Roles and Permissions

Bilko Roles and Permissions

> **Project:** Bilko **Version:** 1.0 **Date:** 2026-02-24 **Status:** Specification **Applies to:** `apps/api/` — `authGuard` + `roleGuard` middleware, `organizationScope`

Overview

Bilko uses Role-Based Access Control (RBAC) with four fixed roles. Roles are assigned per user within an organization. A user belongs to exactly one organization and has exactly one role within it.

Roles defined in Prisma schema (`packages/database/prisma/schema.prisma`):

```
enum UserRole {
  owner
  admin
  accountant
  viewer
}
```

Roles are embedded in the JWT access token claim `role` and enforced on every request via the `roleGuard` middleware. No additional DB lookup is required for authorization at runtime.

Role Definitions

owner

The organization creator. There is exactly one `owner` per organization. The `owner` role is assigned automatically on registration and cannot be granted via invitation.

Capabilities:

- All financial operations (create, edit, approve, send, cancel)
- Full user management (invite, change roles, remove users)
- Organization settings management (name, currency, language, fiscal year)
- Chart of accounts management
- Organization deletion

Restrictions:

- Cannot be invited — assigned only at registration
- Cannot have their own role changed by anyone
- Cannot be removed by any other user

---

admin

A trusted operator with near-full access. Assigned by the `owner` via invitation or role change.

Capabilities:

- All financial operations (create, edit, approve, send, cancel)
- User management: invite users, change roles of non-owner users
- Organization settings management
- Chart of accounts management

Restrictions:

- Cannot change the `owner`'s role
- Cannot delete the `owner`
- Cannot delete the organization
- Cannot promote another user to `owner`

---

accountant

A bookkeeping operator who can perform all financial data entry but cannot administer the organization or its users.

Capabilities:

- Create, edit, and send invoices
- Create and edit expenses (pending only)
- Create manual journal entries (transactions)
- Import bank statements (CSV)

- Reconcile bank transactions with GL
- View all reports and financial data

Restrictions:

- Cannot approve or delete expenses
- Cannot invite or manage users
- Cannot change organization settings
- Cannot create or deactivate accounts (chart of accounts)
- Cannot create or manage bank accounts

---

viewer

Read-only access to all financial data within the organization. Suitable for external accountants, auditors, or stakeholders who need visibility without write access.

Capabilities:

- View all invoices, expenses, contacts, bank accounts, and transactions
- View all reports (dashboard, P&L, balance sheet, cash flow, VAT, trial balance)
- Download invoice PDFs

Restrictions:

- Cannot create, edit, or delete any record
- Cannot approve expenses
- Cannot send invoices
- Cannot import bank statements or reconcile
- Cannot manage users or settings

Permission Inheritance Model

Permissions do not inherit hierarchically. Each role has a discrete, fixed set of permissions. However, higher roles consistently include all permissions of lower roles:

```
viewer ⊂ accountant ⊂ admin ⊂ owner
```

This means:

- Everything a `viewer` can do, an `accountant` can also do
- Everything an `accountant` can do, an `admin` can also do
- Everything an `admin` can do, an `owner` can also do

The single exception is `owner`-exclusive operations (role assignment, organization deletion) which are not part of `admin`'s scope.

Endpoint Access Matrix

Full access matrix for all 50 API endpoints. ✅ = access granted. ❌ = `403 BILKO-9001` returned.

Authentication (no role required)

Endpoint	owner	admin	accountant	viewer	Notes
`POST /auth/register`	—	—	—	—	Public — no auth required
`POST /auth/login`	—	—	—	—	Public — no auth required
`POST /auth/refresh`	—	—	—	—	Cookie-based — no role required
`POST /auth/logout`	✅	✅	✅	✅	Any authenticated user
`GET /auth/me`	✅	✅	✅	✅	Any authenticated user

Organization

Endpoint	owner	admin	accountant	viewer	Notes
`GET /organization`	✅	✅	✅	✅	All roles
`PUT /organization`	✅	✅	✅	✅	Settings change: owner + admin

