

Eugene PubMed Search — End-to-End Flow

Developer walkthrough: HTTP → module-load DI → Neo4j paginated Cypher → PubMedSearchResult JSON → upstream NCBI download + LLM extraction ETL

Audience

Engineers who need a file-referenced trace of the PubMed search endpoints. These routes are the paginated companions to `/count/pmid/*` — clients call the count endpoint first to compute the maximum page number, then page through this endpoint. Every reference uses the form `path/to/file.py:line`.

Reference endpoints

- GET `/pmids/drugs?drug_id=DB00683&page=1&page_size=50`
- GET `/pmids/clinicaltrials?nct_id=NCT00000122&page=1&page_size=50`
- GET `/pmids/geneproteins?gene_protein=PDE3A&page=1&page_size=50`

All three are declared in `src/foundation/router/pubmed_search_router.py` and registered by `src/eugene_ws.py:50` / `src/eugene_ws.py:72` inside `add_routers(app)`.

Relationship to the pubmed-count route

The search routes share the `Neo4jPubmedQueryAdapter` singleton only conceptually with the count route — the count route uses a separate `Neo4jPubmedCountAdapter` (`conf.py:209-216`) while this route uses `Neo4jPubmedQueryAdapter` (`conf.py:219-226`). Both adapters walk the same edges with the same anchors; the search route adds SKIP/LIMIT pagination and projects `pmid` instead of `count(*)`. If the count returns 137 PMIDs for a drug, paging this route with `page_size=50` yields 50 / 50 / 37 across three pages.

How to read this guide

- Section 0 — schema (node labels + edge types the Cypher uses). Read first.
- Sections 1-3 — HTTP, Cypher, response shape, top-down.
- Section 4 — ETL: NCBI fetch + PDF download + LLM-driven graph link step.
- Section 5 — copy-paste debug walk.
- Section 6 — flat reference index.

0. Schema (read this first)

The pubmed-search routes anchor on one of three node types and traverse to a `:Research` node carrying a `pmid` property. The drug + gene_protein variants are a single labeled hop; the clinical trial variant is a two-hop traversal because the graph does not store a direct `ClinicalTrial` -> `Research` edge.

```
(:drug { node_id }) -[:featured_in]- (:Research { pmid })
(:gene_protein { node_name }) -[:analyzed_in]- (:Research { pmid })

(:ClinicalTrial { nct_id }) -[ANY]-(:drug | :gene_protein) -[ANY]- (:Research { pmid })
```

- Node labels come from `FoundationalNodeEnum` in `src/foundation/model/foundational_node_enum.py`: `DRUG` (line 14, label `drug`), `CLINICAL_TRIAL` (line 10, label `ClinicalTrial`), `GENE_PROTEIN` (line 18, label `gene_protein`), and `RESEARCH` (line 33, label `Research`). Note the mixed case — `Research` and `ClinicalTrial` keep `PascalCase` in `Neo4j`, the rest are lowercase.
- Relationship types come from `FoundationalRelationshipEnum` in `src/foundation/model/foundational_relationship_enum.py`: `FEATURED_IN` (line 28, type `featured_in`) and `ANALYZED_IN` (line 29, type `analyzed_in`). The clinical-trial Cypher uses an *unlabeled* relationship pattern (`-[r]-`) on both hops — any edge type is accepted.
- Anchor keys differ per route: drug anchors on `n.node_id`, clinical trial on `start.nct_id` (not `node_id`!), and gene/protein on `n.node_name` (also not `node_id`).
- The `pmid` property lives on `:Research` as `n2.pmid / end.pmid`; the drug + gene routes project it directly, the clinical-trial route projects it as `end.pmid` after the second hop.
- Edges are all **undirected** in the Cypher (`-[r: `featured_in`]` - not `-[r: `featured_in`]->`); the search is symmetric with respect to direction in the graph.
- Like the rest of Eugene there are no `Neo4j` `CREATE CONSTRAINTS`; uniqueness of `pmid` is by ETL convention, not enforced at the DB layer.

1. HTTP entry — the FastAPI router

Wiring

src/eugene_ws.py:50 imports the module as `pubmed_search_router`; src/eugene_ws.py:72 adds it to the includes list inside `add_routers(app)`. The module-level router = `APIRouter(prefix="/pmids", tags=["pubmed"])` (pubmed_search_router.py:29-32) means the final paths are `/pmids/drugs`, `/pmids/clinicaltrials`, and `/pmids/geneproteins`.

