Documentação do Assistente de Voz OpenAI Realtime

1. Visão Geral

O **Assistente de Voz OpenAI Realtime** é uma aplicação web que permite conversas em tempo real usando voz, aproveitando a API Realtime da OpenAI. A aplicação oferece uma interface intuitiva onde os usuários podem falar com um modelo de IA (GPT-4o) e receber respostas em áudio e texto em português brasileiro.

Principais Características

- Interação por voz bidirecional (fala e escuta)
- Respostas em português brasileiro
- Detecção automática do início e fim da fala (VAD)
- Possibilidade de interromper a IA durante as respostas
- Feedback visual do estado da conversa
- Interface responsiva e acessível

2. Arquitetura do Sistema

A aplicação segue uma arquitetura cliente-servidor:

- **Cliente**: Interface web responsiva (HTML/CSS/JavaScript)
- **Servidor**: Backend Python usando Flask e Socket.IO
- **API OpenAI**: Conexão WebSocket com a API Realtime

Fluxo de Comunicação

1. O usuário interage com a interface web
2. A interface web captura áudio do microfone
3. O áudio é enviado para o servidor via Socket.IO
4. O servidor processa e encaminha o áudio para a API OpenAI via WebSocket
5. A API processa o áudio, gera uma resposta e a envia de volta
6. O servidor encaminha a resposta para o cliente
7. O cliente reproduz o áudio e exibe o texto

3. Componentes Principais

3.1 Servidor (server.py)

O backend é construído em Python usando Flask e Socket.IO. Ele funciona como um intermediário entre o cliente e a API Realtime da OpenAI.

Dependências Principais

```
Flask>=3.0
Flask-CORS>=4.0
Flask-SocketIO>=5.3
openai>=1.0
websockets==11.0.3
pydub>=0.25
soundfile>=0.12.1
numpy>=1.24.0
colorlog>=6.7.0
```

Funcionalidades Principais

- Gerenciamento de sessões WebSocket com a API OpenAI
- Processamento de áudio (reformatação, conversão de taxa de amostragem)
- Gerenciamento de conexões com clientes via Socket.IO
- Tratamento de erros e registro de logs detalhados
- Configuração da API para respostas em português brasileiro

Configuração da API Realtime

```python
audio_config = {
    "type": "session.update",
    "session": {
        "input_audio_format": "pcm16",
        "turn_detection": {
            "type": "server_vad",
            "threshold": 0.3,                 # Mais sensível à fala
            "silence_duration_ms": 600,       # Detecta pausa mais rapidamente
            "prefix_padding_ms": 100,         # Menos tempo antes da fala
            "create_response": True,          # Responde automaticamente
            "interrupt_response": True        # Permite interrupção
        },
        "instructions": "Você é um assistente em português do Brasil. Responda ser
        "voice": "alloy"                      # Voz mais neutra
    }
}
```

3.2 Interface Web (index.html)

A interface web é uma página HTML responsiva com estilos CSS embutidos.

Elementos Principais

- Botão de gravação (para iniciar/parar a interação)
- Área de status (mostra o estado atual da aplicação)
- Área de transcrição (mostra o histórico da conversa)
- Indicadores visuais de estado (gravando, ouvindo, respondendo)

Estilos e Feedback Visual

- Indicações visuais claras para diferentes estados:

    - Gravando (borda vermelha pulsante)
    - Detectando fala (borda azul)
    - IA respondendo (borda verde)
- Botão que muda de cor e ícone conforme o contexto:
    - Microfone (padrão)
    - Stop (durante gravação)
    - Mão (para interromper a IA)
- Mensagens de sistema para feedback informativo

3.3 Lógica do Cliente (voice-assistant.js)

O arquivo JavaScript gerencia a captura de áudio, a comunicação com o servidor e a reprodução das respostas.