Users

Endpoint	owner	admin	accountant	viewer	Notes
`GET /users`	✅	✅	✅	✅	User list: owner + admin only

Endpoint	owner	admin	accountant	viewer	Notes
`POST /users/invite`	✅	✅	❌	❌	admin can invite up to admin role
`PUT /users/:id/role`	✅	❌	❌	❌	Role changes: owner only
`DELETE /users/:id`	✅	❌	❌	❌	User removal: owner only

Contacts

Endpoint	owner	admin	accountant	viewer	Notes
`GET /contacts`	✅	✅	✅	✅	All roles
`POST /contacts`	✅	✅	✅	✅	Create: owner + admin + accountant
`GET /contacts/:id`	✅	✅	✅	✅	All roles
`PUT /contacts/:id`	✅	✅	✅	✅	Edit: owner + admin + accountant
`DELETE /contacts/:id`	✅	✅	❌	❌	Soft-delete: owner + admin

Invoices

Endpoint	owner	admin	accountant	viewer	Notes
`GET /invoices`	✅	✅	✅	✅	All roles
`POST /invoices`	✅	✅	✅	✅	Create: owner + admin + accountant
`GET /invoices/:id`	✅	✅	✅	✅	All roles
`PUT /invoices/:id`	✅	✅	✅	✅	Edit (draft only): owner + admin + accountant
`PATCH /invoices/:id/status`	✅	✅	✅	✅	Status change: owner + admin + accountant
`GET /invoices/:id/pdf`	✅	✅	✅	✅	PDF download: all roles

Endpoint	owner	admin	accountant	viewer	Notes
`POST /invoices/:id/send`	✅	✅	✅	❌	Email send: owner + admin + accountant

Expenses

Endpoint	owner	admin	accountant	viewer	Notes
`GET /expenses`	✅	✅	✅	✅	All roles
`POST /expenses`	✅	✅	✅	✅	Create: owner + admin + accountant
`GET /expenses/:id`	✅	✅	✅	✅	All roles
`PUT /expenses/:id`	✅	✅	✅	✅	Edit (pending only): owner + admin + accountant
`PATCH /expenses/:id/approve`	✅	✅	❌	❌	Approve: owner + admin only
`DELETE /expenses/:id`	✅	✅	❌	❌	Delete (pending only): owner + admin

Bank Accounts

Endpoint	owner	admin	accountant	viewer	Notes
`GET /bank-accounts`	✅	✅	✅	✅	All roles
`POST /bank-accounts`	✅	✅	❌	❌	Create: owner + admin
`GET /bank-accounts/:id/transactions`	✅	✅	✅	✅	All roles
`POST /bank-accounts/:id/import`	✅	✅	✅	❌	CSV import: owner + admin + accountant
`POST /bank-accounts/:id/reconcile`	✅	✅	✅	❌	Reconcile: owner + admin + accountant

Reports

Endpoint	owner	admin	accountant	viewer	Notes
`GET /reports/dashboard`	✅	✅	✅	✅	All roles
`GET /reports/profit-loss`	✅	✅	✅	✅	All roles
`GET /reports/balance-sheet`	✅	✅	✅	✅	All roles
`GET /reports/cash-flow`	✅	✅	✅	✅	All roles
`GET /reports/vat`	✅	✅	✅	✅	All roles
`GET /reports/trial-balance`	✅	✅	✅	✅	All roles

Chart of Accounts

Endpoint	owner	admin	accountant	viewer	Notes
`GET /accounts`	✅	✅	✅	✅	All roles
`POST /accounts`	✅	✅	✅	✅	Create: owner + admin
`PUT /accounts/:id`	✅	✅	✅	✅	Edit/deactivate: owner + admin

Transactions (General Ledger)

Endpoint	owner	admin	accountant	viewer	Notes
`GET /transactions`	✅	✅	✅	✅	All roles
`POST /transactions`	✅	✅	✅	✅	Manual journal entry: owner + admin + accountant

---

Settings

Endpoint	owner	admin	accountant	viewer	Notes
`GET /settings/tax-rates`	✅	✅	✅	✅	All roles
`PUT /settings/tax-rates`	✅	✅	❌	❌	Update: owner + admin

Currencies

Endpoint	owner	admin	accountant	viewer	Notes
`GET /currencies`	✅	✅	✅	✅	All roles
`GET /exchange-rates`	✅	✅	✅	✅	All roles

UI Element Visibility Per Role

Frontend elements are conditionally rendered based on the user's role (available in Zustand store from `/auth/me`). This is a display-only optimization — the API enforces permissions independently.

