

Eugene Organization Search — End-to-End Flow

Developer walkthrough: HTTP boundary → regex sanitiser → Neo4j organization scan → clinical-trial fan-out → GenericListResponse

Audience

Engineers onboarding to the Eugene biomedical knowledge graph who need a single, file-referenced trace of the `/organizations/*` endpoints. The route exposes two reads: a wildcard search over `:Organization` nodes and a fan-out from one organization through its `:ClinicalTrial` edges to the drugs / diseases the org is associated with. 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 endpoints

- GET `/organizations/{{organization_name}}?page=&page_size=` — wildcard (`*`, `?`) search over `organization_canonical_name`. Returns `[{org_id, org_name, org_type}, ...]`.
- GET `/organizations/assets/{{organization_id}}?page=&page_size=` — given an org id (e.g. `H003644`), walks `:Organization)-[*]-(:ClinicalTrial)-[*]-(:drug|:disease)` and returns one row per trial / asset edge.

Sample calls

```
curl -s 'http://localhost:8080/organizations/csl*?page=1&page_size=10' | jq .
curl -s 'http://localhost:8080/organizations/assets/H003644?page=1&page_size=25' | jq .
```

Declared in `src/organization/router/organization_search_router.py:23-101`. Router prefix `/organizations`, tag `organizations`.

How to read this guide

- Sections follow the request path top-down: schema → HTTP → adapter / Cypher → mapper → ETL provenance → debug walk → reference index.
- Section 0 untangles the two organization enum members (`ORGANIZATION_V2` vs. `ORGANIZATION_V3`) and the separate `FoundationalNodeEnum.ORGANIZATION` tuple.
- Section 5 lists a concrete debugging walk (curl, breakpoints, Cypher verification).
- Note: pagination uses 1-based page (`gt=0`) and `page_size` (`gt=0, le=500`); arithmetic is handled by the shared `Pagination` helper.

0. Schema (read this first)

The organization subgraph is anchored by a single Neo4j label, `:Organization` (capital O). Three enums collaborate to produce that label string in Cypher — this is the one place where Eugene's mixed-case label conventions matter.

Node label — OrganizationNodeEnum

[src/organization/model/organization_node_enum.py](#)

```
class OrganizationNodeEnum(Enum):
    UNKNOWN = 0, "unknown"
    COMPANY = 1, "company"
    UNIVERSITY = 2, "university"
    NONPROFIT = 3, "nonprofit"
    FOUNDATION = 4, "foundation"
    GOVERNMENT = 5, "government"
    ORGANIZATION = 6, "organization"
    HOSPITAL = 7, "hospital"
    ORGANIZATION_V2 = 8, "organization"
    ORGANIZATION_V3 = 8, "Organization" # &lt;-- used in Cypher
    SUBSIDIARY = 10, "subsidiary"
    ACQUISITION = 11, "acquisition"
    MERGER = 12, "merger"
    DEMERGER = 13, "demerger"
    SPELLING_VARIATION = 14, "organization_spelling_variation"
```

- The adapter only ever reads `ORGANIZATION_V3.value[1]` — i.e. the literal label string `Organization` (capital O). All the lowercased members are historical / for ETL-side use.
- `ORGANIZATION_V2` and `ORGANIZATION_V3` share the same id (8) but differ in label case — only V3 matches what is actually written into Neo4j by the seed scripts.
- The aliases / spelling variations enum members are write-side only; the search adapter does not traverse them.

FoundationalNodeEnum.ORGANIZATION

[src/foundation/model/foundational_node_enum.py:32](#)

```
class FoundationalNodeEnum(Enum):
    ...
    CLINICAL_TRIAL = 4, "ClinicalTrial"
    DISEASE = 7, "disease"
    DRUG = 8, "drug"
    ...
    ORGANIZATION = 32, "Organization"
```

This is the foundation-side mirror of `OrganizationNodeEnum.ORGANIZATION_V3` — same label string (`Organization`), independent id space (32). It is deliberately **not** exposed through the public `Label` enum, so the generic `/labels/{label}` route cannot list organizations; this router is the only HTTP path that can.