Module-load dependency injection

src/foundation/router/pubmed_search_router.py:34-35

```
query_adapter = neo4j_pubmed_query_adapter()
search_result_mapper = pubmed_search_result_mapper()
```

The DI factories at src/foundation/conf/conf.py:219-226

(neo4j_pubmed_query_adapter) and src/foundation/conf/conf.py:271-272

(pubmed_search_result_mapper) resolve to fresh instances bound to the shared

`_neo4j_driver()` singleton. There is no caching layer between FastAPI and the adapter — every HTTP request goes to Neo4j.

Endpoint signatures

```
@router.get("/drugs", response_model=PubmedSearchResult) # line 38
async def find_related_pubmed_docs_by_drug_id(drug_id, page, page_size):
```

```
@router.get("/clinicaltrials", response_model=PubmedSearchResult) # line 79
async def find_related_pubmed_docs_by_clinicaltrial_id(nct_id, page, page_size):
```

```
@router.get("/geneproteins", response_model=PubmedAndGeneSearchResult) # line 122
async def find_related_pubmed_docs_by_geneprotein_id(gene_protein, page, page_size):
```

Query params & validators

- `drug_id` (lines 44-53) — `Query(example="DB00683", max_length=64, min_length=1)` + `AfterValidator(validate_value).validate_value(validate_util.py:35-42)` rejects any of `% _ $ ; : ^ *`. Note: it does *not* enforce the DB prefix — that stricter check lives in `validate_drug_id` (lines 58-68) which this route does not use.
- `nct_id` (lines 85-94) — `max_length=24, min_length=1` + `AfterValidator(validate_clinical_trial_id)(validate_util.py:45-55)`, which delegates to `validate_value` *and* additionally requires the NCT prefix (case-insensitive via `id.upper().startswith("NCT")`).
- `gene_protein` (lines 128-137) — `max_length=24, min_length=1` + `AfterValidator(validate_gene_protein)(validate_util.py:71-77)`; same special-char check, no prefix requirement.
- `page` — required, `gt=0`. `page_size` — required, `gt=0, le=100`. Pydantic returns a 422 if missing or out of range; clients are expected to read the max page from the `/count/pmids/*` endpoints.

Normalization

Each handler calls the matching `case_helper` normalizer before hitting the adapter:

`normalize_drug_id(drug_id)` (line 72), `normalize_clinical_trial_id(nct_id)` (line 113), and `normalize_gene_protein(gene_protein)` (line 156). All three live in

`src/foundation/model/case_helper.py` and are identical — each delegates to `_normalize_upper` which uppercases the value. Casing on the wire (e.g. `db00683` vs. `DB00683`, `pde3a` vs. `PDE3A`) does not miss the Neo4j anchor.

First breakpoint: `pubmed_search_router.py:73, 114, or 157` (the adapter call).

2. Query adapter — the paginated Cypher

Neo4jPubmedQueryAdapter lives at `src/foundation/infra/db/adapter/neo4j_pubmed_query_adapter.py`. Each search entry point opens a session, begins a transaction, calls the matching private helper, and returns a pandas DataFrame (via `tx.run(...).to_df()`).

Public entry points

- `find_related_pubmed_by_drug(drug_id, page, page_size)` (lines 27-40).
- `find_related_pubmed_by_clinicaltrail(nct_id, page, page_size)` (lines 42-55).
- `find_related_pubmed_by_gene(gene, page, page_size)` (lines 57-70).
- All three call `ensure_connection(self.driver)` (reconnect-on-failure helper from `graph.util.util`), open a session, run a single transaction (`session.begin_transaction()`), and the inner helper does `tx.run(search, params).to_df()`.
- Pagination is computed by `Pagination.calculate_limit_and_offset(page, page_size)` from `foundation/infra/db/util/pagination.py`; the resulting integers are interpolated into the Cypher via `%s` (not bound parameters).

