GRC RBAC Implementation Guide - Detailed Integration

Step 1: Create RBAC Module Structure

1.1 Follow this GRC RBAC Directory Structure

```
backend/grc/
├── rbac/                    # NEW: Create this directory
│   ├── __init__.py          # NEW
│   ├── config.py            # NEW: RBAC configuration
│   ├── utils.py             # NEW: RBAC utilities
│   ├── permissions.py       # NEW: DRF permission classes
│   ├── middleware.py        # NEW: RBAC middleware
│   ├── decorators.py        # NEW: RBAC decorators
│   └── views.py             # NEW: RBAC helper views
└── (existing files...)
```

Step 2: Update Django Settings

2.1 Update **backend/backend/settings.py**

```
# Add RBAC middleware to MIDDLEWARE (add near the end)
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'grc.rbac.middleware.GRCRBACMiddleware',  # ADD THIS LINE
]

# Add RBAC configuration
RBAC_CONFIG = {
    'ENABLE_RBAC': True,
    'STRICT_MODE': True,
    'LOG_ACCESS_DENIED': True,
    'EXEMPT_URLS': [
        r'^/admin/',
        r'^/api/auth/',
        r'^/login/',
```

```
        r'^/logout/',
        r'^/register/',
        r'^/test-connection/',
        r'^/test-notification/',
    ]
}
```

Step 3: Update URL Configuration

3.1 Update backend/grc/urls.py

```
# Add RBAC endpoints to your existing urlpatterns
from grc.rbac.views import get_user_permissions, get_user_role

# Find your existing urlpatterns and add these lines:
urlpatterns = [
    # ... your existing patterns ...

    # RBAC endpoints (ADD THESE)
    path('api/user-permissions/', get_user_permissions, name='user-permissions'),
    path('api/user-role/', get_user_role, name='user-role'),

    # ... rest of your patterns ...
]
```

3.2 Update Route Files (Optional - For Granular Control)

For backend/grc/routes/policy.py:

```
# Add imports at the top
from grc.rbac.permissions import (
    PolicyCreatePermission,
    PolicyEditPermission,
    PolicyApprovePermission,
    PolicyViewPermission
)
from rest_framework.permissions import IsAuthenticated

# Update existing views to add permission classes
# Example for policy creation:
@api_view(['POST'])
@permission_classes([IsAuthenticated, PolicyCreatePermission])  # ADD THIS LINE
def create_policy(request):
    # Your existing code remains unchanged
    pass

# Example for policy approval:
@api_view(['PUT'])
```

```
@permission_classes([IsAuthenticated, PolicyApprovePermission])  # ADD THIS LINE
def approve_policy(request, policy_id):
    # Your existing code remains unchanged
    pass
```

For backend/grc/compliance_views.py:

```
# Add imports at the top
from grc.rbac.permissions import (
    ComplianceCreatePermission,
    ComplianceEditPermission,
    ComplianceApprovePermission
)

# Update existing views:
@api_view(['POST'])
@permission_classes([IsAuthenticated, ComplianceCreatePermission])  # ADD THIS
def create_compliance(request):
    # Your existing code unchanged
    pass
```

For backend/grc/audit_views.py:

```
# Add imports
from grc.rbac.permissions import (
    AuditAssignPermission,
    AuditConductPermission,
    AuditReviewPermission
)

# Update views:
@api_view(['POST'])
@permission_classes([IsAuthenticated, AuditAssignPermission])  # ADD THIS
def assign_audit(request):
    # Your existing code unchanged
    pass
```

