

Eugene Node Details — End-to-End Flow

Developer walkthrough: HTTP POST → Pydantic validation → label-agnostic Cypher → pandas DataFrame → GenericNodeDetails response

Audience

Engineers onboarding to the Eugene biomedical knowledge graph who need a file-referenced trace of the node-details 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

POST `/node/details` — declared at `src/foundation/router/node_details_router.py:25-36`.

Sample request:

```
curl -X POST 'http://localhost:18501/node/details' \
  -H 'Content-Type: application/json' \
  -d '{"ids": ["DB00945", "C031180", "DB00436"]}'
```

Sample response (200 OK):

```
[
  {
    "labels": ["drug"],
    "id": "DB00945",
    "value": "Acetylsalicylic acid",
    "source": "DrugBank",
    "description": "..."
  },
  ...
]
```

How to read this guide

- Sections follow the request path top-down: HTTP entry, validation, DI factory chain, Cypher, DataFrame, mapping, response.
- Section 0 (Schema) is non-obvious: the Cypher does **not** filter on any label and relies on three conventional properties — `node_id`, `node_name`, `node_source` — that every canonical Eugene node carries.
- Section 4 points at the upstream ETL pipelines that populate the properties this route reads back.
- 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-details route is the most schema-light route in the codebase. It does not constrain by label and it does not paginate. It returns whatever nodes in the graph match `n.node_id IN $node_ids` and are not hidden.

Required node properties

Each row of the response is built from five Cypher projections, all of which assume the canonical Eugene property names. If a node is missing any of these, the corresponding field on `GenericNodeDetails` will be `None`:

- `n.node_id` — primary natural key; this is the value clients pass in the request body.
- `n.node_name` — human-readable label, mapped to `value` in the response.
- `n.node_source` — ontology of record (e.g. DrugBank, MeSH, ClinVar).
- `n.description` — long-form text used by the UI side panel.
- `n.is_hidden` — boolean flag; rows are excluded when this is explicitly `true` (see filter clause in §2).

Node labels in scope

The Cypher returns `labels(n)` so the caller can branch on the node type. Labels are defined in `src/foundation/model/foundational_node_enum.py`:

- `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"`
- `ORGANIZATION = 32, "Organization", RESEARCH = 33, "Research", INVESTIGATORS = 34, "Investigators"`
- GraphRAG-derived labels `SUMMARY / SUMMARY_FINDING (100, 101)` and `USPTO labels (200, 201)` are also addressable by id, since the query is label-agnostic.

Hard limits

- Maximum batch size: **50 ids**. Enforced twice — in the validator (`validate_util.py:88-92`) and again in the adapter (`Neo4jFoundationalNodeDetailsAdapter.MAX_QUERY_IDS = 50`). The router also de-duplicates with `set()` before calling the provider.
- Forbidden characters in ids: `%, $, ;, :, ^, *` — raises `ValueError` from `validate_util.py:98-104`.

Implication for debugging: if a client gets an empty array back for an id it knows exists in Neo4j, check `is_hidden` first — it is the single most common cause of unexpected omissions.

1. HTTP entry — the FastAPI router

Wiring (eugene_ws.py)

src/eugene_ws.py:42-44 imports the router module as `foundational_node_details_router`. src/eugene_ws.py:55-78 registers it inside `add_routers(app)` which iterates a flat list and calls `app.include_router(router.router)`.

Router file

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

Module-load dependency injection (line 22):

```
node_details_provider = foundational_node_details_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:389-393 — the top of the chain:

```
def foundational_node_details_provider() -> FoundationalNodeDetailsProvider:
    return _foundational_node_details_provider(
        neo4j_foundational_node_details_adapter=neo4j_foundational_node_details_adapter(),
        node_details_mapper=_node_details_mapper(),
    )
```

- src/foundation/conf/conf.py:117-124 builds `Neo4jFoundationalNodeDetailsAdapter(driver=_neo4j_driver())`.
- src/foundation/conf/conf.py:233-234 builds the stateless `NodeDetailsMapper()`.
- src/foundation/conf/conf.py:396-403 composes the `FoundationalNodeDetailsProvider`.

Endpoint handler

src/foundation/router/node_details_router.py:25-36:

