Automated Daily/Weekly Email Digest System

AI-Powered Agent Activity Summaries

Overview

This feature enables agents to opt-in to automated email digests that summarize their pipeline activity, appointments, follow-ups, and action items. Claude AI generates personalized, motivational summaries that help agents start their day organized and focused.

User Experience

What Agents Receive

Daily Digest Example (6:00 AM):

```
Subject: 🎯 Your Tuesday Game Plan - 3 Appointments, 5 Follow-ups

Good morning, Sarah!

TODAY'S PRIORITY: You have 3 appointments scheduled and $47,500
in pipeline value ready to close. Let's make it happen!

═══════════════════════════════════════

📅 TODAY'S APPOINTMENTS

9:30 AM - Mario's Pizza (Presentation)
→ 123 Main St, Carmel IN
→ Decision maker: Mario Rossi
→ Hot lead - they requested a proposal last week!

2:00 PM - City Hardware (Statement Analysis)
→ 456 Oak Ave, Fishers IN
→ Bring: Competitor comparison, savings calculator
```

4:30 PM - Sunrise Bakery (Follow-up)
→ Phone call scheduled
→ Last touch: Sent proposal 5 days ago

📞 FOLLOW-UPS DUE TODAY (5)

🔥 HOT: Downtown Deli - Attempt 2 of 5
Last: Left voicemail Monday
Tip: Try calling before lunch rush (10-11 AM)

⚡ Quick Stop Market - Attempt 3 of 5
Last: Spoke Tuesday, needs to check with partner

⚡ Green Thumb Garden - Attempt 1 of 5
New lead from referral - first contact today!

📋 Tony's Auto Shop - Attempt 4 of 5
📋 Lakeside Cafe - Attempt 2 of 5

⚠️ NEEDS ATTENTION (3 Stale Deals)

• Joe's Diner - No activity in 12 days (Proposal Sent)
• Corner Store - No activity in 8 days (Negotiating)
• Fresh Foods - No activity in 14 days (Appointment Set)

Don't let these go cold! A quick check-in call can reignite the conversation.

📊 YOUR PIPELINE SNAPSHOT

Total Pipeline Value: $127,500
├─ Prospecting: 12 deals ($34,000)
├─ Active Selling: 8 deals ($52,500)
├─ Closing: 3 deals ($41,000) ← Focus here!
└─ This Week: 2 deals won ($18,500) 🎉

Win Rate (30 days): 34% (↑ 5% from last month)

💡 TODAY'S TIP

```
"The fortune is in the follow-up. 80% of sales require
5+ touches, but most reps give up after 2. You're on
attempt 3 with Quick Stop Market - stay persistent!"

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Have a great day! 💪

View your full pipeline: [Open BrochureTracker]
Manage email preferences: [Update Settings]
```

Feature Requirements

1. User Preferences (Profile Settings)

Agents can configure in their Profile:

Email Digest Settings:

- **Enable/Disable**: Toggle digest on/off
- **Frequency**: Daily, Weekly (Monday), or Both
- **Delivery Time**: Preferred time (default 6:00 AM local)
- **Email Address**: Can differ from login email
- **Timezone**: Auto-detected or manual selection

Content Preferences (checkboxes):

- Today’s appointments
- Follow-ups due
- Stale deals needing attention
- Pipeline summary & stats
- Recent wins celebration
- AI-generated tips & motivation
- Quarterly check-ins due (for active merchants)
- New referrals received

Additional Options:

- Include deals from: Today only / Next 3 days / Full week
- Stale deal threshold: 7 days / 14 days / Custom
- Weekly digest day: Monday / Sunday

Technical Implementation

2. Database Schema

```
-- Email digest preferences table
CREATE TABLE email_digest_preferences (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID UNIQUE REFERENCES users(id) ON DELETE CASCADE,
  organization_id UUID REFERENCES organizations(id),

  -- Enable/Disable
  daily_digest_enabled BOOLEAN DEFAULT FALSE,
  weekly_digest_enabled BOOLEAN DEFAULT FALSE,

  -- Delivery Settings
  email_address VARCHAR(255) NOT NULL,
  timezone VARCHAR(50) DEFAULT 'America/New_York',
  daily_send_time TIME DEFAULT '06:00:00',
  weekly_send_day VARCHAR(10) DEFAULT 'monday',  -- monday, sunday
  weekly_send_time TIME DEFAULT '06:00:00',

  -- Content Preferences (what to include)
  include_appointments BOOLEAN DEFAULT TRUE,
  include_followups BOOLEAN DEFAULT TRUE,
  include_stale_deals BOOLEAN DEFAULT TRUE,
  include_pipeline_summary BOOLEAN DEFAULT TRUE,
  include_recent_wins BOOLEAN DEFAULT TRUE,
  include_ai_tips BOOLEAN DEFAULT TRUE,
  include_quarterly_checkins BOOLEAN DEFAULT TRUE,
  include_new_referrals BOOLEAN DEFAULT TRUE,

  -- Additional Settings
  appointment_lookahead_days INTEGER DEFAULT 1,  -- 1 = today only, 3 = next 3 da
  stale_deal_threshold_days INTEGER DEFAULT 7,

  -- Tracking
```

```
  last_daily_sent_at TIMESTAMP,
  last_weekly_sent_at TIMESTAMP,
  total_emails_sent INTEGER DEFAULT 0,

  -- Timestamps
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Index for efficient querying during cron job
CREATE INDEX idx_digest_daily_enabled
  ON email_digest_preferences(daily_digest_enabled, daily_send_time)
  WHERE daily_digest_enabled = TRUE;

CREATE INDEX idx_digest_weekly_enabled
  ON email_digest_preferences(weekly_digest_enabled, weekly_send_day)
  WHERE weekly_digest_enabled = TRUE;

-- Email send history for debugging/analytics
CREATE TABLE email_digest_history (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id),
  digest_type VARCHAR(10) NOT NULL,  -- 'daily' or 'weekly'

  -- Content tracking
  appointments_count INTEGER,
  followups_count INTEGER,
  stale_deals_count INTEGER,
  pipeline_value DECIMAL(12,2),

  -- Delivery status
  status VARCHAR(20) DEFAULT 'pending',  -- pending, sent, failed
  sent_at TIMESTAMP,
  error_message TEXT,

  -- Email metadata
  subject_line TEXT,
  email_provider_id VARCHAR(100),  -- ID from SendGrid/Resend

  created_at TIMESTAMP DEFAULT NOW()
);
```

