# **Singletap-Backend Complete Technical Documentation**

### **Vriddhi Matching Engine v2.0**

**Auto-generated | February 16, 2026**

## Table of Contents

1. [Executive Summary](#X102b2e9de4183984340b81b0bb402a984d3925a)
2. [Technology Stack Overview](#X8c6480821357b7d808defa9c0293fa2cfc9e273)
3. [Python Libraries - Complete Reference](#X6fda8658f795d27ae199ccfb9b3c53e58188ec4)
4. [External APIs Used](#Xd57eed8d0af57f088185a62d2fdea4374a4b70a)
5. [Internal API Endpoints](#X25fa9ce167c4cefbcc6cef5dc3dcbd553874d67)
6. [System Architecture](#X2243a06a074ccfd75fe27328468a1f632d3335b)
7. [Core Components Deep Dive](#Xee2522a0c045e8036af1d4102e28c91fe612544)
8. [Database Architecture](#X514b9a285c079658ecca005795166a1162b1857)
9. [ML Pipeline](#X4fd58185c779897901e15f57fcb511994cc2133)
10. [Observability Stack](#Xd642845f0b74e97b90b61401c7de53b83d34419)
11. [Deployment Configuration](#X21addfc9ebb46f04e6bcf5493b138548bd266d9)
12. [Environment Variables](#X7efbd6d27ccfb484d459bb2b79b7bc2a00cd5f8)
13. [Data Flow & Algorithms](#X645abc88b450092995b951774844a0813b81cc8)
14. [Codebase Discovery Summary](#X7da1cdc08a26bab52f86768c56b81c67ee34cc5)
15. [Architecture Diagrams](#Xbd0a06f01ddd58085e6e1fd8fbee9a221d1e45e)
16. [Actionable Recommendations](#X143b7f9d5571c200174b9397c562e9f8d01e683)

## 1. Executive Summary

**Singletap-Backend** is a semantic matching engine that connects users based on their offerings and requirements. The system uses:

* **Natural Language Processing** to extract structured data from user queries
* **Vector Embeddings** for semantic similarity search
* **Multi-source Ontology Resolution** for synonym/hierarchy matching
* **Boolean Constraint Matching** for precise filtering

### 

### Key Capabilities

| Feature | Technology |
| --- | --- |
| Query Understanding | Fine tuned Mixtral Instruct 7b,gpt-4o |
| Semantic Search | Sentence-Transformers + Qdrant |
| Data Storage | PostgreSQL (Supabase) |
| Ontology Resolution | Wikidata + WordNet + BabelNet |
| Observability | Sentry + OpenTelemetry/Jaeger |

### Architecture Classification

**This is a Python FastAPI ML Backend because:** 1. requirements.txt contains FastAPI, torch, sentence-transformers, OpenAI 2. main.py is a FastAPI application with ML-heavy endpoints 3. Uses Qdrant (vector database) for semantic similarity search 4. Uses GPT-4o for natural language → structured schema extraction 5. No frontend code detected - pure API backend

## 2. Technology Stack Overview

| **Component** | **Technology / Tooling** |
| --- | --- |
| **Language** | Python 3.13+ |
| **Framework** | FastAPI + Uvicorn |
| **Database** | PostgreSQL (Supabase) + Qdrant (Vector DB) |
| **ML Models** | GPT-4o + Mixtral 7B (Fine-Tuned) + Sentence-Transformers |
| **Deployment** | Docker + Railway / Render |
| **Observability** | Sentry + Jaeger (OpenTelemetry) |

LAYERS : 

## 3. Python Libraries - Complete Reference

### 3.1 Web Framework & Server

| Library | Version | Purpose | Why We Use It |
| --- | --- | --- | --- |
| **fastapi** | Latest | Web framework | High-performance async API framework with automatic OpenAPI docs, type validation via Pydantic, and native async support. Chosen for its speed (comparable to Node.js/Go) and developer experience. |
| **uvicorn[standard]** | Latest | ASGI server | Production-grade ASGI server for FastAPI. The [standard] extra includes uvloop and httptools for maximum performance. |
| **pydantic** | Latest | Data validation | Provides runtime type validation for API request/response models. Ensures data integrity at API boundaries. |
| **python-dotenv** | Latest | Environment config | Loads .env files into environment variables. Enables local development without modifying system env. |

### 3.2 Database & Storage

| Library | Version | Purpose | Why We Use It |
| --- | --- | --- | --- |
| **supabase** | Latest | PostgreSQL client | Official Supabase Python client. Provides PostgreSQL access + real-time subscriptions + auth. Chosen for managed PostgreSQL with built-in REST API. |
| **qdrant-client** | ≥1.16.0 | Vector database | Official Qdrant client for vector similarity search. Supports payload filtering, batch operations, and high-performance nearest-neighbor search. |

**Why Supabase + Qdrant?** - Supabase: Managed PostgreSQL for structured data (listings, matches, users) - Qdrant: Specialized vector database for embedding-based semantic search - This hybrid approach gives us both SQL flexibility and vector search performance

### 3.3 Machine Learning & NLP

| Library | Version | Purpose | Why We Use It |
| --- | --- | --- | --- |
| **openai** | Latest | GPT-4o API | Official OpenAI Python client. Used for natural language → structured JSON extraction. GPT-4o provides superior instruction-following for schema extraction. |
| **torch** | CPU build | Deep learning | PyTorch (CPU-only). Required by sentence-transformers for neural network inference. Using CPU build to reduce deployment size and avoid GPU requirements. |
| **sentence-transformers** | Latest | Text embeddings | Generates 384/1024-dimensional embeddings for semantic similarity. Uses transformer models fine-tuned for sentence similarity tasks. |
| **transformers** | ≥4.30.0 | Hugging Face | Core transformer library. Provides model loading, tokenization, and inference for sentence-transformers. |
| **scikit-learn** | ≥1.3.0 | ML utilities | Provides cosine similarity, normalization, and other ML utilities used in embedding comparisons. |
| **numpy** | Latest | Numerical computing | Fundamental array operations for embeddings, distance calculations, and data manipulation. |

**Why This ML Stack?**

User Query → GPT-4o + Fine tuned mistral 7b model (extraction) → Sentence-Transformers (embedding) → Qdrant (search)

* GPT-4o: Best-in-class for structured extraction from natural language
* Sentence-Transformers: Pre-trained models specifically optimized for semantic similarity
* The combination provides both understanding (GPT) and retrieval (embeddings)

### 3.4 NLP & Ontology

| Library | Version | Purpose | Why We Use It |
| --- | --- | --- | --- |
| **nltk** | Latest | WordNet access | Natural Language Toolkit. Provides local access to WordNet lexical database for synonyms, hypernyms, and semantic relationships. |
| **pint** | Latest | Unit conversion | Physical unit handling library. Converts between units (kg↔lb, km↔miles) for numeric constraint normalization. |
| **quantulum3** | Latest | Quantity extraction | Extracts quantities from text (“256gb” → 256 gigabytes). Uses NER-style parsing for unit detection. |

**Why NLTK + Pint + Quantulum3?** - NLTK/WordNet: Local synonym/hierarchy lookup (no API calls, instant) - Pint: Handles all physical unit conversions with dimensional analysis - Quantulum3: Parses natural language quantities into structured values - Together they handle: “looking for 16gb ram laptop” → {memory: 16, unit: "gigabyte"}

### 3.5 HTTP & Networking

| Library | Version | Purpose | Why We Use It |
| --- | --- | --- | --- |
| **requests** | Latest | HTTP client | Simple HTTP client for external API calls (Wikidata, BabelNet, Nominatim). Used by all external service wrappers. |

### 3.6 Observability & Monitoring

| Library | Version | Purpose | Why We Use It |
| --- | --- | --- | --- |
| **structlog** | ≥25.5.0 | Structured logging | Modern structured logging with JSON output support. Provides contextual logging with emoji indicators for better debugging. |
| **sentry-sdk[fastapi]** | ≥2.0.0 | Error tracking | Automatic exception capture, performance monitoring, and error grouping. Integrates with FastAPI for request-level context. |
| **opentelemetry-api** | ≥1.25.0 | Tracing API | OpenTelemetry tracing standards. Provides vendor-neutral tracing API. |
| **opentelemetry-sdk** | ≥1.25.0 | Tracing SDK | OpenTelemetry implementation. Handles span creation, context propagation, and export. |
| **opentelemetry-exporter-otlp** | ≥1.25.0 | Trace export | Exports traces to Jaeger/Grafana via OTLP protocol. Supports both gRPC and HTTP. |
| **opentelemetry-instrumentation-fastapi** | ≥0.46b0 | Auto-instrumentation | Automatically traces all FastAPI endpoints without code changes. |
| **opentelemetry-instrumentation-requests** | ≥0.46b0 | HTTP tracing | Traces outgoing HTTP requests to external APIs. |
| **opentelemetry-instrumentation-logging** | ≥0.46b0 | Log correlation | Adds trace IDs to log messages for correlation. |