Funcionalidades Principais

- Gerenciamento da API de Áudio do navegador (AudioContext, getUserMedia)
- Processamento de áudio (conversão para PCM16, compressão para Base64)
- Comunicação bidirecional com o servidor via Socket.IO
- Reprodução de áudio recebido do servidor
- Atualização da interface conforme o estado da conversa
- Tratamento de erros e feedback visual

Estados de Conversação

- **Inativo**: Aguardando início de gravação
- **Gravando**: Capturando áudio do usuário
- **Ouvindo**: API detectou fala e está processando
- **Processando**: Enviando áudio para a API e aguardando resposta
- **Reproduzindo**: Tocando a resposta da IA
- **Interrompendo**: Cancelando a resposta atual da IA

4. Configuração e Personalização

4.1 Variáveis de Ambiente

- `OPENAI_API_KEY`: Chave da API OpenAI (obrigatória)
- `PORT`: Porta do servidor (padrão: 5000)
- `HOST`: Host do servidor (padrão: 0.0.0.0)

4.2 Parâmetros Ajustáveis

Configuração de VAD (Voice Activity Detection)

- `threshold`: Sensibilidade da detecção de fala (0.0 a 1.0)
- `silence_duration_ms`: Tempo de silêncio para considerar fim da fala
- `prefix_padding_ms`: Tempo de áudio a incluir antes da fala detectada

Configuração do Modelo

- `OPENAI_MODEL`: Modelo da OpenAI a ser utilizado
- `voice`: Voz utilizada para as respostas da IA
- `instructions`: Instruções para o comportamento do modelo

5. Fluxo de Operação

1. **Inicialização**:
   - Carregamento da interface
   - Conexão com o servidor via Socket.IO
   - Preparação das APIs de áudio do navegador
2. **Início da Interação**:
   - Usuário clica no botão de microfone
   - Sistema solicita permissão de acesso ao microfone
   - Início da captura e transmissão de áudio
3. **Processamento da Fala**:
   - Detecção automática do início da fala (VAD)
   - Transmissão contínua do áudio para o servidor
   - Detecção automática do fim da fala
   - Envio do bloco completo para processamento
4. **Geração da Resposta**:
   - Modelo processa o áudio e gera resposta
   - Resposta em áudio e texto enviada de volta em chunks
   - Cliente acumula e reproduz o áudio
5. **Interações Contínuas**:
   - Interface atualizada para permitir nova gravação
   - Histórico da conversa mantido na interface
   - Possibilidade de interrupção durante a resposta

6. Limitações e Considerações

- **Navegadores Suportados**: Chrome, Firefox, Edge e Safari mais recentes
- **Conexão**: Requer conexão estável com a internet
- **Latência**: Respostas podem ter pequenos atrasos dependendo da conexão
- **Compatibilidade de Áudio**: Utiliza as APIs padrão de áudio do navegador
- **Segurança**: A chave da API não deve ser exposta no cliente
- **Recursos de Servidor**: Cada conexão consome recursos significativos
- **Taxa de Amostragem**: Utiliza 16kHz para áudio enviado para a API
- **Duração de Sessão**: Limitada a 30 minutos pela API OpenAI

7. Plano de Otimização (Próximos Passos)

Identificamos vários desafios de desempenho que serão tratados no seguinte plano de implementação:

Fase 1: Melhorias Imediatas de Configuração

1.1 Otimizar configuração de VAD

python

```python
"turn_detection": {
    "type": "server_vad",
    "threshold": 0.25,             # Mais sensível
    "silence_duration_ms": 200,    # Detecta pausa MUITO mais rápido
    "prefix_padding_ms": 50,       # Menos tempo de prefixo
},
```

1.2 Melhorar instruções para respostas curtas e relevantes

python

```python
"instructions": "Você é um assistente em português do Brasil. IMPORTANTE: Dê respo
"temperature": 0.3,  # Mais determinístico
```

1.3 Ajustar tamanho do buffer de áudio no cliente

javascript

```javascript
// Em voice-assistant.js
scriptProcessorNode = audioContext.createScriptProcessor(2048, 1, 1);  // Era 4096
```

Fase 2: Monitoramento e Diagnóstico

2.1 Adicionar logging de desempenho

python

```python
# Adicionar no início do arquivo server.py
import time

# Modificar manage_openai_session para rastrear tempo
async def manage_openai_session(client_sid: str, audio_queue: asyncio.Queue):
    start_time = time.time()

    # [código existente]

    # Após conexão com OpenAI
    logger.info(f"[Perf] Conexão estabelecida: {(time.time() - start_time) * 1000:

    # Após configuração
    config_time = time.time()
    logger.info(f"[Perf] Configuração enviada: {(config_time - start_time) * 1000:

    # [restante do código]

    # No final
    logger.info(f"[Perf] Tempo total da sessão: {(time.time() - start_time) * 1000
```

