

Eugene Similarity Search — End-to-End Flow

Developer walkthrough: HTTP entry → DI wiring → Neo4j GDS nodeSimilarity Cypher → in-memory projection → embedding pipeline → Pydantic response.

Audience

Engineers who need to understand how Eugene answers “what is similar to Lupus?” by running Neo4j Graph Data Science (GDS) `nodeSimilarity.filtered` over a pre-projected in-memory graph, and how the `POST /similarity/{label}` endpoint stitches the request together at query time. Every reference uses the form `path/to/file.py:line` so you can open files in your IDE and step through with a debugger.

Reference endpoints

- `POST /similarity/drug` body `{"values": ["ibuprofen"]}` — rank canonical drugs by neighbourhood overlap with the matched anchor drug(s).
- `POST /similarity/disease` body `{"values": ["lupus"]}` — same shape, but anchored on disease nodes.
- Only those two labels are accepted — see `FacetLabel` and the adapter guard described in Section 2.

How to read this guide

- Section 0 explains the schema and where the “similarity” score actually comes from. It is **not** a vector search over stored embeddings; it is graph topology (GDS `nodeSimilarity` in COSINE mode over a relationship-set vector). The embedding pipeline still exists (Section 4) because FastRP training consumes it, but the live endpoint does not read it.
- Sections 1–3 trace the live request: HTTP → adapter Cypher → the projection it depends on.
- Sections 4–5 cover the offline pipeline: embedding upserts and the optional FastRP training step.
- Section 6 lists the response/Pydantic shape with an example JSON.
- Section 7 is a step-through debugging walk (curl + breakpoints).
- Section 8 is the file-path reference index.

0. Schema (read this first)

The similarity router operates on the canonical foundational graph — the same nodes the rest of Eugene reads. There is no separate similarity table or vector store.

```
(:drug { node_id, node_index, node_name, embeddings?, ... })
(:disease { node_id, node_index, node_name, embeddings?, ... })
// plus all the foundational relationship types
// (indication, contraindication, disease_protein,
// pathway_pathway, off_label_use, exposure_disease, ...)
// connecting them to other foundational labels.
```

- **Label whitelist.** Only DRUG and DISEASE are valid path parameters. Enforced at the type boundary by `FacetLabel` (`src/foundation/router/model/facet_label.py`) and *again* defensively inside the adapter at `neo4j_foundational_similarity_adapter.py:58-62` which raises `ValueError` for anything else.
- **node_id vs node_index.** `node_id` is the public identifier returned to clients (stringified by `toStringOrNull` in the Cypher). `node_index` is the upsert key used by the embedding writer (Section 4).
- **embeddings property.** Optional. Written by `Neo4jFoundationalNodeAdapter.upsert_embeddings` when the ingest pipeline runs (Section 4). The similarity router itself **does not read this property** — it is consumed only by the FastRP training step in Section 5.
- **No Neo4j CONSTRAINTs.** Like every other foundational route in Eugene, idempotency is provided by MERGE on `node_id` / `node_index`, not by a CREATE CONSTRAINT. Do not assume uniqueness is enforced at the database level.
- **Projection.** The GDS in-memory graph is named `eugene_similarity_graph` (`src/foundation/conf/const.py:2`). It must exist before the endpoint can return rows — see Section 3.

1. HTTP entry — the FastAPI router

`src/foundation/router/similarity_router.py`

Module-load dependency injection

Line ~28 of `similarity_router.py` builds the adapter at *import time*:

```
foundational_similarity_adapter = neo4j_foundational_similarity_adapter()
```

The factory lives at `src/foundation/conf/conf.py:177-184`:

```
def neo4j_foundational_similarity_adapter()
    -> Neo4jFoundationalSimilarityAdapter:
    return _neo4j_foundational_similarity_adapter(
        driver=_neo4j_driver())

def _neo4j_foundational_similarity_adapter(driver: Driver)
    -> Neo4jFoundationalSimilarityAdapter:
    return Neo4jFoundationalSimilarityAdapter(driver=driver)
```

This is Eugene's manual DI pattern: one adapter instance per process, sharing a single `neo4j.Driver`. No FastAPI Depends, no per-request construction. The driver pool is what gives the endpoint its throughput.

