432 Hz Music Player - Flutter

Umfassende Projektdokumentation und Spezifikation

1. Projektübersicht

1.1 Zusammenfassung

Das Ziel dieses Projekts ist die Entwicklung eines **professionellen Musik-Players mit 432 Hz Frequenz-Umwandlung** in Flutter. Die App soll auf Mobile-Geräten (Android/iOS) und Desktop laufen und lokale Musik-Dateien mit echtzeitiger Tonhöhen-Anpassung abspielen. Im Gegensatz zu bestehenden 432 Hz Playern soll die App moderne UX, Performance und erweiterte Features wie Equalizer, Sleep Timer und Cloud-Sync bieten.

1.2 Projektziele

- **Primär**: Funktionsfähiger 432 Hz Music Player mit überlegener UX gegenüber bestehenden Apps
- **Sekundär**: Freemium-Monetarisierung mit Premium-Features
- **Tertiär**: Skalierbar für weitere Frequenzen (528 Hz, Binaural Beats) und Streaming später

1.3 Zielgruppen

- **Primär**: Wellness & Spiritual Community (Meditation, Healing, Yoga)
- **Sekundär**: Audiophile, Menschen mit Interesse an Alternative Tuning Systems
- **Technisch**: Flutter Developer mit Audio-Engineering-Interesse

1.4 Marktanalyse

- **Existierende Konkurrenz**: 2-3 Apps im Play Store
- **Marktgröße**: 50-100k aktive Nutzer weltweit in dieser Nische
- **Monetarisierung**: Hohe Bereitschaft für $2.99-9.99 Premium-Features
- **Trend**: Wachsendes Interesse an Frequenz-Therapie und heilender Musik

2. Anforderungsanalyse

2.1 Funktionale Anforderungen (Priorisierung)

Phase 1: MVP (Kritisch)

1. **Audio-Datei-Management**
   - MP3, FLAC, WAV, OGG, AAC Formate
   - Lokale Dateien aus Speicher laden
   - Ordnerstruktur beibehalten
2. **Music Player Engine**
   - Play/Pause/Stop/Seek Funktionen
   - Aktuelle Wiedergabezeit & Duration Display

- Playlist-Navigation (Prev/Next)
- Shuffle & Repeat Modes

3. **432 Hz Feature (Kernfeature!)**
   - Echtzeit Pitch-Shift: 440 Hz → 432 Hz
   - Toggle-Button zum Ein/Ausschalten
   - Keine Qualitätsverluste
   - Funktioniert mit allen Dateitypen
4. **Simple UI**
   - Now-Playing-Screen
   - Playlist View
   - Minimalist Design
   - Basic Player Controls
5. **Datenspeicherung**
   - Playlist speichern/laden
   - Wiedergabe-Position merken
   - Benutzer-Präferenzen

Phase 2: Core Features (Hoch)

6. **Audio Visualizer**
   - Waveform Display während Wiedergabe
   - EQ-Analyzer (Echtzeit-Frequenz-Anzeige)
   - Mehrere Visualizer-Modi
7. **Erweiterte Player-Features**
   - Sleep Timer (Auto-Stop)
   - Time Stretch (Geschwindigkeit ohne Pitch-Änderung)
   - Gapless Playback (Lückenlose Wiedergabe)
8. **Equalizer**
   - 10-Band EQ (für verschiedene Soundprofile)
   - Vorsets (Bass Boost, Treble, Normal, etc.)
   - Custom Presets speichern
9. **Additional Frequencies**
   - 528 Hz Option ("Love Frequency")
   - 639 Hz Option (Harmony)
   - Frequenz-Auswahl-Menu

Phase 3: Premium Features (Mittel)

10. **Metadaten & Bibliothek**
    - ID3 Tag Reading
    - Album Art Display
    - Genre/Artist/Album Sorting
    - Metadata Editing

11. **Theme & Customization**
    - Dark/Light Mode
    - Accent Color Selection
    - Custom Backgrounds
    - Widget Customization
12. **Cloud Integration (Later)**
    - Playlist Cloud Backup
    - Cross-Device Sync
    - User Account System

2.2 Nicht-funktionale Anforderungen

Anforderung	Ziel	Begründung
Performance	< 500ms Start-Zeit	Mobile UX Standard
Memory	< 150MB im Idle	Smartphone-Constraints
Audio Latency	< 100ms UI Response	Smooth Playback-Start
CPU Usage	< 15% bei Wiedergabe	Battery Life
Pitch Accuracy	432 Hz ±2%	Audiophile Standard
Supported Formats	MP3, FLAC, WAV, AAC, OGG	Format Compatibility
UI Responsiveness	60 FPS	Smooth Interaction
Kompatibilität	Android 9+, iOS 12+	Breite Device-Abdeckung

3. Technologie-Stack

3.1 Kern-Technologien

- **Sprache**: Dart 3.5+
- **Framework**: Flutter 3.24+
- **Audio Engine**: just_audio (MIT License)
- **Build-System**: Flutter Build

3.2 Kritische Dependencies

Audio-Playback

```
dependencies:
  just_audio: ^0.9.36              # MIT - Audio Player Engine
  audio_service: ^0.18.12          # MIT - Background Playback Service
  audio_playback_state: ^0.1.0     # MIT - State Management für Audio

dev_dependencies:
  just_audio_platform_interface: ^4.0.0
```

Begründung: `just_audio` ist der beste Flutter Audio Player mit Pitch-Shift-Support, aktiver Entwicklung und MIT-License (volle kommerzielle Freiheit).

