

Eugene PubMed Count — End-to-End Flow

Developer walkthrough: HTTP → module-load DI → Neo4j COUNT Cypher → {search_key, pubmed_count} JSON

Audience

Engineers who need a file-referenced trace of the pubmed-count endpoints. These routes are the cheap companions to the full `/pmids/*` search endpoints — clients typically call the count first to size a result set or compute the maximum page number, then page through the search. Every reference uses the form `path/to/file.py:line`.

Reference endpoints

- GET `/count/pmids/drugs?drug_id=DB00683`
- GET `/count/pmids/clinicaltrials?nct_id=NCT05648006`
- GET `/count/pmids/geneproteins?gene_protein=ABL1`

All three are declared in `src/foundation/router/pubmed_count_router.py` and registered by `src/eugene_ws.py:51` (import) and `src/eugene_ws.py:73` (the includes list inside `add_routers(app)`).

Relationship to the pubmed-search route

The count and search routes are wired through *different* adapters in this codebase:

`Neo4jPubmedCountAdapter` for `/count/pmids/*` and `Neo4jPubmedQueryAdapter` for `/pmids/*`. Both factories live in `src/foundation/conf/conf.py` (lines 209-216 and 219-226) and resolve through the shared `_neo4j_driver()` singleton, so they share a connection pool even though the classes differ.

How to read this guide

- Section 0 — schema (anchor nodes + the rels into `:Research`). Read first.
- Sections 1-3 — HTTP, Cypher, response shape, top-down.
- Section 4 — pointers to the PubMed ETL (`src/download_pubmed.py` + `src/link_pubmed_articles.py`).
- Section 5 — copy-paste debug walk.
- Section 6 — flat reference index.

0. Schema (read this first)

There is a subtle naming gotcha at the heart of this route. The node label written into Neo4j for an ingested PubMed article is `:Research`, not `:pubmed_document`. The `FoundationalNodeEnum.PUBMED_DOCUMENT` symbol exists (`foundational_node_enum.py:45`, label `pubmed_document`) but the count adapter does *not* use it — it walks `FoundationalNodeEnum.RESEARCH` (`foundational_node_enum.py:33`, label `Research`). Anyone wiring up new pubmed reads should verify which label their data actually carries before writing Cypher.

```
(:drug { node_id }) -[:featured_in]- (:Research { pmid })
(:gene_protein { node_name }) -[:analyzed_in]- (:Research { pmid })
(:ClinicalTrial { nct_id }) -[r]-(:drug|:gene_protein)-[r2]-(:Research { pmid })
# clinical-trial route is a 2-hop variable-rel walk; see Section 2.
```

- Anchor 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`).
- Relationship types come from `FoundationalRelationshipEnum` in `src/foundation/model/foundational_relationship_enum.py`: `FEATURED_IN` (line 28, `drug ↔ research`) and `ANALYZED_IN` (line 29, `gene ↔ research`). The clinical-trial Cypher does **not** name a relationship type at all — it uses an untyped `-[r]-` and an `OPTIONAL MATCH` second hop.
- 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 `:Research` node carries a `pmid` property — that is what the clinical-trial Cypher counts (`count(end.pmid)`), whereas the `drug` and `gene` Cypher count node rows (`count(n2)`). The numbers are normally equal because every ingested article has a `pmid`, but null `pmids` would be silently dropped by the clinical-trial form — not by the other two.
- Like the rest of Eugene there are no Neo4j `CREATE CONSTRAINTS`; uniqueness of `pmid` is by ETL convention (`MERGE` on the article during ingest), not enforced by the DB.

1. HTTP entry — the FastAPI router

Wiring

src/eugene_ws.py:51 imports the module as `pubmed_count_router`; src/eugene_ws.py:73 adds it to the includes list inside `add_routers(app)`. The module-level router = `APIRouter(prefix="/count/pmid", tags=["pubmed"])` (pubmed_count_router.py:23-26) means the final paths are `/count/pmid/drugs`, `/count/pmid/clinicaltrials`, and `/count/pmid/geneproteins`.

Module-load dependency injection

src/foundation/router/pubmed_count_router.py:28

```
count_adapter = neo4j_pubmed_count_adapter()
```

The DI factory at `src/foundation/conf/conf.py:209-216` resolves to a `Neo4jPubmedCountAdapter` bound to the shared `_neo4j_driver()` singleton. Resolution happens at **module load time**, not per request — one instance is reused for the life of the process.

