**Automated Elder Care Lead Generation System - Build Specification**

**Project Overview**

**Product:** Automated lead generation platform connecting families seeking elder care with home care agencies.

**Business Model:**

* Capture leads via landing page
* Automatically match and distribute to 3 agencies in the service area
* First agency to claim gets the lead
* Charge $75-200 per lead
* Automated monthly invoicing

**Core Value:** Fully automated system that generates revenue 24/7 with minimal human intervention.

**Success Metrics:**

* 100+ leads/month capacity
* <30 second agency notification time
* 90%+ lead assignment rate
* Automated billing with 95%+ collection rate
* <2 hours/month human involvement

**Tech Stack**

**Core**

* **Framework:** Next.js 14 (App Router)
* **Language:** TypeScript (strict mode)
* **Database:** Supabase (PostgreSQL)
* **Authentication:** Supabase Auth (for agency portal)
* **Email:** Resend API
* **SMS:** Twilio API (optional, for urgent leads)
* **Payments:** Stripe
* **Deployment:** Vercel
* **Monitoring:** Sentry

**UI**

* **Styling:** Tailwind CSS
* **Components:** shadcn/ui
* **Forms:** React Hook Form + Zod
* **Icons:** Lucide React

**Development**

* **Package Manager:** pnpm
* **Linting:** ESLint + Prettier
* **Type Checking:** TypeScript strict
* **Git Hooks:** Husky + lint-staged

**Database Schema**

**Core Tables**

-- Enable UUID extension

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Leads table

CREATE TABLE leads (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

-- Lead information

zip\_code VARCHAR(10) NOT NULL,

city VARCHAR(100),

state VARCHAR(2),

care\_type VARCHAR(50) NOT NULL, -- 'dementia\_care', 'post\_hospital', 'daily\_assistance', 'around\_the\_clock'

urgency VARCHAR(20) NOT NULL, -- 'immediate', 'within\_week', 'within\_month', 'researching'

-- Contact information

contact\_name VARCHAR(200) NOT NULL,

contact\_email VARCHAR(255) NOT NULL,

contact\_phone VARCHAR(20) NOT NULL,

additional\_notes TEXT,

-- Assignment tracking

status VARCHAR(20) NOT NULL DEFAULT 'new', -- 'new', 'notified', 'claimed', 'contacted', 'converted', 'lost'

assigned\_agency\_id UUID REFERENCES agencies(id),

assigned\_at TIMESTAMPTZ,

notified\_agencies UUID[] DEFAULT '{}',

-- Security

claim\_token VARCHAR(64) UNIQUE NOT NULL,

-- Revenue tracking

lead\_value DECIMAL(10,2) NOT NULL DEFAULT 75.00,

converted\_value DECIMAL(10,2),

conversion\_confirmed\_at TIMESTAMPTZ,

-- Analytics

source VARCHAR(50), -- 'google\_ads', 'organic', 'facebook', 'referral'

utm\_campaign VARCHAR(100),

utm\_source VARCHAR(100),

utm\_medium VARCHAR(100),

landing\_page VARCHAR(255),

user\_agent TEXT,

ip\_address INET,

-- Quality scoring

quality\_score INTEGER, -- 0-100, calculated based on completeness and urgency

CONSTRAINT valid\_status CHECK (status IN ('new', 'notified', 'claimed', 'contacted', 'converted', 'lost')),

CONSTRAINT valid\_urgency CHECK (urgency IN ('immediate', 'within\_week', 'within\_month', 'researching')),

CONSTRAINT valid\_care\_type CHECK (care\_type IN ('dementia\_care', 'post\_hospital', 'daily\_assistance', 'around\_the\_clock'))

);

-- Agencies table

CREATE TABLE agencies (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

-- Basic information

name VARCHAR(200) NOT NULL,

contact\_email VARCHAR(255) NOT NULL UNIQUE,

contact\_phone VARCHAR(20),

website VARCHAR(255),

-- Authentication

user\_id UUID UNIQUE, -- Links to Supabase auth.users

-- Service area

service\_zip\_codes TEXT[] NOT NULL DEFAULT '{}',

service\_radius\_miles INTEGER,

service\_center\_lat DECIMAL(10,8),

service\_center\_lng DECIMAL(11,8),

-- Preferences

care\_types TEXT[] DEFAULT '{}', -- Empty array = all types

min\_urgency VARCHAR(20), -- Won't receive leads below this urgency

max\_leads\_per\_day INTEGER,

max\_leads\_per\_month INTEGER,

-- Pricing

lead\_price DECIMAL(10,2) NOT NULL DEFAULT 75.00,

pricing\_model VARCHAR(20) NOT NULL DEFAULT 'per\_lead', -- 'per\_lead' or 'per\_conversion'

conversion\_price DECIMAL(10,2),

-- Current usage

leads\_today INTEGER NOT NULL DEFAULT 0,

leads\_this\_month INTEGER NOT NULL DEFAULT 0,

last\_reset\_daily DATE,

last\_reset\_monthly DATE,

-- Status

status VARCHAR(20) NOT NULL DEFAULT 'active', -- 'active', 'paused', 'suspended'

paused\_until TIMESTAMPTZ,

-- Billing

stripe\_customer\_id VARCHAR(100),

auto\_billing BOOLEAN NOT NULL DEFAULT false,

payment\_terms INTEGER NOT NULL DEFAULT 30, -- Days until payment due

-- Performance metrics

total\_leads\_received INTEGER NOT NULL DEFAULT 0,

total\_leads\_converted INTEGER NOT NULL DEFAULT 0,

conversion\_rate DECIMAL(5,2),

avg\_response\_time\_seconds INTEGER,

avg\_claim\_time\_seconds INTEGER,

-- Notification preferences

email\_notifications BOOLEAN NOT NULL DEFAULT true,

sms\_notifications BOOLEAN NOT NULL DEFAULT false,

sms\_number VARCHAR(20),

notification\_email VARCHAR(255), -- Can differ from contact\_email

-- Quality metrics

quality\_score INTEGER DEFAULT 100, -- Decreases with slow responses, missed leads

CONSTRAINT valid\_status CHECK (status IN ('active', 'paused', 'suspended')),

CONSTRAINT valid\_pricing\_model CHECK (pricing\_model IN ('per\_lead', 'per\_conversion'))

);

-- Lead claims tracking

CREATE TABLE lead\_claims (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

lead\_id UUID NOT NULL REFERENCES leads(id) ON DELETE CASCADE,

agency\_id UUID NOT NULL REFERENCES agencies(id) ON DELETE CASCADE,

-- Timing

notified\_at TIMESTAMPTZ NOT NULL,

claimed\_at TIMESTAMPTZ,

response\_time\_seconds INTEGER,

-- Outcome

successful BOOLEAN NOT NULL DEFAULT false,

claim\_token VARCHAR(64) NOT NULL,

-- Tracking

notification\_method VARCHAR(20), -- 'email', 'sms', 'both'

claim\_method VARCHAR(20), -- 'email\_link', 'dashboard', 'api'

