

Eugene Label Route — End-to-End Flow

Developer walkthrough: HTTP boundary → validators → DI → paginated Neo4j label scan → NameAndIdListResponse

Audience

Engineers onboarding to the Eugene biomedical knowledge graph who need a single, file-referenced trace of the `GET /labels/{label}` endpoint. This is the simplest read route in Eugene — it lists the `node_id / node_name` pairs of every node carrying a given Neo4j label, paginated. Every reference uses the form `path/to/file.py:line` so you can open it directly in your IDE and step through with a debugger.

Reference endpoint

`GET /labels/{label}?page=<n>&page_size=<m>` — `label` is one of nine string values exposed by the `Label` enum (Section 0). The response is a `NameAndIdListResponse`: a count and a list of `{name, id}` objects.

Sample call

```
curl -s 'http://localhost:8080/labels/drug?page=1&page_size=25' | jq .
```

Declared in `src/foundation/router/label_router.py:30-62`. Router prefix `/labels`, tag `foundation`.

How to read this guide

- Sections follow the request path top-down: `schema` → `HTTP` → `validator` → `DI factory` → `adapter` → `Cypher` → `mapper` → `JSON`.
- Section 0 maps the `Label` enum values onto the `FoundationalNodeEnum` tuples that hold the actual Neo4j label strings.
- Section 5 lists a concrete debugging walk (curl, breakpoints, Cypher verification).
- Note: the route is paginated via `page` and `page_size` query params; the adapter does `SKIP/LIMIT` arithmetic via the `Pagination` helper.

0. Schema (read this first)

Two enums collaborate here. `Label` is the public, FastAPI-facing surface — the set of label strings a client is allowed to send. `FoundationalNodeEnum` is the internal surface — a numeric id plus the actual Neo4j label string used in Cypher. The validator translates from one to the other.

Public Label enum

[src/foundation/router/model/label.py](#)

```
from enum import Enum
```

```
class Label(str, Enum):
    anatomy          = "anatomy"
    biological_process = "biological_process"
    disease           = "disease"
    drug              = "drug"
    effect_phenotype  = "effect_phenotype"
    exposure          = "exposure"
    gene_protein      = "gene_protein"
    molecular_function = "molecular_function"
    pathway           = "pathway"
```

Anything outside this nine-value set is rejected with HTTP 422 by FastAPI before the handler runs — the path parameter is typed `Label` and FastAPI enforces enum membership automatically.

Internal FoundationalNodeEnum

[src/foundation/model/foundational_node_enum.py](#)

```
class FoundationalNodeEnum(Enum):
    ANATOMY          = 1, "anatomy"
    BIOLOGICAL_PROCESS = 2, "biological_process"
    CELLULAR_COMPONENT = 3, "cellular_component"
    CLINICAL_TRIAL    = 4, "ClinicalTrial"
    COLLABORATOR      = 5, "collaborator"
    CONDITION         = 6, "condition"
    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"
    # ... and many more (PATENT, PHASE, SPONSOR, DRUG_PRODUCT, ...)
```

Label ↔ Neo4j label string mapping

- Each `FoundationalNodeEnum` member is a `(int_id, label_string)` tuple. The second slot is the Neo4j label that appears verbatim in Cypher.
- Most labels are lowercased Python names: `DRUG.value == (8, "drug")`, `GENE_PROTEIN.value == (12, "gene_protein")`.
- A few are CamelCase to match historical seed data — e.g. `CLINICAL_TRIAL.value == (4, "ClinicalTrial")`, `ORGANIZATION.value == (32, "Organization")`. These are **not** exposed via the public `Label` enum, so the label route cannot list them today.
- The `Label` enum is a deliberate subset — nine biomedical labels. Patent, Sponsor, Organization, Investigators, GraphRAG and PubMed labels are not listable through this route.

