The Quad Chips

Week - 4

Hardware Components and testing

Normalization and Noise Reduction of different audio files

Lung Sound Recorder for portable healthcare devices

Team - 38

The Quad Chips

Sai Praneeth - 2022101097

Venkata Jahnavi - 2022101118

Sai Divya - 2022101090

Vaishnavi Priya - 2022101108

Assosiated Professor - Abhishek Srivastava

Assosiated TA - Santhoshini Thota

Team 38 , Group 4

WEEK 4	Building server, handling the audio files
WEEK 5	Normalization & Noise reduction

Team Member wise Work:

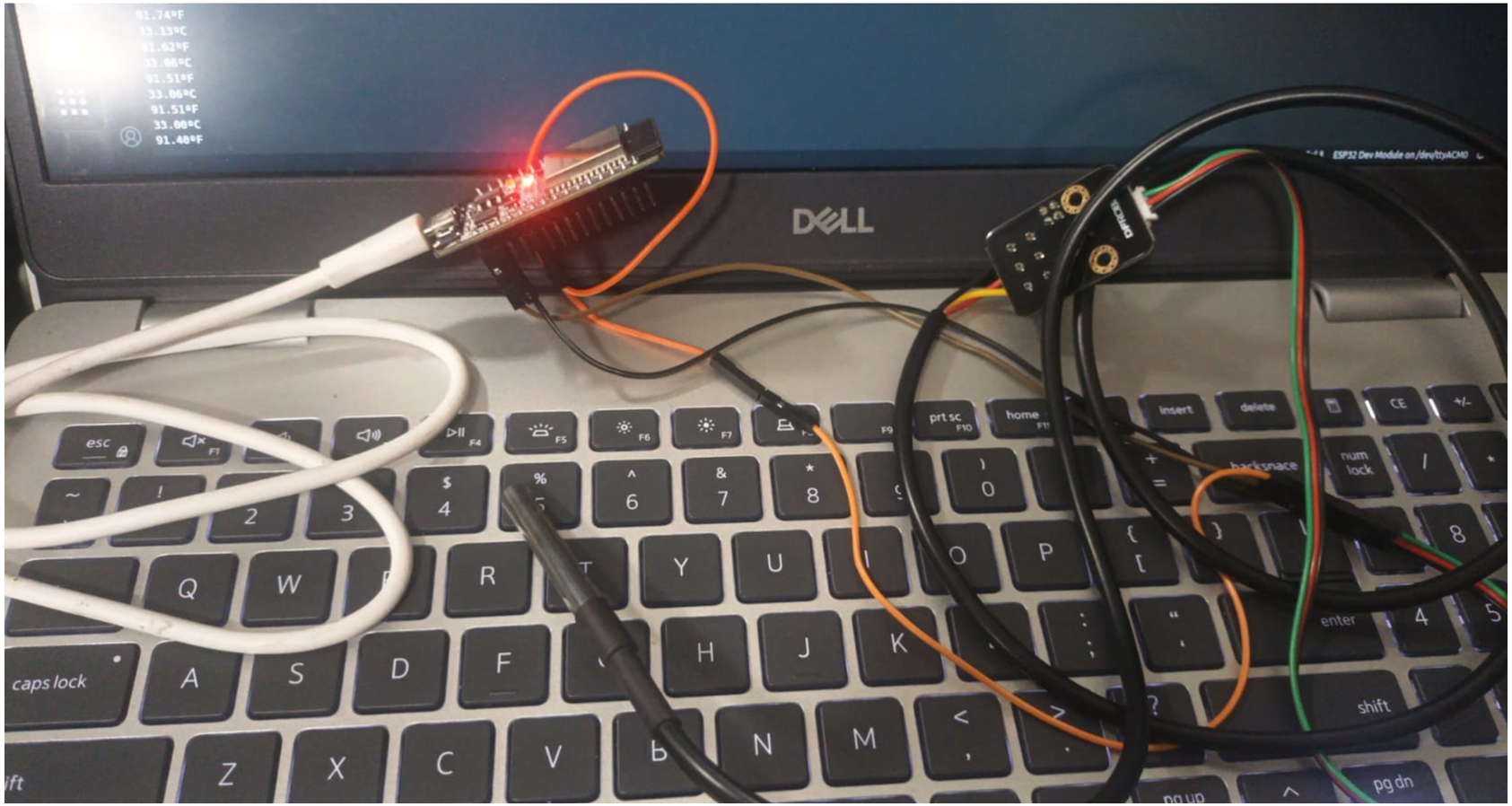
Sai Praneeth (2022101097) - Server code (handling audio files & datapoints)

