Prospect Finder Background Processing Enhancement

Asynchronous AI Search with Push Notifications

Problem Statement

The current Prospect Finder AI search takes a long time to complete (30-60+ seconds). If the user navigates away from the page or the browser tab loses focus, the search times out and fails (as shown by the "Search failed / Load failed" error). Field sales agents need to be able to initiate a search and continue using the app while the AI curates their prospect list in the background.

Solution Overview

Implement a **background job queue system** that:

1. Accepts search requests and immediately returns a job ID
2. Processes the AI-powered prospect search asynchronously in the background
3. Stores results in the database when complete
4. Sends a **push notification** to the user's device when their prospect list is ready
5. Allows users to view their search history and results anytime

---

User Experience Flow

Initiating a Search

```
1. User enters ZIP code (e.g., 46032)
2. User selects business types (e.g., Fast Food Restaurants)
3. User sets radius (e.g., 5 miles) and results count (e.g., 10 businesses)
```

```
4. User taps "Find Prospects"
5. UI immediately shows:
   - "🔍 Search started! We'll notify you when your prospect list is ready."
   - "You can navigate away - we're searching in the background."
   - [View My Searches] button
6. User can freely navigate to other parts of the app
```

Receiving Results

```
1. Background worker completes AI search (30-90 seconds later)
2. Results saved to database
3. Push notification sent to user's device:

   📱 Notification:
   ┌─────────────────────────────────────┐
   │ 🎯 Prospect List Ready!              │
   │ Found 10 Fast Food Restaurants      │
   │ near 46032                          │
   │ Tap to view your prospects          │
   └─────────────────────────────────────┘

4. User taps notification → App opens directly to search results
```

Viewing Search History

```
1. User can access "My Searches" from Prospect Finder page
2. Shows list of all searches:
   - ✅ Completed: "10 Fast Food Restaurants near 46032" - Tap to view
   - ⏳ In Progress: "Restaurants near 90210" - Started 45 sec ago
   - ❌ Failed: "Retail near 10001" - Tap to retry
3. Completed searches show results immediately (no re-fetching)
```

Technical Implementation

1. Database Schema

Create a new table for prospect search jobs:

```
CREATE TABLE prospect_search_jobs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  organization_id UUID REFERENCES organizations(id),
  user_id UUID REFERENCES users(id),

  -- Search Parameters
  location VARCHAR(20) NOT NULL,           -- ZIP code or city
  location_display VARCHAR(100),           -- Human readable location
  business_types JSONB NOT NULL,           -- Array of MCC codes/types selected
  business_types_display VARCHAR(500),     -- Human readable types
  radius_miles INTEGER NOT NULL,
  max_results INTEGER NOT NULL,

  -- Job Status
  status VARCHAR(20) NOT NULL DEFAULT 'pending',  -- pending, processing, complet
  progress INTEGER DEFAULT 0,              -- 0-100 percentage (optional)
  started_at TIMESTAMP,
  completed_at TIMESTAMP,

  -- Results
  results JSONB,                           -- Array of prospect objects when comp
  result_count INTEGER,

  -- Error Handling
  error_message TEXT,
  retry_count INTEGER DEFAULT 0,

  -- Notification
  notification_sent BOOLEAN DEFAULT FALSE,
  notification_sent_at TIMESTAMP,

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

-- Index for quick lookups
CREATE INDEX idx_prospect_jobs_user_status ON prospect_search_jobs(user_id, statu
CREATE INDEX idx_prospect_jobs_pending ON prospect_search_jobs(status) WHERE stat
```

2. API Endpoints

Create Search Job (Returns Immediately)

```
POST /api/prospect-finder/jobs

Request:
{
  "location": "46032",
  "businessTypes": ["5812", "5813"],  // MCC codes
  "businessTypesDisplay": "Fast Food Restaurants",
  "radiusMiles": 5,
  "maxResults": 10
}

Response (immediate, < 500ms):
{
  "success": true,
  "jobId": "abc-123-def",
  "status": "pending",
  "message": "Search started! We'll notify you when ready.",
  "estimatedSeconds": 45
}
```