```
@router.post(
    "/details",
    response_model_exclude_none=True,
)
async def lookup_node_details_by_id(
    node_details_request: Annotated[
        NodeDetailsRequest,
        AfterValidator(validate_node_details_request),
    ],
):
    ids = set(node_details_request.ids)
    return node_details_provider.find_node_details_by_node_ids(ids)
```

- Route prefix `/node` is set on the `APIRouter` (`node_details_router.py:17-20`), so the full path is `POST /node/details`.
- The handler is declared `async` but the provider call is synchronous blocking I/O. This is acceptable for the current traffic profile, but a future change to a thread executor would be a single-line move.

- `response_model_exclude_none=True` drops `None` fields from each `GenericNodeDetails` dict on the wire, keeping the payload small.

Validators

`src/foundation/router/validate_util.py:80-95:`

```
def validate_node_details_request(
    request: NodeDetailsRequest,
) -> NodeDetailsRequest:
    if request is None: return request
    ids = request.ids
    if ids is None: return request
    num_ids = len(ids)
    max = 50
    if num_ids > max:
        raise ValueError(
            f"Too many ids given in request: ... max: {max}"
        )
    for id in ids:
        validate_id(id)
    return request
```

`validate_id` (lines 98-104) rejects ids containing `%$;:^*`. Both validation errors surface to FastAPI as HTTP 422.

Request model

`src/foundation/router/model/node_details_request.py:`

```
class NodeDetailsRequest(BaseModel):
    ids: list[Annotated[
        str,
        Field(None, examples=[
            "C031180", "DB00846", "DB00436"
        ]),
    ]]
]]
```

Set your first breakpoint at `node_details_router.py:35` (`ids = set(node_details_request.ids)`).

2. Provider + Query adapter — the Cypher

Provider (thin orchestration)

`src/foundation/provider/foundational_node_details_provider.py`

- `FoundationalNodeDetailsProvider.find_node_details_by_node_ids` (lines 29-36) 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_details_adapter.py`

- `find_node_details_by_node_ids` (lines 23-30) short-circuits on empty/None input, calls `_ensure_query_limit_or_raise` (the second 50-id guard), runs `ensure_connection(self.driver)` from `graph/util/util.py`, then delegates to `_find_node_details_by_node_ids`.
- `_find_node_details_by_node_ids` (lines 32-51) calls `self.driver.execute_query(...)` with `result_transformer=neo4j.Result.to_df` — the result lands directly as a pandas `DataFrame`, no manual record iteration.
- On `DriverError` or `Neo4jError` the adapter logs the query plus the exception and re-raises — FastAPI surfaces this as HTTP 500.
- `_ensure_query_limit_or_raise` (lines 61-69) raises `ValueError` if the set has more than 50 ids. The router-level validator should make this branch unreachable, but it is defense in depth for any future caller that bypasses the validator.

Generated Cypher (verbatim)

Built by `_build_find_node_details_by_node_ids` (`neo4j_foundational_node_details_adapter.py:53-59`). There is no label or relationship interpolation — the string is a constant:

```
MATCH (n)
WHERE (n.is_hidden IS NULL OR NOT n.is_hidden)
  AND n.node_id IN $node_ids
RETURN labels(n)      AS labels,
       n.node_id      AS id,
       n.node_name     AS value,
       n.node_source   AS source,
       n.description   AS description
```

Note the `IS NULL OR NOT` idiom on `is_hidden` — this means nodes without the property are **included**, only nodes with `is_hidden = true` are excluded. The property was added late in the graph's life and not back-filled, hence the null-tolerant check.

Pagination

None. Unlike `patent_search_router` or `pubmed_search_router`, this route has no `page/page_size` parameters and no `SKIP/LIMIT` clauses. The 50-id cap is the only bound on result size. A request for 50 ids that all match exactly one node each will return at most ~50 rows (more if a node carries the same `node_id` under multiple labels, which Eugene's `MERGE` convention prevents).

DataFrame schema

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

```
columns: ['labels', 'id', 'value', 'source', 'description']
dtypes : object (labels: list[str]), object, object, object, object
shape   : (<=50, 5)
```

The `labels` column is a Python list of strings (Neo4j returns `LabelSet`, the driver coerces to `list`).

The mapper widens it to a `Tuple[str, ...]` on the dataclass.

3. Response mapping — DataFrame to dataclass

Mapper file

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

`NodeDetailsMapper.map` (lines 17-22) is decorated with `@log_time` and short-circuits on a `None` `DataFrame` (this is the empty-ids signal the adapter sends back).