UNIQUE(lead\_id, agency\_id)

);

-- Invoices

CREATE TABLE invoices (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

agency\_id UUID NOT NULL REFERENCES agencies(id) ON DELETE CASCADE,

-- Period

invoice\_month DATE NOT NULL, -- First day of the month

-- Line items (calculated from leads)

leads\_count INTEGER NOT NULL,

amount\_due DECIMAL(10,2) NOT NULL,

-- Status

status VARCHAR(20) NOT NULL DEFAULT 'draft', -- 'draft', 'sent', 'paid', 'overdue', 'cancelled'

sent\_at TIMESTAMPTZ,

due\_date DATE,

paid\_at TIMESTAMPTZ,

-- Stripe integration

stripe\_invoice\_id VARCHAR(100),

stripe\_payment\_intent\_id VARCHAR(100),

stripe\_invoice\_url TEXT,

-- Payment tracking

payment\_method VARCHAR(50), -- 'stripe', 'manual', 'check'

payment\_reference VARCHAR(100),

CONSTRAINT valid\_invoice\_status CHECK (status IN ('draft', 'sent', 'paid', 'overdue', 'cancelled')),

UNIQUE(agency\_id, invoice\_month)

);

-- Invoice line items (detailed breakdown)

CREATE TABLE invoice\_line\_items (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

invoice\_id UUID NOT NULL REFERENCES invoices(id) ON DELETE CASCADE,

lead\_id UUID NOT NULL REFERENCES leads(id),

description TEXT NOT NULL,

amount DECIMAL(10,2) NOT NULL,

created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()

);

-- Analytics events

CREATE TABLE analytics\_events (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

event\_type VARCHAR(50) NOT NULL, -- 'page\_view', 'form\_start', 'form\_submit', 'lead\_claimed', etc.

-- Context

lead\_id UUID REFERENCES leads(id),

agency\_id UUID REFERENCES agencies(id),

-- Data

event\_data JSONB,

-- Request context

user\_agent TEXT,

ip\_address INET,

referrer TEXT,

-- Performance

page\_load\_time\_ms INTEGER,

api\_response\_time\_ms INTEGER

);

-- System configuration

CREATE TABLE config (

key VARCHAR(100) PRIMARY KEY,

value JSONB NOT NULL,

updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()

);

**Indexes**

-- Performance indexes

CREATE INDEX idx\_leads\_status ON leads(status);

CREATE INDEX idx\_leads\_created\_at ON leads(created\_at DESC);

CREATE INDEX idx\_leads\_zip\_code ON leads(zip\_code);

CREATE INDEX idx\_leads\_assigned\_agency ON leads(assigned\_agency\_id) WHERE assigned\_agency\_id IS NOT NULL;

CREATE INDEX idx\_leads\_source ON leads(source);

CREATE INDEX idx\_agencies\_status ON agencies(status) WHERE status = 'active';

CREATE INDEX idx\_agencies\_zip\_codes ON agencies USING GIN(service\_zip\_codes);

CREATE INDEX idx\_agencies\_care\_types ON agencies USING GIN(care\_types);

CREATE INDEX idx\_lead\_claims\_lead ON lead\_claims(lead\_id);

CREATE INDEX idx\_lead\_claims\_agency ON lead\_claims(agency\_id);

CREATE INDEX idx\_lead\_claims\_successful ON lead\_claims(successful) WHERE successful = true;

CREATE INDEX idx\_invoices\_agency\_month ON invoices(agency\_id, invoice\_month);

CREATE INDEX idx\_invoices\_status ON invoices(status);

CREATE INDEX idx\_invoices\_due\_date ON invoices(due\_date) WHERE status != 'paid';

CREATE INDEX idx\_analytics\_event\_type ON analytics\_events(event\_type);

CREATE INDEX idx\_analytics\_created\_at ON analytics\_events(created\_at DESC);

**Database Functions**

-- Function to increment agency lead counts

CREATE OR REPLACE FUNCTION increment\_agency\_leads(p\_agency\_id UUID)

RETURNS void AS $$

DECLARE

today DATE := CURRENT\_DATE;

current\_month DATE := DATE\_TRUNC('month', CURRENT\_DATE)::DATE;

BEGIN

UPDATE agencies

SET

leads\_today = CASE

WHEN last\_reset\_daily = today THEN leads\_today + 1

ELSE 1

END,

leads\_this\_month = CASE

WHEN last\_reset\_monthly = current\_month THEN leads\_this\_month + 1

ELSE 1

END,

last\_reset\_daily = today,

last\_reset\_monthly = current\_month,

total\_leads\_received = total\_leads\_received + 1

WHERE id = p\_agency\_id;

END;

$$ LANGUAGE plpgsql;

-- Function to calculate quality score for a lead

CREATE OR REPLACE FUNCTION calculate\_lead\_quality\_score(p\_lead\_id UUID)

RETURNS INTEGER AS $$

DECLARE

v\_score INTEGER := 0;

v\_lead RECORD;

BEGIN

SELECT \* INTO v\_lead FROM leads WHERE id = p\_lead\_id;

-- Base score

v\_score := 50;

-- Urgency bonus

CASE v\_lead.urgency

WHEN 'immediate' THEN v\_score := v\_score + 30;

WHEN 'within\_week' THEN v\_score := v\_score + 20;

WHEN 'within\_month' THEN v\_score := v\_score + 10;

ELSE v\_score := v\_score + 0;

END CASE;

-- Completeness bonus

IF v\_lead.contact\_phone IS NOT NULL AND LENGTH(v\_lead.contact\_phone) >= 10 THEN

v\_score := v\_score + 10;

END IF;

IF v\_lead.additional\_notes IS NOT NULL AND LENGTH(v\_lead.additional\_notes) > 20 THEN

v\_score := v\_score + 10;

END IF;

RETURN LEAST(v\_score, 100);

END;

$$ LANGUAGE plpgsql;

-- Trigger to update quality score on insert/update

CREATE OR REPLACE FUNCTION update\_lead\_quality\_score()

RETURNS TRIGGER AS $$

BEGIN

NEW.quality\_score := calculate\_lead\_quality\_score(NEW.id);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_update\_lead\_quality\_score

BEFORE INSERT OR UPDATE ON leads

FOR EACH ROW

EXECUTE FUNCTION update\_lead\_quality\_score();

-- Function to update updated\_at timestamp

CREATE OR REPLACE FUNCTION update\_updated\_at\_column()