Jahnavi (2022101118) - hardware component building & testing

Sai Divya (2022101090)
Noise reduction-applying low pass filter and working on
spectral subtraction

Vaishnavi (202101108)-Normalizing audio file & working on wavelet denoising. (coding part).

Testing DS18B20 temperature sensor



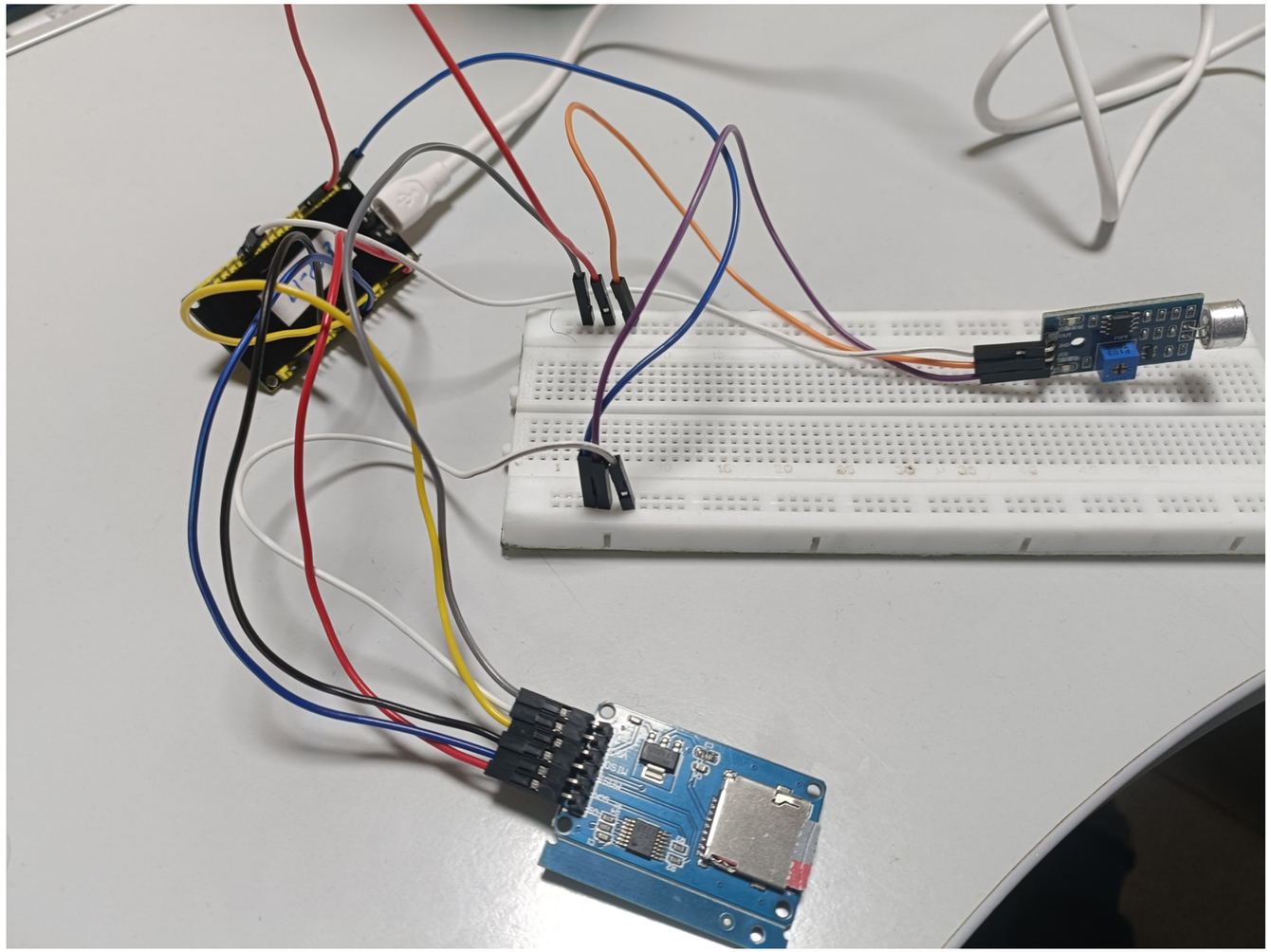
Code for temperature sensor

```
#include <OneWire.h>
#include <DallasTemperature.h>

const int oneWireBus = 4;

OneWire oneWire(oneWireBus);

DallasTemperature sensors(&oneWire);

void setup() {
  Serial.begin(115200);
  sensors.begin();
}

void loop() {
  sensors.requestTemperatures();
  float temperatureC = sensors.getTempCByIndex(0);
  float temperatureF = sensors.getTempFByIndex(0);
  Serial.print(temperatureC);
  Serial.println("ºC");
  Serial.print(temperatureF);
  Serial.println("ºF");
  delay(1000);
}
```

