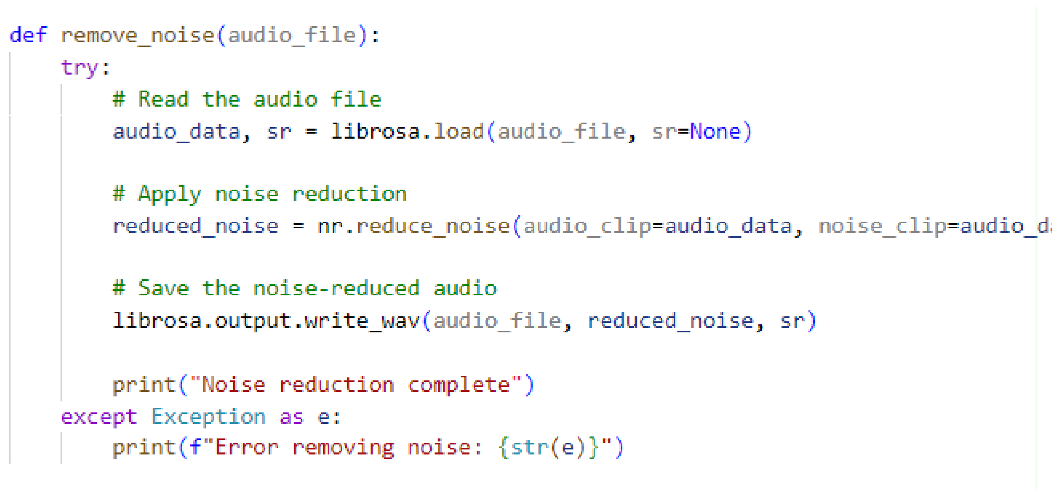
- Maps to the home page of the application.
- Renders the HTML template located at 'index.html' using the render_template function.
- Provides the user with the main interface for the Audio Summarizer.

2. summarize_video_audio Route:

- Handles POST requests triggered when the user submits a YouTube video URL for summarization.
- Creates a directory (data_folder) for storing downloaded content if it doesn't exist.
- Retrieves the YouTube URL from the submitted form data.
- Calls the download_video_audio function to download the YouTube video as audio to the specified directory.
- Calls the extract_audio_ffmeg function to extract audio from the downloaded video using ffmpeg.
- Transcribes the extracted audio using the transcribe_audio_uncheck function.
- Generates a summary of the transcribed text using the summarize_text function.
- Returns the generated summary if successful.

- Provides appropriate error messages if any step fails (e.g., video download failure, audio conversion failure, or transcription failure).

These routes establish the core functionality of the Audio Summarizer, allowing users to interact with the application through the web interface and submit YouTube video URLs for summarization.



```python
def remove_noise(audio_file):
    try:
        # Read the audio file
        audio_data, sr = librosa.load(audio_file, sr=None)

        # Apply noise reduction
        reduced_noise = nr.reduce_noise(audio_clip=audio_data, noise_clip=audio_data)

        # Save the noise-reduced audio
        librosa.output.write_wav(audio_file, reduced_noise, sr)

        print("Noise reduction complete")
    except Exception as e:
        print(f"Error removing noise: {str(e)}")
```

Noise Reduction Function

- Function Name: remove_noise

Steps:

1. Read Audio File:

- Uses the librosa.load function to read the audio file.
- Obtains the audio data and the sample rate (sr) from the loaded audio file.

2. Apply Noise Reduction:

- Utilizes the nr.reduce_noise function from the noisereduce library.
- Applies noise reduction to the audio clip (audio_data) using the same clip as the noise reference (noise_clip = audio_data).

3. Save Noise-Reduced Audio:

- Writes the noise-reduced audio back to the original audio file using librosa.output.write_wav.
- The sr parameter ensures that the saved audio has the same sample rate as the original.

4. Prints Status:

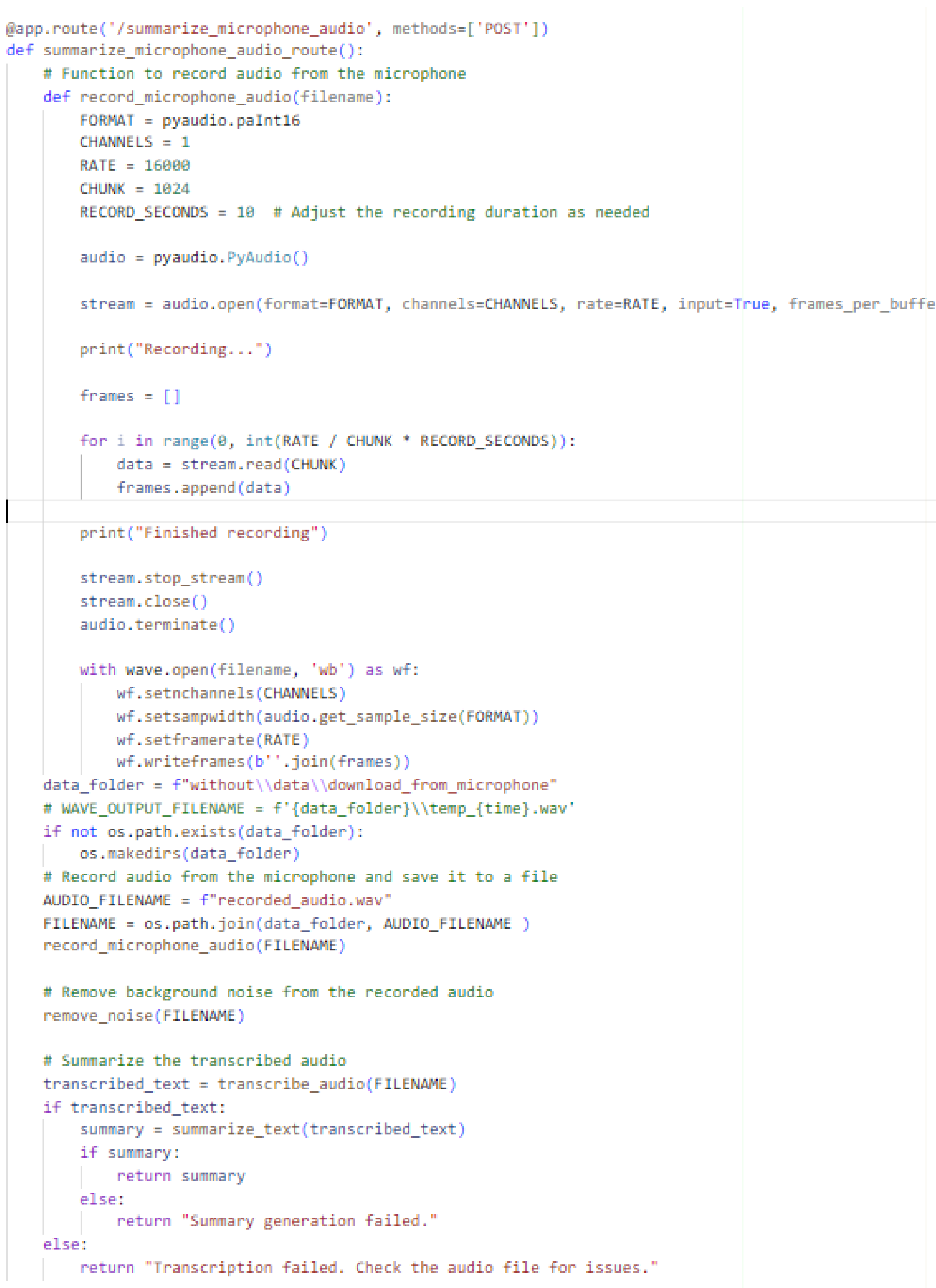
- Prints "Noise reduction complete" if the process is successful.

