# RBAC (Role-Based Access Control) Implementation

Comprehensive guide to the Role-Based Access Control system implemented in the RAS Dashboard API, including permissions, roles, middleware, and best practices.

## 🎯 Overview

The RBAC system provides fine-grained access control through: - **Permissions** - Specific actions users can perform - **Roles** - Collections of permissions - **User-Role Assignments** - Users can have multiple roles - **Middleware** - Automatic permission checking - **Caching** - Performance optimization

## 🏗️ Database Schema

### Core Tables

-- Permissions: Define what actions can be performed  
permissions (id, name, description, category, created\_at, updated\_at)  
  
-- Roles: Group permissions together  
roles (id, name, description, is\_system, is\_default, created\_at, updated\_at)  
  
-- Role-Permission Assignments: Many-to-many  
role\_permissions (id, role\_id, permission\_id)  
  
-- User-Role Assignments: Many-to-many  
user\_roles (id, user\_id, role\_id, assigned\_at, assigned\_by, created\_at, updated\_at)  
  
-- Users: Extended with role enum for backward compatibility  
users (id, username, email, role, status, ...)

### Relationships

Users ←→ UserRoles ←→ Roles ←→ RolePermissions ←→ Permissions

## 🔐 Permission System

### Permission Naming Convention

<resource>:<action>

**Examples:** - users:read - View user information - users:write - Create and update users - users:delete - Delete users - admin:dashboard - Access admin dashboard - reports:read - View reports

### Default Permissions

// Users Category  
users:read, users:write, users:delete  
  
// Roles Category   
roles:read, roles:write, roles:delete  
  
// Permissions Category  
permissions:read, permissions:write  
  
// Admin Category  
admin:dashboard  
  
// System Category  
system:manage  
  
// Reports Category  
reports:read, reports:write

### Permission Categories

Permissions are organized by category for better management: - users - User management - roles - Role management - permissions - Permission management - admin - Administrative functions - system - System-level operations - reports - Reporting and analytics

## 👥 Role System

### Default Roles

#### Admin Role

{  
 name: 'admin',  
 description: 'Full system access',  
 isSystem: true,  
 isDefault: false,  
 permissions: ['\*'] // All permissions  
}

#### User Role

{  
 name: 'user',   
 description: 'Basic user access',  
 isSystem: true,  
 isDefault: true,  
 permissions: ['users:read']  
}

#### Moderator Role

{  
 name: 'moderator',  
 description: 'Moderate content and users',   
 isSystem: false,  
 isDefault: false,  
 permissions: ['users:read', 'users:write', 'roles:read', 'reports:read']  
}

#### Viewer Role

{  
 name: 'viewer',  
 description: 'Read-only access',  
 isSystem: false,   
 isDefault: false,  
 permissions: ['users:read', 'reports:read']  
}

### Role Properties

* **isSystem** - System roles cannot be deleted
* **isDefault** - Default role assigned to new users
* **permissions** - Array of permission names or ’\*’ for all

## 🛡️ Middleware Implementation

### Basic Permission Check

const { requirePermission } = require('../middleware/rbac');  
  
// Require specific permission  
router.get('/admin', requirePermission('admin:dashboard'), controller.adminDashboard);

### Multiple Permission Options

const { requireAnyPermission } = require('../middleware/rbac');  
  
// User needs ANY of these permissions  
router.get('/users', requireAnyPermission(['users:read', 'admin:dashboard']), controller.getUsers);

### All Permissions Required

const { requireAllPermissions } = require('../middleware/rbac');  
  
// User needs ALL of these permissions  
router.post('/admin/users', requireAllPermissions(['users:write', 'admin:dashboard']), controller.createUser);

### Ownership or Permission

const { requireOwnershipOrPermission } = require('../middleware/rbac');  
  
// User can access their own data OR have admin permission  
router.put('/users/:id',   
 requireOwnershipOrPermission('users:write', req => parseInt(req.params.id)),   
 controller.updateUser  
);

## 🚀 Usage Examples

### Protecting Routes

const express = require('express');  
const { authenticateToken } = require('../middleware/auth');  
const { requirePermission, requireAnyPermission } = require('../middleware/rbac');  
  
const router = express.Router();  
  
// All routes require authentication  
router.use(authenticateToken);  
  
// Admin only  
router.get('/admin/dashboard',   
 requirePermission('admin:dashboard'),   
 controller.adminDashboard  
);  
  
// Users with read permission  
router.get('/users',   
 requirePermission('users:read'),   
 controller.getUsers  
);  
  
// Multiple permission options  
router.get('/reports',   
 requireAnyPermission(['reports:read', 'admin:dashboard']),   
 controller.getReports  
);  
  
// Ownership or admin  
router.put('/users/:id',   
 requireOwnershipOrPermission('users:write', req => parseInt(req.params.id)),  
 controller.updateUser  
);

### Checking Permissions in Controllers

const { getUserPermissions } = require('../middleware/rbac');  
  
const controller = {  
 async getUsers(req, res) {  
 // Get user's permissions  
 const permissions = await getUserPermissions(req.user.id);  
   
 // Conditional logic based on permissions  
 if (permissions.has('admin:dashboard')) {  
 // Admin can see all user data  
 users = await userService.getAllUsers({ includeInactive: true });  
 } else {  
 // Regular users see limited data  
 users = await userService.getAllUsers({ includeInactive: false });  
 }  
   
 res.json({ users });  
 }  
};

