Current State: Local Semantic Search Stack for Journal Content
Host & Environment
- Machine: MacBook Pro (Apple Silicon, 2025 M4)
- OS: macOS (Apple Silicon / arm64)
- Shell: zsh
- Tools: docker, ollama, python3.12 + virtualenv, jq

1. Core Services

1.1 Qdrant (Vector Database)
- Deployment: Docker container
  - Container name: qdrant
  - Image: qdrant/qdrant:latest
  - Start command:
    docker start qdrant
- Port mapping:
  - Host: 127.0.0.1:6333
- Health check:
  curl -s http://127.0.0.1:6333/healthz
  -> healthz check passed

- Main collection: articles_ollama
  - Vector size: 768
  - Distance: Cosine
  - Default unnamed vector (no named vector key)
- Count example:
  curl -s -X POST http://127.0.0.1:6333/collections/articles_ollama/points/count -H 'Content-Type: applica
  -> result.count ≈ 160+ (subject to re-ingest)

Payload structure per point:
```
{
  "document": "<chunk text>",
  "metadata": {
    "title": "<string>",
    "doi": "<string or null>",
    "year": <integer or null>,
    "source": "local-pdf",
    "chunk": "<zero-padded chunk id, e.g. '0028'>"
  }
}
```

Indexes configured in Qdrant:
- metadata.title -> keyword
- metadata.doi -> keyword
- metadata.year -> integer
- metadata.source -> keyword

1.2 Ollama (Embeddings)
- Local service: Ollama

- Base URL: http://127.0.0.1:11434
- Embedding model: nomic-embed-text
  - Dimension: 768

Embedding request contract:
POST /api/embeddings
Body:
{
  "model": "nomic-embed-text",
  "prompt": "<text>"
}

Example verification:
curl -s http://127.0.0.1:11434/api/embeddings   -d '{"model":"nomic-embed-text","prompt":"hello world"}' | j
-> 768

2. Python Environment & Ingestion

2.1 Virtualenv
- Path: ~/.venvs/qdrant-mcp-312
- Activate:
  . ~/.venvs/qdrant-mcp-312/bin/activate
- Python: 3.12.x
- Libraries (relevant):
  - pypdf
  - tiktoken
  - requests
  - fastapi
  - uvicorn

2.2 Ingestion Script

- Script path: /Users/jsmith/Ingest.py
- Purpose: read PDF, chunk text, embed via Ollama, upsert into Qdrant collection articles_ollama.

Effective config inside script:
QDRANT_URL = os.environ.get("QDRANT_URL", "http://127.0.0.1:6333").rstrip("/")
COLLECTION = os.environ.get("QDRANT_COLLECTION", "articles_ollama")
OLLAMA_URL = os.environ.get("OLLAMA_URL", "http://127.0.0.1:11434").rstrip("/")
EMBED_MODEL = "nomic-embed-text"
EMBED_DIM = 768

CLI usage pattern:
python /Users/jsmith/Ingest.py   --pdf <path-to-pdf>   --title "<Article Title>"   --doi "<DOI or omit>"   --y

Behavior summary:
1) Read PDF via pypdf from --pdf.
2) Extract text.
3) Chunk using tiktoken, approx:
   - Target tokens per chunk: ~512

- Overlap: ~128 tokens
4) For each chunk:
- Call OLLAMA_URL/api/embeddings with model = nomic-embed-text.
- Receive 768-d float vector.
- Build Qdrant point:
- id: UUID (deterministic per chunk; re-ingest overwrites)
- vector: embedding
- payload:
- document: chunk text
- metadata:
- title: CLI arg
- doi: CLI arg or null
- year: CLI arg or null
- source: "local-pdf"
- chunk: zero-padded chunk index string ("0000", "0001", ...)
5) Upsert batch into Qdrant collection articles_ollama using QDRANT_URL.