**Why This Observability Stack?**

┌─────────────┐ ┌─────────────┐ ┌─────────────┐  
│ Sentry │ │ Jaeger │ │ structlog │  
│ (Errors) │ │ (Traces) │ │ (Logs) │  
└──────┬──────┘ └──────┬──────┘ └──────┬──────┘  
 │ │ │  
 └───────────────────┼───────────────────┘  
 │  
 FastAPI Application

* **Sentry**: Catches errors before users report them
* **Jaeger**: Visualizes request flow across services
* **structlog**: Structured logs for easier debugging

### 3.7 Complete requirements.txt

fastapi  
uvicorn[standard]  
supabase  
qdrant-client>=1.16.0  
python-dotenv  
pydantic  
numpy  
openai  
--extra-index-url https://download.pytorch.org/whl/cpu  
torch  
sentence-transformers  
transformers>=4.30.0  
scikit-learn>=1.3.0  
pint  
quantulum3  
requests  
nltk  
structlog>=25.5.0  
  
# OpenTelemetry for Jaeger distributed tracing  
opentelemetry-api>=1.25.0  
opentelemetry-sdk>=1.25.0  
opentelemetry-exporter-otlp>=1.25.0  
opentelemetry-instrumentation-fastapi>=0.46b0  
opentelemetry-instrumentation-requests>=0.46b0  
opentelemetry-instrumentation-logging>=0.46b0  
  
# Sentry for error tracking and performance monitoring  
sentry-sdk[fastapi]>=2.0.0

## 4. External APIs Used

### 4.1 OpenAI API

| Endpoint | Purpose | File Location |
| --- | --- | --- |
| chat.completions.create | GPT-4o extraction | main.py:538-551, src/core/extraction/gpt\_extractor.py:121-135 |

**Configuration:**

model: "gpt-4o"  
temperature: 0.0  
response\_format: {"type": "json\_object"}

**Why GPT-4o?** - Best structured output following - JSON mode ensures valid JSON response - Temperature 0.0 for deterministic extraction - 100% accuracy on test cases for schema extraction

**Usage Example:**

response = openai\_client.chat.completions.create(  
 model="gpt-4o",  
 messages=[  
 {"role": "system", "content": extraction\_prompt},  
 {"role": "user", "content": query}  
 ],  
 temperature=0.0,  
 response\_format={"type": "json\_object"}  
)

### 4.2 Wikidata API

| Endpoint | Purpose | File Location |
| --- | --- | --- |
| wbsearchentities | Entity search | src/services/external/wikidata\_wrapper.py:57-85 |
| SPARQL endpoint | Hierarchy queries | src/services/external/wikidata\_wrapper.py:105-151 |

**Endpoints:**

api\_endpoint = "https://www.wikidata.org/w/api.php"  
sparql\_endpoint = "https://query.wikidata.org/sparql"

**Why Wikidata?** - Free, no API key required - Comprehensive hierarchical data (P31: instance of, P279: subclass of) - Multilingual canonical labels - Covers: item types, professions, categories

**Example Use Case:**

"dentist" → Wikidata → Q27349 → subclass\_of → Q39631 (physician) → is\_a → "doctor"

**Key Methods:**

# Search for entities  
search\_entity(term: str, language: str = "en", limit: int = 5) -> List[Dict]  
  
# Get canonical label  
get\_canonical\_label(term: str, language: str = "en") -> Optional[str]  
  
# Get superclasses via SPARQL  
get\_superclasses(entity\_id: str, language: str = "en") -> List[Tuple[str, str]]  
  
# Get full hierarchy path  
get\_hierarchy\_path(term: str, max\_depth: int = 5) -> List[List[Dict]]

### 4.3 OpenStreetMap Nominatim API

| Endpoint | Purpose | File Location |
| --- | --- | --- |
| /search | Geocoding | src/services/external/geocoding\_service.py:107-178 |

**Endpoint:**

NOMINATIM\_URL = "https://nominatim.openstreetmap.org/search"

**Why Nominatim?** - Completely free (no API key) - Covers worldwide locations - Returns canonical names (“Bangalore” → “Bengaluru”) - Provides coordinates for distance calculations

**Rate Limiting:** - 1 request per second (Nominatim policy) - File-based caching to minimize API calls

**Key Methods:**

# Geocode a location  
geocode(location\_name: str) -> Optional[Dict]  
# Returns: {"lat": 12.9716, "lng": 77.5946, "canonical\_name": "Bengaluru"}  
  
# Calculate distance between locations  
distance(location1: str, location2: str) -> Optional[float]  
# Returns: distance in kilometers  
  
# Check if within range  
is\_within\_distance(location1: str, location2: str, max\_km: float = 50.0) -> bool

### 4.4 Frankfurter API (Currency Exchange)

| Endpoint | Purpose | File Location |
| --- | --- | --- |
| /latest | Exchange rates | src/services/external/currency\_service.py:154-180 |
| /currencies | Currency list | src/services/external/currency\_service.py:126-152 |

**Endpoint:**

BASE\_URL = "https://api.frankfurter.app"

**Why Frankfurter?** - Free, no API key required - European Central Bank data (reliable) - 30+ currencies including INR, USD, EUR - Daily updates

**Key Methods:**

# Get exchange rate  
get\_rate(from\_currency: str, to\_currency: str = "USD") -> Optional[float]  
  
# Convert amount  
convert(amount: float, from\_currency: str, to\_currency: str = "USD") -> Optional[float]  
  
# Check if valid currency code  
is\_currency\_code(code: str) -> bool

**Caching:** - In-memory cache with 6-hour TTL - Thread-safe via Lock

### 4.5 BabelNet API (Optional)

| Endpoint | Purpose | File Location |
| --- | --- | --- |
| Synset search | Multilingual synonyms | src/services/external/babelnet\_wrapper.py |

**Why BabelNet?** - Largest multilingual semantic network - Combines WordNet + Wikipedia + Wiktionary - Disambiguation via embedding-based context matching - **Requires API key** (optional, fallback to WordNet if unavailable)

### 4.6 WordNet (Local via NLTK)

| Function | Purpose | File Location |
| --- | --- | --- |
| wn.synsets() | Synonym lookup | main.py:274-291 |
| hypernym\_paths() | Hierarchy traversal | canonicalization/resolvers/generic\_categorical\_resolver.py:164-193 |

**Why WordNet?** - Local database (no API calls, instant) - Built into NLTK - Comprehensive English lexical database - Provides: synonyms, hypernyms, hyponyms, derivations

**Example Usage:**

from nltk.corpus import wordnet as wn  
  
# Get synsets  
synsets = wn.synsets("laptop")  
  
# Check if two terms share a synset (true synonyms)  
c\_synsets = set(wn.synsets("laptop"))  
r\_synsets = set(wn.synsets("notebook"))  
if c\_synsets & r\_synsets:  
 return True # They're synonyms  
  
# Get hypernym paths  
for synset in synsets:  
 for path in synset.hypernym\_paths():  
 print(path) # [entity, artifact, device, machine, computer, laptop]

### 4.7 External API Summary Table

| API | Auth Required | Rate Limit | Cache TTL | Purpose |
| --- | --- | --- | --- | --- |
| OpenAI GPT-4o | API Key | Per-account | None | Extraction |
| Wikidata | None | None | In-memory | Ontology |
| Nominatim | None | 1 req/sec | File-based | Geocoding |
| Frankfurter | None | None | 6 hours | Currency |
| BabelNet | API Key | Per-account | In-memory | Synonyms |
| WordNet | None (local) | N/A | N/A | Synonyms |

## 5. Internal API Endpoints

### 5.1 Health & Status

| Endpoint | Method | Purpose | Response |
| --- | --- | --- | --- |
| / | GET | Service status | {status, initialized, service} |
| /health | GET | Health check (load balancers) | {status: "ok"} |
| /ping | GET | Ultra-simple ping | "pong" |

### 5.2 Extraction Endpoints

| Endpoint | Method | Input | Output | File |
| --- | --- | --- | --- | --- |
| /extract | POST | {query: string} | {extracted\_listing: NEW\_SCHEMA} | main.py:557-599 |
| /extract-and-normalize | POST | {query: string} | {extracted\_listing, normalized\_listing} | main.py:601-632 |
| /extract-and-match | POST | {query\_a, query\_b} | {match: boolean, details} | main.py:634-695 |