Endpoint signatures

```
@router.get("/drugs", response_model_exclude_none=True) # line 31
async def count_related_pubmed_docs_by_drug_id(drug_id):
```

```
@router.get("/clinicaltrials", response_model_exclude_none=True) # line 49
async def count_related_pubmed_docs_by_clinical_trial_id(nct_id):
```

```
@router.get("/geneproteins", response_model_exclude_none=True) # line 67
async def count_related_pubmed_docs_by_gene_protein_id(gene_protein):
```

Query params & validators

- `drug_id` (lines 33-42) — `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 — the stricter `validate_drug_id` (lines 58-68) is not used here.
- `nct_id` (lines 51-60) — `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).
- `gene_protein` (lines 69-78) — `max_length=24, min_length=1 + AfterValidator(validate_gene_protein)` (validate_util.py:71-77); same special-char check, no prefix requirement.
- All three routes **require** their query parameter (no defaults); Pydantic returns a 422 if missing.

Normalization

Each handler calls the matching `case_helper` normalizer before hitting the adapter: `normalize_drug_id(drug_id)` (line 44), `normalize_clinical_trial_id(nct_id)` (line 62), and `normalize_gene_protein(gene_protein)` (line 80). All three live in `src/foundation/model/case_helper.py`; they exist so casing on the wire (e.g. `db00683` vs. `DB00683`, or `abl1` vs. `ABL1`) does not miss the Neo4j anchor.

First breakpoint: `pubmed_count_router.py:45, 63, or 81` (the adapter call).

2. Query adapter — the COUNT Cypher

Neo4jPubmedCountAdapter lives at `src/foundation/infra/db/adapter/neo4j_pubmed_count_adapter.py`. Each count entry point opens a session, begins a transaction, calls the matching private helper, and returns the integer.

Public entry points

- `count_related_pubmed_by_drug(drug_id) -> int` (lines 25-34).
- `count_related_pubmed_by_clinicaltrail(nct_id) -> int` (lines 36-47). Note the misspelling — `clinicaltrail` not `clinicaltrial`. The handler (`pubmed_count_router.py:63`) calls it by that name; do not silently rename it.
- `count_related_pubmed_by_gene(gene) -> int` (lines 49-58).
- All three are decorated with `@log_time` (`annotation.timer_annotation`) so the elapsed query time appears in the logs at INFO.
- Each helper does `ensure_connection(self.driver)` (`graph/util/util.py`) before opening a session — a stale driver surfaces here rather than inside the transaction.
- Each inner helper catches (`DriverError`, `Neo4jError`), logs, and re-raises — Neo4j outages surface as a FastAPI 500.

Cypher — `/count/pmid/drugs` (lines 83-92)

```
MATCH (n:`drug`)-[r:`featured_in`]-(:Research)
WHERE n.node_id = $drug_id
RETURN count(n2) as count
```

Cypher — `/count/pmid/clinicaltrials` (lines 117-128)

```
MATCH (start:`ClinicalTrial`)-[r]-(:Research)-[r2]-(:Research)
WHERE start.nct_id = $nct_id
OPTIONAL MATCH (target:`drug`|`gene`)-[r2]-(:Research)
RETURN count(end.pmid) as count
```

Cypher — `/count/pmid/geneproteins` (lines 153-162)

```
MATCH (n:`gene`)-[r:`analyzed_in`]-(:Research)
WHERE n.node_name = $gene
RETURN count(n2) as count
```

COUNT semantics worth noting

- The drug and gene Cypher use `count(n2)` — this counts *matched rows*, not distinct nodes. In this schema every `(anchor)-[r]-(:Research)` edge is one row, so the number equals the number of edges from the anchor, which equals the number of related articles assuming the ETL writes one edge per (anchor, pmid) pair. Duplicate edges would inflate the count; use `count(DISTINCT n2)` if you suspect that.
- The clinical-trial Cypher counts `end.pmid`, not `end`. With `OPTIONAL MATCH` a missing second hop produces a row whose `end` is null and whose `end.pmid` is null, and `count(end.pmid)` skips nulls. Net effect: trials that match a drug/gene but whose drug/gene has no `:Research` neighbour contribute 0, not 1, to the count. Switching to `count(end)` would over-report.
- None of the three Cypher matches are *distinct* on the final node. The two-hop clinical-trial walk can therefore double-count an article that is reachable via both a drug and a gene attached to the same trial. If exact distinct-article counts ever matter, switch to `count(DISTINCT end.pmid)`.

