**AudioApp Known Issues & Areas for Improvement**

**1. Audio file format conversion**

* **Issue:**

The app always records segments in wav format, required for OpenAI Whisper, and simply *copies* them for the user’s preferred format (caf, m4a, etc).

This only renames the file extension without actual transcoding. This means users may find non-.wav files incompatible with some media players.

* **Improvement:**

Integrate AVAssetExportSession to truly convert audio files to the selected format (e.g., m4a).

**2. Audio encryption at rest**

* **Issue:**

The app saves audio files as plain files on disk (either in Documents or tmp). There’s currently no file-level encryption, meaning anyone accessing the app’s sandbox could extract recordings.

* **Improvement:**

Encrypt audio files at rest (using AES with keys stored in Keychain).

Alternatively, use Apple’s NSFileProtection.complete to enforce encryption tied to device lock. For this we could explicitly set the file protection attribute when creating the file. Something like this:

do {

try FileManager.default.setAttributes([.protectionKey: FileProtectionType.complete], ofItemAtPath: fileURL.path)

} catch {

logger.error("Failed to set file protection: \(error.localizedDescription)")

}

**3. SwiftData concurrency + SwiftUI**

* **Issue:**

When writing new RecordingSession or AudioSegment objects, AudioApp immediately saves the context. If the view is observing the same data, this may trigger UI updates more frequently than necessary

* **Improvement:**

Consider batching inserts and deferring saves to reduce disk I/O and UI updates.

Explore ModelActor or SwiftData background contexts to offload heavy operations.

**4. Memory peaks with very large segments**

* **Issue:**

Even though the segments are short (~30 sec) and memory is released after writing, if settings allow larger segment durations like 20 sec, then multiple PCM buffers accumulate in memory.

* **Improvement:**

We might want to think about capping segment length more conservatively or write buffers directly to file to avoid building up large arrays in memory.

**5. Limited offline fallback UX**

* **Issue:**

If the OpenAI Whisper API fails (e.g., no network), fallback to Apple Speech is robust — but there’s no explicit user-facing notification of *which engine* was ultimately used.

* **Improvement:**

Consider logging the fallback choice in the Transcription model or showing a badge indicating “Transcribed Locally” vs “Cloud”.

**6. Lack of background recording**

* **Issue:**

The current AVAudioSession does not handle long background recording sessions (like voice memos or call recorders). If the app is backgrounded, recording stops.

* **Improvement:**

Investigate using the .playAndRecord category with AVAudioSessionCategoryOptions.mixWithOthers and configure background modes for audio to continue capturing.

**7. No deduplication of segments**

* **Issue:**

If the app crashes between writing an audio file and saving its AudioSegment in SwiftData, the file will stay orphaned. Or vice versa: a segment might exist with a missing file.

* **Improvement:**

Implement a simple cleanup scan to reconcile SwiftData against disk, removing orphaned files or dangling database rows.

**Summary**

These issues are **not critical bugs** but reflect normal trade-offs for iOS Audio Recording & Transcription Take-Home Assignment and some were just unfinished due to time constraint.

Each is straightforward to address in future iterations to strengthen feature-set and data privacy.