BBB Directory Bot - Product Requirements Document (PRD)

1. Executive Summary

A vector-based intelligent directory search system for BBB chapters, enabling natural language queries to find member businesses, products, and services. MVP focuses on single chapter (Bhagyanagar) with architecture ready for 70 chapters and 4100 members.

2. Core Architecture

Frontend (Web Bot) → n8n Workflows → OpenAI/Claude → Supabase (Vector DB)
↓
Query Cache (30 days)

3. Database Schema (Final Structure)

```
-- 1. Profiles table with vector support
CREATE TABLE profiles (
    id SERIAL PRIMARY KEY,
    chapter VARCHAR(100) NOT NULL,
    chapter_id INTEGER,
    city VARCHAR(100),
    raw_data TEXT NOT NULL,
    structured_data JSONB,
    tags TEXT[],
    embedding vector(1536),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    is_active BOOLEAN DEFAULT true
);

CREATE INDEX idx_profiles_embedding ON profiles USING ivfflat (embedding
vector_cosine_ops);
CREATE INDEX idx_profiles_chapter ON profiles(chapter);
CREATE INDEX idx_profiles_chapter_embedding ON profiles(chapter, embedding);

-- 2. Query cache with 30-day TTL
CREATE TABLE query_cache (
    query_text TEXT PRIMARY KEY,
```

```
    query_normalized TEXT,
    chapter VARCHAR(100),
    embedding vector(1536),
    results JSONB,
    hit_count INTEGER DEFAULT 1,
    created_at TIMESTAMP DEFAULT NOW(),
    expires_at TIMESTAMP DEFAULT NOW() + INTERVAL '30 days'
);

CREATE INDEX idx_cache_expires ON query_cache(expires_at);
CREATE INDEX idx_cache_chapter ON query_cache(chapter);

-- 3. Semantic clusters for intelligent expansion
CREATE TABLE semantic_clusters (
    id SERIAL PRIMARY KEY,
    primary_term VARCHAR(100),
    related_terms TEXT[],
    cluster_embedding vector(1536),
    category VARCHAR(100)
);

-- 4. System logs for monitoring
CREATE TABLE system_logs (
    id SERIAL PRIMARY KEY,
    log_type VARCHAR(50),
    workflow VARCHAR(100),
    chapter VARCHAR(100),
    query_text TEXT,
    results_count INTEGER,
    confidence_score FLOAT,
    error_message TEXT,
    user_session VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW()
);

-- 5. Chapters table (for future scaling)
CREATE TABLE chapters (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE,
    city VARCHAR(100),
    state VARCHAR(100),
    member_count INTEGER DEFAULT 0,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT NOW()
```

```
);

-- 6. Vector search function
CREATE OR REPLACE FUNCTION vector_search(
    query_embedding vector,
    search_chapter VARCHAR DEFAULT NULL,
    limit_count INTEGER DEFAULT 5
)
RETURNS TABLE(
    id int,
    chapter text,
    raw_data text,
    similarity float
)
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN QUERY
  SELECT
    p.id,
    p.chapter::text,
    p.raw_data::text,
    (1 - (p.embedding <=> query_embedding))::float as similarity
  FROM profiles p
  WHERE
    p.is_active = true
    AND (search_chapter IS NULL OR p.chapter = search_chapter)
  ORDER BY
    CASE WHEN p.chapter = search_chapter THEN 0 ELSE 1 END,
    p.embedding <=> query_embedding
  LIMIT limit_count;
END;
$$;

-- 7. Cache cleanup function
CREATE OR REPLACE FUNCTION cleanup_expired_cache()
RETURNS void AS $$
BEGIN
  DELETE FROM query_cache WHERE expires_at < NOW();
END;
$$ LANGUAGE plpgsql;
```

4. n8n Workflows

Workflow 1: Profile Ingestion

Endpoint: /webhook/profile-ingest
Flow: Webhook → OpenAI Embedding → Store in Supabase

Input:

```
{
  "chapter": "Bhagyanagar",
  "profile_text": "Unstructured profile description..."
}
```

Workflow 2: Search (with Cache)

Endpoint: /webhook/search
Flow:
1. Webhook → Check Cache
2. If Cache Miss → OpenAI Embedding → Vector Search → Store in Cache
3. If Cache Hit → Return Cached Results
4. If Low Confidence (<0.7) → Semantic Expansion → Re-search

Input:

```
{
  "query": "networking equipment",
  "chapter": "Bhagyanagar" // optional, defaults to current chapter
}
```