2.2 Adicionar monitoramento no lado cliente

javascript

```javascript
// Em voice-assistant.js
// Adicionar timestamps nos eventos principais
const startTime = performance.now();

// Modificar handleTextChunk
function handleTextChunk(data) {
    if (!data || !data.text) return;

    // Calcular latência desde o início
    const latencyMs = performance.now() - startTime;
    console.log(`[Perf] Recebido chunk de texto após ${latencyMs.toFixed(2)}ms`);

    // [código existente]
}
```

Fase 3: Otimizações Avançadas

3.1 Mudar para modelo mais leve (se disponível)

python

```python
# Testar modelos alternativos
OPENAI_MODEL = "gpt-4o-mini-realtime"  # ou outro modelo mais rápido
```

3.2 Implementar processamento em chunks do áudio

python

```python
# Em server.py, modificar handle_audio_input_chunk
MAX_CHUNK_SIZE = 16000  # ~1 segundo de áudio
CHUNK_INTERVAL_MS = 500  # Enviar chunks a cada 500ms

@socketio.on('audio_input_chunk')
def handle_audio_input_chunk(data):
    # [código existente]

    # Dividir em chunks menores para processamento mais rápido
    if len(pcm_data) > MAX_CHUNK_SIZE:
        for i in range(0, len(pcm_data), MAX_CHUNK_SIZE):
            chunk = pcm_data[i:i+MAX_CHUNK_SIZE]
            resampled_chunk_base64 = base64.b64encode(chunk).decode('utf-8')
            try:
                audio_queue.put_nowait(resampled_chunk_base64)
            except asyncio.QueueFull:
                logger.warning(f"Fila cheia para {client_sid[:6]}, chunk descarta
    else:
        # [código original para chunks pequenos]
```

3.3 Adicionar limpeza periódica de recursos

python

```python
# Em server.py
import threading

def cleanup_task():
    """Executa limpeza periódica de recursos"""
    cleanup_inactive_connections()
    # Agendar próxima execução
    threading.Timer(60.0, cleanup_task).start()

def cleanup_inactive_connections():
    """Remove conexões inativas e libera recursos"""
    count = 0
    for client_id in list(client_tasks.keys()):
        if client_id not in socketio.server.manager.rooms.get('/', {}):
            logger.info(f"Limpando recursos para cliente inativo: {client_id[:6]}'
            client_tasks.pop(client_id, None)
            client_audio_queues.pop(client_id, None)
            client_sample_rates.pop(client_id, None)
            count += 1
    logger.info(f"Limpeza concluída: {count} conexões removidas")

# Iniciar tarefa de limpeza no startup
cleanup_task()
```

Fase 4: Melhorias na Experiência do Usuário

4.1 Melhorar feedback visual durante processamento

javascript

```javascript
// Em voice-assistant.js, adicionar animação de processamento mais informativa
function updateProcessingVisual(state) {
    const container = document.querySelector('.status-container');

    // Remover estados anteriores
    container.classList.remove('processing', 'listening', 'responding');

    // Adicionar novo estado
    container.classList.add(state);

    // Atualizar mensagem com base no estado
    if (state === 'processing') {
        showMessage("Processando sua fala...", "info");
    } else if (state === 'listening') {
        showMessage("Ouvindo...", "info");
    }
}
```

