**Master Plan: AI-Gedreven Locatie-App voor de Turkse Diaspora (MVP)**

**1.0 Project Foundation**

**1.1 Core Goal & Vision**  
To develop a lightweight, legal, and cost-efficient application that automatically discovers, validates, and displays Turkish-oriented businesses and community hotspots in the Netherlands. The system relies on AI for classification and enrichment, with a "human-in-the-loop" for corrections, ensuring high data quality.

**1.2 MVP Scope (Included)**

* **Geography:** Start with **Rotterdam**; rollout to 2-3 additional cities during the pilot phase.
* **Business Categories:** Restaurants, supermarkets, bakeries, barbershops, services (travel agencies, healthcare, mosques).
* **Data Sources:** **Google Places API** and **Google Custom Search API** (legal methods), supplemented by manual user submissions.
* **AI Tasks:** Classification (relevant/not), categorization, data enrichment (rating, URL), and confidence scoring.
* **Frontend:** A Map and List view with category-specific markers and status badges.
* **Backend & Admin:** A simple CMS for admin actions, comprehensive logging, and an audit trail.

**1.3 Out of Scope (Post-MVP)**

* User accounts, likes, and reviews.
* In-app payments and advertisements.
* Advanced personalization features.

**1.4 Guiding Principles**

* **Legal Compliance:** Strictly no scraping; only use licensed APIs.
* **Privacy-Aware:** Minimize data collection; no PII in logs.
* **Cost-Efficient:** Implement quota management, caching, and batch processing.
* **Idempotent & Robust:** Operations can be repeated safely; system handles failures gracefully.
* **Observability-First:** Logging and metrics from day one.
* **Human-in-the-Loop:** Manual admin oversight for corrections and quality control.

**2.0 Technical Architecture & Stack**

| **Layer** | **Purpose** | **Technology** | **Hosting** |
| --- | --- | --- | --- |
| **Data & Auth** | Database, optional authentication | PostgreSQL (via Supabase) | Supabase (Free) |
| **Backend/API** | Core logic, orchestration, AI control, CMS routes | Python + FastAPI | Render (Free) |
| **AI Layer** | Classification, enrichment, scoring | OpenAI API (GPT-4.1-mini / GPT-4-Turbo) | Via Backend |
| **Search/Discovery** | Legal data sourcing | Google Places API + Google Custom Search API | Google Cloud |
| **Frontend** | Map/List UI for end-users | React + Leaflet + (Vite or Next.js) | GitHub Pages/Vercel |
| **Automation** | Scheduled tasks & queue processing | Render Cron + Database-backed queue | Render + Supabase |

**Environments:** Development → Staging → Production (with separate Supabase databases and API keys).

**3.0 Data Model (PostgreSQL)**

**Core Tables:**

* locations: Main table for all business data.
  + *Identity:* id, place\_id, source
  + *Core:* name, address, lat, lng, category, business\_status
  + *Reputation:* rating, user\_ratings\_total
  + *Lifecycle:* state (CANDIDATE, PENDING\_VERIFICATION, VERIFIED, SUSPENDED, RETIRED), confidence\_score, is\_probable\_not\_open\_yet
  + *Freshness:* first\_seen\_at, last\_seen\_at, last\_verified\_at, next\_check\_at, freshness\_score
  + *Audit:* evidence\_urls[], notes, is\_retired
* ai\_logs: Audit trail for all AI actions.
* tasks: Queue for orchestration (discovery, verification, monitoring).
* training\_data: Gold records for active learning.
* category\_icon\_map: Mapping of business types to frontend categories and icons.

