HRMS RBAC – Setup & API Guide

This guide explains how to **bootstrap the first admin/manager access from the command line**, then manage roles/permissions via the **RBAC REST APIs**.

---

1) Prerequisites

- Python venv activated in `apps/backend`
- PowerShell: `..\.venv\Scripts\Activate.ps1`
- Set Flask app:
  `setx FLASK_APP "hrms_api.wsgi:app"` (PowerShell session) or `$env:FLASK_APP = "hrms_api.wsgi:app"`
- Server (for local testing):
  `python -m flask run --debug`

  If you changed `.env` or config, restart the server.

---

2) One-time Bootstrap (create first role & grant access)

Option A — Built-in CLI (if available)

```
# Give the user the "admin" role
python -m flask grant-admin admin@demo.local
```

> This adds the **admin** role to the user. If your build also ships a grant CLI for permissions, use it to give the admin role the **rbac.manage** permission. If not, use Option B.

Option B — Paste-once script (PowerShell → Python)

Grants **rbac.manage** to the **admin** role and ensures the user has that role:

```
$code = @'
from hrms_api import create_app
from hrms_api.extensions import db
from hrms_api.models.user import User
from hrms_api.models.security import Role, Permission, RolePermission, UserRole

app = create_app()
with app.app_context():
    admin_role = Role.query.filter_by(code="admin").first()
    if not admin_role:
```

```
        admin_role = Role(code="admin")
        # optional display field if your Role model has it
        for f in ("name","title","label","display_name","role_name"):
            if hasattr(admin_role, f): setattr(admin_role, f, "Administrator");
break
        db.session.add(admin_role); db.session.flush()

    p = Permission.query.filter_by(code="rbac.manage").first()
    if not p:
        p = Permission(code="rbac.manage")
        for f in ("name","title","label","display_name","desc","description"):
            if hasattr(p, f): setattr(p, f, "RBAC Manage"); break
        db.session.add(p); db.session.flush()

    if not RolePermission.query.filter_by(role_id=admin_role.id,
permission_id=p.id).first():
        db.session.add(RolePermission(role_id=admin_role.id,
permission_id=p.id))

    u = User.query.filter_by(email="admin@demo.local").first()
    if not u: raise SystemExit("admin@demo.local not found")
    if not UserRole.query.filter_by(user_id=u.id,
role_id=admin_role.id).first():
        db.session.add(UserRole(user_id=u.id, role_id=admin_role.id))

    db.session.commit()
    print("✅ admin -> rbac.manage granted; user assigned to admin role")
'@
$code | python -
```

After this, the user **admin@demo.local** can call RBAC APIs.

---

3) Get a JWT (Login)

Use either endpoint depending on your build:

```
POST /api/v1/auth/login
POST /api/v1/auth/simple-login
Body: { "email": "admin@demo.local", "password": "4445" }
```

Copy `access` or `access_token` and send it as a header on every RBAC API call:

```
Authorization: Bearer <JWT>
```

4) RBAC API Endpoints (what they do & how to use)

Base path: `/api/v1/rbac` (already registered)

4.1 Ensure Defaults (idempotent)

Creates common roles (**employee, manager, hr, admin**) and a minimal permission set. Safe to call many times.

```
POST /api/v1/rbac/ensure-defaults
Headers: Authorization: Bearer <JWT>
Response: { success: true, data: { roles: [...], perms: [...] } }
```

4.2 Roles

- **List roles**

```
GET /api/v1/rbac/roles
Headers: Authorization
```

- **Create/Upsert role** (creates or updates display name)

```
POST /api/v1/rbac/roles
Headers: Authorization, Content-Type: application/json
Body: { "code": "auditor", "name": "Auditor" }
```

- **Delete role**

```
DELETE /api/v1/rbac/roles/{role_code}
Headers: Authorization
```

4.3 Permissions

- **List permissions**

```
GET /api/v1/rbac/perms
Headers: Authorization
```

- **Create/Upsert permission**

```
POST /api/v1/rbac/perms
Headers: Authorization, Content-Type: application/json
Body: { "code": "payroll.read", "name": "Read Payroll" }
```

- **Delete permission**

```
DELETE /api/v1/rbac/perms/{perm_code}
Headers: Authorization
```

4.4 Role ↔ Permission links

- **Grant permission to role**

```
POST /api/v1/rbac/roles/{role_code}/grant
Headers: Authorization, Content-Type: application/json
Body: { "permission": "payroll.read" }
```

- **Revoke permission from role**

```
POST /api/v1/rbac/roles/{role_code}/revoke
Headers: Authorization, Content-Type: application/json
Body: { "permission": "payroll.read" }
```

4.5 User ↔ Role links

- **Assign role to user**

```
POST /api/v1/rbac/users/assign
Headers: Authorization, Content-Type: application/json
Body: { "email": "emp@demo.local", "role": "auditor" }
```

- **Unassign role from user**

```
POST /api/v1/rbac/users/unassign
Headers: Authorization, Content-Type: application/json
Body: { "email": "emp@demo.local", "role": "auditor" }
```

---

5) Typical Workflows

Create a new role and give it specific access

1. Create role `auditor`
   `POST /rbac/roles { code:"auditor", name:"Auditor" }`
2. Create `payroll.read` perm (if not present)
   `POST /rbac/perms { code:"payroll.read", name:"Read Payroll" }`
3. Grant perm to role
   `POST /rbac/roles/auditor/grant { permission:"payroll.read" }`
4. Assign role to a user
   `POST /rbac/users/assign { email:"emp@demo.local", role:"auditor" }`

Clean up

- Revoke perm: `POST /rbac/roles/auditor/revoke { permission:"payroll.read" }`
- Unassign role: `POST /rbac/users/unassign { email:"emp@demo.local", role:"auditor" }`
- Delete role: `DELETE /rbac/roles/auditor`

---

6) Postman Tips

- Collection-level **Authorization: Bearer Token** with `{{token}}`.
- Login request **Tests** script should capture token (key might be `access`, `access_token`, or nested under `data`).
- Every RBAC request must inherit the collection auth or set the header manually.

---

7) Troubleshooting

- **422 Bad Authorization header** → Header must be exactly `Authorization: Bearer <JWT>`.
- **403 Forbidden** → Logged-in user lacks `rbac.manage`. Run bootstrap (Section 2) and login again.
- **500 AttributeError on Role/Permission name** → Your models may use `title/label/display_name`. The RBAC code now auto-detects the display field; ensure server reloaded.
- **Routes not found** → Ensure blueprint registration in `create_app()`:

```
from .rbac import bp as rbac_bp
app.register_blueprint(rbac_bp)
```

---

8) Appendix: Default perms seeded by Ensure-Defaults

```
attendance.read
employees.create
employees.read
employees.update
leave.read
master.departments.read
master.locations.read
rbac.manage
```

> Adjust the list in `rbac_ensure_defaults()` if your canonical naming differs (e.g., `employee.*` vs `employees.*`).

Done. Use this document as the single source for RBAC setup and day-to-day usage.