Musik-Dateien & Metadaten

```
dependencies:
  file_picker: ^6.1.1              # MIT - Datei-Auswahl Dialog
  metadata_god: ^0.3.0             # MIT - Audio Metadaten lesen
  path_provider: ^2.1.1            # BSD - File System Paths
  permission_handler: ^11.0.1      # MIT - Speicher-Zugriff
```

State Management

```
dependencies:
  riverpod: ^2.5.0                 # MIT - State Management
  flutter_riverpod: ^2.5.0         # MIT - Flutter Integration
  riverpod_generator: ^2.4.0       # MIT - Code Generation

dev_dependencies:
  build_runner: ^2.4.0
  riverpod_generator: ^2.4.0
```

Lokale Speicherung

```
dependencies:
  hive_flutter: ^1.1.0             # Apache 2.0 - NoSQL DB
  hive: ^2.2.3                     # Apache 2.0
  hive_generator: ^2.0.0

dev_dependencies:
  hive_generator: ^2.0.0
  build_runner: ^2.4.0
```

UI & UX

```
dependencies:
  flutter_localizations:           # Internationalisierung
  intl: ^0.19.0

  go_router: ^13.0.0               # Navigation

  cached_network_image: ^3.3.0     # Image Caching

  animations: ^2.0.11              # Flutter Animations

  device_info_plus: ^10.0.0        # Device Information

  flutter_hooks: ^0.20.5           # Hooks Pattern (optional aber hilfreich)
```

Utility

```
dependencies:
  freezed_annotation: ^2.4.0       # Immutable Data Classes
  json_serializable: ^6.7.0        # JSON Serialization
  logger: ^2.0.0                   # Logging

dev_dependencies:
```

```
  freezed: ^2.4.0
  build_runner: ^2.4.0
```

Monetarisierung

```
dependencies:
  in_app_purchase: ^2.0.0          # In-App Purchases
  google_mobile_ads: ^4.0.0        # Google AdMob (für Free Tier)
```

3.3 Platform-Spezifische Konfiguration

Android

```
android {
  minSdkVersion 24  // API Level 24 (Android 7.0)
  targetSdkVersion 34

  compileOptions {
    sourceCompatibility JavaVersion.VERSION_17
    targetCompatibility JavaVersion.VERSION_17
  }

  // Android-spezifische Audio-Plugins
  dependencies {
    implementation 'androidx.media3:media3-exoplayer:1.2.0'
    implementation 'androidx.media3:media3-session:1.2.0'
  }
}
```