**/extract Example:**

// Input  
{"query": "need a plumber who speaks kannada"}  
  
// Output  
{  
 "status": "success",  
 "query": "need a plumber who speaks kannada",  
 "extracted\_listing": {  
 "intent": "service",  
 "subintent": "seek",  
 "domain": ["construction & trades"],  
 "primary\_mutual\_category": [],  
 "items": [{"type": "plumbing", "categorical": {}, "min": {}, "max": {}, "range": {}}],  
 "item\_exclusions": [],  
 "other\_party\_preferences": {  
 "categorical": {"language": "kannada"},  
 "min": {},  
 "max": {},  
 "range": {}  
 },  
 "other\_party\_exclusions": [],  
 "self\_attributes": {"categorical": {}, "min": {}, "max": {}, "range": {}},  
 "self\_exclusions": [],  
 "target\_location": {},  
 "location\_match\_mode": "near\_me",  
 "location\_exclusions": [],  
 "reasoning": "User is seeking plumbing services with language preference for Kannada"  
 }  
}

### 5.3 Matching Endpoints

| Endpoint | Method | Input | Output | File |
| --- | --- | --- | --- | --- |
| /match | POST | {listing\_a, listing\_b} | {match: boolean} | main.py:453-472 |
| /search-and-match | POST | {query, user\_id} | {matched\_listings[], match\_count} | main.py:702-845 |
| /search-and-match-direct | POST | {listing\_json, user\_id} | {matches[]} | main.py:848-940 |

**/search-and-match Example:**

// Input  
{  
 "query": "I need a plumber in Bangalore",  
 "user\_id": "user-uuid-123"  
}  
  
// Output  
{  
 "status": "success",  
 "listing\_id": "new-listing-uuid",  
 "match\_ids": ["match-uuid-1", "match-uuid-2"],  
 "query\_text": "I need a plumber in Bangalore",  
 "query\_json": { /\* extracted JSON \*/ },  
 "has\_matches": true,  
 "match\_count": 2,  
 "matched\_listings": [  
 {  
 "listing\_id": "listing-uuid-1",  
 "user\_id": "provider-uuid-1",  
 "data": { /\* listing data \*/ }  
 }  
 ],  
 "message": "Found 2 matches"  
}

### 5.4 Storage Endpoints

| Endpoint | Method | Input | Output | File |
| --- | --- | --- | --- | --- |
| /ingest | POST | {listing, user\_id} | {listing\_id} | main.py:400-430 |
| /store-listing | POST | {query, user\_id} | {listing\_id, extracted\_json} | main.py:943-1053 |
| /search | POST | {listing, limit?} | {candidates[]} | main.py:432-451 |
| /normalize | POST | {listing} | {normalized\_listing} | main.py:474-483 |

### 5.5 Request/Response Models

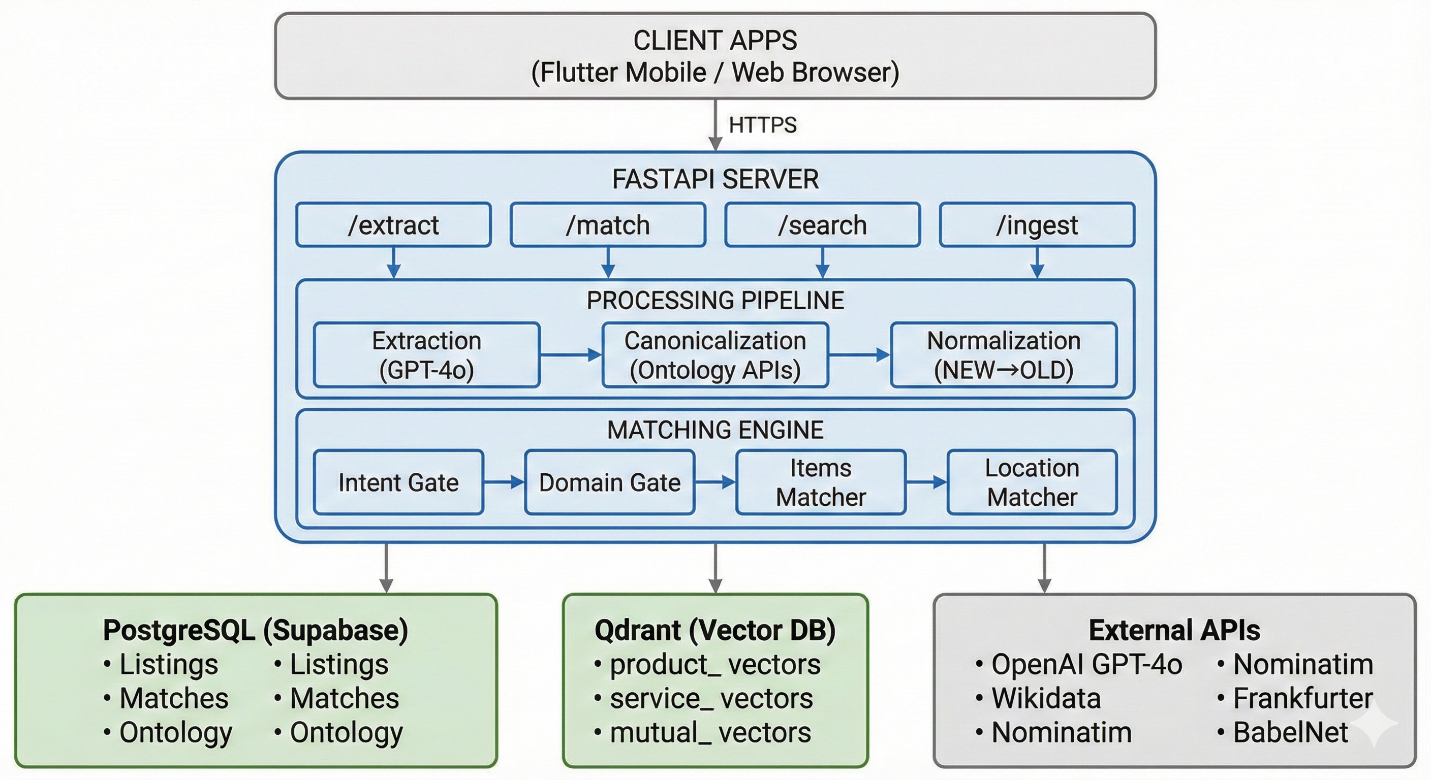
# Pydantic Models (main.py)  
  
class ListingRequest(BaseModel):  
 listing: Dict[str, Any]  
 user\_id: Optional[str] = None  
  
class MatchRequest(BaseModel):  
 listing\_a: Dict[str, Any]  
 listing\_b: Dict[str, Any]  
  
class QueryRequest(BaseModel):  
 query: str  
  
class DualQueryRequest(BaseModel):  
 query\_a: str  
 query\_b: str  
  
class SearchAndMatchRequest(BaseModel):  
 query: str  
 user\_id: str  
  
class StoreListingRequest(BaseModel):  
 query: str  
 user\_id: str  
 match\_id: Optional[str] = None

### 5.6 Complete API Reference

| Endpoint | Method | Auth | Purpose |
| --- | --- | --- | --- |
| / | GET | No | Service status |
| /health | GET | No | Health check |
| /ping | GET | No | Simple ping |
| /extract | POST | No | NL → JSON extraction |
| /extract-and-normalize | POST | No | Extract + normalize |
| /extract-and-match | POST | No | Extract two queries, match |
| /normalize | POST | No | Normalize listing |
| /match | POST | No | Match two listings |
| /search | POST | No | Vector search |
| /ingest | POST | No | Store listing |
| /search-and-match | POST | No | Full pipeline |
| /search-and-match-direct | POST | No | Full pipeline (no GPT) |
| /store-listing | POST | No | Extract + store |