```
def map(self, df: DataFrame | None) -> list[GenericNodeDetails] | None:
    if df is None:
        return None
    return self._map_response_rows(df)
```

Row-wise mapping (`node_details_mapper.py:24-42`):

```
def _map_response_rows(self, df: DataFrame) -> list[GenericNodeDetails]:
    if df is None: return None
    results = [
        el for el in df.apply(self._map_response_row, axis=1)
        if el is not None
    ]
    return results
```

```
def _map_response_row(self, row) -> GenericNodeDetails | None:
    if row is None: return None
    return GenericNodeDetails(
        labels=row["labels"],
        id=row["id"],
        value=row["value"],
        source=row["source"],
        description=row["description"],
    )
```

- `df.apply(..., axis=1)` iterates row-wise. Each row is converted to a frozen, hashable `GenericNodeDetails` dataclass instance.
- There is no field-level cleansing — whatever Neo4j returns is passed through. Empty descriptions and missing sources surface as Python `None`.
- FastAPI serializes the dataclass instances via Pydantic v2's `TypeAdapter` machinery. Because the response is a plain `list[GenericNodeDetails]` (no declared `response_model`), the dataclass field names become the JSON keys.

Response model

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

```
@dataclass(frozen=True, unsafe_hash=True)
class GenericNodeDetails:
    labels: Tuple[str, ...]
    id: str
    value: str
    source: str
    description: 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 be dropped into a Python set in tests without ceremony.

4. Data source — upstream ETL pipelines

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

Drug ingest

- `src/ingest_drug_aliases.py` — CSV ingest that writes `(:drug)`, `(:drug_product)`, `(:drug_synonym)` nodes. Sets `node_source = "DrugBank"`. See `docs/EUGENE_DRUG_ALIAS_FLOW_GUIDE.pdf` for the full trace.
- DrugBank canonical seed is loaded via `bin/seed/csl_behring_assets.cypher`, `bin/seed/biogen_assets.cypher`, and `bin/seed/csl_drug_node_properties.cypher` — these establish the `node_id` primary keys for the alias attach.

Patent ingest

- `src/ingest_uspto_patents.py` — pulls USPTO XML, writes `(:Approved_Patent)`, `(:Patent_Application)`, `(:USPTO_APPLICATION)`, `(:USPTO_PGPUB)` nodes with `node_source = "USPTO"`. See `docs/EUGENE_PATENT_FLOW_GUIDE.pdf`.
- The `node_id` format for patents is the publication number as a string (e.g. `US-2021-0123456-A1`); the inspector panel calls this route directly when the user clicks a patent node on the context graph.

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)`. Source tag: `node_source = "primekg"`.
- description on primeKG nodes is often empty — this is why the inspector panel renders a placeholder for many disease / gene nodes.

Clinical trials

- `src/ingest_clinical_trials.py` — reads ClinicalTrials.gov export and writes `(:ClinicalTrial)`, `(:Sponsor)`, `(:Phase)`, `(:Investigators)`, `(:Intervention)`, `(:Condition)`, `(:PrimaryOutcomeMeasure)`, `(:SecondaryOutcomeMeasure)` nodes. The `node_id` is the NCT id (validated elsewhere via `validate_clinical_trial_id`).

PubMed

- `src/ingest_pubmed.py` — writes `(:PUBMED_DOCUMENT)` and the GraphRAG-derived `(:PUBMED_SUMMARY)` / `(:PUBMED_SUMMARY_FINDING)` nodes. `node_source = "pubmed"`. PMIDs become `node_id`.

