Contents

SECTION 27: FAMILY & TEAM PLANS (v3.6.0)

27.1 Team Plans Overview

Shared subscription plans for families and teams with usage allocation.

27.2 Team Plans Database Schema

```sql
-- migrations/036_team_plans.sql

CREATE TABLE team_plans (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    plan_name VARCHAR(100) NOT NULL,
    plan_type VARCHAR(20) NOT NULL,
    owner_id UUID NOT NULL REFERENCES users(id),
    max_members INTEGER NOT NULL DEFAULT 5,
    total_tokens_monthly BIGINT NOT NULL,
    shared_pool BOOLEAN DEFAULT true,
    billing_email VARCHAR(255),
    stripe_subscription_id VARCHAR(100),
    is_active BOOLEAN DEFAULT true,
    current_period_start TIMESTAMPTZ,
    current_period_end TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE team_members (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    team_id UUID NOT NULL REFERENCES team_plans(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id),
    role VARCHAR(20) NOT NULL DEFAULT 'member',
    token_allocation BIGINT,
    tokens_used_this_period BIGINT DEFAULT 0,
    invited_by UUID REFERENCES users(id),
    invited_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
    accepted_at TIMESTAMPTZ,
    is_active BOOLEAN DEFAULT true,
    UNIQUE(team_id, user_id)
);

CREATE TABLE team_usage_log (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    team_id UUID NOT NULL REFERENCES team_plans(id) ON DELETE CASCADE,
    member_id UUID NOT NULL REFERENCES team_members(id) ON DELETE CASCADE,
    tokens_used INTEGER NOT NULL,
    model VARCHAR(100),
    usage_type VARCHAR(50),
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_team_plans_tenant ON team_plans(tenant_id);
CREATE INDEX idx_team_members_user ON team_members(user_id);
CREATE INDEX idx_team_usage ON team_usage_log(team_id, created_at DESC);

ALTER TABLE team_plans ENABLE ROW LEVEL SECURITY;
ALTER TABLE team_members ENABLE ROW LEVEL SECURITY;
ALTER TABLE team_usage_log ENABLE ROW LEVEL SECURITY;

CREATE POLICY team_plans_isolation ON team_plans USING (tenant_id = current_setting('app.curre
CREATE POLICY team_members_isolation ON team_members USING (
    team_id IN (SELECT id FROM team_plans WHERE tenant_id = current_setting('app.current_tenan
);
CREATE POLICY team_usage_isolation ON team_usage_log USING (
    team_id IN (SELECT id FROM team_plans WHERE tenant_id = current_setting('app.current_tenan
);
```

27.3 Team Service

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

import { Pool } from 'pg';

type PlanType = 'family' | 'team' | 'enterprise';
type MemberRole = 'owner' | 'admin' | 'member';

interface TeamCreate {
    name: string;
    type: PlanType;
    maxMembers: number;
    totalTokensMonthly: number;
    sharedPool?: boolean;
    billingEmail?: string;
}
```

```typescript
export class TeamService {
    private pool: Pool;

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

    async createTeam(tenantId: string, ownerId: string, team: TeamCreate): Promise<string> {
        const result = await this.pool.query(`
            INSERT INTO team_plans (
                tenant_id, plan_name, plan_type, owner_id, max_members,
                total_tokens_monthly, shared_pool, billing_email
            )
            VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
            RETURNING id
        `, [
            tenantId, team.name, team.type, ownerId, team.maxMembers,
            team.totalTokensMonthly, team.sharedPool ?? true, team.billingEmail
        ]);

        const teamId = result.rows[0].id;

        // Add owner as first member
        await this.addMember(teamId, ownerId, 'owner');

        return teamId;
    }

    async addMember(teamId: string, userId: string, role: MemberRole = 'member', invitedBy?: st
        const team = await this.getTeam(teamId);
        const memberCount = await this.getMemberCount(teamId);

        if (memberCount >= team.max_members) {
            throw new Error('Team member limit reached');
        }

        const result = await this.pool.query(`
            INSERT INTO team_members (team_id, user_id, role, invited_by, accepted_at)
            VALUES ($1, $2, $3, $4, CASE WHEN $3 = 'owner' THEN NOW() ELSE NULL END)
            RETURNING id
        `, [teamId, userId, role, invitedBy]);

        return result.rows[0].id;
    }

    async removeMember(teamId: string, userId: string): Promise<void> {
        // Check not removing owner
```

