Contents

SECTION 21: VOICE & VIDEO INPUT (v3.6.0)

21.1 Voice/Video Overview

Integration with Deepgram for speech-to-text and ElevenLabs for text-to-speech.

21.2 Voice Database Schema

```sql
-- migrations/030_voice_video.sql

CREATE TABLE voice_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    session_type VARCHAR(20) NOT NULL,
    input_format VARCHAR(20),
    output_format VARCHAR(20),
    language VARCHAR(10) DEFAULT 'en',
    voice_id VARCHAR(100),
    total_duration_ms INTEGER DEFAULT 0,
    total_cost DECIMAL(10, 6) DEFAULT 0,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    ended_at TIMESTAMPTZ
);

CREATE TABLE voice_transcriptions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES voice_sessions(id) ON DELETE CASCADE,
    audio_url VARCHAR(500),
    transcription TEXT,
    confidence DECIMAL(3, 2),
    duration_ms INTEGER,
    word_timestamps JSONB,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE voice_synthesis (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES voice_sessions(id) ON DELETE CASCADE,
    input_text TEXT NOT NULL,
    audio_url VARCHAR(500),
    voice_id VARCHAR(100) NOT NULL,
    duration_ms INTEGER,
    character_count INTEGER,
    cost DECIMAL(10, 6),
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_voice_sessions_user ON voice_sessions(tenant_id, user_id);
CREATE INDEX idx_voice_transcriptions ON voice_transcriptions(session_id);

ALTER TABLE voice_sessions ENABLE ROW LEVEL SECURITY;
ALTER TABLE voice_transcriptions ENABLE ROW LEVEL SECURITY;
ALTER TABLE voice_synthesis ENABLE ROW LEVEL SECURITY;

CREATE POLICY voice_sessions_isolation ON voice_sessions USING (tenant_id = current_setting('a
CREATE POLICY voice_transcriptions_isolation ON voice_transcriptions USING (
    session_id IN (SELECT id FROM voice_sessions WHERE tenant_id = current_setting('app.curren
);
CREATE POLICY voice_synthesis_isolation ON voice_synthesis USING (
    session_id IN (SELECT id FROM voice_sessions WHERE tenant_id = current_setting('app.curren
);
```

21.3 Voice Service Integration

```
// packages/core/src/services/voice-service.ts

import { Pool } from 'pg';
import { S3Client, PutObjectCommand, GetObjectCommand } from '@aws-sdk/client-s3';
import { getSignedUrl } from '@aws-sdk/s3-request-presigner';

interface DeepgramResponse {
    results: {
        channels: Array<{
            alternatives: Array<{
                transcript: string;
                confidence: number;
                words: Array<{ word: string; start: number; end: number }>;
            }>;
        }>;
    };
    metadata: {
        duration: number;
    };
```

```
}

export class VoiceService {
    private pool: Pool;
    private s3: S3Client;
    private deepgramApiKey: string;
    private elevenLabsApiKey: string;

    constructor(pool: Pool) {
        this.pool = pool;
        this.s3 = new S3Client({});
        this.deepgramApiKey = process.env.DEEPGRAM_API_KEY!;
        this.elevenLabsApiKey = process.env.ELEVENLABS_API_KEY!;
    }

    async createSession(
        tenantId: string,
        userId: string,
        sessionType: 'stt' | 'tts' | 'realtime',
        options?: { language?: string; voiceId?: string }
    ): Promise<string> {
        const result = await this.pool.query(`
            INSERT INTO voice_sessions (tenant_id, user_id, session_type, language, voice_id)
            VALUES ($1, $2, $3, $4, $5)
            RETURNING id
        `, [tenantId, userId, sessionType, options?.language || 'en', options?.voiceId]);

        return result.rows[0].id;
    }

    async transcribe(
        sessionId: string,
        audioBuffer: Buffer,
        format: string = 'webm'
    ): Promise<{ transcription: string; confidence: number; wordTimestamps: any[] }> {
        // Upload audio to S3
        const audioKey = `voice/${sessionId}/${Date.now()}.${format}`;
        await this.s3.send(new PutObjectCommand({
            Bucket: process.env.ASSETS_BUCKET!,
            Key: audioKey,
            Body: audioBuffer,
            ContentType: `audio/${format}`
        }));

        // Call Deepgram API
        const response = await fetch('https://api.deepgram.com/v1/listen?model=nova-2&smart_fo
            method: 'POST',
            headers: {
                'Authorization': `Token ${this.deepgramApiKey}`,
                'Content-Type': `audio/${format}`
            },
            body: audioBuffer
        });

        const data: DeepgramResponse = await response.json();
        const result = data.results.channels[0].alternatives[0];

        // Store transcription
        const audioUrl = await this.getSignedUrl(audioKey);
        await this.pool.query(`
            INSERT INTO voice_transcriptions (session_id, audio_url, transcription, confidence
            VALUES ($1, $2, $3, $4, $5, $6)
        `, [sessionId, audioUrl, result.transcript, result.confidence, data.metadata.duration *

        return {
            transcription: result.transcript,
            confidence: result.confidence,
            wordTimestamps: result.words
        };
    }

    async synthesize(
        sessionId: string,
        text: string,
        voiceId: string = 'EXAVITQu4vr4xnSDxMaL'
    ): Promise<{ audioUrl: string; durationMs: number }> {
        // Call ElevenLabs API
        const response = await fetch(`https://api.elevenlabs.io/v1/text-to-speech/${voiceId}`,
            method: 'POST',
            headers: {
                'Accept': 'audio/mpeg',
                'xi-api-key': this.elevenLabsApiKey,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                text,
                model_id: 'eleven_multilingual_v2',
                voice_settings: {
                    stability: 0.5,
                    similarity_boost: 0.75
                }
            })
        });

        const audioBuffer = Buffer.from(await response.arrayBuffer());
        // Upload to S3
        const audioKey = `voice/${sessionId}/${Date.now()}_synthesis.mp3`;
        await this.s3.send(new PutObjectCommand({
            Bucket: process.env.ASSETS_BUCKET!,
            Key: audioKey,
            Body: audioBuffer,
            ContentType: 'audio/mpeg'
        }));

        const audioUrl = await this.getSignedUrl(audioKey);
        const durationMs = this.estimateDuration(text);
        const cost = text.length * 0.00003; // Approximate ElevenLabs cost

        // Store synthesis record
        await this.pool.query(`
            INSERT INTO voice_synthesis (session_id, input_text, audio_url, voice_id, duration
            VALUES ($1, $2, $3, $4, $5, $6, $7)
        `, [sessionId, text, audioUrl, voiceId, durationMs, text.length, cost]);

        return { audioUrl, durationMs };
    }

    private async getSignedUrl(key: string): Promise<string> {
        const command = new GetObjectCommand({
            Bucket: process.env.ASSETS_BUCKET!,
            Key: key
        });
        return getSignedUrl(this.s3, command, { expiresIn: 3600 });
    }

    private estimateDuration(text: string): number {
        // Rough estimate: ~150 words per minute
        const wordCount = text.split(/\s+/).length;
        return Math.round((wordCount / 150) * 60 * 1000);
    }
}
```