SoundSage - Phase 1

Tools:

AutoGain:

AutoGain is a simple program for automatically Gain-Staging specified audio files. AutoGain works by taking an input from the user regarding where the audio files are and where the user wants the processed audio files to be designated for export. After the files are selected and the output folder is selected, the user will press the begin button to begin processing and as the files are processed you will see them become present in the designated destination directory. Additionally before processing the AutoGain system has an audio_analysis.py script for analyzing the audio prior to processing. In order to ensure the highest quality output and audio fidelity, the AutoGain system only affects the Gain level and does not compress or change anything else about the file *unless otherwise specified and AutoGain only affects the gain level if substantial adjustment is required as specified in the analysis from the audio_analysis.py. Currently AutoGain only has the feature to set the Gain level of specified files to -18LUFS/18DbFS Headroom but in the future we will add the other industry standards as well.

User Interface/User Experience:

A Chatbot for all your Audio Processing needs.

The UI will be intuitive and simple as it will only be a Chatbot interface, although you should not be fooled by its simplicity. Utilizing OpenAI technology we will create a chatbot that can automate any task related to audio processing whether you're working in your DAW or not. Simply specify which project you're working on and SoundSage will use its file scraping/file management features to extract the files from any folder on your harddrive or external SSD.

With the Help of Natural Language Processing we will be able to automatically extract keywords that are related to specific audio processing tasks and create a list of commands for the system to examine and execute right before your eyes.

Command Handling and Audio Processing:

Before we can use your prompt to process any audio, first we need to translate it into a language that the computer understands, such as python.

The idea is that we can analyze your prompt for certain keywords by comparing it to a dataset of words and meanings related to specific tasks in the audio processing field. After comparing it to the dataset, SoundSage should be able to take any set of words that an audio engineer may use to communicate a task, such as gain-staging.

**In this example we will break down how the program will handle commands and process audio.*

1. User interacts with the LLM Chat Interface and inputs the command: "Gain-Stage all of the audio files in this specific folder."
    1.1 Validate the user's command for any errors or inconsistencies.
    1.2 If the command is not valid, return an error message to the user and ask for a new command.

2. LLM interprets the command:

2.1 "Gain-Stage" is interpreted as a call to the AutoGain function.
2.2 "audio files" is interpreted as files with extensions .wav, .mp3, .flac, .aiff, etc.
2.3 "all" is interpreted as a command to select all files in the specified folder.
2.4 "specific folder" is interpreted as the path to the folder containing the audio files to be processed.

3. LLM generates a list of commands to be executed in the correct order to produce the final processed files.

4. LLM triggers a Python script with the following steps:
4.1 The script checks if the specified folder exists and contains valid audio files. If not, it returns an error.
4.1.1 If the folder does not exist, return an error message.
4.1.2 If the folder does not contain valid audio files, return an error message.
4.2 The script navigates to the specified folder and copies the audio files.
4.2.1 If the navigation or copying process fails, return an error message.
4.3 The script navigates to the "PreProcess" folder within the AutoGain directory and pastes the copied audio files.
4.3.1 If the navigation or pasting process fails, return an error message.
4.4 The script sets the input and output directories for the AutoGain program.
4.4.1 If the setting process fails, return an error message.
4.5 The script initiates the AutoGain program, which processes the audio files.
4.5.1 If the AutoGain program fails to run or returns an error, handle the error and return an error message.
4.6 After the processing is complete, the script creates a "New" folder within the original specified folder.
4.6.1 If the creation of the "New" folder fails, return an error message.
4.7 The script navigates to the "PostProcess" folder within the AutoGain directory and copies the processed audio files.
4.7.1 If the navigation or copying process fails, return an error message.
4.8 The script pastes the copied processed audio files into the "New" folder.
4.8.1 If the pasting process fails, return an error message.
4.9 The script sends a signal to the LLM indicating that the process is complete.
4.9.1 If the signal fails to send, return an error message.

5. LLM generates a response to the user, including information about where the processed files are located and how to retrieve them. The response also includes a log of all processing applied, analysis results, and recommended actions for improved audio quality.
5.1 If the response generation fails, return an error message.

6. If necessary, additional scripts are written to integrate the LLM with AutoGain and to allow the LLM to manage files locally on a computer.
6.1 If the writing of additional scripts fails or encounters issues, handle these appropriately and return error messages.

7. The LLM Chat Interface is developed to replace the original UI.
7.1 Ensure the interface is user-friendly and intuitive.
7.2 Implement error handling in the UI to manage any issues that arise during user interaction.
7.3 Test the interface thoroughly to ensure it works as expected.