Example successful ingests:
Article 5:
python /Users/jsmith/Ingest.py --pdf ~/Documents/article5.pdf --title "Article 5" --doi "10.1056
-> [ingest] upserted 38 chunks in ~5s into 'articles_ollama'

Article 4:
python /Users/jsmith/Ingest.py --pdf ~/Documents/article4.pdf --title "Article 4" --doi "10.1056
-> [ingest] upserted 36 chunks in ~7s into 'articles_ollama'

Other ingested items:
- "Demo PDF" and others, typically with doi = null but same metadata structure.

3. Semantic Query Patterns

3.1 Get embedding for a natural-language query
Example query: "aortic valve replacement long-term outcomes"

```
QEMB=$(curl -s http://127.0.0.1:11434/api/embeddings -d '{"model":"nomic-embed-text","prompt":"aortic
```

QEMB is a JSON array string of 768 floats.

3.2 Basic semantic search returning top chunks

```
jq -n --argjson v "$QEMB" '{
  vector: $v,
  limit: 10,
  with_payload: true
}' | curl -s -X POST http://127.0.0.1:6333/collections/articles_ollama/points/search -H 'Content-Type: ap
```

This returns multiple chunks per article.

3.3 Grouped results: one row per article (title + year + doi)

```
jq -n --argjson v "$QEMB" '{
```

```
  vector: $v,
  limit: 200,
  with_payload: true
}' | curl -s -X POST http://127.0.0.1:6333/collections/articles_ollama/points/search      -H 'Content-Type: ap
  [ .result[]
    | {
        score,
        title: .payload.metadata.title,
        doi: .payload.metadata.doi,
        year: .payload.metadata.year,
        chunk: .payload.metadata.chunk
      }
  ]
  | group_by(.title, .doi, .year)
  | map(max_by(.score))
  | sort_by(-.score)
'
```

Example observed result (for the valve query):

```
[
  { "score": 0.5537, "title": "Article 5", "doi": "10.1056/NEJM0000005", "year": 2025, "chunk": "0037" },
  { "score": 0.5175, "title": "Demo PDF", "doi": null, "year": null, "chunk": "0041" },
  { "score": 0.5172, "title": "Article 4", "doi": "10.1056/NEJM0000004", "year": 2024, "chunk": "0031" },
  { "score": 0.5168, "title": "Article 3", "doi": null, "year": 2023, "chunk": "0047" }
]
```

3.4 Ranked DOI list only (semantic → DOI ranking)

To produce: given a natural language query, return DOIs sorted by semantic match quality.

```
QEMB=$(curl -s http://127.0.0.1:11434/api/embeddings   -d '{"model":"nomic-embed-text","prompt":"aortic

jq -n --argjson v "$QEMB" '{
  vector: $v,
  limit: 200,
  with_payload: true
}' | curl -s -X POST http://127.0.0.1:6333/collections/articles_ollama/points/search      -H 'Content-Type: ap
  [ .result[]
    | {
        score,
        doi: .payload.metadata.doi
      }
    | select(.doi != null)
  ]
  | group_by(.doi)
  | map(max_by(.score))
  | sort_by(-.score)
  | map(.doi)
'
```

Example output:

[
  "10.1056/NEJM0000005",
  "10.1056/NEJM0000004"
]

4. Assumptions for IDE / Code Generation

Code can safely assume:
- Qdrant is reachable at http://127.0.0.1:6333 with collection articles_ollama, 768-dim cosine vectors.
- Each point has payload.document (chunk text) and payload.metadata as specified.
- Ollama is reachable at http://127.0.0.1:11434 and supports POST /api/embeddings with model = "nomic-
- /Users/jsmith/Ingest.py exists and accepts arguments:
  --pdf, --title, --doi, --year
  and writes data into articles_ollama as described.

Primary tasks for any new code:
- Encode user query with Ollama.
- Perform Qdrant points/search.
- Aggregate results by doi (and/or title, year) for ranked article-level output.