# TTS Campaign Episode Processor - Project Outline

## 🎯 Project Overview

A web application that converts D&D campaign episode scripts (markdown format) into high-quality TTS audio using ElevenLabs API, with intelligent speaker detection, batch processing, and cost controls.

---

## 📁 Project Structure

```
campaign-tts-processor/
├── frontend/
│   ├── index.html          # Main interface
│   ├── app.js              # Core application logic
│   ├── styles.css          # Clean, focused styling
│   └── components/
│       ├── upload.js       # File upload handling
│       ├── parser.js       # Markdown/script parser
│       ├── preview.js      # Speaker preview component
│       └── processor.js    # TTS processing logic
├── backend/
│   ├── server.js           # Express server
│   ├── api/
│   │   ├── elevenlabs.js   # ElevenLabs API wrapper
│   │   ├── auth.js         # API key management
│   │   └── queue.js        # Processing queue
│   ├── utils/
│   │   ├── parser.js       # Server-side parsing
│   │   ├── splitter.js     # Text chunking for API limits
│   │   └── merger.js       # Audio file merging
│   └── config/
│       └── voices.json     # Voice configuration
├── outputs/                # Generated audio files
├── .env.example            # Environment variables template
└── README.md               # Setup instructions
```

---

## 🔧 Core Features (MVP)

### 1. File Upload & Parsing

javascript

```javascript
// Parse episode format
class EpisodeParser {
    parseScript(markdown) {
        const scenes = [];
        const speakers = new Set();

        // Split by timestamps (e.g., ## **00:47:00**)
        const sections = markdown.split(/##\s*\*\*\d{2}:\d{2}:\d{2}\*\*/);

        sections.forEach(section => {
            const dialogues = [];

            // Match speaker patterns: **SPEAKER:** or **SPEAKER (DESCRIPTION):**
            const speakerRegex = /\*\*([A-Z\s]+)(?:\s*\([^)]+\))?\s*:\*\*/g;

            let match;
            while ((match = speakerRegex.exec(section))) {
                const speaker = match[1].trim();
                speakers.add(speaker);

                // Extract dialogue until next speaker or section end
                const startIndex = match.index + match[0].length;
                const nextMatch = speakerRegex.exec(section);
                const endIndex = nextMatch ? nextMatch.index : section.length;

                const dialogue = section.substring(startIndex, endIndex).trim();

                // Check if dialogue needs chunking (rare but important for DM narration)
                if (dialogue.length > 1000) {
                    console.log(`Long dialogue detected for ${speaker}: ${dialogue.length} char
                }

                dialogues.push({ speaker, text: dialogue });

                // Reset regex position
                speakerRegex.lastIndex = nextMatch ? nextMatch.index : section.length;
            }

            scenes.push(dialogues);
        });

        return { scenes, speakers: Array.from(speakers) };
    }
}
```

## 2. **Speaker Detection & Mapping**

```javascript
// Auto-detect all speakers in episode
const speakers = {
    "JOE": { voice: "Daniel", settings: {...} },
    "TRONALD": { voice: "Adam", settings: {...} },
    "SAXY": { voice: "Bella", settings: {...} },
    // etc...
}
```

## 3. **Processing Interface**

- Upload episode file
- Auto-detect speakers
- Map speakers to ElevenLabs voices
- Preview each speaker's voice
- Set processing options
- Monitor progress
- Download results

