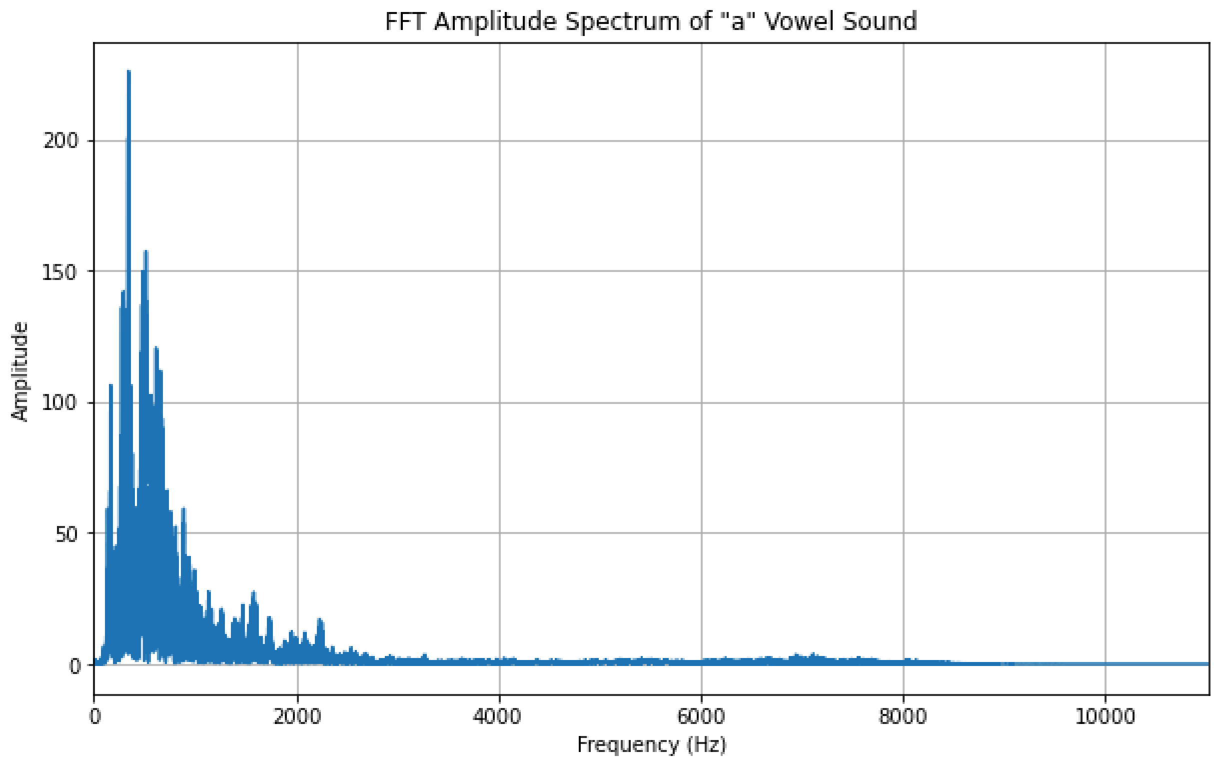
LAB-6

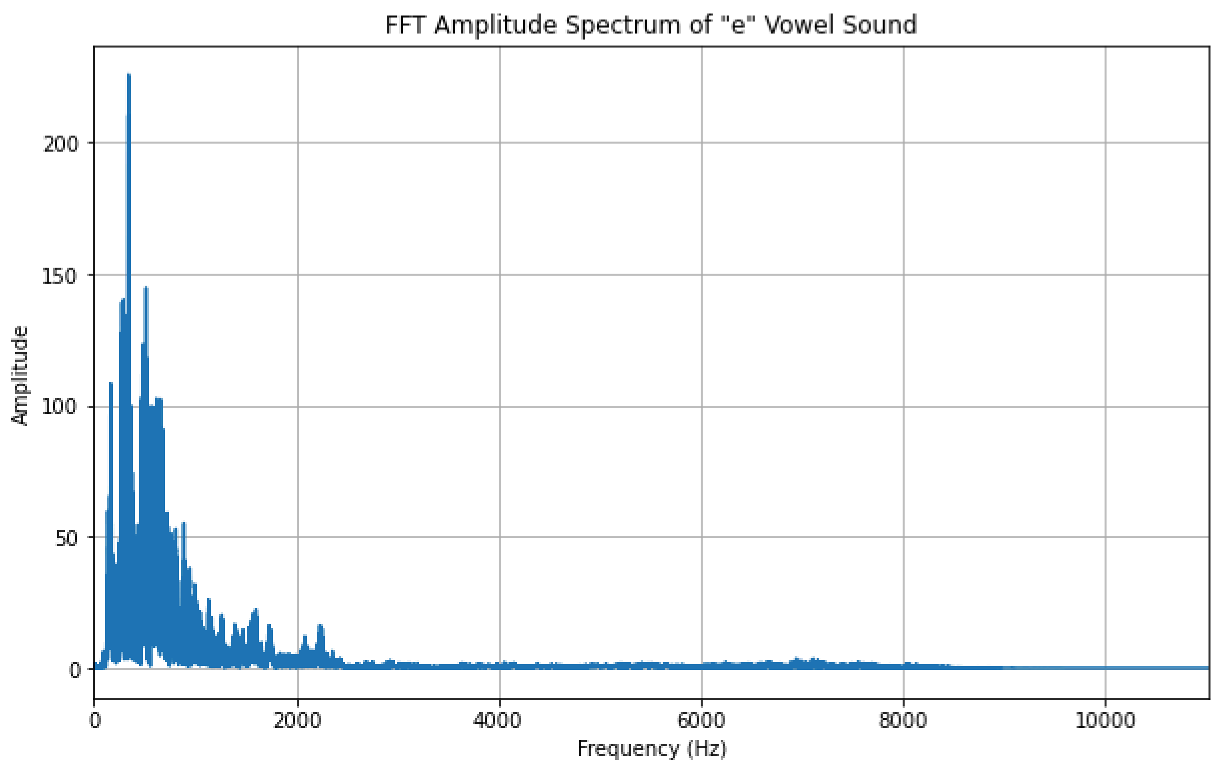
M.HARIVIRINCHI BL.EN.U4AIE21077

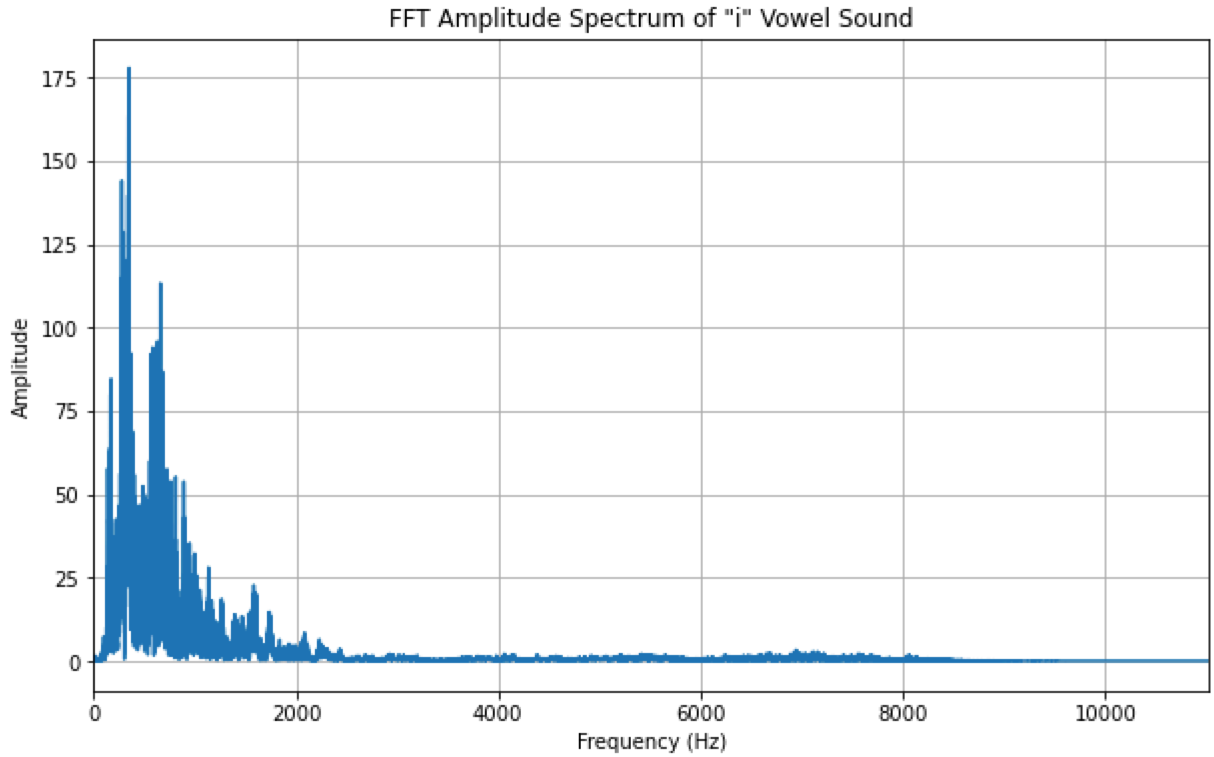
A1.Take a portion of your recorded signal which represents a vowel sound. Perform FFT on the signal snippet and observe the amplitude spectrum. Repeat the same for a few vowelsounds.