Relationship shape

```
(:Organization { org_id, organization_canonical_name, organization_type, ... })
(:Organization)-[*]-(:ClinicalTrial { nct_id, ... })
(:ClinicalTrial)-[*]-(:drug | :disease { node_name, ... })
```

- There is no fixed relationship type wired into the router — the assets query uses an unnamed `-[rel1]-` and reports `type(rel1)` back in the response (see Section 2). The actual edge types depend on what the ClinicalTrials.gov ingest wrote (sponsor, collaborator, intervention, condition, etc.).

- The organization side has a sibling enum `OrganizationRelationshipEnum.HAS_ALIAS` (`src/organization/model/organization_relationship_enum.py`) but it is reserved for the alias subgraph — the search router does not use it.
- Like the rest of Eugene there are no Neo4j `CREATE CONSTRAINT` statements — idempotency comes from `MERGE` on `org_id` at write time.

1. HTTP entry — the FastAPI router

Wiring

src/eugene_ws.py:53 — from organization.router import organization_search_router. src/eugene_ws.py:75 appends organization_search_router to the routers list, and the boot loop later calls app.include_router(router.router).

Router file

[src/organization/router/organization_search_router.py](#)

Imports (lines 1-11)

```
import logging
from typing import Annotated

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

from foundation.router.validate_util import (
    validate_id,
)
from foundation.router.model.generic_list_response import GenericListResponse
from organization.conf.search_conf import organization_search_adapter
```

Router declaration (lines 15-18)

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

Module-load dependency injection (line 20)

```
org_search_adapter = organization_search_adapter()
```

Eugene's manual DI pattern: a module-scope call constructs the adapter once at import-time. The factory lives in src/organization/conf/search_conf.py:27-32:

```
def _neo4j_driver() -> Driver:
    return GraphDbConnectionFactory.remote_neo4j_instance_from_env()

def _organization_search_adapter(driver: Driver) -> Neo4jOrganizationSearchAdapter:
    return Neo4jOrganizationSearchAdapter(driver=driver)

def organization_search_adapter() -> Neo4jOrganizationSearchAdapter:
    return _organization_search_adapter(driver=_neo4j_driver())
```

The driver is constructed from

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. If env vars are missing the router module fails to import and the whole service refuses to start.

Endpoint 1 — name search (lines 23-60)

```
@router.get(
   ("/{organization_name}",
    response_model=GenericListResponse,
    response_model_exclude_none=True,
)
async def list_organization_names(
```

```

organization_name: Annotated[
    str,
    Path(
        description="Limited regex ... Accepts ['*' or '?'] only,"
        " others are escaped",
        max_length=256,
        min_length=0,
        example="csl*",
    ),
],
page: Annotated[int, Query(..., example=1, gt=0)],
page_size: Annotated[int, Query(..., example=10, gt=0, le=500)],
):
    results = org_search_adapter.find_companies_by_name_pattern(
        name_pattern=organization_name, page=page, page_size=page_size
    )
    count = len(results) if results else 0
    return GenericListResponse(count=count, results=results)

```

Endpoint 2 — asset fan-out (lines 63-101)

```

@router.get(
    "/assets/{organization_id}",
    response_model_exclude_none=True,
)
async def list_organization_assets(
    organization_id: Annotated[
        str,
        Path(..., max_length=256, min_length=0, example="H003644"),
        AfterValidator(validate_id),
    ],
    page: Annotated[int, Query(..., example=1, gt=0)],
    page_size: Annotated[int, Query(..., example=10, gt=0, le=500)],
):
    results = org_search_adapter.find_assets_by_org_id(
        org_id=organization_id, page=page, page_size=page_size
    )
    count = len(results) if results else 0
    return GenericListResponse(count=count, results=results)