iOS

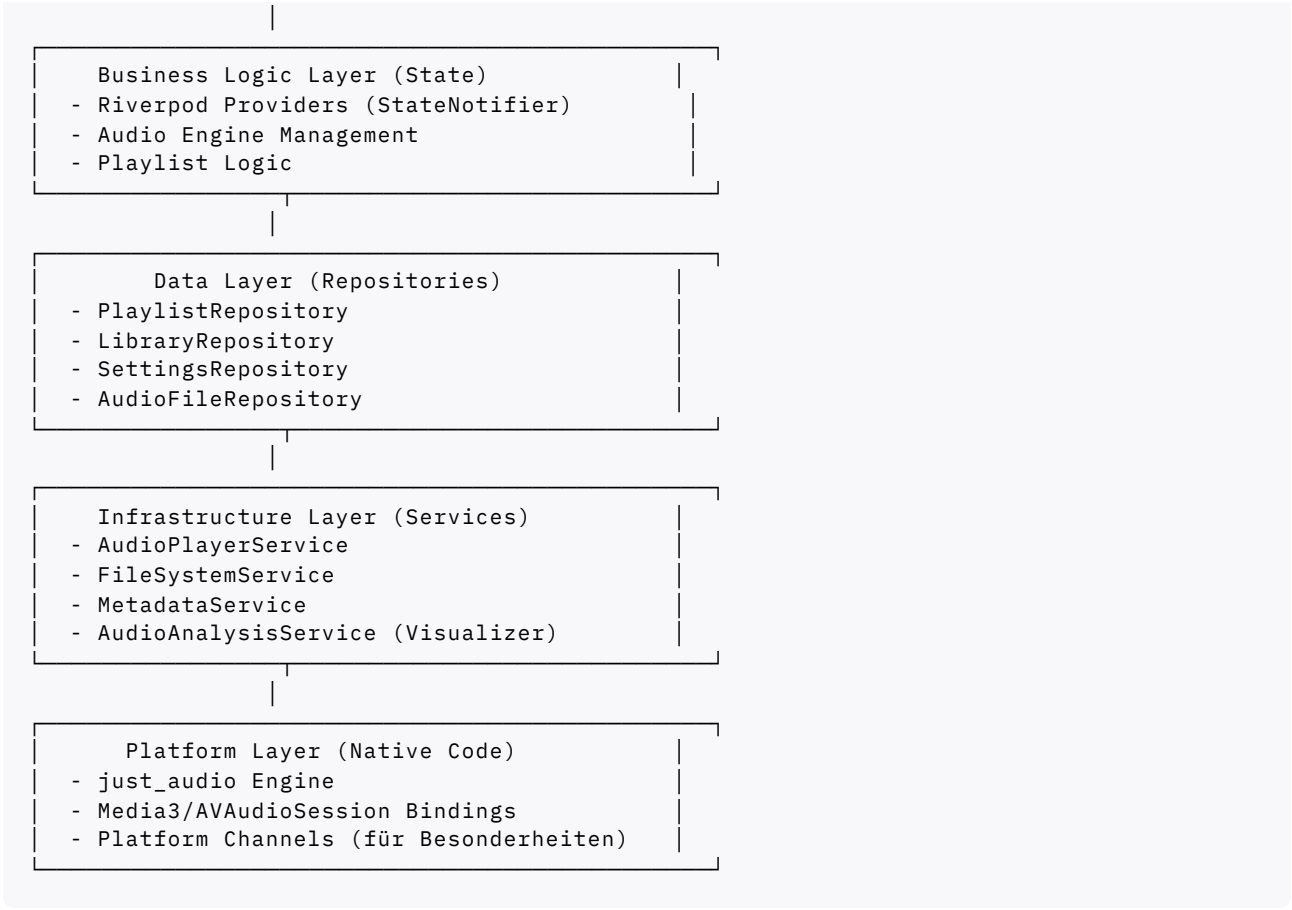
```
platform :ios, '12.0'

// Podfile Konfiguration
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
        '$(inherited)',
        'PERMISSION_CAMERA=1',
        'PERMISSION_PHOTO_LIBRARY=1',
      ]
    end
  end
end
```

4. Architektur-Design

4.1 Schichten-Architektur

```
┌─────────────────────────────────────────────────┐
│        UI Layer (Presentation)                  │
│  - Screens (HomeScreen, NowPlayingScreen)       │
│  - Widgets (PlayerControls, Visualizer)         │
│  - Theme &amp;amp; Localization                          │
└──────────────────┬──────────────────────────────┘
```



4.2 Ordnerstruktur

```
lib/
├── main.dart                          # App-Entry-Point
├── app/
│   ├── config/
│   │   ├── app_config.dart
│   │   ├── themes.dart
│   │   └── localization.dart
│   ├── constants/
│   │   ├── app_constants.dart
│   │   ├── audio_constants.dart
│   │   └── frequencies.dart
│   ├── routes/
│   │   ├── app_router.dart
│   │   └── route_names.dart
│   └── providers.dart
│
├── features/
│   ├── player/
│   │   ├── data/
│   │   │   ├── models/
│   │   │   │   └── audio_file_model.dart
│   │   │   ├── datasources/
│   │   │   │   ├── local/
│   │   │   │   └── audio_service/
│   │   │   └── repositories/
│   │   │       └── player_repository.dart
│   │   ├── domain/
│   │   │   ├── entities/
│   │   │   └── usecases/
│   │   ├── presentation/
│   │   │   ├── screens/
```

```
│   │   │   │   ├── now_playing_screen.dart
│   │   │   │   └── playlist_screen.dart
│   │   │   ├── widgets/
│   │   │   │   ├── player_controls.dart
│   │   │   │   ├── visualizer.dart
│   │   │   │   └── seek_bar.dart
│   │   │   └── providers/
│   │   │       └── player_provider.dart
│   │   └── player_feature.dart
│   │
│   ├── library/
│   │   ├── data/
│   │   ├── domain/
│   │   ├── presentation/
│   │   └── library_feature.dart
│   │
│   ├── equalizer/
│   │   ├── data/
│   │   ├── domain/
│   │   ├── presentation/
│   │   └── equalizer_feature.dart
│   │
│   └── settings/
│       ├── data/
│       ├── domain/
│       ├── presentation/
│       └── settings_feature.dart
│
├── shared/
│   ├── models/
│   │   ├── audio_file.dart
│   │   ├── playlist.dart
│   │   ├── equalizer_preset.dart
│   │   └── frequency_setting.dart
│   ├── services/
│   │   ├── audio/
│   │   │   ├── audio_player_service.dart
│   │   │   ├── metadata_service.dart
│   │   │   └── visualizer_service.dart
│   │   ├── file/
│   │   │   ├── file_system_service.dart
│   │   │   └── permission_service.dart
│   │   └── storage/
│   │       └── hive_service.dart
│   ├── utils/
│   │   ├── extensions/
│   │   ├── helpers/
│   │   └── formatters.dart
│   ├── theme/
│   │   ├── app_colors.dart
│   │   ├── app_text_styles.dart
│   │   └── app_theme.dart
│   ├── widgets/
│   │   └── common_widgets.dart
│   └── exceptions/
│       └── exceptions.dart
│
└── l10n/
    ├── app_en.arb
    ├── app_de.arb
    └── app_fr.arb
```

