LinkedIn AI Assistant - System Design Documentation

Table of Contents

System Overview

Purpose

An AI-powered LinkedIn automation system that:

1. Records conversations between users and contacts via PWA mobile app
2. Uses AI (OpenAI Whisper + GPT-4) to transcribe and analyze conversations
3. Automatically manages LinkedIn connections and messaging via Chrome extension
4. Provides AI autopilot mode for fully automated LinkedIn responses

Three-Component Architecture

Component 1: PWA Mobile App

- Records audio conversations
- Scans LinkedIn QR codes to identify contacts
- Uploads audio to Supabase Storage
- Displays conversation summaries after AI processing

Component 2: Supabase Backend

- PostgreSQL database for all data
- Supabase Auth for user authentication
- Supabase Storage for audio files
- Edge Functions for AI processing (transcription, analysis, response generation)

- Realtime subscriptions for live sync between PWA and Chrome extension

Component 3: Chrome Extension

- Sidebar UI showing contacts and conversation history
- Content script that monitors LinkedIn DOM for new messages
- Auto-pilot mode: AI automatically responds to LinkedIn messages
- Automatic connection request sending with personalized messages

Data Flow

```
User records conversation on PWA
  → Audio uploaded to Supabase Storage
  → Edge Function transcribes (Whisper API)
  → Edge Function analyzes (GPT-4)
  → Data saved to database
  → Realtime sync to Chrome Extension
  → Extension shows contact in sidebar
  → When auto-pilot ON: Extension monitors LinkedIn
  → New message detected → AI generates response → Automatically sent
```

Tech Stack

Frontend

- **Framework**: React 18+ with TypeScript
- **Build Tool**: Vite
- **Styling**: Tailwind CSS
- **State Management**: React hooks
- **Audio Recording**: MediaRecorder API (browser native)
- **QR Scanner**: html5-qrcode library

Backend

- **Platform**: Supabase
- **Database**: PostgreSQL with Row Level Security
- **Authentication**: Supabase Auth (email/password)
- **Storage**: Supabase Storage (audio files)
- **Functions**: Supabase Edge Functions (Deno/TypeScript)
- **Realtime**: Supabase Realtime (WebSocket subscriptions)

AI Services

- **Transcription**: OpenAI Whisper API
- **Analysis & Generation**: OpenAI GPT-4 API

Chrome Extension

- **Manifest Version**: 3
- **Content Scripts**: TypeScript (DOM manipulation)
- **Background**: Service Worker (TypeScript)
- **Build Plugin**: @crxjs/vite-plugin

Database Schema

Tables

1. contacts

Stores information about people the user has recorded conversations with.

```sql
CREATE TABLE contacts (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  name TEXT NOT NULL,
  linkedin_url TEXT NOT NULL UNIQUE,
  linkedin_profile_data JSONB DEFAULT '{}'::jsonb,
  relationship_type TEXT DEFAULT 'professional_contact',
  custom_instructions TEXT,
  auto_pilot_enabled BOOLEAN DEFAULT false,
  status TEXT DEFAULT 'not_contacted',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Constraints
ALTER TABLE contacts ADD CONSTRAINT relationship_type_check
  CHECK (relationship_type IN ('potential_client', 'professional_contact', 'friend', 'colleague', 'other'));

ALTER TABLE contacts ADD CONSTRAINT status_check
  CHECK (status IN ('not_contacted', 'connection_sent', 'connected', 'declined'));
```

Fields:

- `id`: Primary key
- `user_id`: Reference to authenticated user
- `name`: Contact's full name
- `linkedin_url`: Their LinkedIn profile URL (from QR code)
- `linkedin_profile_data`: JSON object with title, company, etc.
- `relationship_type`: Category of relationship
- `custom_instructions`: User-defined instructions for AI responses
- `auto_pilot_enabled`: Whether AI should auto-respond to this contact
- `status`: Current connection status on LinkedIn
- `created_at`, `updated_at`: Timestamps