## 6. System Architecture

### 6.1 High-Level Architecture

  
6.2 Folder Structure

Singletap-backend/  
├── main.py # FastAPI application (1054 lines)  
├── requirements.txt # Python dependencies  
│  
├── src/ # Modular source code  
│ ├── api/ # API routes  
│ │ ├── \_\_init\_\_.py  
│ │ └── routes.py  
│ ├── config/ # Settings, constants, clients  
│ │ ├── \_\_init\_\_.py  
│ │ ├── settings.py # Configuration management  
│ │ ├── constants.py # Application constants  
│ │ └── clients.py # Database clients  
│ ├── core/  
│ │ ├── extraction/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ ├── gpt\_extractor.py # GPT-4o extraction logic  
│ │ │ └── hybrid\_extractor.py # Hybrid GPT + NuExtract  
│ │ ├── matching/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ ├── orchestrator.py # Matching orchestration  
│ │ │ ├── item\_matchers.py # Item matching logic  
│ │ │ ├── item\_array\_matchers.py  
│ │ │ ├── other\_self\_matchers.py  
│ │ │ ├── location\_matcher.py # Location matching  
│ │ │ ├── numeric\_constraints.py  
│ │ │ ├── mutual\_matcher.py  
│ │ │ └── ontology\_resolver.py  
│ │ ├── canonicalization/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ └── resolvers/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ ├── generic\_categorical\_resolver.py  
│ │ │ ├── quantitative\_resolver.py  
│ │ │ └── type\_resolver.py  
│ │ └── schema/  
│ │ ├── \_\_init\_\_.py  
│ │ └── normalizer.py  
│ ├── services/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── embedding/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ └── service.py  
│ │ ├── ranking/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ ├── ranking\_engine.py  
│ │ │ ├── rrf.py # Reciprocal Rank Fusion  
│ │ │ └── cross\_encoder\_wrapper.py  
│ │ ├── retrieval/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ └── service.py  
│ │ ├── storage/  
│ │ │ └── \_\_init\_\_.py  
│ │ └── external/ # External API Wrappers  
│ │ ├── \_\_init\_\_.py  
│ │ ├── wikidata\_wrapper.py # Wikidata API  
│ │ ├── geocoding\_service.py # Nominatim geocoding  
│ │ ├── currency\_service.py # Frankfurter currency  
│ │ ├── babelnet\_wrapper.py # BabelNet API  
│ │ ├── conceptnet\_wrapper.py  
│ │ ├── pint\_wrapper.py # Unit conversion  
│ │ └── quantulum\_wrapper.py # Quantity extraction  
│ ├── setup/  
│ │ ├── \_\_init\_\_.py  
│ │ └── qdrant\_setup.py # Qdrant collection setup  
│ ├── data/  
│ │ ├── loaders/  
│ │ │ ├── \_\_init\_\_.py  
│ │ │ ├── gpc\_loader.py  
│ │ │ └── unspsc\_loader.py  
│ │ └── type\_hierarchy.json  
│ └── utils/  
│ ├── \_\_init\_\_.py  
│ ├── logging.py # structlog configuration  
│ ├── tracing.py # OpenTelemetry setup  
│ ├── sentry.py # Error tracking  
│ └── helpers.py  
│  
├── pipeline/ # Data pipelines  
│ ├── \_\_init\_\_.py  
│ ├── ingestion\_pipeline.py # Supabase + Qdrant ingestion  
│ └── retrieval\_service.py # Candidate retrieval  
│  
├── matching/ # Matching engine  
│ ├── \_\_init\_\_.py  
│ ├── listing\_matcher\_v2.py # Main orchestrator  
│ ├── item\_array\_matchers.py  
│ ├── other\_self\_matchers.py  
│ └── location\_matcher\_v2.py  
│  
├── canonicalization/ # Ontology resolution  
│ ├── \_\_init\_\_.py  
│ ├── orchestrator.py # Main canonicalization entry  
│ ├── ontology\_store.py # Persistence layer  
│ ├── preprocessor.py  
│ ├── disambiguator.py  
│ ├── canonicalizer.py  
│ └── resolvers/  
│ └── generic\_categorical\_resolver.py  
│  
├── schema/ # Schema transformation  
│ ├── \_\_init\_\_.py  
│ └── schema\_normalizer\_v2.py # NEW → OLD schema  
│  
├── embedding/ # Embedding generation  
│ ├── \_\_init\_\_.py  
│ ├── embedding\_builder.py # Text → embedding text  
│ └── model\_provider.py # Singleton model loader  
│  
├── services/ # Legacy services (being migrated)  
│ └── external/  
│ ├── babelnet\_wrapper.py  
│ └── wordnet\_wrapper.py  
│  
├── prompt/ # GPT prompts  
│ └── GLOBAL\_REFERENCE\_CONTEXT.md # Extraction prompt (system message)  
│  
├── migrations/ # SQL migrations  
│ ├── 001\_create\_matches\_table.sql  
│ ├── 002\_create\_listings\_tables.sql  
│ └── 003\_create\_concept\_ontology.sql  
│  
├── tests/ # Test suites  
│ ├── unit\_testing/  
│ │ ├── test\_item\_matchers.py  
│ │ ├── test\_item\_array\_matchers.py  
│ │ ├── test\_location\_matchers.py  
│ │ ├── test\_mutual\_matcher.py  
│ │ ├── test\_normalizer.py  
│ │ ├── test\_numeric\_constraints.py  
│ │ ├── test\_other\_self\_matchers.py  
│ │ ├── test\_ranking\_engine.py  
│ │ ├── test\_embedding\_v2.py  
│ │ └── verify\_qdrant.py  
│ ├── integration\_testing/  
│ │ ├── test\_canonicalization.py  
│ │ ├── test\_extraction\_api.py  
│ │ ├── test\_gpt\_extraction.py  
│ │ ├── test\_qdrant\_search.py  
│ │ └── test\_single\_query.py  
│ ├── feature\_testing/  
│ │ ├── test\_all\_examples.py  
│ │ ├── test\_complete\_flow.py  
│ │ ├── test\_debug\_matches.py  
│ │ ├── test\_e2e\_matching.py  
│ │ ├── test\_flow\_direct.py  
│ │ ├── test\_mutual\_queries.py  
│ │ ├── test\_search\_debug.py  
│ │ └── test\_user\_queries.py  
│ └── files\_functions\_testing/  
│ ├── fix\_test\_files.py  
│ └── test\_schema\_update.py  
│  
├── scripts/ # Utility scripts  
│ ├── check\_tables.py  
│ ├── create\_payload\_indexes.py  
│ ├── create\_qdrant\_collections.py  
│ ├── download\_model.py  
│ ├── fix\_user\_id\_type.py  
│ ├── qdrant\_setup.py  
│ ├── run\_migration.py  
│ └── run\_migration\_direct.py  
│  
├── legacy/ # Deprecated code  
│ ├── cross\_encoder\_wrapper.py  
│ ├── integration\_example\_v2.py  
│ ├── listing\_matcher.py  
│ ├── location\_matchers.py  
│ ├── mutual\_matcher.py  
│ ├── new\_endpoints.py  
│ ├── query\_pipeline\_example.py  
│ ├── ranking\_engine.py  
│ ├── rrf.py  
│ └── schema\_normalizer.py  
│  
├── docs/ # Documentation  
│ └── ARCHITECTURE\_DOCUMENTATION.md # This file  
│  
├── test\_queries/ # Test query files  
│  
├── models/ # ML models directory  
│  
├── nltk\_data/ # NLTK data (WordNet, etc.)  
│  
├── docker-compose.yaml # Docker orchestration  
├── docker-compose.jaeger.yml # Jaeger tracing setup  
├── Procfile # Railway deployment  
├── railway.toml # Railway config  
├── render.yaml # Render deployment  
├── geocoding\_cache.json # Geocoding cache file  
│  
└── README.md # Project README

## 7. Core Components Deep Dive

### 7.1 GPT Extraction Pipeline

**Location:** src/core/extraction/gpt\_extractor.py

**Purpose:** Convert natural language queries into structured 14-field NEW schema.

**Process:**

"looking for a plumber who speaks kannada in bangalore"  
 │  
 ▼  
 ┌────────────────────────┐  
 │ Load System Prompt │  
 │ (GLOBAL\_REFERENCE\_ │  
 │ CONTEXT.md) │  
 └────────────┬───────────┘  
 │  
 ▼  
 ┌────────────────────────┐  
 │ GPT-4o API Call │  
 │ model: "gpt-4o" │  
 │ temperature: 0.0 │  
 │ format: json\_object │  
 └────────────┬───────────┘  
 │  
 ▼  
 ┌────────────────────────┐  
 │ Parse JSON Response │  
 └────────────┬───────────┘  
 │  
 ▼  
{  
 "intent": "service",  
 "subintent": "seek",  
 "domain": ["construction & trades"],  
 "items": [{"type": "plumbing", "categorical": {}}],  
 "other\_party\_preferences": {  
 "categorical": {"language": "kannada"}  
 },  
 "target\_location": {"name": "bangalore"},  
 "location\_match\_mode": "near\_me"  
}

| **Mode** | **Environment Variable** | **Description** |
| --- | --- | --- |
| **GPT-Only (default)** | **EXTRACTION\_MODE=gpt** | **Fast (~5-15s), 100% accuracy.** |
| **Hybrid** | **EXTRACTION\_MODE=hybrid** | **GPT + NuExtract validation layer.** |
| **Hybrid (skip validation)** | **EXTRACTION\_MODE=hybrid**  **+ SKIP\_NUEXTRACT=1** | **Hybrid extractor without Level 2 validation.** |

**Code Structure:**

class GPTExtractor:  
 def \_\_init\_\_(self):  
 self.openai\_client: Optional[OpenAI] = None  
 self.extraction\_prompt: Optional[str] = None  
 self.initialized: bool = False  
  
 def initialize(self, api\_key=None, prompt\_path=None) -> bool:  
 """Load prompt + OpenAI client"""  
  
 def extract(self, query: str, model="gpt-4o", temperature=0.0) -> Dict[str, Any]:  
 """Extract structured schema from query"""