```

Validation layers

- **Path bounds** — both endpoints cap path params at `max_length=256`, `min_length=0`. Note that `min_length=0` means an empty string is accepted at the type level — the route matcher requires *some* path segment, but a single-character or whitespace-only segment will pass.
- **Query bounds** — `page > 0` and `0 < page_size <= 500`. FastAPI returns 422 for out-of-range values. `page_size` is generous (500) compared to other Eugene routes (`le=50`) — reflects the lower cardinality of organizations vs. genes/drugs.
- **AfterValidator(validate_id)** — only the assets endpoint runs `validate_id` (`src/foundation/router/validate_util.py:98-104`), which rejects ids containing `%`, `$`, `;`, `:`, `^`, or `*`. The name-search endpoint deliberately allows `*` (it's a wildcard).
- **No regex validation at the FastAPI layer** for `organization_name` — sanitisation happens in the adapter via `sanitize_regex_pattern` (Section 2).

Set your first breakpoint

At `organization_search_router.py:56` (the `org_search_adapter.find_companies_by_name_pattern(...)` call) or `organization_search_router.py:96` (the assets call). Inspect the raw path param, page, and

page_size before the adapter transforms anything.

2. Query adapter — the two Cyphers

[src/organization/infra/db/neo4j_organization_search_adapter.py](#)

Class & entry points (lines 16-23)

```
class Neo4jOrganizationSearchAdapter:
    def __init__(self, driver: Driver, database: str = "neo4j"):
        self.driver = driver
        self.database = database
```

Two public methods, one per endpoint: `find_companies_by_name_pattern` (line 25) and `find_assets_by_org_id` (line 93). Both call `ensure_connection(self.driver)` ([src/graph/util/util.py](#)) before opening a session — defends against stale connections after a Neo4j restart.

Pattern sanitisation (line 60)

[src/util/regex_validation.py](#)

```
def sanitize_regex_pattern(pattern: str) -> str:
    """Escapes all regex special characters except * and ?
    for basic wildcard support."""
    escaped = re.escape(pattern)
    # Allow * and ? as wildcards by converting them back
    # \* becomes .* (match any characters)
    # \? becomes . (match single character)
    escaped = escaped.replace(r"\*", ".*").replace(r"\?", ".")
    return escaped
```

- Inputs like `cs1*` become `cs1.*`; `a?bott` becomes `a.bott`.
- All other regex metacharacters (`.`, `+`, `(`, `|`, `$`, ...) are passed through `re.escape` so a malicious payload like `.*(.*)+.*$` cannot trigger a catastrophic backtrack on the Neo4j regex engine.
- The Neo4j adapter then prepends `(?i)` to make the regex case-insensitive — the canonical name is stored mixed-case (e.g. "CSL Behring") but users search in lowercase.

Cypher 1 — name pattern search (lines 56-91)

```
MATCH (o:`Organization`)
WHERE o.organization_canonical_name =~ $pattern
RETURN o.org_id          as org_id,
       o.organization_canonical_name as org_name,
       o.organization_type as org_type
SKIP $skip
LIMIT $limit

# params:
# pattern = "(?i)" + sanitize_regex_pattern(organization_name)
# skip    = (page - 1) * page_size
# limit   = page_size
```

- Label ``Organization`` is interpolated from `OrganizationNodeEnum.ORGANIZATION_V3.value[1]` (line 70). The `%s` substitution is safe because the enum is hard-coded — no user input reaches the label slot.
- `pattern`, `skip`, and `limit` are bound parameters (line 73): the only string-interpolated value is the label itself. Compare to the label/count routers, which interpolate everything.
- **No ORDER BY** — pagination is by Neo4j's default scan order. Two requests at the same (`page`, `page_size`) will return the same rows in the same order in practice, but the Cypher does not formally guarantee it. If you see pagination weirdness here, add `ORDER BY org_name`.

- `is_hidden` is **not** checked — soft-deleted orgs (if any) are still returned.