## 4. **Cost Controls**

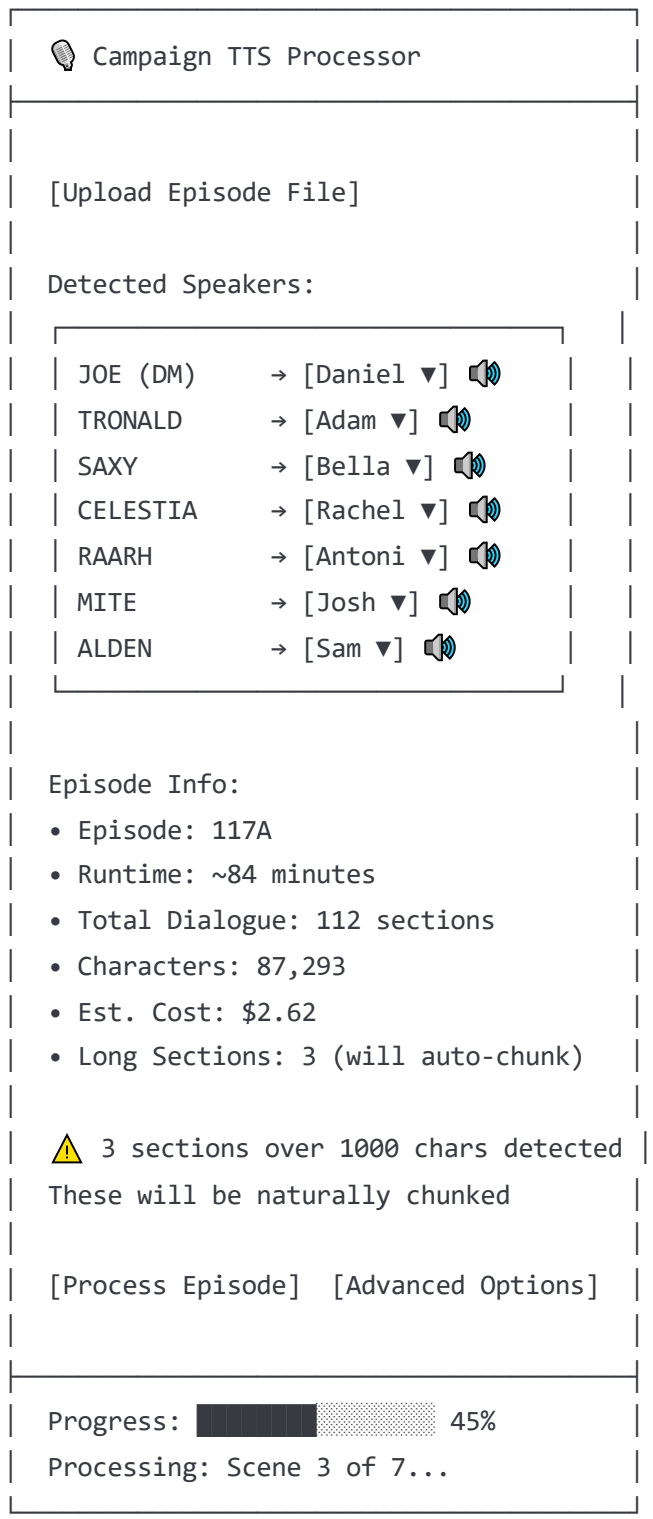
javascript

```javascript
// Failsafes to prevent overspending
const costControls = {
    maxCharactersPerChunk: 1200,    // Optimal for natural speech
    maxCharactersPerRun: 50000,     // Total episode limit
    estimateCost: (characters) => characters * 0.00003,
    requireConfirmation: true,
    dailyLimit: 1000000, // characters
    warningThreshold: 0.8,

    // Auto-chunk detection
    autoChunkThreshold: 1000,  // Chunk anything over 1000 chars

    validateChunk: (text) => {
        if (text.length > 4500) {
            throw new Error('Single chunk exceeds ElevenLabs limit! This should never happen.')
        }
        return true;
    }
}
```