Navigation Sidebar

Nav Item	owner	admin	accountant	viewer
Dashboard	✅	✅	✅	✅
Invoices	✅	✅	✅	✅
Expenses	✅	✅	✅	✅
Banking	✅	✅	✅	✅
Reports	✅	✅	✅	✅
Chart of Accounts	✅	✅	✅	✅ (read-only)
Contacts	✅	✅	✅	✅ (read-only)
Settings	✅	✅	✅	✅
Users & Teams	✅	✅	✅	✅

Action Buttons

Action	owner	admin	accountant	viewer
"New Invoice" button	✓	✓	✓	Hidden
"Send Invoice" button	✓	✓	✓	Hidden
"Mark as Paid" button	✓	✓	✓	Hidden
"Cancel Invoice" button	✓	✓	✓	Hidden
"New Expense" button	✓	✓	✓	Hidden
"Approve Expense" button	✓	✓	Hidden	Hidden
"Delete Expense" button	✓	✓	Hidden	Hidden
"New Contact" button	✓	✓	✓	Hidden
"Edit Contact" button	✓	✓	✓	Hidden
"Delete Contact" button	✓	✓	Hidden	Hidden
"Invite User" button	✓	✓	Hidden	Hidden
"Change Role" dropdown	✓	Hidden	Hidden	Hidden
"Remove User" button	✓	Hidden	Hidden	Hidden
"Add Bank Account" button	✓	✓	Hidden	Hidden
"Import CSV" button	✓	✓	✓	Hidden
"Reconcile" button	✓	✓	✓	Hidden
"New Account" (CoA) button	✓	✓	Hidden	Hidden
"Deactivate Account" button	✓	✓	Hidden	Hidden
"Manual Journal Entry"	✓	✓	✓	Hidden
"Update Settings" button	✓	✓	Hidden	Hidden

Settings Page Sections

Section	owner	admin	accountant	viewer

Organization Info (editable)	☐	☐	—	—
Tax Rates (editable)	☐	☐	—	—
Users List	☐	☐	—	—
Invite User form	☐	☐	—	—
Change User Role	☐	☐	—	—
Remove User	☐	☐	—	—
Delete Organization	☐	☐	—	—

Accountant and viewer roles do not have access to the Settings page — the nav item is hidden and direct URL access returns `403 BILKO-9001`.

Data Scope Rules — Organization-Level Multi-tenancy

Every authenticated user has their `organizationId` embedded in the JWT access token payload:

```
interface AccessTokenPayload {
  sub: string       // User ID
  email: string
  role: UserRole
  orgId: string     // Organization ID — always present
  iat: number
  exp: number
}
```

organizationScope Middleware

The `organizationScope` middleware runs after `authGuard` and `roleGuard` on all data-access endpoints. It attaches `req.organizationId` from the JWT and enforces that every DB query is scoped to that organization.

```
// src/middleware/organization.middleware.ts
function organizationScope(req: AuthRequest, res: Response, next: NextFunction) {
  if (!req.user?.organizationId) {
    return res.status(401).json({ error: { code: 'BILKO-1005', message: 'Authentication is
```

```
required.' } })
  }
  req.organizationId = req.user.organizationId
  next()
}
```

Mandatory Query Scoping

Every Prisma query on organization-owned resources **must** include `where: { organizationId: req.organizationId }`. This prevents cross-organization data leakage even if a user somehow obtains a valid JWT with a different `orgId`.

```
// Example: invoice fetch — always org-scoped
const invoice = await prisma.invoice.findFirst({
  where: {
    id: req.params.id,
    organizationId: req.organizationId,   // MANDATORY — never omit
  }
})

if (!invoice) {
  throw new NotFoundError('BILKO-3001')   // Returns 404 — same as if not found
}
```

Returning `404` (not `403`) when a resource exists in a different organization is intentional — it prevents enumeration of cross-org record IDs.