How the two enums meet

`validate_label` in `src/foundation/router/validate_util.py:13-21` converts the path-parameter `Label` into a `FoundationalNodeEnum` via `str_to_label_enum(label.value)` (`validate_util.py:107-111`):

```
def str_to_label_enum(value: str):
    for el in list(FoundationalNodeEnum):
        if el.value[1] == value:
            return el
    raise ValueError(f"'{value}' is not a valid FoundationalNode")
```

The handler then pulls `label.value[1]` out of the returned enum and hands the raw label string to the adapter. `normalize_node_label` (in `src/graph/util/node_util.py:4`) is applied inside the adapter before backtick-quoting in Cypher.

1. HTTP entry — the FastAPI router

Wiring

src/eugene_ws.py:35 — from foundation.router import label_router as foundation_label_router. src/eugene_ws.py:62 appends foundation_label_router to the routers list, and src/eugene_ws.py:77-78 loops through and calls app.include_router(router.router).

Router file

[src/foundation/router/label_router.py](#)

Imports (lines 1-17):

```
import logging
from typing import Annotated

from fastapi import APIRouter, Path, Query
from pydantic import AfterValidator

from foundation.conf.conf import (
    neo4j_foundational_node_adapter,
)
from foundation.router.model.name_id_list_response import NameAndIdListResponse
from foundation.router.response_util import (
    to_id_list_response,
)
from foundation.router.validate_util import (
    validate_label,
)
from foundation.router.model.label import Label
```

Router declaration (lines 22-25)

```
router = APIRouter(
    prefix="/labels",
    tags=["foundation"],
)
```

Module-load dependency injection (line 27)

```
foundational_node_adapter = neo4j_foundational_node_adapter()
```

Eugene's manual DI pattern: a module-scope call constructs the adapter once at import-time. The factory lives in src/foundation/conf/conf.py:109-114:

```
def neo4j_foundational_node_adapter() -> Neo4jFoundationalNodeAdapter:
    return _neo4j_foundational_node_adapter(driver=_neo4j_driver())

def _neo4j_foundational_node_adapter(
    driver: Driver,
) -> Neo4jFoundationalNodeAdapter:
    return Neo4jFoundationalNodeAdapter(driver=driver)
```

_neo4j_driver() (conf.py:95-96) returns a singleton neo4j.Driver via GraphDbConnectionFactory.remote_neo4j_instance_from_env() — configured from NEO4J_URI, NEO4J_USERNAME, NEO4J_PASSWORD. No framework, no decorators — module import order is the DI graph.

Endpoint handler (lines 30-62)

```
@router.get(
   ("/{label}", response_model=NameAndIdListResponse, response_model_exclude_none=True
```

```

)
async def list_by_label(
    label: Annotated[
        Label,
        Path(
            description="The node type to list. e.g. drug, disease",
            max_length=64,
            min_length=1,
        ),
        AfterValidator(validate_label),
    ],
    page: Annotated[
        int,
        Query(
            description="Page number. See the count endpoint to calculate max page number.",
            example=1,
            gt=0,
        ),
    ],
    page_size: Annotated[
        int,
        Query(
            description="Page size.",
            example=25,
            gt=0,
            le=50,
        ),
    ],
):
    dataframe = foundational_node_adapter.find_by_label(label.value[1], page, page_size)
    return to_id_list_response(dataframe)

```

Three validation layers

- **Path enum constraint** — `Label` (`src/foundation/router/model/label.py`) is a `str`, Enum with nine members. FastAPI rejects anything else with HTTP 422 before the handler runs.
- **Path length** — `min_length=1`, `max_length=64` on the path parameter (defensive; the enum is already shorter).
- **AfterValidator** — `validate_label` (`validate_util.py:13-21`) re-checks membership in `Label` and returns the corresponding `FoundationalNodeEnum` member. **The handler discards this return** — it then re-derives the label string from `label.value[1]` on the original `Label` instance (the `AfterValidator`'s return is never bound to anything).
- **Query bounds** — `page` must be `> 0`; `page_size` must be `> 0` and `<= 50`. FastAPI rejects out-of-range values with 422. **Pages are not server-capped** — a caller can request `page=10000` and get an empty result rather than an error.

Watch-out: redundant validator

Because `label: Annotated[Label, ...]` already constrains the input to a `Label` member, `validate_label` can never actually fail here — FastAPI has already coerced or rejected the value. The `AfterValidator`'s real value is its side-effect-free conversion to `FoundationalNodeEnum`, which (as noted) the handler ignores. Treat this as historical shape that other routes (`count`, `similarity`) still depend on.

