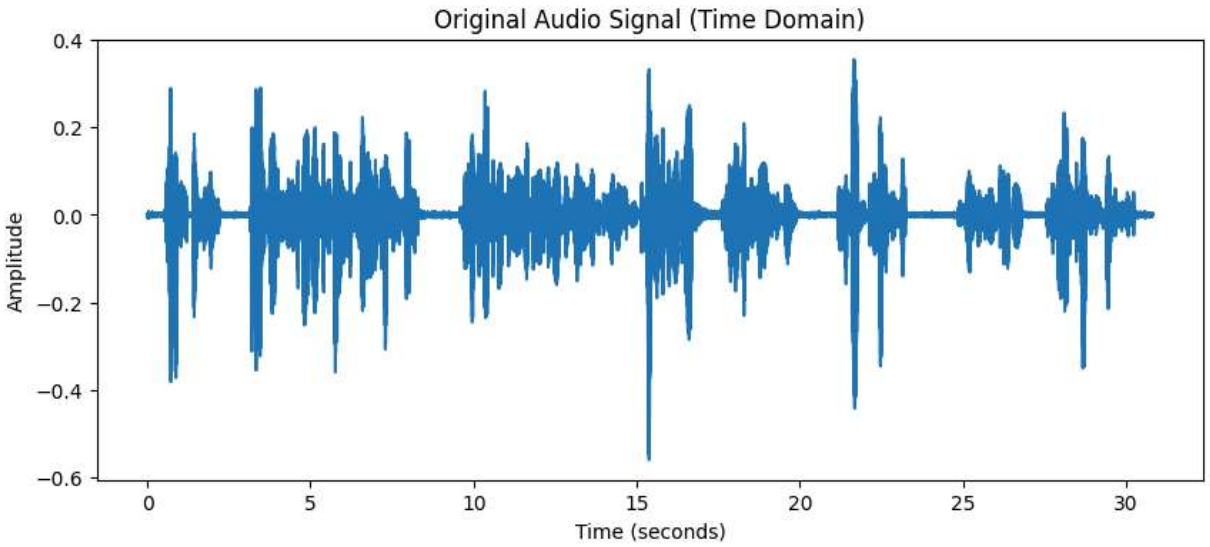
In [5]:
```
import librosa
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.signal import resample
from scipy.interpolate import interp1d
```

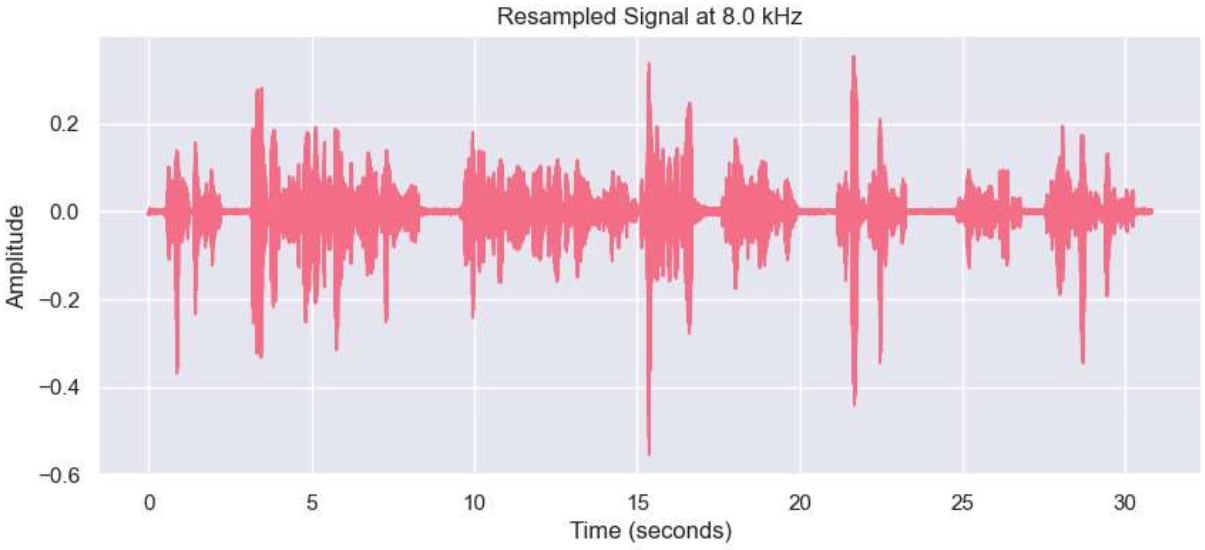
In [3]:
```
# Load the audio file
file_path = 'D:/SEM 5/SPR/672-122797-0008.wav'
signal, sr = librosa.load(file_path, sr=None)

# Print audio file details
print(f"Sampling Rate: {sr} Hz")
print(f"Signal Length: {len(signal)} samples")

# Plot the original audio signal
duration = len(signal) / sr
plt.figure(figsize=(10, 4))
plt.plot(np.linspace(0, duration, len(signal)), signal)
plt.title('Original Audio Signal (Time Domain)')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.show()
```