Handler signature (lines 31-58)

```
@router.post("/{label}", response_model_exclude_none=True)
async def calculate_similarity_by_label_and_values(
    label: Annotated[
        FacetLabel,
        Path(description="The node type to list. e.g. disease, drug",
              max_length=64, min_length=0,
              examples=["disease, drug"]),
        AfterValidator(validate_label),
    ],
    similarity_search_values: SimilaritySearchValues,
):
    label_enum = str_to_label_enum(label.value[1])
    values = (similarity_search_values.values
              if similarity_search_values.values is not None
              else [])
    dataframe = foundational_similarity_adapter\
        .calculate_similar_by_label_and_values(
            label=label_enum, values=values)
    return to_similarity_response(dataframe)
```

Path and body validators

- `FacetLabel` (`src/foundation/router/model/facet_label.py`) is the enum FastAPI uses to coerce the path string before the handler runs. Any value outside its members returns a 422.
- `validate_label` (`src/foundation/router/validate_util.py:13-21`) is an `AfterValidator` that double-checks the value is a member of `Label` and maps it to a `FoundationalNodeEnum`. Belt-and-braces with the type system above.
- `SimilaritySearchValues` (`src/foundation/router/model/similarity_search_values.py`) is a Pydantic model with a single field `values: list[str]`. There is **no** `validate_facet_search_values` on the body in this router — that omission matters; see the FIXME in Section 2.
- `str_to_label_enum` (`src/foundation/router/validate_util.py`) maps the validated string into the `FoundationalNodeEnum` the adapter expects.

Set your first breakpoint

At `similarity_router.py:48` (the `label_enum = ...` line) you can inspect `label.value` and `similarity_search_values.values` before the adapter call.

2. Query adapter — the GDS nodeSimilarity Cypher

`src/foundation/infra/db/adapter/neo4j_foundational_similarity_adapter.py`

Public path

- `calculate_similar_by_label_and_values()` (line ~25) calls `ensure_connection(self.driver)` (`src/graph/util/util.py`) for a per-request driver health check, then delegates to `_calculate_similar_by_label_and_values()` with the projection name read from `SIMILARITY_GRAPH_PROJECTION_NAME`.
- `_calculate_similar_by_label_and_values()` (line ~33) builds the Cypher string and runs `self.driver.execute_query(query=..., result_transformer=neo4j.Result.to_df)`, returning a pandas DataFrame.
- `DriverError` and `Neo4jError` are logged and re-raised. The router has no `try/except`, so they bubble up as a FastAPI 500.

Generated Cypher (after interpolation, for `values=["lupus"]` on the disease label):

```
MATCH (d1:`disease`)
WHERE d1.node_name =~ '(?i).*lupus.*'

CALL gds.nodeSimilarity.filtered.stream('eugene_similarity_graph', {
  degreeCutoff: 1,
  similarityCutoff: .45,
  similarityMetric: "COSINE",
  sourceNodeFilter: [d1]
})
YIELD node1, node2, similarity
RETURN similarity,
  gds.util.asNode(node1).node_name AS name1,
  gds.util.asNode(node2).node_name AS name2,
  toStringOrNull(gds.util.asNode(node1).node_id) AS id1,
  toStringOrNull(gds.util.asNode(node2).node_id) AS id2
ORDER BY similarity DESCENDING, name1, name2
```

Anatomy of the query

- **Label interpolation.** The string after `MATCH (d1:`` is built from `normalize_node_label(label.value[1])` (`src/graph/util/node_util.py`). This is safe because the enum value is constrained to drug / disease; the backticks make Cypher treat it as an identifier.
- **Regex WHERE clause.** The builder iterates `values` and produces one `d1.node_name =~ '(?i).*<value>.*'` per entry, joined by `or`. With `["lupus", "arthritis"]` you get two regex predicates OR'd together.
- `gds.nodeSimilarity.filtered.stream`. The `.filtered` variant accepts a `sourceNodeFilter` so only the regex-matched anchors act as sources of comparisons. This is what makes the call cheap enough for a synchronous HTTP endpoint — the unfiltered procedure compares every node against every other node in the projection.
- `degreeCutoff: 1`. Source nodes with fewer than one outgoing relationship in the projection are skipped (no meaningful neighbour set to compare).
- `similarityCutoff: 0.45`. Pairs scoring below 0.45 are dropped before they cross the wire. There is no explicit top-k clause; results return in descending score order and the client may paginate.