5. Datenmodelle

5.1 Audio File Model

```
@freezed
class AudioFile with _$AudioFile {
  const factory AudioFile({
    required String id,                    // Unique ID (MD5 von Pfad)
    required String filePath,              // Voll-Pfad zur Datei
    required String title,                 // Aus Metadata oder Dateiname
    required String? artist,
    required String? album,
    required String? albumArt,             // Base64 oder null
    required Duration duration,
    required DateTime dateAdded,
    required int? size,                    // Dateigröße in Bytes
    required String? mimeType,             // audio/mpeg, audio/flac, etc.
    required int playCount,                // Wie oft abgespielt
    required Duration? lastPlayedPosition, // Letzte Position
    required DateTime? lastPlayed,
  }) = _AudioFile;

  factory AudioFile.fromJson(Map&lt;String, dynamic&gt; json) =&gt;
      _$AudioFileFromJson(json);
}
```

5.2 Playlist Model

```
@freezed
class Playlist with _$Playlist {
  const factory Playlist({
    required String id,
    required String name,
    required List&lt;String&gt; audioFileIds,    // IDs der Audio-Dateien
    required DateTime created,
    required DateTime? modified,
    required int playCount,
    required bool isSmartPlaylist,         // Automatisch generiert?
  }) = _Playlist;

  factory Playlist.fromJson(Map&lt;String, dynamic&gt; json) =&gt;
      _$PlaylistFromJson(json);
}
```

5.3 Frequency Setting Model

```
@freezed
class FrequencySetting with _$FrequencySetting {
  const factory FrequencySetting({
    required String id,                    // "432", "528", etc.
    required double targetFrequency,       // 432.0, 528.0
    required String displayName,           // "432 Hz - Deep Peace"
    required String description,
    required double pitchShift,            // Berechnet: 432/440 semitones
    required bool isPremium,
    required Color accentColor,
  }) = _FrequencySetting;
}

// Vordefinierten Frequenzen
```

```
const frequencySettings = [
  FrequencySetting(
    id: '432',
    targetFrequency: 432.0,
    displayName: '432 Hz - Deep Peace',
    description: 'The frequency of harmony and healing',
    pitchShift: -0.31767,
    isPremium: false,
    accentColor: Color(0xFF6366F1),
  ),
  FrequencySetting(
    id: '528',
    targetFrequency: 528.0,
    displayName: '528 Hz - Miracles &amp;amp; Healing',
    description: 'The frequency of love and transformation',
    pitchShift: 0.37851,
    isPremium: true,
    accentColor: Color(0xFF06B6D4),
  ),
  // weitere Frequenzen...
];
```

5.4 Equalizer Preset Model

```
@freezed
class EqualizerPreset with _$EqualizerPreset {
  const factory EqualizerPreset({
    required String id,
    required String name,
    required List&amp;lt;double&amp;gt; gains,           // 10 Werte (-12 bis +12 dB)
    required bool isCustom,               // User-created?
  }) = _EqualizerPreset;

  factory EqualizerPreset.fromJson(Map&amp;lt;String, dynamic&amp;gt; json) =&amp;gt;
      _$EqualizerPresetFromJson(json);
}

// Standard-Presets
const defaultPresets = [
  EqualizerPreset(
    id: 'flat',
    name: 'Flat',
    gains: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    isCustom: false,
  ),
  EqualizerPreset(
    id: 'bass_boost',
    name: 'Bass Boost',
    gains: [5, 4, 3, 1, 0, -2, -4, -3, 0, 2],
    isCustom: false,
  ),
  // weitere...
];
```

6. State Management mit Riverpod

6.1 Core Providers

Audio Player Provider

```
@riverpod
class audioPlayer(AudioPlayerRef ref) {
  final player = AudioPlayer();

  ref.onDispose(() =&gt; player.dispose());

  return player;
}

@riverpod
Stream&lt;PlayerState&gt; playerState(PlayerStateRef ref) {
  final player = ref.watch(audioPlayerProvider);
  return player.playerStateStream;
}

@riverpod
Stream&lt;Duration&gt; duration(DurationRef ref) {
  final player = ref.watch(audioPlayerProvider);
  return player.durationStream;
}

@riverpod
Stream&lt;Duration&gt; position(PositionRef ref) {
  final player = ref.watch(audioPlayerProvider);
  return player.positionStream;
}
```