Get Job Status

```
GET /api/prospect-finder/jobs/:jobId

Response:
{
  "jobId": "abc-123-def",
  "status": "completed",  // pending | processing | completed | failed
  "progress": 100,
  "resultCount": 10,
  "results": [...],  // Only included if completed
  "createdAt": "2025-01-31T10:24:00Z",
  "completedAt": "2025-01-31T10:25:15Z"
}
```

List User's Search Jobs

```
GET /api/prospect-finder/jobs

Response:
{
  "jobs": [
    {
      "jobId": "abc-123",
      "status": "completed",
      "location": "46032",
      "businessTypesDisplay": "Fast Food Restaurants",
      "radiusMiles": 5,
      "resultCount": 10,
      "createdAt": "...",
      "completedAt": "..."
    },
    {
      "jobId": "def-456",
      "status": "processing",
      "location": "90210",
      "businessTypesDisplay": "Restaurants",
      "progress": 60,
      "createdAt": "..."
    }
  ]
}
```

Retry Failed Job

```
POST /api/prospect-finder/jobs/:jobId/retry

Response:
{
  "success": true,
  "jobId": "abc-123",
  "status": "pending"
}
```

3. Background Worker Implementation

The key is to process the job **outside** of the HTTP request/response cycle. Options:

Option A: Fire-and-Forget with Internal Endpoint (Recommended for Replit)

```
// When job is created, trigger background processing
app.post('/api/prospect-finder/jobs', async (req, res) => {
  // 1. Create job record
  const job = await db.insert(prospectSearchJobs).values({
    userId: req.user.id,
    organizationId: req.user.organizationId,
    location: req.body.location,
    businessTypes: req.body.businessTypes,
    radiusMiles: req.body.radiusMiles,
    maxResults: req.body.maxResults,
    status: 'pending'
  }).returning();
  // 2. Trigger background processing (don't await!)
  fetch(`${process.env.APP_URL}/api/internal/process-prospect-job`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-Internal-Secret': process.env.INTERNAL_SECRET  // Secure internal endpoi
    },
    body: JSON.stringify({ jobId: job[0].id })
  }).catch(err => console.error('Failed to trigger job:', err));
  // Note: No await - this fires and returns immediately

  // 3. Return immediately to user
  res.json({
    success: true,
    jobId: job[0].id,
    status: 'pending',
    message: "Search started! We'll notify you when ready."
  });
});

// Internal endpoint that does the actual work
app.post('/api/internal/process-prospect-job', async (req, res) => {
  // Verify internal secret
  if (req.headers['x-internal-secret'] !== process.env.INTERNAL_SECRET) {
    return res.status(401).json({ error: 'Unauthorized' });
  }

  const { jobId } = req.body;

  // Return immediately - processing happens after response
  res.json({ received: true });

  // Now do the actual work (after response sent)
  try {
    // Update status to processing
    await db.update(prospectSearchJobs)
      .set({ status: 'processing', startedAt: new Date() })
      .where(eq(prospectSearchJobs.id, jobId));

    // Get job details
    const job = await db.query.prospectSearchJobs.findFirst({
      where: eq(prospectSearchJobs.id, jobId)
    });

    // Run the AI prospect search (your existing logic)
    const results = await runAIProspectSearch({
      location: job.location,
      businessTypes: job.businessTypes,
      radiusMiles: job.radiusMiles,
      maxResults: job.maxResults
    });

    // Save results
    await db.update(prospectSearchJobs)
      .set({
        status: 'completed',
        results: results,
        resultCount: results.length,
        completedAt: new Date()
      })
      .where(eq(prospectSearchJobs.id, jobId));

    // Send push notification
    await sendPushNotification(job.userId, {
      title: '🎯 Prospect List Ready!',
      body: `Found ${results.length} ${job.businessTypesDisplay} near ${job.locat
      data: {
        type: 'prospect_search_complete',
        jobId: jobId,
        url: `/prospect-finder/results/${jobId}`
      }
    });

    // Mark notification as sent
    await db.update(prospectSearchJobs)
      .set({ notificationSent: true, notificationSentAt: new Date() })
      .where(eq(prospectSearchJobs.id, jobId));

  } catch (error) {
    // Mark job as failed
    await db.update(prospectSearchJobs)
      .set({
        status: 'failed',
        errorMessage: error.message,
        completedAt: new Date()
      })
      .where(eq(prospectSearchJobs.id, jobId));

    // Optionally notify user of failure
    await sendPushNotification(job.userId, {
      title: '❌ Search Failed',
      body: `We couldn't complete your search. Tap to retry.`,
      data: {
```

```
        type: 'prospect_search_failed',
        jobId: jobId,
        url: `/prospect-finder/jobs`
      }
    });
  }
});
```