Data Isolation Guarantees

Scenario	Behavior
User accesses own org's invoice	`200` — returns invoice
User accesses invoice from another org	`404 BILKO-3001` — treated as not found
User's JWT has invalid `orgId`	`404 BILKO-2001` — organization not found
Deleted organization's records	Cascade delete (defined in Prisma schema)
User removed from org but JWT still valid	First request returns `404 BILKO-2001`

All 15 database models with `organizationId` field enforce this scoping:

- `Organization`, `User`, `Account`, `Contact`
- `Invoice`, `InvoiceItem`, `Expense`, `Transaction`

- `BankAccount`

Global models (not org-scoped, shared across all organizations):

- `Currency`, `ExchangeRate`, `AccountType` — read-only reference data
- `SchemaVersion` — migration tracking

Role Assignment and Invitation Flow

Registration — Owner Assignment

When an organization is registered, the first (and only) `owner` is created:

```
POST /auth/register
  → Create Organization
  → Create User (role = 'owner')
  → Seed Chart of Accounts (country-based defaults)
  → Return JWT pair
```

The `owner` role cannot be granted by invitation. The `POST /users/invite` endpoint explicitly rejects `role: 'owner'` with `422 BILKO-9003`.

Invitation Flow

An `owner` or `admin` can invite new users with roles `admin`, `accountant`, or `viewer`.

```
POST /users/invite
  { email, fullName, role: 'admin' | 'accountant' | 'viewer' }
  → Validate role (cannot be 'owner')
  → Create user record (passwordHash = null, isActive = false)
  → Generate one-time invite token (JWT, expires in 7 days)
  → Send invite email via SendGrid
  → Return { user, inviteLink }
```

The invitee clicks the link:

```
GET /auth/accept-invite?token=<JWT>
  → Verify token signature + expiry
```

```
  → Prompt user to set password (frontend form)

POST /auth/accept-invite
  { token, password }
  → Hash password
  → Set user.isActive = true
  → Invalidate invite token
  → Return JWT pair (auto-login)
```

Invite constraints:

- Invite link is single-use — consumed on first `POST /auth/accept-invite`
- Invite expires after 7 days (`BILKO-1012`)
- Cannot invite an email that already has a user in the organization (`BILKO-2008`)
- `admin` can invite up to `admin` role (cannot invite users with higher role than themselves)

Role Change Flow

Only the `owner` can change another user's role:

```
PUT /users/:id/role
  { role: 'admin' | 'accountant' | 'viewer' }
  → Validate: caller must be 'owner'
  → Validate: target is not owner (BILKO-2006)
  → Validate: target is not caller (BILKO-2007)
  → Update user.role in DB
  → Log to LoggedAction
  → Return updated user
```

Behavior after role change:

- Change is effective on the **next API request** by the affected user
- Current active JWT is not invalidated immediately (access tokens expire in 15 min)
- On token refresh, the new role is embedded in the new JWT
- For immediate effect, invalidate all refresh tokens (force re-login) — not implemented in MVP

User Removal Flow

Only the `owner` can remove users:

```
DELETE /users/:id
  → Validate: caller must be 'owner'
  → Validate: target is not owner (BILKO-2006)
  → Validate: target is not caller (BILKO-2007)
  → Set user.isActive = false (soft delete — preserves audit trail)
  → Add all user's refresh tokens to blacklist
  → Log to LoggedAction
  → Return 204
```

User data (invoices created, expenses entered) is retained for audit trail purposes. The `users` table record remains with `isActive = false`. The user cannot log in after removal.