- All relationship matches are undirected (`- [r]` - not `- [r] ->`). Direction in the graph does not change the answer.
- If the anchor node does not exist, `count ( . . . )` returns 0 (not null) — the route returns `{"drug": . . . , "pubmed_count": 0}` rather than 404. The `result is None` guard in each helper (e.g. lines 76-77, 110-111, 146-147) is defensive; it would only fire if `tx.run( . . . ).single( )` returned no row at all, which `count ( . . . )` does not.
- Labels and relationship strings are interpolated into the Cypher via `%s` substitution from `FoundationalNodeEnum.value[1]` / `FoundationalRelationshipEnum.value[1]` — the *only* Cypher parameters are the anchor ids (`$drug_id` / `$nct_id` / `$gene`).

3. Response model — just a dict

Like the patent-count route, this endpoint has no Pydantic response model and no mapper. Each handler returns a plain Python dict; FastAPI serializes it directly. The decorator uses `response_model_exclude_none=True` but **no** `response_model= class`, so the body shape is whatever the handler returns.

Verbatim return statements

```
# pubmed_count_router.py:46
return {"drug": drug_id, "pubmed_count": count}

# pubmed_count_router.py:64
return {"nct_id": nct_id, "pubmed_count": count}

# pubmed_count_router.py:82
return {"gene_protein": gene_protein, "pubmed_count": count}
```

Example JSON

```
GET /count/pmid/drugs?drug_id=DB00683
-> { "drug": "DB00683", "pubmed_count": 128 }

GET /count/pmid/clinicaltrials?nct_id=NCT05648006
-> { "nct_id": "NCT05648006", "pubmed_count": 7 }

GET /count/pmid/gene/proteins?gene_protein=ABL1
-> { "gene_protein": "ABL1", "pubmed_count": 312 }
```

Implications

- The search-key field name differs per route (`drug` vs. `nct_id` vs. `gene_protein`). Clients cannot use a single key name to read back the echoed anchor — if you are building a generic UI, branch on the route.
- There is no Pydantic schema for this response, and the OpenAPI doc will show only `{}` (unknown object) for the body. If this becomes painful, lift the shape into e.g. `PubMedCountResponse(BaseModel)` and pass `response_model=PubMedCountResponse`.
- Page-math convention: clients divide `pubmed_count` by the search route's `page_size` (max 100) and ceil to get the max page for `/pmids/*` requests.

4. ETL — how the PubMed nodes get into Neo4j

The count routes are read-only; they assume that `:Research` nodes (with a `pmid` property) and the `:featured_in` / `:analyzed_in` edges to `:drug` and `:gene_protein` anchors already exist. The ingest is a two-stage CLI pipeline that this guide intentionally *references* rather than re-derives — the full PubMed PDF/LLM extraction pipeline deserves its own guide.

Stage 1 — download

[src/download_pubmed.py](#)

CLI entry point that pulls PubMed/PMC articles to disk under `resources/data/pubmed/` (PDFs by default). Wires together a downloader and dotenv config. This stage does not touch Neo4j — it produces files for the linker stage to process.

Stage 2 — link

[src/link_pubmed_articles.py](#)

CLI entry point that turns the downloaded articles into Neo4j nodes and edges. Two modes:

- **From disk** — default. Reads PDFs from `resources/data/pubmed/`, runs LLM extraction to find drug/gene mentions, and merges `:Research` nodes linked to existing canonical `:drug` / `:gene_protein` anchors.
- **From checkpoints** — `--use-checkpoints` flag; replays pre-processed JSON under `output/`. Useful for iterating on the Neo4j upsert without re-running the LLM.
- Both modes go through `eugene_graph_summary_orchestrator()` (DI in `graph/conf/conf.py`); the underlying writes use `MERGE` on `Research.pmid` so re-runs are idempotent.