Option B: Polling-Based Processing

If the fire-and-forget approach has issues, use a polling worker:

```
// Run every 10 seconds to check for pending jobs
async function processProspectJobQueue() {
  const pendingJobs = await db.query.prospectSearchJobs.findMany({
    where: eq(prospectSearchJobs.status, 'pending'),
    orderBy: [asc(prospectSearchJobs.createdAt)],
    limit: 1  // Process one at a time
  });

  for (const job of pendingJobs) {
    await processJob(job);
  }
}

// Start polling when server starts
setInterval(processProspectJobQueue, 10000);  // Every 10 seconds
```

4. Web Push Notifications Setup

Generate VAPID Keys (One-Time Setup)

```
npx web-push generate-vapid-keys
```

Store in environment variables:

- `VAPID_PUBLIC_KEY`
- `VAPID_PRIVATE_KEY`
- `VAPID_EMAIL` (your contact email)

Push Subscription Storage

```
CREATE TABLE push_subscriptions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id),
  endpoint TEXT NOT NULL,
  keys_p256dh TEXT NOT NULL,
  keys_auth TEXT NOT NULL,
  user_agent TEXT,
  created_at TIMESTAMP DEFAULT NOW(),
  last_used_at TIMESTAMP,

  UNIQUE(user_id, endpoint)
);
```

API Endpoints for Push Subscription

```
// Get VAPID public key (client needs this)
app.get('/api/push/vapid-public-key', (req, res) => {
  res.json({ publicKey: process.env.VAPID_PUBLIC_KEY });
});

// Save push subscription
app.post('/api/push/subscribe', async (req, res) => {
  const { subscription } = req.body;

  await db.insert(pushSubscriptions).values({
    userId: req.user.id,
    endpoint: subscription.endpoint,
    keysP256dh: subscription.keys.p256dh,
    keysAuth: subscription.keys.auth,
    userAgent: req.headers['user-agent']
  }).onConflictDoUpdate({
    target: [pushSubscriptions.userId, pushSubscriptions.endpoint],
    set: { lastUsedAt: new Date() }
  });

  res.json({ success: true });
});

// Unsubscribe
app.post('/api/push/unsubscribe', async (req, res) => {
  const { endpoint } = req.body;

  await db.delete(pushSubscriptions)
    .where(and(
      eq(pushSubscriptions.userId, req.user.id),
      eq(pushSubscriptions.endpoint, endpoint)
    ));

  res.json({ success: true });
});
```