Temperature Readings on serial monitor

```
82.06ºF
27.81ºC
82.06ºF
27.81ºC
82.06ºF
27.81ºC
82.06ºF
27.87ºC
82.17ºF
27.87ºC
82.17ºF
27.87ºC
82.17ºF
27.87ºC
82.17ºF
27.87ºC
82.17ºF
27.94ºC
82.29ºF
28.00ºC
82.40ºF
```

Testing MicroSD Card module and Mic module



Python code to convert .aac to .wav

```python
from pydub import AudioSegment
input_file = "1.aac"

output_file = "output_audio.wav"

try:
    audio = AudioSegment.from_file(input_file, format="aac")

    audio.export(output_file, format="wav")

    print(f"Conversion completed. Output saved as {output_file}")

except Exception as e:
    print(f"An error occurred: {str(e)}")
```

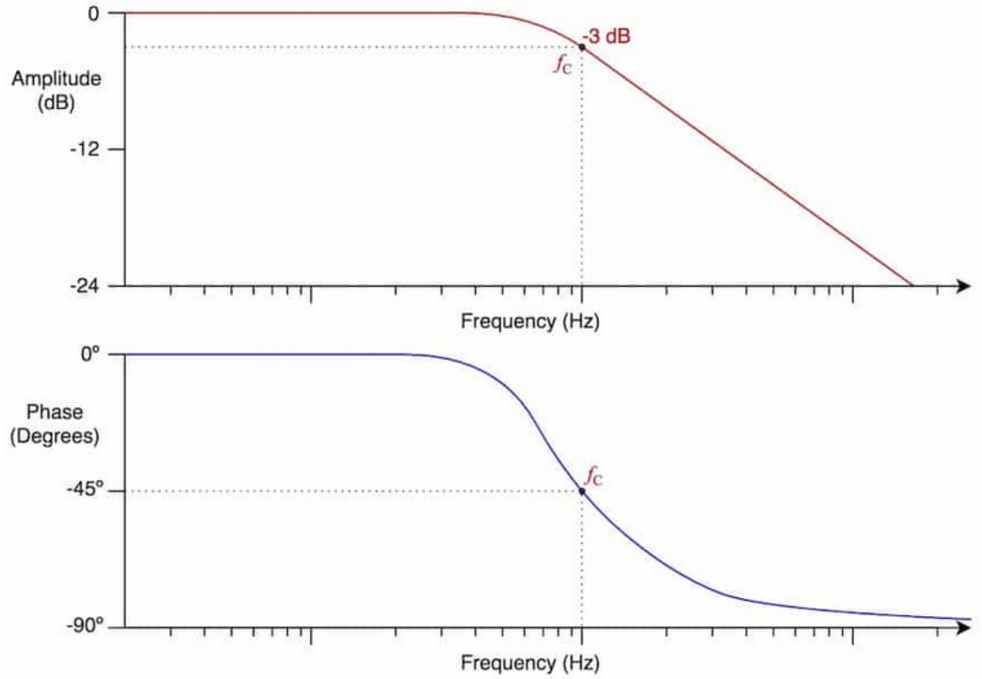
Recorded audio files in .aac are converted to .wav files that enables us listen to the audio.