Step 4: Database Updates

4.1 Update RBAC Table Data

```
-- Update existing role values to match enum values
-- Run these SQL commands on your database:

UPDATE rbac SET Role = 'GRC Administrator' WHERE Role IN ('admin', 'grc_admin', 'administrator');
UPDATE rbac SET Role = 'Policy Manager' WHERE Role IN ('policy_manager', 'policy_mgr');
UPDATE rbac SET Role = 'Policy Approver' WHERE Role IN ('policy_approver', 'policy_reviewer');
UPDATE rbac SET Role = 'Compliance Manager' WHERE Role IN ('compliance_manager', 'compliance_mgr');
```

```
UPDATE rbac SET Role = 'Compliance Officer' WHERE Role IN ('compliance_officer', 'compliance_user');
UPDATE rbac SET Role = 'Compliance Approver' WHERE Role IN ('compliance_approver',
'compliance_reviewer');
UPDATE rbac SET Role = 'Audit Manager' WHERE Role IN ('audit_manager', 'audit_mgr');
UPDATE rbac SET Role = 'Internal Auditor' WHERE Role IN ('internal_auditor', 'auditor');
UPDATE rbac SET Role = 'External Auditor' WHERE Role IN ('external_auditor', 'ext_auditor');
UPDATE rbac SET Role = 'Audit Reviewer' WHERE Role IN ('audit_reviewer', 'audit_approver');
UPDATE rbac SET Role = 'Risk Manager' WHERE Role IN ('risk_manager', 'risk_mgr');
UPDATE rbac SET Role = 'Risk Analyst' WHERE Role IN ('risk_analyst', 'risk_user');
UPDATE rbac SET Role = 'Risk Reviewer' WHERE Role IN ('risk_reviewer', 'risk_approver');
UPDATE rbac SET Role = 'Incident Response Manager' WHERE Role IN ('incident_manager', 'incident_mgr');
UPDATE rbac SET Role = 'Incident Analyst' WHERE Role IN ('incident_analyst', 'incident_user');
UPDATE rbac SET Role = 'Department Manager' WHERE Role IN ('dept_manager', 'department_manager');
UPDATE rbac SET Role = 'End User' WHERE Role IN ('user', 'end_user', 'employee');

-- Add indexes for performance
CREATE INDEX idx_rbac_userid ON rbac(UserId);
CREATE INDEX idx_rbac_role ON rbac(Role);
CREATE INDEX idx_rbac_department ON rbac(Department);
```

Step 5: Minimal View Updates (Gradual Implementation)

5.1 High-Priority Endpoints to Update First

File: backend/grc/views.py (your main views file)

```
# Add these imports at the top
from rest_framework.permissions import IsAuthenticated
from grc.rbac.permissions import (
    PolicyCreatePermission,
    PolicyApprovePermission,
    ComplianceCreatePermission,
    AuditAssignPermission
)

# Example: Update critical policy creation view
@api_view(['POST'])
@permission_classes([IsAuthenticated, PolicyCreatePermission])  # ADD THIS LINE
def create_policy_view(request):
    # Your existing code remains exactly the same
    pass

# Example: Update policy approval view
@api_view(['PUT'])
@permission_classes([IsAuthenticated, PolicyApprovePermission])  # ADD THIS LINE
def approve_policy_view(request, policy_id):
    # Your existing code remains exactly the same
    pass
```

5.2 Update KPI Views (Based on Your Sample)

File: backend/grc/views.py or wherever your KPI views are:

```
# Add analytics permission import
from grc.rbac.permissions import PolicyViewPermission  # For analytics views

# Update your existing KPI views:
@api_view(['GET'])
@permission_classes([IsAuthenticated, PolicyViewPermission])  # ADD THIS LINE
def audit_completion(request):
    """Get audit completion statistics"""
    time_filter = request.GET.get('time_filter', 'month')
    start_date = request.GET.get('start_date')
    end_date = request.GET.get('end_date')

    data = get_audit_completion_stats(time_filter, start_date, end_date)
    return Response(data)

@api_view(['GET'])
@permission_classes([IsAuthenticated, PolicyViewPermission])  # ADD THIS LINE
def audit_cycle_time(request):
    """Get average audit cycle time"""
    time_filter = request.GET.get('time_filter', 'month')
    start_date = request.GET.get('start_date')
    end_date = request.GET.get('end_date')

    data = get_audit_cycle_time(time_filter, start_date, end_date)
    return Response(data)
```

