Building a Professional Audio Editor with Next.js 15, TypeScript & Tailwind CSS

Complete Implementation Guide

Table of Contents

1. Introduction

This comprehensive guide will walk you through building a professional browser-based audio editing application using Next.js 15.5.6, TypeScript, and Tailwind CSS. By the end, you'll have a fully functional audio editor capable of waveform visualization, audio manipulation, effects processing, and multi-format export.

Key Features

- **Audio file upload** with drag-and-drop support
- **Waveform visualization** with zoom and navigation
- **Playback controls** (play, pause, stop, seek)
- **Audio trimming** and region selection
- **Audio effects** (volume, fade, reverb, EQ)
- **Microphone recording** capabilities
- **Multi-format export** (MP3, WAV)
- **Keyboard shortcuts** for efficient editing

2. Technology Stack Overview

Core Framework

- **Next.js 15.5.6**: React framework with App Router
- **React 19**: Latest React with Server Components
- **TypeScript 5.x**: Type-safe development
- **Tailwind CSS**: Utility-first styling

Audio Libraries

Library	Version	Size	Primary Use
wavesurfer.js	7.x	~30KB	Waveform visualization & navigation
Tone.js	latest	~100KB	Audio synthesis & effects
Howler.js	2.x	7KB	Simple, reliable playback
lamejs	latest	~200KB	Client-side MP3 encoding
Web Audio API	Native	Built-in	Core audio processing

UI Components & Utilities

- **@radix-ui/react-slider**: Accessible slider components
- **lucide-react**: Icon library
- **zustand**: Lightweight state management
- **react-dropzone**: File upload handling
- **file-saver**: Client-side file downloads

3. Project Setup & Installation

Step 1: Create Next.js Application

```
npx create-next-app@15.5.6 audio-editor \\
  --typescript \\
  --tailwind \\
  --app \\
  --no-src-dir
```

Step 2: Navigate to Project

```
cd audio-editor
```

Step 3: Install Audio Libraries

```
npm install wavesurfer.js@7 tone@latest howler@2 lamejs
```

Step 4: Install UI Components

```
npm install @radix-ui/react-slider \\
            @radix-ui/react-dialog \\
            lucide-react \\
            clsx
```

Step 5: Install Utilities

```
npm install zustand react-dropzone file-saver
npm install -D @types/file-saver
```

Step 6: Run Development Server

```
npm run dev
```

Visit `http://localhost:3000` to see your application.

4. Core Audio Libraries

4.1 wavesurfer.js - Waveform Visualization

Best for: Displaying interactive audio waveforms with region selection and zoom capabilities.

Key Features:

- HTML5 Audio and Web Audio support
- Responsive and customizable waveforms
- Plugin system for extended functionality
- TypeScript support out of the box

Basic Usage:

```
import WaveSurfer from 'wavesurfer.js';
```

```
const wavesurfer = WaveSurfer.create({
  container: '#waveform',
  waveColor: '#4F4A85',
  progressColor: '#383351',
  url: '/audio.mp3',
  height: 128,
  normalize: true,
  barWidth: 2,
  barGap: 1,
});

// Play/pause
wavesurfer.playPause();

// Seek to position (0-1)
wavesurfer.seekTo(0.5);

// Add region
const region = wavesurfer.addRegion({
  start: 5,
  end: 10,
  color: 'rgba(0, 255, 0, 0.1)',
});
```

4.2 Tone.js - Audio Synthesis & Effects

Best for: Creating audio effects, synthesizers, and complex signal processing.

Key Features:

- Web Audio framework for interactive music
- Built-in effects (reverb, delay, distortion, EQ)
- Transport system for scheduling
- Intuitive musical timing

Basic Usage:

```
import * as Tone from 'tone';

// Create a player
const player = new Tone.Player('/audio.mp3').toDestination();

// Add reverb effect
const reverb = new Tone.Reverb({
  decay: 4,
  preDelay: 0.01,
}).toDestination();

player.connect(reverb);

// Add EQ
const eq = new Tone.EQ3({
  low: -10,
```

```
  mid: 0,
  high: 5,
}).connect(reverb);

player.disconnect();
player.connect(eq);

// Play
await Tone.start();
player.start();
```

4.3 Howler.js - Simple Playback

Best for: Straightforward audio playback with minimal overhead.

Key Features:

- Automatic codec detection and fallback
- Audio sprite support
- Spatial audio (3D positioning)
- Zero dependencies

Basic Usage:

```
import { Howl } from 'howler';

const sound = new Howl({
  src: ['/audio.mp3'],
  volume: 0.5,
  onend: () =&gt; console.log('Finished!'),
});

// Play
sound.play();

// Fade out
sound.fade(1, 0, 1000);

// Spatial audio
sound.pos(10, 0, 0);
```

4.4 lamejs - MP3 Encoding

Best for: Converting audio buffers to MP3 format for export.

