

DrumKitz Audio Processing Website Documentation

Project Overview

DrumKitz is a web-based application designed for music producers and enthusiasts to upload audio loops, preview the waveform, extract individual drum samples, and download these samples in a structured format. The application is built using React, TypeScript, and WaveSurfer.js for audio visualization and processing.

This documentation serves as a detailed guide for understanding the architecture, features, and functionalities of the DrumKitz project, both for current development and future extensions.

Purpose and Features

Primary Goal

To provide users with an easy-to-use platform for processing drum loops or tracks. Users can:

1. Upload audio files (.mp3 or .wav) of up to 10 minutes.
2. Preview the waveform and interact with a 20-second loop section.
3. Detect and extract individual drum sounds using peak detection.
4. Download all extracted drum sounds as a .zip file.

Version Breakdown

- **Version 1:**
 - Basic functionality to upload audio, preview waveform, interact with 20-second loops, and download drum sounds based on peak detection.
 - **Version 2:**
 - Integration with Lalal.ai API for advanced drum separation.
 - Support for full-track uploads and enhanced processing capabilities.
-

Project Structure

```
DrumKitz/
├── public/
│   └── index.html          # Main HTML file for rendering
├── src/
│   ├── assets/            # Static assets (icons, images, etc.)
│   ├── components/        # React components
│   │   ├── AudioUpload.tsx # File upload and drum track selection
│   │   ├── AudioPreview.tsx # Waveform rendering and playback controls
│   │   └── ProcessingOptions.tsx # Audio processing and download functionality
│   ├── utils/             # Helper functions and logic
│   │   └── audioProcessing.ts # Core audio analysis and segmentation
│   ├── styles/            # CSS files
│   │   └── App.css         # Main stylesheet
│   ├── App.tsx            # Main application component
│   ├── main.tsx           # Entry point for React app
│   ├── index.css          # Global styles
│   └── vite-env.d.ts       # Vite TypeScript declarations
├── package.json           # Dependencies and scripts
├── tsconfig.json          # TypeScript configuration
├── vite.config.ts         # Vite configuration
└── README.md              # Project readme (this document)
```

Core Functionalities

1. File Upload and Drum Track Selection

- Users can upload audio files (.mp3 or .wav).
- A checkbox allows users to specify if the uploaded file is a drum track, which influences the processing logic.

2. Waveform Preview

- **Library:** WaveSurfer.js for waveform rendering.
- **Features:**
 - Display the waveform of the uploaded audio.
 - Highlight a movable 20-second loop.
 - Playback controls: Play, Pause, Stop.
 - Zoom in/out and horizontal scroll for waveform navigation.
 - Red cursor line for playback position, visible during play, pause, and stop.

3. Audio Processing

- **Peak Detection:**
 - Peaks are identified as transitions from quiet sections to loud sections.
 - Each peak marks the start of a drum sound, and the sound duration is determined by analyzing subsequent quiet sections.
- **BPM-Based Gridlines:**
 - Automatically detect the BPM of the loop.
 - Generate visual gridlines to match beat positions, aiding in drum sample identification.
- **Sound Segmentation:**
 - Extract short audio segments (e.g., kicks, snares, hi-hats) around each detected peak.
 - Use libraries like `wavefile` to convert each segment into a .wav file.

4. Download Functionality

- All extracted sounds are named sequentially (e.g., `drum_hit_1.wav`, `drum_hit_2.wav`).
 - The files are bundled into a .zip file for download.
-

Implementation Details

Key Algorithms

1. **Peak Detection:**
 - Use `OfflineAudioContext` to process audio data off the main thread.
 - Analyze time-domain data with an `AnalyserNode` to identify amplitude spikes.
 - Peaks are detected based on a threshold, and time intervals are used to isolate individual drum sounds.
2. **BPM Detection:**
 - Libraries like `bpm-detective` analyze audio patterns to estimate beats per minute.
 - Grids are generated at intervals based on BPM to align with rhythmic structures.
3. **Waveform Interaction:**
 - Use WaveSurfer.js regions to highlight the 20-second loop and detected drum hits.
 - Allow users to move and adjust the loop region within the audio.
4. **Download as .zip:**
 - Use `JSZip` to bundle extracted .wav files into a single .zip file for download.

Playback Controls

- **Play:** Starts playback from the current cursor position.
 - **Pause:** Stops playback at the current position, allowing it to resume from the same point.
 - **Stop:** Stops playback and resets the cursor to the start of the waveform.
-

Future Extensions

1. Lalal.ai API Integration

- Use the Lalal.ai API to separate drum tracks from full audio samples.
- Process the separated drum track to extract individual sounds.

2. Enhanced User Interface

- Add an interactive slider for adjusting peak detection sensitivity.
- Display detailed metrics for each detected sound (e.g., duration, start time).
- Enable users to rename drum samples before download.

3. Mobile Responsiveness

- Optimize the layout for smaller screens, ensuring usability on mobile devices.

4. Collaboration Features

- Allow users to save and share their processed samples via cloud storage.
-

How It Works

1. **User Uploads File:**
 - Drag-and-drop or file picker for .mp3/.wav files.
 - Specify if the file is a drum track.
2. **Waveform Preview:**
 - Display the waveform and playback controls.
 - Highlight a 20-second loop with adjustable start and end positions.
3. **Processing:**
 - Detect BPM and peaks in the audio.
 - Extract individual drum sounds based on peak analysis.
4. **Download:**
 - Package extracted sounds as .wav files in a .zip archive.
 - Provide the zip file for download.

Development Tools and Libraries

- **React:** Framework for building the user interface.
 - **TypeScript:** Ensures type safety and scalability.
 - **WaveSurfer.js:** Audio visualization and interaction.
 - **JSZip:** Packaging files into .zip archives.
 - **WaveFile:** Creating .wav files from audio data.
-

Instructions for Setting Up Locally

Clone Repository:

`git clone <repository-url>`

1. `cd DrumKitz`
2. **Install Dependencies:**
`npm install`
3. **Start Development Server:**
`npm run dev`
 - Access the app at <http://localhost:3000>.
4. **Build for Production:**
`npm run build`
 - Production files will be in the `dist/` folder.