Focus Music Processor - Complete Documentation

Table of Contents

Project Overview

The Focus Music Processor is a sophisticated web application that transforms regular music into focus-optimized versions using AI-powered audio separation and processing. The system consists of a FastAPI backend for audio processing and a React frontend for user interaction.

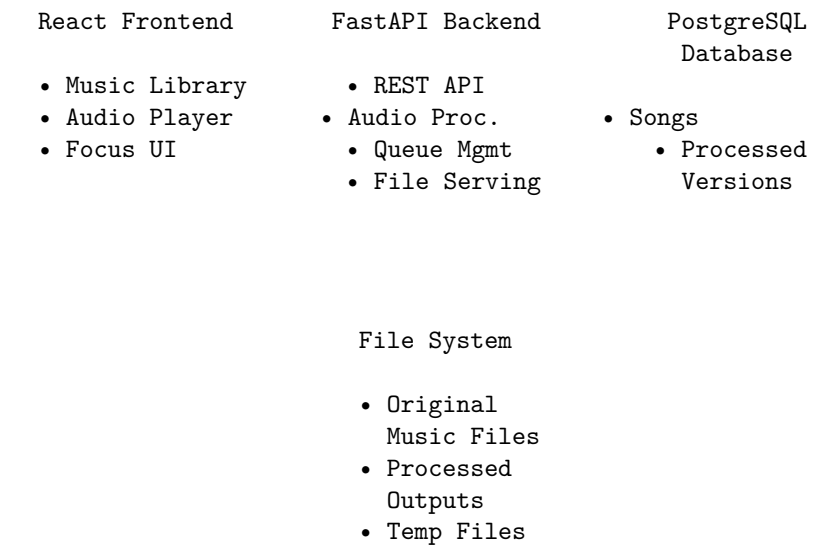
Key Features

- **Music Library Management**: Automatic scanning and metadata extraction
- **AI-Powered Audio Processing**: Uses Spleeter for stem separation
- **Focus Mode Processing**: Customizable intensity and BPM targeting
- **Real-time Processing Queue**: Background job management with progress tracking
- **Streaming Audio Playback**: Efficient audio delivery
- **Modern Web Interface**: Responsive React frontend with Tailwind CSS

Technology Stack

- **Backend**: FastAPI (Python), SQLAlchemy, Spleeter, librosa, PostgreSQL
- **Frontend**: React, Tailwind CSS, Axios
- **Audio Processing**: Spleeter, librosa, pydub, soundfile
- **Database**: PostgreSQL with SQLAlchemy ORM

Architecture



Data Flow

1. **Music Scanning**: Backend scans music folder, extracts metadata
2. **Library Display**: Frontend fetches and displays song list
3. **Focus Processing**: User requests focus version → queued background job
4. **Audio Transformation**: Spleeter separation → audio effects → MP3 export
5. **Playback**: Frontend streams original or processed audio

---

System Requirements

Hardware

- **CPU**: Multi-core processor (audio processing is CPU-intensive)
- **RAM**: Minimum 4GB, recommended 8GB+
- **Storage**: 2-3x music library size for processed files
- **Audio**: Sound card/headphones for playback

Software

- **Python**: 3.8+ (tested with 3.8, 3.13)
- **Node.js**: 16+ for React frontend
- **PostgreSQL**: 12+ for database

- **Operating System**: Windows 10+, macOS 10.15+, Linux

Installation & Setup

1. Clone Repository

```
git clone <repository-url>
cd music-processor-full
```

2. Backend Setup

```
cd backend

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Configure environment
cp .env.example .env  # Edit with your settings
```

3. Frontend Setup

```
cd frontend
npm install
```

4. Database Setup

```
# Create PostgreSQL database
createdb focus_music

# Configure DATABASE_URL in .env
echo "DATABASE_URL=postgresql://user:password@localhost:5432/focus_music" >> .env

# Initialize database
python database.py
```

5. Music Folder Setup

```
# Configure MUSIC_FOLDER in .env
echo "MUSIC_FOLDER=../music" >> .env

# Add music files to the music folder
```

6. Start Services

```
# Terminal 1 - Backend
cd backend
python app.py

# Terminal 2 - Frontend
cd frontend
npm start
```

Access the application at `http://localhost:3000`

Backend Components

1. `app.py` - Main Application

Purpose: FastAPI application entry point and route definitions

Key Features: - Modern lifespan event handling for startup/shutdown - CORS middleware for React frontend communication - RESTful API endpoints for music library and streaming - Database connection management - Processing queue integration