Low pass Filter Code:

```
1   from pydub import AudioSegment
2
3   audio = AudioSegment.from_wav("normalized_cheering.wav")
4
5   filtered_audio = audio.low_pass_filter(3000)
6
7   filtered_audio.export("filtered_audio.wav", format="wav")
```

The above code uses the most basic technique that is to use a low_pass_filter.

Its not the most efficient way.

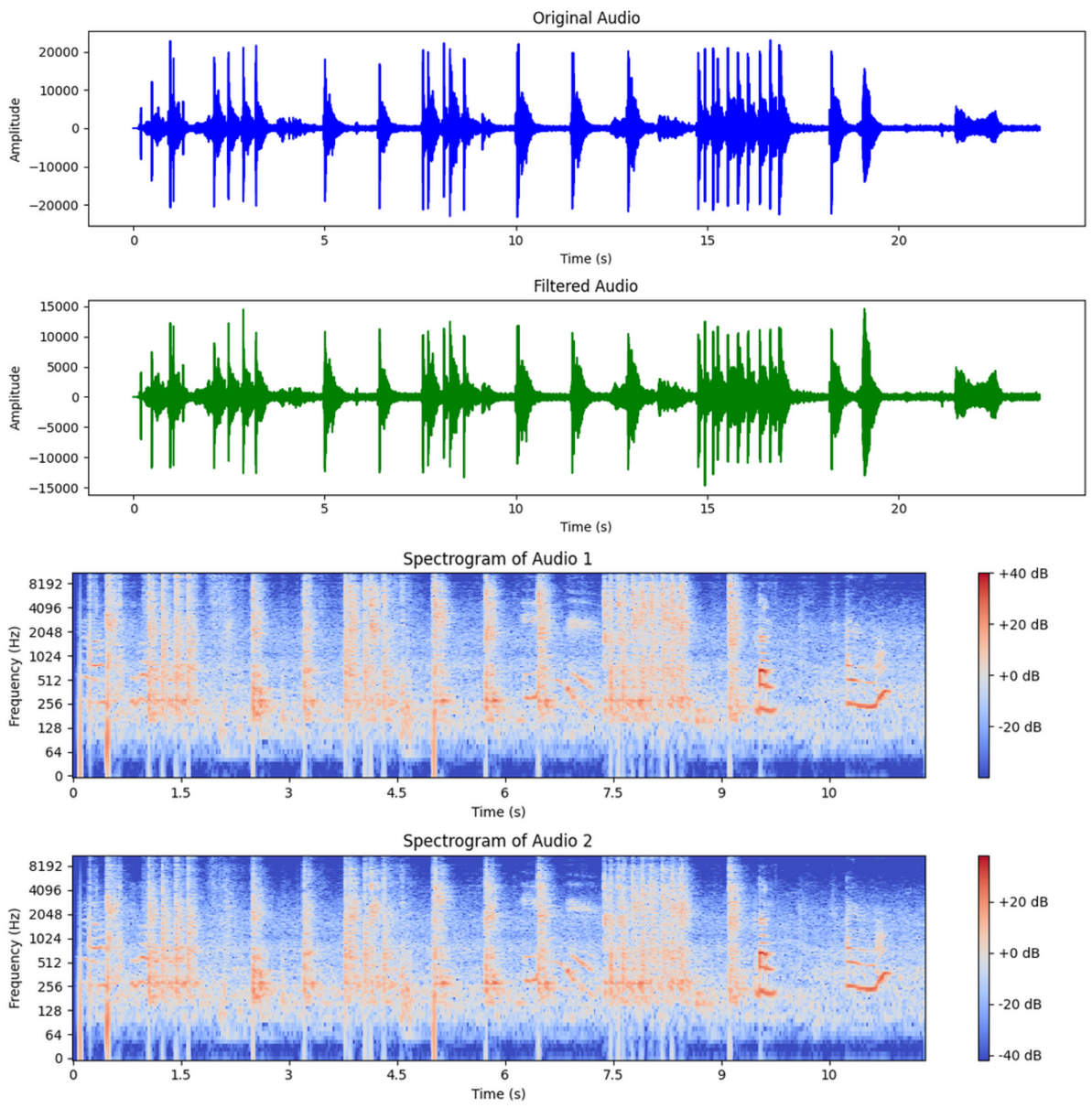


Python code to generate Plot for noise reduction

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from pydub import AudioSegment

original_audio = AudioSegment.from_wav("normalized_cheering.wav")

original_audio_array = np.array(original_audio.get_array_of_samples())

sample_rate = original_audio.frame_rate
time = np.linspace(0, len(original_audio_array) / sample_rate, num=len(original_audio_array))

filtered_audio = AudioSegment.from_wav("filtered_audio.wav")

filtered_audio_array = np.array(filtered_audio.get_array_of_samples())

filtered_time = np.linspace(0, len(filtered_audio_array) / sample_rate, num=len(filtered_audio_array))

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.title("Original Audio")
plt.plot(time, original_audio_array, color='b')
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

plt.subplot(2, 1, 2)
plt.title("Filtered Audio")
plt.plot(filtered_time, filtered_audio_array, color='g')
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

plt.tight_layout()
plt.show()
```

source:http://pydub.com/

Plot before vs after noise reduction




```python
import librosa
import librosa.display
import matplotlib.pyplot as plt
file1 = "output_audio.wav"
audio1, sample_rate1 = librosa.load(file1)
file2 = "filtered_audio.wav"
audio2, sample_rate2 = librosa.load(file2)
stft1 = librosa.stft(audio1)
stft2 = librosa.stft(audio2)
magnitude1 = librosa.amplitude_to_db(abs(stft1))
magnitude2 = librosa.amplitude_to_db(abs(stft2))

time1 = librosa.times_like(magnitude1)
time2 = librosa.times_like(magnitude2)

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
librosa.display.specshow(magnitude1, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram of Audio 1')
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')

plt.subplot(2, 1, 2)
librosa.display.specshow(magnitude2, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram of Audio 2')
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')