Corporate / seed data

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

Hidden-node sources

A few pipelines mark intermediate scaffolding nodes with `is_hidden = true` (GraphRAG summary intermediates, ETL checkpoint markers). These are filtered out by the route's `WHERE` clause and are not visible to clients. If you add a new ETL that creates internal-only nodes, set `is_hidden = true` on them 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 drug id: `curl -X POST localhost:18501/node/details -H "Content-Type: application/json" -d '{"ids": ["DB00945"]}'`.
- Breakpoint at `src/foundation/router/node_details_router.py:35` (`ids = set(...)`). Inspect the request object — the validator has already run by this point.
- Step into `FoundationalNodeDetailsProvider.find_node_details_by_node_ids` (`foundational_node_details_provider.py:30`).
- Step into the adapter at `neo4j_foundational_node_details_adapter.py:36`; the variable `query` holds the Cypher exactly as printed in §2 and `params` is `{"node_ids": [...]}`.
- After `execute_query` returns, inspect the DataFrame: `records.shape`, `records.columns`, `records.iloc[0].to_dict()`.
- Set a final breakpoint inside `NodeDetailsMapper._map_response_row` (`node_details_mapper.py:33`) to confirm field coercion.

Neo4j sanity query (mirrors the route)

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

```
MATCH (n)
WHERE (n.is_hidden IS NULL OR NOT n.is_hidden)
  AND n.node_id IN ['DB00945', 'C031180', 'DB00436']
RETURN labels(n) AS labels,
       n.node_id AS id,
       n.node_name AS value,
       n.node_source AS source,
       n.description AS description
```

Common failure modes and what to check

- **HTTP 422** — the validator rejected an id. Look at the FastAPI error body for Too many ids (>50) or id cannot have a special character.
- **Empty array** for an id you expect to exist: (a) the node has `is_hidden = true`; (b) the id was supplied with the wrong casing — the Cypher matches on `n.node_id` exactly; (c) the node was MERGED under a different natural-key property and lacks a `node_id` entirely.
- **HTTP 500** with a `Neo4jError` in the logs — check the driver health by running any trivial Cypher; `ensure_connection` (adapter line 27) will have logged the failure.
- **Field is null on the wire** — the underlying node lacks the property. Useful for backfill scripts:

```
MATCH (n) WHERE n.node_id = "X" RETURN properties(n).
```

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:42-44 (import)
src/eugene_ws.py:55-78 (include_router)
src/foundation/router/node_details_router.py:17-20 (APIRouter prefix)
src/foundation/router/node_details_router.py:22 (DI hook)
src/foundation/router/node_details_router.py:25-36 (handler)

Validation

src/foundation/router/validate_util.py:80-95 (validate_node_details_request)
src/foundation/router/validate_util.py:98-104 (validate_id)
src/foundation/router/model/node_details_request.py (Pydantic model)

DI factories

src/foundation/conf/conf.py:117-124 (Neo4j adapter factory)
src/foundation/conf/conf.py:233-234 (mapper factory)
src/foundation/conf/conf.py:389-393 (provider factory entry)
src/foundation/conf/conf.py:396-403 (provider composition)

Provider / orchestration

src/foundation/provider/foundational_node_details_provider.py:14-36

Adapter / Cypher

src/foundation/infra/db/adapter/neo4j_foundational_node_details_adapter.py:13-21
class header + MAX_QUERY_IDS = 50
src/foundation/infra/db/adapter/neo4j_foundational_node_details_adapter.py:23-30
public endpoint, guard + ensure_connection
src/foundation/infra/db/adapter/neo4j_foundational_node_details_adapter.py:32-51
driver.execute_query(..., result_transformer=neo4j.Result.to_df)
src/foundation/infra/db/adapter/neo4j_foundational_node_details_adapter.py:53-59
Cypher builder (constant query string)
src/foundation/infra/db/adapter/neo4j_foundational_node_details_adapter.py:61-69
_ensure_query_limit_or_raise
src/graph/util/util.py (ensure_connection)

Mapper / response

src/foundation/mapper/node_details_mapper.py:12-22 (NodeDetailsMapper.map)
src/foundation/mapper/node_details_mapper.py:24-31 (_map_response_rows)
src/foundation/mapper/node_details_mapper.py:33-42 (_map_response_row)
src/foundation/mapper/node_details_mapper_test.py (unit tests)

Model

src/foundation/model/generic_node_details.py (GenericNodeDetails)
src/foundation/router/model/node_details_request.py (NodeDetailsRequest)

Upstream ETL (sources of node_id / node_name / node_source / description)

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

MCP cross-reference (callers of this route)

agents/eugene-mcp/src/tools/eugene_node_tools.py
agents/eugene-mcp/src/tools/eugene_fetch_tools.py