**4.0 AI & Data Processing Design**

**4.1 AI Tasks**

1. **Classification:** Is this business relevant for the Turkish diaspora? Predict category.
   * *Input:* Name, category/type, address, snippet.
   * *Output:* {action: 'keep'/'ignore', category, score}
2. **Enrichment:** Extract website, rating, business status, and opening hours from search results.
   * *Output:* {rating, user\_ratings\_total, business\_status, website, evidence\_urls[]}
3. **Confidence Scoring:** A combined score based on data source, consistency, review volume, and recency.
   * *Output:* confidence\_score (0..1)

**4.2 Quality Assurance**

* **JSON Schema Enforcement:** All AI outputs are validated against a schema; invalid responses trigger a retry.
* **Few-Shot Learning:** Use deterministic examples with Dutch/Turkish patterns.
* **Test Set:** Maintain a gold set from training\_data for regression testing.

**Acceptance Criteria:** Classification precision ≥ 90% on pilot set; JSON validation errors < 1%.

**5.0 System Automation ("The Robot Layer")**

**5.1 Automated Workers (Bots)**

* **DiscoveryBot:** Periodically searches Google Places/APIs on a grid for target cities/categories. Deduplicates and creates new CANDIDATE records.
* **VerificationBot:** Fetches Place Details for candidates, enriches data, and decides on state (VERIFIED/SUSPENDED).
* **MonitorBot:** Re-checks records where next\_check\_at is due, updates business\_status, and manages the record lifecycle.
* **LearningBot:** Updates AI prompts and word lists based on new training\_data and errors.

**5.2 Data Freshness Policy**

* **New + Low Confidence:** 3-7 days
* **"Probable Not Open Yet":** 14 days
* **Verified (few reviews):** 30 days
* **Verified (many reviews):** 60-90 days
* **Temporarily Closed / User-Flagged:** 7-14 days / Immediate

**Acceptance Criteria:** No record is older than 90 days without a re-check; the task queue is processed within 24 hours.

**6.0 Backend API & Services**

**6.1 Key Endpoints**

* GET /api/v1/locations?lat=&lng=&radius=&category= → Returns verified locations.
* POST /admin/add-location → Manually add a gold record.
* POST /pipeline/run → Manually trigger bot workflows (discovery/verification).
* GET /health, GET /version → Monitoring.

**6.2 Logical Services**

* openai\_service: Manages prompts, API calls, and JSON validation.
* google\_service: Handles all Google API calls with field masks and caching.
* db\_service: Manages CRUD operations with idempotent upserts.
* learning\_service: Updates few-shot examples based on training\_data.

**7.0 Frontend Application**

**7.1 Views & Components**

* **MapView:** Leaflet-based map with clustered, category-colored markers.
* **LocationList:** Sortable (distance, rating) and filterable (category) list.
* **LocationCard:** Displays name, category, rating, and a **status badge** ("New", "Coming Soon", "Temporarily Closed", "Closed").
* **Language Toggle:** Switch between Dutch and Turkish UI labels.

**7.2 UX Rules**

* Markers are grey if is\_probable\_not\_open\_yet is true.
* Transparency: Show "Last verified: X days ago".
* Fallbacks: "No rating yet", "Coming Soon".

**Acceptance Criteria:** Loads in < 2s on broadband; mobile-friendly; markers and badges display correctly.

**8.0 Admin & Content Management System (Light)**

**8.1 Admin Functions**

* Manually add, correct, or change the status of locations.
* Add notes to records.
* Export data to CSV for audit.
* View a small dashboard with record counts per state and recent errors.

**Acceptance Criteria:** An admin can correct a data error in < 2 minutes; full audit trail is maintained.

**9.0 Security, Privacy & Compliance**

* **No Scraping:** Only use official, licensed APIs.
* **Secrets Management:** API keys are stored server-side only (in environment variables).
* **Rate Limiting:** Implemented server-side with retries and exponential backoff.
* **Privacy:** No PII logging; data minimization principles.
* **Accessibility:** Basic WCAG compliance (contrast, keyboard navigation).

**10.0 Observability, Quality & Cost Control**

**10.1 Monitoring**

* **Metrics:** New candidates/week, conversion rate to VERIFIED, task error rates, API latency.
* **Alerting:** Email/Webhook for task failure spikes or API quota errors (429s).
* **KPIs:** Data coverage (≥200 verified in pilot city), classification precision (≥90%), freshness coverage.