3. API Endpoints

```
// Get current preferences
GET /api/user/email-digest-preferences
```

```
Response:
{
  "enabled": true,
  "dailyDigestEnabled": true,
  "weeklyDigestEnabled": false,
  "emailAddress": "sarah@email.com",
  "timezone": "America/Indiana/Indianapolis",
  "dailySendTime": "06:00",
  "weeklySendDay": "monday",
  "weeklySendTime": "07:00",
  "includeAppointments": true,
  "includeFollowups": true,
  "includeStaleDeal": true,
  "includePipelineSummary": true,
  "includeRecentWins": true,
  "includeAiTips": true,
  "includeQuarterlyCheckins": true,
  "includeNewReferrals": true,
  "appointmentLookaheadDays": 1,
  "staleDealThresholdDays": 7
}

// Update preferences
PUT /api/user/email-digest-preferences

Request:
{
  "dailyDigestEnabled": true,
  "emailAddress": "sarah.new@email.com",
  "dailySendTime": "07:00",
  "includeAiTips": false,
  // ... any fields to update
}

Response:
{
  "success": true,
  "message": "Preferences updated. Your next digest will be sent tomorrow at 7:00
}

// Send test digest (for preview)
POST /api/user/email-digest-preferences/test

Response:
{
  "success": true,
```

```
  "message": "Test digest sent to sarah@email.com"
}

// Get digest history
GET /api/user/email-digest-history?limit=10

Response:
{
  "history": [
    {
      "id": "...",
      "digestType": "daily",
      "sentAt": "2025-01-31T06:00:00Z",
      "status": "sent",
      "appointmentsCount": 3,
      "followupsCount": 5,
      "subjectLine": "🎯 Your Tuesday Game Plan - 3 Appointments, 5 Follow-ups"
    }
  ]
}
```

4. Cron Job / Scheduled Task

The digest sender runs every 15 minutes and checks which users need emails sent based on their timezone and preferred send time.

```
// lib/email-digest/scheduler.ts

import { db } from '@/db';
import { emailDigestPreferences, users } from '@/db/schema';
import { eq, and, lte, or, isNull } from 'drizzle-orm';
import { generateDigestWithClaude } from './ai-generator';
import { sendEmail } from './email-sender';
import { gatherUserDigestData } from './data-gatherer';

/**
 * Main scheduler function - runs every 15 minutes via cron
 * Checks all timezones and sends digests to users whose send time has passed
 */
export async function processEmailDigests() {
  const now = new Date();

  // Process daily digests
  await processDailyDigests(now);

  // Process weekly digests (check if today is the right day)
```

```
  await processWeeklyDigests(now);
}

async function processDailyDigests(now: Date) {
  // Get all users with daily digest enabled
  const eligibleUsers = await db.query.emailDigestPreferences.findMany({
    where: eq(emailDigestPreferences.dailyDigestEnabled, true),
    with: {
      user: true
    }
  });

  for (const prefs of eligibleUsers) {
    try {
      // Check if it's time to send based on user's timezone
      const userLocalTime = getLocalTime(now, prefs.timezone);
      const sendTime = parseTime(prefs.dailySendTime);

      // Skip if not yet time to send
      if (userLocalTime.hours < sendTime.hours ||
          (userLocalTime.hours === sendTime.hours && userLocalTime.minutes < send
        continue;
      }

      // Skip if already sent today
      if (prefs.lastDailySentAt) {
        const lastSentLocal = getLocalTime(prefs.lastDailySentAt, prefs.timezone)
        if (isSameDay(userLocalTime, lastSentLocal)) {
          continue;
        }
      }

      // Send the digest
      await sendDailyDigest(prefs);

    } catch (error) {
      console.error(`Failed to process daily digest for user ${prefs.userId}:`, e

      // Log failure
      await db.insert(emailDigestHistory).values({
        userId: prefs.userId,
        digestType: 'daily',
        status: 'failed',
        errorMessage: error.message
      });
    }
  }
```

```
}

async function processWeeklyDigests(now: Date) {
  const eligibleUsers = await db.query.emailDigestPreferences.findMany({
    where: eq(emailDigestPreferences.weeklyDigestEnabled, true),
    with: {
      user: true
    }
  });

  for (const prefs of eligibleUsers) {
    try {
      const userLocalTime = getLocalTime(now, prefs.timezone);
      const todayName = getDayName(userLocalTime).toLowerCase();

      // Skip if not the right day
      if (todayName !== prefs.weeklySendDay) {
        continue;
      }

      const sendTime = parseTime(prefs.weeklySendTime);

      // Skip if not yet time
      if (userLocalTime.hours < sendTime.hours ||
          (userLocalTime.hours === sendTime.hours && userLocalTime.minutes < send
        continue;
      }

      // Skip if already sent this week
      if (prefs.lastWeeklySentAt) {
        const daysSinceLastSent = daysBetween(prefs.lastWeeklySentAt, now);
        if (daysSinceLastSent < 6) {
          continue;
        }
      }

      // Send the digest
      await sendWeeklyDigest(prefs);

    } catch (error) {
      console.error(`Failed to process weekly digest for user ${prefs.userId}:`,

      await db.insert(emailDigestHistory).values({
        userId: prefs.userId,
        digestType: 'weekly',
        status: 'failed',
        errorMessage: error.message
```

```
      });
    }
  }
}

async function sendDailyDigest(prefs: EmailDigestPreferences) {
  // Gather all the data for this user
  const data = await gatherUserDigestData(prefs.userId, {
    type: 'daily',
    appointmentLookaheadDays: prefs.appointmentLookaheadDays,
    staleDealThresholdDays: prefs.staleDealThresholdDays,
    includeAppointments: prefs.includeAppointments,
    includeFollowups: prefs.includeFollowups,
    includeStaleDeal: prefs.includeStaleDeal,
    includePipelineSummary: prefs.includePipelineSummary,
    includeRecentWins: prefs.includeRecentWins,
    includeAiTips: prefs.includeAiTips,
    includeQuarterlyCheckins: prefs.includeQuarterlyCheckins,
    includeNewReferrals: prefs.includeNewReferrals
  });

  // Generate personalized email with Claude
  const emailContent = await generateDigestWithClaude(data, 'daily');

  // Send email
  const result = await sendEmail({
    to: prefs.emailAddress,
    subject: emailContent.subject,
    html: emailContent.html,
    text: emailContent.text
  });

  // Update tracking
  await db.update(emailDigestPreferences)
    .set({
      lastDailySentAt: new Date(),
      totalEmailsSent: prefs.totalEmailsSent + 1
    })
    .where(eq(emailDigestPreferences.userId, prefs.userId));

  // Log success
  await db.insert(emailDigestHistory).values({
    userId: prefs.userId,
    digestType: 'daily',
    status: 'sent',
    sentAt: new Date(),
    appointmentsCount: data.appointments.length,
```

```
    followupsCount: data.followups.length,
    staleDealcount: data.staleDeals.length,
    pipelineValue: data.pipelineSummary.totalValue,
    subjectLine: emailContent.subject,
    emailProviderId: result.id
  });
}

// Similar implementation for sendWeeklyDigest...
```