Quick orientation

- If the count for a known drug looks low, the likely cause is an upstream LLM extraction miss in `link_pubmed_articles.py`, *not* a bug in this route. Verify with the Cypher in Section 2 run directly in `cypher-shell`.
- If the count is zero for every drug, suspect either an empty `:Research` table or a missing `:featured_in` edge type. `MATCH (r:`Research`) RETURN count(r);` is the first diagnostic; `MATCH ()-[r:`featured_in`]-() RETURN count(r);` is the second.
- Clinical-trial counts depend on the trial having at least one `:drug` or `:gene_protein` neighbour with `:Research` fan-out. Trials with neither will always report 0 — this is by design.

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);`).
- Hit each endpoint with curl: `curl "http://localhost:8080/count/pmid/drugs?drug_id=DB00683"`, then the `clinicaltrials` and `gene` variants. Expect a JSON dict with `pubmed_count` as an integer.
- Breakpoint at `pubmed_count_router.py:45` (the drug adapter call). Step into `Neo4jPubmedCountAdapter.count_related_pubmed_by_drug` (`neo4j_pubmed_count_adapter.py:26`). Inspect the assembled Cypher string at `neo4j_pubmed_count_adapter.py:83-92` — copy it into `cypher-shell` with `:param drug_id => "DB00683"` and the integer must match the HTTP response.
- Force the validator: `curl ".../count/pmid/drugs?drug_id=DB%24000683"` (URL-encoded \$). Expect a 422 from `validate_value`. Repeat for the `clinicaltrials` route with a missing NCT prefix; expect a 422 from `validate_clinical_trial_id`.
- Cross-check the clinical-trial two-hop count: run the Cypher at `neo4j_pubmed_count_adapter.py:117-128` directly, then re-run it with `count(end.pmid)` swapped to `count(DISTINCT end.pmid)`. A divergence means the trial is reachable to the same article through both a `:drug` and a `:gene_protein` — expected in a well-populated graph, and worth knowing before clients compare counts across routes.
- Suspect a label confusion? Confirm the actual labels in your Neo4j with `CALL db.labels()` `YIELD label WHERE label CONTAINS "Research" OR label CONTAINS "pubmed"` `RETURN label;` — you should see `Research` and possibly `pubmed_document` if older ingest data is still present. The count adapter only reads `:Research`.

6. Reference index (file paths)

Route & wiring

```
src/eugene_ws.py:51,73      (import + add_routers)
src/foundation/router/pubmed_count_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:209-216           (neo4j_pubmed_count_adapter)
src/foundation/conf/conf.py:219-226           (neo4j_pubmed_query_adapter - companion)
```

Adapter (Cypher)

```
src/foundation/infra/db/adapter/neo4j_pubmed_count_adapter.py
:25-34  count_related_pubmed_by_drug
:36-47  count_related_pubmed_by_clinicaltrail
:49-58  count_related_pubmed_by_gene
:60-82  _count_related_pubmed_by_drug      (executor)
:83-92  _generate_pubmed_count_by_drug_id  (Cypher)
:94-115 _count_related_pubmed_by_clinicaltrail
:117-128 _generate_pubmed_count_by_clinicaltrail
:130-151 _count_related_pubmed_by_gene
:153-162 _generate_pubmed_count_by_gene
```

Response

```
(no Pydantic model, no mapper - plain dict returned by handlers)
src/foundation/router/pubmed_count_router.py:46,64,82
```

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      <-- the pubmed node
:45  PUBMED_DOCUMENT     = pubmed_document (defined but unused by this route)
src/foundation/model/foundational_relationship_enum.py
:28  FEATURED_IN         = featured_in   (drug <-> Research)
:29  ANALYZED_IN         = analyzed_in   (gene <-> Research)
```

Companion search route (different adapter, same driver pool)

```
src/foundation/router/pubmed_search_router.py
src/foundation/infra/db/adapter/neo4j_pubmed_query_adapter.py
src/foundation/mapper/pubmed_search_result_mapper.py
```

ETL (cross-reference)

```
src/download_pubmed.py      (stage 1: download PDFs to disk)
src/link_pubmed_articles.py (stage 2: LLM extract + Neo4j upsert)
graph/conf/conf.py          (eugene_graph_summary_orchestrator factory)
```

End of pubmed-count route flow guide.