Authorization & RBAC

Downloadable reference generated from the NetOS Markdown documentation.

NetOS Authorization and RBAC

NetOS uses Microsoft SSO / Microsoft Entra ID for authentication only. Microsoft proves identity. NetOS decides authorization with internal users, roles, permissions, scopes, and field-level filtering.

Security Model

- Default deny: every `/api/v1` route requires an active internal user with `app_access` and the route permission.
- Backend enforcement is authoritative. Frontend hiding is only a usability layer.
- Financial and executive fields are filtered before API responses are returned.
- Permission changes are written to the activity audit log with `event_type="authorization"`.
- Sensitive financial access is covered by the normal API activity log and should be expanded for especially sensitive exports or investor reporting endpoints as those are added.

Identity Flow

1. Microsoft SSO authenticates the user.
2. The reverse proxy forwards trusted identity headers to the API.
3. NetOS normalizes the email and optional Entra object ID.
4. NetOS loads the internal `users` record.
5. If the user is inactive or has no record, access is denied unless they match the bootstrap Super Admin allowlist.
6. Roles and direct user permission overrides are resolved.
7. Route and field authorization is applied.

Authenticated does not mean authorized.

Seeing The Current User

The current API-resolved user is available at:

```
/api/v1/authz/me
```

The main navigation footer also shows the email that NetOS is authorizing and the source of that identity:

- `Microsoft SSO`: the API received and trusted proxy authentication headers.

- `Dev fallback`: the API did not use Microsoft proxy headers and fell back to development auth.

Production should show `Microsoft SSO`. If it shows `Dev fallback`, confirm `ALLOW_DEV_AUTH=false` and `TRUST_PROXY_AUTH_HEADERS=true` in the running API environment and verify the SSO proxy forwards identity headers such as `X-Auth-Request-Email`.

Bootstrap Super Admin

Set:

```
RBAC_BOOTSTRAP_SUPER_ADMIN_EMAILS=admin@example.com,second-admin@example.com
```

When one of those Microsoft-authenticated users first reaches the app, NetOS creates or activates the internal user and assigns the `Super Admin` role. If the variable is not set, NetOS falls back to `ADMIN_EMAIL`; in dev auth mode it also allows `DEV_AUTH_DEFAULT_EMAIL`.

Data Model

Core tables:

- `users`: internal user mapped to email and optional Entra object ID.
- `roles`: named permission bundles.
- `permissions`: granular permission catalog.
- `role_permissions`: permissions included in roles.
- `user_roles`: role assignments, including `scope_type` and `scope_ref_id`.
- `user_permissions`: optional direct allow/deny overrides.
- `permission_scopes`: reusable labels for global, project, site, and department scopes.
- `user_activity_logs`: audit history, including authorization changes.

Scopes currently support `global`, `project`, `site`, and `department`. Global enforcement is live; entity-scope query filters are intended to be applied module by module where project/site assignment data exists.

Default Roles

- `Super Admin`: all permissions.
- `Executive`: operational dashboards and financial summaries, including executive financial visibility.
- `Finance Admin`: full finance, invoices, reporting, and financial field visibility.
- `Project Manager`: project and assigned operational data with project/asset/circuit costs.
- `Network Engineer`: technical inventory edit access with relevant asset and circuit costs.
- `Field Technician`: assigned operational/site tooling, no financial fields by default.
- `Read-Only User`: non-sensitive operational read access only.

Roles are composed of permissions. Do not hard-code role names in new features except for bootstrap or emergency super-admin behavior.

Protecting Routes

Add a permission to `apps/api/app/core/permissions.py`, then add route mapping in `ROUTE_PERMISSION_PREFIXES`.

For explicit endpoint checks:

```
from app.core.auth import require_permission

@router.post("/example", dependencies=[Depends(require_permission("assets_edit"))])
async def create_example(...):
    ...
```

Every route under `/api/v1` also runs `require_route_permission`.

Protecting Fields

Financial fields must not be sent and hidden only in the browser. Use field permissions:

- `field.asset_cost.view`
- `field.circuit_cost.view`
- `field.customer_revenue.view`
- `field.margin.view`
- `field.capex.view`
- `field.executive_summary.view`

Example:

```
from app.core.auth import can_view_field

if not can_view_field(principal, "field.circuit_cost.view"):
    payload["mrc_usd"] = None
    payload["nrc_usd"] = None
```

Exports and search endpoints must use the same filtered payload strategy and must not include protected fields in labels, descriptions, or metadata.

Admin UI

The Admin page includes an Authorization section for:

- creating users
- activating and deactivating users
- assigning roles
- assigning global/project/site/department scopes
- editing role permission composition

The sidebar hides navigation items when the current user lacks the associated view permission.

Setup

Apply migrations:

```
docker exec netos_api_1 alembic upgrade head
```

Restart API/web after code changes:

```
docker restart netos_api_1 netos_web_1
```

Run tests:

```
docker exec netos_api_1 pytest apps/api/tests/test_rbac_authorization.py
```

Security Considerations

- Keep Microsoft Conditional Access and MFA in Entra; NetOS remains compatible because it only consumes the authenticated identity.
- Do not grant `app_access` automatically to every Microsoft user.
- Prefer scoped roles for project/site/department work.
- Use direct user permissions sparingly and audit them.
- Never add wildcard permissions.
- New financial, payroll, compensation, debt, investor reporting, export, or search features must define explicit route and field permissions before release.