Claude.md - FTL Hi-Res Audio Player Documentation

Project Overview

Project Name: FTL Hi-Res Audio Player
Type: Android Music Player Application
Development Approach: AI-Assisted Development with Claude
Primary Goal: Create the world's most advanced audiophile music player

Development Philosophy

This project follows an iterative, AI-assisted development methodology where each feature is designed, implemented, tested, and refined in collaboration with Claude AI. The approach emphasizes:

- **Feature-driven development** - One feature at a time
- **Continuous testing and iteration**
- **Performance-first architecture**
- **User experience excellence**
- **Technical innovation**

Project Structure

```
ftl-hi-res-audio-player/
├── app/                    # Main Android application
│   ├── src/main/kotlin/    # Kotlin source code
│   ├── src/main/res/       # Android resources
│   └── src/test/           # Unit tests
├── audio-engine/           # Native audio processing
│   ├── cpp/                # C++ audio engine
│   └── jni/                # JNI bindings
├── ml-models/              # Machine learning components
│   ├── tensorflow-lite/    # TensorFlow Lite models
│   └── training-data/      # ML training datasets
├── docs/                   # Project documentation
│   ├── architecture/       # System architecture docs
│   ├── api/                # API documentation
│   └── user-guides/        # User documentation
├── scripts/                # Build and automation scripts
├── tests/                  # Integration and UI tests
└── assets/                 # Design assets and resources
```

Claude Interaction Guidelines

When to Use Claude Desktop

- **Project planning and architecture** discussions
- **Complex problem solving** that requires back-and-forth
- **Requirements clarification** and feature specification
- **Code review and optimization** discussions
- **Debugging complex issues** that need explanation

When to Use Claude CLI

- **Active development** and code implementation
- **File creation and modification**
- **Running tests and build processes**
- **Automated code generation**
- **Large codebase refactoring**

Context Management

Project Context Setup

Always provide Claude with:

1. **Current milestone** and feature being developed
2. **Recent changes** made to the codebase
3. **Specific requirements** for the current task
4. **Performance constraints** and technical limitations
5. **Integration points** with existing features

File Organization for Claude

- Keep related files in logical directories
- Use descriptive naming conventions
- Maintain clear separation between layers (UI, business logic, data)
- Document complex algorithms inline
- Keep configuration in easily accessible files

Development Workflow

Feature Development Cycle

1. **Planning Phase** (Claude Desktop)
   - Define feature requirements
   - Design technical approach
   - Identify dependencies and risks
   - Create implementation plan
2. **Implementation Phase** (Claude CLI)
   - Set up development environment
   - Implement core functionality
   - Write unit tests
   - Create initial UI components
3. **Testing Phase** (Claude CLI/Desktop)
   - Run automated tests
   - Perform manual testing
   - Identify and fix bugs
   - Performance optimization
4. **Integration Phase** (Claude CLI)
   - Integrate with existing features
   - Update documentation
   - Commit code changes
   - Prepare for next feature

Code Quality Standards

Kotlin/Android Standards

- Follow Android architecture guidelines (MVVM, Clean Architecture)
- Use Jetpack Compose for UI development
- Implement proper dependency injection with Hilt
- Write comprehensive unit tests (80%+ coverage)
- Use coroutines for asynchronous operations

Performance Standards

- Audio latency: <50ms total
- UI responsiveness: 60fps minimum, 120fps target
- Memory usage: <200MB for UI, <50MB per audio stream
- Battery efficiency: <10% drain per hour during playback
- CPU usage: <15% during normal operation

Security Standards

- Encrypt sensitive user data
- Use secure communication protocols
- Implement proper input validation
- Follow Android security best practices
- Protect against reverse engineering

Technical Architecture

Core Components

Audio Engine

- **Native C++ processing** for maximum performance
- **Multi-threaded architecture** for real-time processing
- **Custom DSP algorithms** for EQ and effects
- **Format support** for all audiophile formats
- **Hardware integration** with external DACs

User Interface

- **Jetpack Compose** for modern, reactive UI
- **Custom animations** with 120fps target
- **Cyberpunk aesthetic** matching project theme
- **Responsive design** for all screen sizes
- **Accessibility support** for all users

Machine Learning

- **TensorFlow Lite** for on-device inference
- **Audio analysis models** for intelligent features

- **User behavior learning** for personalization
- **Privacy-first approach** with local processing
- **Continuous model improvement**