- `similarityMetric`: "COSINE". For each (source, target) pair, treat the multiset of relationships to shared neighbours as a vector and compute cosine similarity. **No learned embeddings are used here** — the `embeddings` property of Section 4 is irrelevant to this call.
- `sourceNodeFilter`: `[d1]`. Passes the node objects matched by the MATCH above. Each invocation of the stream procedure compares every filtered source against every node in the projection.

FIXME: Cypher injection

The source carries an explicit FIXME at `neo4j_foundational_similarity_adapter.py:54-56`:

```
# FIXME: cypher injection issue
# WARNING: this is wrong to inject a user query into the cypher
# however the api does not let me pass a parameter into a query
# when using a regex like this?
```

Each value in the request body is spliced raw into the Cypher regex. The Neo4j `=~` operator does not accept a query parameter in the position used here, so the original author opted for string interpolation. Until that is reworked (e.g. with an APOC procedure that supports parametrised regexes, or a server-side allow-list), treat the `values` field as **untrusted** and validate it upstream. Regex metacharacters in the input will also alter match semantics — not just security, also correctness.

3. Graph projection — building eugene_similarity_graph

`src/foundation/infra/db/adapter/neo4j_project_similarity_graph_adapter.py`

The endpoint will return zero rows (or raise a GDS “graph does not exist” error) until `eugene_similarity_graph` has been projected into Neo4j's in-memory store. The projection is built by `Neo4jProjectSimilarityGraphAdapter.project_similarity_graph()` (line ~31), which itself extends `Neo4jProjectGraphAdapter` (so the `drop_graph_if_exists` helper is inherited).

Projection Cypher template (lines ~85-102)

```
MATCH (source:%s)
OPTIONAL MATCH (source)-[r:%s]-(target:%s)
RETURN gds.graph.project(
    '%s',
    source,
    target,
    {
        relationshipProperties: r { strength: 1 }
    }
)
```

The three label/relationship lists are joined with `|` so the `MATCH` matches the union. Source labels are normalized via `normalize_node_label()` and relationship labels are backtick-escaped via `_escape_label()` (lines 104-105).

Caller

The orchestrator that calls `project_similarity_graph()` composes the lists of foundational source labels, target labels and relationship labels from `FoundationalNodeEnum / FoundationalRelationshipEnum`, and passes `replace=True` on a refresh to drop any stale projection first.

Notes

- The projection is *in-memory only*. It does not survive a Neo4j restart and must be rebuilt as part of bring-up.
- `OPTIONAL MATCH` on the relationship means even nodes with no qualifying edges are projected. Combined with `degreeCutoff: 1` in Section 2, those orphans are filtered out at query time.
- The `strength: 1` relationship property is a constant weight — the projection is effectively unweighted today. Replacing it with a real strength field is a natural extension point for tuning cosine scores.

4. Embedding pipeline (offline)

`src/foundation/infra/embedding/foundational_node_embedding_provider.py`

The live similarity endpoint does **not** read the embeddings property — it relies purely on graph topology (Section 2). The embedding pipeline still exists because (a) embeddings is one of the `featureProperties` consumed by FastRP in Section 5 to produce `prediction_embeddings`, and (b) it is a natural extension point if you want to swap `nodeSimilarity` for a true vector search.

Provider

```
class FoundationalNodeEmbeddingProvider(EmbeddingProvider):
    def __init__(self, model: SentenceTransformer | StaticModel | None
                 = None,
                 model_cache_dir: str | None = None):
        super().__init__(model_cache_dir=model_cache_dir)
        self._model = model or self.init_model()

    def to_embedding(self, value: str) -> ndarray:
        if value is None: return None
        return self._model.encode([value])
```

- The model is either a `sentence_transformers.SentenceTransformer` or a `model2vec.StaticModel`, selected by the concrete `init_model()` implementation on the base class.
- Embeddings are computed once per node value (typically `node_name`) and persisted on the node itself — not in a separate index.