Frequency Provider

```
@riverpod
class currentFrequency(CurrentFrequencyRef ref)
    extends StateNotifier&lt;FrequencySetting&gt; {

  currentFrequency(this.ref)
      : super(frequencySettings.first);

  final CurrentFrequencyRef ref;

  void setFrequency(FrequencySetting frequency) {
    if (frequency.isPremium &amp;&amp; !ref.watch(isPremiumUserProvider)) {
      throw PremiumRequiredException();
    }
    state = frequency;
    ref.read(audioPlayerServiceProvider).setPitchShift(frequency.pitchShift);
  }
}

final currentFrequencyProvider =
    StateNotifierProvider&lt;CurrentFrequency, FrequencySetting&gt;((ref) {
  return CurrentFrequency(ref);
});
```

Playlist Provider

```
@riverpod
Future&lt;List&lt;AudioFile&gt;&gt; currentPlaylist(CurrentPlaylistRef ref) async {
  final playlistId = ref.watch(activePlaylistIdProvider);
  if (playlistId == null) return [];

  final repo = ref.watch(playlistRepositoryProvider);
  return repo.getPlaylistTracks(playlistId);
}

@riverpod
class currentTrackIndex(CurrentTrackIndexRef ref)
    extends StateNotifier&lt;int&gt; {

  currentTrackIndex() : super(0);

  void next() =&gt; state++;
  void previous() =&gt; state = (state - 1).clamp(0, state);
  void jumpTo(int index) =&gt; state = index;
}

final currentTrackIndexProvider =
    StateNotifierProvider&lt;CurrentTrackIndex, int&gt;((ref) {
  return CurrentTrackIndex();
});
```

6.2 Settings & UI Providers

```
@riverpod
class userSettings(UserSettingsRef ref) extends StateNotifier&lt;UserSettings&gt; {
  final repo = ref.watch(settingsRepositoryProvider);

  userSettings(this.ref) : super(UserSettings.defaultSettings());

  Future&lt;void&gt; setDarkMode(bool isDark) async {
    state = state.copyWith(isDarkMode: isDark);
    await repo.saveSetting('darkMode', isDark);
  }

  Future&lt;void&gt; setLanguage(Locale locale) async {
    state = state.copyWith(language: locale);
    await repo.saveSetting('language', locale.languageCode);
  }
}

@riverpod
bool isDarkMode(IsDarkModeRef ref) {
  return ref.watch(userSettingsProvider).isDarkMode;
}

@riverpod
bool isPremiumUser(IsPremiumUserRef ref) {
  return ref.watch(userSettingsProvider).isPremium;
}
```

7. Datenbank-Design mit Hive

7.1 Hive Boxes

```
@HiveType(typeId: 0)
class HiveAudioFile {
  @HiveField(0)
  final String id;

  @HiveField(1)
  final String filePath;

  @HiveField(2)
  final String title;

  @HiveField(3)
  final String? artist;

  @HiveField(4)
  final String? album;

  @HiveField(5)
  final int durationMs;

  @HiveField(6)
  final int playCount;

  @HiveField(7)
  final int? lastPlayedPositionMs;

  @HiveField(8)
  final DateTime dateAdded;
}

@HiveType(typeId: 1)
class HivePlaylist {
  @HiveField(0)
  final String id;

  @HiveField(1)
  final String name;

  @HiveField(2)
  final List&lt;String&gt; audioFileIds;

  @HiveField(3)
  final DateTime created;
}

@HiveType(typeId: 2)
class HiveUserSettings {
  @HiveField(0)
  final bool isDarkMode;

  @HiveField(1)
  final String? language;

  @HiveField(2)
  final String currentFrequencyId;

  @HiveField(3)
  final bool isPremium;

  @HiveField(4)
```

```
  final String? currentEqualizerPresetId;
}
```

7.2 Hive Service

```
class HiveService {
  late Box&lt;HiveAudioFile&gt; audioFileBox;
  late Box&lt;HivePlaylist&gt; playlistBox;
  late Box&lt;HiveUserSettings&gt; settingsBox;

  Future&lt;void&gt; init() async {
    await Hive.initFlutter();

    Hive.registerAdapter(HiveAudioFileAdapter());
    Hive.registerAdapter(HivePlaylistAdapter());
    Hive.registerAdapter(HiveUserSettingsAdapter());

    audioFileBox = await Hive.openBox&lt;HiveAudioFile&gt;('audioFiles');
    playlistBox = await Hive.openBox&lt;HivePlaylist&gt;('playlists');
    settingsBox = await Hive.openBox&lt;HiveUserSettings&gt;('settings');
  }

  // Query-Methoden
  List&lt;HiveAudioFile&gt; searchByTitle(String query) {
    return audioFileBox.values
        .where((file) =&gt; file.title.toLowerCase().contains(query.toLowerCase()))
        .toList();
  }

  List&lt;HiveAudioFile&gt; getRecentlyPlayed(int limit) {
    return audioFileBox.values
        .where((file) =&gt; file.lastPlayedPositionMs != null)
        .toList()
        .sublist(0, limit.clamp(0, audioFileBox.length));
  }
}
```