Send Push Notification Helper

```
import webpush from 'web-push';

webpush.setVapidDetails(
  `mailto:${process.env.VAPID_EMAIL}`,
  process.env.VAPID_PUBLIC_KEY,
  process.env.VAPID_PRIVATE_KEY
);

async function sendPushNotification(userId: string, payload: {
  title: string;
  body: string;
  icon?: string;
  badge?: string;
  data?: Record<string, any>;
}) {
  // Get all subscriptions for this user (they might have multiple devices)
  const subscriptions = await db.query.pushSubscriptions.findMany({
    where: eq(pushSubscriptions.userId, userId)
  });

  const notificationPayload = JSON.stringify({
    title: payload.title,
    body: payload.body,
    icon: payload.icon || '/icons/icon-192.png',
    badge: payload.badge || '/icons/badge-72.png',
    data: payload.data || {},
    timestamp: Date.now()
  });

  // Send to all user's devices
  const results = await Promise.allSettled(
    subscriptions.map(sub =>
      webpush.sendNotification(
        {
          endpoint: sub.endpoint,
          keys: {
            p256dh: sub.keysP256dh,
            auth: sub.keysAuth
          }
        },
        notificationPayload
      )
    )
  );

  // Clean up any expired subscriptions
  for (let i = 0; i < results.length; i++) {
    if (results[i].status === 'rejected') {
      const error = results[i].reason;
      if (error.statusCode === 404 || error.statusCode === 410) {
        // Subscription expired or unsubscribed
        await db.delete(pushSubscriptions)
          .where(eq(pushSubscriptions.id, subscriptions[i].id));
      }
    }
  }

  return results;
}
```

5. Service Worker Updates

Update the existing service worker to handle push notifications:

```
// In service-worker.js or sw.js

// Listen for push events
self.addEventListener('push', (event) => {
  if (!event.data) return;

  const data = event.data.json();

  const options = {
    body: data.body,
    icon: data.icon || '/icons/icon-192.png',
    badge: data.badge || '/icons/badge-72.png',
    vibrate: [100, 50, 100],
    data: data.data,
    actions: [
      { action: 'view', title: 'View Results' },
      { action: 'dismiss', title: 'Dismiss' }
    ],
    tag: data.data?.jobId || 'prospect-search',  // Prevent duplicate notificatio
    renotify: true
  };

  event.waitUntil(
    self.registration.showNotification(data.title, options)
  );
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
  event.notification.close();

  if (event.action === 'dismiss') return;

  const urlToOpen = event.notification.data?.url || '/prospect-finder/jobs';

  event.waitUntil(
    clients.matchAll({ type: 'window', includeUncontrolled: true })
      .then((windowClients) => {
        // Check if app is already open
        for (const client of windowClients) {
          if (client.url.includes(self.location.origin) && 'focus' in client) {
            client.focus();
            client.navigate(urlToOpen);
            return;
          }
        }
        // Open new window if not
        if (clients.openWindow) {
          return clients.openWindow(urlToOpen);
        }
      })
  );
});
```

6. Frontend Implementation

Request Notification Permission & Subscribe

```
// hooks/usePushNotifications.ts

import { useEffect, useState } from 'react';

export function usePushNotifications() {
  const [isSubscribed, setIsSubscribed] = useState(false);
  const [isSupported, setIsSupported] = useState(false);

  useEffect(() => {
    setIsSupported('serviceWorker' in navigator && 'PushManager' in window);
    checkSubscription();
  }, []);

  async function checkSubscription() {
    if (!('serviceWorker' in navigator)) return;

    const registration = await navigator.serviceWorker.ready;
    const subscription = await registration.pushManager.getSubscription();
    setIsSubscribed(!!subscription);
  }

  async function subscribe() {
    try {
      // Request permission
      const permission = await Notification.requestPermission();
      if (permission !== 'granted') {
        throw new Error('Notification permission denied');
      }

      // Get VAPID public key
      const response = await fetch('/api/push/vapid-public-key');
      const { publicKey } = await response.json();

      // Subscribe to push
      const registration = await navigator.serviceWorker.ready;
      const subscription = await registration.pushManager.subscribe({
        userVisibleOnly: true,
        applicationServerKey: urlBase64ToUint8Array(publicKey)
      });

      // Send subscription to server
      await fetch('/api/push/subscribe', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ subscription })
      });

      setIsSubscribed(true);
      return true;
    } catch (error) {
      console.error('Failed to subscribe to push:', error);
      return false;
    }
  }

  async function unsubscribe() {
    const registration = await navigator.serviceWorker.ready;
    const subscription = await registration.pushManager.getSubscription();

    if (subscription) {
      await subscription.unsubscribe();
      await fetch('/api/push/unsubscribe', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ endpoint: subscription.endpoint })
      });
    }

    setIsSubscribed(false);
  }

  return { isSupported, isSubscribed, subscribe, unsubscribe };
}

// Helper function
function urlBase64ToUint8Array(base64String: string) {
  const padding = '='.repeat((4 - base64String.length % 4) % 4);
  const base64 = (base64String + padding)
    .replace(/-/g, '+')
    .replace(/_/g, '/');

  const rawData = window.atob(base64);
  const outputArray = new Uint8Array(rawData.length);

  for (let i = 0; i < rawData.length; ++i) {
    outputArray[i] = rawData.charCodeAt(i);
  }
  return outputArray;
}
```