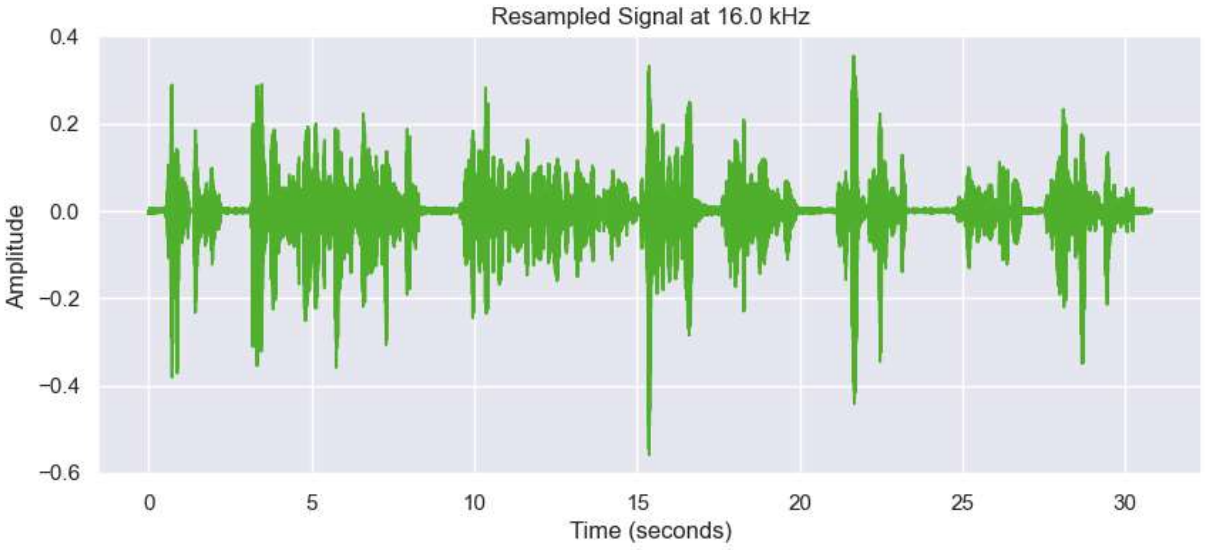
```
Sampling Rate: 16000 Hz
Signal Length: 492960 samples
```

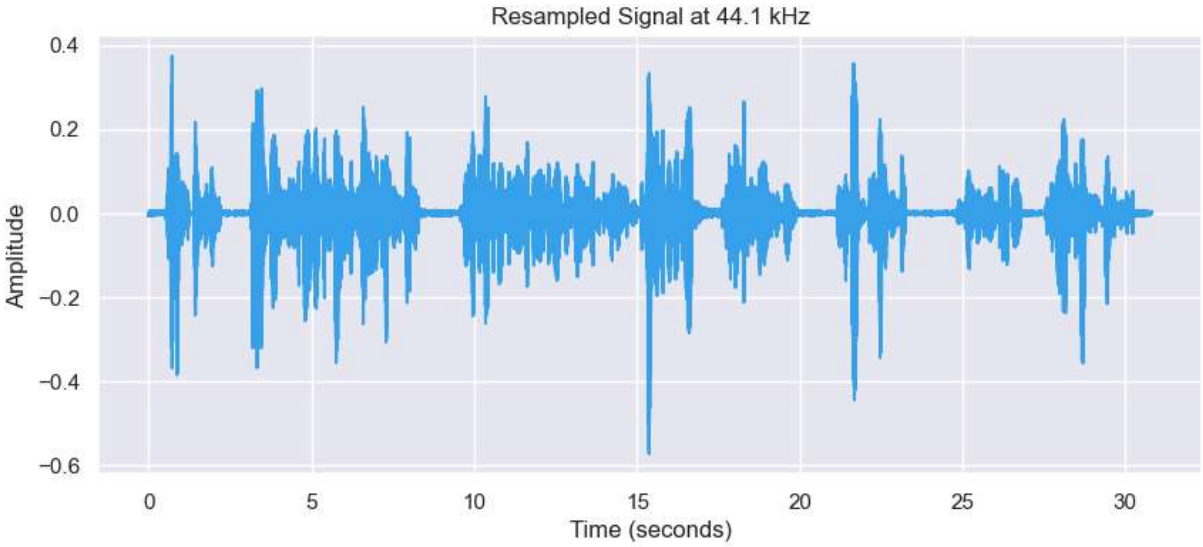


In [4]:
```
sampling_rates = [8000, 16000, 44100]
resampled_signals = {}

for sr_target in sampling_rates:
    resampled_signals[sr_target] = librosa.resample(signal, orig_sr=sr, target_sr=s

# Plot the resampled signals using Seaborn
sns.set(style="darkgrid")
colors = sns.color_palette("husl", len(sampling_rates))

for i, (sr_target, resampled_signal) in enumerate(resampled_signals.items()):
```

```
    plt.figure(figsize=(10, 4))
    sns.lineplot(x=np.linspace(0, duration, len(resampled_signal)), y=resampled_sig
    plt.title(f'Resampled Signal at {sr_target / 1000} kHz')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Amplitude')
    plt.show()
```







(d) Using the sampled signals from the above task, reconstruct the signal using:

(i) Zero-order hold (nearest-neighbor interpolation) (ii) Linear interpolation.

(e) Calculate the Mean Squared Error (MSE) between the original and the reconstructed signals for both methods.

