Application-Specific Role-Based Access Control (RBAC) Model

Table of Contents

Overview

Purpose

This RBAC model provides a flexible, application-specific permission management system designed to integrate seamlessly with external authentication systems (SSO, SAML, OAuth, LDAP). It eliminates the need for duplicate user management while maintaining granular access control.

Key Features

- **Application-Specific Roles**: Each application maintains its own role definitions
- **External User Management**: No user storage - integrates with existing authentication systems
- **Hierarchical Permissions**: Support for role inheritance and resource hierarchies
- **Fine-Grained Access Control**: Resource-level permissions with multiple action types
- **Audit Trail**: Complete change tracking for compliance requirements
- **Performance Optimized**: Indexed for high-performance permission checks

Design Principles

- **Separation of Concerns**: Authentication handled externally, authorization handled internally
- **Single Responsibility**: Each application manages only its own permissions
- **Scalability**: Designed to support multiple applications and thousands of roles
- **Flexibility**: Supports complex permission scenarios and role hierarchies

- **Security**: Built-in audit trails and access control mechanisms

Architecture

High-Level Architecture

```
[External Auth System] → [Your Application] → [RBAC Database]
     (User Identity)      (Role Assignment)    (Permissions)
```

Data Flow

1. **Authentication**: User authenticates via external system (SSO/SAML/OAuth)
2. **Role Assignment**: External system provides user's assigned role codes
3. **Permission Check**: Application queries RBAC system for specific permissions
4. **Access Decision**: System returns allow/deny based on role permissions
5. **Audit Logging**: All permission checks and changes are logged

Integration Points

- **Authentication Layer**: Handles user login and identity verification
- **Authorization Layer**: Manages role assignments and permission mappings
- **Application Layer**: Enforces permissions based on RBAC decisions
- **Audit Layer**: Records all security-related activities

Database Schema

Core Tables

Applications

Defines separate permission universes for different systems.

```sql
applications (
    app_id, app_code, app_name, app_description,
    is_active, created_at, updated_at
)
```

Modules

Logical groupings of functionality within applications.

```sql
modules (
    module_id, app_id, module_code, module_name,
    parent_module_id, is_active, created_at, updated_at
)
```

Resources

Specific items that can be accessed (pages, APIs, data, etc.).

```sql
resources (
    resource_id, module_id, resource_code, resource_name,
    resource_type, resource_path, is_active, created_at, updated_at
)
```

Permissions

Actions that can be performed on resources.

```sql
permissions (
    permission_id, permission_code, permission_name,
    permission_description, is_active, created_at, updated_at
)
```

Roles

Application-specific roles with hierarchical support.

```sql
roles (
    role_id, app_id, role_code, role_name,
    role_level, parent_role_id, is_active, created_at, updated_at
)
```

Key Relationships

- **Applications** → **Modules** (1:N)
- **Modules** → **Resources** (1:N)
- **Resources** ↔ **Permissions** (N:N via resource_permissions)
- **Roles** ↔ **Resource+Permissions** (N:N via role_permissions)

- **Roles** → **Roles** (hierarchical via parent_role_id)

Key Concepts

Application Isolation

Each application maintains completely separate:

- Role definitions
- Permission structures
- Resource hierarchies
- Access control policies

Role Hierarchy

Roles can inherit permissions from parent roles:

```
CFO (Level 4)
└── Finance Manager (Level 3)
    └── Senior Accountant (Level 2)
        └── Accountant (Level 1)
```

Permission Types

Standard permission actions include:

- **READ**: View/access resources
- **WRITE**: Create/modify resources
- **DELETE**: Remove resources
- **EXECUTE**: Perform operations
- **APPROVE**: Approval workflows
- **ADMIN**: Administrative functions

Resource Types

Resources can be categorized as:

- **PAGE**: Web pages/screens
- **API**: REST endpoints/services
- **SERVICE**: Business services
- **MENU**: Navigation items
- **BUTTON**: UI elements
- **DATA**: Data entities

Access Control Models

- **ALLOW**: Explicitly grant permission (default)
- **DENY**: Explicitly deny permission (overrides allows)

Implementation Guide

Phase 1: Schema Setup

1. Create database tables using provided SQL schema
2. Insert standard permissions (READ, WRITE, DELETE, etc.)
3. Set up audit logging triggers
4. Create performance indexes

Phase 2: Application Definition

1. Define your applications in the `applications` table
2. Create modules for logical functionality groups
3. Map all resources (pages, APIs) to appropriate modules
4. Link resources to applicable permissions