Updated Prospect Finder UI

```
// ProspectFinder.tsx

import { useState } from 'react';
import { useMutation, useQuery } from '@tanstack/react-query';
import { usePushNotifications } from '@/hooks/usePushNotifications';
```

```
export function ProspectFinder() {
  const [location, setLocation] = useState('');
  const [businessTypes, setBusinessTypes] = useState([]);
  const [radius, setRadius] = useState(5);
  const [maxResults, setMaxResults] = useState(10);

  const { isSubscribed, subscribe } = usePushNotifications();

  // Start search job
  const startSearch = useMutation({
    mutationFn: async () => {
      // Ensure push notifications are enabled
      if (!isSubscribed) {
        await subscribe();
      }

      const response = await fetch('/api/prospect-finder/jobs', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          location,
          businessTypes,
          radiusMiles: radius,
          maxResults
        })
      });
      return response.json();
    },
    onSuccess: (data) => {
      // Show success toast
      toast({
        title: "🔍 Search Started!",
        description: "We'll notify you when your prospect list is ready. You can
        action: (
          <Button variant="outline" onClick={() => navigate('/prospect-finder/job
            View My Searches
          </Button>
        )
      });
    }
  });

  // Get user's search jobs
  const { data: jobs } = useQuery({
    queryKey: ['prospect-jobs'],
    queryFn: async () => {
      const response = await fetch('/api/prospect-finder/jobs');
    return response.json();
  }
});

const pendingJobs = jobs?.jobs?.filter(j => j.status === 'pending' || j.status

return (
  <div className="prospect-finder">
    <header>
      <h1>Prospect Finder</h1>
      <span className="badge">AI-Powered</span>
    </header>

    {/* Show pending searches banner */}
    {pendingJobs.length > 0 && (
      <div className="pending-searches-banner">
        <Loader2 className="animate-spin" />
        <span>{pendingJobs.length} search(es) in progress</span>
        <Button variant="link" onClick={() => navigate('/prospect-finder/jobs')
          View
        </Button>
      </div>
    )}

    {/* Search Form */}
    <div className="search-form">
      <div className="field">
        <label>Location</label>
        <input
          type="text"
          value={location}
          onChange={(e) => setLocation(e.target.value)}
          placeholder="ZIP code"
        />
      </div>

      <div className="field">
        <label>Business Types</label>
        <BusinessTypeSelector
          value={businessTypes}
          onChange={setBusinessTypes}
        />
      </div>

      <div className="field-row">
        <div className="field">
          <label>Radius</label>
          <select value={radius} onChange={(e) => setRadius(Number(e.target.val
            <option value={1}>1 mile</option>
            <option value={5}>5 miles</option>
            <option value={10}>10 miles</option>
            <option value={25}>25 miles</option>
          </select>
        </div>

        <div className="field">
          <label>Results</label>
          <select value={maxResults} onChange={(e) => setMaxResults(Number(e.ta
            <option value={5}>5 businesses</option>
            <option value={10}>10 businesses</option>
            <option value={25}>25 businesses</option>
            <option value={50}>50 businesses</option>
          </select>
        </div>
      </div>

      <Button
        onClick={() => startSearch.mutate()}
        disabled={startSearch.isPending || !location || businessTypes.length ==
        className="search-button"
      >
        {startSearch.isPending ? (
          <>
            <Loader2 className="animate-spin" />
            Starting Search...
          </>
        ) : (
          <>
            <Search />
            Find Prospects
          </>
        )}
      </Button>
    </div>

    {/* Empty state when no active search */}
    {!startSearch.isPending && pendingJobs.length === 0 && (
      <div className="empty-state">
        <FileSearch className="icon" />
        <h2>Ready to Find Prospects</h2>
        <p>Enter a ZIP code and select business types to discover new merchants
        <p className="hint">
          💡 Searches run in the background - you'll get a notification when re
        </p>
        </div>
      )}
    </div>
  );
}
```

