Contents

SECTION 17: SIMPLE AUTO-RESOLVE API (v3.3.0) **1**

1

17.1 Auto-Resolve Overview 1
17.2 Auto-Resolve Database Schema 1
17.3 Auto-Resolve Service 1

4

SECTION 17: SIMPLE AUTO-RESOLVE API (v3.3.0)

17.1 Auto-Resolve Overview

Intelligent model selection API that automatically picks the best model based on the request.

17.2 Auto-Resolve Database Schema

```sql
-- migrations/026_auto_resolve.sql

CREATE TABLE auto_resolve_requests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    request_type VARCHAR(50) NOT NULL,
    selected_model VARCHAR(100) NOT NULL,
    selection_reason TEXT,
    user_preferences JSONB DEFAULT '{}',
    input_tokens INTEGER,
    output_tokens INTEGER,
    cost DECIMAL(10, 6),
    latency_ms INTEGER,
    success BOOLEAN DEFAULT true,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_auto_resolve_tenant ON auto_resolve_requests(tenant_id, created_at DESC);
CREATE INDEX idx_auto_resolve_model ON auto_resolve_requests(selected_model);

ALTER TABLE auto_resolve_requests ENABLE ROW LEVEL SECURITY;
CREATE POLICY auto_resolve_isolation ON auto_resolve_requests USING (tenant_id = current_setti
```

17.3 Auto-Resolve Service

```ts
// packages/core/src/services/auto-resolve.ts
```

```
import { Pool } from 'pg';
import { BrainRouter } from './brain-router';

interface AutoResolveRequest {
    tenantId: string;
    userId: string;
    prompt: string;
    preferences?: {
        maxCost?: number;
        maxLatencyMs?: number;
        preferredProvider?: string;
        qualityLevel?: 'economy' | 'balanced' | 'premium';
    };
}

interface AutoResolveResult {
    model: string;
    provider: string;
    reason: string;
    estimatedCost: number;
}

export class AutoResolveService {
    private pool: Pool;
    private router: BrainRouter;

    constructor(pool: Pool) {
        this.pool = pool;
        this.router = new BrainRouter(pool);
    }

    async resolve(request: AutoResolveRequest): Promise<AutoResolveResult> {
        // Analyze the prompt
        const analysis = this.analyzePrompt(request.prompt);

        // Get routing result
        const routing = await this.router.route({
            tenantId: request.tenantId,
            userId: request.userId,
            taskType: analysis.taskType,
            inputTokenEstimate: analysis.tokenEstimate,
            maxLatencyMs: request.preferences?.maxLatencyMs,
            maxCost: request.preferences?.maxCost,
            preferredProvider: request.preferences?.preferredProvider,
            requiresVision: analysis.requiresVision,
            requiresAudio: analysis.requiresAudio
        });
```

```
        // Log the request
        await this.pool.query(`
            INSERT INTO auto_resolve_requests (tenant_id, user_id, request_type, selected_mode
            VALUES ($1, $2, $3, $4, $5, $6)
        `, [
            request.tenantId,
            request.userId,
            analysis.taskType,
            routing.model,
            routing.reason,
            JSON.stringify(request.preferences || {})
        ]);

        return {
            model: routing.model,
            provider: routing.provider,
            reason: routing.reason,
            estimatedCost: routing.estimatedCost
        };
    }

    private analyzePrompt(prompt: string): {
        taskType: 'chat' | 'code' | 'analysis' | 'creative' | 'vision' | 'audio';
        tokenEstimate: number;
        requiresVision: boolean;
        requiresAudio: boolean;
    } {
        const lowerPrompt = prompt.toLowerCase();

        let taskType: 'chat' | 'code' | 'analysis' | 'creative' | 'vision' | 'audio' = 'chat';

        if (lowerPrompt.includes('code') || lowerPrompt.includes('function') || lowerPrompt.in
            taskType = 'code';
        } else if (lowerPrompt.includes('analyze') || lowerPrompt.includes('data') || lowerPro
            taskType = 'analysis';
        } else if (lowerPrompt.includes('write') || lowerPrompt.includes('story') || lowerProm
            taskType = 'creative';
        }

        return {
            taskType,
            tokenEstimate: Math.ceil(prompt.length / 4),
            requiresVision: lowerPrompt.includes('image') || lowerPrompt.includes('picture'),
            requiresAudio: lowerPrompt.includes('audio') || lowerPrompt.includes('speech')
        };
    }
}
```