

Eugene Node ID Lookup — End-to-End Flow

Developer walkthrough: HTTP GET → path validator → label-agnostic Cypher (exact or regex) → pandas DataFrame → NodeIdLookup response

Audience

Engineers onboarding to the Eugene biomedical knowledge graph who need a file-referenced trace of the node-id lookup. Every reference uses the form `path/to/file.py:line` so it can be opened directly in an IDE and stepped through with a debugger.

Reference endpoint

GET `/node/find/{node_value}` — declared at `src/foundation/router/node_id_lookup_router.py:22-47`.

Sample requests:

```
# exact match on node_name
curl 'http://localhost:18501/node/find/Flurandrenolide'

# case-insensitive regex (substring) match
curl 'http://localhost:18501/node/find/aspirin?fuzzy_match=true'
```

Sample response (200 OK):

```
{
  "results": [
    { "id": "DB01580", "value": "Flurandrenolide" }
  ],
  "count": 1,
  "query": "Flurandrenolide",
  "fuzzy_match": false
}
```

How to read this guide

- Sections follow the request path top-down: schema, HTTP entry, validation + DI, Cypher, mapping, response, ETL, debugging.
- Section 0 (Schema) is non-obvious: the Cypher does **not** filter on a label — the match is by the `node_name` property alone. This route is therefore label-agnostic by design.
- The `fuzzy_match=true` branch interpolates the value directly into the Cypher as a regex. `validate_value` is the only defense against Cypher injection on this route — section 1 covers the exact character blacklist.
- Section 4 points at the upstream ETL pipelines that populate `node_name`.
- Section 5 contains concrete curl + breakpoint lines + a Neo4j sanity query that mirrors the route's Cypher exactly.
- Section 6 is a flat reference index of every file in the request path.

0. Graph schema and conventions used by this route

The node-id lookup route, like node-details, is schema-light. It does not constrain by label, does not paginate, and returns whatever nodes in the graph carry the requested node_name (or match the regex), minus hidden nodes.

Required node properties

Each row of the response uses two Cypher projections; both assume the canonical Eugene property names:

- `n.node_name` — human-readable label, used both as the WHERE-clause match key and as value on the response.
- `n.node_id` — primary natural key, returned to the caller as `id` so it can be used as input to `/node/details`, `/n-hop`, or any other id-driven route.
- `n.is_hidden` — boolean flag; rows are excluded when this is explicitly `true` (see filter clause in §2). Nodes lacking the property are **included**.

Node labels in scope

Because the Cypher uses `MATCH (n ...)` with no label, every label declared in `src/foundation/model/foundational_node_enum.py` is reachable:

- `DRUG = 8, "drug", DISEASE = 7, "disease", GENE_PROTEIN = 12, "gene_protein", PATHWAY = 15, "pathway"`
- `CLINICAL_TRIAL = 4, "ClinicalTrial", PATENT_APPLICATION = 31, "Patent_Application", APPROVED_PATENT = 30, "Approved_Patent"`
- `DRUG_PRODUCT = 21, "drug_product", DRUG_SYNONYM = 22, "drug_synonym"` — useful because brand and synonym names that fail to match on `( :drug )` will frequently match on these.
- GraphRAG labels `SUMMARY / SUMMARY_FINDING (100, 101)` and PubMed `PUBMED_DOCUMENT / PUBMED_SUMMARY` are also addressable, since the query is label-agnostic.

Relationships

This route does not traverse edges. For completeness, edge types are declared in `src/foundation/model/foundational_relationship_enum.py` — e.g. `HAS_DRUG_ALIAS`, `INDICATION`, `DRUG_PROTEIN`, `DISEASE_PROTEIN`. Once the caller has the `node_id` from this route, downstream routes (`/n-hop`, `/path`) walk those edges.

Hard limits

- Maximum value length: **2048 characters**. Enforced by the FastAPI `Path(max_length=2048)` constraint (`node_id_lookup_router.py:29-34`).
- Forbidden characters in the path value: `%, _, $, ;, :, ^, *` — raises `ValueError` from `validate_util.py:35-42` (the `specials` list, line 10).
- Note: `_` is in the blacklist for this route — a `node_name` containing an underscore (rare but possible on some primeKG ids) cannot be looked up by exact match. Use `fuzzy_match` in that case.
- **No paging**. A fuzzy regex that matches thousands of nodes will return them all in one payload.