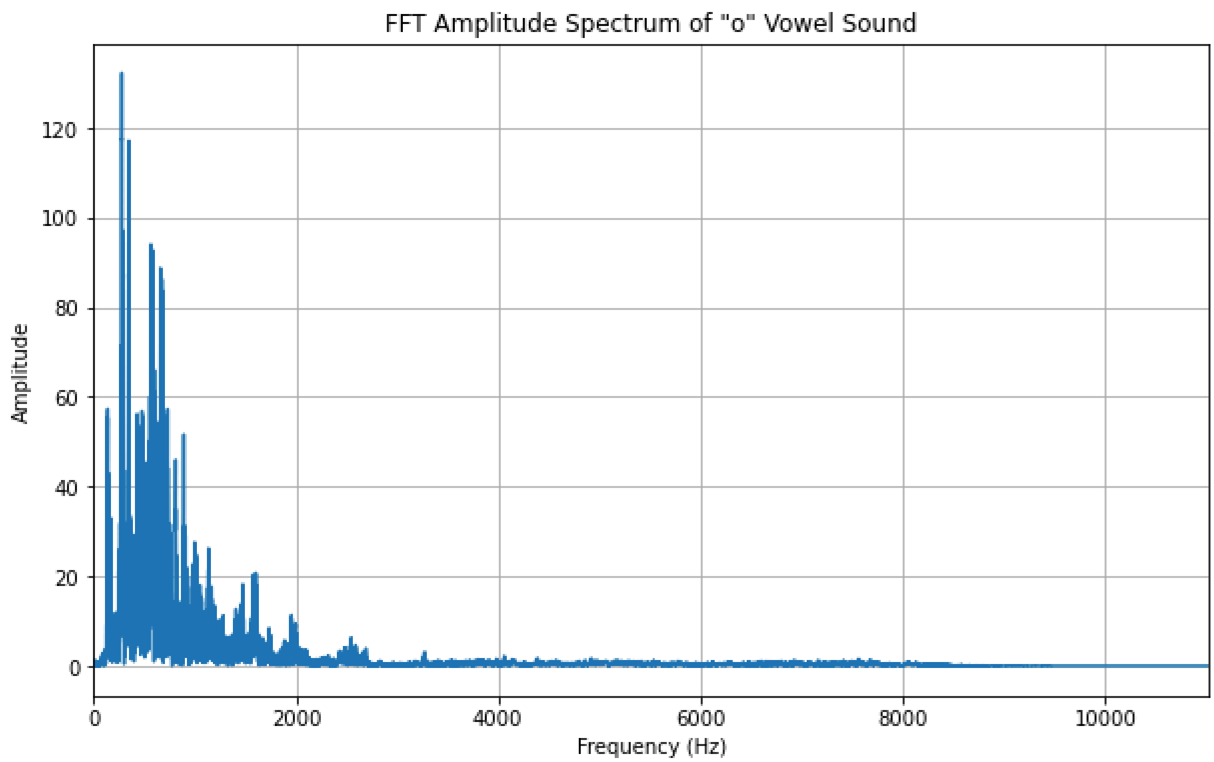
In [8]:
```
import librosa
import matplotlib.pyplot as plt
import numpy as np

def analyze_vowel(audio_file, start_time, end_time, vowel_name):
    """Analyzes a vowel sound from an audio file using FFT and plots the amplitude spe

    Args:
        audio_file (str): Path to the audio file containing the vowel sound.
        start_time (float): Start time (in seconds) of the vowel sound segment.
        end_time (float): End time (in seconds) of the vowel sound segment.
        vowel_name (str): Name of the vowel sound being analyzed (e.g., "a", "e", "i",
    """

    y, sr = librosa.load(audio_file)  # Load audio with sample rate (sr)

    # Extract vowel sound segment
    vowel_segment = y[int(sr * start_time):int(sr * end_time)]

    # Perform FFT
    fft_data = np.fft.fft(vowel_segment)

    # Calculate absolute values for amplitude spectrum
    amplitude_spectrum = np.abs(fft_data)

    # Determine frequencies (half of the spectrum due to Nyquist)
    frequencies = np.linspace(0, sr / 2, len(amplitude_spectrum) // 2 + 1)

    # Plot amplitude spectrum
    plt.figure(figsize=(10, 6))
    plt.plot(frequencies, amplitude_spectrum[:len(frequencies)])
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude')
    plt.title(f'FFT Amplitude Spectrum of "{vowel_name}" Vowel Sound')
    plt.grid(True)
    plt.xlim(0, sr / 2)  # Limit x-axis to display relevant frequencies
    plt.show()

# Example usage (replace with your audio file path and desired vowel segments)
audio_file = "Hari1.wav"  # Modify with your audio file path

# Specify start and end times (in seconds) for each vowel segment
vowel_segments = [
    ("a", 0.5, 5),
    ("e", 1.0, 4),
    ("i", 1.75, 4),
    ("o", 2.5, 5),
    ("u", 3.0, 5),
```