---

2. conversations

Stores recorded conversations and AI analysis.

```sql
CREATE TABLE conversations (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  contact_id UUID REFERENCES contacts(id) ON DELETE CASCADE NOT NULL,
  audio_url TEXT,
  transcription TEXT,
  summary TEXT,
  key_topics TEXT[] DEFAULT ARRAY[]::TEXT[],
  ai_analysis JSONB DEFAULT '{}'::jsonb,
  duration_seconds INTEGER,
  recorded_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  processed BOOLEAN DEFAULT false
);
```

Fields:

- `id`: Primary key
- `user_id`: User who recorded the conversation
- `contact_id`: The contact involved in conversation
- `audio_url`: Path to audio file in Supabase Storage
- `transcription`: Full text transcription from Whisper
- `summary`: AI-generated summary (2-3 sentences)

- `key_topics`: Array of main topics discussed
- `ai_analysis`: JSON with sentiment, action items, etc.
- `duration_seconds`: Length of recording
- `recorded_at`: When conversation was recorded
- `processed`: Whether AI has finished processing

---

3. ai_actions

Logs all actions AI takes on behalf of the user.

```sql
CREATE TABLE ai_actions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  contact_id UUID REFERENCES contacts(id) ON DELETE CASCADE,
  action_type TEXT NOT NULL,
  action_content TEXT,
  success BOOLEAN DEFAULT true,
  metadata JSONB DEFAULT '{}'::jsonb,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Constraint
ALTER TABLE ai_actions ADD CONSTRAINT action_type_check
  CHECK (action_type IN ('connection_request', 'message_sent', 'post_liked', 'comment_posted', 'profile_viewed'));
```

Fields:

- `id`: Primary key
- `user_id`: User whose behalf action was taken
- `contact_id`: Contact receiving the action
- `action_type`: Type of LinkedIn action
- `action_content`: The actual message/comment text
- `success`: Whether action succeeded
- `metadata`: Additional data (incoming message, etc.)
- `created_at`: When action occurred

---

4. linkedin_messages

Stores message history between user and contacts.

```sql
CREATE TABLE linkedin_messages (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  contact_id UUID REFERENCES contacts(id) ON DELETE CASCADE NOT NULL,
  sender TEXT NOT NULL,
  message_text TEXT NOT NULL,
  ai_generated BOOLEAN DEFAULT false,
  sent_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Constraint
ALTER TABLE linkedin_messages ADD CONSTRAINT sender_check
  CHECK (sender IN ('user', 'contact'));
```

Fields:

- `id`: Primary key
- `user_id`: User in the conversation
- `contact_id`: Contact in the conversation
- `sender`: Who sent the message ('user' or 'contact')
- `message_text`: The actual message content
- `ai_generated`: Whether this was AI-generated or human-written
- `sent_at`: Timestamp

Indexes

```sql
CREATE INDEX idx_contacts_user_id ON contacts(user_id);
CREATE INDEX idx_contacts_linkedin_url ON contacts(linkedin_url);
CREATE INDEX idx_conversations_user_id ON conversations(user_id);
CREATE INDEX idx_conversations_contact_id ON conversations(contact_id);
CREATE INDEX idx_ai_actions_contact_id ON ai_actions(contact_id);
CREATE INDEX idx_ai_actions_user_id ON ai_actions(user_id);
CREATE INDEX idx_linkedin_messages_contact_id ON linkedin_messages(contact_id);
CREATE INDEX idx_linkedin_messages_sent_at ON linkedin_messages(sent_at DESC);
```

Row Level Security (RLS)