### 7.2 Canonicalization Pipeline

**Location:** canonicalization/orchestrator.py

**Purpose:** Resolve non-deterministic values to canonical forms.

**Pipeline Flow:**

GPT-4o → NEW Schema → CANONICALIZE → Canonical NEW → Normalize → OLD Schema

**Canonicalization Steps:**

1. **Domain canonicalization** - Lowercase normalization

2. **Item type canonicalization** - Wikidata with domain context

3. **Categorical attribute resolution** - Multi-source ontology

4. **Quantitative normalization** - Pint + Quantulum3

5. **Location canonicalization** - Nominatim geocoding

6. **Flush to ontology DB** - Write-behind persistence

**Resolution Cascade:**

Input Value  
 │  
 ▼  
┌────────────────────┐  
│ Synonym Registry? │ ─Yes→ Return cached concept\_id  
└────────┬───────────┘  
 │No  
 ▼  
┌────────────────────┐  
│ WordNet (local)? │ ─Yes→ Return with hierarchy  
└────────┬───────────┘  
 │No  
 ▼  
┌────────────────────┐  
│ BabelNet (API)? │ ─Yes→ Return with synonyms  
└────────┬───────────┘  
 │No  
 ▼  
┌────────────────────┐  
│ Wikidata (API)? │ ─Yes→ Return with hierarchy  
└────────┬───────────┘  
 │No  
 ▼  
┌────────────────────┐  
│ Lowercase Fallback │  
└────────────────────┘

**OntologyNode Structure:**

@dataclass  
class OntologyNode:  
 concept\_id: str # Canonical identifier  
 concept\_root: str # Root category  
 concept\_path: List[str] # Full path from root  
 parents: List[str] # Direct parents  
 children: List[str] # Direct children  
 siblings: List[str] # Related concepts  
 source: str # wordnet/babelnet/wikidata/fallback  
 confidence: float # 0.0 to 1.0

### 7.3 Schema Normalization

**Location:** schema/schema\_normalizer\_v2.py

**Purpose:** Transform 14-field NEW schema → 12-field OLD schema for matching engine compatibility.

**Field Transformations:**

| NEW Schema Field | OLD Schema Field |
| --- | --- |
| intent | intent |
| subintent | subintent |
| domain | domain |
| primary\_mutual\_category | category |
| items | items |
| item\_exclusions | itemexclusions |
| other\_party\_preferences | other |
| other\_party\_exclusions | otherexclusions |
| self\_attributes | self |
| self\_exclusions | selfexclusions |
| target\_location | location |
| location\_match\_mode | locationmode |
| location\_exclusions | locationexclusions |
| reasoning | reasoning |

**Constraint Flattening:**

# NEW Schema (axis-based)  
{  
 "min": {  
 "capacity": [{"type": "memory", "value": 16, "unit": "gb"}]  
 }  
}  
  
# OLD Schema (flat)  
{  
 "min": {  
 "memory": 16  
 }  
}

**Valid Axes (10 fixed):** - identity, capacity, performance, quality, quantity - time, space, cost, mode, skill

### 7.4 Matching Engine

**Location:** matching/listing\_matcher\_v2.py

**Purpose:** Determine if listing B satisfies listing A’s requirements.

**Direction:** A → B (unidirectional) - A = requester (what they want) - B = candidate (what they offer)

**Matching Rules (Short-Circuit Evaluation):**

def listing\_matches\_v2(A: Dict, B: Dict, implies\_fn=None) -> bool:  
 """  
 Returns True only if B satisfies ALL of A's constraints.  
 Short-circuits on first failure.  
 """  
  
 # ══════════════════════════════════════════════════════  
 # GATE 1: Intent Equality (Rule M-01)  
 # ══════════════════════════════════════════════════════  
 if A["intent"] != B["intent"]:  
 return False # Intent mismatch  
  
 # ══════════════════════════════════════════════════════  
 # GATE 2: SubIntent Rules (Rules M-02, M-03)  
 # ══════════════════════════════════════════════════════  
 intent = A["intent"]  
  
 if intent in ["product", "service"]:  
 # M-02: SubIntent Inverse Rule  
 # buyer/seller, seeker/provider must be opposite  
 if A["subintent"] == B["subintent"]:  
 return False # Both buyers or both sellers  
  
 elif intent == "mutual":  
 # M-03: SubIntent Same Rule  
 # Both must be "connect"  
 if A["subintent"] != B["subintent"]:  
 return False  
  
 # ══════════════════════════════════════════════════════  
 # GATE 3: Domain/Category Intersection (Rules M-05, M-06)  
 # ══════════════════════════════════════════════════════  
 if intent in ["product", "service"]:  
 # M-05: Domain must have non-empty intersection  
 if not has\_intersection(A["domain"], B["domain"]):  
 return False  
  
 elif intent == "mutual":  
 # M-06: Category must have non-empty intersection  
 if not has\_intersection(A["category"], B["category"]):  
 return False  
  
 # ══════════════════════════════════════════════════════  
 # GATE 4: Items Matching (Rules M-07 to M-12)  
 # ══════════════════════════════════════════════════════  
 if intent in ["product", "service"]:  
 if not all\_required\_items\_match(A["items"], B["items"], implies\_fn):  
 return False  
  
 # ══════════════════════════════════════════════════════  
 # GATE 5: Other→Self Matching (Rules M-13 to M-17)  
 # ══════════════════════════════════════════════════════  
 if not match\_other\_to\_self(A["other"], B["self"], implies\_fn):  
 return False  
  
 # ══════════════════════════════════════════════════════  
 # GATE 6: Location Matching (Rules M-23 to M-28)  
 # ══════════════════════════════════════════════════════  
 if not match\_location\_v2(A, B):  
 return False  
  
 # ══════════════════════════════════════════════════════  
 # ALL GATES PASSED  
 # ══════════════════════════════════════════════════════  
 return True

**Matching Rules Reference:**

| Rule | Gate | Logic |
| --- | --- | --- |
| M-01 | Intent | A.intent == B.intent |
| M-02 | SubIntent (product/service) | A.subintent ≠ B.subintent |
| M-03 | SubIntent (mutual) | A.subintent == B.subintent |
| M-04 | Intent-SubIntent Validity | Schema validated |
| M-05 | Domain | A.domain ∩ B.domain ≠ ∅ |
| M-06 | Category | A.category ∩ B.category ≠ ∅ |
| M-07 to M-12 | Items | Type + Categorical + Numeric |
| M-13 to M-17 | Other→Self | Preferences matching |
| M-23 to M-28 | Location | Mode-based matching |

### 7.5 Semantic Implication Engine

**Location:** main.py:217-356

**Purpose:** Check if one categorical term implies another.

**Strategies (in order of evaluation):**

1. **Exact Match**

* if candidate.lower() == required.lower():  
   return True

1. **Curated Synonyms**

* CURATED\_SYNONYMS = {  
   frozenset({"laptop", "notebook"}),  
   frozenset({"cleaning", "housekeeping", "housework"}),  
   frozenset({"couch", "sofa"}),  
   frozenset({"automobile", "car", "auto"}),  
   frozenset({"phone", "telephone", "cellphone", "mobile"}),  
   frozenset({"apartment", "flat"}),  
  }

1. **Wikidata Hierarchy**

* # "dentist" is subclass of "doctor"  
  if wikidata.is\_subclass\_of(candidate, required, max\_depth=3):  
   return True

1. **WordNet Ancestor**

* if resolver.is\_ancestor(required, candidate):  
   return True

1. **WordNet Synset Overlap**

* if candidate\_synsets & required\_synsets:  
   return True # Same synset = true synonyms

1. **Morphological Matching**

* # "plumber" ↔ "plumbing" share root "plumb"  
  if len(common\_prefix) >= 5:  
   return True

1. **BabelNet Synonyms**

* if candidate in babelnet.get\_synonyms(required):  
   return True

### 7.6 Embedding Pipeline

**Location:** embedding/embedding\_builder.py

**Purpose:** Build text representation for embedding generation.

**Strategy by Intent:**

| Intent | Strategy | Example Output |
| --- | --- | --- |
| Product/Service | Structured concatenation | "product buy electronics laptop brand apple ram memory" |
| Mutual | Natural language | "mutual exchange in categories: tutoring offering math tutoring wanting english lessons" |