8. UI-Architektur und Komponenten

8.1 Haupt-Screens

Now Playing Screen Layout

```
┌─────────────────────────────────────┐
│ [Album Art - Large, Tappable]   │
│ (Toggle: Album Art / Visualizer)│
│                                 │
│       [Album Art/Visualizer]    │
│       (60% of screen)           │
│                                 │
├─────────────────────────────────────┤
│ Song Title (Scrolling Text)     │
│ Artist • Album                  │
├─────────────────────────────────────┤
│ [Frequency Selector - Chips]    │
│ [432 Hz] [528 Hz*] [Custom]     │
├─────────────────────────────────────┤
│ [Seek Bar mit Time Display]     │
│ 2:34 / 5:08                     │
├─────────────────────────────────────┤
```

```
| [Player Controls]                |
| [Shuffle] [Prev] [Play] [Next]   |
| [Loop]    [Repeat]  [Queue]      |
├──────────────────────────────────┤
| [Bottom Sheet / Drawer]          |
| &gt; Queue / Playlist              |
| &gt; Equalizer                     |
| &gt; Settings                      |
└──────────────────────────────────┘
```

Library Screen Layout

```
┌──────────────────────────────────┐
| [Search Bar]                     |
├──────────────────────────────────┤
| [Filter Tabs]                    |
| All | Favorites | Recently       |
├──────────────────────────────────┤
| Music List                       |
| ┌──────────────────────────┐     |
| |  Song Title             |    |
| | Artist • 3:45            |    |
| ├──────────────────────────┤     |
| |  Song 2                 |    |
| | Artist • 4:12            |    |
| └──────────────────────────┘     |
| [Load More / Infinite Scroll]    |
└──────────────────────────────────┘
```

8.2 Wichtige Widgets

Now Playing Widget

```
class NowPlayingScreen extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final currentTrack = ref.watch(currentAudioFileProvider);
    final playerState = ref.watch(playerStateProvider);
    final currentFrequency = ref.watch(currentFrequencyProvider);

    return Scaffold(
      body: SafeArea(
        child: Column(
          children: [
            // Album Art / Visualizer
            Expanded(
              flex: 3,
              child: GestureDetector(
                onTap: () =&gt; ref.read(toggleVisualizerProvider),
                child: ref.watch(showVisualizerProvider)
                    ? _buildVisualizer()
                    : _buildAlbumArt(currentTrack?.albumArt),
              ),
            ),

            // Frequency Selector
            Padding(
              padding: EdgeInsets.all(16),
              child: FrequencySelector(
                currentFrequency: currentFrequency,
                onFrequencyChanged: (freq) {
```

```
                  ref.read(currentFrequencyProvider.notifier)
                      .setFrequency(freq);
                },
              ),
            ),

            // Seek Bar
            Padding(
              padding: EdgeInsets.symmetric(horizontal: 16),
              child: StreamBuilder&lt;DurationState&gt;(
                stream: _durationStateStream(ref),
                builder: (context, snapshot) {
                  final durationState = snapshot.data;
                  return SeekBar(
                    duration: durationState?.total ?? Duration.zero,
                    position: durationState?.current ?? Duration.zero,
                    onChangeEnd: (newPosition) {
                      ref.read(audioPlayerProvider).seek(newPosition);
                    },
                  );
                },
              ),
            ),

            // Player Controls
            PlayerControls(),
          ],
        ),
      ),
    );
  }

  Widget _buildAlbumArt(String? artData) {
    return Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          colors: [Color(0xFF6366F1), Color(0xFF8B5CF6)],
        ),
      ),
      child: Center(
        child: artData != null
            ? Image.memory(base64Decode(artData))
            : Icon(Icons.music_note, size: 100),
      ),
    );
  }

  Widget _buildVisualizer() {
    return AudioVisualizer();
  }
}
```