Step 6: Frontend Integration

6.1 Update frontend/src/services/api.js

```
// Add RBAC service functions to your existing API service

// Add this to your existing api.js file:
export const rbacService = {
  // Get user permissions
  async getUserPermissions() {
    try {
      const response = await axios.get('/api/user-permissions/');
      return response.data;
    } catch (error) {
      console.error('Error fetching user permissions:', error);
      throw error;
    }
  },
```

```
 // Get user role
 async getUserRole() {
  try {
   const response = await axios.get('/api/user-role/');
   return response.data;
  } catch (error) {
   console.error('Error fetching user role:', error);
   throw error;
  }
 }
};
```

6.2 Create Permission Mixin

File: frontend/src/mixins/permissionMixin.js (NEW FILE)

```
export const permissionMixin = {
 data() {
  return {
   userPermissions: {},
   userRole: null,
   permissionsLoaded: false
  }
 },

 async created() {
  await this.loadUserPermissions();
 },

 methods: {
  async loadUserPermissions() {
   try {
    const response = await this.$http.get('/api/user-permissions/');
    this.userPermissions = response.data.permissions;
    this.userRole = response.data.role;
    this.permissionsLoaded = true;
   } catch (error) {
    console.error('Error loading permissions:', error);
   }
  },

  hasPermission(module, permission) {
   return this.userPermissions[module]?.[permission] || false;
  },

  canCreatePolicy() {
   return this.hasPermission('policy', 'create');
  },
```

```
  canApprovePolicy() {
    return this.hasPermission('policy', 'approve');
  },

  canCreateCompliance() {
    return this.hasPermission('compliance', 'create');
  },

  canAssignAudit() {
    return this.hasPermission('audit', 'assign');
  },

  canCreateRisk() {
    return this.hasPermission('risk', 'create');
  },

  canCreateIncident() {
    return this.hasPermission('incident', 'create');
  },

  canViewAnalytics() {
    return this.hasPermission('policy', 'analytics') ||
           this.hasPermission('compliance', 'analytics') ||
           this.hasPermission('audit', 'analytics');
  }
 }
}
```

6.3 Update Key Components

File: frontend/src/components/Policy/PolicyDashboard.vue

```
<template>
 <div class="policy-dashboard">
  <!-- Your existing template content -->

  <!-- Update buttons with permission checks -->
  <button
    v-if="canCreatePolicy()"
    @click="createPolicy"
    class="btn btn-primary"
  >
    Create Policy
  </button>

  <button
    v-if="canApprovePolicy()"
    @click="approvePolicy"
```

```
    class="btn btn-success"
   >
    Approve Policies
   </button>

   <!-- Hide analytics if no permission -->
   <div v-if="canViewAnalytics()" class="analytics-section">
    <!-- Your existing analytics content -->
   </div>
  </div>
</template>

<script>
import { permissionMixin } from '@/mixins/permissionMixin.js';

export default {
 name: 'PolicyDashboard',
 mixins: [permissionMixin],  // ADD THIS LINE

 // Your existing component code remains the same
 data() {
  return {
   // Your existing data
  }
 },

 methods: {
  // Your existing methods remain unchanged
 }
}
</script>
```

File: frontend/src/components/Compliance/ComplianceDashboard.vue

```
<template>
 <div class="compliance-dashboard">
  <!-- Update with permission checks -->
  <button
   v-if="canCreateCompliance()"
   @click="createCompliance"
  >
   Create Compliance
  </button>
 </div>
</template>

<script>
import { permissionMixin } from '@/mixins/permissionMixin.js';
```

```
export default {
  name: 'ComplianceDashboard',
  mixins: [permissionMixin],  // ADD THIS LINE
  // ... rest of your existing code
}
</script>
```