5. Data Gatherer

```
// lib/email-digest/data-gatherer.ts

import { db } from '@/db';
import { deals, dealActivities, referrals } from '@/db/schema';
import { eq, and, gte, lte, isNull, or } from 'drizzle-orm';

interface DigestData {
  user: {
    firstName: string;
    lastName: string;
    email: string;
  };
  appointments: Appointment[];
  followups: Followup[];
  staleDeals: Deal[];
  pipelineSummary: PipelineSummary;
  recentWins: Deal[];
  quarterlyCheckins: Merchant[];
  newReferrals: Referral[];
  digestType: 'daily' | 'weekly';
  generatedAt: Date;
}

export async function gatherUserDigestData(
  userId: string,
  options: DigestOptions
): Promise<DigestData> {
  const today = startOfDay(new Date());
  const endDate = addDays(today, options.appointmentLookaheadDays);

  // Get user info
  const user = await db.query.users.findFirst({
    where: eq(users.id, userId)
  });
```

```
// Get appointments
let appointments = [];
if (options.includeAppointments) {
  appointments = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      eq(deals.currentStage, 'appointment_set'),
      gte(deals.appointmentDate, today),
      lte(deals.appointmentDate, endDate)
    ),
    orderBy: [asc(deals.appointmentDate)]
  });
}

// Get follow-ups due
let followups = [];
if (options.includeFollowups) {
  followups = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      or(
        eq(deals.currentStage, 'follow_up'),
        and(
          lte(deals.nextFollowUpAt, endDate),
          not(eq(deals.currentStage, 'sold')),
          not(eq(deals.currentStage, 'dead')),
          not(eq(deals.currentStage, 'active_merchant'))
        )
      )
    ),
    orderBy: [asc(deals.nextFollowUpAt)]
  });
}

// Get stale deals
let staleDeals = [];
if (options.includeStaleDeal) {
  const staleThreshold = subDays(today, options.staleDealThresholdDays);
  staleDeals = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      lte(deals.lastActivityAt, staleThreshold),
      not(eq(deals.currentStage, 'sold')),
      not(eq(deals.currentStage, 'dead')),
      not(eq(deals.currentStage, 'active_merchant'))
    ),
```

```
    orderBy: [asc(deals.lastActivityAt)],
    limit: 10
  });
}

// Get pipeline summary
let pipelineSummary = null;
if (options.includePipelineSummary) {
  pipelineSummary = await calculatePipelineSummary(userId);
}

// Get recent wins (last 7 days)
let recentWins = [];
if (options.includeRecentWins) {
  const weekAgo = subDays(today, 7);
  recentWins = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      eq(deals.currentStage, 'sold'),
      gte(deals.wonAt, weekAgo)
    ),
    orderBy: [desc(deals.wonAt)],
    limit: 5
  });
}

// Get quarterly check-ins due
let quarterlyCheckins = [];
if (options.includeQuarterlyCheckins) {
  quarterlyCheckins = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      eq(deals.currentStage, 'active_merchant'),
      lte(deals.nextQuarterlyCheckinAt, endDate)
    ),
    orderBy: [asc(deals.nextQuarterlyCheckinAt)],
    limit: 5
  });
}

// Get new referrals (last 24 hours for daily, last 7 days for weekly)
let newReferrals = [];
if (options.includeNewReferrals) {
  const lookback = options.type === 'daily' ? subDays(today, 1) : subDays(today
  newReferrals = await db.query.referrals.findMany({
    where: and(
      eq(referrals.referredByAgentId, userId),
```

```
        gte(referrals.createdAt, lookback)
      ),
      orderBy: [desc(referrals.createdAt)]
    });
  }

  return {
    user: {
      firstName: user.firstName,
      lastName: user.lastName,
      email: user.email
    },
    appointments,
    followups,
    staleDeals,
    pipelineSummary,
    recentWins,
    quarterlyCheckins,
    newReferrals,
    digestType: options.type,
    generatedAt: new Date()
  };
}

async function calculatePipelineSummary(userId: string): Promise<PipelineSummary>
  // Get all active deals
  const activeDeals = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      not(eq(deals.currentStage, 'sold')),
      not(eq(deals.currentStage, 'dead'))
    )
  });

  // Group by stage phase
  const prospecting = activeDeals.filter(d =>
    ['prospect', 'cold_call', 'appointment_set'].includes(d.currentStage)
  );
  const activeSelling = activeDeals.filter(d =>
    ['presentation_made', 'proposal_sent', 'statement_analysis', 'negotiating', '
  );
  const closing = activeDeals.filter(d =>
    ['documents_sent', 'documents_signed'].includes(d.currentStage)
  );

  // Calculate values
  const sumValue = (deals: Deal[]) => deals.reduce((sum, d) => sum + (d.estimated
```

```
  // Get wins this week
  const weekAgo = subDays(new Date(), 7);
  const winsThisWeek = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      eq(deals.currentStage, 'sold'),
      gte(deals.wonAt, weekAgo)
    )
  });

  // Calculate win rate (last 30 days)
  const monthAgo = subDays(new Date(), 30);
  const closedDeals = await db.query.deals.findMany({
    where: and(
      eq(deals.assignedAgentId, userId),
      or(eq(deals.currentStage, 'sold'), eq(deals.currentStage, 'dead')),
      gte(deals.closedAt, monthAgo)
    )
  });
  const wins = closedDeals.filter(d => d.currentStage === 'sold').length;
  const winRate = closedDeals.length > 0 ? Math.round((wins / closedDeals.length)

  return {
    totalValue: sumValue(activeDeals),
    prospecting: { count: prospecting.length, value: sumValue(prospecting) },
    activeSelling: { count: activeSelling.length, value: sumValue(activeSelling)
    closing: { count: closing.length, value: sumValue(closing) },
    winsThisWeek: { count: winsThisWeek.length, value: sumValue(winsThisWeek) },
    winRate,
    totalDeals: activeDeals.length
  };
}
```