Visualizer Widget

```
class AudioVisualizer extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final playerState = ref.watch(playerStateProvider);

    return CustomPaint(
      painter: VisualizerPainter(
        frequency: playerState == PlayerState.playing
            ? getAudioFrequencies()
            : [],
```

```
      ),
      size: Size.infinite,
    );
  }
}

class VisualizerPainter extends CustomPainter {
  final List&lt;double&gt; frequency;

  VisualizerPainter({required this.frequency});

  @override
  void paint(Canvas canvas, Size size) {
    // Zeichne Bars für jede Frequenz
    final paint = Paint()
      ..color = Colors.indigo
      ..strokeWidth = 3
      ..strokeCap = StrokeCap.round;

    final width = size.width / frequency.length;

    for (int i = 0; i &lt; frequency.length; i++) {
      final x = i * width + width / 2;
      final height = frequency[i] * size.height / 100;

      canvas.drawLine(
        Offset(x, size.height),
        Offset(x, size.height - height),
        paint,
      );
    }
  }

  @override
  bool shouldRepaint(VisualizerPainter oldDelegate) {
    return oldDelegate.frequency != frequency;
  }
}
```

9. 432 Hz Core Implementation

9.1 Pitch Shift Berechnung

```
class FrequencyCalculator {
  // Standardfrequenz: 440 Hz
  static const double standardFrequency = 440.0;

  /// Berechne Pitch-Shift in Semitones für Zielfrequenz
  static double calculatePitchShift(double targetFrequency) {
    // Formel: 12 * log2(target / standard)
    final ratio = targetFrequency / standardFrequency;
    return 12 * log(ratio) / log(2);
  }

  /// 432 Hz Pitch Shift
  static const double PITCH_432_HZ = -0.31767;

  /// 528 Hz Pitch Shift
  static const double PITCH_528_HZ = 0.37851;

  /// Konvertiere Pitch in Speed (ohne Pitch zu verändern)
  static double pitchToSpeed(double pitchShift) {
```

```
    // Speed = 2^(pitchShift/12)
    return pow(2, pitchShift / 12).toDouble();
  }
}
```

9.2 Audio Player Wrapper mit Pitch-Shift

```
class AudioPlayerService {
  late AudioPlayer _player;
  double _currentPitchShift = 0;

  AudioPlayerService() {
    _player = AudioPlayer();
  }

  /// Spiele Audio-Datei mit optionalen Pitch-Shift ab
  Future&lt;void&gt; playFile(
    String filePath, {
    Duration? startPosition,
    double pitchShift = 0,
  }) async {
    try {
      await _player.setAudioSource(
        AudioSource.file(filePath),
        initialPosition: startPosition,
      );

      _currentPitchShift = pitchShift;
      await _applyPitchShift(pitchShift);

      await _player.play();
    } catch (e) {
      logger.e('Error playing file: $e');
      rethrow;
    }
  }

  /// Wende Pitch-Shift an
  Future&lt;void&gt; _applyPitchShift(double pitchShift) async {
    _currentPitchShift = pitchShift;

    // just_audio unterstützt Pitch direkt:
    // Negative Werte = tiefer, positive = höher
    // -0.31767 für 432 Hz (~-0.5 Semitone)

    await _player.setPitchShift(pitchShift);
  }

  /// Ändere Pitch-Shift live
  Future&lt;void&gt; setPitchShift(double pitchShift) async {
    await _applyPitchShift(pitchShift);
  }

  /// Get Player Streams für UI
  Stream&lt;PlayerState&gt; get playerStateStream =&gt; _player.playerStateStream;
  Stream&lt;Duration?&gt; get durationStream =&gt; _player.durationStream;
  Stream&lt;Duration&gt; get positionStream =&gt; _player.positionStream;

  void dispose() =&gt; _player.dispose();
}
```

9.3 Riverpod Integration für 432 Hz

```
@riverpod
class frequencyManager(FrequencyManagerRef ref)
    extends StateNotifier&lt;FrequencySetting&gt; {

  late final audioPlayerService = ref.watch(audioPlayerServiceProvider);

  frequencyManager(this.ref)
      : super(frequencySettings.first); // Default: 432 Hz

  final FrequencyManagerRef ref;

  Future&lt;void&gt; setFrequency(FrequencySetting frequency) async {
    // Check Premium wenn nötig
    if (frequency.isPremium) {
      final isPremium = ref.watch(isPremiumUserProvider);
      if (!isPremium) {
        throw PremiumFeatureException(
          'Frequency ${frequency.displayName} requires premium'
        );
      }
    }

    // Wende Pitch-Shift an
    await audioPlayerService.setPitchShift(frequency.pitchShift);

    // Update State
    state = frequency;

    // Speichere Preference
    await ref
        .read(settingsRepositoryProvider)
        .saveSetting('frequency', frequency.id);
  }
}

final frequencyManagerProvider =
    StateNotifierProvider&lt;FrequencyManager, FrequencySetting&gt;((ref) {
  return FrequencyManager(ref);
});
```

10. Implementierungs-Roadmap

Phase 1: MVP (Wochen 1-2)

- [ ] Flutter Projekt Setup
- [ ] just_audio Integration
- [ ] Hive Datenbank Setup
- [ ] Audio-Dateien laden (File Picker)
- [ ] Basic Now-Playing UI
- [ ] Play/Pause/Seek Kontrollen
- [ ] **432 Hz Toggle-Feature**
- [ ] Playlist Speicherung