plt.tight_layout()
plt.show()
```

Python code to normalize audio

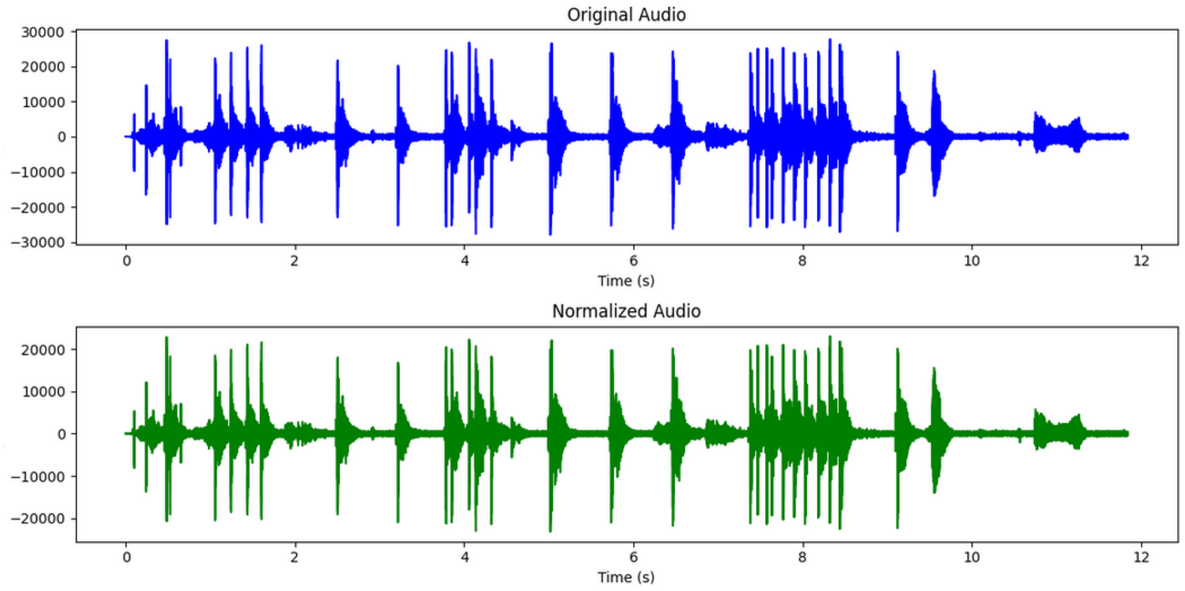
```python
from pydub import AudioSegment

audio = AudioSegment.from_file("output_audio.wav")

target_peak_dBFS = -3

normalized_audio = audio.apply_gain(target_peak_dBFS - audio.max_dBFS)