Enable RLS on all tables and create policies so users can only access their own data:

```sql
ALTER TABLE contacts ENABLE ROW LEVEL SECURITY;
ALTER TABLE conversations ENABLE ROW LEVEL SECURITY;
ALTER TABLE ai_actions ENABLE ROW LEVEL SECURITY;
ALTER TABLE linkedin_messages ENABLE ROW LEVEL SECURITY;

-- Contacts policies
CREATE POLICY "Users can view own contacts" ON contacts FOR SELECT USING (auth.uid() = user_id);
CREATE POLICY "Users can insert own contacts" ON contacts FOR INSERT WITH CHECK (auth.uid() = user_id);
CREATE POLICY "Users can update own contacts" ON contacts FOR UPDATE USING (auth.uid() = user_id);
CREATE POLICY "Users can delete own contacts" ON contacts FOR DELETE USING (auth.uid() = user_id);

-- Similar policies for other tables...
```

API Specifications

Supabase Edge Functions

1. process-audio

Purpose: Transcribe and analyze recorded conversation using AI

Endpoint: `POST /functions/v1/process-audio`

Request Body:

```typescript
{
  audioUrl: string;       // Path in Supabase Storage (e.g., "user-id/timestamp.wav")
  contactId: string;      // UUID of contact
  conversationId: string; // UUID of conversation record
  userId: string;         // UUID of user
}
```

Response:

```typescript
```

```
{
  success: boolean;
  analysis: {
    transcription: string;       // Full text transcription
    summary: string;             // 2-3 sentence summary
    key_topics: string[];        // Array of topics discussed
    suggested_relationship: string; // Suggested relationship type
    action_items: string[];      // Follow-up actions
    sentiment: string;           // 'positive' | 'neutral' | 'negative'
  }
}
```

Processing Logic:

1. Download audio file from Supabase Storage using `audioUrl`
2. Send audio to OpenAI Whisper API for transcription
3. Send transcription to OpenAI GPT-4 with this prompt:

```
Analyze this business conversation and extract:
- A concise summary (2-3 sentences)
- Key topics discussed (array of strings)
- Suggested relationship type (potential_client, professional_contact, friend, colleague)
- Action items or follow-ups needed
- Overall sentiment (positive, neutral, negative)

Return response as JSON.
```

4. Update `conversations` table with transcription, summary, key_topics, ai_analysis
5. Update `contacts` table with suggested relationship type
6. Set `processed = true` on conversation record

---

2. generate-response

Purpose: Generate AI response to a LinkedIn message based on conversation context

Endpoint: `POST /functions/v1/generate-response`

Request Body:

typescript

```
{
  contactId: string;      // UUID of contact who sent message
  incomingMessage: string; // The message text from LinkedIn
  userId: string;         // UUID of user
}
```

Response:

typescript

```
{
  generatedMessage: string; // AI-generated response (under 200 words)
  metadata: {
    model: string;           // e.g., "gpt-4"
    tokens_used: number;
    generation_time_ms: number;
  }
}
```

Processing Logic:

1. Query database for contact info (relationship_type, custom_instructions)
2. Fetch recent message history (last 10 messages) from `linkedin_messages` table
3. Fetch conversation summary from `conversations` table
4. Build GPT-4 prompt:

```
You are helping respond to a LinkedIn message professionally.

Context:
- Relationship: [relationship_type]
- Conversation Summary: [summary from recorded conversation]
- Recent Messages: [last 10 messages]
- Custom Instructions: [custom_instructions if provided]

They just sent: "[incoming_message]"

Generate a natural, professional response (under 200 words).
Match their tone and reference relevant context from your conversation.
```

5. Call OpenAI GPT-4 API
6. Insert record into `ai_actions` table (action_type: 'message_sent')
7. Insert records into `linkedin_messages` table (incoming message + AI response)

8. Return generated message

---

3. send-connection-request

Purpose: Generate personalized LinkedIn connection request message

Endpoint: `POST /functions/v1/send-connection-request`

Request Body:

```typescript
{
  contactId: string;  // UUID of contact
  userId: string;     // UUID of user
}
```

Response:

```typescript
{
  connectionMessage: string;  // Personalized message (max 300 characters)
  success: boolean;
}
```

Processing Logic:

1. Fetch contact and conversation data from database
2. Build GPT-4 prompt:

```
Generate a personalized LinkedIn connection request message.
Maximum 300 characters (LinkedIn limit).

Context:
- Met: [conversation recorded_at date]
- Discussed: [key_topics from conversation]
- Their role: [title at company from linkedin_profile_data]

Be warm, professional, and reference the conversation naturally.
```

3. Call OpenAI GPT-4 API
4. Ensure response is ≤ 300 characters (truncate if needed)
5. Insert record into `ai_actions` table (action_type: 'connection_request')

6. Return message

Component Architecture

PWA Mobile App Components

App Structure

```
App (Root)
├── Router
│   ├── LoginScreen (if not authenticated)
│   ├── RecordingScreen (main screen, if authenticated)
│   └── SuccessScreen (after recording processed)
```

LoginScreen

- **Purpose**: User authentication
- **State**: `email`, `password`, `mode` (login/signup), `loading`, `error`
- **UI**: Email input, password input, submit button, toggle login/signup
- **Actions**: `handleLogin()`, `handleSignup()` using Supabase Auth

RecordingScreen

- **Purpose**: Main interface for recording conversations
- **State**:
  - `isRecording`: boolean
  - `duration`: number (seconds)
  - `contact`: Contact | null
  - `showQRScanner`: boolean
  - `processing`: boolean
- **Child Components**:
  - QRScanner (modal overlay)
  - ContactDisplay (shows scanned contact info)
  - RecordButton (start/stop recording)
  - AudioWaveform (visual feedback during recording)
- **Flow**:
  1. Show "Scan QR Code" button
  2. User scans → Contact info displayed

3. "Start Recording" button becomes active
4. User records → Shows duration and waveform
5. User stops → Upload to Supabase Storage
6. Show processing screen
7. Navigate to SuccessScreen when processed

QRScanner

- **Purpose**: Scan LinkedIn QR codes using device camera
- **Props**: `onScan(linkedinUrl)`, `onClose()`
- **Library**: html5-qrcode
- **Logic**:
  - Request camera permission
  - Scan for QR code
  - Extract LinkedIn URL
  - Validate URL format
  - Call `onScan` callback

SuccessScreen

- **Purpose**: Display conversation summary after AI processing
- **Props**: `conversationId`
- **State**: `conversation`, `loading`
- **UI**:
  - Success checkmark
  - Contact name
  - Summary text
  - Key topics (as chips/tags)
  - Suggested next steps
  - "Record Another" button
  - "View Full Details" button
  - Tip about Chrome extension

Chrome Extension Components

Extension Structure

```
Chrome Extension
├── Sidebar (React App)
│   ├── ContactList
│   └── ContactDetail
├── Content Script (linkedin.ts)
│   └── DOM monitoring & manipulation
└── Background Service Worker
    └── API coordination
```

Sidebar - ContactList

- **Purpose**: Display all contacts from database
- **State**: `contacts`, `loading`, `filter` (all/new/connected)
- **Features**:
  - Search bar
  - Filter tabs
  - Contact cards showing:
    - Name, title, company
    - Status badge (NEW, Connected, etc.)
    - Auto-pilot toggle
    - Last interaction timestamp
    - Conversation summary snippet
- **Realtime**: Subscribe to `contacts` table changes
- **Actions**: Click contact → Navigate to ContactDetail

Sidebar - ContactDetail

- **Purpose**: Show detailed view of a contact
- **Props**: `contact`, `onBack()`
- **State**: `conversation`, `aiActions`, `messages`, `editingRelationship`
- **UI Sections**:
  1. **Header**: Name, LinkedIn link, connection status
  2. **Conversation Summary**: Date, summary, key topics
  3. **Relationship Settings**:
     - Dropdown for relationship type
     - Textarea for custom AI instructions
     - Save button

4. **Auto-Pilot Toggle**:
    - Large toggle switch
    - Confirmation dialog when enabling
    - Status indicator
5. **AI Actions Log**:
    - List of automated actions taken
    - Timestamps
    - Success/failure indicators
6. **Message History**:
    - Timeline of LinkedIn messages
    - Visual distinction between user and AI messages