Upsert Cypher

`src/foundation/infra/db/adapters/neo4j_foundational_node_adapter.py:50-89` — `upsert_embeddings(label, node_index, embeddings):`

```
MERGE (n:`<label>` { node_index: $node_index })
ON MATCH
    SET n.embeddings = $embeddings
RETURN n.node_index
```

- The label is normalised through `normalize_node_label()` and interpolated into the Cypher with backticks (the same pattern as Section 2).
- `node_index` (not `node_id`) is the upsert key. The numpy `ndarray` is converted to a plain list via `get_or_default_ndarray()` so the Neo4j driver can serialise it.
- **Note** the use of `ON MATCH` only — if the node does not already exist, `MERGE` creates a bare node with just the `node_index` property and *no* embeddings. The expectation is that foundational ingest has already created the node; the embedding step decorates it.

5. (Optional) FastRP training

[src/foundation/infra/db/adapters/neo4j_train_embeddings_adapter.py](#)

FastRP (Fast Random Projection) is a GDS-native graph embedding algorithm. It is *not* required for the live endpoint; the call to `train_fastrp_with_write()` is an offline step that decorates each projected node with a low-dimensional vector that downstream consumers (e.g. vector-index-based recommenders, link prediction models) can use.

Training Cypher (lines ~101-121)

```
CALL gds.fastrp.write(
  'eugene_similarity_graph',
  {
    embeddingDimension: 512,
    writeProperty: 'prediction_embeddings',
    iterationWeights: [0, .6, .8],
    normalizationStrength: -0.5,
    randomSeed: 75,
    featureProperties: ['embeddings', 'label_one_hot_encoding'],
    propertyRatio: .50
  }
)
YIELD nodePropertiesWritten
```

- `embeddingDimension: 512` matches the default of the `train_fastrp_with_write()` parameter.
- `iterationWeights: [0, .6, .8]` — three iterations of message-passing. The zero weight on iteration 0 means the initial node features alone do not enter the output; 1-hop and 2-hop neighbourhoods carry weight 0.6 and 0.8.
- `featureProperties` includes `embeddings` (from Section 4) and `label_one_hot_encoding` (written by `Neo4jOnehotEncodingAdapter`, out of scope here). `propertyRatio: .50` blends them with the random projection signal.
- `randomSeed: 75` is hard-coded so reruns are deterministic.
- Output goes to a `prediction_embeddings` property on each projected node (`GRAPH_EMBEDDINGS_FIELD_NAME` in `src/foundation/conf/const.py:5`).

Companion vector index

`create_embedding_index()` (line ~48) issues a `CREATE VECTOR INDEX` with `vector.similarity_function: 'cosine'` so the `prediction_embeddings` can be used by a future vector-search path (not wired into the current router).

6. Response mapper & model

src/foundation/router/response_util.py – to_similarity_response() at line ~42

```
def to_similarity_response(dataframe):
    if dataframe is None:
        return SimilarityResponse(count=0, results=[])
    results = []
    for index, row in dataframe.iterrows():
        results.append(SimilarNodes(
            score=row["similarity"],
            id1=row["id1"], id2=row["id2"],
            name1=row["name1"], name2=row["name2"],
        ))
    return SimilarityResponse(
        count=len(results), results=results)
```

Pydantic models

src/foundation/router/model/similarity_response.py:

```
class SimilarNodes(BaseModel):
    score: float
    id1: str
    name1: str
    id2: str
    name2: str

class SimilarityResponse(BaseModel):
    count: int
    results: list[SimilarNodes]
```

Example response

```
{
  "count": 3,
  "results": [
    { "score": 0.87, "id1": "DIS:0001", "name1": "Lupus",
      "id2": "DIS:0042", "name2": "Sjogren syndrome" },
    { "score": 0.71, "id1": "DIS:0001", "name1": "Lupus",
      "id2": "DIS:0099", "name2": "Rheumatoid arthritis" },
    { "score": 0.52, "id1": "DIS:0001", "name1": "Lupus",
      "id2": "DIS:0107", "name2": "Scleroderma" }
  ]
}
```

- score is the raw GDS similarity (cosine in [0,1]); no further normalisation.
- id1/id2 come from node_id via toStringOrNull so the response field is always a string even if the underlying property is numeric.
- The handler is decorated with response_model_exclude_none=True — any null fields are elided. In practice all five fields are populated by the Cypher projection above.
- The mapper does **not** deduplicate (a, b) vs (b, a). The sourceNodeFilter pins the source side to the matched anchor, so both orderings should not appear unless the anchor regex matches both nodes of a pair.