Search Jobs List Page

```
// ProspectFinderJobs.tsx

export function ProspectFinderJobs() {
  const { data, isLoading, refetch } = useQuery({
    queryKey: ['prospect-jobs'],
    queryFn: async () => {
      const response = await fetch('/api/prospect-finder/jobs');
      return response.json();
    },
    refetchInterval: 5000  // Poll every 5 seconds for status updates
  });

  const retryJob = useMutation({
    mutationFn: async (jobId: string) => {
      const response = await fetch(`/api/prospect-finder/jobs/${jobId}/retry`, {
        method: 'POST'
      });
      return response.json();
    },
    onSuccess: () => refetch()
  });

  if (isLoading) return <LoadingSpinner />;

  const jobs = data?.jobs || [];

  return (
    <div className="prospect-jobs">
      <header>
        <Button variant="ghost" onClick={() => navigate(-1)}>
          <ArrowLeft />
        </Button>
        <h1>My Searches</h1>
      </header>

      {jobs.length === 0 ? (
        <div className="empty-state">
```

```
          <p>No searches yet. Start a new search from Prospect Finder.</p>
          <Button onClick={() => navigate('/prospect-finder')}>
            Find Prospects
          </Button>
        </div>
      ) : (
        <div className="job-list">
          {jobs.map((job) => (
            <div
              key={job.jobId}
              className={`job-card ${job.status}`}
              onClick={() => job.status === 'completed' && navigate(`/prospect-fi
            >
              <div className="job-info">
                <div className="job-title">
                  {job.businessTypesDisplay}
                </div>
                <div className="job-location">
                  📍 {job.location} • {job.radiusMiles} mi radius
                </div>
                <div className="job-time">
                  {formatRelativeTime(job.createdAt)}
                </div>
              </div>

              <div className="job-status">
                {job.status === 'pending' && (
                  <span className="status pending">
                    <Clock className="icon" />
                    Queued
                  </span>
                )}
                {job.status === 'processing' && (
                  <span className="status processing">
                    <Loader2 className="icon animate-spin" />
                    Searching...
                  </span>
                )}
                {job.status === 'completed' && (
                  <span className="status completed">
                    <CheckCircle className="icon" />
                    {job.resultCount} found
                    <ChevronRight />
                  </span>
                )}
                {job.status === 'failed' && (
                  <div className="status failed">
```

```
                    <XCircle className="icon" />
                    Failed
                    <Button
                      size="sm"
                      variant="outline"
                      onClick={(e) => {
                        e.stopPropagation();
                        retryJob.mutate(job.jobId);
                      }}
                    >
                      Retry
                    </Button>
                  </div>
                )}
              </div>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}
```