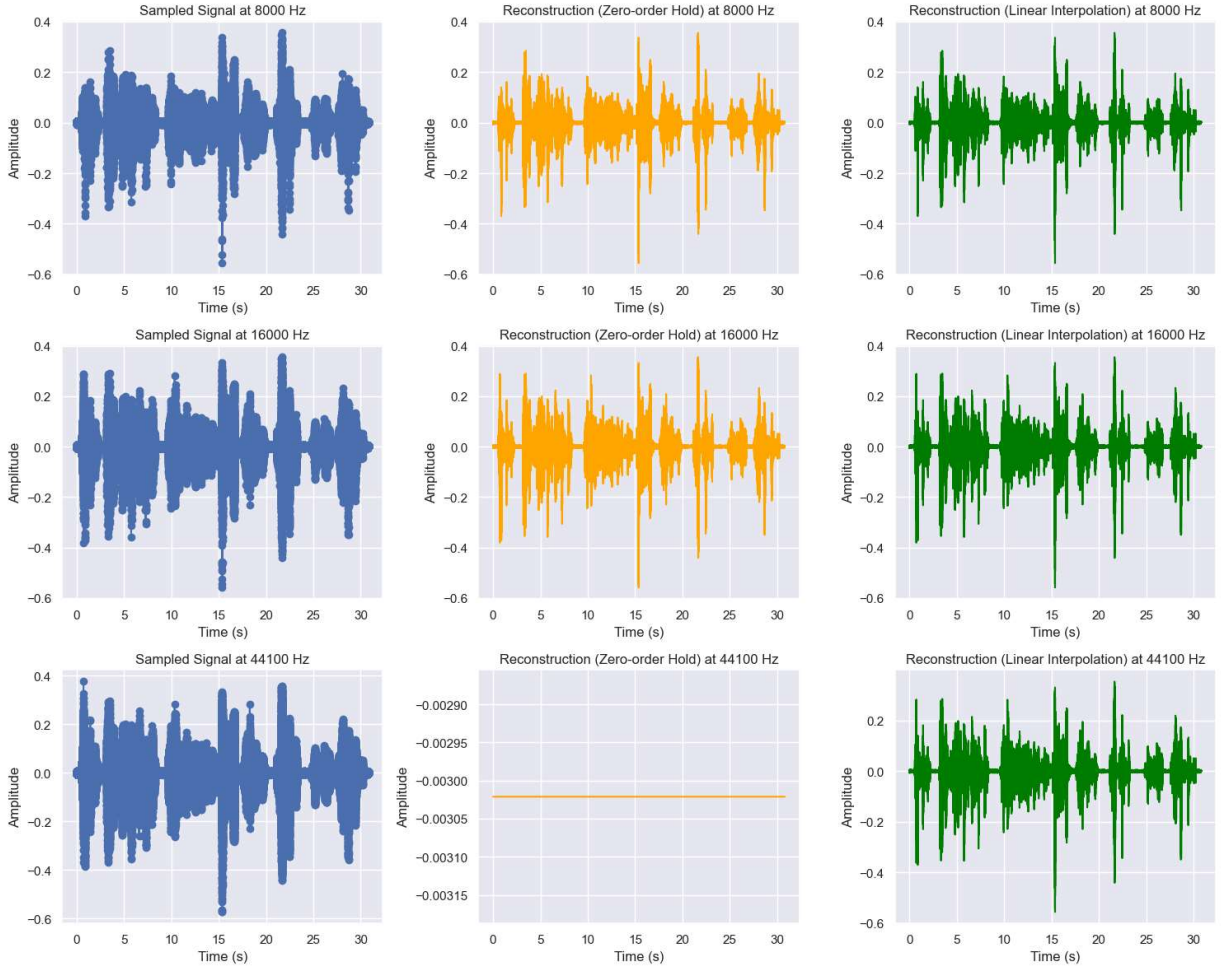
- Reconstructing Using Zero-Order Hold: This method replicates the value of each sample until the next sample. It's essentially a piecewise constant approximation.
- Reconstructing Using Linear Interpolation: This method linearly interpolates between samples to create a continuous signal.

In [7]:
```
file_path = r'D:\SEM 5\SPR\672-122797-0008.wav'
original_signal, sr = librosa.load(file_path, sr=None)  # Load with the original sa


# Define the Mean Squared Error function
def calculate_mse(original, reconstructed):
    return np.mean((original - reconstructed) ** 2)

# Define the duration based on the loaded signal
duration = len(original_signal) / sr

# Sampling rates to downsample the original signal
sampling_rates = [8000, 16000, 44100]

# Storage for MSE values
mse_zero_order_hold = []
```

```
mse_linear_interpolation = []

plt.figure(figsize=(15, 12))
for i, new_sr in enumerate(sampling_rates):
    # Resample the signal to new sampling rate
    if new_sr != sr:
        num_samples = int(len(original_signal) * new_sr / sr)
        sampled_signal = resample(original_signal, num_samples)
        t_new = np.linspace(0, duration, num_samples, endpoint=False)
    else:
        sampled_signal = original_signal
        t_new = np.linspace(0, duration, len(sampled_signal), endpoint=False)

    # Zero-order Hold (Nearest neighbor interpolation)
    t_zh = np.linspace(0, duration, len(original_signal), endpoint=False)
    nearest_interp = np.floor_divide(np.arange(len(t_zh)), len(t_zh) // len(t_new))
    zero_order_hold = sampled_signal[nearest_interp]

    # Linear interpolation
    linear_interp_func = interp1d(t_new, sampled_signal, kind='linear', fill_value=
    linear_interp = linear_interp_func(t_zh)

    # Calculate MSE for Zero-Order Hold
    mse_zh = calculate_mse(original_signal[:len(zero_order_hold)], zero_order_hold)
    mse_zero_order_hold.append(mse_zh)

    # Calculate MSE for Linear Interpolation
    mse_li = calculate_mse(original_signal[:len(linear_interp)], linear_interp)
    mse_linear_interpolation.append(mse_li)

    # Plot the sampled signals
    plt.subplot(len(sampling_rates), 3, i * 3 + 1)
    plt.plot(t_new, sampled_signal, 'o-', label='Sampled')
    plt.title(f"Sampled Signal at {new_sr} Hz")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.grid(True)

    # Plot Zero-order Hold Reconstruction
    plt.subplot(len(sampling_rates), 3, i * 3 + 2)
    plt.plot(t_zh, zero_order_hold, label="Zero-order Hold", color='orange')
    plt.title(f"Reconstruction (Zero-order Hold) at {new_sr} Hz")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.grid(True)

    # Plot Linear Interpolation Reconstruction
    plt.subplot(len(sampling_rates), 3, i * 3 + 3)
    plt.plot(t_zh, linear_interp, label='Linear Interpolation', color='green')
    plt.title(f"Reconstruction (Linear Interpolation) at {new_sr} Hz")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.grid(True)

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