Basic Usage:

```
import lamejs from 'lamejs';

function encodeToMP3(audioBuffer: AudioBuffer): Blob {
  const mp3encoder = new lamejs.Mp3Encoder(
```

```
    audioBuffer.numberOfChannels,
    audioBuffer.sampleRate,
    128 // kbps
  );

  const samples = audioBuffer.getChannelData(0);
  const sampleBlockSize = 1152;
  const mp3Data: Int8Array[] = [];

  for (let i = 0; i &lt; samples.length; i += sampleBlockSize) {
    const sampleChunk = samples.subarray(i, i + sampleBlockSize);
    const mp3buf = mp3encoder.encodeBuffer(
      convertFloat32ToInt16(sampleChunk)
    );
    if (mp3buf.length &gt; 0) {
      mp3Data.push(mp3buf);
    }
  }

  const mp3buf = mp3encoder.flush();
  if (mp3buf.length &gt; 0) {
    mp3Data.push(mp3buf);
  }

  return new Blob(mp3Data, { type: 'audio/mp3' });
}

function convertFloat32ToInt16(buffer: Float32Array): Int16Array {
  const l = buffer.length;
  const buf = new Int16Array(l);
  for (let i = 0; i &lt; l; i++) {
    buf[i] = Math.min(1, buffer[i]) * 0x7FFF;
  }
  return buf;
}
```

4.5 Web Audio API - Native Browser API

Best for: Low-level audio processing, custom effects, and real-time analysis.

Core Concepts:

```
// Create audio context
const audioContext = new AudioContext();

// Load audio file
const response = await fetch('/audio.mp3');
const arrayBuffer = await response.arrayBuffer();
const audioBuffer = await audioContext.decodeAudioData(arrayBuffer);

// Create source
const source = audioContext.createBufferSource();
source.buffer = audioBuffer;
```

```
// Create gain node (volume control)
const gainNode = audioContext.createGain();
gainNode.gain.value = 0.5;

// Create filter
const filter = audioContext.createBiquadFilter();
filter.type = 'lowpass';
filter.frequency.value = 1000;

// Connect nodes
source.connect(filter);
filter.connect(gainNode);
gainNode.connect(audioContext.destination);

// Play
source.start(0);
```

5. Project Architecture

5.1 Folder Structure

```
audio-editor/
├── app/
│   ├── layout.tsx                # Root layout
│   ├── page.tsx                  # Home page
│   ├── editor/
│   │   └── page.tsx              # Editor page
│   └── globals.css               # Tailwind directives
├── components/
│   ├── audio-editor/
│   │   ├── AudioPlayer.tsx       # Main player component
│   │   ├── Waveform.tsx          # Waveform display
│   │   ├── Controls.tsx          # Playback controls
│   │   ├── Timeline.tsx          # Timeline with markers
│   │   ├── EffectsPanel.tsx      # Effects controls
│   │   ├── ExportPanel.tsx       # Export functionality
│   │   └── FileUpload.tsx        # File upload component
│   └── ui/
│       ├── Button.tsx            # Reusable button
│       ├── Slider.tsx            # Range slider
│       └── Card.tsx              # Card container
├── hooks/
│   ├── useAudioContext.ts        # Audio context management
│   ├── useWaveform.ts            # Wavesurfer integration
│   ├── useAudioRecorder.ts       # Recording functionality
│   └── useAudioEffects.ts        # Effects processing
├── lib/
│   ├── audio-processor.ts        # Audio processing utilities
│   ├── audio-encoder.ts          # MP3/WAV encoding
│   └── audio-effects.ts          # Effect implementations
├── store/
│   └── audio-store.ts            # Zustand state management
├── types/
```

```
│   └── audio.ts                  # TypeScript interfaces
└── utils/
    ├── audio-utils.ts            # Helper functions
    └── format-time.ts            # Time formatting
```

5.2 Component Hierarchy

```
App
├── Layout
│   ├── Header
│   └── Footer
└── EditorPage
    ├── FileUpload
    ├── AudioPlayer
    │   ├── Waveform
    │   ├── Controls
    │   │   ├── PlayButton
    │   │   ├── PauseButton
    │   │   ├── StopButton
    │   │   └── SeekBar
    │   └── Timeline
    ├── EffectsPanel
    │   ├── VolumeControl
    │   ├── ReverbControl
    │   ├── EQControl
    │   └── FadeControl
    └── ExportPanel
        ├── FormatSelector
        └── ExportButton
```

6. Component Implementation

6.1 Main Layout (app/layout.tsx)

```
import type { Metadata } from 'next';
import { Inter } from 'next/font/google';
import './globals.css';

const inter = Inter({ subsets: ['latin'] });

export const metadata: Metadata = {
  title: 'Audio Editor Pro',
  description: 'Professional browser-based audio editing',
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
```

```
    &lt;html lang="en"&gt;
      &lt;body className={inter.className}&gt;
        &lt;main className="min-h-screen bg-gradient-to-br from-slate-900 to-slate-800"&g
          {children}
        &lt;/main&gt;
      &lt;/body&gt;
    &lt;/html&gt;
  );
}
```

6.2 File Upload Component

```
'use client';

import { useCallback } from 'react';
import { useDropzone } from 'react-dropzone';
import { Upload } from 'lucide-react';

interface FileUploadProps {
  onFileSelect: (file: File) =&gt; void;
}

export function FileUpload({ onFileSelect }: FileUploadProps) {
  const onDrop = useCallback((acceptedFiles: File[]) =&gt; {
    if (acceptedFiles.length &gt; 0) {
      onFileSelect(acceptedFiles[^0]);
    }
  }, [onFileSelect]);

  const { getRootProps, getInputProps, isDragActive } = useDropzone({
    onDrop,
    accept: {
      'audio/*': ['.mp3', '.wav', '.ogg', '.m4a', '.aac'],
    },
    maxFiles: 1,
  });

  return (
    <div>
      &lt;input {...getInputProps()} /&gt;
      &lt;Upload className="mx-auto h-12 w-12 text-slate-400 mb-4" /&gt;
      <p>
        {isDragActive ? 'Drop audio file here' : 'Drag &amp; drop audio file'}
      </p>
      <p>
        or click to browse (MP3, WAV, OGG, M4A, AAC)
      </p>
    </div>
  );
}
```

