

Technical Assessment: Real-Time Audio Systems Engineer

Company: Quantum Strides

Estimated Duration: 6-8 Hours

1. Overview

The objective of this assignment is to evaluate your proficiency in building low-latency, real-time audio applications within a browser environment. You are required to develop a lightweight voice bot that listens for a specific wake word, transcribes speech in real-time, queries a local knowledge base, and responds via streaming text-to-speech.

This task assesses your ability to manage WebSockets, handle binary audio data streams, and optimize client-side performance to meet strict latency requirements.

2. Project Objective

Build a serverless (client-side only) voice assistant that activates on the wake word "**Hey Qplus**". Upon activation, the application must process user speech and provide an audible response with an end-to-end latency (Speech End -> Response Start) of <1.2 seconds.

3. Technical Requirements

The solution must be implemented primarily in vanilla JavaScript, HTML, and CSS. While you may use helper libraries for specific tasks (detailed below), the core logic and state management should demonstrate your understanding of the underlying APIs.

3.1. Wake Word Activation (Client-Side DSP)

Implement a client-side mechanism to detect the phrase "**Hey Qplus**".

- **Technology:** You may use **anything preferred**.
- **Behavior:**
 - The microphone must remain in a "listening" state without streaming data to the cloud.
 - Upon detecting the wake word, the application must immediately switch to an "active" state and provide a visual cue to the user.

3.2. Streaming Speech-to-Text (STT)

Upon activation, the application must stream audio to a cloud provider via WebSockets.

- **Allowed Providers:** AssemblyAI Realtime, Deepgram Realtime, or OpenAI Realtime API or Anything. Closed Sourced will be preferred.

- **Constraints:**
 - Use **WebSockets** for data transmission. HTTP REST APIs are not permitted for audio streaming.
 - Display real-time partial transcripts and the final transcript in the user interface.

3.3. Knowledge Base & Logic

Use a predefined local JSON dataset to determine the bot's response.

- **Format:** Create a file named `knowledge_base.json` containing simple question-answer pairs.
 - *Example:* `{"what is qplus": "Qplus is an AI platform designed for automation."}`

3.4. Streaming Text-to-Speech (TTS)

The response must be synthesized and played back with minimal latency.

- **Allowed Providers:** OpenAI Realtime, AssemblyAI TTS, or ElevenLabs Streaming.
- **Constraints:**
 - Receive audio data via a **WebSocket stream**.

3.5. User Interface (UI)

The UI should be minimal and functional, built using HTML and CSS.

- **Controls:** A "Start" button to initialize the audio context.
- **Status Display:** Clearly indicate the current state: *Idle*, *Listening for Wake Word*, *Processing*, *Speaking*.
- **Transcript View:** A dedicated area showing the live transcription and the selected response.

4. Architecture & High-Level Design (HLD)

In addition to the code, you are required to submit a High-Level Design document explaining your engineering approach. This document should include:

1. **Architecture Diagram:** A visual representation of the system flow, detailing the movement of data from the microphone \rightarrow browser logic \rightarrow external WebSocket APIs \rightarrow audio output.
2. **Technical Nuances:**
 - **Audio Pipeline:** Explain your choice of Web Audio API nodes (e.g., `ScriptProcessorNode` vs. `AudioWorkletNode`) and how you handle PCM data conversion.
 - **Latency Optimization:** Describe specific techniques used to minimize the Time-to-First-Byte (TTFB) and ensure the 1.2-second target is met.
 - **State Management:** How the application handles connection states, errors, and re-connection logic.

5. Deliverables & Submission

Please host your solution in a public GitHub repository and share the link. The repository must contain:

1. **Source Code:** Complete, modularized codebase including `index.html`, `script.js`, `style.css`, and `knowledge_base.json`.
2. **HLD Document:** A PDF or Markdown file containing the architecture diagram and technical explanation as described in Section 4.
3. **Demo Video (Mandatory):** A 30 to 60-second screen recording demonstrating the working application.
 - The video must clearly show the wake word activation, the real-time transcription, and the audio response.
 - *Please upload this to a platform like Loom, YouTube (Unlisted), or Google Drive and include the link in your README.*
4. **README.md:** Clear instructions on how to run the project locally, including where to insert necessary API keys.

6. Evaluation Criteria

Your submission will be evaluated based on the following:

Category	Description	Weight
System Architecture	Quality of the HLD diagram and justification of technical choices regarding the audio pipeline.	High
Latency Performance	Adherence to the \$le 1.2\$ second latency target and implementation of non-blocking, chunked playback.	Critical
Code Quality	Cleanliness, modularity, and error handling of the JavaScript implementation.	Medium
Functional Completeness	Accuracy of wake-word detection, STT integration, and knowledge base retrieval.	Medium

Note: This assignment requires the use of API keys from third-party providers. Please ensure you do not commit active API keys to the public GitHub repository. Use a .env file approach or prompt the user to enter keys in the UI.