Cypher 2 — assets fan-out (lines 120-141)

```
MATCH (start:`Organization`)-[rel1]-(ct:`ClinicalTrial`)-[rel2]-(end:`disease`|`drug`)
WHERE start.org_id = $org_id
RETURN start.organization_type           as org_type,
       start.organization_canonical_name as org_name,
       type(rel1)                       as org_to_ct_rel,
       ct.nct_id                        as trial,
       type(rel2)                       as ct_to_asset_rel,
       labels(end)                      as entity_labels,
       end.node_name                    as entity_name
ORDER BY start.organization_canonical_name, trial desc
SKIP $skip
LIMIT $limit
```

params: org_id, skip, limit

- Three labels interpolated: `OrganizationNodeEnum.ORGANIZATION_V3.value[1]` = `Organization`, `FoundationalNodeEnum.CLINICAL_TRIAL.value[1]` = `ClinicalTrial`, and the `_join_labels([DISEASE, DRUG])` helper produces ``disease`|`drug`` (lines 169-174).
- `-[rel1]- / -[rel2]-` are **unnamed** — any edge type between `:Organization` and `:ClinicalTrial` (and between trial and asset) is walked. The actual type is reported back via `type(rel1)` so the caller can introspect it.
- `end.node_name` relies on every drug/disease node carrying a `node_name` property — same contract as the drug-alias route. A missing `node_name` surfaces as null in the response (not a 500 here because the response model is untyped).
- **TODO marker** at line 123: the docstring explicitly notes patents and pubmed docs are not yet walked. If you extend this, add labels to `_join_labels(...)` on line 135.
- `ORDER BY ... trial desc` stabilises pagination — rows are stable across requests at the same page.

Pagination helper

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

```
(limit, skip) = Pagination.calculate_limit_and_offset(page, page_size)
# page=1, page_size=10 -> limit=10, skip=0
# page=3, page_size=25 -> limit=25, skip=50
```


3. Mapper & response model

There is no separate mapper class for this route — the adapter already produces plain Python dicts (lines 79-86 for names, lines 151-160 for assets). The router wraps them in `GenericListResponse` and returns.

Response model

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

```
from pydantic import BaseModel
```

```
class GenericListResponse(BaseModel):
```

```
    count: int
    results: list[dict]
```

- Deliberately untyped: `results: list[dict]`. This is the same envelope used by patent / pubmed search; each row's keys vary per endpoint.
- `count` is the length of the returned page, **not** the grand total. There is no dedicated count endpoint for organizations today — pagination math is the caller's responsibility.
- `response_model=GenericListResponse` is declared only on the name-search endpoint (line 25). The assets endpoint uses `response_model_exclude_none=True` without an explicit model — FastAPI infers from the returned object at runtime.
- Empty results → `GenericListResponse(count=0, results=[])`; the route never returns 404.

Row shape — name search

```
{
  "org_id": "H003644",
  "org_name": "CSL Behring",
  "org_type": "COMPANY"
}
```

Row shape — assets fan-out

```
{
  "org_type": "COMPANY",
  "org_name": "CSL Behring",
  "org_to_ct_rel": "sponsor",
  "trial": "NCT04354805",
  "ct_to_asset_rel": "has_intervention",
  "entity_labels": "drug",
  "entity_name": "garadacimab"
}
```

Note `entity_labels` is a comma-joined string, not an array — the adapter joins the `labels(end)` list with ", " (line 158). A drug-only node is "drug"; a node tagged both :drug and :approved_drug would appear as "drug, approved_drug".