RETURNS TRIGGER AS $$

BEGIN

NEW.updated\_at = NOW();

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_leads\_updated\_at BEFORE UPDATE ON leads FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();

CREATE TRIGGER update\_agencies\_updated\_at BEFORE UPDATE ON agencies FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();

CREATE TRIGGER update\_invoices\_updated\_at BEFORE UPDATE ON invoices FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();

**Row Level Security (RLS)**

-- Enable RLS

ALTER TABLE leads ENABLE ROW LEVEL SECURITY;

ALTER TABLE agencies ENABLE ROW LEVEL SECURITY;

ALTER TABLE lead\_claims ENABLE ROW LEVEL SECURITY;

ALTER TABLE invoices ENABLE ROW LEVEL SECURITY;

-- Policies for agencies table (agencies can only see their own data)

CREATE POLICY "Agencies can view own data" ON agencies

FOR SELECT USING (auth.uid() = user\_id);

CREATE POLICY "Agencies can update own data" ON agencies

FOR UPDATE USING (auth.uid() = user\_id);

-- Policies for leads (agencies can only see leads assigned to them or available to claim)

CREATE POLICY "Agencies can view assigned leads" ON leads

FOR SELECT USING (

assigned\_agency\_id IN (SELECT id FROM agencies WHERE user\_id = auth.uid())

OR id IN (

SELECT lead\_id FROM lead\_claims

WHERE agency\_id IN (SELECT id FROM agencies WHERE user\_id = auth.uid())

)

);

-- Policies for invoices (agencies can only see their own invoices)

CREATE POLICY "Agencies can view own invoices" ON invoices

FOR SELECT USING (

agency\_id IN (SELECT id FROM agencies WHERE user\_id = auth.uid())

);

-- Service role has full access (bypass RLS)

**Application Structure**

elder-care-lead-gen/

├── app/

│ ├── layout.tsx # Root layout

│ ├── page.tsx # Landing page (lead capture)

│ ├── thank-you/

│ │ └── page.tsx # Post-submission confirmation

│ ├── claim/

│ │ └── [token]/

│ │ └── page.tsx # Agency claim page

│ ├── agency/

│ │ ├── layout.tsx # Agency portal layout

│ │ ├── login/

│ │ │ └── page.tsx # Agency login

│ │ ├── dashboard/

│ │ │ └── page.tsx # Available leads + claimed leads

│ │ ├── leads/

│ │ │ ├── page.tsx # Lead history

│ │ │ └── [id]/

│ │ │ └── page.tsx # Lead details

│ │ ├── billing/

│ │ │ ├── page.tsx # Invoices

│ │ │ └── [invoiceId]/

│ │ │ └── page.tsx # Invoice details

│ │ └── settings/

│ │ └── page.tsx # Agency settings

│ ├── admin/

│ │ ├── dashboard/

│ │ │ └── page.tsx # Admin overview

│ │ ├── agencies/

│ │ │ └── page.tsx # Manage agencies

│ │ └── analytics/

│ │ └── page.tsx # System analytics

│ └── api/

│ ├── leads/

│ │ ├── submit/

│ │ │ └── route.ts # Submit new lead

│ │ ├── assign/

│ │ │ └── route.ts # Assign lead to agencies

│ │ └── claim/

│ │ └── route.ts # Claim a lead

│ ├── agencies/

│ │ ├── register/

│ │ │ └── route.ts # Register new agency

│ │ └── update-settings/

│ │ └── route.ts # Update agency settings

│ ├── webhooks/

│ │ └── stripe/

│ │ └── route.ts # Stripe webhook handler

│ └── cron/

│ ├── generate-invoices/

│ │ └── route.ts # Monthly invoice generation

│ ├── send-invoice-reminders/

│ │ └── route.ts # Send payment reminders

│ └── reset-daily-counts/

│ └── route.ts # Reset daily lead counts

├── components/

│ ├── ui/ # shadcn/ui components

│ ├── LeadForm.tsx # Main lead capture form

│ ├── LeadCard.tsx # Display lead in dashboard

│ ├── AgencyNav.tsx # Agency portal navigation

│ └── InvoiceTable.tsx # Invoice display

├── lib/

│ ├── supabase/

│ │ ├── client.ts # Supabase client (browser)

│ │ ├── server.ts # Supabase client (server)

│ │ └── admin.ts # Supabase admin client

│ ├── services/

│ │ ├── lead-assignment.ts # Core lead assignment logic

│ │ ├── notifications.ts # Email/SMS notifications

│ │ ├── billing.ts # Invoice generation

│ │ └── analytics.ts # Analytics tracking

│ ├── utils/

│ │ ├── geo.ts # Zip code distance calculations

│ │ ├── validation.ts # Zod schemas

│ │ └── formatting.ts # Display formatters

│ └── constants.ts # App constants

├── types/

│ └── index.ts # TypeScript types

├── emails/

│ ├── agency-notification.tsx # React Email template

│ ├── lead-details.tsx # React Email template

│ └── invoice.tsx # React Email template

└── public/

└── images/

**Core Implementation**

**Lead Submission API**

// app/api/leads/submit/route.ts

import { NextRequest, NextResponse } from 'next/server';

import { z } from 'zod';

import { createServerClient } from '@/lib/supabase/server';

import { assignLeadToAgencies } from '@/lib/services/lead-assignment';

import { trackEvent } from '@/lib/services/analytics';

import crypto from 'crypto';

const leadSchema = z.object({

zipCode: z.string().regex(/^\d{5}$/),

careType: z.enum(['dementia\_care', 'post\_hospital', 'daily\_assistance', 'around\_the\_clock']),

urgency: z.enum(['immediate', 'within\_week', 'within\_month', 'researching']),

contactName: z.string().min(2).max(200),

contactEmail: z.string().email(),

contactPhone: z.string().regex(/^\d{10,}$/),

additionalNotes: z.string().optional(),

source: z.string().optional(),

utmCampaign: z.string().optional(),

utmSource: z.string().optional(),

utmMedium: z.string().optional(),

});

export async function POST(request: NextRequest) {

try {

const body = await request.json();

const validated = leadSchema.parse(body);

const supabase = createServerClient();

// Get city/state from zip code

const geoData = await fetch(`https://api.zippopotam.us/us/${validated.zipCode}`).then(r => r.json());

const city = geoData.places?.[0]?.['place name'];

const state = geoData.places?.[0]?.['state abbreviation'];

// Generate unique claim token

const claimToken = crypto.randomBytes(32).toString('hex');

// Calculate lead value based on urgency

const leadValue = validated.urgency === 'immediate' ? 100 :

validated.urgency === 'within\_week' ? 75 : 50;

// Create lead

const { data: lead, error } = await supabase

.from('leads')