5. Error Handling:

- Catches and handles exceptions, printing an error message if any issues occur during the noise reduction process.

Note:

- The noisereduce library is used for noise reduction, taking advantage of its functionality to enhance the quality of the audio by reducing unwanted background noise.



```
@app.route('/summarize_microphone_audio', methods=['POST'])
def summarize_microphone_audio_route():
    # Function to record audio from the microphone
    def record_microphone_audio(filename):
        FORMAT = pyaudio.paInt16
        CHANNELS = 1
        RATE = 16000
        CHUNK = 1024
        RECORD_SECONDS = 10  # Adjust the recording duration as needed

        audio = pyaudio.PyAudio()

        stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE, input=True, frames_per_buffe

        print("Recording...")

        frames = []

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

        print("Finished recording")

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

        with wave.open(filename, 'wb') as wf:
            wf.setnchannels(CHANNELS)
            wf.setsampwidth(audio.get_sample_size(FORMAT))
            wf.setframerate(RATE)
            wf.writeframes(b''.join(frames))
    data_folder = f"without\\data\\download_from_microphone"
    # WAVE_OUTPUT_FILENAME = f'{data_folder}\\temp_{time}.wav'
    if not os.path.exists(data_folder):
        os.makedirs(data_folder)
    # Record audio from the microphone and save it to a file
    AUDIO_FILENAME = f"recorded_audio.wav"
    FILENAME = os.path.join(data_folder, AUDIO_FILENAME )
    record_microphone_audio(FILENAME)

    # Remove background noise from the recorded audio
    remove_noise(FILENAME)

    # Summarize the transcribed audio
    transcribed_text = transcribe_audio(FILENAME)
    if transcribed_text:
        summary = summarize_text(transcribed_text)
        if summary:
            return summary
        else:
            return "Summary generation failed."
    else:
        return "Transcription failed. Check the audio file for issues."
```

The **summarize_microphone_audio_route** function is a comprehensive process designed to capture, process, and summarize audio recorded from a microphone. Here's a breakdown of its key functionalities:

1. Recording Parameters:

- Defines parameters such as audio format, channels, sampling rate, chunk size, and recording duration using the pyaudio library.

2. Record Microphone Audio:

- Utilizes pyaudio to capture audio from the microphone.
- Records audio for a specified duration, saving it as a WAV file.

3. Create Data Folder:

- Checks for the existence of a designated data folder for microphone recordings.
- Creates the folder using os.makedirs if it doesn't exist.

4. Set Filename and Path:

- Establishes the filename and full path for the recorded audio file within the designated data folder.

5. Record Audio:

- Calls the record_microphone_audio function, responsible for managing the audio recording process.

6. Remove Background Noise:

- Applies the remove_noise function from the librosa and noisereduce libraries to enhance audio quality by reducing background noise.

7. Transcribe Audio:

- Uses the transcribe_audio function, powered by the speech_recognition library, to convert the recorded audio into textual content.