Phase 3: Role Design

1. Analyze business requirements for role definitions
2. Create role hierarchy based on organizational structure
3. Assign permissions to roles based on job functions
4. Test role inheritance and permission propagation

Phase 4: Integration

1. Configure SSO/external authentication system
2. Implement role assignment mechanism
3. Create permission checking service/API
4. Integrate with application security layer

Phase 5: Testing & Validation

1. Test all role combinations and scenarios
2. Validate permission inheritance
3. Verify audit logging functionality
4. Performance test permission checks

Integration with SSO Systems

SAML Integration

```xml
<!-- Sample SAML Assertion with Roles -->
<saml:AttributeStatement>
    <saml:Attribute Name="roles">
        <saml:AttributeValue>PAYROLL_MANAGER</saml:AttributeValue>
        <saml:AttributeValue>HR_OFFICER</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute Name="applications">
        <saml:AttributeValue>PAYROLL_APP</saml:AttributeValue>
        <saml:AttributeValue>HR_APP</saml:AttributeValue>
    </saml:Attribute>
</saml:AttributeStatement>
```

OAuth/OIDC Integration

```json
{
  "sub": "user123",
  "email": "user@company.com",
  "roles": ["PAYROLL_MANAGER", "HR_OFFICER"],
  "applications": ["PAYROLL_APP", "HR_APP"]
}
```

LDAP Integration

```
# Sample LDAP attributes
memberOf: CN=PayrollManagers,OU=Roles,DC=company,DC=com
memberOf: CN=HROfficers,OU=Roles,DC=company,DC=com
```

Role Mapping Strategy

1. **Direct Mapping**: External role names match internal role codes
2. **Attribute Mapping**: Map external attributes to internal roles
3. **Group Mapping**: Map external groups to internal role groups
4. **Custom Mapping**: Use transformation logic for complex scenarios

API Design Patterns

Permission Check API

```http
POST /api/rbac/check-permission
Content-Type: application/json

{
  "user_roles": ["PAYROLL_MANAGER"],
  "app_code": "PAYROLL_APP",
  "resource_code": "PAYROLL_CHECKER",
  "permission_code": "APPROVE"
}

Response:
{
  "allowed": true,
  "role_matched": "PAYROLL_MANAGER",
  "reason": "Role has explicit APPROVE permission on resource"
}
```

Bulk Permission Check API

```http
POST /api/rbac/check-permissions-bulk
Content-Type: application/json

{
  "user_roles": ["PAYROLL_MANAGER"],
  "app_code": "PAYROLL_APP",
  "checks": [
    {"resource_code": "PAYROLL_MAKER", "permission_code": "READ"},
    {"resource_code": "PAYROLL_CHECKER", "permission_code": "APPROVE"},
    {"resource_code": "ERROR_CORRECTION", "permission_code": "WRITE"}
  ]
}
```

User Menu API

```http
GET /api/rbac/user-menu?roles=PAYROLL_MANAGER&app=PAYROLL_APP

Response:
{
  "menu_items": [
    {
      "resource_code": "PAYROLL_DASHBOARD",
      "resource_name": "Dashboard",
      "resource_path": "/payroll/dashboard",
      "permissions": ["READ"]
    }
  ]
}
```

Security Considerations

Input Validation

- Validate all role codes against known roles
- Sanitize resource and permission codes
- Implement rate limiting on permission checks
- Log suspicious permission check patterns

Access Control

- Implement least privilege principle
- Regular role and permission audits
- Automatic role expiration/review processes
- Separation of duties for sensitive operations

Data Protection

- Encrypt sensitive configuration data
- Secure database connections
- Implement database-level access controls
- Regular security patches and updates

Audit Requirements

- Log all permission changes
- Track permission check patterns

- Maintain immutable audit trails
- Regular compliance reporting

Performance Optimization

Database Optimization

```sql
-- Key performance indexes
CREATE INDEX idx_role_permissions_lookup ON role_permissions(role_id, resource_id, permission_i
CREATE INDEX idx_user_permission_check ON roles(app_id, role_code, is_active);
CREATE INDEX idx_resource_lookup ON resources(module_id, resource_code, is_active);
```

Caching Strategy

1. **Role Permissions Cache**: Cache complete role permission sets
2. **User Session Cache**: Cache user's effective permissions during session
3. **Application Cache**: Cache application structure (modules, resources)
4. **Negative Cache**: Cache denied permissions to avoid repeated checks

Query Optimization

- Use prepared statements for repeated queries
- Implement connection pooling
- Optimize JOIN operations with proper indexing
- Use materialized views for complex permission lookups