Phase 2: Core Features (Wochen 3-4)

- [ ] Visualizer Widget
- [ ] Frequency Selector (432, 528 Hz)
- [ ] Library Screen mit Search
- [ ] Shuffle & Repeat Modi
- [ ] Sleep Timer
- [ ] Recent/Favorites Funktionalität

Phase 3: Polish & Premium (Wochen 5)

- [ ] 10-Band Equalizer
- [ ] Theme Selection (Dark/Light)
- [ ] Lokalisation (EN, DE, weitere)
- [ ] Settings Screen
- [ ] In-App Purchase Setup
- [ ] Ad Integration (Free Tier)

Phase 4: Veröffentlichung (Woche 6)

- [ ] Google Play Listing
- [ ] Screenshots & Promo Video
- [ ] Beta Testing
- [ ] App Review & Approval
- [ ] Launch & Marketing

11. Performance-Richtlinien

Memory Management

- Cache Album-Art (max 50 neueste)
- Dispose Player bei Screen-Wechsel
- Nutze `AutoDispose` für Riverpod Providers
- Lazy-Load Playlist Items

Audio Processing

- Pitch-Shift nutze native just_audio
- FFT für Visualizer bei <10ms Update-Rate
- Backgroundservice für Playback über Screen Lock

UI Performance

- Nutze `ListView.builder` für lange Listen
- Debounce Search-Queries (300ms)
- Nutze `const` Constructors aggressiv

12. Testing-Strategie

Unit Tests

```
test('432 Hz pitch shift berechnet korrekt', () {
  const targetFrequency = 432.0;
  final pitchShift = FrequencyCalculator.calculatePitchShift(targetFrequency);

  expect(pitchShift, closeTo(-0.31767, 0.001));
});

test('Audio-Datei parsing extrahiert Metadata', () {
  // Test mit lokaler Test-Datei
});
```

Widget Tests

```
testWidgets('Frequency selector zeigt alle Frequenzen', (tester) async {
  await tester.pumpWidget(
    MaterialApp(
      home: FrequencySelector(
        onFrequencyChanged: (_) {},
      ),
    ),
  );

  expect(find.text('432 Hz - Deep Peace'), findsOneWidget);
  expect(find.text('528 Hz - Miracles &amp;amp; Healing'), findsOneWidget);
});
```

13. Monetarisierung & Launch-Strategie

Freemium Model

Free Tier:

- 432 Hz (Standard)
- Basic Player
- Ads (unobtrusive)
- 3 Playlists

Premium ($2.99 one-time):

- Ad-free
- 528 Hz + weitere Frequenzen
- Unlimited Playlists
- Custom Equalizer Presets
- Cloud Backup (später)
- Early Access zu neuen Features

Launch-Strategie

1. **Beta**: TestFlight/Firebase App Distribution (100 Testers)
2. **Soft Launch**: Einzelne Länder (DE, US)
3. **Full Launch**: Global Rollout mit marketing
4. **Update Strategy**: Monthly Feature Updates

Marketing Keywords für Play Store

- 432 Hz Music Player
- Healing Music App
- Frequency Tuning
- Audio Healing
- Spiritual Music Player
- Meditation Music
- Music Therapy
- Alternative Tuning

14. Häufige Fehler vermeiden

1. ✖ **Keine Format-Support**: Nur MP3 unterstützen
   ✔ **Lösung**: Mindestens MP3, FLAC, WAV, AAC
2. ✖ **Pitch-Shift klingt holprig**: Billige Implementation
   ✔ **Lösung**: native just_audio Engine nutzen
3. ✖ **App crasht bei großen Playlists**: Alles laden
   ✔ **Lösung**: Pagination / Lazy Loading
4. ✖ **Kein Error Handling**: Datei nicht gefunden
   ✔ **Lösung**: Try-catch + user-freundliche Meldungen
5. ✖ **Memory Leaks**: Player nicht disposed
   ✔ **Lösung**: Consumer/AutoDispose nutzen

15. Erfolgs-Metriken

KPIs zum Tracken:

- Downloads (Ziel: 1k im ersten Monat)
- Daily Active Users (DAU)
- Conversion Rate Free → Premium (Ziel: 5-10%)
- Average Session Duration (Ziel: > 30 min)
- Retention Rate (Ziel: >40% nach 7 Tagen)
- Rating (Ziel: > 4.5 Stars)

16. Roadmap für Zukunft (Post-Launch)

Q2 2025

- [ ] Streaming Integration (Spotify/YouTube Music)
- [ ] Offline Sync
- [ ] Social Sharing (Playlists)

Q3 2025

- [ ] WebOS App (LG Smart TV)
- [ ] Tizen App (Samsung Smart TV)

Q4 2025

- [ ] Binaural Beats Feature
- [ ] Meditation Guided Sessions
- [ ] Community Playlists