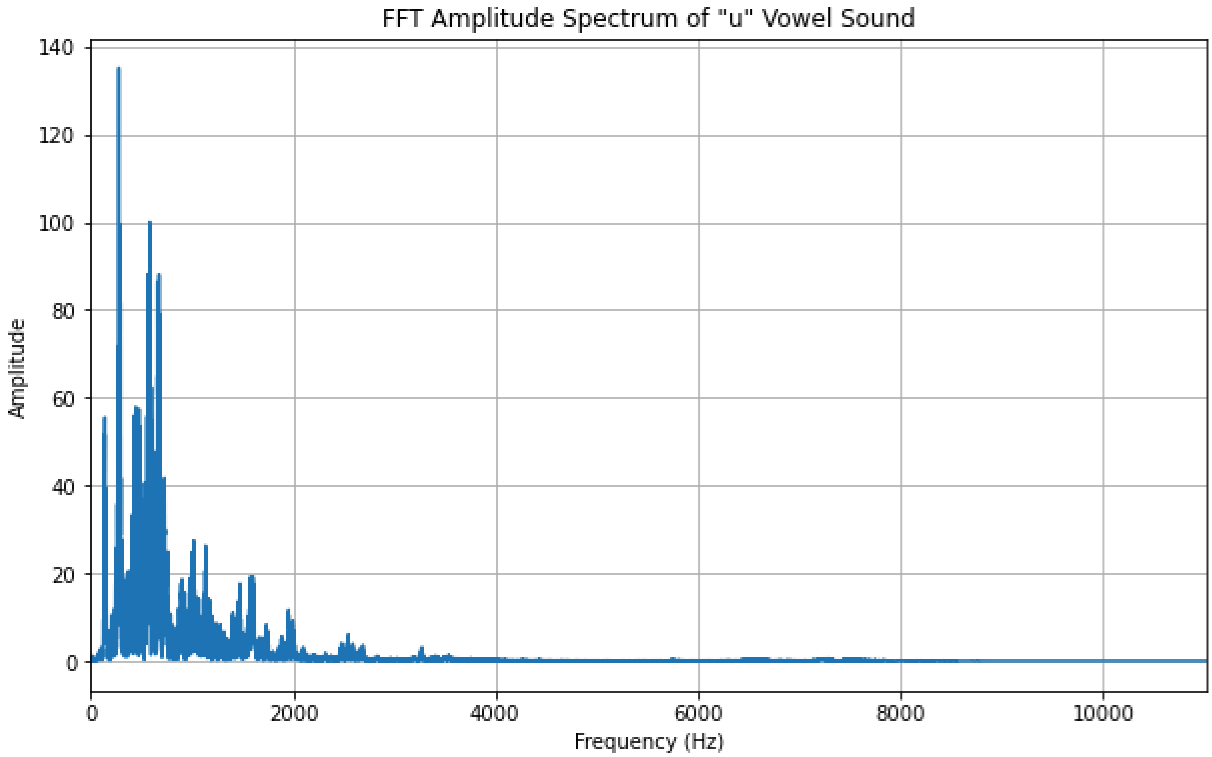
```
]

for vowel_name, start_time, end_time in vowel_segments:
    analyze_vowel(audio_file, start_time, end_time, vowel_name)
```






FFT Amplitude Spectrum of "i" Vowel Sound
Amplitude
Frequency (Hz)
0
25
50
75
100
125
150
175
0
2000
4000
6000
8000
10000

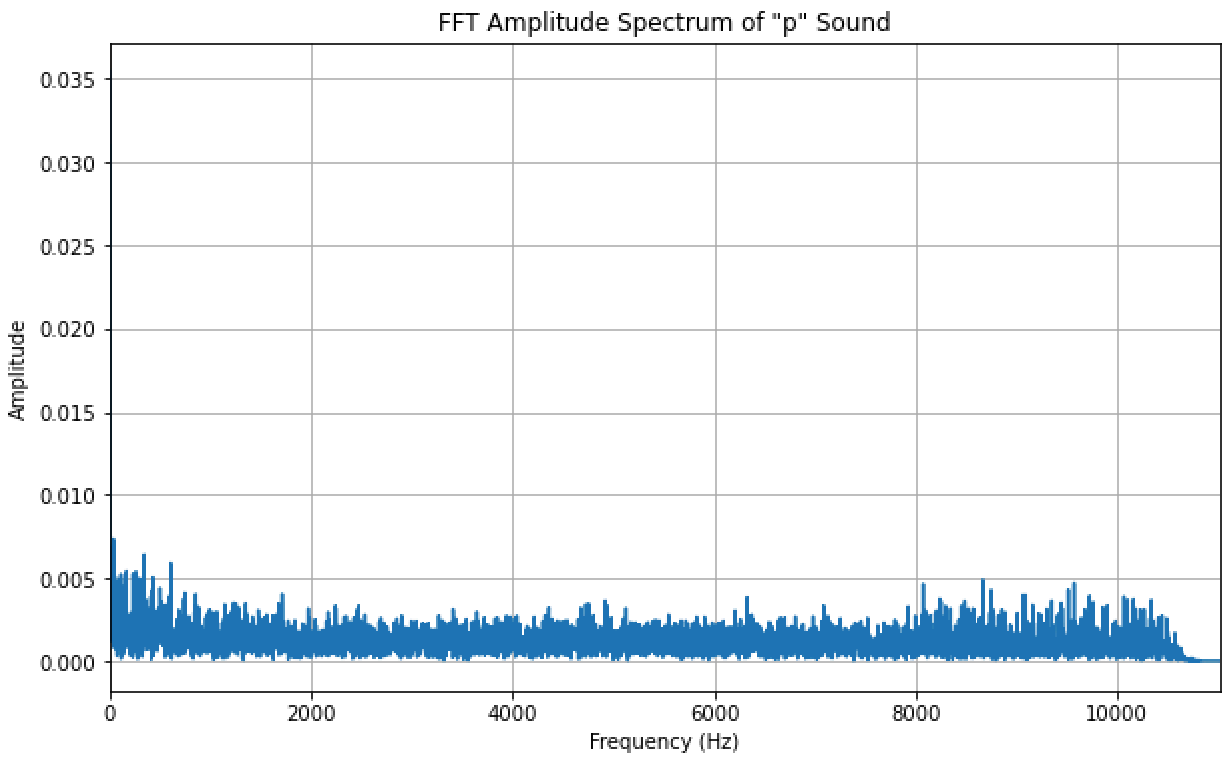

FFT Amplitude Spectrum of "o" Vowel Sound
Amplitude
Frequency (Hz)
0
20
40
60
80
100
120
0
2000
4000
6000
8000
10000