6.3 Waveform Component

```
'use client';

import { useEffect, useRef } from 'react';
import WaveSurfer from 'wavesurfer.js';
import RegionsPlugin from 'wavesurfer.js/dist/plugins/regions';

interface WaveformProps {
  audioUrl: string;
  onReady?: (wavesurfer: WaveSurfer) =&gt; void;
}

export function Waveform({ audioUrl, onReady }: WaveformProps) {
  const containerRef = useRef&lt;HTMLDivElement&gt;(null);
  const wavesurferRef = useRef&lt;WaveSurfer | null&gt;(null);

  useEffect(() =&gt; {
    if (!containerRef.current) return;

    // Create wavesurfer instance
    const wavesurfer = WaveSurfer.create({
      container: containerRef.current,
      waveColor: '#4F4A85',
      progressColor: '#383351',
      cursorColor: '#fff',
      barWidth: 2,
      barGap: 1,
      height: 128,
      normalize: true,
      plugins: [
        RegionsPlugin.create(),
      ],
    });

    // Load audio
    wavesurfer.load(audioUrl);

    // Ready callback
    wavesurfer.on('ready', () =&gt; {
      onReady?.(wavesurfer);
    });

    wavesurferRef.current = wavesurfer;

    return () =&gt; {
      wavesurfer.destroy();
    };
  }, [audioUrl, onReady]);

  return (
    <div>
      <div>
    </div>
```

```
  );
}
```

6.4 Playback Controls

```
'use client';

import { Play, Pause, Square, SkipBack, SkipForward } from 'lucide-react';
import { Button } from '@/components/ui/Button';

interface ControlsProps {
  isPlaying: boolean;
  onPlay: () =&gt; void;
  onPause: () =&gt; void;
  onStop: () =&gt; void;
  onSkipBackward: () =&gt; void;
  onSkipForward: () =&gt; void;
}

export function Controls({
  isPlaying,
  onPlay,
  onPause,
  onStop,
  onSkipBackward,
  onSkipForward,
}: ControlsProps) {
  return (
    <div>
      &lt;Button
        variant="ghost"
        size="icon"
        onClick={onSkipBackward}
        className="text-slate-300 hover:text-white"
      &gt;
        &lt;SkipBack className="h-5 w-5" /&gt;
      &lt;/Button&gt;

      {isPlaying ? (
        &lt;Button
          variant="primary"
          size="icon"
          onClick={onPause}
          className="h-12 w-12"
        &gt;
          &lt;Pause className="h-6 w-6" /&gt;
        &lt;/Button&gt;
      ) : (
        &lt;Button
          variant="primary"
          size="icon"
          onClick={onPlay}
          className="h-12 w-12"
        &gt;
          &lt;Play className="h-6 w-6 ml-1" /&gt;
```

```
        &lt;/Button&gt;
      )}

      &lt;Button
        variant="ghost"
        size="icon"
        onClick={onStop}
        className="text-slate-300 hover:text-white"
      &gt;
        &lt;Square className="h-5 w-5" /&gt;
      &lt;/Button&gt;

      &lt;Button
        variant="ghost"
        size="icon"
        onClick={onSkipForward}
        className="text-slate-300 hover:text-white"
      &gt;
        &lt;SkipForward className="h-5 w-5" /&gt;
      &lt;/Button&gt;
    </div>
  );
}
```

7. Custom Hooks for Audio Processing

7.1 useAudioContext Hook

```
import { useEffect, useState, useRef } from 'react';

export function useAudioContext() {
  const [audioContext, setAudioContext] = useState&lt;AudioContext | null&gt;(null);
  const contextRef = useRef&lt;AudioContext | null&gt;(null);

  useEffect(() =&gt; {
    // Create audio context
    const AudioContextClass = window.AudioContext ||
                              (window as any).webkitAudioContext;

    if (!AudioContextClass) {
      console.error('Web Audio API not supported');
      return;
    }

    const ctx = new AudioContextClass();
    contextRef.current = ctx;
    setAudioContext(ctx);

    // Resume context on user interaction (browser requirement)
    const resumeContext = async () =&gt; {
      if (ctx.state === 'suspended') {
        await ctx.resume();
      }
```

```
    };

    document.addEventListener('click', resumeContext, { once: true });

    return () =&gt; {
      ctx.close();
    };
  }, []);

  return audioContext;
}
```