Implication for debugging: if the response is `count: 0` for a value you know exists in Neo4j, check (1) case — the exact path matches case-sensitively, (2) `is_hidden`, (3) the value was actually stored

on node_name (not node_id or a label-specific property).

1. HTTP entry — the FastAPI router

Wiring (eugene_ws.py)

src/eugene_ws.py:39-41 imports the router module as foundation_node_id_lookup_router. src/eugene_ws.py:55-78 registers it inside the routers list (entry at line 63), and a downstream loop calls app.include_router(router.router) on each.

Router file

src/foundation/router/node_id_lookup_router.py

Module-load dependency injection (line 19):

```
node_id_provider = foundational_node_id_provider()
```

This is Eugene's manual DI pattern — a plain factory call at module import time, no framework, no decorators, no FastAPI Depends. Because the provider is instantiated at import, the Neo4j driver inside it is also constructed eagerly. Cold-start latency on this route is therefore effectively zero after the first import.

DI factory chain

src/foundation/conf/conf.py:372-376 — the top of the chain:

```
def foundational_node_id_provider() -> FoundationalNodeIdProvider:
    return _foundational_node_id_provider(
        neo4j_foundational_node_adapter=neo4j_foundational_node_adapter(),
        node_id_lookup_mapper=_node_id_lookup_mapper(),
    )
```

- src/foundation/conf/conf.py:109-114 builds Neo4jFoundationalNodeAdapter(driver=_neo4j_driver()).
- src/foundation/conf/conf.py:229-230 builds the stateless NodeIdLookupMapper().
- src/foundation/conf/conf.py:379-386 composes the FoundationalNodeIdProvider.

Endpoint handler

src/foundation/router/node_id_lookup_router.py:22-47:

```
@router.get(
    "/find/{node_value}",
    response_model_exclude_none=True,
)
async def lookup_node_id_by_value(
    node_value: Annotated[
        str,
        Path(
            description="The node value to search",
            max_length=2048,
            min_length=0,
            example="Flurandrenolide",
        ),
        AfterValidator(validate_value),
    ],
    fuzzy_match: Annotated[
        bool,
        Query(
            title="True to do a fuzzy search, ... A fuzzy match is a regex match",
```

```

        example=False,
    ),
    ] = False,
):
    return node_id_provider.find_node_id_by_node_name(
        value=node_value, fuzzy_match=fuzzy_match
    )

```

- Route prefix `/node` is set on the APIRouter (`node_id_lookup_router.py:14-17`), so the full path is `GET /node/find/{node_value}`.
- The handler is declared `async` but the provider call is synchronous blocking I/O. Adequate for the current traffic profile.
- `response_model_exclude_none=True` drops `None` fields from the `NodeIdLookup` dataclass on the wire.
- FastAPI's `AfterValidator` runs the validator **after** Pydantic type coercion. A non-string value never reaches `validate_value`.

Validator

`src/foundation/router/validate_util.py:35-42:`

```

specials = ["%", "_", "$", ";", ":", "^", "*"]

def validate_value(value: str) -> str:
    if value is None:
        raise ValueError("value cannot be None")
    is_valid = not has_special_char(value)
    if not is_valid:
        raise ValueError(
            f"value cannot have a special character: {specials}"
        )
    return value

```

`has_special_char` (line 114) delegates to `has_invalid_char` (lines 118-128), a short-circuit linear scan. Validation errors surface to FastAPI as HTTP 422.

Set your first breakpoint at `node_id_lookup_router.py:45` (`return node_id_provider.find_node_id_by_node_name(...)`).

2. Provider + Query adapter — the Cypher

Provider (thin orchestration)

`src/foundation/provider/foundational_node_id_provider.py`

- `FoundationalNodeIdProvider.find_node_id_by_node_name` (lines 27-37) is decorated with `@log_time` and does exactly two things: (1) call the adapter to get a `DataFrame`, (2) hand the `DataFrame` to the mapper.
- No business logic lives here. This separation makes it trivial to swap the adapter for a fake in tests.