.insert({

zip\_code: validated.zipCode,

city,

state,

care\_type: validated.careType,

urgency: validated.urgency,

contact\_name: validated.contactName,

contact\_email: validated.contactEmail,

contact\_phone: validated.contactPhone,

additional\_notes: validated.additionalNotes,

claim\_token: claimToken,

lead\_value: leadValue,

source: validated.source || 'direct',

utm\_campaign: validated.utmCampaign,

utm\_source: validated.utmSource,

utm\_medium: validated.utmMedium,

ip\_address: request.headers.get('x-forwarded-for') || request.headers.get('x-real-ip'),

user\_agent: request.headers.get('user-agent'),

})

.select()

.single();

if (error) throw error;

// Track analytics

await trackEvent({

event\_type: 'lead\_submitted',

lead\_id: lead.id,

event\_data: {

care\_type: validated.careType,

urgency: validated.urgency,

source: validated.source,

},

});

// Trigger async lead assignment (don't wait for it)

assignLeadToAgencies(lead.id).catch(err => {

console.error('Lead assignment failed:', err);

// Log to monitoring service

});

return NextResponse.json({

success: true,

message: 'Thank you! Local agencies will contact you within 24 hours.',

leadId: lead.id,

});

} catch (error) {

if (error instanceof z.ZodError) {

return NextResponse.json(

{ error: 'Invalid form data', details: error.errors },

{ status: 400 }

);

}

console.error('Lead submission error:', error);

return NextResponse.json(

{ error: 'Failed to submit lead. Please try again.' },

{ status: 500 }

);

}

}

**Lead Assignment Service**

// lib/services/lead-assignment.ts

import { createAdminClient } from '@/lib/supabase/admin';

import { sendAgencyNotification } from './notifications';

import { calculateDistance } from '@/lib/utils/geo';

export async function assignLeadToAgencies(leadId: string) {

const supabase = createAdminClient();

// Get lead details

const { data: lead, error: leadError } = await supabase

.from('leads')

.select('\*')

.eq('id', leadId)

.single();

if (leadError || !lead) {

throw new Error(`Lead not found: ${leadId}`);

}

// Find matching agencies

const { data: allAgencies } = await supabase

.from('agencies')

.select('\*')

.eq('status', 'active');

if (!allAgencies || allAgencies.length === 0) {

console.error('No active agencies found');

// TODO: Alert admin

return;

}

// Filter agencies by service area

const matchingAgencies = allAgencies.filter(agency => {

// Check if agency serves this zip code

if (agency.service\_zip\_codes.includes(lead.zip\_code)) {

return true;

}

// Check if within radius (if agency uses radius-based service area)

if (agency.service\_radius\_miles && agency.service\_center\_lat && agency.service\_center\_lng) {

const distance = calculateDistance(

agency.service\_center\_lat,

agency.service\_center\_lng,

lead.zip\_code // This would need geocoding

);

return distance <= agency.service\_radius\_miles;

}

return false;

});

// Filter by preferences

const eligibleAgencies = matchingAgencies.filter(agency => {

// Check care type preference

if (agency.care\_types.length > 0 && !agency.care\_types.includes(lead.care\_type)) {

return false;

}

// Check urgency preference

if (agency.min\_urgency) {

const urgencyOrder = ['researching', 'within\_month', 'within\_week', 'immediate'];

const agencyMinIndex = urgencyOrder.indexOf(agency.min\_urgency);

const leadUrgencyIndex = urgencyOrder.indexOf(lead.urgency);

if (leadUrgencyIndex < agencyMinIndex) {

return false;

}

}

// Check daily/monthly caps

if (agency.max\_leads\_per\_day && agency.leads\_today >= agency.max\_leads\_per\_day) {

return false;

}

if (agency.max\_leads\_per\_month && agency.leads\_this\_month >= agency.max\_leads\_per\_month) {

return false;

}

// Check if paused

if (agency.paused\_until && new Date(agency.paused\_until) > new Date()) {

return false;

}

return true;

});

if (eligibleAgencies.length === 0) {

console.error('No eligible agencies found for lead', leadId);

// TODO: Alert admin, expand search criteria, or refund customer

return;

}

// Prioritize by quality score and conversion rate

const sortedAgencies = eligibleAgencies.sort((a, b) => {

const scoreA = (a.quality\_score || 50) + ((a.conversion\_rate || 0) \* 10);

const scoreB = (b.quality\_score || 50) + ((b.conversion\_rate || 0) \* 10);

return scoreB - scoreA;

});

// Select top 3 agencies

const selectedAgencies = sortedAgencies.slice(0, 3);

// Update lead with notified agencies

await supabase

.from('leads')

.update({

status: 'notified',

notified\_agencies: selectedAgencies.map(a => a.id),

})

.eq('id', leadId);

// Create claim records

const claimRecords = selectedAgencies.map(agency => ({

lead\_id: leadId,

agency\_id: agency.id,

claim\_token: lead.claim\_token,

notified\_at: new Date().toISOString(),

notification\_method: agency.sms\_notifications ? 'both' : 'email',

}));

await supabase.from('lead\_claims').insert(claimRecords);

// Send notifications to all selected agencies

await Promise.all(

selectedAgencies.map(agency =>

sendAgencyNotification(agency, lead)

.catch(err => console.error(`Failed to notify agency ${agency.id}:`, err))

)

);

console.log(`Lead ${leadId} assigned to ${selectedAgencies.length} agencies`);

}

**Notification Service**

// lib/services/notifications.ts

import { Resend } from 'resend';

import { Twilio } from 'twilio';

import AgencyNotificationEmail from '@/emails/agency-notification';

import LeadDetailsEmail from '@/emails/lead-details';

const resend = new Resend(process.env.RESEND\_API\_KEY);

const twilioClient = process.env.TWILIO\_ACCOUNT\_SID

? new Twilio(process.env.TWILIO\_ACCOUNT\_SID, process.env.TWILIO\_AUTH\_TOKEN)

: null;

export async function sendAgencyNotification(agency: any, lead: any) {

const claimUrl = `${process.env.NEXT\_PUBLIC\_APP\_URL}/claim/${lead.claim\_token}?agency=${agency.id}`;

// Send email

if (agency.email\_notifications) {

try {

await resend.emails.send({

from: 'Elder Care Leads <leads@eldercareleads.com>',

to: agency.notification\_email || agency.contact\_email,

subject: `New ${formatUrgency(lead.urgency)} Lead in ${lead.city}`,

react: AgencyNotificationEmail({

agencyName: agency.name,

lead,

claimUrl,

}),

});

} catch (error) {

console.error('Email notification failed:', error);

