

Eugene Release Notes — End-to-End Flow

Developer walkthrough: in-process Python dict → FastAPI router → JSON response — the simplest meta route in Eugene

Audience

Engineers who need to know how Eugene advertises its own changelog. The `GET /releases` endpoint is a single-file, zero-dependency route: no Neo4j, no adapter, no mapper, no validator. The response body is literally a Python dict declared inside the handler. This guide exists so the route is documented alongside the others — and so the process for updating release notes is unambiguous. Every reference uses the form `path/to/file.py:line`.

Reference endpoint

- `GET /releases` — no path parameters, no query parameters, no auth dependencies declared on the route itself.
- Response: a JSON object keyed by version label (e.g. "v2.2 - 9/16/25") whose values are arrays of human-readable changelog strings.

Caveat

Release notes are **hard-coded** in the router module — there is no file on disk, no database row, no CMS. Updating release notes means editing `src/router/release_notes_router.py` and shipping a new build of `eugene_ws`. Substitute "Data source" for "Schema" in the next section: there is no graph schema to read; the data source is the literal dict inside the handler.

0. Data source (read this first)

Where the release notes content lives on disk:

```
src/router/release_notes_router.py
line 13:  async def get_releases():
line 14:      releases = { ... }    # the entire payload
line 42:      return releases
```

- There is no separate .md, .json, or .yaml file. The handler constructs the dict on every call and returns it directly.
- Keys are free-form version-and-date strings — e.g. "v1", "v1.1 - 8/1/25", "v2.2 - 9/16/25". No semver enforcement, no ISO date.
- Values are `list[str]` — one bullet per line, sentence case, no markdown.
- Because the dict is rebuilt per request, there is no caching layer and no startup hook. Memory cost is trivial.
- Like the rest of Eugene there is no test fixture pinning the response shape — clients must be tolerant of new keys appearing.

Verbatim payload (release_notes_router.py:14-41)

```
releases = {
    "v1": ["initial release of foundational graph endpoints"],
    "v1.1 - 8/1/25": [
        "updated foundational graph endpoints to include ids",
        "added endpoints to power ui",
        "added drug aliases endpoint",
        "added company aliases endpoint",
    ],
    "v1.2 - 8/28/25": [
        "added pagination and count endpoint",
        "added more node type labels to endpoints",
        "removed one hop endpoint in favor of n hop endpoint",
        "changed company aliases to organization aliases",
    ],
    "v2.0 - 9/4/25": [
        "loaded eugene 2.0 data. Including patents, clinical trails, and organizations",
        "added search and count patents by drug id endpoints",
    ],
    "v2.1 - 9/11/25": [
        "added endpoints for patent searches by clinical trial or gene protein target",
        "updated drug synonym endpoint to return ids and to accept a drug id parameter",
    ],
    "v2.2 - 9/16/25": [
        "added a database stats endpoint",
        "added endpoints for pubmed searches by drug, clinical trial or gene protein target",
        "updated release notes",
    ],
}
```

1. HTTP entry

`src/router/release_notes_router.py`

- Router declared at line 6: `APIRouter(prefix="", tags=["releases"])`. Empty prefix — the path is mounted at the application root.
- Single route: `GET /releases` (line 12). No path params, no query params, no `response_model`, no `Depends(...)`.
- The handler is declared `async def` but performs no I/O — Python returns immediately with the literal dict.

Verbatim handler (`release_notes_router.py:6-42`)

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

@router.get("/releases")
async def get_releases():
    releases = {
        "v1": ["initial release of foundational graph endpoints"],
        # ...see section 0 for the full payload...
    }
    return releases
```

Registration

`src/eugene_ws.py:33, 59` — `add_routers()` imports the module as `release_notes_router` and includes it via `app.include_router(router.router)` alongside `root_router` and `health_router`.

```
# src/eugene_ws.py:33
from router import root_router, health_router, release_notes_router