Search Results Page

```
// ProspectFinderResults.tsx

export function ProspectFinderResults() {
  const { jobId } = useParams();

  const { data: job, isLoading } = useQuery({
    queryKey: ['prospect-job', jobId],
    queryFn: async () => {
      const response = await fetch(`/api/prospect-finder/jobs/${jobId}`);
      return response.json();
    }
  });

  if (isLoading) return <LoadingSpinner />;
  if (!job) return <NotFound />;

  return (
    <div className="prospect-results">
      <header>
        <Button variant="ghost" onClick={() => navigate('/prospect-finder/jobs')}
```

```
          <ArrowLeft />
        </Button>
        <div>
          <h1>{job.businessTypesDisplay}</h1>
          <p>📍 {job.location} • {job.radiusMiles} mi • {job.resultCount} results
        </div>
      </header>

      <div className="results-list">
        {job.results?.map((prospect, index) => (
          <ProspectCard
            key={index}
            prospect={prospect}
            onConvertToDeal={() => {/* existing convert logic */}}
          />
        ))}
      </div>
    </div>
  );
}
```

Notification Permission Flow

First-Time Setup

When user first tries to search, prompt for notification permission:

```
function NotificationPermissionPrompt({ onAllow, onSkip }) {
  return (
    <div className="notification-prompt">
      <Bell className="icon" />
      <h3>Enable Notifications</h3>
      <p>
        Get notified when your prospect search is complete.
        You can search in the background while using other features.
      </p>
      <div className="actions">
        <Button onClick={onAllow}>Enable Notifications</Button>
        <Button variant="ghost" onClick={onSkip}>Not Now</Button>
      </div>
    </div>
  );
}
```

Settings Page Option

Add to Profile/Settings:

```
function NotificationSettings() {
  const { isSupported, isSubscribed, subscribe, unsubscribe } = usePushNotificati

  if (!isSupported) {
    return <p>Push notifications are not supported on this device.</p>;
  }

  return (
    <div className="setting-row">
      <div>
        <label>Push Notifications</label>
        <p>Get notified when prospect searches complete</p>
      </div>
      <Switch
        checked={isSubscribed}
        onCheckedChange={(checked) => checked ? subscribe() : unsubscribe()}
      />
    </div>
  );
}
```

Error Handling

Job Failures

- Retry up to 3 times automatically with exponential backoff
- After 3 failures, mark as failed and notify user
- User can manually retry from the jobs list

Network Issues

- If notification fails to send, job still marked complete
- User can see completed jobs in their list
- Add in-app notification center as backup

Testing Checklist

1. Start a search and navigate away immediately
2. Verify job appears in “My Searches” with pending/processing status
3. Verify push notification received when complete
4. Tap notification and verify it opens results page
5. Test with app fully closed (not just backgrounded)
6. Test on iOS Safari (may need PWA install for notifications)
7. Test retry for failed jobs
8. Test multiple concurrent searches
9. Verify results persist and can be viewed later

---

iOS Safari Considerations

Push notifications on iOS Safari require:

- App must be added to Home Screen (PWA)
- User must explicitly enable notifications in iOS Settings
- Only works on iOS 16.4+

Show appropriate guidance to iOS users:

```
function iOSPushGuide() {
  return (
    <div className="ios-guide">
      <p>To receive notifications on iPhone:</p>
      <ol>
        <li>Tap the Share button <Share /> in Safari</li>
        <li>Select "Add to Home Screen"</li>
        <li>Open the app from your Home Screen</li>
        <li>Enable notifications when prompted</li>
      </ol>
    </div>
  );
}
```

Dependencies to Add

```
npm install web-push
```

Add to environment variables:

```
VAPID_PUBLIC_KEY=<generated>
VAPID_PRIVATE_KEY=<generated>
VAPID_EMAIL=support@pcbancard.com
INTERNAL_SECRET=<random-secure-string>
```

Summary

This enhancement transforms the Prospect Finder from a blocking synchronous search into an asynchronous background job system with push notifications. Sales agents can:

1. Start a search in 2 seconds
2. Continue using the app (or close it entirely)
3. Receive a push notification when results are ready
4. Tap notification to go directly to results
5. View search history anytime

This dramatically improves the user experience for field agents who can't wait 60+ seconds staring at a loading screen.

End of Specification