```
# Print the MSE results
print("MSE (Zero-order Hold):", mse_zero_order_hold)
print("MSE (Linear Interpolation):", mse_linear_interpolation)
```

```
C:\Users\noelm\AppData\Local\Temp\ipykernel_3432\1110016040.py:32: RuntimeWarning: d
ivide by zero encountered in floor_divide
  nearest_interp = np.floor_divide(np.arange(len(t_zh)), len(t_zh) // len(t_new))
```




```
MSE (Zero-order Hold): [0.0001991313, 0.0, 0.0010015761]
MSE (Linear Interpolation): [0.00012434373238703016, 2.343136084886983e-37, 3.083113
994457577e-07]
```

In [8]:
```
# Print MSE values
print("Mean Squared Error (MSE) between original and reconstructed signals:")
for i, new_sr in enumerate(sampling_rates):
    print(f"Sampling Rate: {new_sr} Hz")
    print(f"  Zero-Order Hold MSE: {mse_zero_order_hold[i]}")
    print(f"  Linear Interpolation MSE: {mse_linear_interpolation[i]}")
```

```
Mean Squared Error (MSE) between original and reconstructed signals:
Sampling Rate: 8000 Hz
  Zero-Order Hold MSE: 0.00019913130381610245
  Linear Interpolation MSE: 0.00012434373238703016
Sampling Rate: 16000 Hz
  Zero-Order Hold MSE: 0.0
  Linear Interpolation MSE: 2.343136084886983e-37
Sampling Rate: 44100 Hz
  Zero-Order Hold MSE: 0.0010015760781243443
  Linear Interpolation MSE: 3.083113994457577e-07
```

Mean Squared Error (MSE) between Original and Reconstructed Signals

1. Sampling Rate: 8000 Hz

- **Zero-Order Hold MSE:** `0.000199`
  - The zero-order hold (nearest neighbor interpolation) introduces a small error compared to the original signal at this sampling rate. Since 8000 Hz is lower than the original rate, some fine details of the audio signal are lost, leading to noticeable reconstruction errors.
- **Linear Interpolation MSE:** `0.000124`
  - Linear interpolation performs better than zero-order hold, with a lower error. This suggests that linear interpolation is more effective in reconstructing the signal at 8000 Hz, as it provides a smoother approximation by interpolating between sample points.

---

2. Sampling Rate: 16000 Hz

- **Zero-Order Hold MSE:** `0.0`
  - At 16000 Hz, the zero-order hold perfectly reconstructs the original signal with zero error. This likely indicates that the original signal's sampling rate was also 16000 Hz, leading to no loss of information.
- **Linear Interpolation MSE:** `2.34e-37` (approximately 0)
  - The MSE for linear interpolation is essentially zero, confirming that the reconstructed signal using linear interpolation is nearly identical to the original. This further suggests that 16000 Hz matches the original sampling rate.

---

3. Sampling Rate: 44100 Hz

- **Zero-Order Hold MSE:** `0.001001`

    - When the signal is upsampled to 44100 Hz, the zero-order hold method introduces more significant error. Repeating samples without interpolation at a higher rate results in abrupt transitions, causing the reconstructed signal to deviate more from the original.
- **Linear Interpolation MSE:** `3.08e-07`