Content Script (linkedin.ts)

- **Purpose**: Monitor LinkedIn DOM and automate actions
- **Key Functions**:
    1. **initializeMonitoring()**
        - Sets up MutationObserver on LinkedIn's messaging container
        - Watches for new incoming messages
    2. **isNewIncomingMessage(node)**
        - Checks if DOM node is a new message from someone else (not user)
    3. **extractMessageData(node)**
        - Extracts: sender name, LinkedIn URL, message text, timestamp
    4. **handleNewMessage(node)**
        - Sends message data to background script
        - Background checks if auto-pilot enabled
        - If yes, triggers AI response
    5. **sendLinkedInMessage(text)**
        - Finds message input box (`.msg-form__contenteditable`)
        - Inserts AI-generated text
        - Triggers input event
        - Clicks send button (`.msg-form__send-button`)
    6. **sendConnectionRequest(profileUrl, message)**
        - Navigates to profile if needed
        - Clicks "Connect" button

- Optionally adds personalized note
- Clicks "Send"

DOM Selectors (LinkedIn class names):

```typescript
const SELECTORS = {
  messageList: '.msg-s-message-list-container',
  incomingMessage: '.msg-s-event-listitem:not(.msg-s-message-list__event--from-self)',
  messageBox: '.msg-form__contenteditable',
  sendButton: '.msg-form__send-button',
  messageText: '.msg-s-event-listitem__body',
  senderProfile: '.msg-s-message-group__profile-link'
};
```

Background Service Worker

- **Purpose**: Coordinate between content script and Supabase backend
- **Key Functions**:
  1. **handleNewLinkedInMessage(messageData)**
     - Receives message data from content script
     - Queries Supabase: Is auto-pilot enabled for this contact?
     - If yes: Call `generate-response` Edge Function
     - Send AI response back to content script
     - Content script sends the message
     - Log action to database
  2. **sendConnectionRequest(contactId)**
     - Call `send-connection-request` Edge Function
     - Get generated connection message
     - Send to content script to execute
     - Update contact status to 'connection_sent'

User Flows

Flow 1: Record Conversation

Actors: User, PWA App, Supabase Backend, AI Services

Steps:

1. User opens PWA on mobile device
2. User logs in (if not authenticated)
3. **Main Screen appears**
4. User taps "Scan QR Code"
5. QR scanner opens using device camera
6. User scans contact's LinkedIn QR code
7. **Contact info displayed** (name, title, company)
8. User taps "Start Recording"
9. **Recording begins** - duration counter and waveform shown
10. User has conversation with contact
11. User taps "Stop Recording"
12. **Processing begins**:
    - Audio uploaded to Supabase Storage
    - `conversation` record created in database
    - Edge Function `process-audio` triggered
    - Whisper API transcribes audio
    - GPT-4 analyzes transcription
    - Database updated with results
13. **Success screen shown**:
    - Conversation summary
    - Key topics
    - Suggested relationship type
14. User can:
    - View full details
    - Record another conversation
    - Close app

Database Changes:

- New record in `contacts` table
- New record in `conversations` table
- `conversations.processed` set to `true` after AI processing

Flow 2: Automatic Connection Request (Chrome Extension)

Actors: User, Chrome Extension, Supabase Backend, LinkedIn

Steps:

1. User opens LinkedIn in Chrome browser
2. Chrome extension loads automatically
3. Extension sidebar shows recent contacts
4. **New contact appears** (from recent PWA recording)
   - Marked with "NEW" badge
   - Status: "not_contacted"
5. Extension automatically (no user action needed):
   - Calls `send-connection-request` Edge Function
   - Receives personalized connection message (≤300 chars)
6. Content script:
   - Navigates to contact's LinkedIn profile
   - Clicks "Connect" button
   - Adds personalized note with AI-generated message
   - Clicks "Send"
7. **Database updated**:
   - `contacts.status` = 'connection_sent'
   - New record in `ai_actions` table
8. Extension sidebar updates:
   - Status badge changes to "Connection Sent"
   - Timestamp shown

User sees: Contact automatically moved from "not contacted" to "connection sent" without any manual action.