```
        const member = await this.getMember(teamId, userId);
        if (member?.role === 'owner') {
            throw new Error('Cannot remove team owner');
        }

        await this.pool.query(`
            UPDATE team_members SET is_active = false WHERE team_id = $1 AND user_id = $2
        `, [teamId, userId]);
    }

    async acceptInvitation(teamId: string, userId: string): Promise<void> {
        await this.pool.query(`
            UPDATE team_members SET accepted_at = NOW() WHERE team_id = $1 AND user_id = $2
        `, [teamId, userId]);
    }

    async allocateTokens(teamId: string, userId: string, tokens: number): Promise<void> {
        await this.pool.query(`
            UPDATE team_members SET token_allocation = $3 WHERE team_id = $1 AND user_id = $2
        `, [teamId, userId, tokens]);
    }

    async recordUsage(teamId: string, userId: string, tokensUsed: number, model?: string): Pro
        const member = await this.getMember(teamId, userId);
        if (!member) throw new Error('Not a team member');

        // Check allocation if not shared pool
        const team = await this.getTeam(teamId);
        if (!team.shared_pool && member.token_allocation) {
            if (member.tokens_used_this_period + tokensUsed > member.token_allocation) {
                throw new Error('Token allocation exceeded');
            }
        }

        // Update member usage
        await this.pool.query(`
            UPDATE team_members SET tokens_used_this_period = tokens_used_this_period + $3
            WHERE team_id = $1 AND user_id = $2
        `, [teamId, userId, tokensUsed]);

        // Log usage
        await this.pool.query(`
            INSERT INTO team_usage_log (team_id, member_id, tokens_used, model)
            VALUES ($1, $2, $3, $4)
        `, [teamId, member.id, tokensUsed, model]);
    }

    async getTeamUsageStats(teamId: string): Promise<any> {
```

```
    const team = await this.getTeam(teamId);

    const members = await this.pool.query(`
        SELECT tm.*, u.email, u.display_name
        FROM team_members tm
        JOIN users u ON tm.user_id = u.id
        WHERE tm.team_id = $1 AND tm.is_active = true
    `, [teamId]);

    const totalUsed = members.rows.reduce((sum: number, m: any) => sum + (m.tokens_used_thi

    return {
        teamId,
        planName: team.plan_name,
        totalTokensMonthly: team.total_tokens_monthly,
        tokensUsed: totalUsed,
        tokensRemaining: team.total_tokens_monthly - totalUsed,
        members: members.rows.map((m: any) => ({
            userId: m.user_id,
            email: m.email,
            displayName: m.display_name,
            role: m.role,
            tokensUsed: m.tokens_used_this_period,
            allocation: m.token_allocation
        }))
    };
}

async resetPeriodUsage(teamId: string): Promise<void> {
    await this.pool.query(`
        UPDATE team_members SET tokens_used_this_period = 0 WHERE team_id = $1
    `, [teamId]);

    const now = new Date();
    const nextMonth = new Date(now.getFullYear(), now.getMonth() + 1, now.getDate());

    await this.pool.query(`
        UPDATE team_plans
        SET current_period_start = NOW(), current_period_end = $2
        WHERE id = $1
    `, [teamId, nextMonth]);
}

private async getTeam(teamId: string) {
    const result = await this.pool.query(`SELECT * FROM team_plans WHERE id = $1`, [teamId]
    return result.rows[0];
}
```

```
    private async getMember(teamId: string, userId: string) {
        const result = await this.pool.query(
            `SELECT * FROM team_members WHERE team_id = $1 AND user_id = $2`,
            [teamId, userId]
        );
        return result.rows[0];
    }

    private async getMemberCount(teamId: string): Promise<number> {
        const result = await this.pool.query(
            `SELECT COUNT(*) FROM team_members WHERE team_id = $1 AND is_active = true`,
            [teamId]
        );
        return parseInt(result.rows[0].count);
    }
}
```