Proposal: Client-Side Real-Time Voice Transformation using JUCE

1 Introduction

This proposal describes a client-side architecture for real-time voice transformation for Resonate. The goal is to allow speakers to select from predefined voice styles while maintaining low latency, strong privacy guarantees, and cross-platform consistency. All audio processing is performed locally on the device, and only processed audio is transmitted over the network.

2 Objectives

The primary objectives of this proposal are:

- Enable predefined, selectable voice styles for speakers.
- Perform all voice transformation on-device.
- Avoid server-side audio processing or storage.
- Maintain real-time performance suitable for live audio rooms.
- Share DSP logic across Android and iOS using C++.
- Integrate seamlessly with an existing Flutter + LiveKit codebase.

3 Design Principles

- **Client-side processing**: All DSP runs locally on the device.
- **Privacy-first**: Raw microphone audio never leaves the device.
- **Low latency**: Processing must meet real-time constraints.

4 High-Level Architecture

The system consists of four layers:

1. **Flutter UI Layer** — voice selection and preview controls.
2. **Native Audio Capture Layer** — microphone input (iOS / Android).
3. **JUCE DSP Engine (C++)** — real-time voice transformation.
4. **LiveKit Transport Layer** — customizing to take the processed DSP input.

5 Correct Audio Flow

Audio flows directly from the microphone to the DSP engine before being published:

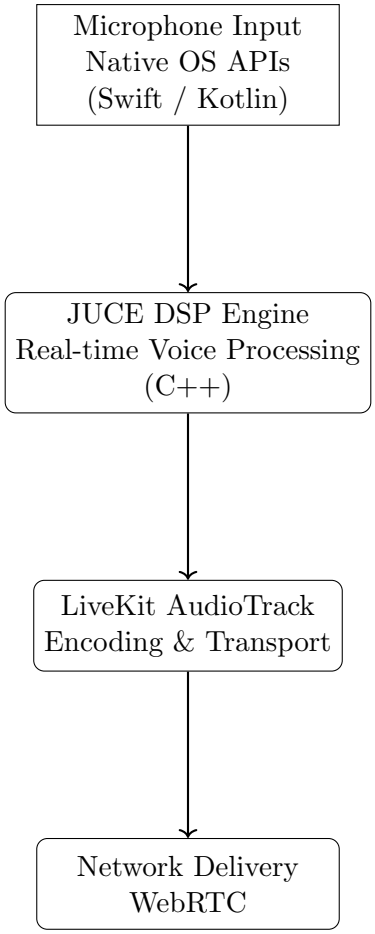

Figure 1: Client-side real-time voice processing and transport flow

6 Repository Integration

6.1 Flutter Layer

Flutter is responsible only for UI and control signals.

```
lib/
  services/
    voice_engine_channel.dart
```

Flutter communicates with native code using platform channels.

6.2 Shared JUCE DSP Engine

All DSP logic is implemented once using JUCE and shared across platforms.

```
/dsp/
  voice_engine/
    |- VoiceEngine.h
    |- VoiceEngine.cpp
    |- VoiceProfiles.h
```

6.3 Android Integration

JUCE is compiled using the Android NDK and accessed via JNI.

```
android/app/src/main/
  cpp/
    |- voice_engine/
    |- VoiceEngineJNI.cpp
  kotlin/
    |- VoiceEnginePlugin.kt
```

6.4 iOS Integration

JUCE is compiled as a static library and linked using Objective-C++.

```
ios/Runner/
  |- VoiceEngineWrapper.mm
  |- VoiceEngineWrapper.h
```

Native layers handle:

- Microphone capture
- Calling JUCE DSP
- Publishing processed audio to LiveKit
- Receiving Flutter control commands

7 Preview Mode

Preview mode allows users to test voice styles locally:

- Audio is captured from the microphone.
- JUCE DSP applies the selected voice profile.
- Audio is played back locally.
- No data is transmitted or stored.

8 Performance Considerations

The DSP engine is designed to be real-time safe:

- No dynamic memory allocation in the audio thread.
- Fixed-size audio buffers.
- JUCE DSP overhead is approximately 1–5% CPU.
- No additional latency introduced by the DSP layer.

9 Security and Privacy

- Raw microphone audio is never transmitted.
- Voice transformation is user-controlled and optional.
- No server-side voice modification occurs.