Contents

SECTION 20: REAL-TIME COLLABORATION (v3.6.0)

20.1 Collaboration Overview

Real-time collaborative editing using Yjs CRDT for shared workspaces.

20.2 Collaboration Database Schema

```sql
-- migrations/029_realtime_collaboration.sql

CREATE TABLE collaboration_rooms (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    room_name VARCHAR(100) NOT NULL,
    room_type VARCHAR(50) NOT NULL DEFAULT 'document',
    created_by UUID NOT NULL REFERENCES users(id),
    yjs_document BYTEA,
    is_active BOOLEAN DEFAULT true,
    max_participants INTEGER DEFAULT 10,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE room_participants (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    room_id UUID NOT NULL REFERENCES collaboration_rooms(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id),
    role VARCHAR(20) NOT NULL DEFAULT 'editor',
    cursor_position JSONB,
    is_online BOOLEAN DEFAULT false,
    joined_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    last_seen TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(room_id, user_id)
);

CREATE TABLE collaboration_history (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    room_id UUID NOT NULL REFERENCES collaboration_rooms(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id),
    action_type VARCHAR(50) NOT NULL,
    action_data JSONB,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_collab_rooms_tenant ON collaboration_rooms(tenant_id);
CREATE INDEX idx_room_participants ON room_participants(room_id);
CREATE INDEX idx_collab_history ON collaboration_history(room_id, created_at DESC);

ALTER TABLE collaboration_rooms ENABLE ROW LEVEL SECURITY;
ALTER TABLE room_participants ENABLE ROW LEVEL SECURITY;
ALTER TABLE collaboration_history ENABLE ROW LEVEL SECURITY;

CREATE POLICY collab_rooms_isolation ON collaboration_rooms USING (tenant_id = current_setting
CREATE POLICY room_participants_isolation ON room_participants USING (
    room_id IN (SELECT id FROM collaboration_rooms WHERE tenant_id = current_setting('app.curre
);
CREATE POLICY collab_history_isolation ON collaboration_history USING (
    room_id IN (SELECT id FROM collaboration_rooms WHERE tenant_id = current_setting('app.curre
);
```

20.3 Yjs Collaboration Provider

```
// packages/core/src/services/yjs-provider.ts

import * as Y from 'yjs';
import { WebsocketProvider } from 'y-websocket';
import { Pool } from 'pg';

export class YjsCollaborationProvider {
    private pool: Pool;
    private documents: Map<string, Y.Doc> = new Map();
    private providers: Map<string, WebsocketProvider> = new Map();

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

    async getOrCreateRoom(
        tenantId: string,
        roomId: string,
        userId: string
    ): Promise<{ doc: Y.Doc; provider: WebsocketProvider }> {
        // Check if document already exists in memory
        if (this.documents.has(roomId)) {
```

```
        return {
            doc: this.documents.get(roomId)!,
            provider: this.providers.get(roomId)!
        };
    }

    // Load from database or create new
    const result = await this.pool.query(
        `SELECT yjs_document FROM collaboration_rooms WHERE id = $1 AND tenant_id = $2`,
        [roomId, tenantId]
    );

    const doc = new Y.Doc();

    if (result.rows[0]?.yjs_document) {
        Y.applyUpdate(doc, result.rows[0].yjs_document);
    }

    // Create WebSocket provider
    const wsUrl = process.env.YJS_WEBSOCKET_URL || 'wss://collab.radiant.ai';
    const provider = new WebsocketProvider(wsUrl, roomId, doc);

    // Set up persistence
    doc.on('update', async (update: Uint8Array) => {
        await this.persistDocument(roomId, Y.encodeStateAsUpdate(doc));
    });

    this.documents.set(roomId, doc);
    this.providers.set(roomId, provider);

    // Add user as participant
    await this.addParticipant(roomId, userId);

    return { doc, provider };
}

async addParticipant(roomId: string, userId: string): Promise<void> {
    await this.pool.query(`
        INSERT INTO room_participants (room_id, user_id, is_online)
        VALUES ($1, $2, true)
        ON CONFLICT (room_id, user_id)
        DO UPDATE SET is_online = true, last_seen = NOW()
    `, [roomId, userId]);
}

async removeParticipant(roomId: string, userId: string): Promise<void> {
    await this.pool.query(`
        UPDATE room_participants SET is_online = false, last_seen = NOW()
```

```
            WHERE room_id = $1 AND user_id = $2
        `, [roomId, userId]);
    }

    async updateCursor(roomId: string, userId: string, position: any): Promise<void> {
        await this.pool.query(`
            UPDATE room_participants SET cursor_position = $3, last_seen = NOW()
            WHERE room_id = $1 AND user_id = $2
        `, [roomId, userId, JSON.stringify(position)]);
    }

    private async persistDocument(roomId: string, update: Uint8Array): Promise<void> {
        await this.pool.query(`
            UPDATE collaboration_rooms SET yjs_document = $2, updated_at = NOW()
            WHERE id = $1
        `, [roomId, Buffer.from(update)]);
    }

    async getParticipants(roomId: string): Promise<any[]> {
        const result = await this.pool.query(`
            SELECT rp.*, u.email, u.display_name
            FROM room_participants rp
            JOIN users u ON rp.user_id = u.id
            WHERE rp.room_id = $1
        `, [roomId]);
        return result.rows;
    }

    async logAction(roomId: string, userId: string, actionType: string, data: any): Promise<vo
        await this.pool.query(`
            INSERT INTO collaboration_history (room_id, user_id, action_type, action_data)
            VALUES ($1, $2, $3, $4)
        `, [roomId, userId, actionType, JSON.stringify(data)]);
    }
}
```