Code Structure:

Project Directory contents:

>SoundSage
  >Workbench
    >PreProcess
    >PostProcess
    >Archive
    >AudioTools
      >AutoGain
        audio_analysis.py
        audio_processor.py
        main.py
        >nenv* *see **Directories*** for more details…
 Autogain_interaction.py
 chatbot_code_writer.py
 Completion_handling.py
 Error_handling.py
 File_management.py
 Main.py
 Menu_functions.py
 Openai_interaction.py
 Send_button_functionality.py
 Template_code.py

Directories:

SoundSage:

This is the folder containing the entirety of the project.

Workbench:

This is where the SoundSage system will operate, bringing files inside of the environment before analysis and applying any sort of processing.

PreProcess:

This is where the SoundSage system will store files before they are processed. SoundSage will navigate to the user specified folder where the specified files can be found and migrate them by copying the specified files, moving to the PreProcess folder within the WorkBench and pasting the specified files in to the PreProcess folder.

PostProcess:

This is where the SoundSage system will store the processed files. Once the processing is complete, SoundSage will navigate to the original specified folder where the specified files were found before they were moved to PreProcess and create a new Folder with the same name as the original specified folder where the specified files were found but with an added “-processed” at the end of the name.
Ex. the original specified folder where the specified files were found is named “Audio Clips”, the new folder post processing that the system creates for the new processed audio files outside of the environment and on the hard drive (typically this will be within a ProTools session file, but lets forget that for now) essentially the new folders name will be “Audio Clips-processed” and the folder path should be “Audio Clips/Audio Clips-processed”.

Archive:
This is where the SoundSage system will store, chat logs, analysis data logs and a backlog of processed and unprocessed files in case there are any issues in the future where the files are lost, the user can find them and claim them back.

AudioTools:
This is where the tools that the language model will call on to execute custom commands from the user prompt it currently contains AutoGain, but in the future we will work on integrating more tools like an Auto EQ or an AutoDynamics, AutoPan, AutoPitch, AutoDither, AutoDelay, AutoReverb, AutoQuantize, AutoEdit, AutoEverything etc….

AutoGain
This folder contains the AutoGain tool, a tool that simply automates gain staging by analyzing the file and either boosting, turning down or simply just doing nothing to the specified file.

Directories:

main.py:
This will be the main script that runs your application. It will import and use the functions from the other scripts.openai_interaction.py: This script will handle all interactions with the OpenAI API. It will contain functions for sending user inputs to the API and receiving responses.

```
from tkinter import messagebox, Menu, Tk, Text, Button, Scrollbar
import openai_interaction
import menu_functions
import send_button_functionality

def main():
    # Initialize the application
    root = Tk()
    root.title("Chat Bot")
    root.geometry("400x500")
    root.resizable(width=False, height=False)

    # Create the main menu
    main_menu = Menu(root)
    root.config(menu=main_menu)

    # Add the file menu to the main menu
    file_menu = Menu(root)
    main_menu.add_cascade(label="File", menu=file_menu)

    # Add commands to the file menu
    file_menu.add_command(label="New..", command=menu_functions.new_file)
    file_menu.add_command(label="Save As..", command=menu_functions.save_as)
    file_menu.add_command(label="Exit", command=root.quit)

    # Add the rest of the menu options to the main menu
    main_menu.add_command(label="Edit", command=menu_functions.edit)
    main_menu.add_command(label="Quit", command=root.quit)

    # Create the chat window
    chatWindow = Text(root, bd=1, bg="black",  width="50", height="8", font=("Arial", 23), foreground="#00ffff")
    chatWindow.place(x=6,y=6, height=385, width=370)

    # Create the message window
    messageWindow = Text(root, bd=0, bg="black",width="30", height="4", font=("Arial", 23), foreground="#00ffff")
    messageWindow.place(x=128, y=400, height=88, width=260)
```

```
    # Create the scrollbar
    scrollbar = Scrollbar(root, command=chatWindow.yview, cursor="star")
    scrollbar.place(x=375,y=5, height=385)

    # Create the send button
    sendButton = Button(root, text="Send",  width="12", height=5,
                bd=0, bg="#0080ff", activebackground="#00bfff",foreground='#ffffff',font=("Arial", 12),
                command=lambda: send_button_functionality.send_button_click(None, messageWindow, chatWindow))
    sendButton.place(x=6, y=400, height=88)

    # Run the application
    root.mainloop()

if __name__ == "__main__":
    main()
```

menu_functions.py:

This script will contain functions that get executed when the menu options are selected. For example, it might contain a function for creating a new file when "New.." is selected, a function for saving the current state of the application when "Save As.." is selected, and so on.

```
from tkinter import filedialog

def new_file()
    # Create a new file
    print("Creating a new file...")


def save_as():
    # Save the current state of the application
    print("Saving the application state...")

def edit():
    # Edit the current state of the application
    print("Editing the application state...")
```

send_button_functionality.py:

This script will contain the function that gets executed when the "Send" button is clicked. This function will take the user's input, send it to the OpenAI API (using a function from openai_interaction.py), and display the response in the chat window.

```
import openai_interaction


def send_button_click(event, input_field, chat_window):
    # Get the user's input
    user_input = input_field.get("1.0", 'end-1c')

    # Send the user's input to the OpenAI API
    response = openai_interaction.send_input_to_openai(user_input)

    # Display the response in the chat window
    chat_window.insert('end', '\n' + 'ChatBot: ' + response)
```

autogain_interaction.py:

This script will contain functions for interacting with the AutoGain software. It will contain functions for sending commands to the software and handling its responses.

```
def send_command_to_autogain(command):


    # Send a command to the AutoGain software
    # You'll need to replace this with the actual code for sending a command to AutoGain
    print(f"Sending command to AutoGain: {command}")

def handle_autogain_response(response):
    # Handle a response from the AutoGain software
    # You'll need to replace this with the actual code for handling a response from AutoGain
    print(f"Handling AutoGain response: {response}")
```

file_management.py:

This script will contain functions for handling file management tasks. This might include functions for navigating to a specified folder, copying files, and pasting files.

```
import os
import shutil


def navigate_to_folder(folder_path):
    # Navigate to a specified folder
    os.chdir(folder_path)

def copy_file(source_path, destination_path):
    # Copy a file from source_path to destination_path
    shutil.copy2(source_path, destination_path)

def paste_file(file_path, destination_path):
    # Paste a file at destination_path
    shutil.copy2(file_path, destination_path)
```

error_handling.py:

This script will contain functions for handling errors that might occur during the execution of your application. It might also contain functions for providing progress updates to the user.

```
def handle_error(error):


    # Handle an error
    print(f"An error occurred: {error}")

def provide_progress_update(progress):
    # Provide a progress update to the user
    print(f"Progress: {progress}%")
```

completion_handling.py:

This script will contain functions for handling the completion of the process. This might include a function for sending a signal to the LLM to respond that the process is complete, and a function for checking the new folder to ensure that the process completed successfully.

```
def signal_completion_to_llm():


    # Signal to the LLM that the process is complete
    print("Process complete. Signaling to LLM...")

def check_new_folder(new_folder_path):
    # Check the new folder to ensure that the process completed successfully
    if os.path.exists(new_folder_path):
        print("New folder exists. Process completed successfully.")
    else:
        print("New folder does not exist. Process did not complete successfully.")
```

template_code.py:

This script will contain the template code that the chatbot will modify. It will be designed to work with the specified files and directories.

```
import os


def template_code(directory_path):
    # Navigate to the specified directory
    os.chdir(directory_path)

    # Perform some operation on the files in the directory
    for filename in os.listdir(directory_path):
        print(f"Processing file: {filename}")

    # Return a message indicating that the operation is complete
    return "Operation complete."
```

chatbot_code_writer.py:

This script will contain the code that modifies the template code based on the user's input. It will use the OpenAI API to generate the necessary Python code.

```
import openai_interaction

def write_code(user_input):
    # Send the user's input to the OpenAI API
    response = openai_interaction.send_input_to_openai(user_input)

    # Write the response from the API to a Python file
    with open("generated_code.py", "w") as file:
        file.write(response)

    # Return a message indicating that the code has been written
    return "Code written to generated_code.py."
```