Example responses

GET /organizations/csl?page=1&page_size=3

```
{
  "count": 3,
  "results": [
    {"org_id": "H003644", "org_name": "CSL Behring", "org_type": "COMPANY"},
    {"org_id": "H001288", "org_name": "CSL Limited", "org_type": "COMPANY"},
    {"org_id": "H009922", "org_name": "CSL Plasma Inc.", "org_type": "COMPANY"}
  ]
}
```

```
}
```

```
GET /organizations/assets/H003644?page=1&page_size=2
```

```
{
```

```
  "count": 2,
```

```
  "results": [
```

```
    {"org_type": "COMPANY", "org_name": "CSL Behring",
```

```
     "org_to_ct_rel": "sponsor",
```

```
     "trial": "NCT04354805",
```

```
     "ct_to_asset_rel": "has_condition",
```

```
     "entity_labels": "disease",
```

```
     "entity_name": "hereditary angioedema"},
```

```
    {"org_type": "COMPANY", "org_name": "CSL Behring",
```

```
     "org_to_ct_rel": "sponsor",
```

```
     "trial": "NCT04354805",
```

```
     "ct_to_asset_rel": "has_intervention",
```

```
     "entity_labels": "drug",
```

```
     "entity_name": "garadacimab"}]
```

```
}
```

4. ETL — where the :Organization nodes come from

The search route is read-only. The :Organization nodes it scans are produced by a separate resolution pipeline driven from JSON checkpoint files.

CLI entry point

`src/ingest_organizations.py`

```
def main():
    load_dotenv()
    args = build_args()
    checkpoint_dir = Path(args.checkpoint_basedir)
    _ingest_resolution_checkpoint_dir(checkpoint_dir=checkpoint_dir)

def _ingest_resolution_checkpoint_dir(checkpoint_dir: Path) -> None:
    orchestrator = organization_ingest_orchestrator()
    ids = orchestrator.ingest(checkpoint_dir=checkpoint_dir)
    logger.info(f"Loaded and ingested {len(ids)} unique organization(s)")
```

Default checkpoint dir is `./output/checkpoint/organization` (a tree of JSON files produced by the upstream resolver under `src/organization/analyzer/` / `src/organization/provider/`). Override with `--checkpoint-basedir`.

Orchestrator

`src/organization/provider/organization_ingest_orchestrator.py`

```
class OrganizationIngestOrchestrator:
    def ingest(self, checkpoint_dir: Path | None) -> list[str]:
        resolutions = self._load_resolution_checkpoint_dir(checkpoint_dir)
        merged = self.organization_resolution_multipass_merge_orchestrator.merge(
            resolutions=resolutions
        )
        self._assign_ids(resolutions=merged)
        # todo: ingest into neo4j
        # return self.ingest_organizations(organizations_resolutions=merged)
        return [el.official_name for el in merged]
```

- **Heads-up:** the Neo4j upsert call is commented out (line 52). Today the CLI *resolves & merges* organization records and assigns ids, but does **not** actually write to the graph. The :Organization nodes the search route reads today were seeded by other paths (Cypher seed scripts under `bin/seed/` and the ClinicalTrials.gov ingest, which creates org nodes as a side-effect of trial sponsor/collaborator edges).
- When the TODO is unblocked, the writer is `Neo4jOrganizationAdapter.upsert_organization_resolutions` (`src/organization/infra/db/neo4j_organization_adapter.py`). It is wired but not invoked.
- The resolution loader is `OrganizationResolutionLoader` (`src/organization/load/organization_resolution_loader.py`); the data class is `OrganizationResolution` (`src/organization/model/organization_resolution.py`) and holds `official_name`, `parent`, `subsidiaries`, `acquisitions`, `spelling_variations`, `mergers`, `demergers`, `organization_type`, `query_term`, `id`.

Property contract the search Cypher depends on

- Every :Organization node must carry `org_id`, `organization_canonical_name`, `organization_type`. The first is the bound id in the assets endpoint; the second is the `=~` target in the name endpoint; the third populates `org_type` in the response.

- No CREATE CONSTRAINT statements — uniqueness comes from MERGE on org_id at write time. Two write paths with disagreeing ids will produce duplicate orgs with the same canonical name (visible in the search response).
- For asset fan-out: nodes must additionally have edges to :ClinicalTrial nodes (any rel type) and those trials must in turn link to :drug / :disease nodes with node_name set.