Flow 3: Auto-Pilot Message Response

Actors: Contact (on LinkedIn), Chrome Extension, Supabase Backend, AI Services

Prerequisites:

- Contact's `auto_pilot_enabled` = true
- User is browsing LinkedIn with extension active

Steps:

1. Contact sends message to user on LinkedIn
2. **Content script detects** new message via MutationObserver
3. Content script extracts:
   - Sender's name
   - Sender's LinkedIn URL
   - Message text
4. Content script sends to background service worker
5. **Background worker**:
   - Queries Supabase: `SELECT * FROM contacts WHERE linkedin_url = ?`
   - Checks: Is `auto_pilot_enabled` = true?
6. If yes:
   - Calls Edge Function `generate-response`
   - Edge Function:
     - Fetches conversation context
     - Fetches message history
     - Builds GPT-4 prompt with context
     - Generates appropriate response
   - Returns AI-generated message
7. **Background sends response** to content script
8. **Content script**:
   - Finds message input box on LinkedIn
   - Inserts AI-generated text
   - Clicks send button
   - Message sent automatically
9. **Database updated**:
   - Incoming message saved to `linkedin_messages` (sender: 'contact')
   - AI response saved to `linkedin_messages` (sender: 'user', ai_generated: true)
   - New record in `ai_actions` (action_type: 'message_sent')
10. **Extension sidebar** shows notification:
    - "AI responded to [Contact Name]"
    - Shows the sent message
    - User can view full conversation

User sees: Message appears in their LinkedIn chat as if they typed it, but it was fully automated.

File Structure

PWA Mobile App

```
pwa-app/
├── public/
│   ├── manifest.json          # PWA manifest
│   ├── service-worker.js      # PWA service worker for offline
│   ├── icons/                 # App icons (72px to 512px)
│   └── index.html
├── src/
│   ├── components/
│   │   ├── LoginScreen.tsx
│   │   ├── RecordingScreen.tsx
│   │   ├── QRScanner.tsx
│   │   ├── RecordButton.tsx
│   │   ├── ContactDisplay.tsx
│   │   ├── ProcessingScreen.tsx
│   │   ├── SuccessScreen.tsx
│   │   └── AudioWaveform.tsx
│   ├── hooks/
│   │   ├── useAudioRecorder.ts  # MediaRecorder logic
│   │   ├── useQRScanner.ts      # QR scanning logic
│   │   ├── useSupabase.ts       # Supabase client wrapper
│   │   └── useAuth.ts           # Authentication state
│   ├── lib/
│   │   ├── supabase.ts          # Supabase client initialization
│   │   └── utils.ts             # Helper functions
│   ├── types/
│   │   └── index.ts             # TypeScript interfaces
│   ├── styles/
│   │   └── globals.css          # Tailwind + custom styles
│   ├── App.tsx                  # Root component with routing
│   ├── main.tsx                 # React entry point
│   └── vite-env.d.ts
├── .env.local                   # Environment variables
├── package.json
├── tsconfig.json
├── vite.config.ts
├── tailwind.config.js
└── README.md
```

