Music Chord Analyzer Mobile Application

Technical Implementation Documentation (For AI Development)

---

1. Project Overview

Application Name

Music Chord Analyzer (Working Title)

Core Objective

Develop a **cross-platform mobile application** using **React Native** that detects and displays musical chords in **real time** from any audio source, including external music streaming applications, using a **Picture-in-Picture (PIP)** floating overlay.

The system must prioritize: - Real-time audio processing - Minimal latency - No full pre-analysis of songs before playback - Continuous chord detection during playback - Cross-platform compatibility (Android + iOS where permitted)

---

2. Technology Stack

Frontend

- React Native (latest stable)
- TypeScript (mandatory)
- React Navigation
- Reanimated (UI animations)

Native Modules (Required)

Custom native bridges will be required: - Android: Kotlin - iOS: Swift

Audio Processing

- Native audio capture APIs
- Real-time DSP (Digital Signal Processing)
- FFT-based frequency analysis
- Chord detection algorithm

Suggested libraries (extendable): - Superpowered SDK / Oboe (Android) - AVAudioEngine (iOS) - RN Native Modules for microphone/system audio capture

State Management

- Zustand or Redux Toolkit

Storage

- AsyncStorage / MMKV
- Local database (SQLite or Realm)

---

3. Application Architecture

High-Level Architecture

```
React Native UI
        ↓
State Management Layer
        ↓
Native Bridge Layer
        ↓
Real-Time Audio Engine (Native)
        ↓
Chord Detection Engine
        ↓
Overlay / PIP Renderer
```

Design Principles

- Event-driven architecture
- Streaming audio pipeline
- Non-blocking UI thread
- Background processing threads
- GPU-safe overlay rendering

---

4. Core Functional Modules

1. Real-Time Audio Detection Engine
2. Chord Recognition Engine
3. Picture-in-Picture Overlay System
4. Local Music Player & Analyzer
5. Styling & Customization Engine
6. Playback Controller
7. Audio Manipulation Tools

---

5. Screen Architecture

The application contains **two primary screens**.

SCREEN 1 — Local Song Chord Analyzer

Purpose

Analyze songs stored locally on the device and provide full playback + chord tools.

Layout Sections

5.1 Song Selection Area

- Search icon
- Local device audio picker
- Supported formats:
- MP3
- WAV
- FLAC
- AAC
- OGG
- M4A
- Any OS-supported audio codec

5.2 Playback Controls

Must include: - Play / Pause - Skip Forward / Next - Skip Backward / Previous - Fast Forward - Rewind - Shuffle - Repeat - Stop - Seek Bar / Progress Bar - Current Position Time - Duration / Remaining Time

5.3 Song Interaction Features

- Favorite / Like
- Lyrics display
- Up Next / Queue system

5.4 Instrument Selection

User selects instrument type: - Guitar - Piano - Ukulele - Bass - Custom instruments (future expansion)

Display all detected chords mapped to chosen instrument.

5.5 Audio Manipulation Tools

Below chord display:

- Tempo control
- Transpose control
- Melody suppressor (vocal reduction)
- AB Loop system
- Set A marker
- Set B marker
- Loop playback between markers

Processing Requirement

⚠️ IMPORTANT: - Do NOT scan entire song at startup. - Chords must be detected progressively in real time.

SCREEN 2 — Real-Time Overlay (PIP Mode)

Purpose

Allow users to view chords while using external music apps.

Main Components

5.6 PIP Toggle Switch

- Toggle ON → Activate floating PIP window
- Toggle OFF → Disable overlay

When enabled: - App enters Picture-in-Picture overlay mode - Overlay remains visible over other applications

5.7 PIP Behavior Requirements

- Always-on-top floating window
- Draggable
- Resizable (optional enhancement)
- Minimal battery usage
- Transparent background support

5.8 Real-Time Detection

While in PIP: - Capture audio from system output or microphone (platform dependent) - Detect chords continuously - Update chord text in real time

Latency target: - < 150ms update delay

No buffering-based pre-analysis allowed.

---

6. PIP Styling Customization

Below the toggle button provide styling controls:

Text Customization

- Font family
- Font size
- Font weight
- Text color

- Background color
- Opacity
- Shadow
- Outline
- Alignment

Layout Options

- Single chord display
- Multi-line progression display
- Animated transitions

Changes must apply instantly to the PIP overlay.

---

7. Real-Time Audio Processing Requirements

Audio Input Sources

1. Local playback (Screen 1 player)
2. External app audio (Spotify, YouTube Music, etc.)

Detection Pipeline

```
Audio Stream
 → Noise Reduction
 → Windowing
 → FFT
 → Harmonic Analysis
 → Pitch Detection
 → Chord Classification
 → UI Update
```

Performance Constraints

- Continuous streaming processing
- No blocking operations
- Low CPU usage
- Battery optimized
- Background-safe execution

---

8. Chord Detection System

Functional Expectations

- Detect major, minor, diminished, augmented chords
- Detect seventh chords
- Handle inversions (basic level)
- Smooth chord transitions

- Confidence scoring

Output Model

```
{
  chord: "Cmaj7",
  confidence: 0.91,
  timestamp: 123.45
}
```

9. Permissions Required

Android

- RECORD_AUDIO
- SYSTEM_ALERT_WINDOW (overlay permission)
- FOREGROUND_SERVICE
- MEDIA_LIBRARY access

iOS

- Microphone permission
- Background audio mode
- PIP capability (platform restrictions apply)

10. Background Processing

- Audio engine runs in foreground service (Android)
- Separate processing thread
- UI updates via event emitter

11. Performance Targets

Metric	Target
Detection latency	<150ms
CPU usage	<25% average
Memory usage	<200MB
Startup time	<2 seconds

12. Data Persistence

Store locally: - User styling preferences - Favorites - Recent songs - Playback positions

---

13. Error Handling

Must gracefully handle: - Unsupported audio formats - Permission denial - Audio capture failure - Background restrictions

---

14. UX Requirements

- Minimal UI latency
- Smooth animation transitions
- Responsive overlay movement
- Non-intrusive overlay design

---

15. Future Expansion (Non-Blocking)

- AI chord prediction smoothing
- Cloud chord database
- MIDI input support
- Live instrument detection
- Auto key detection

---

16. Development Constraints (MANDATORY)

1. Real-time detection only.
2. No full-song preprocessing.
3. Must work while other music apps are active.
4. Must support all common audio formats.
5. React Native UI + Native audio engine architecture.

---

17. Deliverables Expected From AI Developer

- React Native project structure
- Native audio processing modules
- PIP overlay implementation
- Real-time chord detection engine
- Playback system
- Styling system
- Performance optimization

END OF DOCUMENT