7.2 useWaveform Hook

```
import { useEffect, useRef, useState } from 'react';
import WaveSurfer from 'wavesurfer.js';
import RegionsPlugin, { Region } from 'wavesurfer.js/dist/plugins/regions';

interface UseWaveformOptions {
  onReady?: () =&gt; void;
  onPlay?: () =&gt; void;
  onPause?: () =&gt; void;
  onFinish?: () =&gt; void;
}

export function useWaveform(
  containerRef: React.RefObject&lt;HTMLDivElement&gt;,
  audioUrl: string | null,
  options?: UseWaveformOptions
) {
  const wavesurferRef = useRef&lt;WaveSurfer | null&gt;(null);
  const regionsPluginRef = useRef&lt;RegionsPlugin | null&gt;(null);
  const [isReady, setIsReady] = useState(false);
  const [isPlaying, setIsPlaying] = useState(false);
  const [duration, setDuration] = useState(0);
  const [currentTime, setCurrentTime] = useState(0);

  useEffect(() =&gt; {
    if (!containerRef.current || !audioUrl) return;

    // Create regions plugin
    const regions = RegionsPlugin.create();
    regionsPluginRef.current = regions;

    // Create wavesurfer
    const wavesurfer = WaveSurfer.create({
      container: containerRef.current,
      waveColor: '#4F4A85',
      progressColor: '#383351',
      cursorColor: '#ffffff',
      barWidth: 2,
      barGap: 1,
      height: 128,
      normalize: true,
      plugins: [regions],
```

```
    });

    wavesurferRef.current = wavesurfer;

    // Event listeners
    wavesurfer.on('ready', () =&gt; {
      setIsReady(true);
      setDuration(wavesurfer.getDuration());
      options?.onReady?.();
    });

    wavesurfer.on('play', () =&gt; {
      setIsPlaying(true);
      options?.onPlay?.();
    });

    wavesurfer.on('pause', () =&gt; {
      setIsPlaying(false);
      options?.onPause?.();
    });

    wavesurfer.on('finish', () =&gt; {
      setIsPlaying(false);
      options?.onFinish?.();
    });

    wavesurfer.on('timeupdate', (time) =&gt; {
      setCurrentTime(time);
    });

    // Load audio
    wavesurfer.load(audioUrl);

    return () =&gt; {
      wavesurfer.destroy();
    };
  }, [containerRef, audioUrl, options]);

  const play = () =&gt; wavesurferRef.current?.play();
  const pause = () =&gt; wavesurferRef.current?.pause();
  const stop = () =&gt; {
    wavesurferRef.current?.stop();
    setCurrentTime(0);
  };
  const seekTo = (progress: number) =&gt; wavesurferRef.current?.seekTo(progress);

  const addRegion = (start: number, end: number, color?: string) =&gt; {
    return regionsPluginRef.current?.addRegion({
      start,
      end,
      color: color || 'rgba(0, 123, 255, 0.1)',
      drag: true,
      resize: true,
    });
  };
```

```
  const clearRegions = () =&gt; regionsPluginRef.current?.clearRegions();

  return {
    wavesurfer: wavesurferRef.current,
    isReady,
    isPlaying,
    duration,
    currentTime,
    play,
    pause,
    stop,
    seekTo,
    addRegion,
    clearRegions,
  };
}
```

7.3 useAudioRecorder Hook

```
import { useState, useRef, useCallback } from 'react';

export function useAudioRecorder() {
  const [isRecording, setIsRecording] = useState(false);
  const [recordedBlob, setRecordedBlob] = useState&lt;Blob | null&gt;(null);
  const mediaRecorderRef = useRef&lt;MediaRecorder | null&gt;(null);
  const chunksRef = useRef&lt;Blob[]&gt;([]);

  const startRecording = useCallback(async () =&gt; {
    try {
      const stream = await navigator.mediaDevices.getUserMedia({
        audio: true
      });

      const mediaRecorder = new MediaRecorder(stream);
      mediaRecorderRef.current = mediaRecorder;
      chunksRef.current = [];

      mediaRecorder.ondataavailable = (event) =&gt; {
        if (event.data.size &gt; 0) {
          chunksRef.current.push(event.data);
        }
      };

      mediaRecorder.onstop = () =&gt; {
        const blob = new Blob(chunksRef.current, { type: 'audio/webm' });
        setRecordedBlob(blob);
        stream.getTracks().forEach(track =&gt; track.stop());
      };

      mediaRecorder.start();
      setIsRecording(true);
    } catch (error) {
      console.error('Error starting recording:', error);
    }
  }, []);
```

```
  const stopRecording = useCallback(() =&gt; {
    if (mediaRecorderRef.current &amp;&amp; isRecording) {
      mediaRecorderRef.current.stop();
      setIsRecording(false);
    }
  }, [isRecording]);

  const clearRecording = useCallback(() =&gt; {
    setRecordedBlob(null);
    chunksRef.current = [];
  }, []);

  return {
    isRecording,
    recordedBlob,
    startRecording,
    stopRecording,
    clearRecording,
  };
}
```