Adapter file

`src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py`

- `find_node_id_by_node_name` (lines 25-31) short-circuits on a `None` value, calls `ensure_connection(self.driver)` from `graph/util/util.py`, then delegates to `_find_node_id_by_node_name`.
- `_find_node_id_by_node_name` (lines 171-193) branches on `fuzzy_match`: exact path uses a parameterised query with `params = {"node_name": value}`; fuzzy path interpolates the value directly into the Cypher string and runs with `params = {}`.
- In both branches the result lands as a pandas `DataFrame` via `result_transformer_=neo4j.Result.to_df`.
- On `DriverError` or `Neo4jError` the adapter logs the query plus the exception and re-raises — FastAPI surfaces this as HTTP 500.

Generated Cypher — exact path (verbatim)

Built by `build_find_node_id_by_node_name`

(`neo4j_foundational_node_adapter.py:195-200`). Constant query, no label, no interpolation:

```
MATCH (n { node_name: $node_name })
WHERE n.is_hidden IS NULL OR NOT n.is_hidden
RETURN n.node_id, n.node_name
```

Generated Cypher — fuzzy path (verbatim)

Built by `build_fuzzy_find_node_id_by_node_name`

(`neo4j_foundational_node_adapter.py:202-209`). The supplied value is concatenated into the regex literal via Python %-formatting; the validator's character blacklist is the only guard:

```
MATCH (n)
WHERE n.node_name =~ '(?i).*<value>.*'
  AND (n.is_hidden IS NULL OR NOT n.is_hidden)
RETURN n.node_id, n.node_name
```

The `(?i)` prefix makes the regex case-insensitive and `.*...*` makes it a substring match. So `fuzzy_match=true` on `aspirin` matches `Aspirin`, `Acetylsalicylic acid (aspirin)`, etc.

Note the `IS NULL OR NOT` idiom on `is_hidden` — nodes without the property are **included**; only nodes with `is_hidden = true` are excluded.

DataFrame schema

The `DataFrame` returned by `execute_query(..., to_df)` has columns:

```
columns: ['n.node_id', 'n.node_name']
dtypes : object, object
```

```
shape : (<= match count, 2)
```

Note the column names retain the Cypher `n .` prefix — the adapter does not alias them in the RETURN clause. The mapper indexes into the DataFrame with the literal strings `row["n . node_id"]` and `row["n . node_name"]`.

3. Response mapping — DataFrame to dataclass

Mapper file

[src/foundation/mapper/node_id_lookup_mapper.py](#)

`NodeIdLookupMapper.map` (lines 19-29) is decorated with `@log_time`. A `None` `DataFrame` produces an empty-result `NodeIdLookup` that still echoes the query and `fuzzy_match` flag:

```
def map(self, value: str, fuzzy_match: bool, df: DataFrame | None)
    -> NodeIdLookup:
    if df is None:
        return NodeIdLookup(
            query=value, count=0, fuzzy_match=fuzzy_match, results=()
        )
    results = self._map_response_rows(df)
    return NodeIdLookup(
        results=results, count=len(results),
        query=value, fuzzy_match=fuzzy_match,
    )
```

Row-wise mapping (`node_id_lookup_mapper.py:31-42`):

```
def _map_response_rows(self, df: DataFrame) -> Tuple[IdAndValue, ...]:
    if df is None: return None
    return tuple(df.apply(self._map_response_row, axis=1))

def _map_response_row(self, row) -> IdAndValue | None:
    if row is None: return None
    return IdAndValue(
        id=row["n.node_id"],
        value=row["n.node_name"],
    )
```

- `df.apply(..., axis=1)` iterates row-wise. Each row becomes a frozen `IdAndValue` dataclass instance.
- `results` is a `Tuple[IdAndValue, ...]` (note: tuple, not list) — this matches the field type on `NodeIdLookup` and preserves dataclass hashability.
- FastAPI serializes the dataclass via Pydantic v2's `TypeAdapter` machinery. With `response_model_exclude_none=True` on the route, any `None` field is dropped from the JSON payload.

Response model

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

```
@dataclass(frozen=True, unsafe_hash=True)
class NodeIdLookup:
    results: Tuple[IdAndValue, ...]
    count: int
    query: str
    fuzzy_match: bool
```

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

```
@dataclass(frozen=True, unsafe_hash=True)
class IdAndValue:
    id: str
    value: str
```

The `frozen=True, unsafe_hash=True` pair lets the UI cache these objects in React Query's normalized cache by structural equality, and lets them drop into a Python set in tests without ceremony.