Cypher — /pmids/drugs (lines 95-108)

```
MATCH (n:`drug`)-[r:`featured_in`]- (n2:`Research`)
WHERE n.node_id = $drug_id
RETURN n.node_id as drug_id, n.node_name as drug_name, n2.pmid as pmid
ORDER BY pmid DESC
SKIP {offset}
LIMIT {limit}
```

Cypher — /pmids/clinicaltrials (lines 137-152)

```
MATCH (start:`ClinicalTrial`)-[r]- (target:`drug`|`gene_protein`)
WHERE start.nct_id = $nct_id
OPTIONAL MATCH (target)-[r2]- (end:`Research`)
RETURN start.nct_id as nct_id, target.node_id as target_id, end.pmid as pmid
ORDER BY nct_id, target_id, pmid DESC
SKIP {offset}
LIMIT {limit}
```

Cypher — /pmids/geneproteins (lines 177-190)

```
MATCH (n:`gene_protein`)-[r:`analyzed_in`]- (n2:`Research`)
WHERE n.node_name = $gene
RETURN n.node_name as node_name, n2.pmid as pmid
ORDER BY pmid DESC
SKIP {offset}
LIMIT {limit}
```

Pagination and ordering semantics worth noting

- Ordering is by `pmid DESC` (a lexicographic string sort, not numeric) for the drug and gene routes. PMIDs happen to be numeric strings of similar length so the ordering looks newest-first in practice, but a 7-digit pmid will sort before an 8-digit one of higher numeric value. If ordering correctness matters, cast: `ORDER BY toInteger(n2.pmid) DESC`.
- The clinical-trial route additionally orders by `nct_id, target_id` before `pmid DESC` — this groups results by intermediate target before pmid, which means paginating a single `nct_id` can interleave PMIDs from different drugs/genes within the same page.

- The clinical-trial route uses `OPTIONAL MATCH` on the second hop and projects `end.pmid`; if a matched target has no linked `:Research` node the row still appears with `pmid = null`. The mapper filters those out (see Section 3, `_to_row`).
- `SKIP` and `LIMIT` are **integers interpolated into the Cypher string** — the only bound parameter is the anchor id (`$drug_id / $nct_id / $gene`). FastAPI's range constraints (`gt=0, le=100`) plus Pagination's integer math make this safe.
- Each helper catches `(DriverError, Neo4jError)`, logs, and re-raises — Neo4j outages surface as a FastAPI 500.
- Labels and relationship strings are interpolated via `%s` substitution from `FoundationalNodeEnum.value[1] / FoundationalRelationshipEnum.value[1]`.

3. Mapper / response model

PubmedSearchResultMapper

(src/foundation/mapper/pubmed_search_result_mapper.py) converts the adapter's DataFrame into a typed frozen dataclass declared in src/foundation/model/pubmed/. The drug and clinical-trial routes return PubmedSearchResult; the gene/protein route returns PubmedAndGeneSearchResult (same shape but echoes gene_protein instead of search_id).

Models (verbatim)

```
# src/foundation/model/pubmed/pubmed_search_result.py
@dataclass(frozen=True, unsafe_hash=True)
class PubmedSearchResult:
    search_id: str
    count: int
    results: list[PubmedSearchResultRow]

# src/foundation/model/pubmed/pubmed_and_gene_search_result.py
@dataclass(frozen=True, unsafe_hash=True)
class PubmedAndGeneSearchResult:
    gene_protein: str
    count: int
    results: list[PubmedSearchResultRow]

# src/foundation/model/pubmed/pubmed_search_result_row.py
@dataclass(frozen=True, unsafe_hash=True)
class PubmedSearchResultRow:
    pmid: str
```

Mapper behaviour

- map_drug_response (lines 17-30) and map_clinicaltrail_response (lines 32-45) both return PubmedSearchResult(search_id=..., count=..., results=[]) when the DataFrame is None or empty.
- map_gene_response (lines 47-62) returns PubmedAndGeneSearchResult(gene_protein=..., count=0, results=[]) for the empty case.
- Row conversion is _to_row (lines 64-69): it reads the pmid column, skips rows where pmid is None (this is what filters out the OPTIONAL MATCH no-Research rows from the clinical trial route), and stringifies the value into PubmedSearchResultRow(pmid=str(row["pmid"])).
- All three map methods call _check_id_and_log (lines 71-76) which compares the request id to the first row's echo column and logs a WARNING if they differ — the response still ships, so any mismatch surfaces in logs rather than as a 5xx.
- count in the response is len(results) after the None-pmid filter; it is the size of the returned page, **not** the total matching across all pages. The total-across-all-pages number comes from /count/pmids/*.
- response_model_exclude_none=True on every route means absent/None fields are stripped from the JSON, but for this schema every field is required so the output is stable.