Output:

```
{
  "success": true,
  "query": "networking equipment",
  "results_found": 3,
  "profiles": [
    {
      "rank": 1,
      "chapter": "Bhagyanagar",
      "profile": "Profile text...",
      "similarity_score": "88.3%"
    }
  ],
  "from_cache": false
```

}

5. Semantic Clustering (Implementation)

Pre-populate clusters:

INSERT INTO semantic_clusters (primary_term, related_terms, category) VALUES
('networking', ARRAY['LAN', 'WAN', 'WiFi', 'router', 'switch', 'firewall', 'ethernet', 'broadband', 'internet', 'connectivity'], 'Technology'),
('legal', ARRAY['lawyer', 'advocate', 'attorney', 'contract', 'agreement', 'court', 'litigation', 'dispute'], 'Professional Services'),
('ayurvedic', ARRAY['ayurveda', 'herbal', 'natural', 'wellness', 'organic', 'traditional', 'holistic', 'medicine'], 'Healthcare'),
('software', ARRAY['application', 'app', 'program', 'IT', 'computer', 'technology', 'digital', 'coding'], 'Technology'),
('accounting', ARRAY['GST', 'tax', 'finance', 'bookkeeping', 'audit', 'CA', 'chartered accountant', 'filing'], 'Professional Services');

6. Implementation Phases

MVP Phase (Current)

- ✅ Vector storage and embeddings
- ✅ Basic search with OpenAI
- ✅ Profile ingestion
- ⬜ Query caching (30 days)
- ⬜ Semantic clustering
- ⬜ Error logging

Phase 2 (Post-MVP)

- ⬜ Multi-chapter search
- ⬜ Feedback collection
- ⬜ Advanced filtering (city, tags)
- ⬜ Bulk profile upload

Phase 3 (Scale)

- ⬜ 70 chapters support
- ⬜ 4100+ profiles
- ⬜ Analytics dashboard
- ⬜ Auto-semantic learning

7. Key Design Decisions

1. **Embedding Model**: OpenAI text-embedding-ada-002 (1536 dimensions)
2. **Cache Duration**: 30 days (configurable)
3. **Search Scope**: Default to current chapter, option for all
4. **Similarity Threshold**: 0.7 for high confidence
5. **Result Limit**: 5 profiles per search (configurable)

8. Performance Targets

- Search response: < 2 seconds
- Cache hit rate: > 70%
- Similarity accuracy: > 80%
- System uptime: 99.9%

9. Cost Optimization

- Cache reduces OpenAI calls by 70-80%
- Semantic clustering reduces failed searches
- Batch embeddings for bulk uploads

Next Steps

1. **Run the complete SQL schema** in Supabase
2. **Update n8n workflows** to include caching
3. **Populate semantic clusters**
4. **Test with real data**

Shall we proceed with implementing the cache logic in your n8n workflows?

BBB Directory Bot - Updated PRD & Current Status

Project Overview

An AI-powered business directory system for BBB chapters using vector embeddings, natural language search, and intelligent caching. Currently implementing MVP for Bhagyanagar chapter with architecture supporting 70 chapters and 4100 members.

Database Schema (Current State)

```
-- 1. Profiles table with vector support
CREATE TABLE profiles (
    id SERIAL PRIMARY KEY,
    mobile_number VARCHAR(15) UNIQUE,
    company_name VARCHAR(255),
    member_number VARCHAR(50),
    email_id VARCHAR(255),
    alternate_number VARCHAR(15),
    alternate_email VARCHAR(255),
    website VARCHAR(255),
    whatsapp_number VARCHAR(15),
    chapter VARCHAR(100) NOT NULL,
    chapter_id INTEGER,
    city VARCHAR(100),
    raw_data TEXT NOT NULL,
    structured_data JSONB,
    suggested_keywords TEXT[],
    approved_keywords TEXT[],
    tags TEXT[],
    embedding vector(1536),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    is_active BOOLEAN DEFAULT true
);

-- 2. Query cache with 30-day TTL
CREATE TABLE query_cache (
    query_text TEXT,
    query_normalized TEXT,
    chapter VARCHAR(100) DEFAULT 'Bhagyanagar',
    embedding vector(1536),
    results JSONB,
    hit_count INTEGER DEFAULT 1,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    expires_at TIMESTAMP DEFAULT NOW() + INTERVAL '30 days',
    is_active BOOLEAN DEFAULT true,
    CONSTRAINT unique_query_chapter UNIQUE (query_normalized, chapter)
);

-- 3. Semantic clusters for intelligent expansion
CREATE TABLE semantic_clusters (
```

```
    id SERIAL PRIMARY KEY,
    primary_term VARCHAR(100) UNIQUE,
    related_terms TEXT[],
    cluster_embedding vector(1536),
    category VARCHAR(100),
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT NOW()
);

-- 4. System logs (not yet implemented)
CREATE TABLE system_logs (
    id SERIAL PRIMARY KEY,
    log_type VARCHAR(50),
    workflow VARCHAR(100),
    chapter VARCHAR(100),
    query_text TEXT,
    results_count INTEGER,
    confidence_score FLOAT,
    error_message TEXT,
    user_session VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW()
);

-- 5. Vector search function
CREATE OR REPLACE FUNCTION vector_search(query_embedding vector)
RETURNS TABLE(
    id int,
    chapter text,
    raw_data text,
    similarity float
)
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN QUERY
  SELECT
    p.id,
    p.chapter::text,
    p.raw_data::text,
    (1 - (p.embedding <=> query_embedding))::float as similarity
  FROM profiles p
  WHERE p.is_active = true
  ORDER BY p.embedding <=> query_embedding
  LIMIT 5;
```

```
END;
$$;
```