7. Suggested debugging walk

- **Step 1 — bring up the stack.** `docker-compose up eugene_neo4j eugene_ws`. Confirm `.env` points the API at the right Neo4j and that the GDS plugin is installed (Neo4j Browser: `RETURN gds.version()`).
- **Step 2 — verify the projection exists.** Open Neo4j Browser and run `CALL gds.graph.list()` `YIELD graphName, nodeCount, relationshipCount`. You must see `eugene_similarity_graph` with non-zero counts. If it is missing, run the orchestrator that calls `Neo4jProjectSimilarityGraphAdapter.project_similarity_graph(replace=True)` (Section 3) before retrying the endpoint.
- **Step 3 — (optional) refresh embeddings.** Only if you plan to consume `prediction_embeddings` elsewhere: re-run the foundational embedding ingest (`FoundationalNodeEmbeddingProvider` → `Neo4jFoundationalNodeAdapter.upsert_embeddings`) and then `Neo4jTrainEmbeddingsAdapter.train_fastrp_with_write(SIMILARITY_GRAPH_PROJECTION_NAME)`.
- **Step 4 — curl the endpoint.** `curl -sS -X POST http://localhost:8000/similarity/disease -H 'Content-Type: application/json' -d '{"values": ["lupus"]}' | jq ..` Expect a JSON document matching Section 6. Empty results usually means either the projection is missing, the regex matched no anchor, or all pairs fell below the `0.45` cutoff.
- **Step 5 — breakpoint sweep.** Set breakpoints at `similarity_router.py:48` (inspect the `parsed_label.value` and request body), `neo4j_foundational_similarity_adapter.py:36` (log the assembled query string — copy it into Neo4j Browser to see the raw rows before `DataFrame` conversion), and `response_util.py:49` (watch the row-by-row Pydantic mapping).
- **Step 6 — force a known failure.** Try `POST /similarity/gene_protein` — FastAPI should return a 422 because `gene_protein` is not a member of `FacetLabel`. Then call the adapter directly from a Python shell with `FoundationalNodeEnum.GENE_PROTEIN` to hit the `ValueError` guard at `neo4j_foundational_similarity_adapter.py:62`.

8. Reference index

HTTP entry

```
src/foundation/router/similarity_router.py      # router + DI L28, handler L31-58
src/foundation/router/model/facet_label.py      # FacetLabel enum (drug, disease)
src/foundation/router/model/similarity_search_values.py
src/foundation/router/validate_util.py          # validate_label L13-21, str_to_label_enum
```

DI / wiring

```
src/foundation/conf/conf.py                    # factory L177-184
src/foundation/conf/const.py                   # SIMILARITY_GRAPH_PROJECTION_NAME
```

Query adapter (live request path)

```
src/foundation/infra/db/adapter/
  neo4j_foundational_similarity_adapter.py      # _build_similarity_query L51-96
src/graph/util/util.py                         # ensure_connection
src/graph/util/node_util.py                   # normalize_node_label
```

Projection (offline; required before endpoint works)

```
src/foundation/infra/db/adapter/
  neo4j_project_similarity_graph_adapter.py    # project_similarity_graph L31-66
src/foundation/infra/db/adapter/neo4j_project_graph_adapter.py
```

Embedding pipeline (offline; feeds FastRP, not the endpoint)

```
src/foundation/infra/embedding/
  foundational_node_embedding_provider.py      # SentenceTransformer / model2vec
src/foundation/infra/db/adapter/
  neo4j_foundational_node_adapter.py          # upsert_embeddings L50-89
```

FastRP training (optional)

```
src/foundation/infra/db/adapter/
  neo4j_train_embeddings_adapter.py            # train_fastrp_with_write L25-46
                                              # create_embedding_index L48-68
```

Response

```
src/foundation/router/response_util.py         # to_similarity_response L42-59
src/foundation/router/model/similarity_response.py
```

Schema enums

```
src/foundation/model/foundational_node_enum.py # DRUG, DISEASE, ...
src/foundation/model/foundational_relationship_enum.py
```