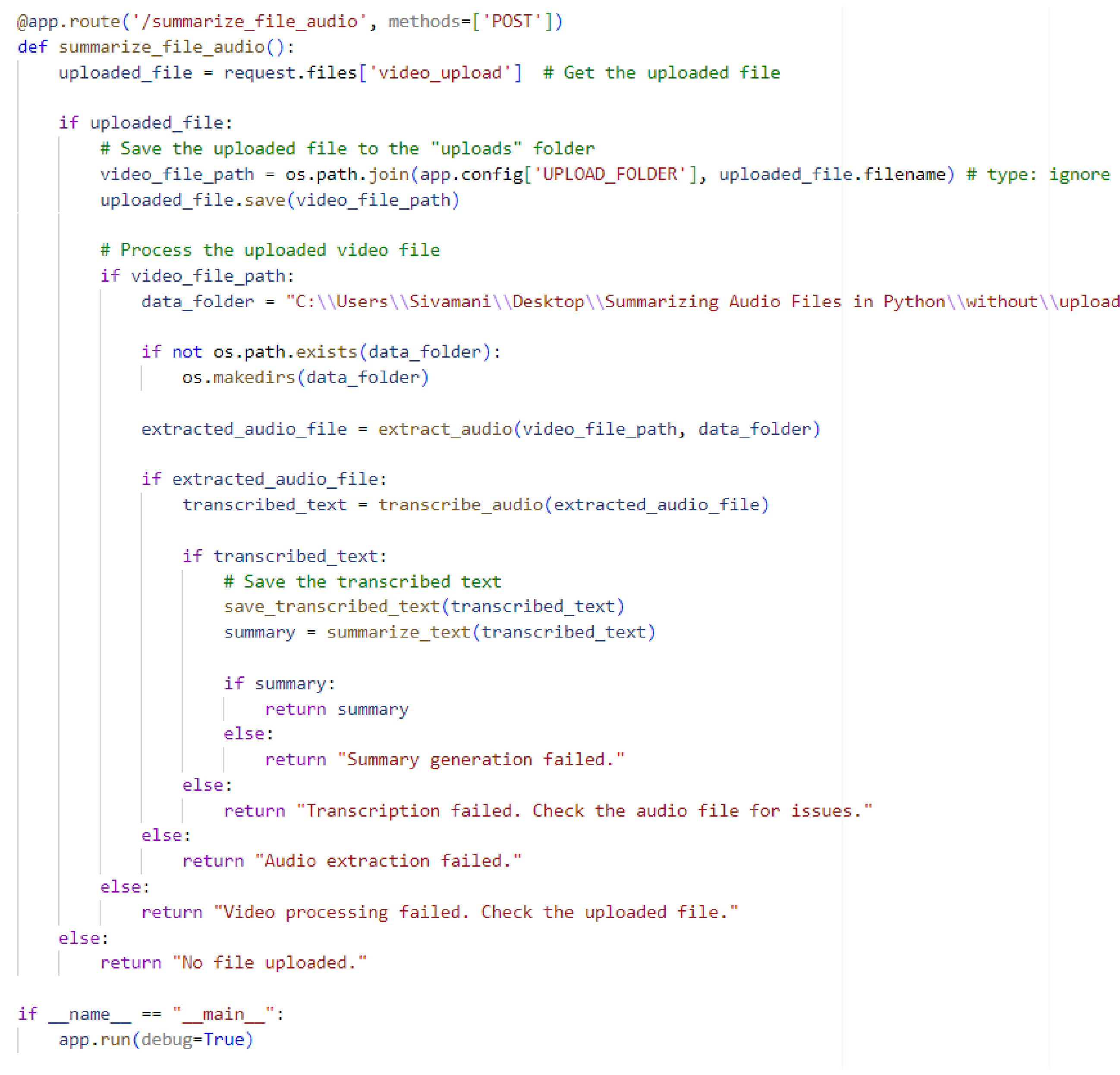
8. Summarize Transcribed Text:

- Leverages the summarize_text function, employing the Summarizer library, to generate a concise summary of the transcribed text.

9. Check and Return Results:

- Verifies the success of transcription and summarization processes.
- Returns the generated summary if successful; otherwise, provides informative error messages for failed steps.

Note: This function encapsulates the entire workflow, showcasing the integration of various libraries to achieve the desired outcome of summarizing audio content recorded from a microphone.



```
@app.route('/summarize_file_audio', methods=['POST'])
def summarize_file_audio():
    uploaded_file = request.files['video_upload']  # Get the uploaded file

    if uploaded_file:
        # Save the uploaded file to the "uploads" folder
        video_file_path = os.path.join(app.config['UPLOAD_FOLDER'], uploaded_file.filename) # type: ignore
        uploaded_file.save(video_file_path)

        # Process the uploaded video file
        if video_file_path:
            data_folder = "C:\\Users\\Sivamani\\Desktop\\Summarizing Audio Files in Python\\without\\upload

            if not os.path.exists(data_folder):
                os.makedirs(data_folder)

            extracted_audio_file = extract_audio(video_file_path, data_folder)

            if extracted_audio_file:
                transcribed_text = transcribe_audio(extracted_audio_file)

                if transcribed_text:
                    # Save the transcribed text
                    save_transcribed_text(transcribed_text)
                    summary = summarize_text(transcribed_text)

                    if summary:
                        return summary
                    else:
                        return "Summary generation failed."
                else:
                    return "Transcription failed. Check the audio file for issues."
            else:
                return "Audio extraction failed."
        else:
            return "Video processing failed. Check the uploaded file."
    else:
        return "No file uploaded."

if __name__ == "__main__":
    app.run(debug=True)
```

This route, **/summarize_file_audio**, handles the summarization process for audio files uploaded by users. Here's an overview of the key steps:

1. Uploaded File Handling:

- Retrieves the uploaded file from the request using request.files['video_upload'].

2. Save Uploaded File:

- Saves the uploaded file to the designated "uploads" folder, creating the folder if it doesn't exist.
- The file path is then stored in the variable video_file_path.

3. Process Uploaded Video File:

- Checks if the video_file_path is valid.
- Creates a data folder for processing uploaded files if it doesn't exist.
- Calls the extract_audio function to extract audio from the video file.

4. Transcribe Audio:

- Uses the transcribe_audio function to convert the extracted audio to text.

5. Save Transcribed Text:

- Saves the transcribed text to a file using the save_transcribed_text function.

6. Summarize Transcribed Text:

- Applies the summarize_text function to generate a summary of the transcribed text.