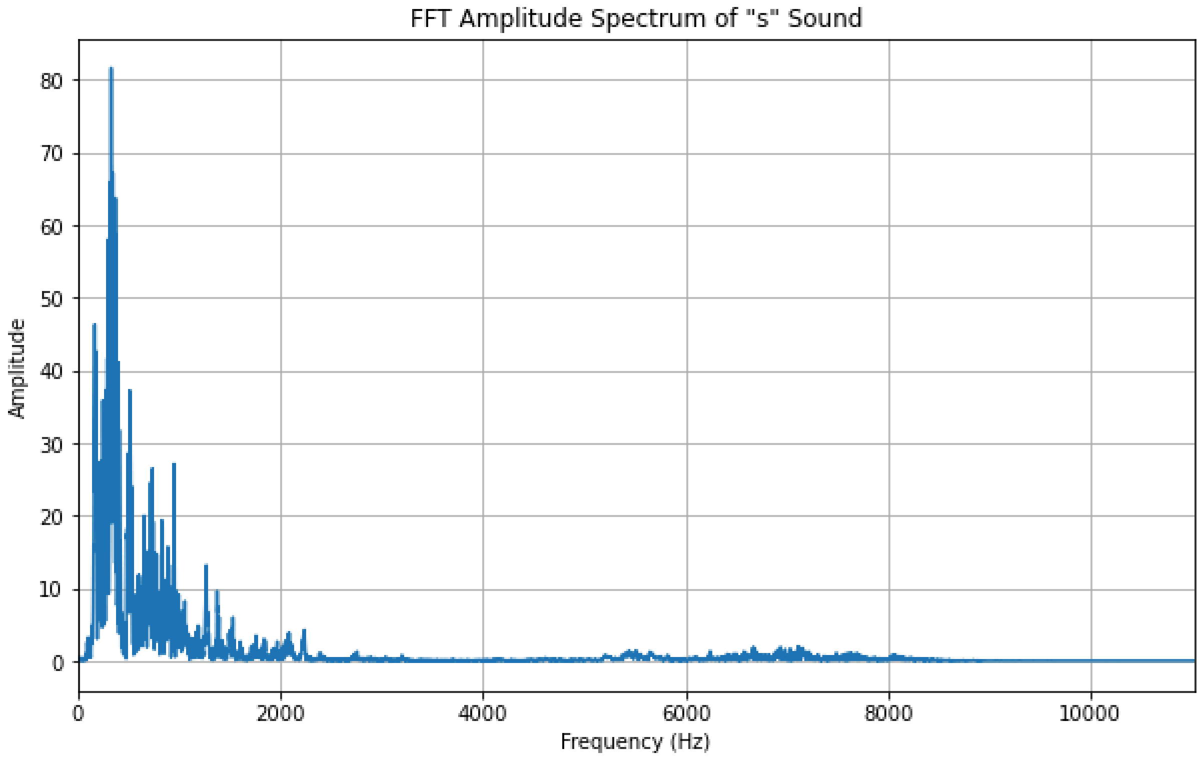


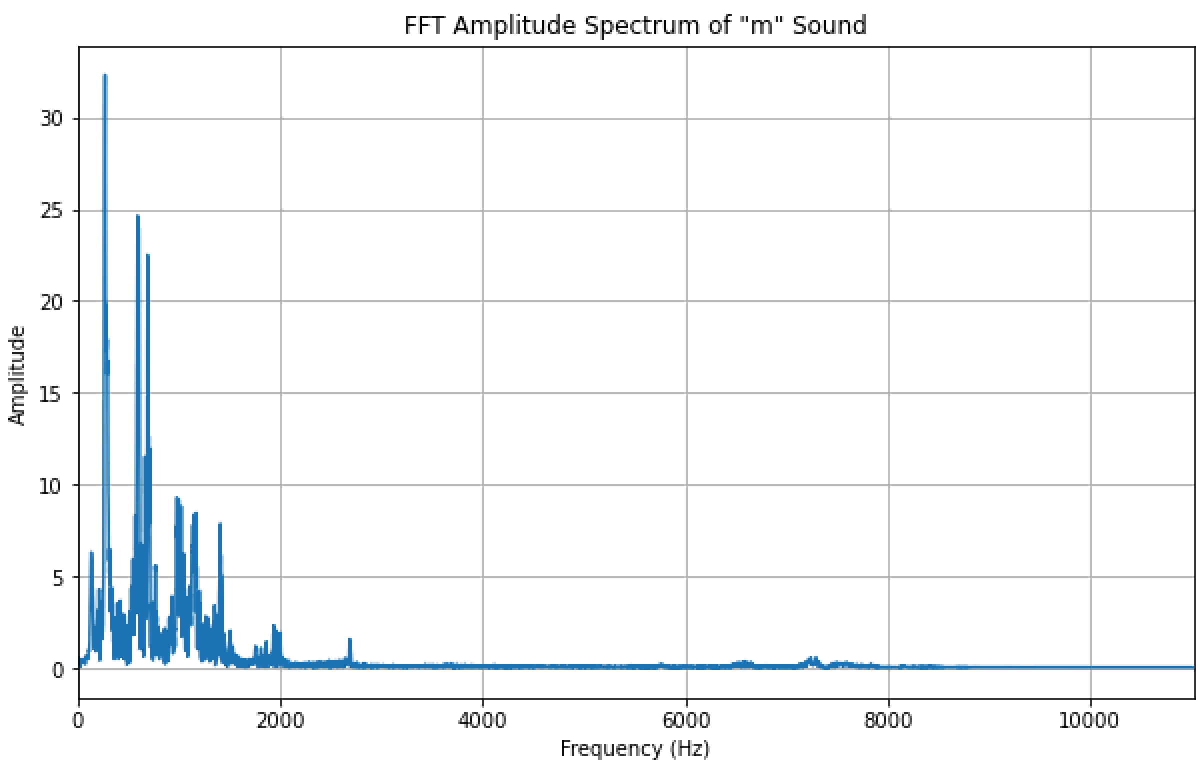
A2. Repeat the A1 for a consonant sound. Perform the same for a few consonant sounds.