8. Audio Effects & Processing

8.1 Effects Implementation (lib/audio-effects.ts)

```
import * as Tone from 'tone';

export class AudioEffects {
  private player: Tone.Player | null = null;
  private reverb: Tone.Reverb | null = null;
  private eq: Tone.EQ3 | null = null;
  private gain: Tone.Gain | null = null;

  async initialize(audioUrl: string) {
    // Create player
    this.player = new Tone.Player(audioUrl);

    // Create effects
    this.reverb = new Tone.Reverb({
      decay: 2,
      preDelay: 0.01,
    });

    this.eq = new Tone.EQ3({
      low: 0,
      mid: 0,
      high: 0,
    });

    this.gain = new Tone.Gain(1);

    // Connect chain
```

```
    this.player.chain(this.eq, this.reverb, this.gain, Tone.Destination);

    await Tone.loaded();
  }

  async play() {
    await Tone.start();
    this.player?.start();
  }

  pause() {
    this.player?.stop();
  }

  setVolume(volume: number) {
    if (this.gain) {
      this.gain.gain.value = volume;
    }
  }

  setReverb(decay: number) {
    if (this.reverb) {
      this.reverb.decay = decay;
    }
  }

  setEQ(low: number, mid: number, high: number) {
    if (this.eq) {
      this.eq.low.value = low;
      this.eq.mid.value = mid;
      this.eq.high.value = high;
    }
  }

  applyFadeIn(duration: number) {
    if (this.gain) {
      this.gain.gain.linearRampTo(1, duration);
    }
  }

  applyFadeOut(duration: number) {
    if (this.gain) {
      this.gain.gain.linearRampTo(0, duration);
    }
  }

  dispose() {
    this.player?.dispose();
    this.reverb?.dispose();
    this.eq?.dispose();
    this.gain?.dispose();
  }
}
```

8.2 Audio Processor Utilities (lib/audio-processor.ts)

```
export class AudioProcessor {
  private audioContext: AudioContext;

  constructor(audioContext: AudioContext) {
    this.audioContext = audioContext;
  }

  async loadAudioFile(file: File): Promise&lt;AudioBuffer&gt; {
    const arrayBuffer = await file.arrayBuffer();
    return await this.audioContext.decodeAudioData(arrayBuffer);
  }

  trimAudio(
    audioBuffer: AudioBuffer,
    startTime: number,
    endTime: number
  ): AudioBuffer {
    const sampleRate = audioBuffer.sampleRate;
    const startSample = Math.floor(startTime * sampleRate);
    const endSample = Math.floor(endTime * sampleRate);
    const length = endSample - startSample;

    const trimmedBuffer = this.audioContext.createBuffer(
      audioBuffer.numberOfChannels,
      length,
      sampleRate
    );

    for (let channel = 0; channel &lt; audioBuffer.numberOfChannels; channel++) {
      const sourceData = audioBuffer.getChannelData(channel);
      const targetData = trimmedBuffer.getChannelData(channel);

      for (let i = 0; i &lt; length; i++) {
        targetData[i] = sourceData[startSample + i];
      }
    }

    return trimmedBuffer;
  }

  mergeAudioBuffers(buffers: AudioBuffer[]): AudioBuffer {
    if (buffers.length === 0) {
      throw new Error('No buffers to merge');
    }

    const sampleRate = buffers[^0].sampleRate;
    const numberOfChannels = buffers[^0].numberOfChannels;

    // Calculate total length
    const totalLength = buffers.reduce((sum, buffer) =&gt;
      sum + buffer.length, 0
    );

    const mergedBuffer = this.audioContext.createBuffer(
```

```
    numberOfChannels,
    totalLength,
    sampleRate
  );

  let offset = 0;
  buffers.forEach(buffer =&gt; {
    for (let channel = 0; channel &lt; numberOfChannels; channel++) {
      const sourceData = buffer.getChannelData(channel);
      const targetData = mergedBuffer.getChannelData(channel);
      targetData.set(sourceData, offset);
    }
    offset += buffer.length;
  });

  return mergedBuffer;
}

normalizeAudio(audioBuffer: AudioBuffer): AudioBuffer {
  const normalized = this.audioContext.createBuffer(
    audioBuffer.numberOfChannels,
    audioBuffer.length,
    audioBuffer.sampleRate
  );

  for (let channel = 0; channel &lt; audioBuffer.numberOfChannels; channel++) {
    const sourceData = audioBuffer.getChannelData(channel);
    const targetData = normalized.getChannelData(channel);

    // Find peak
    let peak = 0;
    for (let i = 0; i &lt; sourceData.length; i++) {
      peak = Math.max(peak, Math.abs(sourceData[i]));
    }

    // Normalize
    const factor = 1 / peak;
    for (let i = 0; i &lt; sourceData.length; i++) {
      targetData[i] = sourceData[i] * factor;
    }
  }

  return normalized;
}

async audioBufferToWav(audioBuffer: AudioBuffer): Promise&lt;Blob&gt; {
  const numberOfChannels = audioBuffer.numberOfChannels;
  const sampleRate = audioBuffer.sampleRate;
  const format = 1; // PCM
  const bitDepth = 16;

  const bytesPerSample = bitDepth / 8;
  const blockAlign = numberOfChannels * bytesPerSample;

  const data = new Float32Array(audioBuffer.length * numberOfChannels);
  for (let channel = 0; channel &lt; numberOfChannels; channel++) {
```

```
      const channelData = audioBuffer.getChannelData(channel);
      for (let i = 0; i &lt; audioBuffer.length; i++) {
        data[i * numberOfChannels + channel] = channelData[i];
      }
    }

    const dataLength = data.length * bytesPerSample;
    const buffer = new ArrayBuffer(44 + dataLength);
    const view = new DataView(buffer);

    // Write WAV header
    this.writeString(view, 0, 'RIFF');
    view.setUint32(4, 36 + dataLength, true);
    this.writeString(view, 8, 'WAVE');
    this.writeString(view, 12, 'fmt ');
    view.setUint32(16, 16, true); // fmt chunk size
    view.setUint16(20, format, true);
    view.setUint16(22, numberOfChannels, true);
    view.setUint32(24, sampleRate, true);
    view.setUint32(28, sampleRate * blockAlign, true);
    view.setUint16(32, blockAlign, true);
    view.setUint16(34, bitDepth, true);
    this.writeString(view, 36, 'data');
    view.setUint32(40, dataLength, true);

    // Write audio data
    let offset = 44;
    for (let i = 0; i &lt; data.length; i++) {
      const sample = Math.max(-1, Math.min(1, data[i]));
      view.setInt16(offset, sample * 0x7FFF, true);
      offset += 2;
    }

    return new Blob([buffer], { type: 'audio/wav' });
  }

  private writeString(view: DataView, offset: number, string: string) {
    for (let i = 0; i &lt; string.length; i++) {
      view.setUint8(offset + i, string.charCodeAt(i));
    }
  }
}
```