7. Result Handling:

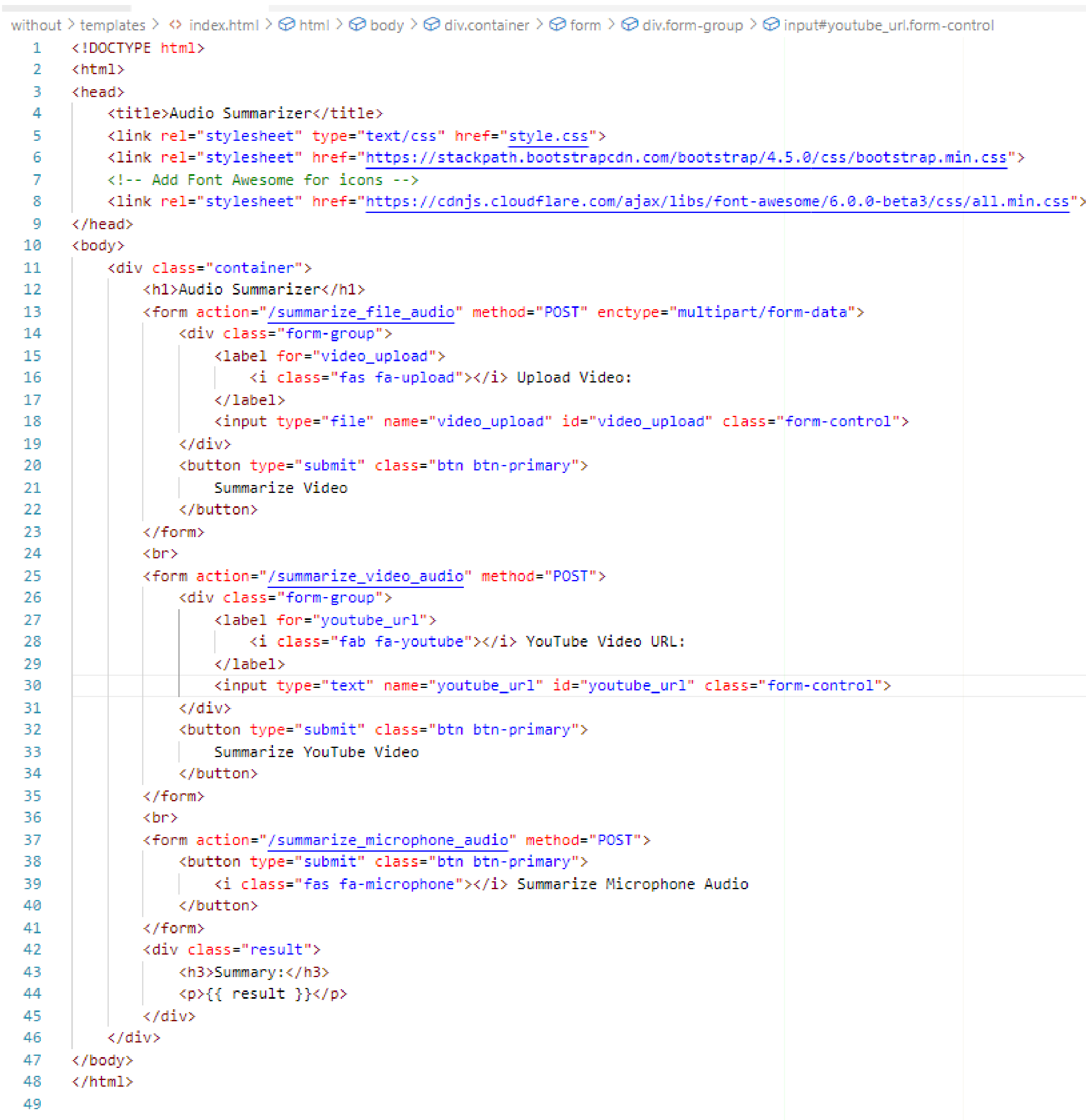
- Checks the success of transcription and summarization processes.
- Returns the generated summary if successful; otherwise, provides error messages for failed steps.

8. Run the Application:

- If this script is executed as the main program, the Flask application is run with debugging enabled.

This route allows users to upload video files, extracts audio, transcribes the audio to text, generates a summary, and provides the summary as the output.

Index.html(For home page)


without > templates > index.html > html > body > div.container > form > div.form-group > input#youtube_url.form-control

```
<!DOCTYPE html>
<html>
<head>
    <title>Audio Summarizer</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
    <!-- Add Font Awesome for icons -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
</head>
<body>
    <div class="container">
        <h1>Audio Summarizer</h1>
        <form action="/summarize_file_audio" method="POST" enctype="multipart/form-data">
            <div class="form-group">
                <label for="video_upload">
                    <i class="fas fa-upload"></i> Upload Video:
                </label>
                <input type="file" name="video_upload" id="video_upload" class="form-control">
            </div>
            <button type="submit" class="btn btn-primary">
                Summarize Video
            </button>
        </form>
        <br>
        <form action="/summarize_video_audio" method="POST">
            <div class="form-group">
                <label for="youtube_url">
                    <i class="fab fa-youtube"></i> YouTube Video URL:
                </label>
                <input type="text" name="youtube_url" id="youtube_url" class="form-control">
            </div>
            <button type="submit" class="btn btn-primary">
                Summarize YouTube Video
            </button>
        </form>
        <br>
        <form action="/summarize_microphone_audio" method="POST">
            <button type="submit" class="btn btn-primary">
                <i class="fas fa-microphone"></i> Summarize Microphone Audio
            </button>
        </form>
        <div class="result">
            <h3>Summary:</h3>
            <p>{{ result }}</p>
        </div>
    </div>
</body>
</html>
```

This HTML code defines the structure of a web page for the "Audio Summarizer" application. Here's a breakdown of its components:

1. Document Type Declaration:

- <!DOCTYPE html> specifies the HTML version used in the document.

2. HTML Structure:

- The HTML document is wrapped in <html> tags.

3. Head Section:

- Contains metadata and links to external resources.
- Title: "Audio Summarizer."

- Stylesheets:
    - "style.css" for custom styles.
    - Bootstrap 4.5.0 CSS.
    - Font Awesome for icons.

4. Body Section:
- <body> contains the main content of the page.

5. Container Division:
- <div class="container"> wraps the entire content for styling.

6. Page Title:
- <h1> displays the title "Audio Summarizer."

7. File Upload Form:
- <form> for uploading video files.
- "Upload Video" label and a file input with the ID "video_upload."
- Submit button labeled "Summarize Video."

8. YouTube Video URL Form:
- <form> for summarizing YouTube videos.
- "YouTube Video URL" label and a text input with the ID "youtube_url."
- Submit button labeled "Summarize YouTube Video."

9. Microphone Audio Form:
- <form> for summarizing microphone audio.
- Submit button labeled "Summarize Microphone Audio" with a microphone icon.