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 `organization_search_router.py:20` will fail loudly otherwise.
- Smoke test: `curl -s 'http://localhost:8080/organizations/csl*?page=1&page_size=5' | jq ..` Expect a 200 with up to five CSL-* orgs. Then `/organizations/assets/H003644?page=1&page_size=10` for the fan-out.
- Set a breakpoint at `neo4j_organization_search_adapter.py:60` (`safe_pattern = sanitize_regex_pattern(name_pattern)`). Try inputs like `csl*`, `?bbott`, `.*(.*)+$` — the first two should yield meaningful Neo4j regexes (`csl.*`, `.bbott`), the third should be fully escaped and match nothing.
- Step into `neo4j_organization_search_adapter.py:78` and watch `tx.run(query, params)` execute. The query plus params is logged at INFO; copy/paste both into Neo4j Browser to verify the row count matches the response's count.
- Force the empty path: request `/organizations/zzzzzzzz?page=1&page_size=10`. Expect a 200 with `count=0` — no 404, no exception.
- For the assets endpoint, breakpoint at `neo4j_organization_search_adapter.py:150` and inspect `record["entity_labels"]` before the `", ".join(...)` call — the raw value is a Python list; remember the response collapses it to a string.

6. Reference index (file paths)

Router

```
src/organization/router/organization_search_router.py
src/foundation/router/validate_util.py             (validate_id)
src/foundation/router/model/generic_list_response.py
```

Wiring

```
src/eugene_ws.py                                (lines 53, 75)
src/organization/conf/search_conf.py             (factories L15-32)
src/graph/infra/db/graph_db_connection_factory.py
```

Adapter (Cypher generation + execution)

```
src/organization/infra/db/neo4j_organization_search_adapter.py
src/util/regex_validation.py                     (sanitize_regex_pattern)
src/foundation/infra/db/util/pagination.py       (Pagination)
src/graph/util/util.py                           (ensure_connection)
```

Schema enums

```
src/organization/model/organization_node_enum.py (ORGANIZATION_V3 -> 'Organization')
src/organization/model/organization_relationship_enum.py
src/foundation/model/foundational_node_enum.py   (CLINICAL_TRIAL, DRUG, DISEASE, ORGANIZATION)
```

ETL (write side -- currently TODO into Neo4j)

```
src/ingest_organizations.py
src/organization/conf/conf.py
src/organization/provider/organization_ingest_orchestrator.py
src/organization/provider/organization_resolution_multipass_merge_orchestrator.py
src/organization/provider/organization_id_generator.py
src/organization/load/organization_resolution_loader.py
src/organization/mapper/organization_resolution_mapper.py
src/organization/model/organization_resolution.py
src/organization/model/canonical_organization_key.py
src/organization/infra/db/neo4j_organization_adapter.py (writer, not yet called)
```

Sibling read route

```
src/organization/router/organization_alias_search_router.py
```

Key takeaways

- `/organizations/{{organization_name}}` accepts only `*` and `?` wildcards — everything else is regex-escaped by `sanitize_regex_pattern` before reaching Neo4j. Catastrophic-backtrack payloads are neutralised.
- The label string in Cypher is hard-coded via `OrganizationNodeEnum.ORGANIZATION_V3.value[1] = Organization` (capital O). Do not confuse it with the lowercased `ORGANIZATION_V2` or with `FoundationalNodeEnum.ORGANIZATION` (same string, different enum, id 32).
- The assets endpoint walks any relationship type between `:Organization`, `:ClinicalTrial`, and `:drug / :disease` — the edge type is reported back via `type(rel1)`, `type(rel2)`. Patents and pubmed are not yet included (TODO at adapter line 123).
- The Neo4j writer for the resolution pipeline is wired but commented out in the orchestrator — current `:Organization` data comes from seed Cypher and the ClinicalTrials.gov ingest, not from the JSON checkpoint pipeline.