Important Routes: - `GET /` - Health check - `GET /songs` - Library with pagination and search - `GET /audio/{song_id}` - Audio streaming - `POST /scan` - Trigger music library rescan - `/focus/*` - Focus processing endpoints (via router)

2. `audio_utils.py` - Audio Analysis & Utilities

Purpose: Comprehensive audio analysis and utility functions

Key Classes: - **AudioTimeCalculator**: Synchronization between original/processed audio - **AudioAnalyzer**: Advanced audio analysis (tempo, key, complexity) - **AudioValidator**: Input validation for processing requests

Key Functions: - `analyze_audio_file()` - Extracts tempo, key, spectral features - `calculate_sync_position()` - Critical for seamless playback transitions - `estimate_processing_time()` - User experience enhancement - `validate_processing_request()` - Prevents invalid processing jobs

3. `focus_processor.py` - Audio Processing Core

Purpose: AI-powered audio transformation for focus enhancement

Processing Pipeline: 1. **Spleeter Separation**: Isolates vocals/accompaniment using deep learning 2. **Focus Transformation**: Applies audio effects based on intensity 3. **Tempo Adjustment**: Optional BPM targeting 4. **MP3 Export**: High-quality compressed output

Key Methods: - `process_for_focus()` - Main processing orchestrator - `apply_focus_effects()` - Audio effect application - `numpy_to_mp3()` - Efficient audio export

4. `processing_queue.py` - Job Management

Purpose: Background processing with progress tracking

Key Classes: - **JobStatus**: Enum for job states (pending, running, completed, failed, cancelled) - **ProcessingJob**: Job data structure with progress tracking - **ProcessingQueue**: In-memory queue with concurrent job management

Features: - Asynchronous job processing - Progress tracking and ETAs - Job cancellation support - Error handling and recovery

5. `models.py` - Database Schema

Purpose: SQLAlchemy ORM models for data persistence

Models: - **Song**: Music metadata and file information - **ProcessedVersion**: Focus-processed file variants

Design Principles: - Nullable fields for graceful metadata handling - Indexed fields for performance - Timestamp tracking for debugging - Availability flags for missing file handling

6. `music_scanner.py` - Library Management

Purpose: Automatic music library discovery and metadata extraction

Features: - Recursive folder scanning - Multi-format support (MP3, M4A, FLAC, WAV) - Robust metadata extraction using mutagen - Graceful error handling for corrupted files

7. `database.py` - Database Management

Purpose: Database connection and session management

Features: - SQLAlchemy engine configuration - Session factory with dependency injection - Table creation and management - Development utilities (reset, etc.)

8. `config.py` - Configuration Management

Purpose: Centralized application settings

Settings: - Database connection strings - Music folder paths - Supported audio formats - Server configuration

---

Frontend Components

1. `App.js` - Main Application Component

Purpose: Root React component managing global state

State Management: - `currentSong` - Currently selected/playing track - `library` - Complete music library - `focusExpanded` - Focus controls visibility

Key Features: - Song selection handling - Library management - Focus mode integration - Keyboard shortcuts (future)

2. `components/Library.js` - Music Library Interface

Purpose: Display and interact with music collection

Features: - Paginated song listing - Search functionality - Song selection and metadata display - Responsive design

3. `components/Controls.js` - Audio Playback Controls

Purpose: Standard music player interface

Features: - Play/pause/stop controls - Progress bar with seeking - Volume control - Time display

4. `components/focus/FocusControls.js` - Focus Processing Interface

Purpose: Configure and manage focus processing

Features: - Intensity slider (0-100%) - BPM targeting options - Processing presets - Job status monitoring

5. `services/api.js` - Backend Communication

Purpose: Centralized API client for backend communication

Features: - RESTful API calls - Error handling - Response formatting - Base URL configuration

---

API Documentation

Music Library Endpoints

GET /songs Get songs with optional pagination and search

Parameters: - `limit` (int): Maximum songs to return (1-1000, default 100) - `offset` (int): Pagination offset (default 0) - `search` (string): Search in title/artist/album

Response:

```
[
  {
    "id": 1,
    "title": "Song Title",
    "artist": "Artist Name",
    "album": "Album Name",
    "duration": 180.5,
    "filepath": "artist/album/song.mp3"
  }
]
```

GET /songs/{song_id} Get detailed song information

Response:

```
{
  "id": 1,
  "title": "Song Title",
  "artist": "Artist Name",
  "album": "Album Name",
  "duration": 180.5,
  "bpm": 120.0,
  "file_size": 4567890,
  "created_at": "2025-01-15T10:30:00Z"
}
```

GET /audio/{song_id} Stream audio file (supports range requests for seeking)

Response: Audio stream (MP3/M4A/etc.)