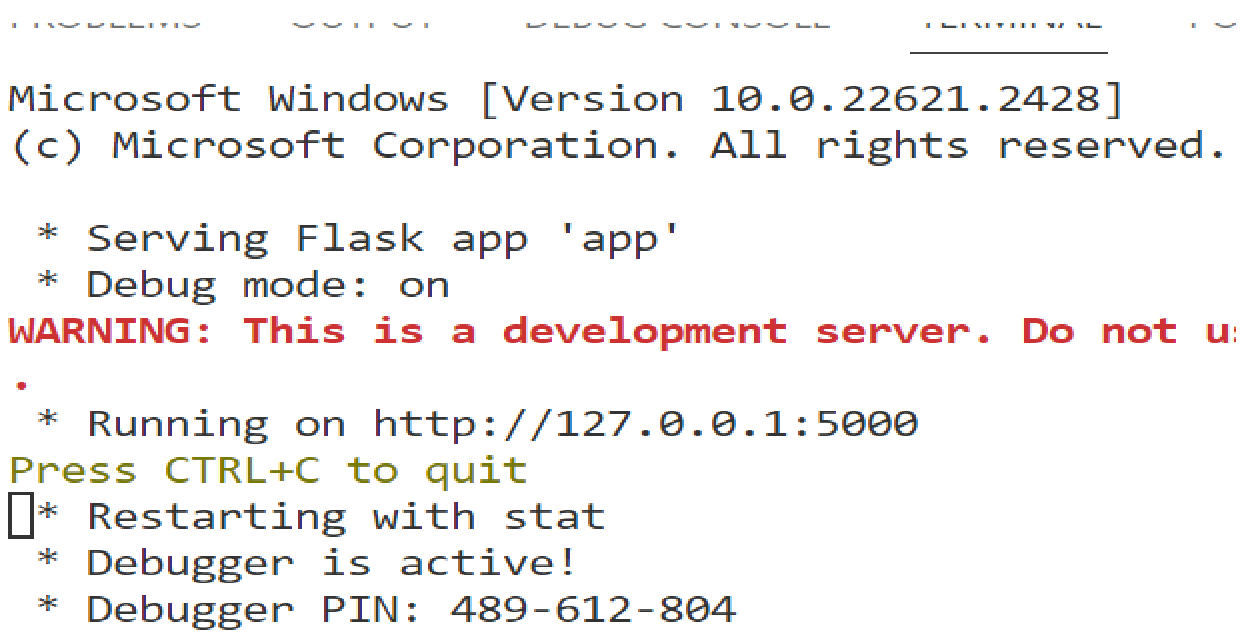
10. Result Display:
- <div class="result"> for displaying the summary.
- <h3> for the "Summary" heading.
- <p> to display the actual summary fetched from the Flask application ({{ result }}).

This HTML structure provides a user interface for interacting with different audio summarization features. The forms allow users to upload video files, input YouTube video URLs, and summarize microphone audio, with the results displayed on the page.

<u>Output:</u>

Compile:

C:/Users/Sivamani/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/Sivamani/OneDrive/Desktop/Summarizing Audio Files in Python/without/app.py"



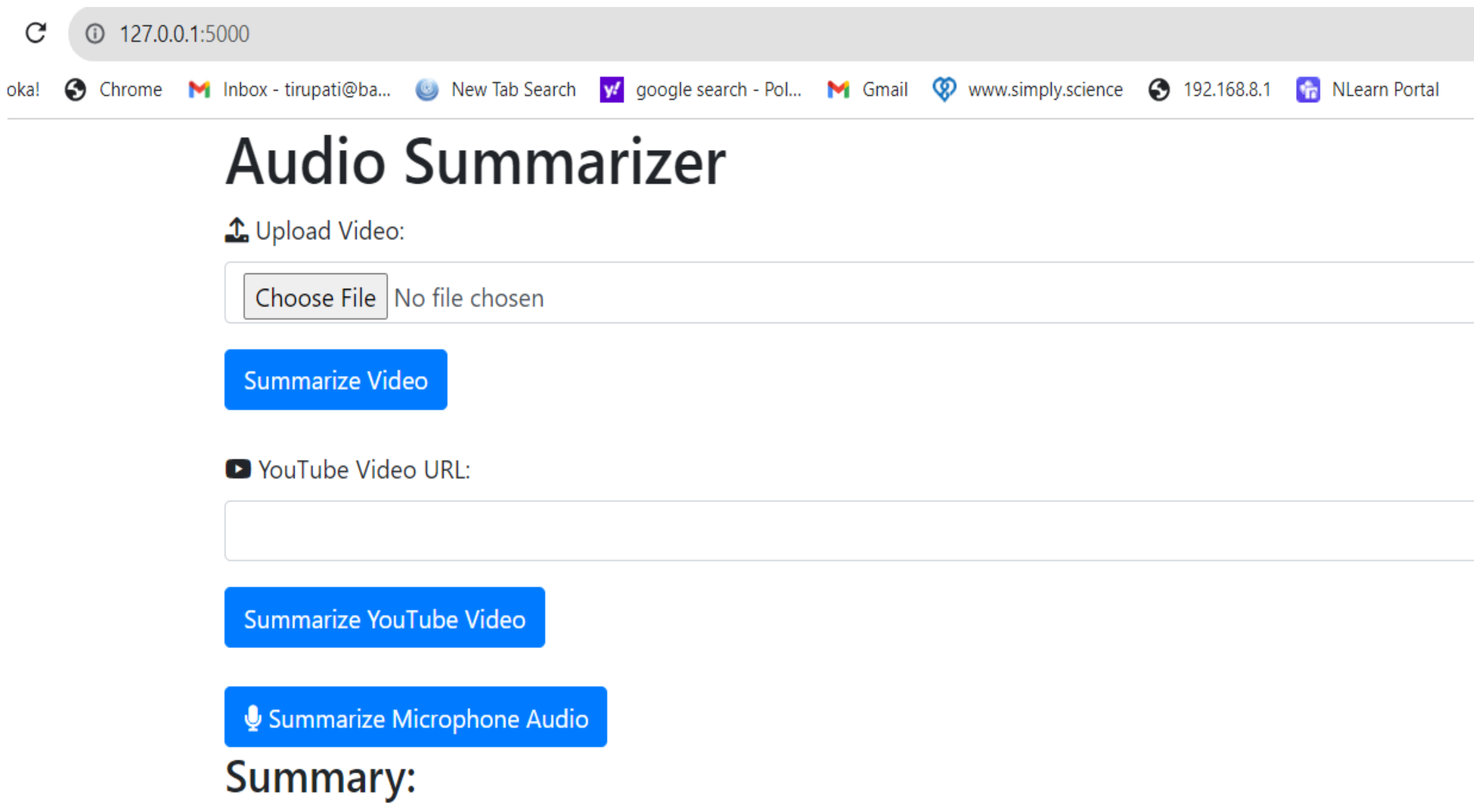
```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not u
.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 489-612-804
```

Open your web browser and type "http://127.0.0.1:5000" in the address bar. Press Enter, and it will redirect you to the "index.html" page, prompting you to enter an ID.