**Code:**

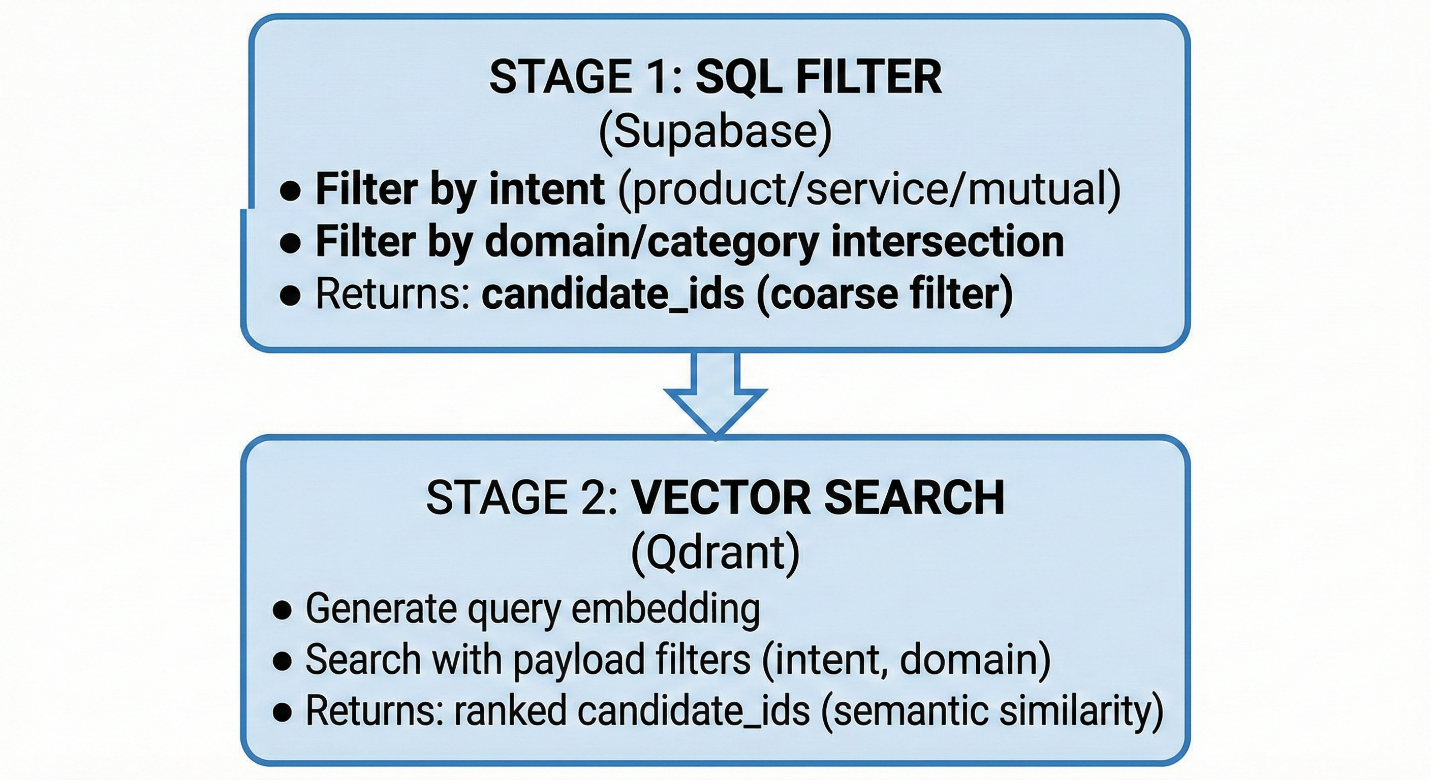
def build\_embedding\_text(listing: Dict[str, Any]) -> str:  
 """Route to appropriate builder based on intent."""  
 intent = listing["intent"]  
  
 if intent in ["product", "service"]:  
 return build\_embedding\_text\_product\_service(listing)  
 elif intent == "mutual":  
 return build\_embedding\_text\_mutual(listing)  
 else:  
 raise ValueError(f"Unknown intent: {intent}")  
  
def build\_embedding\_text\_product\_service(listing: Dict) -> str:  
 """Concatenate: intent + subintent + domain + items (type + categorical)"""  
 parts = []  
  
 if listing.get("intent"):  
 parts.append(listing["intent"])  
  
 if listing.get("subintent"):  
 parts.append(listing["subintent"])  
  
 for domain in listing.get("domain", []):  
 parts.append(str(domain))  
  
 for item in listing.get("items", []):  
 if item.get("type"):  
 parts.append(str(item["type"]))  
 for key, value in item.get("categorical", {}).items():  
 parts.append(str(key))  
 parts.append(str(value))  
  
 return " ".join(parts)

### 7.7 Retrieval Pipeline

**Location:** pipeline/retrieval\_service.py

**Purpose:** Retrieve candidate listings for matching.

**Two-Stage Retrieval:**



**Code:**

def retrieve\_candidates(  
 clients: RetrievalClients,  
 query\_listing: Dict[str, Any],  
 limit: int = 100,  
 use\_sql\_filter: bool = True  
) -> List[str]:  
 """  
 Pipeline:  
 1. SQL filter (Supabase) - optional  
 2. Qdrant vector search with payload filters  
 3. Return candidate listing\_ids  
  
 NO ranking. NO boolean matching. NO scoring returned.  
 """

## 8. Database Architecture

### 8.1 PostgreSQL Schema (Supabase)

**Listings Tables:**

-- Product Listings  
CREATE TABLE IF NOT EXISTS product\_listings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID NOT NULL,  
 match\_id UUID REFERENCES matches(match\_id),  
 data JSONB NOT NULL, -- Complete listing object  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Indexes  
CREATE INDEX idx\_product\_user ON product\_listings(user\_id);  
CREATE INDEX idx\_product\_created ON product\_listings(created\_at DESC);  
CREATE INDEX idx\_product\_match ON product\_listings(match\_id);  
  
-- Same structure for: service\_listings, mutual\_listings

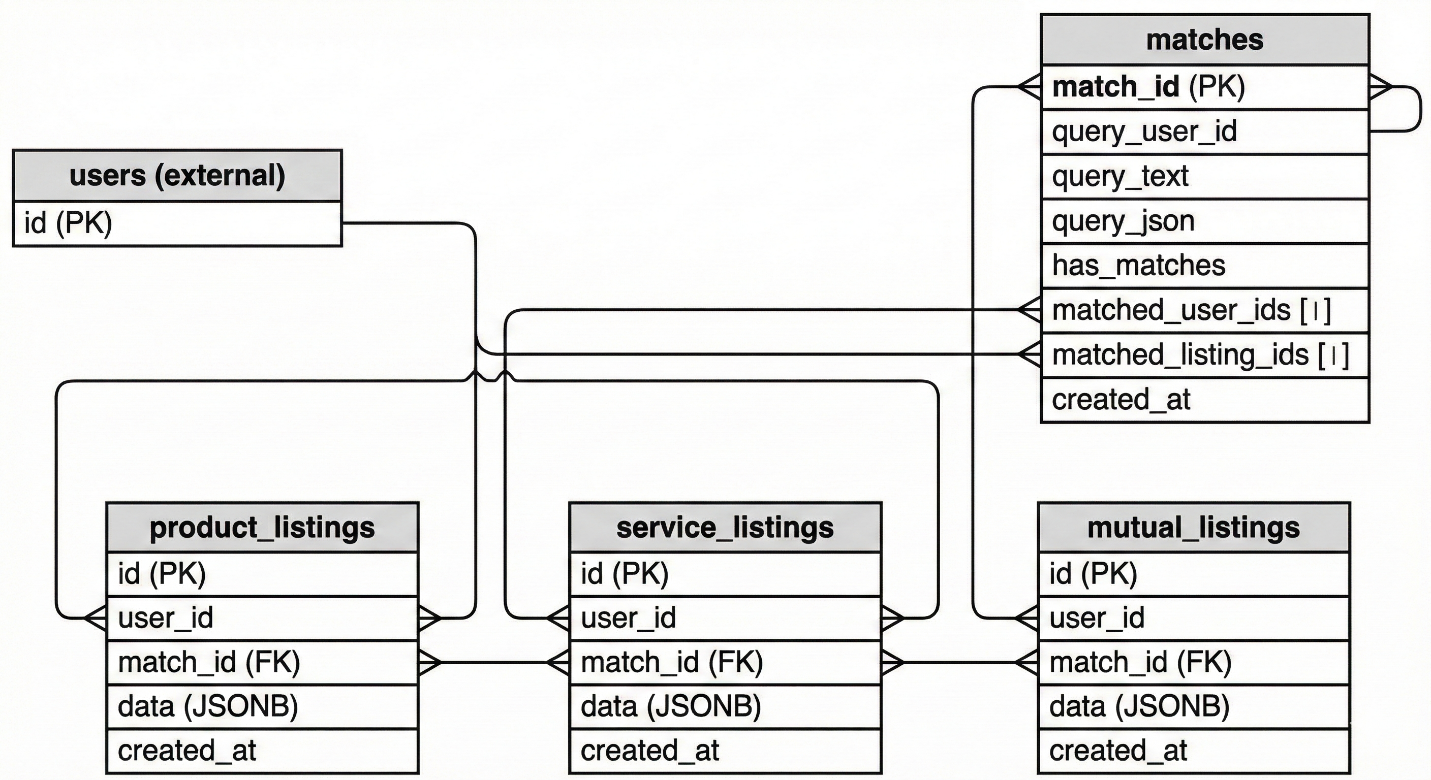
**Matches Table:**

CREATE TABLE IF NOT EXISTS matches (  
 match\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 query\_user\_id UUID NOT NULL,  
 query\_text TEXT NOT NULL,  
 query\_json JSONB NOT NULL,  
 has\_matches BOOLEAN NOT NULL,  
 match\_count INTEGER NOT NULL DEFAULT 0,  
 matched\_user\_ids UUID[] DEFAULT '{}',  
 matched\_listing\_ids UUID[] DEFAULT '{}',  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Indexes  
CREATE INDEX idx\_matches\_user ON matches(query\_user\_id);  
CREATE INDEX idx\_matches\_created ON matches(created\_at DESC);  
CREATE INDEX idx\_matches\_has\_matches ON matches(has\_matches);