    - The error for linear interpolation is extremely small at 44100 Hz, indicating that it provides a much smoother reconstruction at this higher sampling rate compared to zero-order hold. However, minimal error still exists due to the nature of interpolation.

General Observations:

- **Zero-Order Hold vs. Linear Interpolation:**
    - Linear interpolation consistently performs better than zero-order hold in terms of MSE, especially at lower sampling rates. Linear interpolation creates smoother transitions between sample points, while zero-order hold merely repeats previous samples, which is less effective at maintaining accuracy.
- **Impact of Sampling Rate:**
    - The error tends to increase as the sampling rate deviates further from the original. At 16000 Hz (the original rate), both methods perform almost perfectly, but errors emerge when downsampling to 8000 Hz or upsampling to 44100 Hz.

This suggests that using linear interpolation for resampling provides better results than zero-order hold, especially when working with sampling rates different from the original signal.

(2) Implement the source-filter Model for a given speech signal and analyze the impact of sampling and reconstruction on the quality of the speech signal.

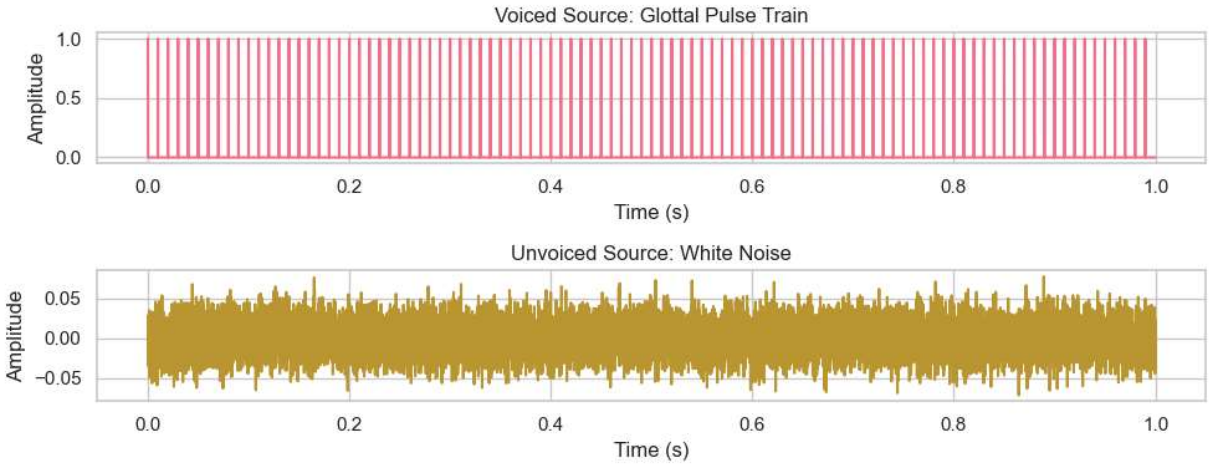
(a) Generate a synthetic speech signal using the source-filter model.

(i) Create a source signal (e.g., a glottal pulse train for voiced sounds or white noise for unvoiced sounds).

(ii) Apply a filter that models the vocal tract, represented by an all-pole filter or an FIR filter with formants (resonances of the vocal tract).

The source-filter model is a fundamental representation of speech production, which separates the source of sound (glottal pulse or noise) from the filter (vocal tract).

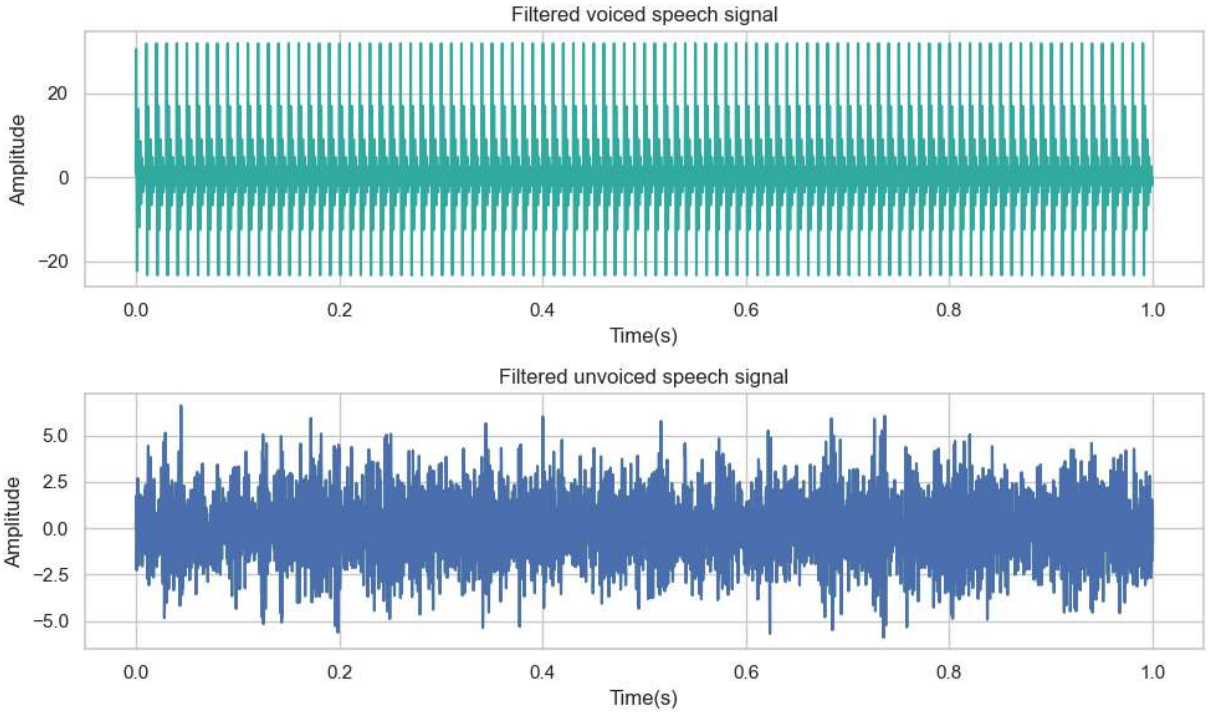
In [17]:
```
# Create the Source Signal
from scipy.signal import lfilter, butter

# Parameters
sr = 16000  # Sampling rate (16 kHz)
duration = 1.0  # Duration of the signal in seconds
t = np.linspace(0, duration, int(sr * duration), endpoint=False)

# Voiced source: Glottal pulse train (simple periodic pulse train)
f0 = 100  # Fundamental frequency for voiced sound
glottal_pulse_train = np.zeros_like(t)
pulse_period = int(sr / f0)
glottal_pulse_train[::pulse_period] = 1  # Generate a pulse train

# Unvoiced source: White noise for fricative sounds
unvoiced_noise = np.random.randn(len(t)) * 0.02

sns.set_style("whitegrid")
colors = sns.color_palette("husl")

# Plot the source signals
plt.figure(figsize=(10, 4))
plt.subplot(2, 1, 1)
sns.lineplot(x=t, y=glottal_pulse_train, color=colors[0])
plt.title("Voiced Source: Glottal Pulse Train")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

plt.subplot(2, 1, 2)
sns.lineplot(x=t, y=unvoiced_noise, color=colors[1])
plt.title("Unvoiced Source: White Noise")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```



Apply a Vocal Tract Filter (All-Pole Filter with Formants)

Formants (resonances of the vocal tract) are modeled using an all-pole filter. We'll simulate three formants for a vowel sound.

