Overview

Build a robust iOS audio recording application that handles real-world audio challenges, integrates with backend transcription services, and efficiently manages large datasets with SwiftData. This assignment evaluates your expertise in audio system management, data persistence, network operations, and iOS best practices.

Submission:

1. GitHub repository with complete Xcode project
2. A robust iOS audio app that records audio, transcribes it in 30-second segments, and saves the recordings and transcriptions using SwiftData. It's to be designed to handle challenges like audio interruptions (Siri, iPhone), audio route change, or the combination of both and background recording.

Core Requirements

1. Audio Recording System

Build a production-ready audio recording system that demonstrates mastery of iOS audio frameworks.

Requirements:

- **Audio Session Management**: Properly configure session for recording with appropriate categories and options
- **Route Change Handling and Audio Interruption Recovery**: Gracefully handle audio route changes (headphones plugged/unplugged, Bluetooth connections, etc.). Handle phone calls, notifications, and other audio interruptions with automatic resumption
- **Background Recording**: Support recording continuation when app enters background (with proper entitlements)
- **Recording Quality**: Configurable audio quality settings (sample rate, bit depth, format)
- **Real-time Audio Monitoring**: Optional real-time audio level visualization

Technical Details:

- Use `AVAudioEngine`
- Implement proper audio session observation and notification handling
- Handle edge cases like recording failures, storage limitations, and permission denials

2. Timed Backend Transcription

Implement intelligent audio segmentation and backend transcription integration.

Requirements:

- **Automatic Segmentation**: Split recordings into configurable time segments (default: 30 seconds)
- **API Integration**: Send audio segments to a real transcription endpoint (openai whisper for example)
- **Retry Logic**: Implement exponential backoff for failed transcription requests
- **Concurrent Processing**: Handle multiple transcription requests efficiently
- **Secure Transmission**: Use proper encryption for audio data transmission
- **Offline Queuing**: Queue segments for transcription when network is unavailable
- **Fallback to local models:** if transcription failed consecutively for 5+ times, fallback to local speech to text models, i.e. apple transcription service/ local whisper model

3. SwiftData Integration

Implement a scalable data model using SwiftData for managing large datasets.

Data Model Requirements:

- **Recording Session and Transcription segments**: Store transcription text, processing metadata
- **Relationships**: Proper relationships between sessions, segments, and transcriptions
- **Performance**: Optimize for datasets with 1000+ sessions and 10,000+ segments

4. User Interface & Experience

Create an intuitive interface that scales to large datasets.

UI Requirements:

- **Recording Controls**: Start/stop/pause recording with visual feedback
- **Session List**: Efficient list displaying sessions grouped by date with search/filter
- **Session Detail**: Show segments with transcription status and text
- **Real-time Updates**: Live updates during recording and transcription
- **Performance**: Smooth scrolling with large datasets using proper list virtualization
- **Accessibility**: Full VoiceOver support and accessibility labels

Design Considerations:

- Handle loading states gracefully
- Implement pull-to-refresh and pagination
- Show transcription progress indicators

- Provide offline/online status indicators

5. Error Handling & Edge Cases

Demonstrate robust error handling across all system components.

Must Handle:

- Audio permission denied/revoked
- Insufficient storage space
- Network failures during transcription
- App termination during recording
- Audio route changes mid-recording
- Background processing limitations
- Transcription service errors
- Data corruption scenarios

Technical Specifications

Performance Requirements

- **Memory Management**: Efficient memory usage with large audio files
- **Battery Optimization**: Minimize battery drain during extended recording
- **Storage Management**: Implement audio file cleanup strategies

Security Requirements

- **Data Encryption**: Encrypt audio files at rest
- **Token Management**: Secure storage of API tokens using Keychain
- **Privacy**: Follow iOS privacy best practices for microphone access

Deliverables

1. Code Submission

- Complete Xcode project in GitHub repository
- Clear README with setup instructions
- Proper git history showing development process
- Code comments explaining complex audio/concurrency logic

2. Documentation

- **Architecture Document**: Explain your architectural decisions

- **Audio System Design**: Detail your approach to audio route changes and interruptions
- **Data Model Design**: Explain your SwiftData schema and performance optimizations
- **Known Issues**: Document any limitations or areas for improvement

3. Testing

- **Unit Tests**: Test core business logic and data models
- **Integration Tests**: Test audio system and API integration
- **Edge Case Tests**: Test error scenarios and recovery
- **Performance Tests**: Basic performance testing for large datasets

Bonus Points

- **Audio Visualization**: Real-time waveform or level meters
- **Export Functionality**: Export sessions in various formats
- **Advanced Search**: Full-text search across transcriptions
- **Custom Audio Processing**: Noise reduction or audio enhancement
- **Widget Support**: iOS widget for quick recording access

Questions?

Feel free to make reasonable assumptions and document them. We're looking for production-quality code that demonstrates deep iOS expertise and thoughtful problem-solving.

Good luck!