}

}

// Send SMS for immediate leads

if (agency.sms\_notifications && agency.sms\_number && lead.urgency === 'immediate' && twilioClient) {

try {

await twilioClient.messages.create({

to: agency.sms\_number,

from: process.env.TWILIO\_PHONE\_NUMBER,

body: `URGENT: New ${formatCareType(lead.care\_type)} lead in ${lead.city}. Claim now: ${claimUrl}`,

});

} catch (error) {

console.error('SMS notification failed:', error);

}

}

}

export async function sendLeadDetails(agency: any, lead: any) {

try {

await resend.emails.send({

from: 'Elder Care Leads <leads@eldercareleads.com>',

to: agency.notification\_email || agency.contact\_email,

subject: `Lead Details - ${lead.contact\_name}`,

react: LeadDetailsEmail({

agencyName: agency.name,

lead,

}),

});

} catch (error) {

console.error('Failed to send lead details:', error);

throw error;

}

}

export async function sendLeadClaimedNotification(agency: any, lead: any) {

try {

await resend.emails.send({

from: 'Elder Care Leads <leads@eldercareleads.com>',

to: agency.notification\_email || agency.contact\_email,

subject: `Lead Already Claimed - ${lead.city}`,

html: `

<p>Hi ${agency.name},</p>

<p>The lead in ${lead.city} for ${formatCareType(lead.care\_type)} has already been claimed by another agency.</p>

<p>You'll receive notification when the next lead in your area becomes available.</p>

<p>Best regards,<br>Elder Care Leads</p>

`,

});

} catch (error) {

console.error('Failed to send claimed notification:', error);

}

}

function formatUrgency(urgency: string): string {

const map: Record<string, string> = {

immediate: 'Immediate',

within\_week: 'This Week',

within\_month: 'This Month',

researching: 'Researching',

};

return map[urgency] || urgency;

}

function formatCareType(careType: string): string {

const map: Record<string, string> = {

dementia\_care: 'Dementia Care',

post\_hospital: 'Post-Hospital Care',

daily\_assistance: 'Daily Assistance',

around\_the\_clock: '24/7 Care',

};

return map[careType] || careType;

}

**Lead Claim API**

// app/api/leads/claim/route.ts

import { NextRequest, NextResponse } from 'next/server';

import { createServerClient } from '@/lib/supabase/server';

import { sendLeadDetails, sendLeadClaimedNotification } from '@/lib/services/notifications';

import { trackEvent } from '@/lib/services/analytics';

export async function POST(request: NextRequest) {

try {

const { token, agencyId } = await request.json();

if (!token || !agencyId) {

return NextResponse.json({ error: 'Missing parameters' }, { status: 400 });

}

const supabase = createServerClient();

// Get lead by token

const { data: lead, error: leadError } = await supabase

.from('leads')

.select('\*')

.eq('claim\_token', token)

.single();

if (leadError || !lead) {

return NextResponse.json({ error: 'Invalid claim token' }, { status: 404 });

}

// Check if lead already claimed (atomic check)

if (lead.status === 'claimed') {

// Get agency info for notification

const { data: agency } = await supabase

.from('agencies')

.select('\*')

.eq('id', agencyId)

.single();

if (agency) {

await sendLeadClaimedNotification(agency, lead);

}

return NextResponse.json({

error: 'Lead already claimed',

message: 'Another agency claimed this lead first.',

}, { status: 409 });

}

// Verify agency was notified about this lead

if (!lead.notified\_agencies.includes(agencyId)) {

return NextResponse.json({ error: 'Unauthorized' }, { status: 403 });

}

// Get claim start time for response time calculation

const { data: claimRecord } = await supabase

.from('lead\_claims')

.select('notified\_at')

.eq('lead\_id', lead.id)

.eq('agency\_id', agencyId)

.single();

const claimedAt = new Date();

const responseTimeSeconds = claimRecord

? Math.floor((claimedAt.getTime() - new Date(claimRecord.notified\_at).getTime()) / 1000)

: null;

// Atomic claim operation

const { data: updated, error: updateError } = await supabase

.from('leads')

.update({

status: 'claimed',

assigned\_agency\_id: agencyId,

assigned\_at: claimedAt.toISOString(),

})

.eq('id', lead.id)

.eq('status', 'notified') // Only update if still in notified state

.select()

.single();

if (updateError || !updated) {

// Race condition - another agency claimed it

return NextResponse.json({

error: 'Lead already claimed',

message: 'Another agency claimed this lead just now.',

}, { status: 409 });

}

// Update claim record

await supabase

.from('lead\_claims')

.update({

successful: true,

claimed\_at: claimedAt.toISOString(),

response\_time\_seconds: responseTimeSeconds,

claim\_method: 'email\_link',

})

.eq('lead\_id', lead.id)

.eq('agency\_id', agencyId);

// Increment agency counters

await supabase.rpc('increment\_agency\_leads', { p\_agency\_id: agencyId });

// Get full agency details

const { data: agency } = await supabase

.from('agencies')

.select('\*')

.eq('id', agencyId)

.single();

if (!agency) {

return NextResponse.json({ error: 'Agency not found' }, { status: 404 });

}

// Send lead details to winning agency

await sendLeadDetails(agency, lead);

// Notify other agencies that lead was claimed

const otherAgencies = lead.notified\_agencies.filter((id: string) => id !== agencyId);

for (const otherAgencyId of otherAgencies) {

const { data: otherAgency } = await supabase

.from('agencies')

.select('\*')

.eq('id', otherAgencyId)

.single();

if (otherAgency) {

await sendLeadClaimedNotification(otherAgency, lead);

}

}

// Track analytics

await trackEvent({

event\_type: 'lead\_claimed',

lead\_id: lead.id,

agency\_id: agencyId,

event\_data: {

response\_time\_seconds: responseTimeSeconds,

claim\_method: 'email\_link',

},

});

return NextResponse.json({

success: true,

lead: {

id: lead.id,

name: lead.contact\_name,

email: lead.contact\_email,

phone: lead.contact\_phone,

city: lead.city,

state: lead.state,

zipCode: lead.zip\_code,

careType: lead.care\_type,

urgency: lead.urgency,

notes: lead.additional\_notes,

},

});

} catch (error) {

console.error('Claim error:', error);

return NextResponse.json(

{ error: 'Failed to claim lead. Please try again.' },

{ status: 500 }

);

}

}

**Automated Invoice Generation**

// app/api/cron/generate-invoices/route.ts

import { NextRequest, NextResponse } from 'next/server';

import { createAdminClient } from '@/lib/supabase/admin';

import { Resend } from 'resend';

import InvoiceEmail from '@/emails/invoice';

import Stripe from 'stripe';

const stripe = new Stripe(process.env.STRIPE\_SECRET\_KEY!, { apiVersion: '2023-10-16' });

const resend = new Resend(process.env.RESEND\_API\_KEY);

export async function GET(request: NextRequest) {

// Verify cron secret

const authHeader = request.headers.get('authorization');

if (authHeader !== `Bearer ${process.env.CRON\_SECRET}`) {

return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });

}

const supabase = createAdminClient();

// Calculate last month

const now = new Date();

const lastMonth = new Date(now.getFullYear(), now.getMonth() - 1, 1);

const invoiceMonth = lastMonth.toISOString().slice(0, 7) + '-01';

// Get all active agencies

const { data: agencies } = await supabase

.from('agencies')

.select('\*')

.neq('status', 'suspended');

if (!agencies) {

return NextResponse.json({ error: 'No agencies found' }, { status: 404 });

}

const results = [];

for (const agency of agencies) {

try {

// Get leads from last month

const { data: leads, count } = await supabase

.from('leads')

.select('\*', { count: 'exact' })

.eq('assigned\_agency\_id', agency.id)

.gte('assigned\_at', lastMonth.toISOString())

.lt('assigned\_at', new Date(now.getFullYear(), now.getMonth(), 1).toISOString());

if (!count || count === 0) {

results.push({ agency: agency.name, status: 'no\_leads' });

continue;

}

// Calculate total amount

const totalAmount = leads!.reduce((sum, lead) => {

return sum + Number(lead.lead\_value);

}, 0);

// Check if invoice already exists

const { data: existingInvoice } = await supabase

.from('invoices')

.select('id')

.eq('agency\_id', agency.id)

.eq('invoice\_month', invoiceMonth)

.single();

if (existingInvoice) {

results.push({ agency: agency.name, status: 'already\_exists' });

continue;

}

// Create invoice

const dueDate = new Date();

dueDate.setDate(dueDate.getDate() + agency.payment\_terms);

const { data: invoice, error: invoiceError } = await supabase

.from('invoices')

.insert({

agency\_id: agency.id,

invoice\_month: invoiceMonth,

leads\_count: count,

amount\_due: totalAmount,

due\_date: dueDate.toISOString().split('T')[0],

status: 'draft',

})

.select()

.single();

if (invoiceError) throw invoiceError;

// Create line items

const lineItems = leads!.map(lead => ({

invoice\_id: invoice.id,

lead\_id: lead.id,

description: `${formatCareType(lead.care\_type)} lead in ${lead.city} - ${formatDate(lead.assigned\_at)}`,

amount: lead.lead\_value,

}));

await supabase.from('invoice\_line\_items').insert(lineItems);

// Handle billing based on agency settings

if (agency.auto\_billing && agency.stripe\_customer\_id) {

// Create Stripe invoice

const stripeInvoice = await stripe.invoices.create({

customer: agency.stripe\_customer\_id,

auto\_advance: true,

collection\_method: 'charge\_automatically',

description: `Elder Care Leads - ${formatMonth(invoiceMonth)}`,

metadata: {

invoice\_id: invoice.id,

agency\_id: agency.id,

},

});

// Add line items to Stripe invoice

for (const lineItem of lineItems) {

await stripe.invoiceItems.create({

customer: agency.stripe\_customer\_id,

invoice: stripeInvoice.id,

amount: Math.round(Number(lineItem.amount) \* 100), // Convert to cents

currency: 'usd',

description: lineItem.description,

});

}

// Finalize and send

const finalizedInvoice = await stripe.invoices.finalizeInvoice(stripeInvoice.id);

// Update database with Stripe info

await supabase

.from('invoices')

.update({

status: 'sent',

sent\_at: new Date().toISOString(),

stripe\_invoice\_id: stripeInvoice.id,

stripe\_invoice\_url: finalizedInvoice.hosted\_invoice\_url,

})

.eq('id', invoice.id);

results.push({ agency: agency.name, status: 'stripe\_billed', amount: totalAmount });

} else {

// Send email invoice

await resend.emails.send({

from: 'Elder Care Leads Billing <billing@eldercareleads.com>',

to: agency.contact\_email,

subject: `Invoice for ${formatMonth(invoiceMonth)} - $${totalAmount.toFixed(2)}`,

react: InvoiceEmail({

agencyName: agency.name,

invoice,

lineItems,

}),

});

await supabase

.from('invoices')

.update({

status: 'sent',

sent\_at: new Date().toISOString(),

})

.eq('id', invoice.id);

results.push({ agency: agency.name, status: 'email\_sent', amount: totalAmount });

}

} catch (error) {

console.error(`Failed to generate invoice for ${agency.name}:`, error);

results.push({ agency: agency.name, status: 'error', error: String(error) });

}

}

return NextResponse.json({

success: true,

month: invoiceMonth,

results

});

}

function formatCareType(careType: string): string {

const map: Record<string, string> = {

dementia\_care: 'Dementia Care',

post\_hospital: 'Post-Hospital Care',

daily\_assistance: 'Daily Assistance',

around\_the\_clock: '24/7 Care',

};

return map[careType] || careType;

}

function formatDate(dateString: string): string {

return new Date(dateString).toLocaleDateString('en-US', {

month: 'short',

day: 'numeric',

year: 'numeric',

});

}

function formatMonth(dateString: string): string {

return new Date(dateString).toLocaleDateString('en-US', {

month: 'long',

year: 'numeric',

});

}

**Landing Page Implementation**

// app/page.tsx

import LeadForm from '@/components/LeadForm';

export default function HomePage() {

return (

<div className="min-h-screen bg-gradient-to-b from-blue-50 to-white">

<header className="container mx-auto px-4 py-6">

<h1 className="text-2xl font-bold text-blue-900">Elder Care Help</h1>

</header>

<main className="container mx-auto px-4 py-12">

<div className="max-w-3xl mx-auto">

<div className="text-center mb-12">

<h2 className="text-4xl md:text-5xl font-bold text-gray-900 mb-4">

Find Trusted Home Care for Your Parent

</h2>

<p className="text-xl text-gray-600 mb-2">

We'll connect you with 3 vetted home care agencies in your area.

</p>

<p className="text-lg text-gray-500">

Free, no obligation. Get responses within 24 hours.