In [9]:
```
import librosa
import matplotlib.pyplot as plt
import numpy as np

def analyze_sound(audio_file, start_time, end_time, sound_name):
    """Analyzes a sound segment from an audio file using FFT and plots the amplitude s

    Args:
        audio_file (str): Path to the audio file containing the sound.
        start_time (float): Start time (in seconds) of the sound segment.
        end_time (float): End time (in seconds) of the sound segment.
        sound_name (str): Name of the sound being analyzed (vowel or consonant).
    """

    y, sr = librosa.load(audio_file)  # Load audio with sample rate (sr)

    # Extract sound segment
    sound_segment = y[int(sr * start_time):int(sr * end_time)]

    # Perform FFT
    fft_data = np.fft.fft(sound_segment)

    # Calculate absolute values for amplitude spectrum
    amplitude_spectrum = np.abs(fft_data)

    # Determine frequencies (half of the spectrum due to Nyquist)
    frequencies = np.linspace(0, sr / 2, len(amplitude_spectrum) // 2 + 1)

    # Plot amplitude spectrum
    plt.figure(figsize=(10, 6))
    plt.plot(frequencies, amplitude_spectrum[:len(frequencies)])
    plt.xlabel('Frequency (Hz)')
```

```
    plt.ylabel('Amplitude')
    plt.title(f'FFT Amplitude Spectrum of "{sound_name}" Sound')
    plt.grid(True)
    plt.xlim(0, sr / 2)  # Limit x-axis to display relevant frequencies
    plt.show()

# Example usage (replace with your audio file path and desired sound segments)
audio_file = "Hari1.wav"  # Modify with your audio file path

# Specify start and end times (in seconds) for each consonant segment
consonant_segments = [
    ("p", 0.5, 1.0),  # Plosive consonant (typically has broadband noise)
    ("s", 2.0, 2.5),  # Fricative consonant (often has high-frequency energy)
    ("m", 3.5, 4.0),  # Nasal consonant (may show formants around 200-400 Hz)
]

for sound_name, start_time, end_time in consonant_segments:
    analyze_sound(audio_file, start_time, end_time, sound_name)
```





