ASSIT: Real-Time WebRTC Voice Bot Pipeline (Media Systems Challenge)

Background

You are building a real-time AI calling system that handles:

• live voice streaming
• interruption (barge-in)
• low-latency responses
• deterministic media pipelines

This assignment simulates the Media Gateway layer of that system using WebRTC instead of SIP.

You are NOT required to build any UI or AI model.

Your task is to implement real-time audio streaming + interruption logic.

Objective

Build a browser → WebRTC → Node.js media server → fake AI → browser duplex audio pipeline with:

• low latency
• streaming audio
• barge-in cancellation
• session isolation

Target Architecture

Browser Microphone
↓
WebRTC
↓
Node.js Media Server
↓
Fake STT (Streaming Simulator)
↓
Bot Logic
↓
Fake TTS (Streaming Audio Generator)

↓
Node.js Media Server
↓
WebRTC
↓
Browser Speaker

Technology Constraints

Required

• Node.js (>=18)
• WebRTC server library:
o wrtc OR
o mediasoup OR
o pion-webrtc (Node binding)

Allowed
• Express / Fastify (for signaling APIs)
• WebSocket (for signaling)
Forbidden
Cloud STT/TTS APIs
ElevenLabs / Google / Whisper
TURN servers
Pre-recorded audio files
Playing WAV files instead of streaming PCM
Functional Requirements
Browser Client Requirements
Implement a simple webpage that:
Must:
• Capture microphone audio using getUserMedia
• Establish WebRTC connection with server
• Send microphone audio track

• Play remote audio track received from server
UI Requirements:
Minimal UI is enough:
• Start Call button
• Stop Call button
• Connection status indicator
No video needed.
Reference Docs
• Media capture:
https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia
• WebRTC basics:
https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
Server-Side WebRTC Media Handling (CORE)
Your Node.js server must:
A) Accept WebRTC Peer Connections
Implement signaling using HTTP or WebSocket:
• SDP offer from browser
• SDP answer from server
References
• wrtc:
https://github.com/node-webrtc/node-webrtc
• mediasoup:
https://mediasoup.org/documentation/
• WebRTC signaling overview:
https://webrtc.org/getting-started/peer-connections
B) Access Raw Audio Frames
You MUST:
• Extract incoming audio track PCM data

-- 3 of 8 --
• Convert to:
o 16kHz
o mono
o signed 16-bit PCM
• Chunk into 20ms frames
Target frame size:
16,000 samples/sec
20ms = 320 samples
320 samples × 2 bytes = 640 bytes per frame
Reference
• Web Audio PCM processing:
https://developer.mozilla.org/en-US/docs/Web/API/AudioWorklet
• Audio track processing:
https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamTrackProcessor
Fake STT (Streaming Simulation)
Implement a local Fake STT module.
Behavior
Fake STT must:
• Receive audio frames
• Detect voice activity (simple RMS or amplitude threshold)
• After detecting speech for ~500ms:
o Emit PARTIAL transcript
o Emit FINAL transcript after silence
Example Output
{
type: "partial",
text: "hel"
}
-- 4 of 8 --
{
type: "final",
text: "hello"
}
Text can be static (hardcoded).
Important
You are simulating:
• streaming behavior
• latency
• async transcript arrival
NOT speech recognition.
Fake TTS (Streaming Audio Generator)
Implement a Fake TTS service.
Behavior
Fake TTS must:

• Accept text input
• Generate synthetic audio:
o sine wave
o tone
o noise pattern
• Stream PCM chunks progressively (20ms per chunk)
• NOT return full audio buffer at once
Reference
• Generating PCM audio:
https://developer.mozilla.org/en-US/docs/Web/API/AudioBuffer
• PCM basics:
https://en.wikipedia.org/wiki/Pulse-code_modulation
-- 5 of 8 --
Barge-In (MANDATORY)
This is the most important part.
Requirement
While Fake TTS is playing:
If user starts speaking:
• Immediately STOP audio playback
• Flush TTS output buffer
• Switch system back to LISTENING mode
Constraints
Maximum acceptable stop delay:
< 300ms from voice detection
Implementation Expectation
Use:
• AbortController
• Cancellation tokens
• Playback queue flush
NOT timeouts or flags only.
Session State Machine
You must implement:
IDLE
LISTENING
PROCESSING
SPEAKING
INTERRUPTED
State transitions must be explicit.
-- 6 of 8 --
Performance Logging
Log:
• WebRTC connection time
• Fake STT response latency
• Fake TTS streaming start time
• Barge-in cancellation latency

Deliverables

Candidate must submit:

1) Source Code Repository

Must include:

• Client code
• Server code
• Fake STT module
• Fake TTS module

3) Demo

Show:

• Voice flowing
• Bot speaking
• User interrupting bot
• Immediate stop of playback

Evaluation Criteria

Area Weight

WebRTC handling 25%

Audio processing 20%

Barge-in logic 25%

State management 15%

Code quality 15%

Auto-Rejection Conditions

Submission is rejected if:

No barge-in support

Full audio file playback

No raw PCM handling

Single global session

Blocking code

Uses cloud STT/TTS

Bonus (Optional)

Extra credit if candidate implements:

• Multi-session support
• Packet jitter buffering
• Audio resampling
• Partial transcript handling

Final Warning To Candidate

This is a real-time systems test.

UI polish, styling, animations are irrelevant.

We are evaluating:

• streaming logic
• concurrency handling
• interruption control
• production-grade thinking