6. Claude AI Email Generator

```
// lib/email-digest/ai-generator.ts

import Anthropic from '@anthropic-ai/sdk';

const anthropic = new Anthropic({
  apiKey: process.env.ANTHROPIC_API_KEY
});

interface EmailContent {
  subject: string;
```

```
  html: string;
  text: string;
}

export async function generateDigestWithClaude(
  data: DigestData,
  type: 'daily' | 'weekly'
): Promise<EmailContent> {

  const prompt = buildPrompt(data, type);

  const response = await anthropic.messages.create({
    model: 'claude-sonnet-4-20250514',
    max_tokens: 4000,
    messages: [
      {
        role: 'user',
        content: prompt
      }
    ]
  });

  const content = response.content[0].text;

  // Parse Claude's response (it will return JSON with subject, body sections)
  const parsed = JSON.parse(content);

  // Build HTML email
  const html = buildHtmlEmail(parsed, data);
  const text = buildPlainTextEmail(parsed, data);

  return {
    subject: parsed.subject,
    html,
    text
  };
}

function buildPrompt(data: DigestData, type: 'daily' | 'weekly'): string {
  const dayOfWeek = new Intl.DateTimeFormat('en-US', { weekday: 'long' }).format(

  return `You are writing a ${type} email digest for a field sales representative
Your goal is to help them start their ${type === 'daily' ? 'day' : 'week'} organi

Write in a warm, encouraging, but professional tone. Be concise and actionable. U

Here is their data for ${type === 'daily' ? 'today' : 'this week'}:
```

```
${data.appointments.length > 0 ? `
APPOINTMENTS (${data.appointments.length}):
${data.appointments.map(apt => `
- ${apt.businessName}
  Time: ${formatTime(apt.appointmentDate)}
  Address: ${apt.businessAddress || 'No address'}
  Contact: ${apt.contactName || 'Unknown'}
  Stage: ${apt.currentStage}
  Notes: ${apt.notes || 'None'}
  Temperature: ${apt.temperature}
`).join('\n')}
` : 'No appointments scheduled.'}

${data.followups.length > 0 ? `
FOLLOW-UPS DUE (${data.followups.length}):
${data.followups.map(fu => `
- ${fu.businessName}
  Follow-up attempt: ${fu.followUpAttemptCount} of ${fu.maxFollowUpAttempts}
  Last contact: ${fu.lastFollowUpAt ? formatDate(fu.lastFollowUpAt) : 'Never'}
  Last outcome: ${fu.lastFollowUpOutcome || 'N/A'}
  Temperature: ${fu.temperature}
`).join('\n')}
` : 'No follow-ups due.'}

${data.staleDeals.length > 0 ? `
STALE DEALS NEEDING ATTENTION (${data.staleDeals.length}):
${data.staleDeals.map(deal => `
- ${deal.businessName}
  Days since activity: ${daysSince(deal.lastActivityAt)}
  Stage: ${deal.currentStage}
  Value: $${deal.estimatedMonthlyVolume?.toLocaleString() || '0'}
`).join('\n')}
` : 'No stale deals - great job staying on top of your pipeline!'}

${data.pipelineSummary ? `
PIPELINE SUMMARY:
- Total Pipeline Value: $${data.pipelineSummary.totalValue.toLocaleString()}
- Prospecting: ${data.pipelineSummary.prospecting.count} deals ($${data.pipelineS
- Active Selling: ${data.pipelineSummary.activeSelling.count} deals ($${data.pipe
- Closing: ${data.pipelineSummary.closing.count} deals ($${data.pipelineSummary.c
- Wins This Week: ${data.pipelineSummary.winsThisWeek.count} deals ($${data.pipel
- 30-Day Win Rate: ${data.pipelineSummary.winRate}%
` : ''}

${data.recentWins.length > 0 ? `
RECENT WINS 🎉:
```

```
${data.recentWins.map(win => `- ${win.businessName} ($${win.estimatedMonthlyVolum
` : ''}

${data.quarterlyCheckins.length > 0 ? `
QUARTERLY CHECK-INS DUE (${data.quarterlyCheckins.length}):
${data.quarterlyCheckins.map(m => `- ${m.businessName} (last check-in: ${formatDa
` : ''}

${data.newReferrals.length > 0 ? `
NEW REFERRALS RECEIVED (${data.newReferrals.length}):
${data.newReferrals.map(ref => `- ${ref.businessName} (referred by ${ref.referrer
` : ''}

Based on this data, generate a ${type} email digest. Return your response as JSON

{
  "subject": "An engaging subject line with emoji, mentioning key numbers (appoin
  "greeting": "Personalized greeting using their first name",
  "priorityStatement": "One sentence highlighting the biggest opportunity or prio
  "appointmentsSummary": "Brief summary of appointments with key details and any
  "followupsSummary": "Brief summary of follow-ups, highlighting hot leads and su
  "staleDealsSummary": "Brief summary of stale deals with encouragement to re-eng
  "pipelineSummary": "One or two sentences about their pipeline health, celebrati
  "motivationalTip": "A helpful sales tip relevant to their current situation, or
  "signoff": "A brief, encouraging sign-off"
}

Important guidelines:
- Be specific - reference actual business names and numbers from the data
- Be encouraging but not cheesy
- Prioritize actionable items
- If they have deals in the "Closing" stage, emphasize those as highest priority
- If they have hot leads, call those out specifically
- Keep the total email readable in under 2 minutes
- Today is ${dayOfWeek}`;
}

function buildHtmlEmail(parsed: any, data: DigestData): string {
  return `
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>${parsed.subject}</title>
  <style>
    body {
```

```
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Helveti
  line-height: 1.6;
  color: #333;
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
  background-color: #f5f5f5;
}
.container {
  background: white;
  border-radius: 12px;
  padding: 30px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
}
.header {
  text-align: center;
  margin-bottom: 30px;
}
.logo {
  font-size: 24px;
  font-weight: bold;
  color: #6366f1;
}
.greeting {
  font-size: 20px;
  margin-bottom: 10px;
}
.priority {
  background: linear-gradient(135deg, #6366f1 0%, #8b5cf6 100%);
  color: white;
  padding: 15px 20px;
  border-radius: 8px;
  margin-bottom: 25px;
  font-weight: 500;
}
.section {
  margin-bottom: 25px;
  padding-bottom: 20px;
  border-bottom: 1px solid #eee;
}
.section:last-of-type {
  border-bottom: none;
}
.section-title {
  font-size: 14px;
  font-weight: 600;
  color: #6366f1;
```

```
  text-transform: uppercase;
  letter-spacing: 0.5px;
  margin-bottom: 12px;
}
.appointment-card, .deal-card {
  background: #f8f9fa;
  border-radius: 8px;
  padding: 12px 15px;
  margin-bottom: 10px;
}
.appointment-time {
  font-weight: 600;
  color: #6366f1;
}
.business-name {
  font-weight: 600;
  font-size: 16px;
}
.detail {
  color: #666;
  font-size: 14px;
}
.hot-badge {
  background: #ef4444;
  color: white;
  padding: 2px 8px;
  border-radius: 12px;
  font-size: 11px;
  font-weight: 600;
}
.warm-badge {
  background: #f59e0b;
  color: white;
  padding: 2px 8px;
  border-radius: 12px;
  font-size: 11px;
  font-weight: 600;
}
.pipeline-stats {
  display: flex;
  flex-wrap: wrap;
  gap: 15px;
}
