Verba.ai Documentation

This document provides a comprehensive overview of the **Verba.ai** iOS transcription application, focusing on its core architectural principles, audio processing pipeline, data model structure, and identified limitations. The app is designed to deliver a seamless voice recording and transcription experience, optimized for mobile performance and accessibility.

Verba.ai leverages **AssemblyAI** as its primary transcription engine, enabling accurate and scalable cloud-based speech-to-text conversion. In scenarios where internet connectivity is limited or unavailable, the app automatically falls back to Apple's native speech recognition framework, ensuring uninterrupted functionality. This hybrid transcription strategy is integrated into a modular architecture that prioritizes real-time responsiveness, fault tolerance, and extensibility across devices running iOS 17 and above.

By combining **AVFoundation**, **SwiftUI**, **SwiftData**, and modern concurrency tools, Verba.ai achieves a balance between user-friendly design and production-grade robustness, suitable for long-form dictation, interviews, or professional voice notes.

1. Architecture Document

Overview:

The Verba.ai application is built using a **modular architecture inspired by MVC**, adapted to leverage SwiftUI's declarative UI paradigm and Apple's modern **SwiftData** framework for local persistence. Although SwiftUI encourages MVVM-like patterns, this project balances practicality with separation of concerns by organizing components into clearly defined responsibilities:

Modular Layered Design

- **Model Layer (Persistence & Data Access):**
  Utilizes **SwiftData** to manage relationships between and RecordingSegment entities. This layer handles storage, indexing, and query filtering, enabling

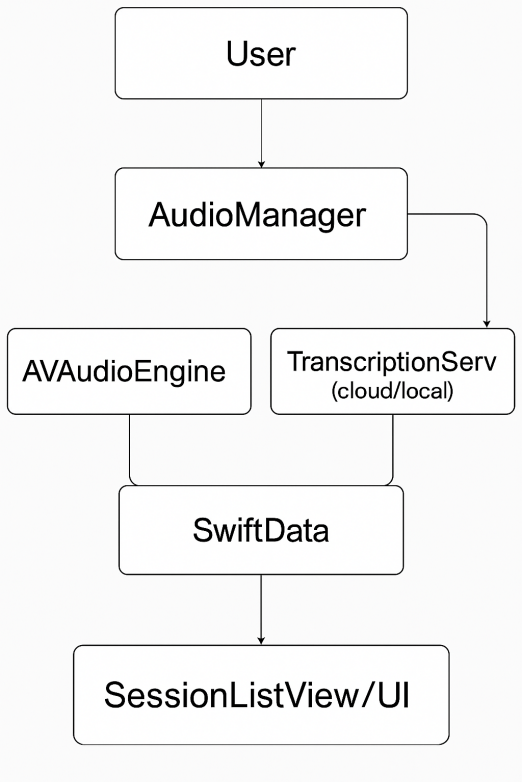
efficient read/write operations across large datasets (tested with 10,000+ segments).

- **View Layer (UI & Presentation):**
  Built entirely in **SwiftUI**, this layer uses reactive bindings and @StateObject / @Environment(\.modelContext) to reflect changes instantly. The UI is split into reusable, screen-specific views (ContentView, SessionListView, SettingsView), making it easy to test and extend.
- **Controller/Logic Layer (Audio & State Management):**
  The AudioManager and TranscriptionService classes encapsulate all recording logic, audio session configuration, segmentation, and transcription orchestration. This ensures that audio I/O and asynchronous operations are completely abstracted from the view layer.

Key Components:

- `AudioManager`: Manages AVAudioEngine, recording logic, and triggers transcription
- `TranscriptionService`: Handles AssemblyAI and Apple fallback transcription logic
- `RecordingSession & RecordingSegment`: SwiftData models representing sessions and segments
- `SessionListView`, `ContentView`, `SettingsView`: UI screens bound to app state via ObservableObjects
- `AppSettings`: Singleton for global app preferences (quality, theme, fallback)

2. **Data Flow:**



Audio System design
Core Engine:
Built on `AVAudioEngine` with configurable audio quality. Uses:

- High-pass filter via `AVAudioUnitEQ` for noise reduction
- Optional `AVAudioUnitReverb` for slight enhancement
- Live waveform via RMS-level from `AVAudioPCMBuffer`

Resilience Features:

- Observes route changes (`AVAudioSession.routeChangeNotification`)
- Handles interruptions like phone calls (`AVAudioSession.interruptionNotification`)
- Supports background mode (with entitlements)
- Records in 30s segments with fallback to local Apple transcription after 5 cloud failures

Error Handling Includes:

- Disk space checks before recording
- Audio permission prompts
- Session fallback restart on crash/interruption

Data Model Design

SwiftData Models:

- `RecordingSession`

`fileName: String`
`createdAt: Date`
`@Relationship var segments: [RecordingSegment]`

- `RecordingSegment`

`fileName: String`
`transcription: String`
`createdAt: Date`
`@Relationship var session: RecordingSession`

Design Considerations:

- Indexed by date for fast grouping/filtering
- One-to-many relationship between sessions and segments
- Can scale to 10,000+ segments with smooth list virtualization and pagination
- Segment-level search via in-memory + SQLite FTS fallback

Known Issues

1. Apple fallback transcription may not always match Assembly AI quality
2. Speech recognition fails silently if permissions are revoked
3. AVAudioEngine fails to restart if multiple interruptions occur rapidly
4. Export only supports `.csv`; `.Json` or `.txt` pending
5. Settings are not persisted across app restarts (UserDefaults optional)

Performance Optimizations

- Streamed audio writing using buffer taps (low memory footprint)
- Lazy pagination (`loadMoreSessions`) for infinite scroll
- Real-time RMS sampling for UI waveform, not FFT (battery-efficient)
- All UI-bound logic runs on `MainActor` to avoid SwiftUI race conditions

Security & Privacy

- App requests microphone and speech recognition permission
- Audio files are stored locally and never uploaded unless using AssemblyAI
- API keys are stored in `Info.plist` for now
- No persistent analytics or telemetry

For more technical details, refer to the codebase and comments inline (e.g., AudioManager.swift, SessionListView.swift).