

# Eugene Count Route — End-to-End Flow

Developer walkthrough: ETL-seeded Neo4j labels → FastAPI router → label validator → Cypher count (n) → JSON response

---

## Audience

Engineers who need to understand how the generic GET /count/{label} endpoint resolves a user-supplied label string into a Neo4j node label and returns the total node count. This is the simplest read endpoint in Eugene — one path parameter, one Cypher, one JSON object — which makes it a useful reference for the validator + adapter + router pattern used everywhere else. Every reference uses the form `path/to/file.py:line`.

## Reference endpoint

- GET /count/{label} — e.g. /count/drug, /count/disease, /count/gene\_protein.
- Response: { "count": <int>, "label": "<normalized>" }.

## Caveat

The route relies on whatever node labels the ETL pipelines have produced in Neo4j. There are no constraints or uniqueness rules — a label that exists in the enum but has never been ingested simply returns `count = 0`. The `Label Pydantic` enum (the validator) is intentionally narrower than the full `FoundationalNodeEnum`: only the foundational biomedical labels are exposed via this endpoint; counts for patents, pubmed documents, organizations, etc. live behind dedicated routers.

## 0. Schema (read this first)

The count endpoint touches a single node label and no relationships:

```
(:anatomy          { node_id, node_name, ... })
(:biological_process { node_id, node_name, ... })
(:disease          { node_id, node_name, ... })
(:drug             { node_id, node_name, drug_bank_id, ... })
(:effect_phenotype { node_id, node_name, ... })
(:exposure         { node_id, node_name, ... })
(:gene_protein     { node_id, node_name, ... })
(:molecular_function { node_id, node_name, ... })
(:pathway          { node_id, node_name, ... })
```

- Permitted labels are enumerated by `Label` in `src/foundation/router/model/label.py` — nine values: `anatomy`, `biological_process`, `disease`, `drug`, `effect_phenotype`, `exposure`, `gene_protein`, `molecular_function`, `pathway`.
- Each is mapped back to a `FoundationalNodeEnum` entry by `str_to_label_enum()` in `src/foundation/router/validate_util.py:107-111`. The enum tuple is `(int_id, lowercase_label)`; the second element is what gets interpolated into Cypher.
- No relationships are walked. The Cypher is `MATCH (n:`<label>`) RETURN count(n)`.
- Like the rest of Eugene there are no Neo4j `CREATE CONSTRAINTs` — the count is simply whatever currently exists with the requested label.

# 1. HTTP entry

`src/foundation/router/count_router.py`

- Router prefix `/count`, tag `foundation` (lines 18-21).
- Module-load DI at line 23: `foundational_node_count_adapter = neo4j_foundational_node_count_adapter()` — one driver-bound adapter per process, reused across requests.
- Single route: `GET /count/{label}` (line 26-28). `response_model=dict[str, int | str]` with `response_model_exclude_none=True`.
- Path param is `label: Label` (the Pydantic enum) with FastAPI `Path(min_length=1, max_length=64)` and an `AfterValidator(validate_label)` — so an unknown string is rejected as a 422 before it ever touches Neo4j.

## Verbatim handler (`count_router.py:26-42`)

```
@router.get(
   ("/{label}", response_model=dict[str, int | str], response_model_exclude_none=True
)
async def count_by_label(
    label: Annotated[
        Label,
        Path(
            description="The node type to count. e.g. drug, disease",
            max_length=64,
            min_length=1,
        ),
        AfterValidator(validate_label),
    ],
):
    label_value = label.value[1]
    count = foundational_node_count_adapter.count_all_by_label(label_value)
    return {"count": count, "label": label_value}
```

## Validator (`validate_util.py:13-21`)

```
def validate_label(label: Label) -> FoundationalNodeEnum:
    logger.info(f"validating label: {label}")
    is_valid = label in Label
    if not is_valid:
        valid_labels = [el.value for el in list(Label)]
        err_msg = f"invalid label {label}. Valid labels: {valid_labels}"
        raise ValueError(err_msg)
    node_type = str_to_label_enum(label.value)
    return node_type
```

Subtlety: `validate_label` returns a `FoundationalNodeEnum`, but the route signature is still typed as `Label`. Because the handler reads `label.value[1]` — the second tuple element — the code happens to work whether the runtime object is the narrow `Label` enum (value is a plain string, so `[1]` would actually grab the second character) or the broader `FoundationalNodeEnum` (value is a tuple). In practice FastAPI runs the `AfterValidator` after coercion, and the validator returns the enum — so `label.value[1]` is the lowercase label string. Worth knowing when stepping through.