Set your first breakpoint

At `label_router.py:61` (the `foundational_node_adapter.find_by_label(...)` call). Inspect:

- `label` — the raw `Label` enum instance.
- `label.value` — the underlying string (e.g. "drug").
- `page`, `page_size` — the pagination ints.

2. Query adapter — the Cypher

[src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py](#)

Public entry point (lines 40-44)

```
def find_by_label(
    self, label: str, page: int, page_size: int
) -> pd.DataFrame | None:
    ensure_connection(self.driver)
    return self._find_by_label(label, page, page_size)
```

`ensure_connection` from `src/graph/util/util.py` pings the driver before each request — defends against stale connections after Neo4j restarts.

Inner execution (lines 105-119)

```
def _find_by_label(
    self, label: str, page: int, page_size: int
) -> pd.DataFrame | None:
    query = self._build_find_by_label(label, page, page_size)
    logger.debug(f"find all: {query}")
    try:
        records = self.driver.execute_query(
            query=query,
            result_transformer=neo4j.Result.to_df,
        )
        logger.debug(f"cypher response: {records}")
        return records
    except (DriverError, Neo4jError) as exception:
        logging.error("%s raised an error: \n%s", query, exception)
        raise exception
```

- `Result.to_df` hands back a pandas DataFrame — two columns, `node_id` and `node_name`.
- **No parameter binding** — `label`, `offset`, and `limit` are all string-interpolated into the Cypher (see the `FIXME` comment in `neo4j_foundational_facet_adapter.py` for prior art on why this happens). For this route the risk is mitigated by the enum constraint on `label` and the integer types on `page` / `page_size` — nothing user-typed reaches the Cypher.

Cypher generator (lines 130-142)

```
def _build_find_by_label(self, label: str, page: int, page_size: int) -> str:
    limit, offset = Pagination.calculate_limit_and_offset(page, page_size)
    lookup_node_query = """MATCH (n:`%s`)
        RETURN n.node_id as node_id, n.node_name as node_name
        ORDER BY node_name
        SKIP %s
        LIMIT %s
    """ % (
        normalize_node_label(label),
        offset,
        limit,
    )
    return lookup_node_query
```

Pagination math

[src/foundation/infra/db/util/pagination.py](#)

```
class Pagination:
    @staticmethod
    def calculate_limit_and_offset(
        page: int = 1, page_size: int = 25
    ) -> Tuple[int, int]:
```

```

if page < 1 or page_size < 1:
    raise ValueError(...)
limit = page_size
offset = (page - 1) * page_size
return limit, offset

```

So page=2, page_size=25 → SKIP 25 LIMIT 25 (rows 26..50 in name order). The FastAPI handler has already enforced page > 0 and 0 < page_size <= 50, so the ValueError branch is unreachable from HTTP.

Generated Cypher for /labels/drug?page=1&page_size=25

```

MATCH (n:`drug`)
  RETURN n.node_id as node_id, n.node_name as node_name
  ORDER BY node_name
  SKIP 0
  LIMIT 25

```

Generated Cypher for /labels/gene_protein?page=3&page_size=10

```

MATCH (n:`gene_protein`)
  RETURN n.node_id as node_id, n.node_name as node_name
  ORDER BY node_name
  SKIP 20
  LIMIT 10

```

Step-by-step what the Cypher does

- **MATCH** — scans every node carrying the requested label. The label string is normalised via `normalize_node_label(label)` (`src/graph/util/node_util.py:4`) before backtick-quoting. Backticks are required because some labels (e.g. ClinicalTrial, off-label use) contain mixed case or punctuation.
- **RETURN** — only the two columns the mapper needs: `node_id` and `node_name`. There is no `is_hidden` filter here (unlike the drug-alias route) — hidden nodes appear in label listings.
- **ORDER BY node_name** — alphabetical so pagination is stable. Note: the order is by the `node_name` alias defined in RETURN, not the property directly — equivalent but worth knowing if you adapt the query.
- **SKIP / LIMIT** — interpolated ints, never bound params. Safe because the handler types them as `int`.
- **No index hint** — relies on Neo4j's label index for the MATCH and an in-memory sort for the ORDER BY. At >10k nodes per label this can get slow; consider a property index on `node_name` if you see latency.