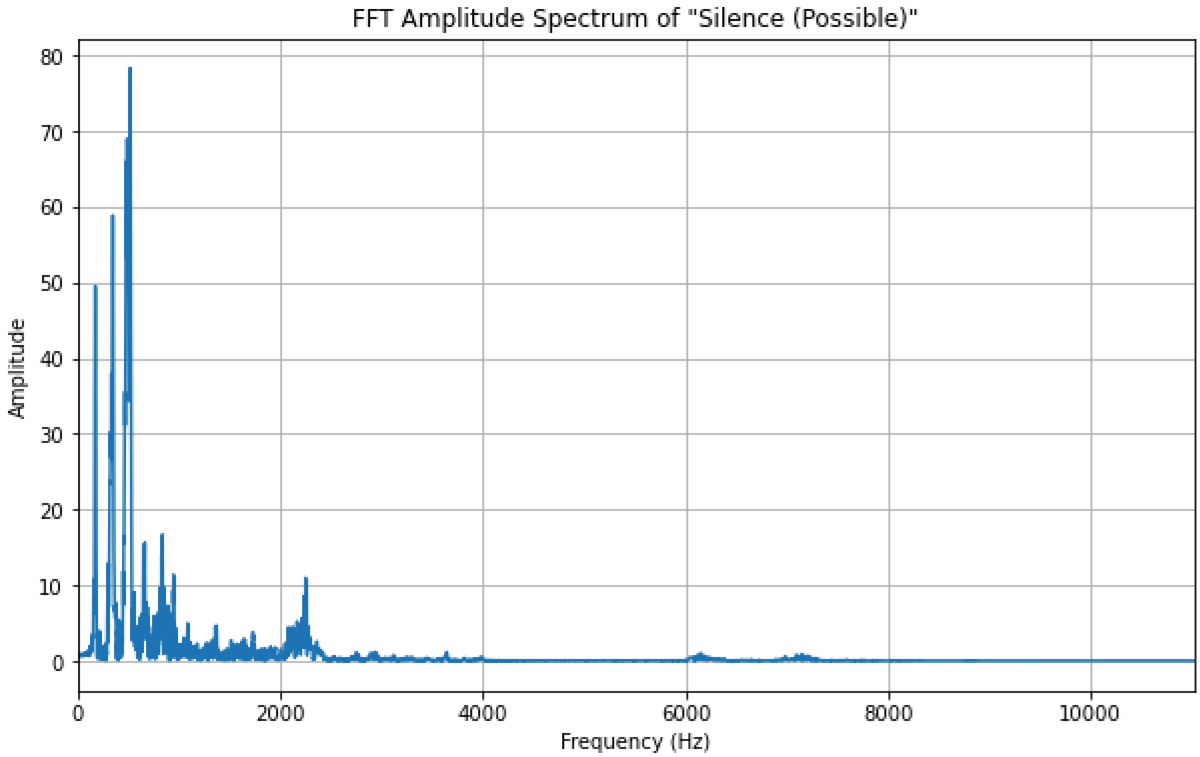


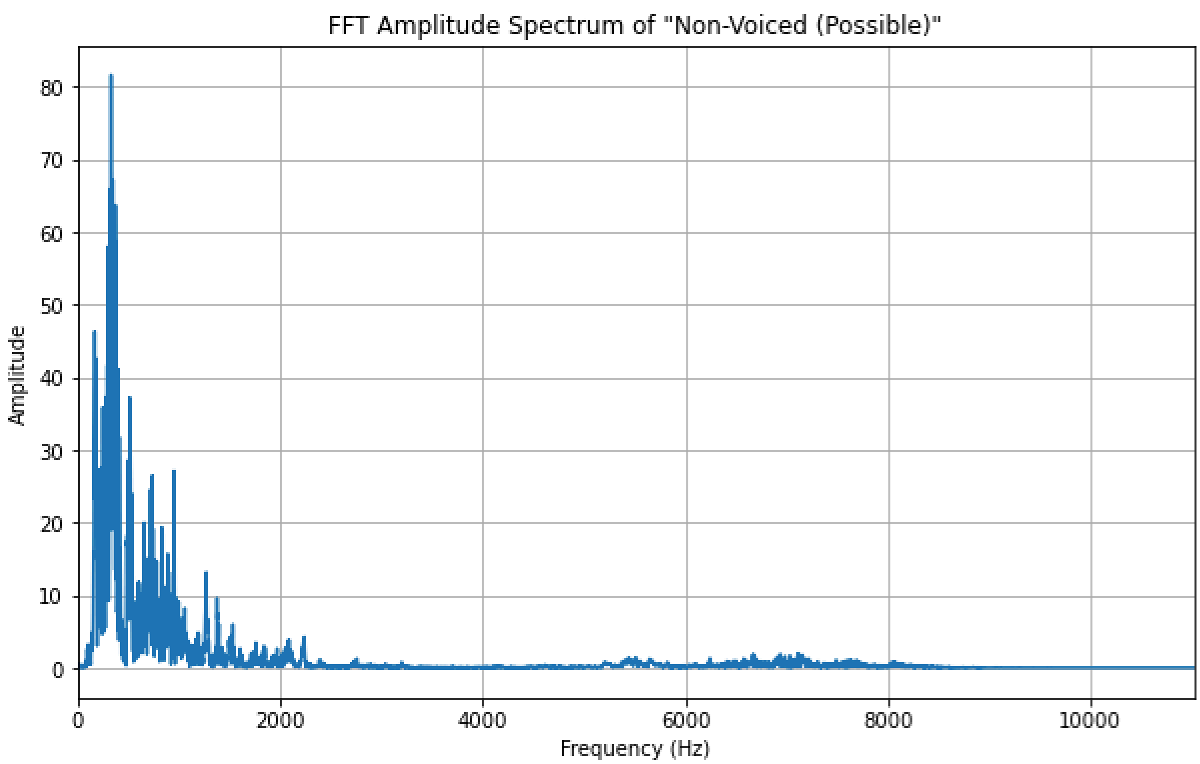
A3. Repeat A2 for few slices of silence & non-voiced portions of the recorded speech signal.

In [10]:

```
import librosa
import matplotlib.pyplot as plt
import numpy as np

def analyze_sound(audio_file, start_time, end_time, sound_name):
    """Analyzes a sound segment from an audio file using FFT and plots the amplitude s

    Args:
```

```
        audio_file (str): Path to the audio file containing the sound.
        start_time (float): Start time (in seconds) of the sound segment.
        end_time (float): End time (in seconds) of the sound segment.
        sound_name (str): Name of the sound being analyzed (vowel, consonant, silence,
    """

    y, sr = librosa.load(audio_file)  # Load audio with sample rate (sr)

    # Extract sound segment
    sound_segment = y[int(sr * start_time):int(sr * end_time)]

    # Perform FFT
    fft_data = np.fft.fft(sound_segment)

    # Calculate absolute values for amplitude spectrum
    amplitude_spectrum = np.abs(fft_data)

    # Determine frequencies (half of the spectrum due to Nyquist)
    frequencies = np.linspace(0, sr / 2, len(amplitude_spectrum) // 2 + 1)

    # Plot amplitude spectrum
    plt.figure(figsize=(10, 6))
    plt.plot(frequencies, amplitude_spectrum[:len(frequencies)])
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude')
    plt.title(f'FFT Amplitude Spectrum of "{sound_name}"')
    plt.grid(True)
    plt.xlim(0, sr / 2)  # Limit x-axis to display relevant frequencies
    plt.show()

# Example usage (replace with your audio file path)
audio_file = "Hari1.wav"  # Modify with your audio file path

# Identify potential silence and non-voiced portions (adapt based on your audio)
possible_silence = (0.5, 1.5)  # Adjust start and end times for a suspected silent seg
possible_non_voiced = (2.0, 2.5)  # Adjust start and end times for a suspected non-voi

# Analyze silence and non-voiced portions
analyze_sound(audio_file, *possible_silence, sound_name="Silence (Possible)")
analyze_sound(audio_file, *possible_non_voiced, sound_name="Non-Voiced (Possible)")

# Additional considerations
print("**Note:**")
print("- Actual silence may have very low amplitude across frequencies.")
print("- Non-voiced speech may have some low-level energy present.")
print("- Consider using voice activity detection (VAD) for more accurate segmentation.
```