</p>

</div>

<div className="bg-white rounded-lg shadow-xl p-8">

<LeadForm />

</div>

<div className="mt-8 text-center text-sm text-gray-500">

<p>✓ Free service</p>

<p>✓ No obligation</p>

<p>✓ Agencies contact you</p>

</div>

</div>

</main>

</div>

);

}

// components/LeadForm.tsx

'use client';

import { useState } from 'react';

import { useRouter } from 'next/navigation';

import { useForm } from 'react-hook-form';

import { zodResolver } from '@hookform/resolvers/zod';

import { z } from 'zod';

import { Button } from '@/components/ui/button';

import { Input } from '@/components/ui/input';

import { Label } from '@/components/ui/label';

import { RadioGroup, RadioGroupItem } from '@/components/ui/radio-group';

import { Textarea } from '@/components/ui/textarea';

const formSchema = z.object({

zipCode: z.string().regex(/^\d{5}$/, 'Please enter a valid 5-digit zip code'),

careType: z.enum(['dementia\_care', 'post\_hospital', 'daily\_assistance', 'around\_the\_clock']),

urgency: z.enum(['immediate', 'within\_week', 'within\_month', 'researching']),

contactName: z.string().min(2, 'Please enter your name'),

contactEmail: z.string().email('Please enter a valid email'),

contactPhone: z.string().regex(/^\d{10}$/, 'Please enter a 10-digit phone number'),

additionalNotes: z.string().optional(),

});

type FormData = z.infer<typeof formSchema>;

export default function LeadForm() {

const router = useRouter();

const [isSubmitting, setIsSubmitting] = useState(false);

const {

register,

handleSubmit,

formState: { errors },

setValue,

watch,

} = useForm<FormData>({

resolver: zodResolver(formSchema),

});

const onSubmit = async (data: FormData) => {

setIsSubmitting(true);

try {

const response = await fetch('/api/leads/submit', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(data),

});

if (!response.ok) {

throw new Error('Submission failed');

}

router.push('/thank-you');

} catch (error) {

console.error('Form submission error:', error);

alert('Something went wrong. Please try again.');

setIsSubmitting(false);

}

};

return (

<form onSubmit={handleSubmit(onSubmit)} className="space-y-6">

<div>

<Label htmlFor="zipCode">Your Parent's Zip Code \*</Label>

<Input

id="zipCode"

{...register('zipCode')}

placeholder="e.g., 85001"

className="mt-1"

/>

{errors.zipCode && (

<p className="text-sm text-red-600 mt-1">{errors.zipCode.message}</p>

)}

</div>

<div>

<Label>What type of care do you need? \*</Label>

<RadioGroup

onValueChange={(value) => setValue('careType', value as any)}

className="mt-2 space-y-2"

>

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

<RadioGroupItem value="dementia\_care" id="dementia" />

<Label htmlFor="dementia" className="font-normal cursor-pointer">

Dementia/Alzheimer's Care

</Label>

</div>

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

<RadioGroupItem value="post\_hospital" id="hospital" />

<Label htmlFor="hospital" className="font-normal cursor-pointer">

Post-Hospital Recovery

</Label>

</div>

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

<RadioGroupItem value="daily\_assistance" id="daily" />

<Label htmlFor="daily" className="font-normal cursor-pointer">

Daily Living Assistance

</Label>

</div>

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

<RadioGroupItem value="around\_the\_clock" id="24\_7" />

<Label htmlFor="24\_7" className="font-normal cursor-pointer">

24/7 Care

</Label>

</div>

</RadioGroup>

{errors.careType && (

<p className="text-sm text-red-600 mt-1">{errors.careType.message}</p>

)}

</div>

<div>

<Label>When do you need care? \*</Label>

<RadioGroup

onValueChange={(value) => setValue('urgency', value as any)}

className="mt-2 space-y-2"

>

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

<RadioGroupItem value="immediate" id="immediate" />

<Label htmlFor="immediate" className="font-normal cursor-pointer">

Immediately

</Label>

</div>

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

<RadioGroupItem value="within\_week" id="week" />

<Label htmlFor="week" className="font-normal cursor-pointer">

Within a week

</Label>

</div>

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

<RadioGroupItem value="within\_month" id="month" />

<Label htmlFor="month" className="font-normal cursor-pointer">

Within a month

</Label>

</div>

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

<RadioGroupItem value="researching" id="research" />

<Label htmlFor="research" className="font-normal cursor-pointer">

Just researching options

</Label>

</div>

</RadioGroup>

{errors.urgency && (

<p className="text-sm text-red-600 mt-1">{errors.urgency.message}</p>

)}

</div>

<div>

<Label htmlFor="contactName">Your Name \*</Label>

<Input

id="contactName"

{...register('contactName')}

placeholder="John Smith"

className="mt-1"

/>

{errors.contactName && (

<p className="text-sm text-red-600 mt-1">{errors.contactName.message}</p>

)}

</div>

<div>

<Label htmlFor="contactEmail">Your Email \*</Label>

<Input

id="contactEmail"

type="email"

{...register('contactEmail')}

placeholder="john@example.com"

className="mt-1"

/>

{errors.contactEmail && (

<p className="text-sm text-red-600 mt-1">{errors.contactEmail.message}</p>

)}

</div>

<div>

<Label htmlFor="contactPhone">Your Phone Number \*</Label>

<Input

id="contactPhone"

{...register('contactPhone')}

placeholder="5555551234"

className="mt-1"

/>

{errors.contactPhone && (

<p className="text-sm text-red-600 mt-1">{errors.contactPhone.message}</p>

)}

</div>

<div>

<Label htmlFor="additionalNotes">Additional Information (Optional)</Label>

<Textarea

id="additionalNotes"

{...register('additionalNotes')}

placeholder="Any specific needs or preferences..."

className="mt-1"

rows={4}

/>

</div>

<Button

type="submit"

disabled={isSubmitting}

className="w-full bg-blue-600 hover:bg-blue-700 text-white font-semibold py-3 text-lg"

>

{isSubmitting ? 'Submitting...' : 'Get Matched With Agencies'}

</Button>

<p className="text-xs text-gray-500 text-center">

By submitting, you agree to be contacted by home care agencies in your area.

</p>

</form>

);

}

**Week-by-Week Build Plan**

**Week 1: Foundation (Days 1-7)**

**Day 1: Project Setup**

* Initialize Next.js 14 with TypeScript
* Install dependencies (Tailwind, shadcn/ui, etc.)
* Set up Supabase project
* Configure environment variables
* Set up Git repository
* Deploy skeleton to Vercel

**Day 2-3: Database Setup**

* Create all database tables using provided schema
* Set up indexes
* Create database functions
* Configure RLS policies
* Test database connections
* Seed with test data (3-5 test agencies)

**Day 4-5: Core API Routes**

* Implement /api/leads/submit
* Implement /api/leads/assign
* Implement /api/leads/claim
* Test full lead flow end-to-end

**Day 6-7: Landing Page & Form**

* Build responsive landing page
* Implement lead capture form with validation
* Build thank-you page
* Test form submission

**Deliverable:** Working lead submission and assignment system

**Week 2: Automation & Notifications (Days 8-14)**

**Day 8-9: Email System**

* Set up Resend account
* Create email templates (React Email)
* Implement agency notification emails
* Implement lead details emails
* Implement lead claimed notifications
* Test all email flows