.stat {
  flex: 1;
  min-width: 120px;
  text-align: center;
```

```
  padding: 15px;
  background: #f8f9fa;
  border-radius: 8px;
}
.stat-value {
  font-size: 24px;
  font-weight: bold;
  color: #6366f1;
}
.stat-label {
  font-size: 12px;
  color: #666;
}
.tip-box {
  background: #fef3c7;
  border-left: 4px solid #f59e0b;
  padding: 15px;
  border-radius: 0 8px 8px 0;
  margin-top: 20px;
}
.tip-title {
  font-weight: 600;
  color: #92400e;
  margin-bottom: 5px;
}
.cta-button {
  display: inline-block;
  background: #6366f1;
  color: white;
  padding: 12px 24px;
  border-radius: 8px;
  text-decoration: none;
  font-weight: 500;
  margin-top: 20px;
}
.footer {
  text-align: center;
  margin-top: 30px;
  padding-top: 20px;
  border-top: 1px solid #eee;
  color: #666;
  font-size: 13px;
}
.footer a {
  color: #6366f1;
}
.win-celebration {
```

```
      background: linear-gradient(135deg, #10b981 0%, #059669 100%);
      color: white;
      padding: 15px;
      border-radius: 8px;
      text-align: center;
      margin-bottom: 20px;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <div class="logo">📊 BrochureTracker</div>
    </div>

    <div class="greeting">${parsed.greeting}</div>

    <div class="priority">${parsed.priorityStatement}</div>

    ${data.recentWins.length > 0 ? `
    <div class="win-celebration">
      🎉 ${data.recentWins.length} deal${data.recentWins.length > 1 ? 's' : ''} w
      ($${data.pipelineSummary?.winsThisWeek.value.toLocaleString()})
    </div>
    ` : ''}

    ${parsed.appointmentsSummary ? `
    <div class="section">
      <div class="section-title">📅 Today's Appointments</div>
      <p>${parsed.appointmentsSummary}</p>
      ${data.appointments.map(apt => `
        <div class="appointment-card">
          <div class="appointment-time">${formatTime(apt.appointmentDate)}</div>
          <div class="business-name">${apt.businessName}</div>
          <div class="detail">📍 ${apt.businessAddress || 'No address'}</div>
          <div class="detail">👤 ${apt.contactName || 'Unknown contact'}</div>
          ${apt.temperature === 'hot' ? '<span class="hot-badge">🔥 HOT</span>' :
        </div>
      `).join('')}
    </div>
    ` : ''}

    ${parsed.followupsSummary ? `
    <div class="section">
      <div class="section-title">📞 Follow-ups Due (${data.followups.length})</di
      <p>${parsed.followupsSummary}</p>
      ${data.followups.slice(0, 5).map(fu => `
```

```
    <div class="deal-card">
      <div class="business-name">
        ${fu.businessName}
        ${fu.temperature === 'hot' ? '<span class="hot-badge">🔥 HOT</span>'
          fu.temperature === 'warm' ? '<span class="warm-badge">⚡ WARM</span
      </div>
      <div class="detail">Attempt ${fu.followUpAttemptCount} of ${fu.maxFollo
      ${fu.lastFollowUpOutcome ? `<div class="detail">Last: ${fu.lastFollowUp
    </div>
  `).join('')}
  ${data.followups.length > 5 ? `<p class="detail">+ ${data.followups.length
</div>
` : ''}

${parsed.staleDealsSummary ? `
<div class="section">
  <div class="section-title">⚠️ Needs Attention</div>
  <p>${parsed.staleDealsSummary}</p>
  ${data.staleDeals.slice(0, 3).map(deal => `
    <div class="deal-card">
      <div class="business-name">${deal.businessName}</div>
      <div class="detail">${daysSince(deal.lastActivityAt)} days since last a
    </div>
  `).join('')}
</div>
` : ''}

${data.pipelineSummary ? `
<div class="section">
  <div class="section-title">📊 Pipeline Snapshot</div>
  <p>${parsed.pipelineSummary}</p>
  <div class="pipeline-stats">
    <div class="stat">
      <div class="stat-value">$${(data.pipelineSummary.totalValue / 1000).toF
      <div class="stat-label">Total Pipeline</div>
    </div>
    <div class="stat">
      <div class="stat-value">${data.pipelineSummary.closing.count}</div>
      <div class="stat-label">Ready to Close</div>
    </div>
    <div class="stat">
      <div class="stat-value">${data.pipelineSummary.winRate}%</div>
      <div class="stat-label">Win Rate</div>
    </div>
  </div>
</div>
` : ''}
```

```
    ${parsed.motivationalTip ? `
    <div class="tip-box">
      <div class="tip-title">💡 Today's Tip</div>
      <div>${parsed.motivationalTip}</div>
    </div>
    ` : ''}

    <div style="text-align: center;">
      <a href="${process.env.APP_URL}/deals" class="cta-button">Open My Pipeline<
    </div>

    <div class="footer">
      <p>${parsed.signoff}</p>
      <p>
        <a href="${process.env.APP_URL}/profile/email-preferences">Manage email p
        <a href="${process.env.APP_URL}">Open BrochureTracker</a>
      </p>
    </div>
  </div>
</body>
</html>
  `;
}

function buildPlainTextEmail(parsed: any, data: DigestData): string {
  // Build plain text version for email clients that don't support HTML
  let text = `${parsed.greeting}\n\n`;
  text += `${parsed.priorityStatement}\n\n`;
  text += `━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n\n`;

  if (parsed.appointmentsSummary) {
    text += `📅 TODAY'S APPOINTMENTS\n\n`;
    text += `${parsed.appointmentsSummary}\n\n`;
    data.appointments.forEach(apt => {
      text += `• ${formatTime(apt.appointmentDate)} - ${apt.businessName}\n`;
      text += `  ${apt.businessAddress || 'No address'}\n`;
      text += `  Contact: ${apt.contactName || 'Unknown'}\n\n`;
    });
    text += `━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n\n`;
  }

  if (parsed.followupsSummary) {
    text += `📞 FOLLOW-UPS DUE (${data.followups.length})\n\n`;
    text += `${parsed.followupsSummary}\n\n`;
    data.followups.slice(0, 5).forEach(fu => {
      text += `• ${fu.businessName} - Attempt ${fu.followUpAttemptCount}/${fu.max
```

```
    });
    text += `\n━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n\n`;
  }

  if (parsed.staleDealsSummary) {
    text += `⚠️ NEEDS ATTENTION\n\n`;
    text += `${parsed.staleDealsSummary}\n\n`;
    text += `━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n\n`;
  }

  if (parsed.pipelineSummary) {
    text += `📊 PIPELINE SNAPSHOT\n\n`;
    text += `${parsed.pipelineSummary}\n\n`;
    text += `Total Value: $${data.pipelineSummary.totalValue.toLocaleString()}\n`
    text += `Win Rate: ${data.pipelineSummary.winRate}%\n\n`;
    text += `━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n\n`;
  }

  if (parsed.motivationalTip) {
    text += `💡 TODAY'S TIP\n\n`;
    text += `${parsed.motivationalTip}\n\n`;
    text += `━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n\n`;
  }

  text += `${parsed.signoff}\n\n`;
  text += `View your pipeline: ${process.env.APP_URL}/deals\n`;
  text += `Manage preferences: ${process.env.APP_URL}/profile/email-preferences\n

  return text;
}
```