### 8.2 Qdrant Collections

### 

### 8.3 Data Model Relationships



## 9. ML Pipeline

### 9.1 Model Configuration

| Component | Model | Dimension | Purpose |
| --- | --- | --- | --- |
| Extraction | GPT-4o | N/A | Natural Language → JSON |
| Embedding (default) | all-MiniLM-L6-v2 | 384 | Fast, lightweight |
| Embedding (large) | BAAI/bge-large-en-v1.5 | 1024 | Better quality |

### 9.2 Model Selection

# Environment variable  
EMBEDDING\_MODEL = os.environ.get("EMBEDDING\_MODEL", "all-MiniLM-L6-v2")  
  
# Dimension depends on model  
EMBEDDING\_DIM = 1024 if "large" in EMBEDDING\_MODEL else 384

### 

### 9.3 Embedding Generation Flow

Listing Object  
 │  
 ▼  
┌─────────────────────┐  
│ build\_embedding\_text│ ← Converts structured data to text  
│ (embedding\_builder│  
│ .py) │  
└──────────┬──────────┘  
 │  
 ▼  
 "product buy electronics laptop brand apple..."  
 │  
 ▼  
┌─────────────────────┐  
│ SentenceTransformer │ ← Generates vector  
│ .encode(text) │  
│ │  
│ Model: all-MiniLM │  
│ or BAAI/bge-large │  
└──────────┬──────────┘  
 │  
 ▼  
 [0.023, -0.156, 0.089, ...] (384/1024 dimensions)  
 │  
 ▼  
┌─────────────────────┐  
│ Qdrant Collection │ ← Stored for search  
│ │  
│ - Point ID: UUID │  
│ - Vector: [...] │  
│ - Payload: {...} │  
└─────────────────────┘

### 9.4 Model Provider (Singleton)

# embedding/model\_provider.py  
  
\_embedding\_model = None  
  
def get\_embedding\_model():  
 """Get singleton embedding model instance."""  
 global \_embedding\_model  
 if \_embedding\_model is None:  
 from sentence\_transformers import SentenceTransformer  
 \_embedding\_model = SentenceTransformer(EMBEDDING\_MODEL)  
 return \_embedding\_model

## 10. Observability Stack

### 10.1 Structured Logging (structlog)

**Location:** src/utils/logging.py

**Features:** - Emoji indicators for log types - JSON output for production - Contextual logging with key-value pairs - Timestamp formatting

**Emoji Indicators:**

| **Emoji** | **Key** | **Usage** |
| --- | --- | --- |
| 🚀 | start | Server startup, initialization |
| ✅ | success | Successful operations |
| ❌ | error | Errors and failures |
| ⚠️ | warning | Warnings |
| ℹ️ | info | Informational messages |
| 🔍 | search | Search operations |
| 💾 | store | Storage operations |
| 🎯 | match | Matching operations |
| 🤖 | extract | GPT extraction |
| 🔎 | filter | Filtering operations |
| 🧠 | semantic | Semantic matching |
| ⚖️ | boolean | Boolean matching |
| 📍 | location | Location matching |

**Usage:**

from src.utils.logging import get\_logger  
  
log = get\_logger(\_\_name\_\_)  
  
log.info("Server starting", emoji="start", port=8000)  
# Output: 🚀 Server starting port=8000  
  
log.error("Connection failed", emoji="error", error=str(e))  
# Output: ❌ Connection failed error="..."  
  
log.info("Found matches", emoji="match", count=5, user\_id="abc")  
# Output: 🎯 Found matches count=5 user\_id=abc

**Configuration:**

configure\_structlog(  
 json\_output=False, # True for production  
 log\_level="INFO",  
 include\_timestamp=True  
)

### 10.2 Error Tracking (Sentry)

**Location:** src/utils/sentry.py

**Captures:** - Unhandled exceptions - HTTP errors (4xx, 5xx) - Performance traces - User context

**Configuration:**

sentry\_sdk.init(  
 dsn=SENTRY\_DSN,  
 environment=ENVIRONMENT, # development/staging/production  
 release=f"vriddhi-matching-engine@{SERVICE\_VERSION}",  
  
 # Performance monitoring  
 traces\_sample\_rate=0.1 if ENVIRONMENT == "production" else 1.0,  
 profiles\_sample\_rate=0.1 if ENVIRONMENT == "production" else 1.0,  
  
 # Integrations  
 integrations=[  
 FastApiIntegration(transaction\_style="endpoint"),  
 StarletteIntegration(transaction\_style="endpoint"),  
 ],  
  
 # Privacy  
 send\_default\_pii=False,  
  
 # Filter noise  
 before\_send=\_filter\_events,  
)

**Event Filtering:**

def \_filter\_events(event, hint):  
 """Filter out noisy events."""  
 # Don't send health check errors  
 if transaction in ["/health", "/ping", "/"]:  
 return None  
  
 # Don't send 404 errors  
 if status\_code == 404:  
 return None  
  
 return event

### 

### 10.3 Distributed Tracing (OpenTelemetry → Jaeger)

**Location:** src/utils/tracing.py

**Auto-instrumented:** - FastAPI endpoints - HTTP requests (external APIs) - Logging (trace ID correlation)

**Configuration:**

# Environment Variables  
TRACING\_ENABLED = os.getenv("TRACING\_ENABLED", "false")  
OTLP\_ENDPOINT = os.getenv("OTLP\_ENDPOINT", "http://localhost:4317")  
OTEL\_SERVICE\_NAME = os.getenv("OTEL\_SERVICE\_NAME", "vriddhi-matching-engine")  
OTEL\_SERVICE\_VERSION = os.getenv("OTEL\_SERVICE\_VERSION", "2.0.0")

**Custom Spans:**

from src.utils.tracing import get\_tracer, traced  
  
# Decorator approach  
@traced("extract-listing")  
async def extract\_endpoint(request):  
 ...  
  
# Context manager approach  
tracer = get\_tracer(\_\_name\_\_)  
with tracer.start\_as\_current\_span("my-operation") as span:  
 span.set\_attribute("user\_id", user\_id)  
 # ... your code

**Graceful Degradation:** - Returns no-op tracer if OpenTelemetry not installed - Returns no-op tracer if TRACING\_ENABLED=false - Code works identically with or without tracing

## 11. Deployment Configuration

### 11.1 Docker Compose

# docker-compose.yaml  
services:  
 web:  
 build: .  
 ports:  
 - "${PORT}:${PORT}"  
 depends\_on:  
 - qdrant  
 environment:  
 - SUPABASE\_URL=${SUPABASE\_URL}  
 - SUPABASE\_KEY=${SUPABASE\_KEY}  
 - QDRANT\_HOST=qdrant  
 - QDRANT\_PORT=6333  
 - OPENAI\_API\_KEY=${OPENAI\_API\_KEY}  
  
 qdrant:  
 image: qdrant/qdrant:latest  
 ports:  
 - "6333:6333" # REST API  
 - "6334:6334" # gRPC  
 volumes:  
 - qdrant\_storage:/qdrant/storage  
  
volumes:  
 qdrant\_storage:

### 11.2 Docker Compose with Jaeger

# docker-compose.jaeger.yml  
services:  
 jaeger:  
 image: jaegertracing/jaeger:2.0  
 ports:  
 - "16686:16686" # Jaeger UI  
 - "4317:4317" # OTLP gRPC  
 - "4318:4318" # OTLP HTTP  
 environment:  
 - COLLECTOR\_OTLP\_ENABLED=true

### 11.3 Railway Deployment

**Procfile:**

web: uvicorn main:app --host 0.0.0.0 --port $PORT

**railway.toml:**

[build]  
builder = "nixpacks"  
  