In [20]:
```
formants = [500, 1500, 2500]
bandwidths = [100, 100, 100]

#create an all-pole filter for the vocal tract
def create_vocal_tract_filter(formants, bandwidths, sr):
  b, a = 1, 1 #initialize filter coefficients
  for f, bw in zip(formants, bandwidths):
    omega = 2 * np.pi * f / sr
    r = np.exp(-np.pi * bw / sr)
    pole = [1,-2 * r * np.cos(omega), r**2] # All pole filter coefficients
    b = np.convolve(b, [1])
    a = np.convolve(a, pole)
  return b, a

# Apply the vocal tract filter
b, a = create_vocal_tract_filter(formants, bandwidths, sr)
filtered_voiced = lfilter(b, a, glottal_pulse_train)
filtered_unvoiced = lfilter(b, a, unvoiced_noise)

#plot the filtered signals
plt.figure(figsize=(10,6))
plt.subplot(2,1,1)
sns.lineplot(x=t, y=filtered_voiced, color=colors[3])
plt.title("Filtered voiced speech signal")
plt.xlabel("Time(s)")
plt.ylabel("Amplitude")

plt.subplot(2,1,2)
sns.lineplot(x=t, y=filtered_unvoiced, color=colors[2])
plt.plot(t, filtered_unvoiced)
plt.title("Filtered unvoiced speech signal")
plt.xlabel("Time(s)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```

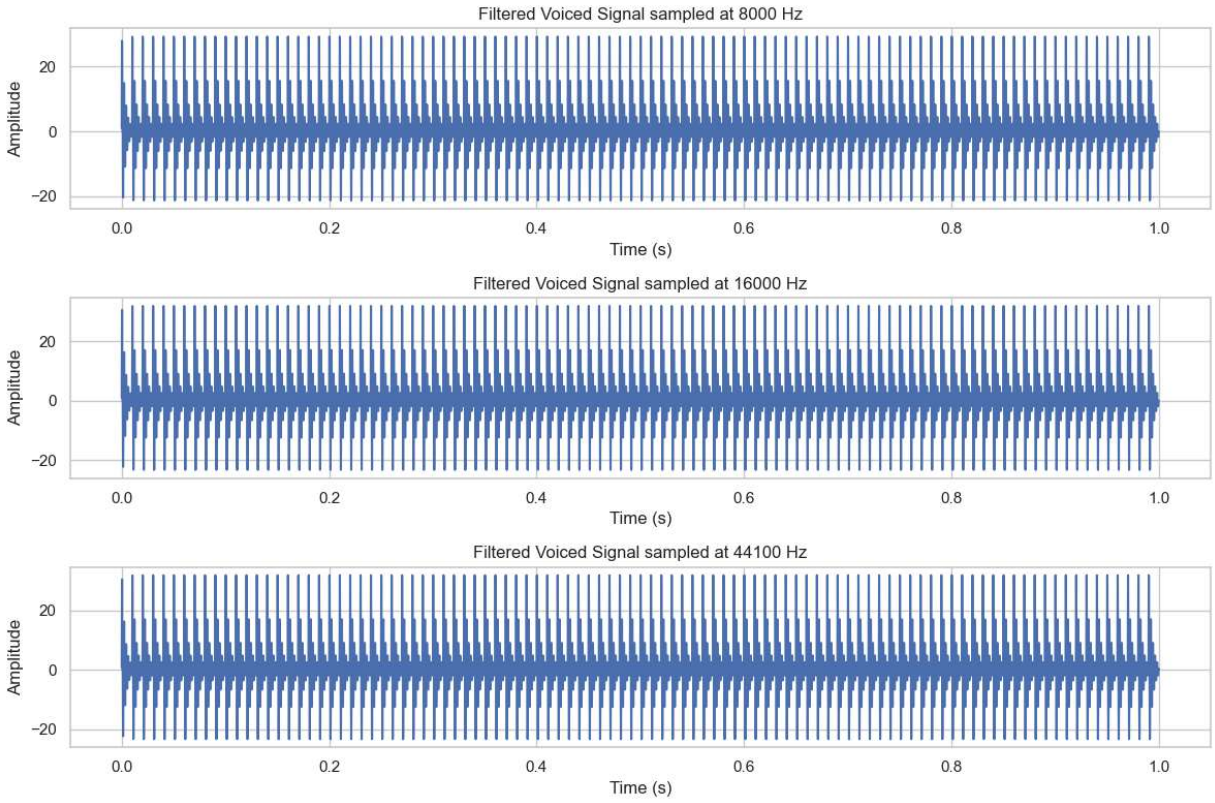


The glottal pulse train becomes smoother, simulating the resonances in the vocal tract.

The white noise also becomes filtered, emphasizing specific frequency components.

(c)Sample the Speech Signal at Different Rates

In [21]:
```
from scipy.signal import resample

sampling_rates = [8000, 16000, 44100]

plt.figure(figsize=(12,8))
for i, new_sr in enumerate(sampling_rates):
    num_samples = int(len(filtered_voiced) * new_sr / sr)
    sampled_signal = resample(filtered_voiced, num_samples)
    t_new = np.linspace(0, duration, num_samples, endpoint=False)

    plt.subplot(len(sampling_rates), 1, i + 1)
    plt.plot(t_new, sampled_signal)
    plt.title(f"Filtered Voiced Signal sampled at {new_sr} Hz")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.grid(True)