Monitoring and Metrics

- Track permission check response times
- Monitor cache hit ratios
- Alert on unusual permission patterns
- Database performance monitoring

Sample Queries

Basic Permission Check

sql

```sql
-- Check if role has specific permission on resource
SELECT COUNT(*) > 0 as has_permission
FROM role_permissions_view
WHERE role_code = 'PAYROLL_MANAGER'
AND app_code = 'PAYROLL_APP'
AND resource_code = 'PAYROLL_CHECKER'
AND permission_code = 'APPROVE'
AND access_type = 'ALLOW';
```

Get User's Accessible Resources

sql

```sql
-- Get all resources user can access with their permissions
SELECT DISTINCT resource_code, resource_name, resource_path, permission_code
FROM role_permissions_view
WHERE role_code IN ('PAYROLL_MANAGER', 'HR_OFFICER')
AND app_code = 'PAYROLL_APP'
AND access_type = 'ALLOW'
ORDER BY resource_name, permission_code;
```

Role Hierarchy Query

sql

```sql
-- Get complete role hierarchy for an application
SELECT * FROM app_role_hierarchy
WHERE app_code = 'PAYROLL_APP'
ORDER BY depth, role_level;
```

Administrative Queries

```sql
-- Find all roles with admin permissions
SELECT DISTINCT r.role_code, r.role_name, a.app_name
FROM roles r
JOIN applications a ON r.app_id = a.app_id
JOIN role_permissions rp ON r.role_id = rp.role_id
JOIN permissions p ON rp.permission_id = p.permission_id
WHERE p.permission_code = 'ADMIN'
AND r.is_active = TRUE;

-- Audit: Find resources without any role assignments
SELECT r.resource_code, r.resource_name, a.app_name
FROM resources r
JOIN modules m ON r.module_id = m.module_id
JOIN applications a ON m.app_id = a.app_id
LEFT JOIN role_permissions rp ON r.resource_id = rp.resource_id
WHERE rp.resource_id IS NULL
AND r.is_active = TRUE;
```

Maintenance and Administration

Regular Maintenance Tasks

Weekly Tasks

- Review audit logs for suspicious activities
- Check for unused roles or permissions
- Validate role assignments against business requirements
- Monitor system performance metrics

Monthly Tasks

- Role and permission cleanup
- Update role descriptions and documentation
- Review and update role hierarchies
- Performance tuning and optimization

Quarterly Tasks

- Complete security audit of roles and permissions
- Review and update access control policies
- Validate compliance with regulatory requirements
- Disaster recovery testing

Administrative Procedures

Adding New Applications

1. Create application record in `applications` table
2. Define modules and resources structure
3. Create application-specific roles
4. Map resources to permissions
5. Assign permissions to roles
6. Test complete permission structure
7. Update documentation

Role Management

1. **Creating Roles**: Define clear role purpose and scope
2. **Permission Assignment**: Follow least privilege principle
3. **Role Hierarchy**: Maintain logical inheritance structure
4. **Role Retirement**: Disable rather than delete for audit trail

Emergency Procedures

1. **Access Revocation**: Immediate role deactivation procedures
2. **Permission Escalation**: Temporary elevated access procedures
3. **System Lockdown**: Emergency security measures
4. **Incident Response**: Security breach response procedures

Monitoring and Alerting

Key Metrics to Monitor

- Permission check response times
- Failed permission check rates
- Role modification frequency
- Unusual access patterns
- Database performance metrics

Alert Conditions

- Multiple failed permission checks from same user
- Administrative role assignments
- Bulk permission changes

- System performance degradation
- Audit log anomalies

Backup and Recovery

Backup Strategy

- Daily automated backups of RBAC database
- Weekly full system backups
- Monthly backup validation testing
- Offsite backup storage for disaster recovery

Recovery Procedures

- Point-in-time recovery capabilities
- Role and permission restoration procedures
- Audit trail preservation during recovery
- Testing and validation after recovery

Conclusion

This RBAC model provides a robust, scalable solution for application-specific access control while maintaining integration flexibility with external authentication systems. The design emphasizes security, performance, and maintainability while supporting complex organizational permission structures.

For successful implementation, focus on:

1. Clear role definitions aligned with business requirements
2. Proper integration with existing authentication infrastructure
3. Comprehensive testing of all permission scenarios
4. Regular monitoring and maintenance procedures
5. Strong audit and compliance capabilities

The system is designed to grow with your organization and can be extended to support additional applications and more complex permission scenarios as needed.