7. Email Sender (using Resend)

```
// lib/email-digest/email-sender.ts

import { Resend } from 'resend';

const resend = new Resend(process.env.RESEND_API_KEY);

interface SendEmailParams {
  to: string;
  subject: string;
  html: string;
  text: string;
}
```

```
export async function sendEmail(params: SendEmailParams) {
  const { to, subject, html, text } = params;

  const result = await resend.emails.send({
    from: 'BrochureTracker <digest@brochuretracker.com>',
    to: [to],
    subject: subject,
    html: html,
    text: text,
    headers: {
      'X-Entity-Ref-ID': `digest-${Date.now()}`,  // Prevent threading in Gmail
    }
  });

  return result;
}
```

8. Cron Job Setup

For Replit, you can use a simple interval-based approach or integrate with a cron service:

```
// server/cron.ts

import { processEmailDigests } from '@/lib/email-digest/scheduler';

// Run every 15 minutes
const DIGEST_INTERVAL = 15 * 60 * 1000; // 15 minutes in milliseconds

export function startCronJobs() {
  // Initial run on server start (after a short delay)
  setTimeout(() => {
    processEmailDigests().catch(console.error);
  }, 10000); // 10 second delay after server start

  // Then run every 15 minutes
  setInterval(() => {
    processEmailDigests().catch(console.error);
  }, DIGEST_INTERVAL);

  console.log('Email digest cron job started - running every 15 minutes');
}

// Alternative: Use node-cron for more precise scheduling
import cron from 'node-cron';

export function startCronJobsWithNodeCron() {
```

```
  // Run at minute 0, 15, 30, 45 of every hour
  cron.schedule('0,15,30,45 * * * *', () => {
    console.log('Running email digest processor...');
    processEmailDigests().catch(console.error);
  });

  console.log('Email digest cron job scheduled');
}
```

Call this when your server starts:

```
// server/index.ts

import { startCronJobs } from './cron';

// ... other server setup ...

// Start cron jobs
startCronJobs();
```