Wire shape

```
{
  "results": [
    { "id": "DB00945", "value": "Acetylsalicylic acid" },
    { "id": "DB00945-syn-1", "value": "Aspirin" }
  ],
  "count": 2,
  "query": "aspirin",
  "fuzzy_match": true
}
```

4. Data source — upstream ETL pipelines

This route is read-only and label-agnostic, so its data surface is every Eugene ingest pipeline that writes a `node_name`. Each pipeline writes nodes via `MERGE (n:`label` { node_id: $id }) ON CREATE SET n.node_name = ...`, which is the contract this Cypher reads back.

Drug ingest

- `src/ingest_drug_aliases.py` — CSV ingest that writes `(:drug)`, `(:drug_product)`, `(:drug_synonym)` nodes. The `node_name` on a drug is the DrugBank generic name; on a synonym it is the synonym string itself. See `docs/EUGENE_DRUG_ALIAS_FLOW_GUIDE.pdf` for the full trace.
- Because synonyms have their own nodes, a fuzzy lookup for a brand name typically returns the `(:drug_synonym)` entry first; the caller then follows `HAS_DRUG_ALIAS` to reach the canonical `(:drug)` node.

Patent ingest

- `src/ingest_uspto_patents.py` — pulls USPTO XML, writes `(:Approved_Patent)`, `(:Patent_Application)`, `(:USPTO_APPLICATION)`, `(:USPTO_PGPUB)` nodes. The `node_name` is the patent title (long, often punctuation-heavy).
- Fuzzy lookups on patent titles are common — the regex path is the intended use case there. Note the `_` character is in the validator blacklist, which can bite when looking up titles that contain underscores (rare for USPTO titles but possible in internal seed data).

Biomedical knowledge graph (primeKG)

- `src/ingest_primekg.py` — loads the primeKG TSVs into `(:disease)`, `(:gene_protein)`, `(:pathway)`, `(:anatomy)`, `(:biological_process)`, `(:cellular_component)`, `(:molecular_function)`, `(:effect_phenotype)`. `node_name` comes straight from the source TSV.
- Gene/protein names from primeKG are HGNC symbols (e.g. TP53, BRCA1) — exact matches work well for these.

Clinical trials

- `src/ingest_clinical_trials.py` — reads ClinicalTrials.gov export and writes `(:ClinicalTrial)`, `(:Sponsor)`, `(:Phase)`, `(:Investigators)`, `(:Intervention)`, `(:Condition)` nodes. `node_name` for a trial is the trial title; for a sponsor it is the organisation name; for a condition it is the condition string.

PubMed

- `src/ingest_pubmed.py` — writes `(:PUBMED_DOCUMENT)` and the GraphRAG-derived `(:PUBMED_SUMMARY)` / `(:PUBMED_SUMMARY_FINDING)` nodes. `node_name` on a document is the article title.

Corporate / seed data

- Seed cypher files in `bin/seed/` establish `(:Organization)`, `(:Research)`, `(:Collaborator)`, and the canonical `(:drug)` stubs the drug-alias pipeline later decorates.

Hidden-node sources

GraphRAG summary intermediates and ETL checkpoint markers carry `is_hidden = true` and are filtered out by the route's `WHERE` clause. If you add a new ETL that creates internal-only nodes, set `is_hidden = true` rather than picking a “private” label name.

5. Suggested debugging walk

- Start FastAPI locally: `uvicorn src.eugene_ws:app --port 18501`.
- Hit the endpoint with a known-good name: `curl localhost:18501/node/find/Flurandrenolide`. Then try the fuzzy form: `curl 'localhost:18501/node/find/aspirin?fuzzy_match=true'`.
- Breakpoint at `src/foundation/router/node_id_lookup_router.py:45`. Inspect `node_value` and `fuzzy_match` — the validator has already run by this point.
- Step into `FoundationalNodeIdProvider.find_node_id_by_node_name` (`foundational_node_id_provider.py:28`).
- Step into the adapter at `neo4j_foundational_node_adapter.py:171`; the variable `query` holds the Cypher exactly as printed in §2 and `params` is either `{"node_name": value}` (exact) or `{}` (fuzzy).
- After `execute_query` returns, inspect the DataFrame: `records.shape`, `records.columns` (note the `n.` prefix on column names), `records.iloc[0].to_dict()`.