AutoGain Code:

main.py:

```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk, ImageEnhance
import audio_analysis
import audio_processor


def open_files():
    global audio_files
    audio_files = list(filedialog.askopenfilenames(filetypes=[("Audio files", "*.wav")]))
    file_label.config(text=" ".join(audio_files))

def choose_output_directory():
    global output_directory
    output_directory = filedialog.askdirectory()
    output_label.config(text=output_directory)

def begin_process():
    for audio_file in audio_files:
        analyzed_data = audio_analysis.analyze_audio(audio_file)
        audio_processor.process_audio(audio_file, analyzed_data, output_directory)

root = tk.Tk()
root.title("AutoGain by SoundSage")

# Make the window square and resizable
root.geometry("1200x1200")
root.resizable(True, True)

audio_files = []
output_directory = ""

# Load the logo image
logo_image = Image.open("LOGO.png")

# Resize the logo and set the opacity
logo_width, logo_height = logo_image.size
logo_image = logo_image.resize((int(1.25 * logo_width), int(1.25 * logo_height)), Image.ANTIALIAS)
logo_image = ImageEnhance.Brightness(logo_image).enhance(0.5)

# Create a PhotoImage instance of the logo
logo_photo = ImageTk.PhotoImage(logo_image)
```

```
# Add the logo as a background image with the desired background color
logo_label = tk.Label(root, image=logo_photo, bg="#DBC6B1")
logo_label.place(x=0, y=0, relwidth=1, relheight=1)

# Set the window color
root.configure(bg="#DBC6B1")

# Create UI elements with custom styling
file_button = tk.Button(root, text="Choose Stems", command=open_files, bg="white", fg="black", font=("Helvetica", 10, "bold"), bd=0, height=2,
width=20)
file_button.place(relx=0.1, rely=0.25, anchor=tk.CENTER)

file_label = tk.Label(root, text="", wraplength=400, bg="#F1EAE2", fg="black")
file_label.place(relx=0.038, rely=0.30, anchor=tk.CENTER)

output_button = tk.Button(root, text="Export to...", command=choose_output_directory, bg="white", fg="black", font=("Helvetica", 10, "bold"), bd=0,
height=2, width=20)
output_button.place(relx=0.1, rely=0.45, anchor=tk.CENTER)

output_label = tk.Label(root, text="", wraplength=400, bg="#F1EAE2", fg="black")
output_label.place(relx=0.038, rely=0.50, anchor=tk.CENTER)

begin_button = tk.Button(root, text="Begin", command=begin_process, bg="white", fg="black", font=("Helvetica", 14, "bold"), bd=0, height=4, width=23)
begin_button.place(relx=0.5, rely=0.9, anchor=tk.CENTER)

root.mainloop()
```

audio_analysis.py:

```
import librosa
import numpy as np
from pyloudnorm import Meter


def analyze_audio(audio_file):
    y, sr = librosa.load(audio_file, sr=None)  # Added `sr=None` to avoid resampling
    meter = Meter(sr)
    lufs = meter.integrated_loudness(y)

    # Compute the maximum decibels full scale (dBFS) and subtract the headroom
    max_dbfs = np.max(librosa.amplitude_to_db(y)) - 18  # Subtract the headroom

    # Compute the number of bits per sample
    bits_per_sample = y.dtype.itemsize * 8

    # Compute the number of audio channels (1 for mono, 2 for stereo)
    num_channels = 1 if len(y.shape) == 1 else y.shape[1]

    return {'lufs': lufs, 'max_dbfs': max_dbfs, 'bits_per_sample': bits_per_sample, 'num_channels': num_channels}
```

audio_processor.py:

```
import os
import ffmpy
import shutil
import subprocess
import json
def process_audio(audio_file, analyzed_data, output_directory):
    target_lufs = -18
    headroom = 18

    gain_adjustment = target_lufs - analyzed_data['lufs']
    output_file = os.path.join(output_directory, 'processed_' + os.path.basename(audio_file))

    # Get the input file's bits per sample and sample rate
    ffprobe = ffmpy.FFprobe(
        inputs={audio_file: None},
        global_options='-v error -show_entries stream=bits_per_sample,sample_rate -of json'
```

```
)
ffprobe_output = ffprobe.run(stdout=subprocess.PIPE)[0].decode()
ffprobe_json = json.loads(ffprobe_output)
bits_per_sample = int(ffprobe_json['streams'][0]['bits_per_sample'])
sample_rate = int(ffprobe_json['streams'][0]['sample_rate'])

# Set the output sample format based on the input bits per sample
sample_format = {16: 'pcm_s16le', 32: 'pcm_s32le'}.get(bits_per_sample, 'pcm_s16le')

if gain_adjustment != 0 or analyzed_data['max_dbfs'] > target_lufs + headroom:
    # Apply gain adjustment and set the output sample format
    ff = ffmpy.FFmpeg(
        inputs={audio_file: None},
        outputs={
            output_file: f'-ar {sample_rate} -acodec {sample_format} -af volume={gain_adjustment}dB'}
    )
    ff.run()

else:
    # No processing needed, just copy the input file to the output file
    shutil.copyfile(audio_file, output_file)
```