3. Mapper & response model

The adapter returns a pandas DataFrame; `to_id_list_response` in `src/foundation/router/response_util.py:24-39` converts it into the typed `NameAndIdListResponse` that FastAPI then serialises.

Mapper function

`src/foundation/router/response_util.py`

```
def to_id_list_response(
    dataframe: pd.DataFrame | None,
) -> NameAndIdListResponse:
    if dataframe is None or len(dataframe) == 0:
        return NameAndIdListResponse(count=0, results=[])

    results = [el for el in dataframe.apply(_to_name_and_id, axis=1) if el is not None]
    return NameAndIdListResponse(count=len(results), results=results)

def _to_name_and_id(row: pd.Series) -> NameAndId | None:
    name_key = "node_name"
    id_key = "node_id"
    if row is None or name_key not in row or row[name_key] is None:
        return None
    return NameAndId(name=row[name_key], id=row[id_key])
```

- Empty / null DataFrame → `NameAndIdListResponse(count=0, results=[])` — the route never returns 404 for an empty label.
- Rows with a missing `node_name` are dropped silently. Rows with `node_name` present but `node_id` missing are **not** dropped — the resulting `NameAndId(id=None)` will fail Pydantic validation (`id` is typed `str`). In practice every node has both, but if you see a 500 from this route, check for orphan nodes missing `node_id`.
- `count` is the length of the returned page, **not** the total number of nodes with that label. Use `GET /count/{label}` (the count router) to get the grand total for pagination math.

Response models

`src/foundation/router/model/name_id_list_response.py`

```
from pydantic import BaseModel
from foundation.router.model.name_and_id import NameAndId
```

```
class NameAndIdListResponse(BaseModel):
    count: int
    results: list[NameAndId]
```

`src/foundation/router/model/name_and_id.py`

```
from pydantic import BaseModel

class NameAndId(BaseModel):
    name: str
    id: str
```

Example response — `GET /labels/drug?page=1&page_size=5`

```
{
  "count": 5,
  "results": [
    { "name": "abatacept",      "id": "DB01281" },
    { "name": "abciximab",     "id": "DB00054" },
    { "name": "acetaminophen", "id": "DB00316" },
    { "name": "acetylsalicylic acid", "id": "DB00945" },
```

```
    { "name": "adalimumab",      "id": "DB000051" }  
  ]  
}
```

Example response — GET /labels/exposure?page=99&page_size=25

```
{  
  "count": 0,  
  "results": []  
}
```

Empty page = past the end of the data, not an error. The HTTP status is still 200. The `response_model_exclude_none=True` flag on the decorator strips any None fields from individual NameAndId entries (defensive; both fields are required).

4. Data source — where do label data come from?

The label route is read-only. It never writes. The nodes it enumerates are seeded by a mixture of ETL jobs and Cypher seed files; the route just trusts that the labels are present in Neo4j.

ETL inputs (the biomedical core)

- `:drug`, `:disease`, `:gene_protein`, `:anatomy`, `:biological_process`, `:effect_phenotype`, `:exposure`, `:molecular_function`, `:pathway` — all originate from the PrimeKG-style TSV ingest under `src/foundation/load/`. Each loader writes one MERGE per row keyed by `node_id` (DrugBank id, MONDO id, etc.).
- Seed Cypher under `bin/seed/` (e.g. `cs1_behring_assets.cypher`) adds CSL-specific drug nodes on top of the PrimeKG core. Both paths use MERGE on `node_id` so re-runs are idempotent.
- Aliases (the `:drug_product` / `:drug_synonym` labels) are **not** exposed via this route — they live outside the Label enum. To list them you'd add an enum value and re-seed.