Example JSON — /pmids/drugs

GET /pmids/drugs?drug_id=DB00683&page=1&page_size=3

```
{
  "search_id": "DB00683",
  "count": 3,
```

```
"results": [  
  { "pmid": "39912345" },  
  { "pmid": "39812003" },  
  { "pmid": "39711842" }  
]  
}
```

Example JSON — /pmids/geneproteins

GET /pmids/geneproteins?gene_protein=PDE3A&page=1&page_size=2

```
{  
  "gene_protein": "PDE3A",  
  "count": 2,  
  "results": [  
    { "pmid": "39801122" },  
    { "pmid": "39600711" }  
  ]  
}
```


4. ETL pipeline — how PMIDs get into Neo4j

The search route is read-only; it assumes the `:Research` nodes and their `:featured_in / :analyzed_in` edges already exist. The pipeline has two top-level entry points, both shell scripts that boot the venv and run a Python module.

Stage 1 — NCBI download

[src/download_pubmed.py](#)

- CLI: `--search-terms` (a list of disease/drug strings) or `--download-by-pmcids` (an explicit list of PMC ids). The two are mutually exclusive (`add_mutually_exclusive_group`, line 50).
- Wiring: `pubmed_orchestrator()` from `pubmed.conf.conf` returns a `PubmedOrchestrator` (`src/pubmed/provider/pubmed_orchestrator.py`) composed of a search provider, a fetch provider, a download provider, and a Neo4j article adapter.
- For each pmc id, `orchestrator.fetch_and_download` sleeps `FETCH_THROTTLE_SECS = 10` (line 13) to respect NCBI rate limits, calls `PubmedFetchProvider.fetch_by_id(pmcid)` for metadata, picks a `pdf_uri` or falls back to the doi (`_pick_pdf_name`, lines 61-65), downloads the PDF via `PubmedDownloadProvider.download(...)`, and upserts the `:Research` node via `Neo4jArticleAdapter.upsert_article(article)`.
- On any exception per pmc id the orchestrator counts a failure and continues; nothing is retried inside the process (re-run the CLI to retry).

Stage 2 — link articles into the graph

[src/link_pubmed_articles.py](#)

- CLI flag: `--use-checkpoints`. When set, `process_checkpoints` (lines 58-93) reads a hard-coded list of PDF file names (`generate_files_to_load()`), maps each to a checkpoint directory under `output/<file_hash>`, loads pre-extracted triples via `stored_triples_loader().load([checkpoint])`, then feeds them into `EugeneGraphSummaryOrchestrator.load_from_entities(pmcid, entities)` and `summarize_communities(...)`.
- When the flag is omitted, `build_knowledge_graph` (lines 96-108) drives a `KnowledgeGraphDirAnalyzer` over `./resources/data/pubmed` (PDFs only) which runs the per-document LLM extraction live — expensive, but used to bootstrap new corpora.
- The summary orchestrator (`eugene_graph_summary_orchestrator(max_workers=3)`, line 40) is the component that emits the `:featured_in` and `:analyzed_in` edges between `:drug / :gene_protein` anchors and the `:Research` node. The drug-anchor edges come from extracted drug mentions; the gene-anchor edges from extracted gene/protein mentions.
- Embeddings are computed during the load via `FoundationalNodeEmbeddingProvider` (see `conf.py`: 247-248); they support the similarity route, not the pubmed search route itself.

Fix-up tooling

`src/fix_pubmed_articles.py` (CLI driving `PubmedFixOrchestrator` from `src/pubmed/provider/pubmed_fix_orchestrator.py`) is the maintenance script for repairing `Research` nodes whose metadata is stale (missing keywords, missing embedding, etc.). It does not create new edges — only the two stages above add the edges this search route walks.