## ⚡ Performance Optimization

### Permission Caching

// Permissions are cached for 5 minutes per user  
const CACHE\_TTL = 5 \* 60 \* 1000; // 5 minutes  
  
// Cache is automatically managed  
const permissions = await getUserPermissions(userId); // Cached result

### Cache Management

const { clearUserPermissionCache, clearAllPermissionCache } = require('../middleware/rbac');  
  
// Clear cache when user roles change  
await assignRolesToUser(userId, newRoleIds);  
clearUserPermissionCache(userId);  
  
// Clear all cache when permissions/roles change  
await updateRolePermissions(roleId, newPermissionIds);  
clearAllPermissionCache();

## 🛠️ Management Operations

### Role Management

const rbacService = require('../services/rbacService');  
  
// Create new role  
const newRole = await rbacService.createRole({  
 name: 'editor',  
 description: 'Content editor',  
 isSystem: false,  
 isDefault: false  
});  
  
// Assign permissions to role  
await rbacService.assignPermissionsToRole(newRole.id, [  
 permissionIds.find(p => p.name === 'users:read').id,  
 permissionIds.find(p => p.name === 'reports:write').id  
]);

### User Role Assignment

// Assign roles to user  
await rbacService.assignRolesToUser(userId, [adminRoleId, moderatorRoleId]);  
  
// Get user's roles  
const userRoles = await rbacService.getUserRoles(userId);  
  
// Remove specific role from user  
await rbacService.removeRoleFromUser(userId, roleId);

### Permission Management

// Create new permission  
const newPermission = await rbacService.createPermission({  
 name: 'analytics:read',  
 category: 'analytics',   
 description: 'View analytics data'  
});  
  
// Get role permissions  
const rolePermissions = await rbacService.getRolePermissions(roleId);

## 🔍 Querying RBAC Data

### Check User Permissions

# See user-role assignments  
npm run db:query user-roles  
  
# See role-permission assignments   
npm run db:query role-permissions  
  
# See all permissions by category  
npm run db:query permissions:by-category

### Database Queries

-- Get all permissions for a user  
SELECT DISTINCT p.name, p.category, p.description  
FROM users u  
JOIN user\_roles ur ON u.id = ur.user\_id  
JOIN roles r ON ur.role\_id = r.id   
JOIN role\_permissions rp ON r.id = rp.role\_id  
JOIN permissions p ON rp.permission\_id = p.id  
WHERE u.id = $1;  
  
-- Get users with specific permission  
SELECT DISTINCT u.username, u.email  
FROM users u  
JOIN user\_roles ur ON u.id = ur.user\_id  
JOIN roles r ON ur.role\_id = r.id  
JOIN role\_permissions rp ON r.id = rp.role\_id   
JOIN permissions p ON rp.permission\_id = p.id  
WHERE p.name = 'admin:dashboard';

## 🎯 Best Practices

### 1. Principle of Least Privilege

// Give users minimum permissions needed  
const basicUserPermissions = ['users:read'];  
const moderatorPermissions = ['users:read', 'users:write', 'reports:read'];  
const adminPermissions = ['\*']; // All permissions

### 2. Use Descriptive Permission Names

// Good  
'users:read', 'reports:export', 'admin:dashboard'  
  
// Avoid  
'read', 'write', 'admin'

### 3. Organize by Categories

// Group related permissions  
const userPermissions = ['users:read', 'users:write', 'users:delete'];  
const reportPermissions = ['reports:read', 'reports:write', 'reports:export'];

### 4. Handle Permission Changes Gracefully

// Always clear cache when permissions change  
await updateUserRoles(userId, newRoles);  
clearUserPermissionCache(userId);  
  
// Check permissions exist before using  
if (permissions.has('new:feature')) {  
 // Feature is available  
}

### 5. Audit Permission Usage

// Log permission checks for security auditing  
const hasPermission = permissions.has('sensitive:operation');  
if (hasPermission) {  
 logger.info(`User ${userId} accessed sensitive operation`);  
}

## 🔧 Troubleshooting

### Permission Denied Issues

// Debug user permissions  
const permissions = await getUserPermissions(userId);  
console.log('User permissions:', Array.from(permissions));  
  
// Check role assignments  
const roles = await rbacService.getUserRoles(userId);  
console.log('User roles:', roles);

### Cache Issues

// Clear cache if permissions seem stale  
clearUserPermissionCache(userId);  
// or  
clearAllPermissionCache();

### Performance Issues

// Check cache hit rates  
// Consider increasing cache TTL for stable permissions  
// Use database indexes on foreign keys

## 🚀 Advanced Features

### Dynamic Permissions

// Generate permissions based on context  
const dynamicPermission = `project:${projectId}:read`;  
if (permissions.has(dynamicPermission)) {  
 // User can read this specific project  
}

### Hierarchical Roles

// Implement role inheritance  
const roleHierarchy = {  
 'admin': ['moderator', 'user'],  
 'moderator': ['user'],  
 'user': []  
};

### Conditional Permissions

// Time-based or context-based permissions  
const isBusinessHours = new Date().getHours() >= 9 && new Date().getHours() <= 17;  
if (permissions.has('admin:dashboard') && isBusinessHours) {  
 // Admin access during business hours only  
}

This RBAC system provides a robust, scalable foundation for managing access control in your application while maintaining performance and security.