9. Frontend Settings UI

```
// components/EmailDigestSettings.tsx

import { useState } from 'react';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import { Switch } from '@/components/ui/switch';
import { Input } from '@/components/ui/input';
import { Select } from '@/components/ui/select';
import { Button } from '@/components/ui/button';
import { Card, CardHeader, CardTitle, CardContent } from '@/components/ui/card';
import { Label } from '@/components/ui/label';
import { Checkbox } from '@/components/ui/checkbox';
import { toast } from '@/components/ui/use-toast';
import { Mail, Clock, Calendar, Send } from 'lucide-react';

export function EmailDigestSettings() {
  const queryClient = useQueryClient();

  // Fetch current preferences
  const { data: prefs, isLoading } = useQuery({
    queryKey: ['email-digest-preferences'],
    queryFn: async () => {
      const res = await fetch('/api/user/email-digest-preferences');
      return res.json();
```

```
  }
});

// Update preferences
const updatePrefs = useMutation({
  mutationFn: async (updates: any) => {
    const res = await fetch('/api/user/email-digest-preferences', {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(updates)
    });
    return res.json();
  },
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ['email-digest-preferences'] });
    toast({ title: 'Preferences saved', description: 'Your email digest setting
  }
});

// Send test email
const sendTest = useMutation({
  mutationFn: async () => {
    const res = await fetch('/api/user/email-digest-preferences/test', { method
    return res.json();
  },
  onSuccess: (data) => {
    toast({ title: 'Test email sent', description: `Check ${prefs?.emailAddress
  }
});

const [formState, setFormState] = useState<any>(null);

// Initialize form state when prefs load
useEffect(() => {
  if (prefs && !formState) {
    setFormState(prefs);
  }
}, [prefs]);

if (isLoading || !formState) {
  return <LoadingSpinner />;
}

const handleChange = (field: string, value: any) => {
  setFormState((prev: any) => ({ ...prev, [field]: value }));
};
```

```
const handleSave = () => {
  updatePrefs.mutate(formState);
};

const timezones = [
  { value: 'America/New_York', label: 'Eastern Time (ET)' },
  { value: 'America/Chicago', label: 'Central Time (CT)' },
  { value: 'America/Denver', label: 'Mountain Time (MT)' },
  { value: 'America/Los_Angeles', label: 'Pacific Time (PT)' },
  { value: 'America/Indiana/Indianapolis', label: 'Indiana (ET)' },
  // Add more as needed
];

const timeOptions = Array.from({ length: 24 }, (_, i) => {
  const hour = i;
  const ampm = hour >= 12 ? 'PM' : 'AM';
  const displayHour = hour === 0 ? 12 : hour > 12 ? hour - 12 : hour;
  return {
    value: `${hour.toString().padStart(2, '0')}:00`,
    label: `${displayHour}:00 ${ampm}`
  };
});

return (
  <div className="email-digest-settings space-y-6">
    <Card>
      <CardHeader>
        <CardTitle className="flex items-center gap-2">
          <Mail className="h-5 w-5" />
          Email Digest Settings
        </CardTitle>
      </CardHeader>
      <CardContent className="space-y-6">

        {/* Email Address */}
        <div className="space-y-2">
          <Label htmlFor="email">Delivery Email Address</Label>
          <Input
            id="email"
            type="email"
            value={formState.emailAddress}
            onChange={(e) => handleChange('emailAddress', e.target.value)}
            placeholder="your@email.com"
          />
          <p className="text-sm text-muted-foreground">
            Digests will be sent to this address
          </p>
```

```
</div>

{/* Timezone */}
<div className="space-y-2">
  <Label>Your Timezone</Label>
  <Select
    value={formState.timezone}
    onValueChange={(value) => handleChange('timezone', value)}
  >
    {timezones.map(tz => (
      <SelectItem key={tz.value} value={tz.value}>
        {tz.label}
      </SelectItem>
    ))}
  </Select>
</div>

<div className="border-t pt-6">
  <h3 className="font-semibold mb-4 flex items-center gap-2">
    <Calendar className="h-4 w-4" />
    Daily Digest
  </h3>

  <div className="flex items-center justify-between mb-4">
    <div>
      <Label>Enable Daily Digest</Label>
      <p className="text-sm text-muted-foreground">
        Receive a summary every morning
      </p>
    </div>
    <Switch
      checked={formState.dailyDigestEnabled}
      onCheckedChange={(checked) => handleChange('dailyDigestEnabled',
    />
  </div>

  {formState.dailyDigestEnabled && (
    <div className="ml-4 space-y-4 border-l-2 pl-4">
      <div className="space-y-2">
        <Label>Send Time</Label>
        <Select
          value={formState.dailySendTime}
          onValueChange={(value) => handleChange('dailySendTime', value
        >
          {timeOptions.map(time => (
            <SelectItem key={time.value} value={time.value}>
              {time.label}
```

```
            </SelectItem>
          ))}
        </Select>
      </div>
    </div>
  )}
</div>

<div className="border-t pt-6">
  <h3 className="font-semibold mb-4 flex items-center gap-2">
    <Calendar className="h-4 w-4" />
    Weekly Digest
  </h3>

  <div className="flex items-center justify-between mb-4">
    <div>
      <Label>Enable Weekly Digest</Label>
      <p className="text-sm text-muted-foreground">
        Receive a weekly summary
      </p>
    </div>
    <Switch
      checked={formState.weeklyDigestEnabled}
      onCheckedChange={(checked) => handleChange('weeklyDigestEnabled',
    />
  </div>

  {formState.weeklyDigestEnabled && (
    <div className="ml-4 space-y-4 border-l-2 pl-4">
      <div className="grid grid-cols-2 gap-4">
        <div className="space-y-2">
          <Label>Day</Label>
          <Select
            value={formState.weeklySendDay}
            onValueChange={(value) => handleChange('weeklySendDay', val
          >
            <SelectItem value="monday">Monday</SelectItem>
            <SelectItem value="sunday">Sunday</SelectItem>
          </Select>
        </div>
        <div className="space-y-2">
          <Label>Time</Label>
          <Select
            value={formState.weeklySendTime}
            onValueChange={(value) => handleChange('weeklySendTime', va
          >
            {timeOptions.map(time => (
```

```
                <SelectItem key={time.value} value={time.value}>
                  {time.label}
                </SelectItem>
              ))}
            </Select>
          </div>
        </div>
      </div>
    )}
</div>

<div className="border-t pt-6">
  <h3 className="font-semibold mb-4">Content Preferences</h3>
  <p className="text-sm text-muted-foreground mb-4">
    Choose what to include in your digests
