Contents

SECTION 25: FOCUS MODES & CUSTOM PERSONAS (v3.6.0)

25.1 Focus Modes Overview

Pre-configured AI behavior profiles and custom persona creation.

25.2 Personas Database Schema

```sql
-- migrations/034_focus_personas.sql

CREATE TABLE focus_modes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID REFERENCES tenants(id),
    mode_name VARCHAR(100) NOT NULL,
    display_name VARCHAR(200) NOT NULL,
    description TEXT,
    icon VARCHAR(50),
    system_prompt TEXT NOT NULL,
    default_model VARCHAR(100),
    settings JSONB DEFAULT '{}',
    is_system BOOLEAN DEFAULT false,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE user_personas (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    persona_name VARCHAR(100) NOT NULL,
    display_name VARCHAR(200),
    avatar_url VARCHAR(500),
    system_prompt TEXT NOT NULL,
    voice_id VARCHAR(100),
    personality_traits JSONB DEFAULT '[]',
    knowledge_domains JSONB DEFAULT '[]',
```

```
    conversation_style JSONB DEFAULT '{}',
    is_public BOOLEAN DEFAULT false,
    usage_count INTEGER DEFAULT 0,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE persona_usage_log (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    persona_id UUID NOT NULL REFERENCES user_personas(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id),
    chat_id UUID REFERENCES chats(id),
    tokens_used INTEGER,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_focus_modes_tenant ON focus_modes(tenant_id) WHERE tenant_id IS NOT NULL;
CREATE INDEX idx_user_personas_user ON user_personas(tenant_id, user_id);
CREATE INDEX idx_persona_usage ON persona_usage_log(persona_id, created_at DESC);

ALTER TABLE focus_modes ENABLE ROW LEVEL SECURITY;
ALTER TABLE user_personas ENABLE ROW LEVEL SECURITY;
ALTER TABLE persona_usage_log ENABLE ROW LEVEL SECURITY;

-- System modes visible to all, tenant modes to tenant only
CREATE POLICY focus_modes_policy ON focus_modes USING (
    is_system = true OR tenant_id IS NULL OR tenant_id = current_setting('app.current_tenant_i
);
CREATE POLICY user_personas_isolation ON user_personas USING (
    tenant_id = current_setting('app.current_tenant_id')::UUID AND
    (user_id = current_setting('app.user_id')::UUID OR is_public = true)
);
CREATE POLICY persona_usage_isolation ON persona_usage_log USING (
    persona_id IN (SELECT id FROM user_personas WHERE tenant_id = current_setting('app.current

-- Insert default focus modes
INSERT INTO focus_modes (mode_name, display_name, description, icon, system_prompt, is_system)
('general', 'General Assistant', 'Versatile AI for any task', 'MessageSquare', 'You are a helpf
('code', 'Code Expert', 'Programming and development focus', 'Code', 'You are an expert softwa
('writer', 'Creative Writer', 'Creative writing and content creation', 'PenTool', 'You are a cr
('analyst', 'Data Analyst', 'Data analysis and insights', 'BarChart', 'You are a data analyst.
('researcher', 'Research Assistant', 'Deep research and fact-finding', 'Search', 'You are a res
```

25.3 Persona Service

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

```typescript
import { Pool } from 'pg';

interface PersonaCreate {
    name: string;
    displayName?: string;
    systemPrompt: string;
    avatarUrl?: string;
    voiceId?: string;
    traits?: string[];
    domains?: string[];
    style?: Record<string, any>;
    isPublic?: boolean;
}

export class PersonaService {
    private pool: Pool;

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

    async getFocusModes(tenantId?: string): Promise<any[]> {
        const result = await this.pool.query(`
            SELECT * FROM focus_modes
            WHERE is_active = true AND (is_system = true OR tenant_id IS NULL OR tenant_id = $1
            ORDER BY is_system DESC, mode_name
        `, [tenantId]);

        return result.rows;
    }

    async createPersona(tenantId: string, userId: string, persona: PersonaCreate): Promise<str
        const result = await this.pool.query(`
            INSERT INTO user_personas (
                tenant_id, user_id, persona_name, display_name, system_prompt,
                avatar_url, voice_id, personality_traits, knowledge_domains,
                conversation_style, is_public
            )
            VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11)
            RETURNING id
        `, [
            tenantId,
            userId,
            persona.name,
            persona.displayName,
            persona.systemPrompt,
            persona.avatarUrl,
            persona.voiceId,
```

```
            JSON.stringify(persona.traits || []),
            JSON.stringify(persona.domains || []),
            JSON.stringify(persona.style || {}),
            persona.isPublic || false
        ]);

        return result.rows[0].id;
    }

    async getUserPersonas(tenantId: string, userId: string): Promise<any[]> {
        const result = await this.pool.query(`
            SELECT * FROM user_personas
            WHERE tenant_id = $1 AND (user_id = $2 OR is_public = true)
            ORDER BY usage_count DESC
        `, [tenantId, userId]);

        return result.rows;
    }

    async getPersona(personaId: string): Promise<any> {
        const result = await this.pool.query(`SELECT * FROM user_personas WHERE id = $1`, [pers
        return result.rows[0];
    }

    async updatePersona(personaId: string, updates: Partial<PersonaCreate>): Promise<void> {
        const fields: string[] = [];
        const values: any[] = [personaId];
        let paramIndex = 2;

        if (updates.name) { fields.push(`persona_name = $${paramIndex++}`); values.push(updates
        if (updates.displayName) { fields.push(`display_name = $${paramIndex++}`); values.push
        if (updates.systemPrompt) { fields.push(`system_prompt = $${paramIndex++}`); values.pus
        if (updates.avatarUrl) { fields.push(`avatar_url = $${paramIndex++}`); values.push(upda
        if (updates.voiceId) { fields.push(`voice_id = $${paramIndex++}`); values.push(updates
        if (updates.traits) { fields.push(`personality_traits = $${paramIndex++}`); values.pus
        if (updates.domains) { fields.push(`knowledge_domains = $${paramIndex++}`); values.pus
        if (updates.style) { fields.push(`conversation_style = $${paramIndex++}`); values.push
        if (typeof updates.isPublic === 'boolean') { fields.push(`is_public = $${paramIndex++}

        fields.push('updated_at = NOW()');

        await this.pool.query(`UPDATE user_personas SET ${fields.join(', ')} WHERE id = $1`, va
    }

    async logUsage(personaId: string, userId: string, chatId?: string, tokensUsed?: number): Pr
        await this.pool.query(`
            INSERT INTO persona_usage_log (persona_id, user_id, chat_id, tokens_used)
            VALUES ($1, $2, $3, $4)
```

```
        `, [personaId, userId, chatId, tokensUsed]);

        await this.pool.query(`
            UPDATE user_personas SET usage_count = usage_count + 1 WHERE id = $1
        `, [personaId]);
    }

    async buildPrompt(personaId: string, userMessage: string): Promise<string> {
        const persona = await this.getPersona(personaId);
        if (!persona) throw new Error('Persona not found');

        const traits = persona.personality_traits as string[];
        const domains = persona.knowledge_domains as string[];

        let prompt = persona.system_prompt;

        if (traits.length > 0) {
            prompt += `\n\nPersonality traits: ${traits.join(', ')}.`;
        }

        if (domains.length > 0) {
            prompt += `\n\nAreas of expertise: ${domains.join(', ')}.`;
        }

        return prompt;
    }
}
```