Step 7: Testing Implementation

7.1 Create Test User Script

File: backend/grc/management/commands/setup_rbac_test_users.py

```
from django.core.management.base import BaseCommand
from grc.models import Users, RBAC

class Command(BaseCommand):
    help = 'Create test users for RBAC testing'

    def handle(self, *args, **options):
        # Create test users
        test_users = [
            {
                'username': 'grc_admin',
                'email': 'admin@grc.com',
                'role': 'GRC Administrator',
                'department': 'IT'
            },
            {
                'username': 'policy_mgr',
                'email': 'policy@grc.com',
                'role': 'Policy Manager',
                'department': 'Legal'
            },
            {
                'username': 'auditor',
                'email': 'auditor@grc.com',
                'role': 'Internal Auditor',
                'department': 'Audit'
            },
            {
                'username': 'end_user',
                'email': 'user@grc.com',
                'role': 'End User',
                'department': 'Finance'
            }
        ]
```

```
        for user_data in test_users:
            # Create user if doesn't exist
            user, created = Users.objects.get_or_create(
                UserName=user_data['username'],
                defaults={
                    'email': user_data['email'],
                    'Password': 'test123'  # Use proper hashing in production
                }
            )

            if created:
                self.stdout.write(f"Created user: {user_data['username']}")

            # Create/update RBAC entry
            rbac, created = RBAC.objects.get_or_create(
                UserId=user.UserId,
                defaults={
                    'Email': user_data['email'],
                    'Role': user_data['role'],
                    'Department': user_data['department']
                }
            )

            if created:
                self.stdout.write(f"Created RBAC for: {user_data['username']}")

        self.stdout.write(
            self.style.SUCCESS('Successfully created test users')
        )
```

7.2 Run Test Setup

```
# Navigate to backend directory
cd backend/

# Run the test user creation
python manage.py setup_rbac_test_users

# Start the development server
python manage.py runserver
```

7.3 Test Different User Roles

```
# Test API endpoints with different users
curl -X GET "http://localhost:8000/api/user-permissions/" \
    -H "Authorization: Bearer <token_for_grc_admin>"

curl -X GET "http://localhost:8000/api/user-permissions/" \
    -H "Authorization: Bearer <token_for_end_user>"
```

Step 8: Gradual Rollout Strategy

Phase 1: Enable Middleware (Day 1)

1. Deploy RBAC files
2. Enable middleware in settings
3. Test with GRC Administrator account
4. Monitor logs for any issues

Phase 2: Critical Endpoints (Days 2-3)

1. Add permission classes to:
    - Policy creation/approval
    - Compliance creation/approval
    - Audit assignment
    - Risk creation

Phase 3: All Endpoints (Days 4-7)

1. Add permission classes to remaining views
2. Update all frontend components
3. Full user testing

Phase 4: Optimization (Week 2)

1. Add caching for permissions
2. Performance tuning
3. Add detailed logging

Step 9: Troubleshooting

9.1 Common Issues and Solutions

Issue: "Authentication required" errors

```
# Solution: Check if user is properly authenticated
# Add debug logging to middleware:

class GRCRBACMiddleware(MiddlewareMixin):
    def process_request(self, request):
        print(f"User: {request.user}")
        print(f"Authenticated: {request.user.is_authenticated}")
        # ... rest of middleware
```

Issue: "Insufficient permissions" errors

```
# Solution: Check role mapping
# Add debug view:

@api_view(['GET'])
def debug_user_role(request):
    user_id = request.user.id
    role = RBACUtils.get_user_role(user_id)
    return Response({
        'user_id': user_id,
        'role': role,
        'department': RBACUtils.get_user_department(user_id)
    })
```

9.2 Performance Monitoring

```
# Add to settings.py for monitoring
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'rbac_file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': 'rbac.log',
        },
    },
    'loggers': {
        'grc.rbac': {
            'handlers': ['rbac_file'],
            'level': 'INFO',
            'propagate': True,
        },
    },
}
```

Implementation Checklist

- [ ] Create RBAC directory and files
- [ ] Update Django settings
- [ ] Add RBAC URLs
- [ ] Update database role values
- [ ] Create test users
- [ ] Update critical views with permission classes
- [ ] Create frontend permission mixin

- [ ] Update key Vue components
- [ ] Test with different user roles
- [ ] Deploy gradual rollout
- [ ] Monitor and optimize

This implementation respects your existing architecture while providing comprehensive RBAC functionality with minimal disruption to your current codebase.