**Day 10-11: Lead Assignment Logic**

* Implement geo-matching algorithm
* Implement preference filtering
* Implement agency caps/limits
* Implement prioritization (quality score, conversion rate)
* Test with various scenarios

**Day 12-13: Claim System**

* Build claim landing page (/claim/[token])
* Implement atomic claim logic (race condition handling)
* Track response times
* Test simultaneous claims

**Day 14: Testing & Refinement**

* End-to-end testing with real scenarios
* Fix bugs
* Optimize database queries
* Add error monitoring

**Deliverable:** Fully automated lead distribution and claiming

**Week 3: Agency Portal (Days 15-21)**

**Day 15-16: Authentication**

* Set up Supabase Auth
* Implement agency registration
* Implement agency login
* Implement password reset

**Day 17-18: Agency Dashboard**

* Build dashboard layout
* Display available leads (real-time with Supabase subscriptions)
* Display claimed leads
* Display lead history
* Implement one-click claim from dashboard

**Day 19-20: Agency Settings**

* Build settings page
* Implement service area management
* Implement notification preferences
* Implement capacity limits
* Test all settings updates

**Day 21: Agency Billing View**

* Display current month's leads
* Display invoice history
* Show payment status
* Add Stripe payment integration (basic)

**Deliverable:** Functional agency portal

**Week 4: Billing & Analytics (Days 22-28)**

**Day 22-23: Automated Invoicing**

* Implement invoice generation cron job
* Integrate Stripe for auto-billing
* Create invoice email template
* Test invoice generation

**Day 24: Payment Reminders**

* Implement overdue invoice detection
* Send payment reminder emails
* Update invoice statuses

**Day 25-26: Analytics & Tracking**

* Implement analytics event tracking
* Build admin dashboard (basic)
* Display key metrics:
  + Leads per day/week/month
  + Assignment rate
  + Claim rate
  + Revenue tracking
  + Agency performance

**Day 27: Optimization**

* Performance optimization (Lighthouse audit)
* SEO optimization
* Error handling improvements
* Add monitoring (Sentry)

**Day 28: Launch Preparation**

* Final end-to-end testing
* Load testing
* Set up production environment
* Deploy to production
* Monitor for issues

**Deliverable:** Production-ready automated lead generation system

**Environment Variables**

# .env.local

# App

NEXT\_PUBLIC\_APP\_URL=https://eldercareleads.com

# Supabase

NEXT\_PUBLIC\_SUPABASE\_URL=your\_supabase\_url

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your\_anon\_key

SUPABASE\_SERVICE\_ROLE\_KEY=your\_service\_role\_key

# Resend

RESEND\_API\_KEY=your\_resend\_api\_key

# Twilio (optional)

TWILIO\_ACCOUNT\_SID=your\_twilio\_sid

TWILIO\_AUTH\_TOKEN=your\_twilio\_token

TWILIO\_PHONE\_NUMBER=+1234567890

# Stripe

STRIPE\_SECRET\_KEY=your\_stripe\_secret\_key

STRIPE\_PUBLISHABLE\_KEY=your\_stripe\_publishable\_key

STRIPE\_WEBHOOK\_SECRET=your\_webhook\_secret

# Cron Jobs

CRON\_SECRET=your\_random\_secret\_string

# Internal API

INTERNAL\_API\_SECRET=your\_internal\_secret

# Monitoring

SENTRY\_DSN=your\_sentry\_dsn

**Deployment Configuration**

**Vercel Configuration**

{

"buildCommand": "pnpm build",

"devCommand": "pnpm dev",

"installCommand": "pnpm install",

"framework": "nextjs",

"outputDirectory": ".next"

}

**Cron Jobs (vercel.json)**

{

"crons": [

{

"path": "/api/cron/generate-invoices",

"schedule": "0 0 1 \* \*"

},

{

"path": "/api/cron/send-invoice-reminders",

"schedule": "0 9 \* \* \*"

},

{

"path": "/api/cron/reset-daily-counts",

"schedule": "0 0 \* \* \*"

}

]

}

**Testing Strategy**

**Unit Tests**

* Geo-matching functions
* Validation schemas
* Utility functions

**Integration Tests**

* Lead submission flow
* Assignment algorithm
* Claim system (race conditions)
* Invoice generation

**End-to-End Tests**

* Full user journey (submit lead → agency claims)
* Agency registration → settings → claiming
* Invoice generation → payment

**Load Testing**

* 100 concurrent lead submissions
* 50 simultaneous claim attempts
* Database performance under load

**Monitoring & Alerting**

**Critical Alerts**

* Failed lead assignment (no agencies available)
* Email delivery failures
* Database connection errors
* Payment processing failures

**Performance Monitoring**

* API response times
* Database query performance
* Email delivery rates
* Page load times

**Business Metrics**

* Leads per day
* Assignment rate (% of leads assigned)
* Claim rate (% of assigned leads claimed)
* Average claim time
* Revenue per month
* Agency churn rate

**Success Criteria**

**Technical**

* ✅ 99.9% uptime
* ✅ <2s API response times
* ✅ <30s lead assignment time
* ✅ Zero race condition bugs in claiming
* ✅ 100% email delivery rate

**Business**

* ✅ 90%+ leads assigned to agencies
* ✅ 80%+ claimed within 1 hour
* ✅ 95%+ invoice collection rate
* ✅ <2 hours/month human intervention
* ✅ $5,000/month revenue by month 3

**Post-Launch Optimization**

**Month 2**

* A/B test landing page copy
* Optimize Google Ads campaigns
* Add more agency partners
* Implement agency quality scoring

**Month 3**

* Add SMS notifications for immediate leads
* Build mobile app for agencies
* Implement automated quality follow-up
* Add lead feedback loop

**Month 4-6**

* Expand to multiple cities
* Add additional service types (memory care, elder law)
* Implement dynamic pricing based on urgency
* Build affiliate program for referrals

**Security Considerations**

* All sensitive data encrypted at rest (Supabase handles this)
* RLS policies prevent unauthorized access
* Rate limiting on public APIs
* Input sanitization on all user inputs
* CSRF protection (Next.js handles this)
* Secure webhook verification (Stripe)
* Regular security audits
* PCI compliance for payment processing (Stripe handles this)

**Compliance**

* HIPAA considerations (no PHI collected)
* TCPA compliance for SMS (opt-in required)
* CAN-SPAM compliance for emails (unsubscribe links)
* GDPR compliance (data deletion on request)
* Terms of Service
* Privacy Policy

**Notes**

This system is designed for maximum automation and profitability. Once built and tested, it should require minimal human intervention while generating consistent revenue. The key is maintaining high-quality agency partnerships and ensuring fast lead distribution and claiming.