9. Export & File Management

9.1 Audio Encoder (lib/audio-encoder.ts)

```
import lamejs from 'lamejs';
import { saveAs } from 'file-saver';

export class AudioEncoder {
  static async exportToMP3(
    audioBuffer: AudioBuffer,
```

```
    fileName: string = 'audio.mp3'
  ): Promise&lt;void&gt; {
    const mp3encoder = new lamejs.Mp3Encoder(
      audioBuffer.numberOfChannels,
      audioBuffer.sampleRate,
      128 // bitrate
    );

    const mp3Data: Int8Array[] = [];
    const sampleBlockSize = 1152;

    if (audioBuffer.numberOfChannels === 1) {
      // Mono
      const samples = this.convertFloat32ToInt16(
        audioBuffer.getChannelData(0)
      );

      for (let i = 0; i &lt; samples.length; i += sampleBlockSize) {
        const sampleChunk = samples.subarray(i, i + sampleBlockSize);
        const mp3buf = mp3encoder.encodeBuffer(sampleChunk);
        if (mp3buf.length &gt; 0) {
          mp3Data.push(mp3buf);
        }
      }
    } else {
      // Stereo
      const left = this.convertFloat32ToInt16(audioBuffer.getChannelData(0));
      const right = this.convertFloat32ToInt16(audioBuffer.getChannelData(1));

      for (let i = 0; i &lt; left.length; i += sampleBlockSize) {
        const leftChunk = left.subarray(i, i + sampleBlockSize);
        const rightChunk = right.subarray(i, i + sampleBlockSize);
        const mp3buf = mp3encoder.encodeBuffer(leftChunk, rightChunk);
        if (mp3buf.length &gt; 0) {
          mp3Data.push(mp3buf);
        }
      }
    }

    // Flush remaining data
    const mp3buf = mp3encoder.flush();
    if (mp3buf.length &gt; 0) {
      mp3Data.push(mp3buf);
    }

    const blob = new Blob(mp3Data, { type: 'audio/mp3' });
    saveAs(blob, fileName);
  }

  static async exportToWAV(
    audioBuffer: AudioBuffer,
    fileName: string = 'audio.wav'
  ): Promise&lt;void&gt; {
    const processor = new AudioProcessor(new AudioContext());
    const blob = await processor.audioBufferToWav(audioBuffer);
    saveAs(blob, fileName);
```

```
  }

  private static convertFloat32ToInt16(buffer: Float32Array): Int16Array {
    const l = buffer.length;
    const buf = new Int16Array(l);
    for (let i = 0; i &lt; l; i++) {
      const s = Math.max(-1, Math.min(1, buffer[i]));
      buf[i] = s &lt; 0 ? s * 0x8000 : s * 0x7FFF;
    }
    return buf;
  }
}
```

9.2 Export Panel Component

```
'use client';

import { useState } from 'react';
import { Download } from 'lucide-react';
import { Button } from '@/components/ui/Button';
import { AudioEncoder } from '@/lib/audio-encoder';

interface ExportPanelProps {
  audioBuffer: AudioBuffer | null;
  fileName: string;
}

export function ExportPanel({ audioBuffer, fileName }: ExportPanelProps) {
  const [isExporting, setIsExporting] = useState(false);
  const [format, setFormat] = useState&lt;'mp3' | 'wav'&gt;('mp3');

  const handleExport = async () =&gt; {
    if (!audioBuffer) return;

    setIsExporting(true);
    try {
      const outputFileName = `\${fileName.replace(/\\.[^/.]+$/, '')}.\${format}`;

      if (format === 'mp3') {
        await AudioEncoder.exportToMP3(audioBuffer, outputFileName);
      } else {
        await AudioEncoder.exportToWAV(audioBuffer, outputFileName);
      }
    } catch (error) {
      console.error('Export failed:', error);
    } finally {
      setIsExporting(false);
    }
  };

  return (
    <div>
      <h3>Export Audio</h3>

      <div>
```

```
        &lt;label className="text-sm text-slate-300"&gt;Format&lt;/label&gt;
        <div>
          &lt;button
            onClick={() =&gt; setFormat('mp3')}
            className={`
              px-4 py-2 rounded-lg font-medium transition-colors
              \${format === 'mp3'
                ? 'bg-blue-600 text-white'
                : 'bg-slate-700 text-slate-300 hover:bg-slate-600'
              }
            `}
          &gt;
            MP3
          &lt;/button&gt;
          &lt;button
            onClick={() =&gt; setFormat('wav')}
            className={`
              px-4 py-2 rounded-lg font-medium transition-colors
              \${format === 'wav'
                ? 'bg-blue-600 text-white'
                : 'bg-slate-700 text-slate-300 hover:bg-slate-600'
              }
            `}
          &gt;
            WAV
          &lt;/button&gt;
        </div>
      </div>

      &lt;Button
        onClick={handleExport}
        disabled={!audioBuffer || isExporting}
        className="w-full"
      &gt;
        &lt;Download className="mr-2 h-4 w-4" /&gt;
        {isExporting ? 'Exporting...' : `Export as \${format.toUpperCase()}`}
      &lt;/Button&gt;
    </div>
  );
}
```