5. Suggested debugging walk

- Start the service locally: `uvicorn src.eugene_ws:app --reload`. Confirm Neo4j is up (`docker compose ps eugene-neo4j`) and that the pubmed subgraph is loaded (`MATCH (r:`Research`) RETURN count(r);` should be non-zero).
- Hit each endpoint with curl: `curl "http://localhost:8080/pmids/drugs?drug_id=DB00683&page=1&page_size=10"`, then the clinicaltrials (e.g. NCT00000122) and gene/proteins (e.g. PDE3A) variants. Expect `{ "search_id": ..., "count": N, "results": [...] }` (or `gene_protein` key for the third route).
- Breakpoint at `pubmed_search_router.py:73` (the adapter call). Step into `Neo4jPubmedQueryAdapter.find_related_pubmed_by_drug` (`neo4j_pubmed_query_adapter.py:27`). Inspect the assembled Cypher string at `neo4j_pubmed_query_adapter.py:91-108` (it is also emitted at INFO via the `logger.info(f"search: {search}")` on line 78).
- Copy that Cypher into `cypher-shell` and run with `:param drug_id => "DB00683"`. Row count must equal count in the HTTP response. If it does not, candidate explanations are (a) `normalize_drug_id` mutated the id (re-check the log line searching for related pubmed by drug: ...), (b) you are pointed at a different Neo4j instance than the API is, or (c) you forgot the SKIP/LIMIT when running by hand — the API always paginates.
- Force the validator: `curl ".../pmids/drugs?drug_id=DB%24000683&page=1&page_size=10"` (URL-encoded \$). Expect a 422 from `validate_value`. Repeat for the clinicaltrials route with a missing NCT prefix (e.g. `?nct_id=00000122`); expect a 422 from `validate_clinical_trial_id`. Repeat with `page_size=101`; expect 422 from FastAPI's `le=100`.
- Cross-check pagination against the count route: `curl ".../count/pmids/drugs?drug_id=DB00683"` and confirm `ceil(pubmed_count / page_size)` pages exist. Page 1 + page 2 + ... should sum back to `pubmed_count`; divergence indicates a duplicate `:featured_in` edge or a `count(*)` vs `count(DISTINCT)` mismatch worth investigating on the count adapter side.

6. Reference index (file paths)

Route & wiring

```
src/eugene_ws.py:50,72      (import + add_routers)
src/foundation/router/pubmed_search_router.py
src/foundation/router/validate_util.py:35-42  (validate_value)
src/foundation/router/validate_util.py:45-55  (validate_clinical_trail_id)
src/foundation/router/validate_util.py:71-77  (validate_gene_protein)
src/foundation/model/case_helper.py           (normalize_*)
src/foundation/conf/conf.py:219-226           (neo4j_pubmed_query_adapter)
src/foundation/conf/conf.py:271-272           (pubmed_search_result_mapper)
```

Adapter (Cypher)

```
src/foundation/infra/db/adapter/neo4j_pubmed_query_adapter.py
:27-40  find_related_pubmed_by_drug
:42-55  find_related_pubmed_by_clinicaltrail
:57-70  find_related_pubmed_by_gene
:72-108 _find_related_pubmed_by_drug + Cypher generator
:110-152 _find_related_pubmed_by_clinicaltrail + Cypher generator
:154-190 _find_related_pubmed_by_gene + Cypher generator
src/foundation/infra/db/util/pagination.py      (Pagination.calculate_limit_and_offset)
```

Mapper & response models

```
src/foundation/mapper/pubmed_search_result_mapper.py
:17-30  map_drug_response
:32-45  map_clinicaltrail_response
:47-62  map_gene_response
:64-69  _to_row
:71-76  _check_id_and_log
src/foundation/model/pubmed/pubmed_search_result.py
src/foundation/model/pubmed/pubmed_and_gene_search_result.py
src/foundation/model/pubmed/pubmed_search_result_row.py
```

Domain model (labels & rel types)

```
src/foundation/model/foundational_node_enum.py
:10  CLINICAL_TRIAL = ClinicalTrial
:14  DRUG            = drug
:18  GENE_PROTEIN    = gene_protein
:33  RESEARCH        = Research
src/foundation/model/foundational_relationship_enum.py
:28  FEATURED_IN    = featured_in
:29  ANALYZED_IN    = analyzed_in
```

Companion count route (same anchors, count(*) instead of pagination)

```
src/foundation/router/pubmed_count_router.py
src/foundation/infra/db/adapter/neo4j_pubmed_count_adapter.py
src/foundation/conf/conf.py:209-216      (neo4j_pubmed_count_adapter)
```

ETL

```
src/download_pubmed.py                (NCBI fetch + PDF download CLI)
src/pubmed/provider/pubmed_orchestrator.py  (search + fetch + download + upsert)
src/pubmed/provider/pubmed_search_provider.py (NCBI e-search wrapper)
src/pubmed/provider/pubmed_fetch_provider.py (NCBI e-fetch wrapper)
src/pubmed/provider/pubmed_download_provider.py (PDF retrieval)
src/pubmed/infra/db/neo4j_article_adapter.py (Research-node upsert)
src/link_pubmed_articles.py             (LLM extraction + edge build CLI)
src/fix_pubmed_articles.py              (metadata + embedding fix-up)
src/pubmed/provider/pubmed_fix_orchestrator.py
```

End of pubmed-search route flow guide.