Focus Processing Endpoints

POST /focus/process Submit focus processing job

Request Body:

```
{
  "song_id": 1,
  "focus_intensity": 75,
  "target_bpm": 80
}
```

Response:

```
{
  "job_id": "uuid-string",
  "estimated_completion_time": 45.0,
```

```
  "message": "Processing job submitted successfully"
}
```

GET /focus/status/{job_id} Get job status and progress

Response:

```
{
  "job_id": "uuid-string",
  "status": "running",
  "progress_percent": 65,
  "current_step": "Applying focus effects",
  "estimated_completion": "2025-01-15T10:35:00Z",
  "result": null
}
```

DELETE /focus/cancel/{job_id} Cancel running or pending job

Response:

```
{
  "success": true,
  "message": "Job cancelled successfully"
}
```

Audio Processing Pipeline

1. Input Validation

```
# Validate file exists and is processable
is_valid, error = validate_processing_request(file_path, intensity, target_bpm)
```

Checks: - File existence and accessibility - File size limits (max 100MB) - Duration limits (10s - 10min) - Parameter validation (intensity 0-100, BPM 30-200)

2. Audio Analysis

```
# Extract comprehensive audio features
analysis = AudioAnalyzer.analyze_audio_file(file_path)
```

Extracted Features: - Duration and sample rate - Tempo (BPM) detection - Spectral centroid (brightness) - Dynamic range analysis - Key signature estimation - Processing complexity assessment

3. Spleeter Separation

```
# AI-powered source separation
separator = Separator('spleeter:2stems')
stems = separator.separate(audio_data)
```

Output Stems: - **Vocals**: Singing and lead instruments - **Accompaniment**: Background music, drums, bass

4. Focus Transformation

```
# Apply focus-enhancing effects
processed_audio = apply_focus_effects(stems, intensity)
```

Effects Applied (based on intensity): - **Low-pass filtering**: Reduces high-frequency distractions - **Vocal reduction**: Minimizes lyrical distractions - **Stereo widening**: Enhances spatial perception - **Dynamic compression**: Smooths volume variations - **EQ adjustments**: Optimizes frequency balance

5. Tempo Adjustment (Optional)

```
# Adjust BPM if target specified
if target_bpm > 0:
    processed_audio = adjust_tempo(processed_audio, original_bpm, target_bpm)
```

6. Export and Storage

```
# High-quality MP3 export
output_path = numpy_to_mp3(processed_audio, sample_rate, output_file)
```

Export Settings: - Format: MP3 - Bitrate: 320kbps (high quality) - Sample rate: Preserved from original

---

Database Schema

Songs Table

```
CREATE TABLE songs (
    id SERIAL PRIMARY KEY,
    filename VARCHAR NOT NULL,
    filepath VARCHAR UNIQUE NOT NULL,
    file_size INTEGER,
    file_format VARCHAR,

    title VARCHAR,
    artist VARCHAR,
    album VARCHAR,
    genre VARCHAR,
```

```
    year INTEGER,
    track_number INTEGER,

    duration FLOAT,
    bitrate INTEGER,
    sample_rate INTEGER,
    bpm FLOAT,

    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    is_available BOOLEAN DEFAULT TRUE
);

CREATE INDEX idx_songs_title ON songs(title);
CREATE INDEX idx_songs_artist ON songs(artist);
CREATE INDEX idx_songs_filepath ON songs(filepath);
```

Processed Versions Table

```
CREATE TABLE processed_versions (
    id SERIAL PRIMARY KEY,
    song_id INTEGER,

    focus_intensity INTEGER NOT NULL,
    target_bpm FLOAT,

    processed_filepath VARCHAR NOT NULL,
    processed_duration FLOAT,
    actual_bpm FLOAT,

    created_at TIMESTAMP DEFAULT NOW(),
    file_size INTEGER
);

CREATE INDEX idx_processed_song_id ON processed_versions(song_id);
```