Permission Flow Diagram

```
flowchart TD
    REQUEST["API Request\nGET/POST/PUT/PATCH/DELETE /api/v1/*"] --> HELMET["Helmet\nSecurity headers"]
    HELMET --> CORS["CORS\nOrigin validation"]
    CORS --> RL["Rate Limiter\nper-IP / per-user"]
    RL -->|"429 BILKO-9005"| R429["429 Too Many Requests"]
    RL --> LOGGER["Morgan Logger\nHTTP access log"]
    LOGGER --> AUTH_GUARD["authGuard\nExtract Bearer token"]

    AUTH_GUARD -->|"No Authorization header"| R401A["401 BILKO-1005\nNo token"]
    AUTH_GUARD -->|"Token expired"| R401B["401 BILKO-1003\nToken expired"]
    AUTH_GUARD -->|"Invalid signature"| R401C["401 BILKO-1004\nInvalid token"]
    AUTH_GUARD -->|"Valid JWT"| EXTRACT["Extract claims\nsub, email, role, orgId"]

    EXTRACT --> ROLE_GUARD["roleGuard(allowedRoles)\nCheck role membership"]
    ROLE_GUARD -->|"Role not in allowedRoles"| R403["403 BILKO-9001\nInsufficient permissions"]
    ROLE_GUARD -->|"Role authorized"| ORG_SCOPE["organizationScope\nAttach req.organizationId"]

    ORG_SCOPE --> VALIDATE["Zod Validation\nschema.parse(req.body)"]
    VALIDATE -->|"Schema errors"| R422["422 BILKO-9003\nValidation failed"]
    VALIDATE -->|"Valid body"| HANDLER["Route Handler\n(module controller)"]
```

```
    HANDLER --> DB_QUERY["Prisma Query\nWHERE organizationId = req.organizationId"]
    DB_QUERY -->|"Record not in org"| R404["404 BILKO-Xxxx\nNot found"]
    DB_QUERY -->|"Business rule violation"| R400["400 BILKO-Xxxx\nBad request"]
    DB_QUERY -->|"DB error"| R500["500 BILKO-9006\nDatabase error"]
    DB_QUERY -->|"Success"| AUDIT["LoggedAction INSERT\nAppend-only audit trail"]
    AUDIT --> RESPONSE["200/201/204\nSuccess Response"]

    style R401A fill:#dc2626,color:#fff
    style R401B fill:#dc2626,color:#fff
    style R401C fill:#dc2626,color:#fff
    style R403 fill:#ea580c,color:#fff
    style R404 fill:#ca8a04,color:#fff
    style R400 fill:#ca8a04,color:#fff
    style R422 fill:#ca8a04,color:#fff
    style R429 fill:#ca8a04,color:#fff
    style R500 fill:#7c3aed,color:#fff
    style RESPONSE fill:#16a34a,color:#fff
    style DB_QUERY fill:#336791,color:#fff
    style AUDIT fill:#dc2626,color:#fff
```

Role Assignment Diagram

```
flowchart LR
    subgraph "Registration"
        REG["POST /auth/register"] --> OWNER["owner\n(auto-assigned)"]
    end

    subgraph "Invitation — by owner or admin"
        INVITE["POST /users/invite\nrole: admin | accountant | viewer"] --> PENDING["Pending
User\n(isActive = false)"]
        PENDING -->|"Accepts invite within 7 days"| ACTIVE["Active User\n(assigned role)"]
        PENDING -->|"Invite expires"| EXPIRED["BILKO-1012\nInvalid invite"]
    end

    subgraph "Role Changes — by owner only"
        OWNER -->|"PUT /users/:id/role"| CHANGE["Role updated\nEffective on next JWT refresh"]
    end
```

```
    subgraph "User Removal — by owner only"
        OWNER -->|"DELETE /users/:id"| SOFT_DEL["isActive = false\nRefresh tokens
invalidated"]
    end

    style OWNER fill:#00E5A0,color:#000
    style ACTIVE fill:#16a34a,color:#fff
    style SOFT_DEL fill:#dc2626,color:#fff
    style EXPIRED fill:#ca8a04,color:#fff
```