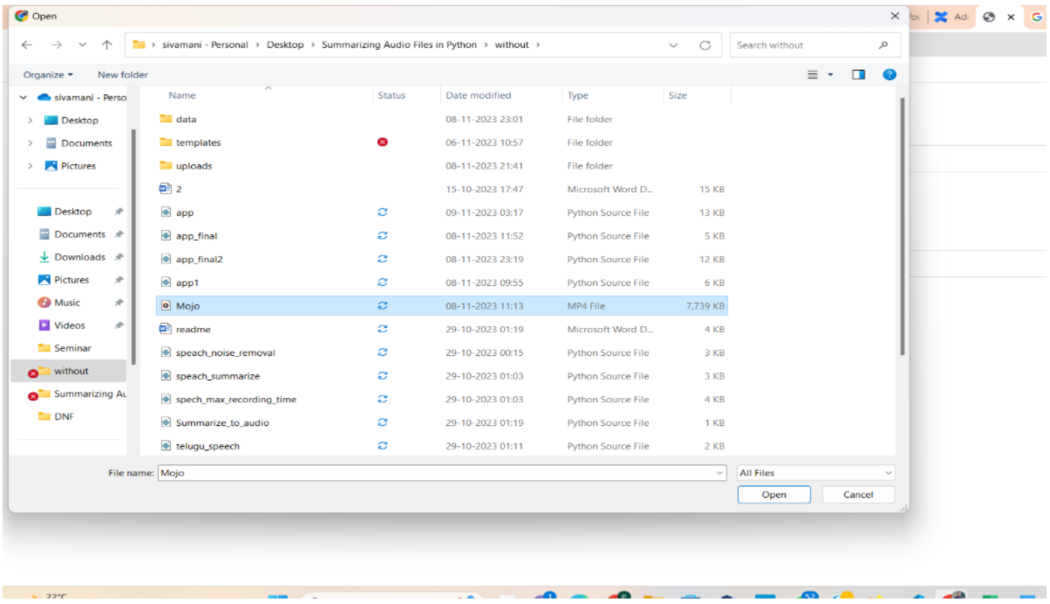
RUN:

Redirects to index.html


127.0.0.1:5000
oka! Chrome Inbox - tirupati@ba... New Tab Search google search - Pol... Gmail www.simply.science 192.168.8.1 NLearn Portal
Audio Summarizer
Upload Video:
Choose File No file chosen
Summarize Video
YouTube Video URL:
Summarize YouTube Video
Summarize Microphone Audio
Summary:

Next You can upload a file by selecting "Choose File" and selecting the audio or video file you want to summarize

I. Choose a video file from your device by clicking "Choose File" and selecting the desired video file from your directory.

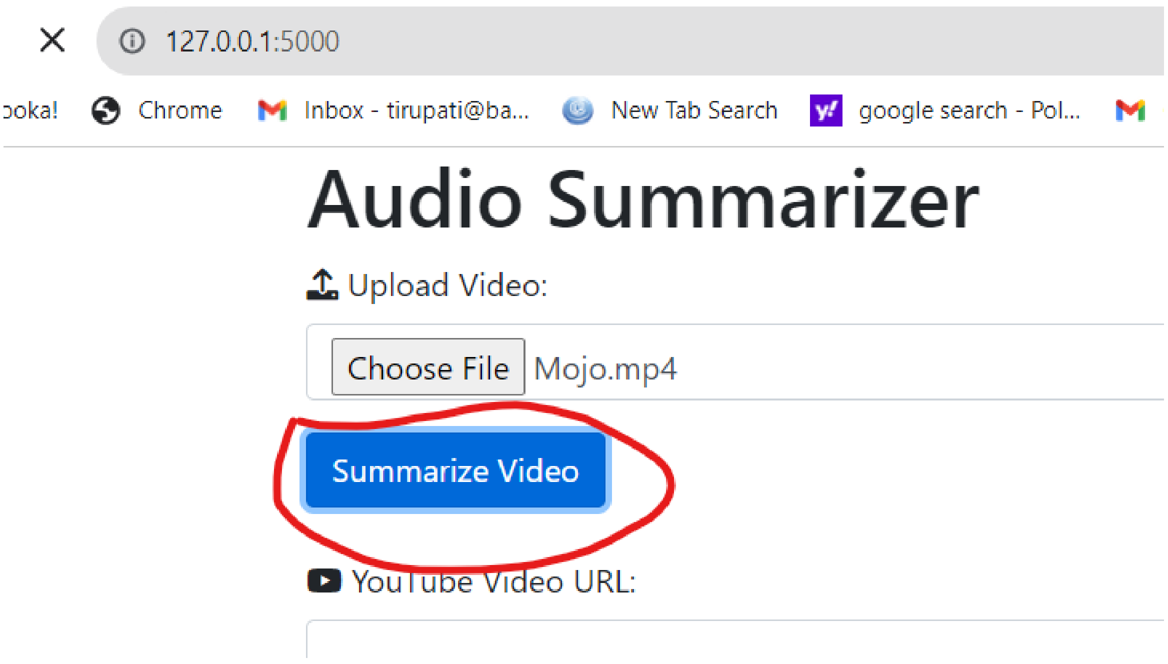


Name	Status	Date modified	Type	Size
data		08-11-2023 23:01	File folder	
templates	✗	06-11-2023 10:57	File folder	
uploads		08-11-2023 21:41	File folder	
2		15-10-2023 17:47	Microsoft Word D...	15 KB
app	↻	09-11-2023 03:17	Python Source File	13 KB
app_final	↻	08-11-2023 11:52	Python Source File	5 KB
app_final2	↻	08-11-2023 23:19	Python Source File	12 KB
app1	↻	08-11-2023 09:55	Python Source File	6 KB
Mojo	↻	08-11-2023 11:13	MP4 File	7,739 KB
readme	↻	29-10-2023 01:19	Microsoft Word D...	4 KB
speach_noise_removal	↻	29-10-2023 00:15	Python Source File	3 KB
speach_summarize	↻	29-10-2023 01:03	Python Source File	3 KB
spech_max_recording_time	↻	29-10-2023 01:03	Python Source File	4 KB
Summarize_to_audio	↻	29-10-2023 01:19	Python Source File	1 KB
telugu_speech	↻	29-10-2023 01:11	Python Source File	2 KB