Data Architecture

Local Storage

```
Room Database:
├── music_library/          # Song metadata and file paths
├── playlists/              # User-created playlists
├── eq_presets/             # Equalizer configurations
├── user_preferences/       # App settings and preferences
├── listening_history/      # Play counts and timestamps
└── audio_analysis/         # Cached frequency analysis data
```

Cloud Storage (Optional)

- **Encrypted user profiles** with preference backup
- **Cross-device synchronization** for playlists and settings
- **Anonymous usage analytics** for product improvement
- **Secure API communication** with backend services

Testing Strategy

Unit Testing

- **Coverage target:** 80%+ for all Kotlin code
- **Testing framework:** JUnit 5, Mockito, Truth
- **Audio testing:** Custom test harness for audio quality
- **Performance testing:** Automated benchmarks

Integration Testing

- **UI testing:** Espresso for Android UI components
- **Audio pipeline testing:** End-to-end audio processing tests
- **Cross-feature testing:** Ensure features work together
- **Device compatibility:** Test on multiple Android versions

User Testing

- **Alpha testing:** Internal testing with development team
- **Beta testing:** Closed testing with audiophile community
- **Usability testing:** User experience validation
- **Performance testing:** Real-world usage scenarios

Deployment and Distribution

Development Builds

- **Debug builds** for development testing
- **Automated CI/CD** pipeline for continuous integration
- **Internal distribution** via Firebase App Distribution
- **Performance monitoring** with crash reporting

Release Builds

- **Google Play Console** for production distribution
- **Staged rollout** to minimize risk
- **A/B testing** for feature validation
- **User feedback collection** and analysis

Version Management

- **Semantic versioning** (MAJOR.MINOR.PATCH)
- **Feature flags** for gradual feature rollout
- **Backward compatibility** maintenance
- **Database migration** strategies

Success Metrics

Technical Metrics

- **Audio quality:** THD+N < 0.001%
- **Performance:** <50ms latency, >8h battery life
- **Stability:** <0.1% crash rate
- **Compatibility:** 99%+ Android devices (API 24+)

User Metrics

- **App Store rating:** >4.7/5 stars

- **User retention:** >80% after 30 days
- **Feature adoption:** >60% users engage with advanced features
- **Support efficiency:** <1% users need support

Business Metrics

- **Market position:** Top 3 in audiophile category
- **Revenue growth:** $100K+ monthly within 6 months
- **User acquisition:** 100K+ active users within first year
- **Industry recognition:** Featured by major tech publications

Communication Guidelines

Working with Claude

Effective Prompting

- **Be specific** about requirements and constraints
- **Provide context** about current project state
- **Ask focused questions** rather than broad requests
- **Include error messages** and specific problems
- **Specify output format** (code, documentation, analysis)

Context Preservation

- **Save important discussions** for future reference
- **Document architectural decisions** made with Claude
- **Keep track of feature implementation** progress
- **Maintain changelog** of Claude-assisted developments

Iteration Process

- **Start simple** and build complexity gradually
- **Test frequently** and validate assumptions
- **Refactor continuously** for better code quality
- **Document learnings** from each iteration
- **Celebrate achievements** and milestones

Project Timeline and Milestones

Phase 1: Foundation (Weeks 1-2)

- Basic audio playback engine
- Cyberpunk UI foundation
- Development environment setup
- Initial testing framework

Phase 2: Audio Enhancement (Weeks 3-4)

- 10-band graphic equalizer
- Sub-bass enhancement system
- Audio format support expansion
- Performance optimization

Phase 3: Advanced Features (Weeks 5-8)

- 32-band parametric EQ upgrade
- Hi-res audio support
- Neural network visualizations
- Advanced UI components

Phase 4: Intelligence Layer (Weeks 9-12)

- AI-powered audio analysis
- Smart EQ recommendations
- Voice control integration
- Machine learning features

Phase 5: Premium Features (Weeks 13-16)

- Cloud synchronization
- Advanced audio analysis tools
- Premium subscription features
- Final polish and optimization

Maintenance and Evolution

Ongoing Development

- **Regular updates** with new features and bug fixes
- **Performance monitoring** and optimization
- **User feedback integration** into development roadmap
- **Technology stack updates** and improvements

Community Engagement

- **Open source components** where appropriate
- **Developer documentation** for extensibility
- **User community building** around the product
- **Feedback collection** and feature request management

This document serves as the living guide for the FTL Hi-Res Audio Player project development. It should be updated regularly as the project evolves and new insights are gained through AI-assisted development.