[deploy]  
startCommand = "uvicorn main:app --host 0.0.0.0 --port $PORT"  
healthcheckPath = "/health"  
healthcheckTimeout = 300

### 

### 11.4 Render Deployment

**render.yaml:**

services:  
 - type: web  
 name: vriddhi-backend  
 runtime: python  
 buildCommand: pip install -r requirements.txt  
 startCommand: uvicorn main:app --host 0.0.0.0 --port $PORT  
 healthCheckPath: /health  
 envVars:  
 - key: SUPABASE\_URL  
 sync: false  
 - key: SUPABASE\_KEY  
 sync: false  
 - key: OPENAI\_API\_KEY  
 sync: false

## 12. Environment Variables

### 12.1 Required Variables

| Variable | Purpose | Example |
| --- | --- | --- |
| SUPABASE\_URL | PostgreSQL connection | https://xxx.supabase.co |
| SUPABASE\_KEY | Supabase API key | eyJ... |
| OPENAI\_API\_KEY | GPT-4o access | sk-... |

### 12.2 Vector Database Variables

| Variable | Purpose | Default |
| --- | --- | --- |
| QDRANT\_ENDPOINT | Qdrant Cloud URL | - |
| QDRANT\_API\_KEY | Qdrant Cloud key | - |
| QDRANT\_HOST | Local Qdrant host | localhost |
| QDRANT\_PORT | Local Qdrant port | 6333 |

### 12.3 ML Configuration Variables

| Variable | Purpose | Default |
| --- | --- | --- |
| EMBEDDING\_MODEL | Model selection | all-MiniLM-L6-v2 |
| EXTRACTION\_MODE | gpt or hybrid | gpt |
| SKIP\_NUEXTRACT | Skip NuExtract in hybrid | 0 |
| USE\_HYBRID\_EXTRACTION | Legacy hybrid flag | 0 |
| USE\_NEW\_PIPELINE | Canonicalization pipeline | 1 |

### 12.4 Observability Variables

| Variable | Purpose | Default |
| --- | --- | --- |
| SENTRY\_ENABLED | Enable Sentry | false |
| SENTRY\_DSN | Sentry DSN | - |
| TRACING\_ENABLED | Enable Jaeger | false |
| OTLP\_ENDPOINT | Jaeger OTLP endpoint | http://localhost:4317 |
| OTLP\_HEADERS | OTLP auth headers | - |
| OTEL\_SERVICE\_NAME | Service name for traces | vriddhi-matching-engine |
| OTEL\_SERVICE\_VERSION | Service version | 2.0.0 |
| ENVIRONMENT | Environment name | development |

### 12.5 External API Variables

| Variable | Purpose | Default |
| --- | --- | --- |
| BABELNET\_API\_KEY | BabelNet access | - (optional) |

### 12.6 Server Variables

| Variable | Purpose | Default |
| --- | --- | --- |
| PORT | Server port | 8000 |

## 13. Data Flow & Algorithms

### 13.1 Complete Request Flow

### 

### 13.2 Matching Algorithm Complexity

| Gate | Time Complexity | Description |
| --- | --- | --- |
| Intent | O(1) | String equality |
| SubIntent | O(1) | String equality/inequality |
| Domain | O(n) | Set intersection |
| Items | O(n×m) | n required items × m candidate items |
| Other→Self | O(k) | k categorical keys |
| Location | O(1) | Mode-based comparison |

**Overall Complexity:** O(n × m × k) where: - n = number of required items - m = number of candidate items - k = number of categorical attributes

### 13.3 Semantic Implication Algorithm

def semantic\_implies(candidate\_val: str, required\_val: str) -> bool:  
 """  
 Check if candidate\_val implies required\_val.  
 Returns True if candidate satisfies requirement.  
 """  
 c, r = candidate\_val.lower().strip(), required\_val.lower().strip()  
  
 # 1. Exact match (O(1))  
 if c == r:  
 return True  
  
 # 2. Curated synonyms (O(n) where n = synonym groups)  
 for syn\_group in CURATED\_SYNONYMS:  
 if c in syn\_group and r in syn\_group:  
 return True  
  
 # 3. Wikidata hierarchy (O(d) where d = max\_depth)  
 if wikidata.is\_subclass\_of(c, r, max\_depth=3):  
 return True  
  
 # 4. WordNet ancestor (O(p) where p = hypernym path length)  
 if resolver.is\_ancestor(r, c):  
 return True  
  
 # 5. WordNet synset overlap (O(s1 × s2) where s = synsets)  
 if get\_synsets(c) & get\_synsets(r):  
 return True  
  
 # 6. Morphological matching (O(min(len(c), len(r))))  
 if common\_prefix\_length(c, r) >= 5:  
 return True  
  
 # 7. BabelNet synonyms (O(API call))  
 if c in babelnet.get\_synonyms(r) or r in babelnet.get\_synonyms(c):  
 return True  
  
 return False

## 14. Codebase Discovery Summary

Total Files: ~80+ Python files  
Core Language: Python 3.13  
Primary Framework: FastAPI + Uvicorn  
Database: PostgreSQL (Supabase) + Qdrant (Vector)  
ML Models: GPT-4o + Sentence-Transformers  
External APIs: Wikidata, Nominatim, Frankfurter, BabelNet  
  
Key Components Found:  
 ✅ Backend APIs (FastAPI REST endpoints)  
 ✅ ML Pipeline (GPT-4o extraction + embeddings)  
 ✅ Semantic Matching Engine (Multi-strategy boolean matching)  
 ✅ Vector Database (Qdrant for semantic search)  
 ✅ Relational Database (PostgreSQL via Supabase)  
 ✅ Canonicalization Pipeline (Wikidata, WordNet, BabelNet)  
 ✅ Observability (Sentry + OpenTelemetry/Jaeger)  
 ✅ DevOps/CI (Docker + Railway/Render)  
  
Lines of Code:  
 - main.py: ~1,054 lines  
 - Total Python: ~15,000+ lines  
 - Tests: ~50+ test files

## 15. Architecture Diagrams

### 

### 15.1 System Architecture Diagram

## 

## 16. Actionable Recommendations

### 16.1 High Priority

| Issue | Location | Recommendation |
| --- | --- | --- |
| SQL Filter Optimization | retrieval\_service.py:135-165 | Replace Python-side filtering with PostgreSQL ?| operator for array overlap |
| Duplicate Legacy Code | matching/, schema/, canonicalization/ vs src/core/ | Complete migration to src/ structure, deprecate root modules |
| Embedding Model Loading | main.py startup | Implement lazy loading or warm-up endpoint |

### 16.2 Medium Priority

| Issue | Location | Recommendation |
| --- | --- | --- |
| Rate Limiting | /extract endpoint | Add rate limiting for OpenAI API calls |
| Caching | External API wrappers | Add Redis cache for Wikidata/BabelNet responses |
| Batch Ingestion | ingestion\_pipeline.py | Optimize with async/parallel writes |

### 16.3 Technical Debt

| Location | Issue | Severity |
| --- | --- | --- |
| main.py:238-246 | Hardcoded CURATED\_SYNONYMS | Medium |
| retrieval\_service.py:285-289 | Qdrant ID filter not implemented | Low |
| Root-level matching/, schema/ | Duplicate of src/core/ | Medium |

### 16.4 Scalability Recommendations

| Component | Concern | Mitigation |
| --- | --- | --- |
| GPT-4o | Rate limits, latency | Batch requests, response caching |
| Qdrant | Memory for 1024D vectors | Use 384D model, collection sharding |
| Supabase | Connection pooling | PgBouncer, async queries |
| Embedding | Model load time | Pre-warm, singleton pattern |

## Summary

**Singletap-Backend** is a sophisticated semantic matching engine that combines:

1. **GPT-4o** for natural language understanding
2. **Multi-source ontology resolution** (Wikidata, WordNet, BabelNet)
3. **Vector embeddings** for semantic similarity search
4. **Boolean constraint matching** for precise filtering
5. **Comprehensive observability** (Sentry, Jaeger, structlog)

The architecture follows a clear pipeline:

Extract → Canonicalize → Normalize → Retrieve → Match → Store

Each stage is optimized for its specific task, with fallback mechanisms and caching to ensure reliability and performance.

**Architecture Grade: A-**

| Aspect | Assessment |
| --- | --- |
| Code Quality | Good - modular, well-documented |
| Test Coverage | Unit + Integration + Feature tests present |
| Deployment Readiness | Production-ready (Docker, PaaS configs) |
| Observability | Excellent (Sentry + Jaeger + structlog) |
| Scalability | Medium - needs SQL optimization, caching |
| Security | API key management via env vars |

*Generated by Principal Architect Analysis* *Singletap-Backend v2.0 | February 16, 2026*