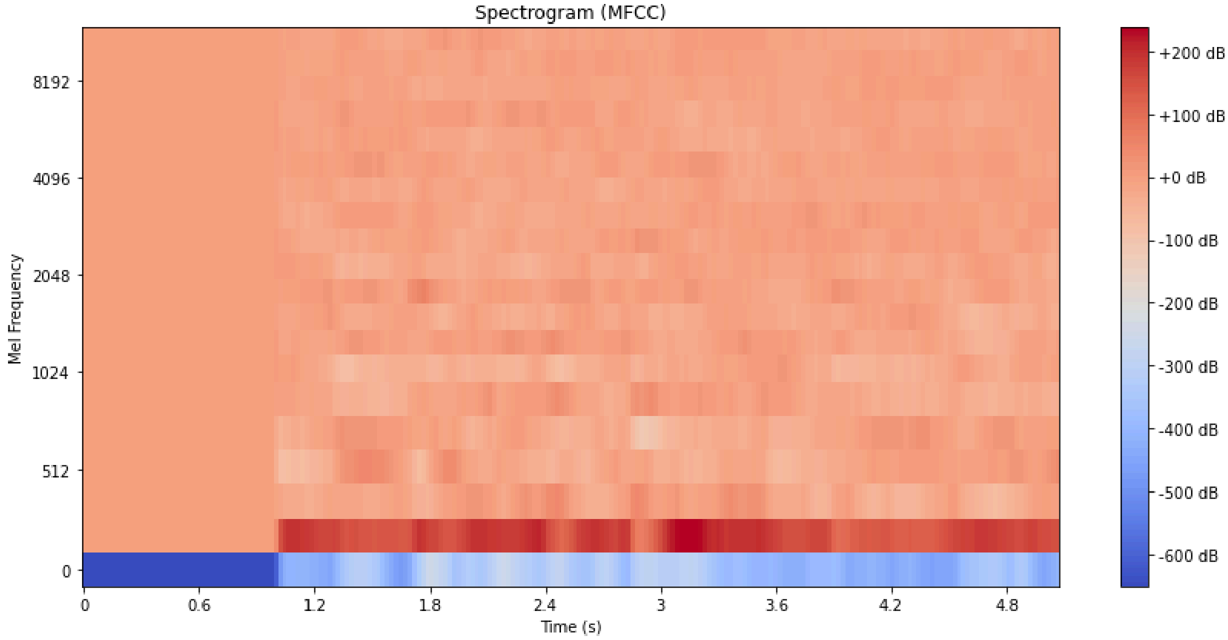





```
**Note:**
- Actual silence may have very low amplitude across frequencies.
- Non-voiced speech may have some low-level energy present.
- Consider using voice activity detection (VAD) for more accurate segmentation.
```

A4. Now you have acquainted yourself with spectral amplitudes of various consonants and vowel-basedphonemes. Generate the spectrogram of thesignal and observe the change points of the signals with associatedspeech segments. Observe to identify the consonants and vowels from the spectrogram.

In [11]:
```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

def generate_spectrogram(audio_file):
    """Generates a spectrogram of the audio signal and attempts to identify vowel-to-c

    Args:
        audio_file (str): Path to the audio file containing the speech signal.

    Returns:
        None
    """

    y, sr = librosa.load(audio_file)  # Load audio with sample rate (sr)

    # Compute spectrogram using Mel-frequency cepstral coefficients (MFCCs)
    spectrogram = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)

    # Generate spectrogram visualization (adjust x/y labels as needed)
    plt.figure(figsize=(12, 6))
    librosa.display.specshow(spectrogram, sr=sr, x_axis='time', y_axis='mel')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Spectrogram (MFCC)')
    plt.xlabel('Time (s)')
    plt.ylabel('Mel Frequency')
    plt.tight_layout()
    plt.show()

    # Identify potential vowel-to-consonant transitions (heuristics based on intensity
    average_intensity = np.mean(spectrogram)
    transition_points = []
    for i in range(1, spectrogram.shape[1]):
        if spectrogram[0, i] < average_intensity and spectrogram[0, i - 1] > average_i
            transition_points.append(i * (1 / sr))  # Convert frame index to time in s
        elif spectrogram[0, i] > average_intensity and spectrogram[0, i - 1] < average
            transition_points.append(i * (1 / sr))

    # Print potential consonant and vowel segments (heuristic identification)
    if transition_points:
        print("**Potential Consonant-Vowel Transitions (Times in seconds):")
        for i, point in enumerate(transition_points):
            if i % 2 == 0:  # Even indices likely indicate consonant starts
                print(f"- Consonant start at {point:.2f}")
            else:  # Odd indices likely indicate vowel starts
                print(f"- Vowel start at {point:.2f}")
        print("**Note:** This is a heuristic approach based on intensity. More advance
    else:
        print("No clear transitions detected.")

# Example usage (replace with your audio file path)
audio_file = "Hari1.wav"  # Modify with your audio file path
generate_spectrogram(audio_file)
```




```
No clear transitions detected.
```