



Mega-Directory System Codex Tasks

Initial Implementation Tasks

- **T01: Setup Project Scaffolding and Configuration** – Initialize the code repository and development environment with the chosen web framework (e.g. Django/Flask or Node.js) and required libraries. Configure environment variables for database connections, API keys (for geocoding), and routing settings.
- **T02: Design Directory Page Model** – Create a `DirectoryPage` data model that supports custom category and location groupings as well as SEO metadata fields (title, description, keywords). Ensure each page can be managed manually and mapped via configuration to a subdomain or subdirectory.
- **T03: Implement Config-Driven Routing** – Develop routing logic that reads configuration (e.g. YAML or JSON) to map incoming requests to directory pages by subdomain or URL path. Support both subdomain-based (e.g. `category.example.com`) and subdirectory-based (e.g. `example.com/category/`) routing.
- **T04: Define Category Model and Endpoints** – Implement a `Category` model/schema to represent listing categories. Include fields for hierarchy (parent/child categories) if needed. Create API endpoints (CRUD) for managing categories.
- **T05: Define Listing Model with Multi-Address Support** – Develop a `Listing` model to store business or place information. Include support for multiple associated addresses per listing. Establish relations between listings and categories, and between listings and addresses.
- **T06: Define Address Model and Linkages** – Create an `Address` model to capture street address details (street, city, state, postal code, country). Link addresses to listings (one-to-many for multi-address listings). Ensure address validation and normalization.
- **T07: Implement Global Location Hierarchy** – Build a location hierarchy with models for Country, State/Province, City, and Postal Code. Populate or integrate this data globally. Implement fallback logic to handle countries without postal codes or states (for example, treat city-level or country-level as fallback when subdivisions are absent).
- **T08: Integrate Geolocation Service** – Develop a geocoding service layer that first attempts to geolocate addresses via `geocode.maps.co`. If that fails or is disabled, fall back to the Google Maps Geocoding API. Make the choice of primary vs. fallback configurable via environment variables.
- **T09: Implement Listing Geocoding Logic** – On creation or update of an address/listing, use the geolocation service to auto-populate latitude and longitude fields. Handle API rate limiting and errors gracefully. Provide a manual override option in admin if geocoding fails.
- **T10: Build RESTful API Server** – Set up the API server with full CRUD operations for Directory Pages, Categories, Listings, and Addresses. Ensure proper routing, validation, and error handling. Include endpoints to query listings by category, location, or directory page.
- **T11: Develop Python Crawler Framework** – Create a standalone Python project for the scraper/ingestion pipeline. Set up a job structure (e.g. using scripts or a simple CLI) that can fetch data from various sources independently of the web app.
- **T12: Implement Scraping Modules (HTML/JSON/Text)** – Using BeautifulSoup (for HTML) and JSON/text parsing, write scraper modules to ingest listings from different formats. These should extract relevant fields (name, description, address, etc.) from input sources.

- **T13: Integrate LLM-based Enrichment (OpenRouter)** – (Optional) Hook into a free LLM endpoint via OpenRouter to clean, normalize, or enrich scraped data (e.g. summarizing descriptions, extracting metadata). Ensure this runs as part of the ingestion pipeline.
- **T14: Create Data Ingestion Pipeline** – Connect the scraping modules to the API or database: after scraping (and optional enrichment), transform and push data into the system's Listings/Addresses via API calls or direct DB operations. Include logging of successes/failures.
- **T15: Build Admin Moderation Interface** – Develop a web-based admin UI for reviewing incoming listings. Organize pending listings by their associated directory page. Display listing details and scraped source info to the admin user.
- **T16: Implement Batch Approval/Reject in Admin** – In the admin interface, add the ability to select multiple listings and approve or reject them in bulk. Track the review status (approved, rejected, pending) on each listing. Notify submitters or log actions accordingly.

Phase 2 Tasks (User Features)

- **T17: Implement User Authentication (Magic Links and Credentials)** – Create a user account system. Support frictionless login via magic links sent to email and traditional email/password login. Secure the process with token expiration and encryption.
- **T18: Develop User Collections (Favorites Lists)** – Allow logged-in users to create personal collections or “lists” of favorite listings. Implement CRUD operations for these collections, and endpoints/UI for adding/removing listings to a user’s collections.
- **T19: Add Listing Voting/Ratings** – Enable users to upvote or rate listings. Design a voting or rating model and attach it to users and listings. Ensure endpoints and UI for casting votes and displaying aggregate scores. Prevent duplicate votes.
- **T20: Implement Listing Reporting Mechanism** – Let users report listings for issues (e.g. spam, outdated info). Create a **Report** model to log user reports with reasons. Add admin views to review and act on reported listings (e.g. follow up or remove the listing).

Each task above is identified by a unique ID (T01-T20) and covers a distinct piece of the system as specified, from initial setup and core features through the planned Phase 2 user enhancements. The tasks are organized to reflect logical development flow and separate the first-phase requirements from the later user-focused features.