10. Deployment & Optimization

10.1 Build Configuration

Update `next.config.js`:

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  // Optimize for production
  reactStrictMode: true,

  // Enable SWC minification
```

```
  swcMinify: true,

  // Webpack configuration for audio files
  webpack: (config) =&gt; {
    config.module.rules.push({
      test: /\\.(mp3|wav|ogg|m4a|aac)$/,
      use: {
        loader: 'file-loader',
        options: {
          publicPath: '/_next/static/sounds/',
          outputPath: 'static/sounds/',
          name: '[name].[hash].[ext]',
        },
      },
    });

    return config;
  },

  // Headers for SharedArrayBuffer (required for FFmpeg.wasm if used)
  async headers() {
    return [
      {
        source: '/:path*',
        headers: [
          {
            key: 'Cross-Origin-Opener-Policy',
            value: 'same-origin',
          },
          {
            key: 'Cross-Origin-Embedder-Policy',
            value: 'require-corp',
          },
        ],
      },
    ];
  },
};

module.exports = nextConfig;
```

10.2 Performance Optimization Tips

1. **Lazy Load Audio Libraries**:

```
import dynamic from 'next/dynamic';

const AudioEditor = dynamic(
  () =&gt; import('@/components/audio-editor/AudioPlayer'),
  { ssr: false }
);
```

2. **Use Web Workers for Heavy Processing**:

Create `workers/audio-worker.ts`:

```
self.addEventListener('message', async (event) =&gt; {
  const { type, data } = event.data;

  if (type === 'encode-mp3') {
    // Perform MP3 encoding in worker
    const result = await encodeToMP3(data);
    self.postMessage({ type: 'encoded', data: result });
  }
});
```

3. **Optimize Audio Loading**:

```
// Use streaming for large files
async function streamAudio(url: string) {
  const response = await fetch(url);
  const reader = response.body?.getReader();

  // Process chunks as they arrive
  while (reader) {
    const { done, value } = await reader.read();
    if (done) break;
    // Process chunk
  }
}
```

10.3 Deployment

Vercel Deployment:

```
# Install Vercel CLI
npm i -g vercel

# Deploy
vercel --prod
```

Netlify Deployment:

```
# Build command
npm run build

# Publish directory
.next
```

Conclusion

You now have a complete guide to building a professional audio editing web application with Next.js 15, TypeScript, and Tailwind CSS. This application leverages powerful browser APIs and lightweight libraries to provide a seamless audio editing experience entirely in the browser.

Key Takeaways

- **Modern Stack**: Next.js 15 with App Router, TypeScript, and Tailwind CSS
- **Powerful Audio Processing**: Web Audio API, wavesurfer.js, Tone.js
- **Client-Side Processing**: No server required for audio manipulation
- **Professional Features**: Waveform visualization, effects, recording, export

Next Steps

1. Add more audio effects (compression, distortion, pitch shifting)
2. Implement undo/redo with command pattern
3. Add keyboard shortcuts for power users
4. Create preset management system
5. Add collaborative editing features
6. Implement project save/load functionality

Happy coding!

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26]

[27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99]

⁂

1. https://www.youtube.com/watch?v=sYw9tlGeFrg
2. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Using_Web_Audio_API
3. https://www.youtube.com/watch?v=hdI0sMQAYcs
4. https://dyte.io/blog/web-audio-api/
5. https://github.com/madzadev/audio-player
6. https://nextjs.org/blog/next-15-5
7. https://dev.to/adamchaudhry/understanding-web-audio-apis-a-comprehensive-guide-1nce
8. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
9. https://www.youtube.com/watch?v=jE4FUFERfqI
10. https://audiographiclab.com/library/tonejs
11. https://github.com/goldfire/howler.js
12. https://wavesurfer.xyz
13. https://tonejs.github.io

14. https://github.com/katspaugh/wavesurfer.js
15. https://github.com/JorenSix/ffmpeg.audio.wasm
16. https://github.com/zhuker/lamejs
17. https://www.geeksforgeeks.org/javascript/how-to-record-and-play-audio-in-javascript/
18. https://github.com/ffmpegwasm/ffmpeg.wasm
19. https://www.youtube.com/watch?v=IrOulguDIjo
20. https://blog.logrocket.com/building-audio-player-react/
21. https://github.com/Rajatm544/react-audio-editor
22. https://www.newline.co/courses/fullstack-react-with-typescript-masterclass/build-a-keyboard-app-with-a-react-hook-and-the-web-audio-api
23. https://www.youtube.com/watch?v=hBuoUgOKKqY
24. https://flowbite.com/docs/forms/radio/
25. https://flowbite.com/application-ui/demo/audio/in-call/
26. https://www.subframe.com/templates/tags/audio-player
27. https://ayeshasahar.hashnode.dev/script-to-audio-generator-web-app-using-nextjs-tailwindcss-typescript-python-audiostack-fastapi
28. https://www.geeksforgeeks.org/javascript/web-audio-api/
29. https://stackoverflow.com/questions/14823465/looking-for-audio-library-for-editing-audio-files
30. https://www.techradar.com/best/best-free-audio-editors
31. https://stackoverflow.com/questions/73723558/audio-files-in-next-js
32. https://www.nch.com.au/wavepad/index.html
33. https://nextjs.org/blog/next-15
34. https://web.dev/patterns/media/audio-effects
35. https://nextjs.org/docs/app/guides/backend-for-frontend
36. https://www.w3.org/TR/webaudio-1.1/
37. https://sourceforge.net/projects/howler-js.mirror/
38. https://dev.to/snelson723/wavesurferjs-4k22
39. https://www.cyberlink.com/blog/the-top-audio-editors/453/free-audio-editing-software
40. https://dev.to/snelson723/tonejs-and-the-web-audio-api-36cj
41. https://github.com/gluckgames/howler.js.2.0
42. https://wavesurfer.xyz/faq/
43. https://www.c-sharpcorner.com/article/composing-music-in-javascript-using-tone-js/
44. https://howlerjs.com
45. https://wavesurfer.xyz/examples/
46. https://stackoverflow.com/questions/70910548/play-loaded-audio-file-with-tone-js-web-audio-framework
47. https://cran.r-project.org/web/packages/howler/vignettes/howler.html
48. https://github.com/Tonejs
49. https://www.infoq.com/news/2018/11/howlerjs-audio-modern-web/
50. https://www.npmjs.com/search?q=wavesurfer

51. https://www.youtube.com/watch?v=ftGuDvm5zDQ
52. https://uikit.webclown.net/wavesurfer/v6.6.4/docs/
53. https://transloadit.com/devtips/real-time-video-filters-in-browsers-with-ffmpeg-and-webcodecs/
54. https://scribbler.live/2024/12/05/Coverting-Wav-to-Mp3-in-JavaScript-Using-Lame-js.html
55. https://addpipe.com/simple-recorderjs-demo/
56. https://stackoverflow.com/questions/28099493/running-ffmpeg-in-browser-options
57. https://github.com/higuma/mp3-lame-encoder-js
58. https://addpipe.com/webaudiorecorder-demo/
59. https://stackoverflow.com/questions/21764632/how-to-encode-pcm-data-to-mp3-in-javascript
60. https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API/Using_the_MediaStream_Recording_API
61. https://www.npmjs.com/package/@diffusion-studio/ffmpeg-js
62. https://codesandbox.io/examples/package/lamejs
63. https://www.youtube.com/watch?v=sbf6vQ8xym4
64. https://www.ffmpeg.org
65. https://zapier.com/blog/best-audio-editor/
66. https://www.npmjs.com/search?q=mp3
67. https://stackoverflow.com/questions/50263732/how-to-record-audio-from-web-browser-and-download-it-with-js
68. https://ffmpegwasm.netlify.app
69. https://www.jsdelivr.com/package/npm/lame
70. https://www.youtube.com/watch?v=kjQtsVZ2OFM
71. https://www.w3.org/press-releases/2021/webaudio/
72. https://www.youtube.com/watch?v=nWC89T8rWsU
73. https://ej2.syncfusion.com/react/documentation/rich-text-editor/audio
74. https://akoskm.com/building-an-audio-transcription-app-with-nextjs/
75. https://www.npmjs.com/package/react-audio-voice-recorder
76. https://docs.speechmatics.com/voice-agents-flow/guides/nextjs-guide
77. https://www.npmjs.com/package/@chayns/react-audio-editor
78. https://dzone.com/articles/implementing-spatial-audio-with-web-audio-api
79. https://codesandbox.io/s/react-audio-component-33wpc
80. https://dev.to/chaoocharles/build-and-deploy-a-realtime-video-call-application-nextjs-webrtc-socketio-fha
81. https://reactscript.com/audio-editor-react/
82. https://p.bdir.in/p/audio-editor-for-react/6906
83. https://scribbler.live/2024/05/18/Web-Audio-API-Interactive-Sound-Applications-in-JavaScript.html
84. https://stackblitz.com/edit/stackblitz-starters-t5nci5
85. https://nextjs.org/docs/app
86. https://www.mux.com/blog/tailwind-css-audio-player-theme-in-media-chrome
87. https://reactflow.dev/learn/tutorials/react-flow-and-the-web-audio-api
88. https://next-intl.dev/docs/getting-started/app-router

89. https://www.letsbuildui.dev/articles/building-an-audio-player-with-react-hooks/
90. https://www.audacityteam.org
91. https://nextjs.org/blog/building-apis-with-nextjs
92. https://nextjs.org/docs/app/getting-started
93. https://tailwindflex.com/@abhi/music-player
94. https://stackoverflow.com/questions/70644845/react-web-audio-api-play-pause-and-export-loaded-audio-file
95. https://nextjs.org/docs/app/getting-started/installation
96. https://daisyui.com/?lang=en
97. https://joeyreyes.dev/blog/web-audio-api-in-react
98. https://clerk.com/docs/getting-started/quickstart
99. https://www.reddit.com/r/tailwindcss/comments/ykabxu/visual_editors_for_tailwind_what_are_you_using/