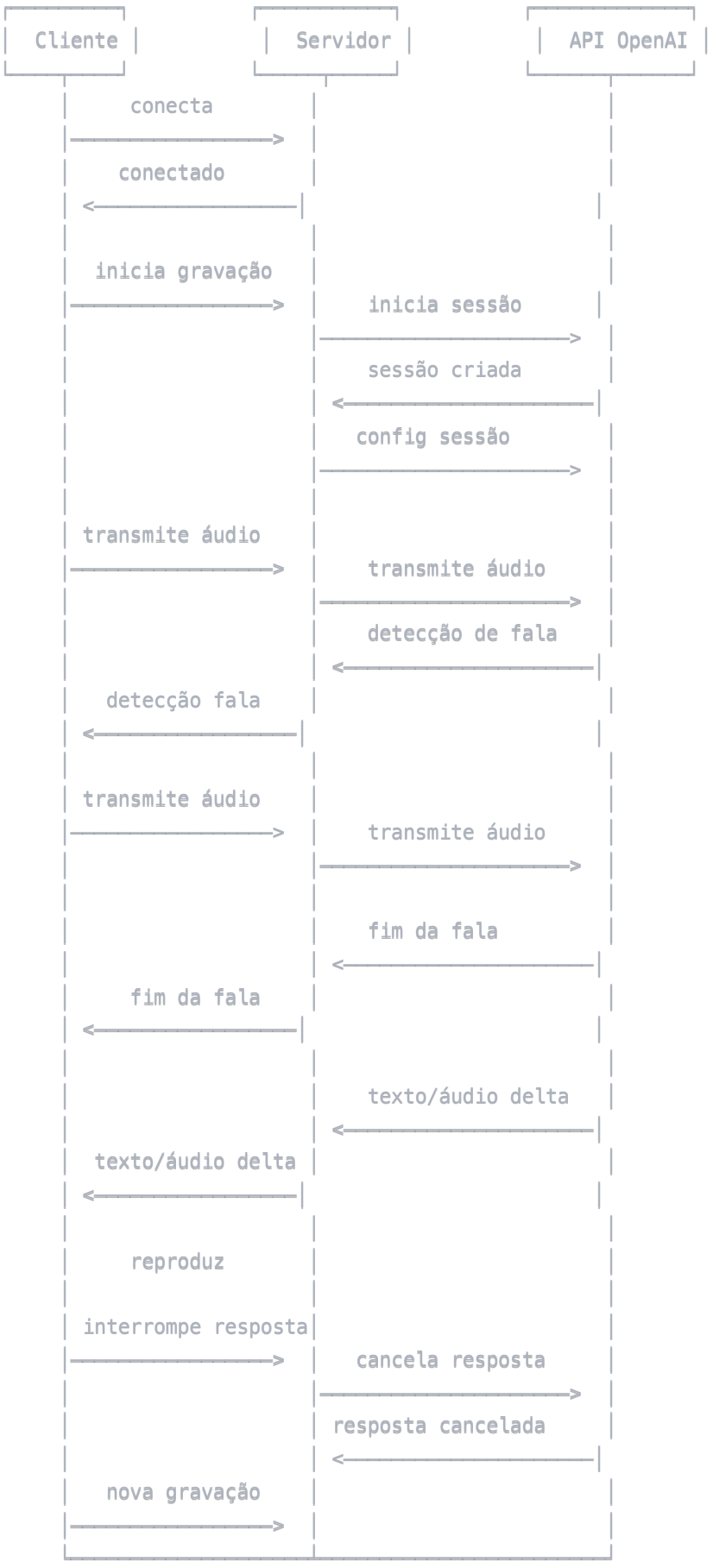
4.2 Implementar contexto mínimo para melhorar relevância

```python
# Em server.py, modificar gerenciamento de sessão para manter histórico mínimo
# Adicionar à configuração de sessão:
last_messages = []  # Armazenar últimas 2-3 mensagens

# Ao receber input do usuário
if last_user_query:
    last_messages.append(f"Usuário: {last_user_query}")
    if len(last_messages) > 3:
        last_messages.pop(0)

# Adicionar à configuração
audio_config = {
    # [configuração existente]
    "session": {
        # [configuração existente]
        "context": "\n".join(last_messages),
    }
}
```

Cronograma de Implementação

1. **Fase 1**: Implementação imediata - melhorias de configuração com alto impacto
2. **Fase 2**: 1-2 dias - adição de monitoramento para diagnóstico
3. **Fase 3**: 3-5 dias - otimizações baseadas nos dados de monitoramento
4. **Fase 4**: Semana seguinte - melhorias na experiência do usuário

Apêndice: Diagrama de Sequência


Cliente
Servidor
API OpenAI
conecta
conectado
inicia gravação
inicia sessão
sessão criada
config sessão
transmite áudio
transmite áudio
detecção de fala
detecção fala
transmite áudio
transmite áudio
fim da fala
fim da fala
texto/áudio delta
texto/áudio delta
reproduz
interrompe resposta
cancela resposta
resposta cancelada
nova gravação