

Eugene Root Route — End-to-End Flow

Developer walkthrough: HTTP boundary → bearer auth dependency → static welcome payload

Audience

Engineers onboarding to the Eugene biomedical knowledge graph who need a single, file-referenced trace of GET / — the thinnest route in the service. There is no database call, no adapter, and no mapper: the handler is three lines and returns a two-key dict. The only moving part is the `Depends( get_current_user )` dependency, which validates the caller's JWT before the body is produced. Every reference uses the form `path/to/file.py:line` so you can open it directly in your IDE.

Reference endpoint

GET / — no path params, no query params, no body. Requires a valid bearer token. Response is a plain JSON object with two keys, message and user.

Sample call

```
curl -s 'http://localhost:8080/' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5...' | jq .
```

Declared in `src/router/root_router.py:11-13`. Router prefix "" (empty), tag root.

0. Purpose (read this first)

This route exists for three concrete reasons, all operational rather than functional:

- **Liveness signal** — load balancers, container probes, and curl-from-laptop sanity checks need a cheap endpoint that returns 200 when the process is up. The dedicated `/health/*` routes live on the health router; this root route serves the additional purpose of confirming that auth and routing are wired correctly end-to-end.
- **Auth smoke test** — because the handler depends on `Depends(get_current_user)`, a 200 from `GET /` proves that (a) your bearer token parses, (b) the Entra ID / JWT validator is reachable, and (c) the resulting user claims are being injected into handler arguments. A 401 here narrows the problem to auth.
- **Service banner** — the literal welcome string (`"Welcome to the euGENE API (+Microsoft Entra ID)!"`) acknowledges the Microsoft Entra ID identity provider in front of Eugene, so an operator hitting the root with a working token sees confirmation of which auth backend is wired in.

What it deliberately is not

- **Not a health check.** Real health probes go to `/health` (see `src/router/health_router.py`). Container orchestrators should target that path because it does not require a bearer token.
- **Not an OpenAPI landing page.** FastAPI's generated docs live at `/docs` and `/redoc`; the root route does not redirect there.
- **Not a version endpoint.** Build / release metadata is served by `release_notes_router` (`src/router/`). The root route returns a fixed string only.

1. HTTP entry — the FastAPI router

Wiring

src/eugene_ws.py:33 — from router import root_router, health_router, release_notes_router. src/eugene_ws.py:57 appends root_router to the routers list (immediately after the auth router so it is registered very early). The loop at the bottom of add_routers calls app.include_router(router.router) on each, which is how the bare / path gets mounted on the FastAPI app.

Router file (verbatim)

src/router/root_router.py

```
from fastapi import APIRouter, Depends
```

```
from router.auth.auth import get_current_user
```

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

```
@router.get("/")
```

```
async def app_root(user: dict = Depends(get_current_user)):
```

```
    return {"message": "Welcome to the euGENE API (+Microsoft Entra ID)!", "user": user}
```

Things to notice

- **Empty prefix.** prefix="" means the route binds at the literal / of the app. Most other Eugene routers use a path prefix (/labels, /drugs/aliases, etc.); this is the only one without.
- **Tag.** tags=["root"] groups the operation under a root section in the auto-generated OpenAPI / Swagger UI at /docs.
- **No response_model.** Unlike most Eugene routes the handler does not declare a Pydantic response model — the returned dict is serialized as-is by FastAPI's default jsonable_encoder. See Section 3 for what this means in practice.
- **async def but no I/O.** The handler awaits nothing. The dependency itself is synchronous (see Section 2); FastAPI still happily resolves it.

2. Handler internals — the auth dependency

The handler body itself is a single return statement. All non-trivial work happens before the body runs, inside FastAPI's dependency-injection step that resolves `Depends(get_current_user)`.

Dependency definition

`src/router/auth/auth.py`

```
from fastapi import Depends
from fastapi.security import HTTPAuthorizationCredentials

from router.auth.eugene_jwt import validate_eugene_access_token
from router.auth.conf import SECURITY

def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(SECURITY),
):
    token = credentials.credentials
    return validate_eugene_access_token(token)
```

Resolution order

- FastAPI sees the handler signature `user: dict = Depends(get_current_user)` and walks `get_current_user` first.
- `get_current_user` itself depends on `SECURITY` — an `HTTPBearer` (or equivalent) instance from `src/router/auth/conf.py`. FastAPI parses the `Authorization` header and produces an `HTTPAuthorizationCredentials` dataclass whose `.credentials` attribute holds the raw bearer token string.
- `validate_eugene_access_token(token)` performs the actual signature / claims check. On success it returns a dict of user claims; on failure it raises an `HTTPException` that short-circuits the request with 401.
- The returned dict is bound to the handler's `user` parameter and embedded directly in the response body.

What happens if the header is missing

- `SECURITY (HTTPBearer(...))` rejects requests with no `Authorization` header before `get_current_user` even runs. The status code (401 vs 403) depends on the `auto_error` flag used to construct it — see `src/router/auth/conf.py`.
- If the header is present but malformed, `SECURITY` raises directly. If the header is well-formed but the token is invalid, `validate_eugene_access_token(token)` raises. Either way the handler body never executes and the client sees a 4xx.