Chrome Extension

```
chrome-extension/
├── public/
│   ├── manifest.json          # Chrome extension manifest v3
│   └── icons/                 # Extension icons (16px to 128px)
├── src/
│   ├── sidebar/               # React app for popup/sidebar
│   │   ├── components/
│   │   │   ├── ContactList.tsx
│   │   │   ├── ContactCard.tsx
│   │   │   ├── ContactDetail.tsx
│   │   │   ├── AutoPilotToggle.tsx
│   │   │   ├── MessageHistory.tsx
│   │   │   ├── AIActionLog.tsx
│   │   │   └── RelationshipSettings.tsx
│   │   ├── hooks/
│   │   │   ├── useContacts.ts
│   │   │   ├── useSupabase.ts
│   │   │   └── useRealtime.ts   # Realtime subscription logic
│   │   ├── App.tsx
│   │   ├── main.tsx
│   │   ├── index.html
│   │   └── styles.css
│   ├── content/               # Content script (runs on LinkedIn)
│   │   ├── linkedin.ts        # Main content script logic
│   │   └── selectors.ts       # LinkedIn DOM selectors
│   ├── background/            # Service worker
│   │   ├── service-worker.ts    # Background script logic
│   │   └── api.ts             # API call helpers
│   ├── lib/
│   │   ├── supabase.ts
│   │   └── utils.ts
│   └── types/
│       └── index.ts
├── .env.local
├── package.json
├── tsconfig.json
├── vite.config.ts             # Uses @crxjs/vite-plugin
├── tailwind.config.js
└── README.md
```

Supabase Edge Functions

```
supabase/
├── functions/
│   ├── process-audio/
│   │   ├── index.ts          # Main function handler
│   │   └── utils.ts          # Helper functions
│   ├── generate-response/
│   │   ├── index.ts          # Main function handler
│   │   └── prompts.ts        # GPT-4 prompt templates
│   ├── send-connection-request/
│   │   ├── index.ts          # Main function handler
│   │   └── templates.ts      # Message templates
│   └── _shared/              # Shared utilities
│       ├── supabase.ts       # Supabase client for functions
│       ├── openai.ts         # OpenAI client wrapper
│       └── types.ts          # Shared types
├── migrations/
│   └── 20250101000000_initial_schema.sql  # Database schema
└── config.toml               # Supabase config
```

Implementation Requirements

Environment Variables

PWA & Chrome Extension (.env.local):

```
env

VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key
```

Supabase Edge Functions (set via Supabase Dashboard):

```
env

OPENAI_API_KEY=sk-...
SUPABASE_SERVICE_ROLE_KEY=your-service-role-key
```

Key TypeScript Interfaces

```
typescript
```

```typescript
export interface Contact {
  id: string;
  user_id: string;
  name: string;
  linkedin_url: string;
  linkedin_profile_data: {
    title?: string;
    company?: string;
    location?: string;
  };
  relationship_type: 'potential_client' | 'professional_contact' | 'friend' | 'colleague' | 'other';
  custom_instructions: string | null;
  auto_pilot_enabled: boolean;
  status: 'not_contacted' | 'connection_sent' | 'connected' | 'declined';
  created_at: string;
  updated_at: string;
}

export interface Conversation {
  id: string;
  user_id: string;
  contact_id: string;
  audio_url: string;
  transcription: string | null;
  summary: string | null;
  key_topics: string[];
  ai_analysis: {
    sentiment?: string;
    action_items?: string[];
  };
  duration_seconds: number;
  recorded_at: string;
  processed: boolean;
}

export interface AIAction {
  id: string;
  user_id: string;
  contact_id: string;
  action_type: 'connection_request' | 'message_sent' | 'post_liked' | 'comment_posted';
  action_content: string | null;
  success: boolean;
  metadata: Record<string, any>;
  created_at: string;
}
```

```typescript
export interface LinkedInMessage {
  id: string;
  user_id: string;
  contact_id: string;
  sender: 'user' | 'contact';
  message_text: string;
  ai_generated: boolean;
  sent_at: string;
}
```

Build Configuration

PWA vite.config.ts:

```typescript
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import { VitePWA } from 'vite-plugin-pwa';

export default defineConfig({
  plugins: [
    react(),
    VitePWA({
      registerType: 'autoUpdate',
      manifest: {
        name: 'LinkedIn AI Assistant',
        short_name: 'LinkedInAI',
        theme_color: '#0A66C2'
      }
    })
  ]
});
```