## Registration

`src/eugene_ws.py:36, 61` — `add_routers()` imports the module as `foundation_count_router` and includes it via `app.include_router(router.router)`.

## 2. Query adapter (verbatim Cypher)

`src/foundation/infra/db/adapter/neo4j_foundational_node_count_adapter.py`

- Constructed in `src/foundation/conf/conf.py:99-106`:  
`neo4j_foundational_node_count_adapter()` returns a `Neo4jFoundationalNodeCountAdapter` built around the shared `_neo4j_driver()`.
- Public entry: `count_all_by_label(label: str) -> int` (line 22). Calls `ensure_connection(self.driver)` first — the standard Eugene pattern for transparently reconnecting a stale Neo4j driver before issuing a query.
- Builds the Cypher with `_build_count_all_by_label()` (line 40) and executes via `driver.execute_query(query=query, result_transformer=neo4j.Result.single)` — one record back, take the count column.
- Errors from the driver (`DriverError`, `Neo4jError`) are logged and re-raised; FastAPI renders them as 500s.

### Cypher (`neo4j_foundational_node_count_adapter.py:40-47`)

```
MATCH (n:`%s`)
RETURN count(n) as count
```

The `%s` is filled with the lowercase label string via Python %-formatting — **not** a parameterised query. That is safe here only because the value has already been constrained to the nine-entry `Label` enum by `validate_label`; any other call site that wanted to reuse this adapter would need to apply the same validation before calling. The commented-out `normalize_node_label(label)` in the source is a reminder that earlier iterations normalised the label here — responsibility now lives in the router-level validator.

### Execute path (lines 26-38)

```
def _count_all_by_label(self, label: str) -> int:
    query = self._build_count_all_by_label(label)
    logger.info(f"count all: {query}")
    try:
        record = self.driver.execute_query(
            query=query,
            result_transformer=neo4j.Result.single,
        )
        logger.info(f"cypher response: {record}")
        return record["count"]
    except (DriverError, Neo4jError) as exception:
        logging.error("%s raised an error: \n%s", query, exception)
        raise exception
```

### 3. Mapper / response model

Unlike the more elaborate routes (drug-alias, n-hop, facts), the count endpoint has no mapper class — the handler builds the JSON object inline. The response is typed at the FastAPI layer as `dict[str, int | str]`:

```
@router.get(
   ("/{label}", response_model=dict[str, int | str], response_model_exclude_none=True
)
```

#### Example response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "count": 4173,
  "label": "drug"
}
```

- `count` — integer; the result of `count (n)` in Neo4j. Zero is a perfectly valid answer (label exists in the enum but no nodes have been ingested with that label yet).
- `label` — the lowercase, normalized label string actually used in the Cypher. Echoing it back lets clients confirm what they queried (relevant because FastAPI is case-sensitive on the path param but the validator is case-fixed via the enum).
- Errors: a label outside the `Label` enum yields a `422 Unprocessable Entity` with the standard FastAPI validation envelope; a driver error yields a `500`.

#### Example error (422)

