Technical Specification: High-Precision Web DSP Tuner

This Technical Architecture & Implementation Guide is designed to be handed directly to a Senior Software Architect or an AI Coding Agent.
It defines the mathematical, architectural, and logical constraints required to build a "Class A" precision stroboscopic-grade tuner on the web.

Version: 1.0 **Target System:** Web (React/AudioWorklet) + Python (Validation) **Core Objective:** Real-time monophonic pitch detection with sub-cent accuracy (<1 cent), implementing Inharmonicity Compensation (Stretched Tuning) for guitar.

1. System Architecture Overview

To achieve zero-latency performance and prevent UI-blocking during heavy DSP operations, the application must adhere to a strict **Dual-Thread Architecture**.

1.1 The Audio Thread (DSP Core)

- **Context:** AudioWorkletGlobalScope
- **Responsibility:** Raw buffer ingestion, windowing, autocorrelation, sub-sample interpolation.
- **Constraint:** strictly **no garbage collection (GC)** during the render loop. Arrays must be pre-allocated (TypedArrays).
- **Output:** Stream of { fundamentalFrequency: float, confidence: float } messages to the Main Thread.

1.2 The Main Thread (UI & State)

- **Context:** React (Main JS Thread)
- **Responsibility:** Visualization (needle/strobe), State Management (Zustand/Context), User Input.
- **Smoothing:** Application of a temporal smoothing filter (e.g., Kalman Filter or simple Moving Average) to the incoming frequency data before rendering to prevent visual jitter.

2. Core DSP Algorithm: Modified YIN

Standard FFT (Fast Fourier Transform) is insufficient for low-frequency precision (e.g., Low E string ~82Hz) due to bin resolution limits. The system **must implement the YIN Algorithm** (or MPM) in the AudioWorklet.

2.1 The YIN Implementation Steps

The agent must implement the following 5 distinct steps in the DSP processor:

1. **Difference Function:** Instead of standard autocorrelation, calculate the squared difference: *Where x is the signal buffer, \tau is the lag (delay).*
2. **Cumulative Mean Normalized Difference (CMND):** Normalize the function to eliminate the "zero-lag" peak issue found in standard autocorrelation.
3. **Absolute Thresholding:** Select the first valley in d'(\tau) that drops below a threshold (recommended: 0.10 or 0.15) to avoid "octave errors" (mistaking the fundamental for the 2nd harmonic).
4. **Parabolic Interpolation (Crucial for Cents Precision):** The integer lag \tau is too coarse. The agent must fit a parabola through the valley point and its two neighbors to find the fractional minimum. *This step converts "nearest semitone" accuracy into "micro-cent" accuracy.*
5. **Best Local Estimate:** Convert the refined lag period back to frequency:

3. The "Stretched Tuning" Logic (Inharmonicity)

Guitar strings behave like stiff rods, causing overtones to be sharp relative to the fundamental. To mimic the **Railsback Curve** (piano) for guitar, we must target "Sweetened" frequencies rather than standard Equal Temperament (12-TET).

3.1 Offset Strategy

The application should use a ReferenceMap that applies specific cent offsets to the target detection logic.

Standard 12-TET Targets (440Hz base):

- E2: 82.41 Hz
- A2: 110.00 Hz
- ...

Required Stretched/Sweetened Logic: The tuner must calculate the deviation not from 12-TET, but from the **Compensated Target**.

- **Formula for Cents Deviation:**
- **Suggested Offset Table (Guitar Sweetener):**
  - **E2 (Low):** -2.0 cents (Compensate for attack sharpness)
  - **A2:** -1.0 cents
  - **D3:** 0 cents
  - **G3:** 0 cents
  - **B3:** -0.5 cents (Harmonic major 3rd adjustment)
  - **E4 (High):** 0 cents
- *Advanced:* Implement an **Inharmonicity Coefficient (B)** parameter that stretches the target frequency based on the string stiffness formula: *(For this MVP, the fixed Offset Table is recommended for stability).*

4. Implementation Directives

4.1 Frontend Stack (React + Tailwind)

- **Canvas vs. DOM:** Use HTML5 <canvas> for the tuner needle/strobe. DOM manipulation (divs) at 60fps will cause jitter.
- **State Management:** Use useRef for the frequency values to avoid React render-cycle batching. Only trigger state updates for *significant* visual changes (throttling).

4.2 Python Validation Suite (The "Lab")

Before deploying JS code, the algorithms must be verified in Python.

- **Libraries:** numpy, scipy, librosa.
- **Task:** Create a script validate_tuning.py.
    1. Generate synthetic sine waves with known inharmonicity (e.g., a wave at 82.41Hz + slight sharp overtones).
    2. Run the Python implementation of the YIN algorithm on this data.
    3. Assert that the detected pitch is within **+/- 0.5 cents** of the ground truth.

4.3 Data Structures (Pseudocode)

AudioWorklet Processor (JS):

```
class TunerProcessor extends AudioWorkletProcessor {
  constructor() {
    super();
    this.buffer = new Float32Array(2048); // Fixed buffer size
    this.sampleRate = 44100;
  }

  process(inputs, outputs, parameters) {
    const input = inputs[0];
    const channel = input[0];

    // 1. Fill Buffer
    // 2. Run YIN Algorithm (Bitwise optimized)
    // 3. Post Message
    this.port.postMessage({ pitch: calculatedPitch, confidence:
confidence });

    return true;
  }
}
```

5. Development Roadmap for the Coding Agent

1. **Phase I: The DSP Prototype (Python)**
    - **Action:** Write a Python script implementing YIN.
    - **Input:** WAV files from the **GuitarSet** dataset (specifically monophonic stems).

    - **Output:** CSV comparing Detected_Hz vs Annotation_Hz.
    - **Goal:** Tuning the "Threshold" parameter to ensure E2 (Low E) is detected reliably without octave jumping.
2. **Phase II: AudioWorklet Translation**
    - **Action:** Port the Python YIN logic to vanilla JavaScript (ES6) inside an AudioWorkletProcessor.
    - **Constraint:** Zero memory allocation inside the process() loop. Reuse arrays.
3. **Phase III: The Stretched Integration**
    - **Action:** Build the TuningSystem class in React.
    - **Logic:** Input raw Hz -> Lookup Target Note -> Apply Offset -> Return Cents deviation.
4. **Phase IV: UI Visualization**
    - **Action:** Build the Tailwind UI. Use requestAnimationFrame loop to interpolate the needle position between the raw frequency updates coming from the Worklet.

6. Resources & References for the Architect

- **Primary Paper:** *YIN, a fundamental frequency estimator for speech and music* (De Cheveigné & Kawahara).
- **Dataset: GuitarSet** (via Zenodo) – specifically "Mic" recordings for testing noise resilience.
- **Compensated Tuning Reference:** Peterson Tuners "Sweetened Tuning" whitepapers (for offset values).