# src/eugene_ws.py:55-78
routers = [
    auth_router,
    root_router,
    health_router,
    release_notes_router,
    # ...other routers...
]
for router in routers:
    app.include_router(router.router)
```

2. Handler internals

There is essentially nothing to walk through: no file read, no format conversion, no transformation. The handler builds a dict literal and returns it. FastAPI / Starlette then runs the default JSONResponse encoder over the return value.

What FastAPI does to the dict on the way out

- No `response_model` was declared — so Pydantic does **not** validate or coerce the payload. The dict goes straight to the encoder.
- `jsonable_encoder` walks the dict: string keys are kept as-is, string values inside the lists are kept as-is. No timestamps, no enums, no special types — the encoding is a no-op.
- Content-Type: `application/json`. Status: 200 OK.
- Because there is no I/O, the route is effectively rate-limited by the SlowAPI middleware that `eugene_ws.py` attaches to the whole app (see section 1) — not by any per-route limiter.

Logging

The module declares `logger = logging.getLogger(__name__)` (line 4) but does not log anything inside the handler. Request-level logging comes from the JSON access-log middleware configured in `eugene_ws.py`.

3. Response model

There is no Pydantic model. The response shape is whatever the dict happens to be on the day you ship.

Example response (200)

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "v1": ["initial release of foundational graph endpoints"],
  "v1.1 - 8/1/25": [
    "updated foundational graph endpoints to include ids",
    "added endpoints to power ui",
    "added drug aliases endpoint",
    "added company aliases endpoint"
  ],
  "v1.2 - 8/28/25": [ ... ],
  "v2.0 - 9/4/25": [ ... ],
  "v2.1 - 9/11/25": [ ... ],
  "v2.2 - 9/16/25": [
    "added a database stats endpoint",
    "added endpoints for pubmed searches by drug, clinical trial or gene protein target",
    "updated release notes"
  ]
}
```

Informal schema

- Top level: dict[str, list[str]].
- Keys: free-form version labels. Current entries follow "vMAJOR.MINOR - M/D/YY" except for "v1" which has no date.
- Values: ordered list of human-readable changelog strings; oldest first within each version.
- Errors: none expected — the route has no failure mode short of the process being down. There is no 404 or 422 path.

Known typos in the payload (verbatim)

- clinical trails → should be clinical trials ("v2.0 - 9/4/25").
- clincal trial → should be clinical trial ("v2.1 - 9/11/25" and "v2.2 - 9/16/25").
- Fixing them is a one-line change but is a behavior change for any client that pattern-matches the strings — leave them alone unless you are sure no downstream system is keying off the typos.

4. Maintenance — how to update release notes

Because the payload is a Python literal, updating release notes is a code change. There is no admin UI and no runtime hot-reload.

Procedure

- Open `src/router/release_notes_router.py`.
- Add a new top-level key to the `releases` dict. Follow the existing convention: "vMAJOR.MINOR - M/D/YY". Insert the new key **after** the previous one so insertion order (which Python preserves and FastAPI emits in JSON) reads oldest-to-newest.
- Value is a `list[str]`. One bullet per string. Lower-case sentence fragments to match the existing style (no terminal period).
- Commit, run the unit tests if any cover this route, and ship a new build of `eugene_ws`. The route picks up the change on the next process start — there is no cache to bust.
- Optional: if you want a typed contract, declare a `response_model=dict[str, list[str]]` on the decorator. This is currently absent.

Migration paths (if the dict outgrows the file)

- Move the payload to `resources/release_notes.json` and load it once at module import. Adds disk I/O at startup but keeps the handler trivial.
- Move the payload to a markdown file under `docs/` and parse it at request time. Adds a markdown dependency but lets non-engineers edit the changelog.
- Either change would warrant a real Pydantic response model and a smoke test pinning the shape.

5. Suggested debugging walk

- Bring the stack up: `docker -compose up eugene_ws`. Neo4j is not required — this route never touches the graph.
- Smoke test: `curl -sS http://localhost:8000/releases | jq ..` Expect a JSON object with six top-level keys ordered v1 through v2.2.
- Confirm the registration: `curl -sS http://localhost:8000/openapi.json | jq '.paths."/releases"'`. The tags field should show ["releases"].
- Breakpoint at `src/router/release_notes_router.py:14`; inspect the releases dict. Any divergence between what you see in the debugger and what curl returns points at middleware (e.g. CORS, gzip), not at the handler.
- If the route is missing in production, check `eugene_ws.py:33` and `eugene_ws.py:59` — both lines must mention `release_notes_router` or the include loop will skip it.

6. Reference index

Data source (the dict itself)

<code>src/router/release_notes_router.py</code>	<code># lines 14-41, the releases dict</code>
---	---

HTTP entry

<code>src/router/release_notes_router.py</code>	<code># GET /releases (line 12)</code>
<code>src/eugene_ws.py</code>	<code># add_routers() L33, include L59</code>

Surrounding routers (same neighbourhood)

<code>src/router/root_router.py</code>	<code># GET /</code>
<code>src/router/health_router.py</code>	<code># GET /health</code>

OpenAPI / discoverability

<code>GET /openapi.json</code>	<code>-> .paths."/releases"</code>
<code>GET /docs</code>	<code>-> tag "releases"</code>