**Oscar Herrera**

**Tuner and DAW**

**Functional Requirements:**

These are the core features the user directly interacts with:

1. **Microphone Input Selection:**
   * **Input:** The user selects a microphone from a dropdown list of available audio input devices.
   * **Operation:** The system uses the selected microphone as the audio source.
   * **Output:** The system receives audio signals from the chosen microphone.
2. **Start/Stop Tuning:**
   * **Input:** The user clicks a "Start" button to begin the tuning process.
   * **Operation:** The system starts recording audio from the selected microphone and begins analyzing the frequency content in real-time. The timer for UI updates is initiated.
   * **Output:** The system continuously displays the detected frequency and corresponding musical note, and updates the visual meter and waveform display.
   * **Input:** The user clicks a "Stop" button to end the tuning process.
   * **Operation:** The system stops recording audio and halts the real-time analysis and UI updates. The timer is stopped.
   * **Output:** The frequency and note labels, meter, and waveform display show a "stopped" state or the last detected values.
3. **Frequency Display:**
   * **Input:** Real-time audio data from the microphone.
   * **Operation:** The system performs FFT analysis on the audio data to determine the dominant frequency. Smoothing algorithms are applied to stabilize the reading.
   * **Output:** A numerical display of the detected frequency in Hertz (Hz), updated in real-time.
4. **Note Display:**
   * **Input:** The detected frequency in Hertz.
   * **Operation:** The system converts the detected frequency to the closest musical note name (e.g., A, C#, E) and octave number. Hysteresis is applied to prevent rapid note switching.
   * **Output:** A display of the detected musical note name and octave, updated in real-time.
5. **Visual Tuning Meter:**
   * **Input:** The current detected frequency and the target frequency of the displayed note.
   * **Operation:** The system calculates the difference between the current and target frequencies and displays this deviation visually on a meter.
   * **Output:** A graphical meter indicating whether the pitch is sharp, flat, or in tune relative to the displayed note.
6. **Waveform Display:**
   * **Input:** Real-time audio data from the microphone.
   * **Operation:** The system displays a visual representation of the incoming audio waveform.
   * **Output:** A scrolling or updating graphical display of the audio waveform.
7. **Basic DAW Recording (Integrated):**
   * **Input:** User clicks a "DAW Record" button.
   * **Operation:** The system (via interaction with a separate WaveInForm) starts recording audio from a selected input device to a WAV file.
   * **Output:** An audio file is saved to a specified location.
   * **Input:** User clicks a "DAW Stop" button.
   * **Operation:** The system stops the DAW recording process.
   * **Output:** The WAV audio file is finalized.
8. **Basic DAW Playback (Integrated):**
   * **Input:** User clicks a "DAW Play" button, provided a recorded WAV file exists.
   * **Operation:** The system plays back the recorded WAV audio file.
   * **Output:** Audio is played through the system's default audio output device.

**Non-Functional Requirements:**

1. **Performance:**
   * **Real-time Responsiveness:** The tuner should provide near real-time feedback on pitch changes with minimal noticeable latency (frequency and note updates within 100-200 milliseconds).
   * **CPU Usage:** The application should utilize a reasonable amount of CPU resources to avoid impacting the overall system performance during operation.
   * **FFT Processing Speed:** The FFT analysis should be performed efficiently to ensure timely frequency detection.
2. **Reliability:**
   * **Stable Operation:** The application should operate consistently without crashing or freezing during normal use.
   * **Accurate Frequency Detection:** The frequency detection algorithm should provide accurate readings for typical musical instrument tones.
   * **Robust Audio Input Handling:** The system should handle different audio input levels and devices gracefully.
3. **Maintainability:**
   * **Code Clarity:** The codebase should be well-organized, commented, and easy to understand for future modifications and bug fixes.
   * **Modular Design:** The application should be structured in a modular way to facilitate easier updates and the addition of new features.
4. **Portability:**
   * **Windows Compatibility:** The application is currently being developed for the Windows operating system. Future portability to other platforms (e.g., macOS, Linux) might be considered.
5. **Security:**
   * **Audio Input Permissions:** The application should request and handle audio input permissions appropriately, respecting user privacy.
6. **Scalability:**
   * The underlying audio processing and UI update mechanisms should be efficient enough to handle continuous audio input without performance degradation.
7. **Reusability:**
   * **Modular Components:** Certain components, such as the FFT analysis and note detection logic, could potentially be reused in other audio-related applications.
8. **Flexibility:**
   * **Parameter Adjustments:** While not fully implemented with a user interface, the underlying parameters like FFT length, smoothing factors, and hysteresis thresholds can be adjusted in the code to fine-tune the tuner's behavior for different instruments or user preferences.
   * **Potential for Feature Expansion:** The architecture should allow for the future addition of features like different tuning modes (chromatic, specific instrument tunings), temperaments, or advanced analysis tools.