Development Workflow

Adding New Features

1. **Backend API Changes**:

```
# Add new endpoint to app.py or focus_api.py
# Add database models if needed
# Write unit tests
# Update API documentation
```

2. **Frontend Integration**:

```
# Add API calls to services/api.js
# Create/update React components
# Add styling with Tailwind
# Test user interactions
```

3. **Audio Processing**:

```
# Extend FocusProcessor class
# Add new effects to apply_focus_effects()
# Update audio analysis if needed
# Test with various audio formats
```

Testing Strategy

1. **Unit Tests** (Backend):

```
# Test audio utilities
pytest audio_utils.py

# Test processing pipeline
pytest focus_processor.py
```

2. **Integration Tests**:

```
# Test full API endpoints
pytest test_api.py

# Test database operations
pytest test_database.py
```

3. **Frontend Tests**:

```
# Component tests
npm test

# E2E tests (future)
npm run e2e
```

Performance Optimization

1. **Audio Processing**:
   - Cache Spleeter model initialization
   - Parallel processing for multiple jobs
   - Optimize audio buffer sizes
   - Use faster audio formats for intermediates
2. **Database**:
   - Add indexes for common queries
   - Implement connection pooling

- Cache frequent metadata queries

3. **Frontend**:
   - Implement virtual scrolling for large libraries
   - Add audio preloading for smooth playback
   - Optimize bundle size with code splitting

Troubleshooting

Common Issues

1. Spleeter Installation Problems

```
# Issue: TensorFlow compatibility
# Solution: Use specific versions
pip install tensorflow==2.13.0
pip install spleeter==2.4.0
```

2. Audio File Not Found

```
# Check file paths are relative to music folder
# Verify MUSIC_FOLDER configuration
# Check file permissions
```

3. Database Connection Errors

```
# Verify PostgreSQL is running
systemctl status postgresql

# Check connection string
psql "postgresql://user:password@localhost:5432/focus_music"
```

4. Processing Fails Silently

```
# Check logs for detailed errors
tail -f backend.log

# Verify audio file format support
# Check available disk space
```

Performance Issues

1. Slow Processing

- **Cause**: Limited CPU resources
- **Solution**: Reduce concurrent jobs, optimize audio settings

2. Memory Usage

- **Cause**: Large audio files, Spleeter model
- **Solution**: Process in chunks, limit file sizes

3. Frontend Lag

- **Cause**: Large music libraries
- **Solution**: Implement pagination, virtual scrolling

Future Enhancements

Short Term (Next Version)

1. **Enhanced Focus Modes**:
   - Binaural beats integration
   - Nature sounds mixing
   - Custom preset creation
2. **User Experience**:
   - Processing progress visualization
   - Batch processing for multiple songs
   - Playlist creation and management
3. **Audio Quality**:
   - Higher quality separation models
   - Lossless processing options
   - Advanced audio analysis

Medium Term

1. **Cloud Integration**:
   - Cloud storage for processed files
   - Distributed processing
   - User accounts and sync
2. **Mobile App**:
   - React Native companion app
   - Offline playback capability
   - Background processing
3. **AI Improvements**:
   - Custom model training
   - Personalized focus optimization
   - Real-time adaptation

Long Term

1. **Enterprise Features**:
   - Multi-user support

    - Admin dashboard
    - Usage analytics
2. **Advanced Processing**:
    - Real-time processing
    - Live streaming integration
    - VR/AR audio experiences
3. **Platform Expansion**:
    - Desktop applications
    - Browser extensions
    - Smart speaker integration

Contributing

Code Style

- **Python**: Follow PEP 8, use type hints
- **JavaScript**: Use ESLint/Prettier configuration
- **Comments**: Explain why, not what
- **Documentation**: Update docs with code changes

Submitting Changes

1. Fork repository
2. Create feature branch
3. Write tests for new functionality
4. Update documentation
5. Submit pull request with detailed description

Development Environment

```
# Install development dependencies
pip install -r requirements-dev.txt
npm install --include=dev

# Run linting
flake8 backend/
eslint frontend/src/

# Run tests
pytest backend/
npm test frontend/
```

License & Credits

Dependencies

- **Spleeter**: AI source separation (MIT License)
- **librosa**: Audio analysis (ISC License)
- **FastAPI**: Web framework (MIT License)
- **React**: Frontend framework (MIT License)

Credits

- Audio processing algorithms based on research in music information retrieval
- Focus enhancement techniques derived from cognitive psychology studies
- UI/UX inspired by modern music streaming applications