---

## 💻 Frontend Design

### Main Interface Layout




---

## 🔌 ElevenLabs Integration

### API Configuration

javascript

```javascript
// backend/api/elevenlabs.js
const ElevenLabs = require('elevenlabs-api');

class TTSProcessor {
    constructor(apiKey) {
        this.client = new ElevenLabs({ apiKey });
        this.voiceCache = new Map();
        this.chunker = new DialogueChunker();
    }

    async processDialogue(text, speaker, settings) {
        // Smart chunking for long dialogue (mainly DM narration)
        const chunks = this.chunker.chunkLongDialogue(text, speaker);
        const audioFiles = [];

        for (const chunk of chunks) {
            console.log(`Processing ${speaker} chunk ${chunk.chunkNumber} (${chunk.text.length}

            const voice = this.voiceCache.get(speaker) ||
                          await this.loadVoice(speaker);

            const audio = await this.client.textToSpeech({
                text: chunk.text,
                voice_id: voice.id,
                model_id: "eleven_monolingual_v1",
                voice_settings: {
                    stability: settings.stability || 0.75,
                    similarity_boost: settings.similarity || 0.75,
                    style: settings.style || 0.5,
                    use_speaker_boost: true
                }
            });

            audioFiles.push({
                speaker,
                chunkNumber: chunk.chunkNumber,
                audio,
                naturalBreak: chunk.naturalBreak
            });
        }

        return audioFiles;
    }
}
```

## Smart Text Chunking for Natural Speech

javascript

```
// Intelligent chunking for natural TTS flow
class DialogueChunker {
    constructor() {
        // ElevenLabs sounds best with 700-1200 chars
        this.optimalChunkSize = 1000;
        this.maxChunkSize = 1200;
        this.minChunkSize = 500;
    }

    chunkLongDialogue(text, speaker) {
        // Most dialogue is under 1000 chars, but DM sections can be longer
        if (text.length < this.maxChunkSize) {
            return [{ text, speaker, chunkNumber: 1 }];
        }

        const chunks = [];
        const paragraphs = text.split(/\n\n+/);
        let currentChunk = '';
        let chunkNumber = 1;

        for (const para of paragraphs) {
            // If adding this paragraph exceeds optimal size, split here
            if (currentChunk &&
                (currentChunk.length + para.length > this.optimalChunkSize)) {

                // Save current chunk
                chunks.push({
                    text: currentChunk.trim(),
                    speaker,
                    chunkNumber: chunkNumber++,
                    naturalBreak: true
                });
                currentChunk = para;
            } else {
                // Add to current chunk
                currentChunk += (currentChunk ? '\n\n' : '') + para;
            }

            // Emergency split if single paragraph is too long
            if (currentChunk.length > this.maxChunkSize) {
                const sentences = currentChunk.match(/[^.!?]+[.!?]+/g) || [];
                let tempChunk = '';

                for (const sentence of sentences) {
                    if (tempChunk.length + sentence.length > this.optimalChunkSize) {
                        chunks.push({
```

```
                        text: tempChunk.trim(),
                        speaker,
                        chunkNumber: chunkNumber++,
                        naturalBreak: false
                    });
                    tempChunk = sentence;
                } else {
                    tempChunk += ' ' + sentence;
                }
            }
            currentChunk = tempChunk;
        }
    }

    // Don't forget the last chunk
    if (currentChunk.trim()) {
        chunks.push({
            text: currentChunk.trim(),
            speaker,
            chunkNumber: chunkNumber,
            naturalBreak: true
        });
    }

    return chunks;
}

// Example output for DM narration:
// Chunk 1: "You step into the courtyard..." (743 chars) ✓
// Chunk 2: "The statues around you..." (891 chars) ✓
// Chunk 3: "As you examine the seal..." (656 chars) ✓
}
```

## Real-World Example

javascript

```javascript
// Your transcript rarely has super long sections, but when it does:
const dmNarration = `You step into what was once a grand courtyard...
[700 characters of description]

The ground is cracked flagstone, weeds pushing through...
[500 characters of description]

Scattered across the area are the remains...
[800 characters of description]`;

const chunks = chunker.chunkLongDialogue(dmNarration, 'JOE');
// Returns 3 natural chunks instead of awkward mid-sentence breaks
```

---

## Security & Error Handling

### API Key Management

javascript

```javascript
// Never expose API keys in frontend
// backend/.env
ELEVENLABS_API_KEY=your_key_here
DAILY_CHAR_LIMIT=1000000
COST_PER_CHAR=0.00003

// backend/auth.js
const validateRequest = (req, res, next) => {
    const { characters } = req.body;
    const cost = characters * process.env.COST_PER_CHAR;

    if (cost > 50) { // $50 limit
        return res.status(403).json({
            error: 'Cost limit exceeded',
            estimated: cost
        });
    }
    next();
};
```