Workflow 1: Profile Ingestion

Endpoint: `https://n8n.srv1017206.hstgr.cloud/webhook-test/profile-ingest`

Nodes:

1. **Webhook** - Receives profile data
2. **Code (Prepare Embedding)** - Formats for OpenAI
3. **HTTP Request (OpenAI Embedding)** - Gets vector embedding
4. **Code (Prepare Keywords Request)** - Formats for keyword extraction
5. **HTTP Request (OpenAI Keywords)** - Extracts keywords
6. **Code (Parse Keywords)** - Parses JSON response
7. **Code (Prepare Upsert Data)** - Combines all data, extracts mobile number
8. **HTTP Request (Upsert Profile)** - Stores in Supabase
9. **HTTP Request (Trigger Clusters)** - Calls semantic cluster workflow

Workflow 2: Semantic Cluster Generation

Endpoint:
`https://n8n.srv1017206.hstgr.cloud/webhook-test/generate-clusters`

Nodes:

1. **Webhook** - Receives profile text and keywords
2. **Code (Prepare Cluster Request)** - Creates OpenAI prompt
3. **HTTP Request (OpenAI)** - Generates semantic clusters
4. **Code (Parse Clusters)** - Parses JSON response
5. **Loop Over Items** - Processes each cluster
6. **HTTP Request (Upsert Cluster)** - Stores each cluster in Supabase

Workflow 3: Search (WITH CACHING - IN PROGRESS)

Endpoint: `https://n8n.srv1017206.hstgr.cloud/webhook-test/search`

Current Issues:

1. Cache hit path incorrectly continues to "Store in Cache"
2. Cache miss with empty array stops execution

3. Return Cached Results node has data structure issues

Intended Flow:

```
Webhook
  ↓
Check Cache (HTTP GET to query_cache)
  ↓
IF (Cache Decision)
  ├─[TRUE: Cache Hit]→ Return Cached Results → END
  └─[FALSE: Cache Miss]→ Prepare Query → OpenAI Embedding →
                         Prepare Vector Search → Supabase Search →
                         Format Results → Prepare Cache Entry →
                         Store in Cache → END
```

Current Status: 10/14 Tasks

Completed ✅:

1. Database structure with pgvector
2. Profile ingestion with embeddings
3. Mobile-based duplicate prevention
4. Keyword extraction (AI-powered)
5. Semantic cluster generation (AI-powered)
6. Vector similarity search
7. Basic search workflow
8. Company/contact fields
9. Multi-chapter ready structure
10. Cache table structure

In Progress 🔄:

11. Query caching implementation (90% - fixing flow issues)

Pending ⏳:

12. System logging
13. Web Bot UI
14. Load 50 real profiles

Next Steps:

1. Fix cache workflow branching issue
2. Test complete flow with multiple queries
3. Build web interface
4. Deploy to production

API Test Commands:

```
# Profile ingestion
curl -X POST https://n8n.srv1017206.hstgr.cloud/webhook-test/profile-ingest \
-H "Content-Type: application/json" \
-d '{
  "chapter": "Bhagyanagar",
  "company_name": "Company Name",
  "profile_text": "Full profile description with mobile 9XXXXXXXXX"
}'

# Search (should use cache)
curl -X POST https://n8n.srv1017206.hstgr.cloud/webhook-test/search \
-H "Content-Type: application/json" \
-d '{"query": "networking equipment", "chapter": "Bhagyanagar"}'
```

The main issue now is fixing the search workflow's cache branching logic.