Property contract the mapper depends on

- Every node returned must carry both `node_id` and `node_name`. The seed scripts enforce this; if an ad-hoc import skips `node_id`, that row will surface as a 500 from this route (Pydantic validation failure on `NameAndId.id`).
- **No constraints, no uniqueness** — Eugene's Neo4j has no `CREATE CONSTRAINT` statements. De-duplication is achieved purely by MERGE on `node_id`. If two ETL paths use different `node_ids` for the same biological entity, you will see the same name twice with different ids in the response.
- The `is_hidden` property is honoured by some routes (drug-alias search, node-id lookup) but **not** by this one. Hidden nodes are listed.

How to confirm what's in your database

```
// Total nodes per label exposed by /labels:
CALL db.labels() YIELD label
WHERE label IN ['anatomy', 'biological_process', 'disease', 'drug',
               'effect_phenotype', 'exposure', 'gene_protein',
               'molecular_function', 'pathway']
CALL apoc.cypher.run('MATCH (n:`'+label+'`) RETURN count(n) AS n', {})
YIELD value
RETURN label, value.n AS count
ORDER BY count DESC;
```

If a label returns 0 here, every page of GET `/labels/` will be empty. That is a data problem, not a route bug.

5. Suggested debugging walk

- Bring the stack up: `docker-compose up eugene_neo4j eugene_ws`. Confirm `NEO4J_URI`, `NEO4J_USERNAME`, `NEO4J_PASSWORD` are set; `module-import DI at label_router.py:27` will fail loudly otherwise.
- Issue a request: `curl -s 'http://localhost:8080/labels/drug?page=1&page_size=5' | jq .` — expect a 200 with five rows. Now try `/labels/patent` — expect a 422 because `patent` is not in the `Label` enum.
- Set a breakpoint at `label_router.py:61` (`dataframe = foundational_node_adapter.find_by_label(...)`). Inspect `label.value` (the raw string) and confirm `page/page_size` came through as ints.
- Step into `neo4j_foundational_node_adapter.py:130` (`_build_find_by_label`). The fully interpolated Cypher is logged at `DEBUG` — set `LOG_LEVEL=DEBUG` to see it. Copy/paste into Neo4j Browser to verify the row count matches count in the response.
- Force the empty path: request `/labels/exposure?page=10000&page_size=50`. Confirm a 200 with `count=0` — the response is `NameAndIdListResponse(count=0, results=[])` from the mapper's null-DataFrame guard at `response_util.py:27-28`.
- Cross-check pagination: hit `/labels/drug?page=1&page_size=50` then `/labels/drug?page=2&page_size=50` and verify the first id of page 2 is alphabetically just after the last id of page 1. If they overlap or skip, look for non-deterministic `node_name` values (whitespace, casing).

6. Reference index (file paths)

Router & request models

src/foundation/router/label_router.py
src/foundation/router/model/label.py
src/foundation/router/validate_util.py (validate_label, str_to_label_enum)

Wiring

src/eugene_ws.py (lines 35, 62, 77-78)
src/foundation/conf/conf.py (lines 95-96, 109-114)

Adapter (Cypher generation + execution)

src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py
src/foundation/infra/db/util/pagination.py
src/graph/util/util.py (ensure_connection)
src/graph/util/node_util.py (normalize_node_label)

Mapper & response models

src/foundation/router/response_util.py (to_id_list_response)
src/foundation/router/model/name_id_list_response.py
src/foundation/router/model/name_and_id.py

Schema enums

src/foundation/model/foundational_node_enum.py (label tuples)

Related routes that share the validator / adapter

src/foundation/router/count_router.py (grand totals per label)
src/foundation/router/node_id_lookup_router.py
src/foundation/router/facet_router.py

Seed data (where the listed nodes come from)

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

Key takeaways

- GET /labels/{{label}} is the thinnest read path in Eugene: enum-validated path, paginated label scan, two-column DataFrame, typed list response.
- The Label enum is a deliberate nine-value subset of FoundationalNodeEnum. Patents, sponsors, organizations and clinical trials are **not** listable through this route.
- All Cypher is string-interpolated — safe here because the label is enum-bounded and pagination args are typed int. Don't copy this pattern into routes that accept free-form input.
- The route never returns 404 and never filters is_hidden. For pagination math, pair with GET /count/{{label}}; this route does not return the grand total.