### Error Recovery

```javascript
// Retry failed chunks
async function processWithRetry(chunk, maxRetries = 3) {
    for (let i = 0; i < maxRetries; i++) {
        try {
            return await processChunk(chunk);
        } catch (error) {
            if (i === maxRetries - 1) throw error;
            await delay(1000 * (i + 1)); // Exponential backoff
        }
    }
}
```

---

## 📝 Implementation Steps (for Cursor)

### Step 1: Basic Setup

```bash
# Create project structure
mkdir campaign-tts-processor
cd campaign-tts-processor
npm init -y
npm install express dotenv multer
npm install elevenlabs-node-api  # or official SDK
```

### Step 2: File Upload & Parsing

Tell Cursor:

> "Create a file upload interface that accepts markdown files. Parse the markdown to extract speaker names (format: **SPEAKER_NAME:**) and their dialogue. Show me a preview of detected speakers."

### Step 3: Voice Mapping Interface

Tell Cursor:

> "Create a speaker-to-voice mapping interface. Each detected speaker should have a dropdown to select an ElevenLabs voice and a preview button to test the voice with sample text."

### Step 4: Processing Engine

Tell Cursor:

> "Create a processing engine that takes the parsed dialogue and sends it to ElevenLabs API. For any dialogue over 1000 characters, use the DialogueChunker class to split at paragraph breaks. Show

> which sections are being chunked in the UI. Save audio files with format: Episode_117A_Scene_1_JOE_chunk1.mp3"

### Step 5: Audio Merging

Tell Cursor:

> "Create a system to merge all generated audio files in the correct order based on timestamps. Output a single Episode_117A_Complete.mp3 file."

---

## 🎨 UI/UX Guidelines

### Design Principles

- **Clarity First**: Show what's happening at every step
- **Cost Visible**: Always show estimated cost before processing
- **Progress Tracking**: Real-time updates during processing
- **Error Recovery**: Clear error messages with solutions

### Color Scheme

```css
:root {
    --primary: #5b21b6;      /* Purple - main actions */
    --secondary: #1e293b;    /* Dark slate - headers */
    --success: #10b981;      /* Green - completed */
    --warning: #f59e0b;      /* Amber - warnings */
    --danger: #ef4444;       /* Red - errors */
    --bg: #f8fafc;           /* Light background */
    --text: #334155;         /* Main text */
}
```

---

## 🚀 Advanced Features (Phase 2)

### 1. Multi-Voice Scenes

- Detect when multiple characters speak together
- Process overlapping dialogue appropriately

### 2. Sound Effects Integration

- Add ambient sounds between scenes
- Include dice rolling sounds for combat

### 3. **Voice Memory**

- Save speaker-to-voice mappings
- Auto-apply to future episodes

### 4. **Batch Processing**

- Queue multiple episodes
- Process overnight with rate limiting

---

## 📋 Testing Checklist

- [ ] Upload markdown file
- [ ] Correctly parse all speakers
- [ ] Map voices and preview
- [ ] Calculate accurate cost
- [ ] Process small test section
- [ ] Handle API errors gracefully
- [ ] Merge audio correctly
- [ ] Download final file

---

## 🖇️ Cursor Communication Guide

When implementing with Cursor, use these exact phrases:

1. **"Create the basic file structure as shown in the project outline"**
2. **"Implement the episode parser to extract speakers and dialogue from markdown"**
3. **"Build the speaker detection and voice mapping interface"**
4. **"Create the ElevenLabs API integration with proper error handling"**
5. **"Add cost calculation and confirmation before processing"**
6. **"Implement the audio processing queue with progress tracking"**
7. **"Create the audio file merger to combine all segments"**

Remember to test each component individually before connecting them!

---

## 📌 Important Notes

- **NEVER** put API keys in frontend code
- **ALWAYS** show cost estimates before processing
- **CHUNK** text properly to avoid API limits
- **SAVE** progress in case of failures

- **TEST** with small sections first

This focused approach gives you exactly what you need without unnecessary complexity. Start with the MVP and add features as needed!