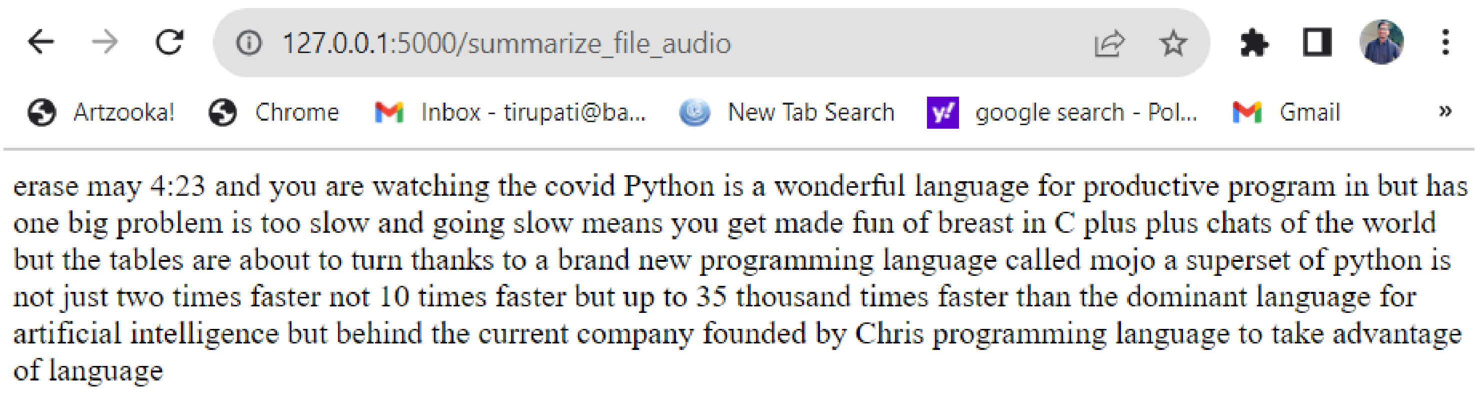
File name: Mojo

All Files

II. Click on Summarize Video



III. Output: The summarize of the upload file will be display



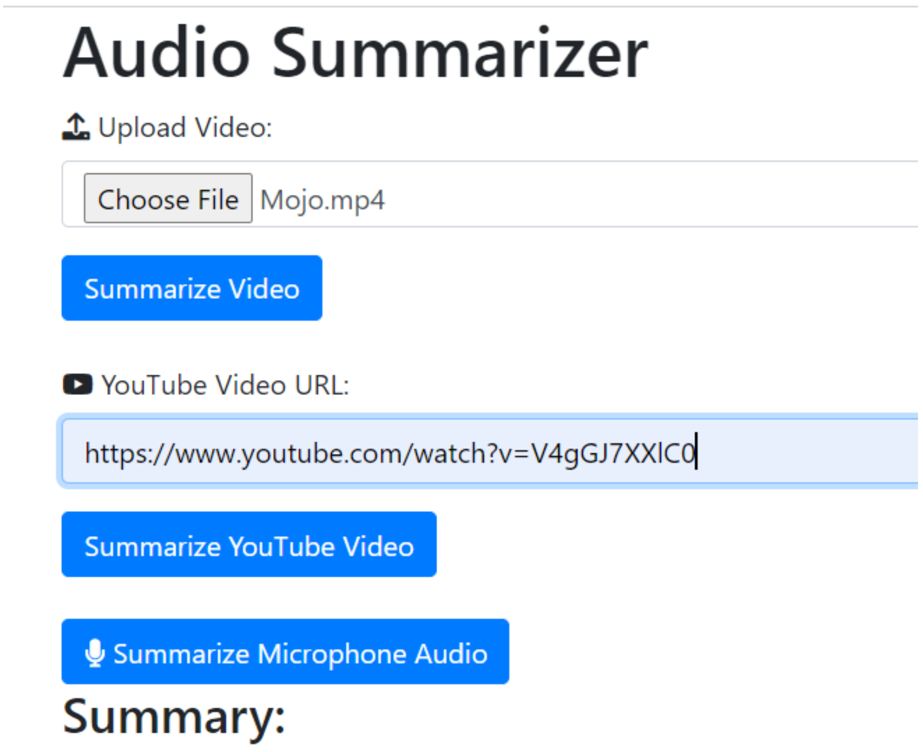
erase may 4:23 and you are watching the covid Python is a wonderful language for productive program in but has one big problem is too slow and going slow means you get made fun of breast in C plus plus chats of the world but the tables are about to turn thanks to a brand new programming language called mojo a superset of python is not just two times faster not 10 times faster but up to 35 thousand times faster than the dominant language for artificial intelligence but behind the current company founded by Chris programming language to take advantage of language

on the new website page

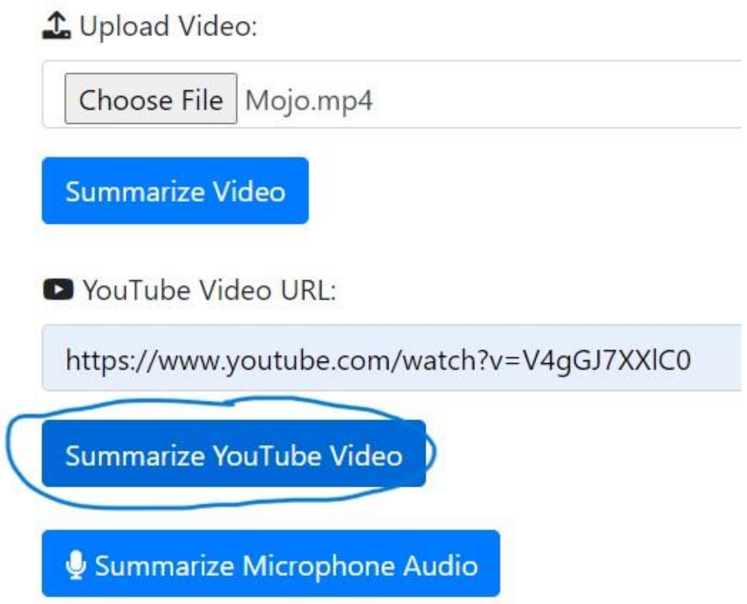
IV. Then Return to the Home Page

Paste a YouTube video URL into the provided field and click "Summarize YouTube Video" to generate a summary.