Set your first breakpoint

At `src/router/root_router.py:13` (the return line). Inspect:

- `user` — the dict produced by `validate_eugene_access_token(token)`. The shape is whatever the JWT validator returns (claims map — typically `sub`, `email`, `name`, `roles`, etc.).

3. Response model

The handler returns a Python dict literal. FastAPI serializes it through the default `jsonable_encoder` and the route has no `response_model`, so what you see on the wire is exactly the dict structure, with the user claims nested under `user`.

Response shape (verbatim from the handler)

```
{
  "message": "Welcome to the euGENE API (+Microsoft Entra ID)!",
  "user":    { ... claims from validate_eugene_access_token ... }
}
```

Example response with concrete claims

```
{
  "message": "Welcome to the euGENE API (+Microsoft Entra ID)!",
  "user": {
    "sub": "a3b8...",
    "email": "engineer@example.com",
    "name": "Engineer Eugene",
    "roles": ["eugene.reader"],
    "iss": "https://login.microsoftonline.com/<tenant>/v2.0",
    "aud": "<client-id>",
    "exp": 1747700000
  }
}
```

Implications of having no `response_model`

- **Claim shape is uncontracted.** Whatever `validate_eugene_access_token(token)` returns is what the client gets. If the validator starts including new claim fields, the response widens silently. There is no Pydantic schema to break a downstream consumer.
- **No `response_model_exclude_none` filter.** Any `None` values in the claims dict are serialized as JSON `null`.
- **OpenAPI shows 200 with no schema.** Swagger UI at `/docs` will document the operation but cannot show a response example. Clients should treat this body as opaque except for the `message` key.

Status codes

- 200 OK — valid bearer token, claims returned.
- 401 Unauthorized — missing / malformed header, or invalid token (raised inside `validate_eugene_access_token(token)`).
- 403 Forbidden — possible depending on how SECURITY (HTTPBearer) is configured.

4. Use in production — heartbeat and smoke

Because the route does no I/O beyond JWT validation, it is cheap enough to call repeatedly. It is best thought of as an auth-aware heartbeat rather than a feature endpoint.

Recommended uses

- **Post-deploy smoke test.** CI/CD pipelines hit GET / with a service-principal token to confirm a freshly deployed container has come up *and* is wired to the correct Entra ID tenant. A 200 here is stronger evidence than an unauthenticated /health ping.
- **Token validation harness.** When a client team reports "my token isn't working," ask them to curl GET / first. A 401 isolates the problem to the token; a 200 isolates it to a downstream route.
- **Banner / hello-world.** Operators logging in to a new environment via Swagger UI see the welcome string and confirmed claims, which orients them quickly.

Anti-patterns

- **Don't use it as a Kubernetes liveness probe.** Probes shouldn't require a bearer token; use /health instead.
- **Don't parse user as a stable contract.** The claims dict shape is whatever `validate_eugene_access_token(token)` returns; a real client should call a dedicated user-info endpoint if one exists, not scrape this route's response.
- **Don't poll at high frequency without caching the token.** Each call hits the JWT validator; under heavy polling that becomes the bottleneck rather than `app_root` itself.

Observability tips

- Successful calls log nothing handler-side — the handler is three lines. Any log line you see for this route comes from the FastAPI access log or the JWT validator.
- If you need to confirm a particular caller hit the root, instrument the access log to include the JWT subject claim (sub) for GET / responses.

5. Suggested debugging walk

- Bring the service up: `docker -compose up eugene_ws`. Hit `curl -s -i http://localhost:8080/` with no header — expect a 401/403 from SECURITY. This confirms the route is registered and auth is active.
- Now hit it with a valid token: `curl -s -H 'Authorization: Bearer <jwt>' http://localhost:8080/ | jq ..` Expect 200 with the welcome string and a populated user object. Set a breakpoint at `src/router/root_router.py:13` and inspect user to confirm the claim set you expect (sub, email, roles).
- If 401 with a token you believe is valid: step into `src/router/auth/auth.py:11` to inspect `credentials.credentials`, then into `validate_eugene_access_token` (`src/router/auth/eugene_jwt.py`) to see which check fails (signature, issuer, audience, expiry).

6. Reference index (file paths)

Router	
src/router/root_router.py	(entire file, 13 lines)
Wiring	
src/eugene_ws.py	(line 33 import, line 57 list)
Auth dependency	
src/router/auth/auth.py	(get_current_user)
src/router/auth/conf.py	(SECURITY = HTTPBearer(...))
src/router/auth/eugene_jwt.py	(validate_eugene_access_token)
Related operational routes	
src/router/health_router.py	(unauthenticated liveness)
src/router/release_notes_router.py	(build / version metadata)

Key takeaways

- GET / is a three-line, auth-gated hello-world. No DB, no adapter, no mapper, no response model.
- Its job is operational: confirm the service is up *and* the Entra ID JWT path is healthy. For pure liveness, use /health instead because it doesn't require a token.
- The user field in the response is whatever validate_eugene_access_token(token) returns — treat it as opaque, do not contract on individual claim keys.
- It is the only Eugene router that mounts at an empty prefix; everything else is namespaced under a path segment.