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



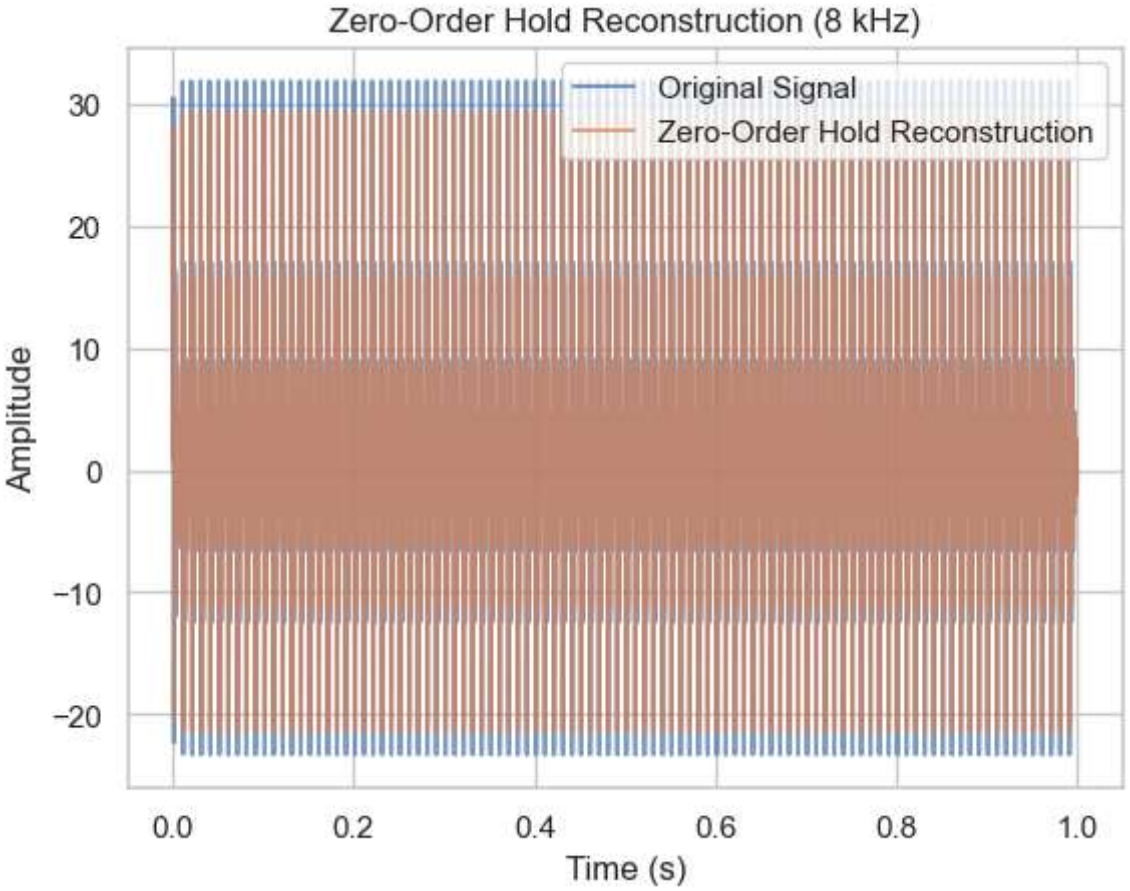
Sampling is typically applied to the filtered voiced signal rather than the unvoiced signal in the context of speech processing because voiced speech exhibits a more periodic structure, which makes the effects of sampling more apparent and critical for speech quality.

(d) Reconstruct the Signal Using Interpolation

(i) Zero-Order Hold (Nearest Neighbor)

In [22]:
```
from scipy.interpolate import interp1d

def zero_order_hold_reconstruction(t_sampled, sampled_signal, t_original):
    # Zero-order hold (nearest-neighbor)
    zoh = interp1d(t_sampled, sampled_signal, kind='nearest', fill_value="extrapola
    return zoh(t_original)

num_samples = int(len(filtered_voiced) * 8000 / sr)
sampled_signal_8k = resample(filtered_voiced, num_samples)
t_new_8k = np.linspace(0, duration, num_samples, endpoint=False)

reconstructed_zoh = zero_order_hold_reconstruction(t_new_8k, sampled_signal_8k, t)

# Plot the reconstructed signal
plt.plot(t, filtered_voiced, label="Original Signal", alpha=0.7)
plt.plot(t, reconstructed_zoh, label="Zero-Order Hold Reconstruction", alpha=0.7)
plt.title("Zero-Order Hold Reconstruction (8 kHz)")
plt.xlabel("Time (s)")
```

```
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.show()
```