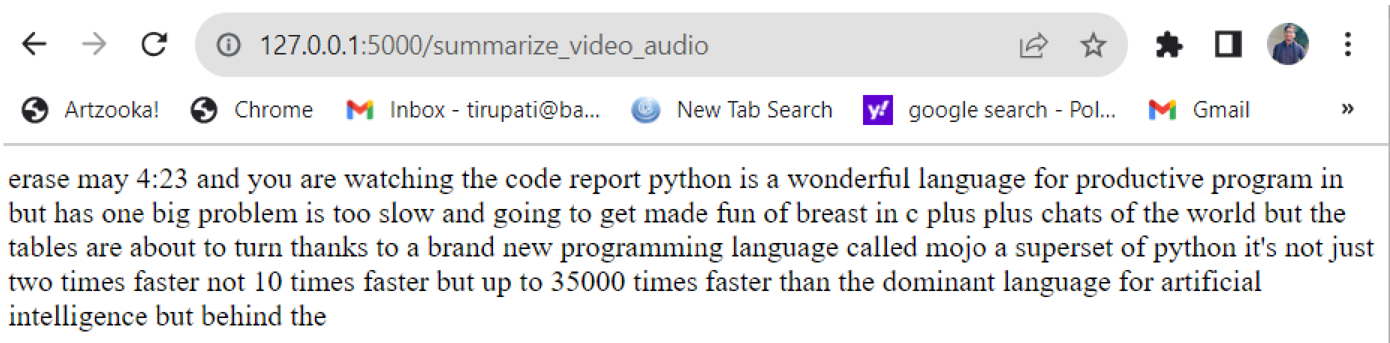
I. Paste the url of any vedio



II. Click on the Summarize the youTube video



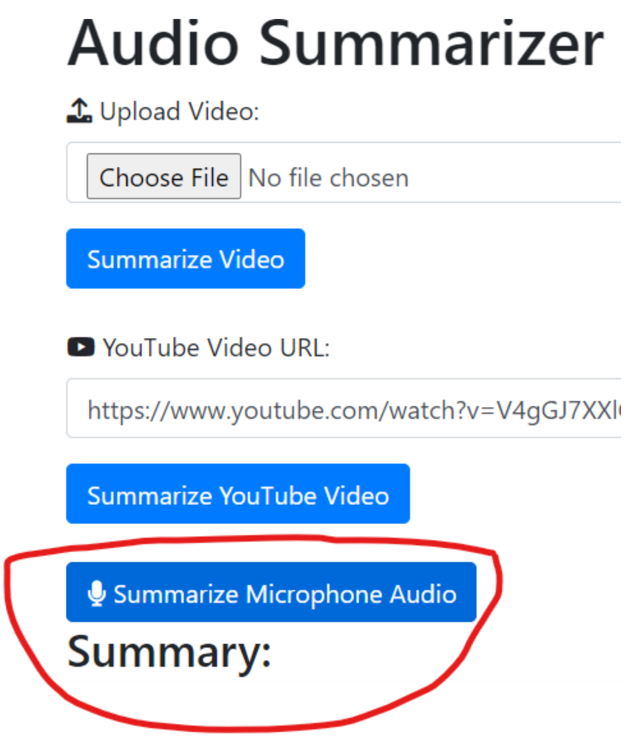
III. Output: The summarize of the upload file will be display on the new website page

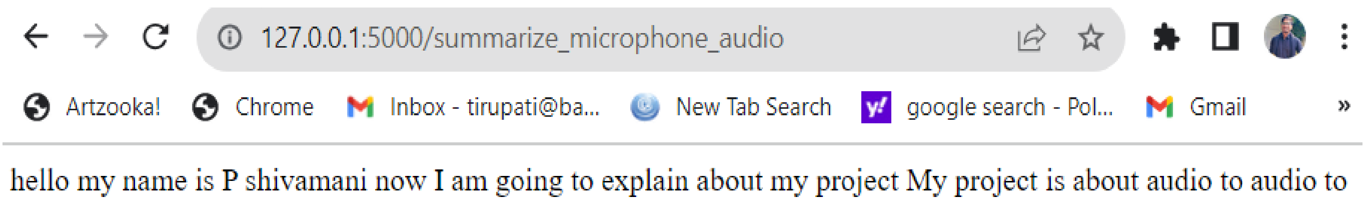


erase may 4:23 and you are watching the code report python is a wonderful language for productive program in but has one big problem is too slow and going to get made fun of breast in c plus plus chats of the world but the tables are about to turn thanks to a brand new programming language called mojo a superset of python it's not just two times faster not 10 times faster but up to 35000 times faster than the dominant language for artificial intelligence but behind the

IV. Then Return to the Home Page

Click "Summarize Microphone Audio" and speak into your microphone. The system will transcribe and summarize your spoken words.





hello my name is P shivamani now I am going to explain about my project My project is about audio to audio to

5. CONCLUSION

In this project, the aim was to bridge the gap between the abundance of audio content available and the ability to extract valuable insights from it. We recognized the growing need for efficient audio summarization tools in various fields such as research, journalism, education, and professional analysis.

To address this need, we developed an Audio Summarizer web application using Flask, which serves as a centralized platform for users to conveniently summarize audio content from different sources. By integrating functionalities like video file uploads, YouTube video URL summarization, and real-time microphone audio summarization, we aimed to provide users with a versatile tool adaptable to their diverse needs and preferences.

The key to the success of our application lies in the robustness of its audio processing capabilities. Leveraging powerful libraries such as pytube, moviepy, speech_recognition, and noisereduce, we ensured that the application could handle tasks like audio extraction, transcription, and noise reduction with efficiency and accuracy.

Moreover, we paid close attention to the user interface design, employing HTML and CSS to create a clean and intuitive interface. By incorporating responsive design principles and maintaining consistent styling throughout the application, we aimed to enhance the user experience and make the summarization process as seamless as possible.

In essence, our Audio Summarizer project represents a harmonious blend of web development and audio processing technologies. It not only provides users with a practical tool for extracting insights from audio content but also showcases Python's versatility in creating user-friendly applications tailored to specific needs.