normalized_audio.export("normalized_cheering.wav", format="wav")
```

A target peak of -3dB is chosen according to the generated outputs when other numbers are selected.
The overall shape of the waveform remains the same, but it's "scaled up" or "scaled down" so that it fits within the desired loudness range.

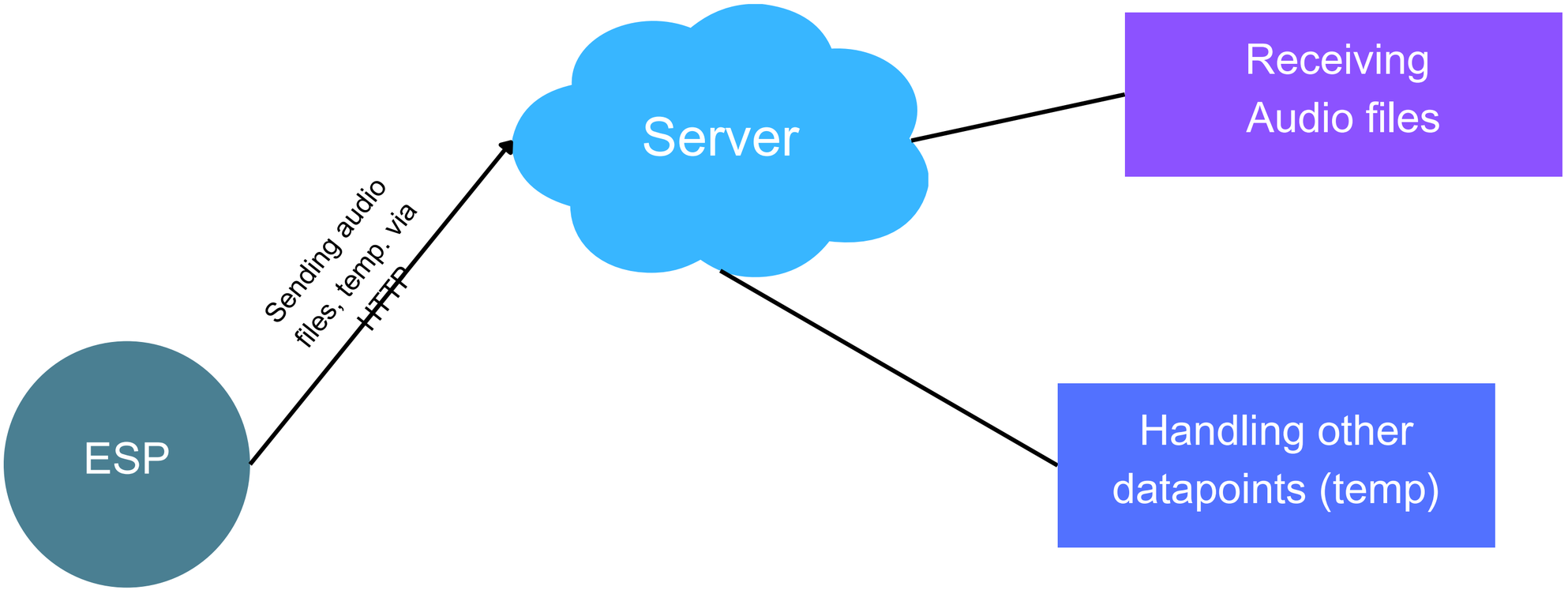
Python code to plot graphs

This code is used to plot the graphs of waveform and normalized waveform
For analysing the difference in wave for before and after normalization.

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile
from pydub import AudioSegment

original_audio = AudioSegment.from_file("output_audio.wav")

normalized_audio = AudioSegment.from_file("normalized_cheering.wav")

sample_rate_orig, audio_data_orig = wavfile.read("output_audio.wav")
sample_rate_norm, audio_data_norm = wavfile.read("normalized_cheering.wav")

time_orig = np.arange(0, len(audio_data_orig)) / sample_rate_orig
time_norm = np.arange(0, len(audio_data_norm)) / sample_rate_norm

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.title("Original Audio")
plt.plot(time_orig, audio_data_orig, color='b')
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

plt.subplot(2, 1, 2)
plt.title("Normalized Audio")
plt.plot(time_norm, audio_data_norm, color='g')
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

plt.tight_layout()
plt.show()
```



Handling Files, data with Server



Audio Files

- We first need to connect to the server through http, so, we need to store the IP address of the server in ESP code.
- Audio files are locally stored in an SD card connected to the ESP via SD card module. Now, these files are stored in ‘/sdcard’ folder in the local system.

- Once recorded, we take the saved audio file from the SD card and send it to server via HTTP client in the ESP code.
- Content-type of the data should be set "octet-stream" as we are dealing with .wav files.

Two ways (to transfer)

Once recorded and save in the SD card, there are two ways we can send the complete file. Either we have to

- Directly send the complete audio file across
- Or split them into smaller chunks (buffers) and send them one-by-one. We end the request after all the chunks are sent.

```
String route = "upload-audio";
http.begin(client, serverName+route);
http.addHeader("File_name", fname);
http.addHeader("Content-Type", "application/octet-stream");
int httpResponseCode = http.POST(buff2, sizeof(buff2));
```

Handling other datapoints

- Similar to what we did in the case of audio files, but here, we use JSON type of data to transfer the datapoints (temperature, etc.)
- Handler functions of these requests to routes are in a python code in the server. (Flask server)
- We store those temperatures in a JSON file in the server itself.

```
String route = "post-data";
String url = serverName+route;
http.begin(client, url);
http.addHeader("Content-Type", "application/json");
int httpResponseCode = http.POST("{\"type\":" + String(type) + ",\"data\":"+String(data));
Serial.println(httpResponseCode);
```

Next Week Work

Wavelet Denoising

Implementing FFT

Spectogram Analysis