Neo4j sanity query (mirrors the route)

Run these in the Neo4j browser at `http://localhost:7474` with the same values you sent to the route. The row set should match the HTTP response one-for-one:

```
// exact path
MATCH (n { node_name: 'Flurandrenolide' })
WHERE n.is_hidden IS NULL OR NOT n.is_hidden
RETURN n.node_id, n.node_name;

// fuzzy path (interpolation done client-side here)
MATCH (n)
WHERE n.node_name =~ '(?i).*aspirin.*'
  AND (n.is_hidden IS NULL OR NOT n.is_hidden)
RETURN n.node_id, n.node_name;
```

Common failure modes and what to check

- **HTTP 422** — the validator rejected the value. Look at the FastAPI error body for `value cannot have a special character` or `String should have at most 2048 characters`.
- **count: 0** for a value you expect to exist: (a) the node has `is_hidden = true`; (b) wrong case — the exact path matches case-sensitively, try `fuzzy_match=true`; (c) the value lives on a different property (e.g. `synonym` on a `(:drug)` node rather than on a `(:drug_synonym)` node's `node_name`).
- **Unexpectedly huge result on fuzzy** — the regex `. *x.*` matches across all 22+ label types. Narrow with a longer value, or follow up with the label-aware `/node/details` or a label-scoped route.
- **HTTP 500** with a `Neo4jError` in the logs — almost always a malformed regex (unbalanced bracket, dangling quantifier). Check the query string at `neo4j_foundational_node_adapter.py:180`.

6. Reference index (file:line)

Schema

src/foundation/model/foundational_node_enum.py
src/foundation/model/foundational_relationship_enum.py

HTTP entry and wiring

src/eugene_ws.py:39-41 (import)
src/eugene_ws.py:55-78 (routers list + include_router)
src/foundation/router/node_id_lookup_router.py:14-17 (APIRouter prefix)
src/foundation/router/node_id_lookup_router.py:19 (DI hook)
src/foundation/router/node_id_lookup_router.py:22-47 (handler)

Validation

src/foundation/router/validate_util.py:10 (specials list)
src/foundation/router/validate_util.py:35-42 (validate_value)
src/foundation/router/validate_util.py:114-128 (has_special_char / has_invalid_char)

DI factories

src/foundation/conf/conf.py:109-114 (Neo4j adapter factory)
src/foundation/conf/conf.py:229-230 (mapper factory)
src/foundation/conf/conf.py:372-376 (provider factory entry)
src/foundation/conf/conf.py:379-386 (provider composition)

Provider / orchestration

src/foundation/provider/foundational_node_id_provider.py:14-37

Adapter / Cypher

src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py:16-23
class header
src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py:25-31
public endpoint (find_node_id_by_node_name)
src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py:171-193
_find_node_id_by_node_name (exact/fuzzy branch + execute_query)
src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py:195-200
build_find_node_id_by_node_name (exact Cypher)
src/foundation/infra/db/adapter/neo4j_foundational_node_adapter.py:202-209
build_fuzzy_find_node_id_by_node_name (regex Cypher)
src/graph/util/util.py (ensure_connection)

Mapper / response

src/foundation/mapper/node_id_lookup_mapper.py:14-29 (NodeIdLookupMapper.map)
src/foundation/mapper/node_id_lookup_mapper.py:31-34 (_map_response_rows)
src/foundation/mapper/node_id_lookup_mapper.py:36-42 (_map_response_row)
src/foundation/mapper/node_id_lookup_mapper_test.py (unit tests)

Model

src/foundation/model/node_id_lookup.py (NodeIdLookup)
src/foundation/model/id_and_value.py (IdAndValue)

Upstream ETL (sources of node_name)

src/ingest_drug_aliases.py
src/ingest_uspto_patents.py
src/ingest_primekg.py
src/ingest_clinical_trials.py
src/ingest_pubmed.py
bin/seed/csl_behring_assets.cypher
bin/seed/biogen_assets.cypher
bin/seed/csl_drug_node_properties.cypher

Related routes (consume node_id returned by this route)

src/foundation/router/node_details_router.py (POST /node/details)
src/foundation/router/n_hop_router.py (n-hop traversal)
src/foundation/router/search_path_router.py (shortest path)