```
$ curl -sS http://localhost:8000/count/bogus | jq .
{
  "detail": [
    {
      "type": "enum",
      "loc": ["path", "label"],
      "msg": "Input should be 'anatomy', 'biological_process', ...",
      "input": "bogus"
    }
  ]
}
```

## 4. Data source / ETL

The count route is purely read-side — it has no ETL of its own. The numbers it returns are a side effect of every other ingest pipeline that writes nodes carrying one of the nine permitted labels.

### Where the labels come from

- `:drug` nodes — seeded by `bin/seed/csl_behring_assets.cypher`, `bin/seed/biogen_assets.cypher`, and the DrugBank/PrimeKG ingests under `src/ingest_*.py`. Drug aliases (`:drug_product`, `:drug_synonym`) are *not* counted by this route — only the canonical `:drug` label is in the `Label` enum.
- `:disease`, `:gene_protein`, `:pathway`, `:anatomy`, `:biological_process`, `:molecular_function`, `:effect_phenotype`, `:exposure` — loaded by the PrimeKG / Hetionet foundational ingest scripts; see the loaders under `src/foundation/load/`.
- Patents, pubmed documents, organizations, clinical trials, etc. have their own count routers (`patent_count_router.py`, `pubmed_count_router.py`) registered alongside this one in `eugene_ws.py:36-73`.

### Enum source of truth

```
src/foundation/model/foundational_node_enum.py
ANATOMY           = 1, 'anatomy'
BIOLOGICAL_PROCESS = 2, 'biological_process'
DISEASE           = 7, 'disease'
DRUG              = 8, 'drug'
EFFECT_PHENOTYPE  = 9, 'effect_phenotype'
EXPOSURE          = 10, 'exposure'
GENE_PROTEIN      = 12, 'gene_protein'
MOLECULAR_FUNCTION = 14, 'molecular_function'
PATHWAY           = 15, 'pathway'
```

`FoundationalRelationshipEnum`

(`src/foundation/model/foundational_relationship_enum.py`) is unused by this route — `count(n)` never traverses an edge.

## 5. Suggested debugging walk

- Bring the stack up: `docker-compose up eugene_neo4j eugene_ws`. Confirm the foundational seed has been applied (`cypher-shell < bin/seed/csl_behring_assets.cypher`) so at least some `:drug` nodes exist.
- Smoke test the happy path: `curl -sS http://localhost:8000/count/drug | jq ..`  
Expect `{"count": >0, "label": "drug"}`.
- Force a 422: `curl -sS http://localhost:8000/count/bogus`. Breakpoint at `count_router.py:29` — the `AfterValidator(validate_label)` fires first; you should never reach the handler body for an invalid label.
- Step into the adapter: breakpoint at `neo4j_foundational_node_count_adapter.py:40`. Inspect the assembled Cypher string — copy/paste it into Neo4j Browser to confirm the count matches.
- Watch the logs — the adapter emits `logger.info(f"count all: {query}")` and `logger.info(f"cypher response: {record}")`; `grep 'count all'` on the `eugene_ws` container is usually the fastest way to see what was actually run.
- Sanity-check the enum mapping: in a Python REPL, from `foundation.router.validate_util` import `str_to_label_enum`; `str_to_label_enum('gene_protein')` should return `FoundationalNodeEnum.GENE_PROTEIN`; passing a value not in any enum tuple raises `ValueError`.

## 6. Reference index

### Schema

```
src/foundation/router/model/label.py           # the 9-value Label enum
src/foundation/model/foundational_node_enum.py # full label registry
src/foundation/model/foundational_relationship_enum.py # unused by /count
```

### HTTP entry

```
src/foundation/router/count_router.py          # GET /count/{label}
src/foundation/router/validate_util.py         # validate_label (L13), str_to_label_enum (L107)
src/eugene_ws.py                              # add_routers() L36, include L61
```

### Adapter / Cypher

```
src/foundation/infra/db/adapter/neo4j_foundational_node_count_adapter.py
src/foundation/conf/conf.py                  # factory L99-106
src/graph/util/util.py                      # ensure_connection
```

### Related count endpoints

```
src/foundation/router/patent_count_router.py
src/foundation/router/pubmed_count_router.py
```

### Seed data feeding the counts

```
bin/seed/csl_behring_assets.cypher
bin/seed/biogen_assets.cypher
src/foundation/load/*.py
```