  </p>

  <div className="space-y-3">
    <div className="flex items-center space-x-2">
      <Checkbox
        id="appointments"
        checked={formState.includeAppointments}
        onCheckedChange={(checked) => handleChange('includeAppointments
      />
      <Label htmlFor="appointments">📅 Appointments</Label>
    </div>

    <div className="flex items-center space-x-2">
      <Checkbox
        id="followups"
        checked={formState.includeFollowups}
        onCheckedChange={(checked) => handleChange('includeFollowups',
      />
      <Label htmlFor="followups">📞 Follow-ups due</Label>
    </div>

    <div className="flex items-center space-x-2">
      <Checkbox
        id="stale"
        checked={formState.includeStaleDeal}
        onCheckedChange={(checked) => handleChange('includeStaleDeal',
      />
      <Label htmlFor="stale">⚠️ Stale deals needing attention</Label>
    </div>

    <div className="flex items-center space-x-2">
      <Checkbox
```

```
          id="pipeline"
          checked={formState.includePipelineSummary}
          onCheckedChange={(checked) => handleChange('includePipelineSumm
        />
        <Label htmlFor="pipeline">📊 Pipeline summary & stats</Label>
      </div>

      <div className="flex items-center space-x-2">
        <Checkbox
          id="wins"
          checked={formState.includeRecentWins}
          onCheckedChange={(checked) => handleChange('includeRecentWins',
        />
        <Label htmlFor="wins">🎉 Recent wins celebration</Label>
      </div>

      <div className="flex items-center space-x-2">
        <Checkbox
          id="tips"
          checked={formState.includeAiTips}
          onCheckedChange={(checked) => handleChange('includeAiTips', che
        />
        <Label htmlFor="tips">💡 AI-generated tips & motivation</Label>
      </div>

      <div className="flex items-center space-x-2">
        <Checkbox
          id="checkins"
          checked={formState.includeQuarterlyCheckins}
          onCheckedChange={(checked) => handleChange('includeQuarterlyChe
        />
        <Label htmlFor="checkins">🤖 Quarterly check-ins due</Label>
      </div>

      <div className="flex items-center space-x-2">
        <Checkbox
          id="referrals"
          checked={formState.includeNewReferrals}
          onCheckedChange={(checked) => handleChange('includeNewReferrals
        />
        <Label htmlFor="referrals">🤝 New referrals received</Label>
      </div>
    </div>
  </div>

  <div className="border-t pt-6">
    <h3 className="font-semibold mb-4">Additional Options</h3>
```

```
      <div className="space-y-4">
        <div className="space-y-2">
          <Label>Appointment lookahead</Label>
          <Select
            value={formState.appointmentLookaheadDays?.toString()}
            onValueChange={(value) => handleChange('appointmentLookaheadDay
          >
            <SelectItem value="1">Today only</SelectItem>
            <SelectItem value="3">Next 3 days</SelectItem>
            <SelectItem value="7">Full week</SelectItem>
          </Select>
        </div>

        <div className="space-y-2">
          <Label>Stale deal threshold</Label>
          <Select
            value={formState.staleDealThresholdDays?.toString()}
            onValueChange={(value) => handleChange('staleDealThresholdDays'
          >
            <SelectItem value="5">5 days</SelectItem>
            <SelectItem value="7">7 days</SelectItem>
            <SelectItem value="14">14 days</SelectItem>
          </Select>
          <p className="text-sm text-muted-foreground">
            Deals with no activity after this many days will be flagged
          </p>
        </div>
      </div>
    </div>

    <div className="flex gap-3 pt-4">
      <Button onClick={handleSave} disabled={updatePrefs.isPending}>
        {updatePrefs.isPending ? 'Saving...' : 'Save Preferences'}
      </Button>

      <Button
        variant="outline"
        onClick={() => sendTest.mutate()}
        disabled={sendTest.isPending || !formState.dailyDigestEnabled && !f
      >
        <Send className="h-4 w-4 mr-2" />
        {sendTest.isPending ? 'Sending...' : 'Send Test Email'}
      </Button>
    </div>

  </CardContent>
```

```
      </Card>

      {/* Email History */}
      <Card>
        <CardHeader>
          <CardTitle>Recent Digests</CardTitle>
        </CardHeader>
        <CardContent>
          <EmailDigestHistory />
        </CardContent>
      </Card>
    </div>
  );
}

function EmailDigestHistory() {
  const { data, isLoading } = useQuery({
    queryKey: ['email-digest-history'],
    queryFn: async () => {
      const res = await fetch('/api/user/email-digest-history?limit=5');
      return res.json();
    }
  });

  if (isLoading) return <LoadingSpinner />;
  if (!data?.history?.length) {
    return <p className="text-muted-foreground text-sm">No digests sent yet.</p>;
  }

  return (
    <div className="space-y-2">
      {data.history.map((entry: any) => (
        <div key={entry.id} className="flex items-center justify-between py-2 bor
          <div>
            <p className="font-medium text-sm">{entry.subjectLine}</p>
            <p className="text-xs text-muted-foreground">
              {entry.digestType === 'daily' ? 'Daily' : 'Weekly'} • {formatDate(e
            </p>
          </div>
          <div className={`text-xs px-2 py-1 rounded ${
            entry.status === 'sent' ? 'bg-green-100 text-green-700' :
            entry.status === 'failed' ? 'bg-red-100 text-red-700' :
            'bg-gray-100 text-gray-700'
          }`}>
            {entry.status}
          </div>
        </div>
```

```
        ))}
      </div>
    );
  }
```

Dependencies

Add these packages:

```
npm install resend node-cron @anthropic-ai/sdk
```

Add these environment variables:

```
RESEND_API_KEY=re_xxxxxxxxxxx
ANTHROPIC_API_KEY=sk-ant-xxxxxxxxxxx
APP_URL=https://brochuretracker.com
```

Summary

This implementation provides:

1. ✅ **Opt-in daily/weekly email digests**
2. ✅ **Customizable email address** (can differ from login)
3. ✅ **Timezone-aware delivery** (sends at user's preferred local time)
4. ✅ **Configurable content** (appointments, follow-ups, pipeline stats, etc.)
5. ✅ **Claude AI-generated personalized summaries** (motivational, actionable)
6. ✅ **Beautiful HTML emails** with mobile-responsive design
7. ✅ **Test email feature** for previewing
8. ✅ **Delivery history** for troubleshooting
9. ✅ **Automatic scheduling** via cron job

The Claude AI writes personalized, motivational content that references the agent's actual deals by name, highlights priorities, and includes relevant sales tips based on their current pipeline situation.

End of Specification