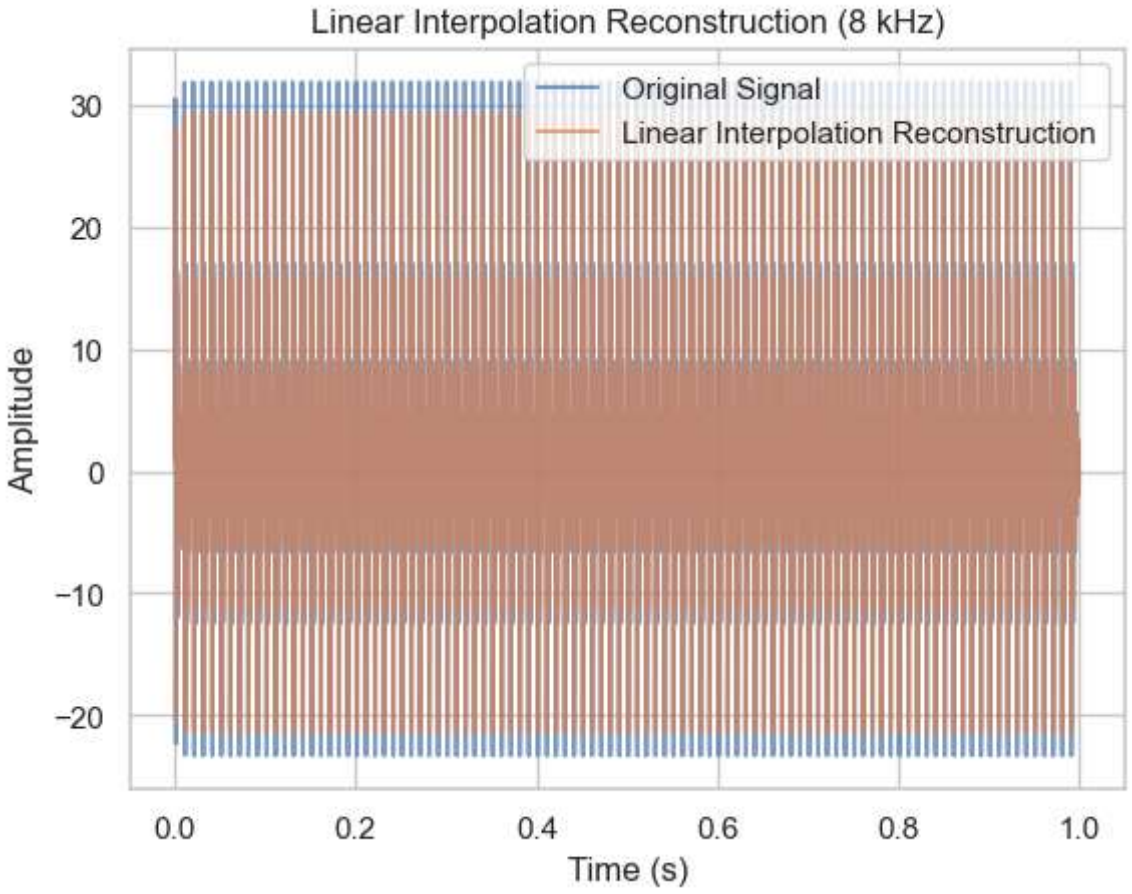


When reconstructing signals sampled at higher rates like 16 kHz or 44.1 kHz, the degradation in quality is less significant since the sampling rate is already high enough to capture most of the speech signal's frequency content. In contrast, reconstructing signals sampled at 8 kHz can lead to more perceptible differences from the original, especially for voiced components, making this rate a more interesting case for studying the effectiveness of reconstruction algorithms.

(ii) Linear Interpolation

In [23]:
```
def linear_interpolation_reconstruction(t_sampled, sampled_signal, t_original):
    # Linear interpolation
    lin_interp = interp1d(t_sampled, sampled_signal, kind='linear', fill_value="ext
    return lin_interp(t_original)

# Linear interpolation reconstruction
reconstructed_lin = linear_interpolation_reconstruction(t_new_8k, sampled_signal_8k

# Plot the reconstructed signal
plt.plot(t, filtered_voiced, label="Original Signal", alpha=0.7)
plt.plot(t, reconstructed_lin, label="Linear Interpolation Reconstruction", alpha=0
plt.title("Linear Interpolation Reconstruction (8 kHz)")
```

```
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.show()
```



In [24]:
```
from sklearn.metrics import mean_squared_error

# Calculate MSE for Zero-Order Hold
mse_zoh = mean_squared_error(filtered_voiced, reconstructed_zoh)

# Calculate MSE for Linear Interpolation
mse_lin = mean_squared_error(filtered_voiced, reconstructed_lin)

print(f"Mean Squared Error for Zero-Order Hold: {mse_zoh}")
print(f"Mean Squared Error for Linear Interpolation: {mse_lin}")
```

```
Mean Squared Error for Zero-Order Hold: 2.962709761469885
Mean Squared Error for Linear Interpolation: 0.23264393124830865
```

Inference

- The source-filter model simulates voiced and unvoiced speech sounds by generating a glottal pulse train for voiced sounds and white noise for unvoiced sounds. When applied to the vocal tract filter (modeled as an all-pole or FIR filter), the resulting speech signals show distinct patterns that replicate natural speech characteristics.

- Higher sampling rates result in better reconstruction quality, regardless of the interpolation method, though linear interpolation consistently provides more accurate results.
- Linear interpolation should be preferred over zero-order hold for reconstructing speech signals, particularly when maintaining speech quality and clarity is crucial at lower sampling rates.