**10.2 Cost Management**

* Limit batch sizes (100-200/run).
* Use Fields Mask in Google APIs to request only necessary data.
* Implement caching and a data freshness policy to avoid unnecessary API calls.
* Use the most cost-effective AI models (e.g., GPT-4.1-mini) with concise prompts.

**Acceptance Criteria:** MVP pilot costs < ~€25/month; no quota bursts.

**11.0 Project Execution & Roadmap**

**11.1 indicative Timeline (6-8 Weeks)**

* **Week 1 - Foundation:** Setup repos, environments, skeleton backend, frontend scaffold.
* **Week 2 - Data Ingest:** Implement legal discovery via Google APIs for Rotterdam.
* **Week 3 - AI Core:** Build and test classification and verification steps.
* **Week 4 - Frontend:** Develop and refine the Map and List views.
* **Week 5 - Automation:** Implement the MonitorBot and admin CMS.
* **Week 6 - Hardening:** Setup observability, alerts, rate limiting, and finalize content.
* **Weeks 7-8 - Pilot Rollout:** Add 2 more cities and fine-tune the system.

**11.2 Epics & Sample Stories (Simplified)**

1. **Epic: Foundation & Infra**
   * *Story:* Setup Supabase projects and tables. (DoD: All tables with constraints exist).
2. **Epic: Legal Discovery**
   * *Story:* DiscoveryBot grid-search for Rotterdam. (DoD: ≥500 candidates, no duplicates).
3. **Epic: AI Core**
   * *Story:* Implement classification with JSON validation. (DoD: 95% valid JSON, <3% retries).
4. **Epic: Frontend**
   * *Story:* MapView with markers and status badges. (DoD: 5 category icons display correctly).
5. **Epic: Continuous Monitoring**
   * *Story:* Implement freshness scheduler. (DoD: No record >90 days without a re-check).

**12.0 Repository Structure (Monorepo)**

text

turkish-diaspora-app/

├── Backend/ # FastAPI Application

│ ├── app/

│ │ ├── api/routers/ # (health, locations, admin, pipeline)

│ │ ├── core/ # (config, logging, security)

│ │ ├── models/ # Pydantic schemas

│ │ ├── services/ # (db, openai, google, learning)

│ │ └── workers/ # (discovery, verification, monitor, learning bots)

│ ├── tests/

│ └── requirements.txt

├── Frontend/ # React Application (Vite or Next.js)

│ ├── src/

│ │ ├── components/ # (MapView, LocationList, LocationCard, Filters)

│ │ ├── lib/ # (api, geo helpers)

│ │ ├── hooks/ # (useLocationData, useUserPosition)

│ │ └── i18n/ # (nl.json, tr.json)

│ └── package.json

├── Docs/ # (plan, architecture, api-spec, runbooks)

├── Infra/

│ ├── supabase/ # SQL migration files

│ └── render.yaml # Deployment config

└── .github/workflows/ # CI/CD pipelines

**13.0 Risks & Mitigation**

| **Risk** | **Mitigation** |
| --- | --- |
| **API Quota Limits / 429s** | Use field masks, caching, batch limits, and exponential backoff. |
| **Incorrect AI Classification** | Use gold training data, test sets, confidence thresholds, and human review. |
| **Outdated Business Info** | Implement and enforce the data freshness policy with the MonitorBot. |
| **API Key Exposure** | Keys stored server-side only, in environment variables; regular rotation. |
| **Legal/Compliance Issues** | Strict no-scraping policy; publish privacy statement; data minimization. |

This master plan consolidates all the critical information from your document into a single, actionable guide. It provides the strategic overview, technical depth, and project management framework needed to execute the project successfully.

Would you like me to generate any specific artifacts from this plan, such as the detailed file structure with placeholder files, the OpenAPI specification outline, or the SQL for the database tables?