Middleware Implementation Reference

```
// src/middleware/auth.middleware.ts

type UserRole = 'owner' | 'admin' | 'accountant' | 'viewer'

// Step 1: Verify JWT and attach user to request
async function authGuard(req: AuthRequest, res: Response, next: NextFunction) {
  const authHeader = req.headers.authorization

  if (!authHeader?.startsWith('Bearer ')) {
    return res.status(401).json({ error: { code: 'BILKO-1005', message: 'Authentication is
required.' } })
  }

  const token = authHeader.substring(7)

  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET!) as AccessTokenPayload
    req.user = {
      id: payload.sub,
      email: payload.email,
      role: payload.role,
      organizationId: payload.orgId,
    }
    next()
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
```

```
      return res.status(401).json({ error: { code: 'BILKO-1003', message: 'Session expired.
Please refresh your token.' } })
    }
    return res.status(401).json({ error: { code: 'BILKO-1004', message: 'Invalid token. Please
log in again.' } })
  }
}

// Step 2: Check role authorization
function roleGuard(allowedRoles: UserRole[]) {
  return (req: AuthRequest, res: Response, next: NextFunction) => {
    if (!req.user) {
      return res.status(401).json({ error: { code: 'BILKO-1005', message: 'Authentication is
required.' } })
    }

    if (!allowedRoles.includes(req.user.role)) {
      return res.status(403).json({
        error: {
          code: 'BILKO-9001',
          message: 'You do not have permission to perform this action.',
          details: {
            required: allowedRoles,
            current: [req.user.role],
          },
        },
      })
    }

    next()
  }
}

// Step 3: Apply organization scope
function organizationScope(req: AuthRequest, res: Response, next: NextFunction) {
  req.organizationId = req.user!.organizationId
  next()
}

// Convenience: all authenticated roles
```

```
const allRoles: UserRole[] = ['owner', 'admin', 'accountant', 'viewer']

// Usage in routes:
router.post('/invoices',
  authGuard,
  roleGuard(['owner', 'admin', 'accountant']),
  organizationScope,
  validate(CreateInvoiceSchema),
  invoicesController.create
)

router.get('/invoices',
  authGuard,
  roleGuard(allRoles),
  organizationScope,
  invoicesController.list
)

router.patch('/expenses/:id/approve',
  authGuard,
  roleGuard(['owner', 'admin']),
  organizationScope,
  expensesController.approve
)
```

Summary Table

Capability	owner	admin	accountant	viewer
Invoices				
View invoices	✅	✅	✅	✅
Create / edit invoice	✅	✅	✅	❌
Send invoice	✅	✅	✅	❌
Mark invoice paid / cancel	✅	✅	✅	❌
Expenses				
View expenses	✅	✅	✅	✅

Capability	owner	admin	accountant	viewer
Create / edit expense	✓	✓	✓	✗
Approve expense	✓	✓	✗	✗
Delete expense	✓	✓	✗	✗
Contacts				
View contacts	✓	✓	✓	✓
Create / edit contact	✓	✓	✓	✗
Deactivate contact	✓	✓	✗	✗
Banking				
View bank accounts & transactions	✓	✓	✓	✓
Create bank account	✓	✓	✗	✗
Import CSV / reconcile	✓	✓	✓	✗
GL Transactions				
View transactions	✓	✓	✓	✓
Create manual journal entry	✓	✓	✓	✗
Reports				
View all reports	✓	✓	✓	✓
Chart of Accounts				
View accounts	✓	✓	✓	✓
Create / edit / deactivate accounts	✓	✓	✗	✗
Settings				
View tax rates	✓	✓	✓	✓
Update tax rates	✓	✓	✗	✗
Update org details	✓	✓	✗	✗
Users				
View user list	✓	✓	✗	✗
Invite user	✓	✓	✗	✗
Change user role	✓	✗	✗	✗
Remove user	✓	✗	✗	✗
Organization				

Capability	owner	admin	accountant	viewer
Delete organization	☐	☐	☐	☐

End of Roles and Permissions Documentation

Revision #3
Created 2026-02-24 23:10:43 UTC by John
Updated 2026-05-31 20:04:05 UTC by John