Karaoke LRC -> Video (Offline Flutter) — Project Documentation

Intro (Hindi): Yeh document ek offline, backend-free Flutter app project describe karta hai jisme user Musixmatch-style real-time syncing se `.lrc` banaega aur woh LRC audio + background image/video ke saath on-device FFmpeg se karaoke MP4 mein convert ho jayega.

1. Project Overview Goal: Build a pure Flutter mobile app (Android/iOS) that allows users to: - Play a song and create real-time synced timestamps (Musixmatch-style tapping) to produce an `.lrc` file. - Convert that `.lrc` into an `.ass` subtitle file for enhanced karaoke effects. - Burn the `.ass` subtitles onto a background image or short background video and produce a final MP4 using FFmpeg, all on-device (no backend).

Scope & Constraints: - Everything runs on the device: audio playback, timestamping, LRC/ASS generation and FFmpeg rendering. - Heavy renders may be slow on low-end devices. Recommend offering lower-res presets. - No cloud storage, no user accounts.

2. Key Features 1. Import audio (MP3/WAV/AAC) and lyrics (plain text or LRC) 2. Musixmatch-style real-time tap-to-sync editor 3. Waveform visualization to help timing corrections 4. Edit timestamps manually (+/- ms) and undo/redo 5. Export LRC or ASS; render MP4 using ffmpeg_kit_flutter 6. Choose background image or looping background MP4; select resolution 7. Save projects locally and share final MP4

3. Tech Stack (On-device) - Flutter (stable channel) - Dart - Packages (recommended): - just_audio (playback) - audio_waveforms or waveform_flutter (waveform visualization) - ffmpeg_kit_flutter (on-device FFmpeg) - file_picker (file selection) - path_provider (save files) - share_plus (share outputs) - provider / riverpod (state management) - permission_handler (storage/audio permissions)

4. High-level App Flow 1. New Project -> select audio file 2. Load Lyrics -> paste or import TXT/LRC 3. Sync Mode -> Play audio; user taps a "Mark" button for each lyric line in real time 4. Review Mode -> Play and preview synced lyrics; adjust times manually; waveform view for accuracy 5. Export Step -> Choose background (image/video), resolution, and render 6. Render -> Convert LRC->ASS, run FFmpeg to create MP4, save & share

5. Data Models (Local) - Project: id, title, audioPath, lyricsPath, createdAt, modifiedAt - LyricLine: index, text, startMs, endMs - RenderPreset: resolution, bitrate, bgPath, outputPath

Storage: store meta as JSON in app folder (use path_provider getApplicationDocumentsDirectory).

6. LRC Format and Parsing LRC example: [00:12.34] This is the first line [00:16.78] Second line here

Parsing steps: - For each line, RegExp (\[(\d+):(\d+\.\d+)\](.*)) - Convert mm:ss.xx to milliseconds - When saving LRC from tap-sync, write lines with timestamps in ascending order

7. LRC -> ASS Conversion (Dart) ASS gives richer style and karaoke tags. Basic approach: 1. Read LRC lines and build timed events (start and end times). End time can be next line start or start + default display (e.g., 3s) for last line. 2. Create ASS header (Script Info + Styles) 3. For each line, format Dialogue: 0,Start,End,Style,,0,0,0,,Text or use karaoke {\\k} tags if per-syllable timing is known.

Note: Without per-syllable timing you can still highlight whole-line with style changes.

8. Example Converters & Commands A. Dart: LRC timestamp (ms) -> ASS timestamp ASS timestamp format: H:MM:SS.CS (centiseconds). Convert ms -> hours, minutes, seconds.centiseconds.

B. Example ASS header (minimal) [Script Info] Title: Karaoke ScriptType: v4.00+ PlayResX: 1280 PlayResY: 720

[V4+ Styles] Format: Name, Fontname, Fontsize, PrimaryColour, OutlineColour, Bold, Italic, BorderStyle, Outline, Shadow, Alignment, MarginL, MarginR, MarginV, Encoding Style:
Default,Arial,48,&H00FFFFFF,&H00000000,0,0,1,2,0,2,10,10,10,1

[Events] Format: Layer, Start, End, Style, Name, MarginL, MarginR, MarginV, Effect, Text

C. Example FFmpeg Commands (used in ffmpeg_kit_flutter) - Background image: ffmpeg -loop 1 -i bg.jpg -i song.mp3 -vf "ass=lyrics.ass,scale=1280:720" -shortest -c:v libx264 -preset medium -crf 18 -c:a aac output.mp4

- Background video: ffmpeg -i bg.mp4 -i song.mp3 -vf "ass=lyrics.ass" -map 0:v -map 1:a -c:v libx264 -preset medium -crf 18 -c:a aac -shortest output.mp4

Integration tips: pass these commands to FFmpegKit.executeAsync() and listen for progress and completion callbacks.

9. UI/UX Design (Screens) 1. Home / Projects — list projects with thumbnails 2. Import — choose audio & lyrics 3. Sync Editor (main): - Large waveform view with zoom and scrub - Current line preview & big "Mark" button - Line list with start times editable inline - Playback controls (play/pause/seek/loop) 4. Design — choose background, fonts, size, color 5. Render & Export — choose resolution and start render

UX tips: - Use large tap targets for Mark button - Show visual feedback (ripple + timestamp) when tap happens - Allow quick +/- 50ms adjustments - Implement undo for last mark

10. Implementation Details (important snippets) A. Marking a line (simplified) void markLine(int idx) async { final pos = await audioPlayer.position; // Duration lyrics[idx].startMs = pos.inMilliseconds; // Optionally set previous line's endMs to this start }

B. Save LRC Write lines like [mm:ss.xx] text using String formatting; use two decimal places for centiseconds.

C. Convert LRC -> ASS (pseudo) - Parse lines -> list of (startMs, text) - For each i: start = startMs[i]; end = startMs[i+1] or start+3000 - Format as ASS Dialogue lines

11. Performance & Memory - Rendering video on-device is CPU & memory heavy. - Offer presets: 720p (fast), 1080p (slower) - While rendering: show a progress spinner and prevent other heavy actions; consider running FFmpeg in background thread (ffmpeg_kit does this). - Recommend limiting background video length (loop or short clip) to reduce processing.

12. Testing & QA - Unit tests: LRC parser, ASS generator - Integration tests: small sample audio + LRC -> run ffmpeg command and verify output file exists - Manual QA: test on low, mid, high-end devices to estimate render time

13. Roadmap & Milestones MVP (2–3 weeks) - File import, tap-to-sync, save LRC, basic export with static bg image

Phase 2 (2 weeks) - Waveform editor, manual adjustments, LRC->ASS conversion with basic styles

Phase 3 (2–3 weeks) - Background video support, presets, better ASS styling, share feature

Phase 4 - Optional: local ML for auto-alignment (heavy), or optional cloud alignment as premium feature

14. Limitations & Legal - App must warn users about copyrighted songs and ownership - Rendering copyrighted music and sharing may have legal implications—include Terms & Disclaimer

15. Appendix - Useful resources & references to package docs (just_audio, ffmpeg_kit_flutter) - Example sample test files to include in repo

End of document.