Integration Examples of RBAC with GRC

```
# Integration Example 1: Update your main views.py file
# File: backend/grc/views.py
#
================================================================================
=

# Add these imports at the top of your existing views.py file
from rest_framework.permissions import IsAuthenticated
from grc.rbac.permissions import (
    PolicyViewPermission,
    AuditViewPermission,
    ComplianceViewPermission,
    RiskViewPermission,
    IncidentViewPermission
)

# Update your existing KPI views (based on your sample)
@api_view(['GET'])
@permission_classes([IsAuthenticated, AuditViewPermission])  # 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, AuditViewPermission])  # 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)
```

```
@api_view(['GET'])
@permission_classes([IsAuthenticated, AuditViewPermission])  # ADD THIS LINE
def finding_rate(request):
    """Get average findings per audit"""
    time_filter = request.GET.get('time_filter', 'month')
    start_date = request.GET.get('start_date')
    end_date = request.GET.get('end_date')

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

#
==========================================================================
=
# Integration Example 2: Update your audit_views.py file
# File: backend/grc/audit_views.py
#
==========================================================================
=

# Add these imports at the top
from rest_framework.permissions import IsAuthenticated
from grc.rbac.permissions import (
    AuditAssignPermission,
    AuditConductPermission,
    AuditReviewPermission,
    AuditViewPermission
)

# Example updates for your audit views (add permission classes):

@api_view(['POST'])
@permission_classes([IsAuthenticated, AuditAssignPermission])  # ADD THIS LINE
def assign_audit(request):
    """Your existing audit assignment logic"""
    # Your existing code remains unchanged
    pass

@api_view(['POST'])
@permission_classes([IsAuthenticated, AuditConductPermission])  # ADD THIS LINE
```

```
def submit_audit(request):
    """Your existing audit submission logic"""
    # Your existing code remains unchanged
    pass

@api_view(['PUT'])
@permission_classes([IsAuthenticated, AuditReviewPermission])  # ADD THIS LINE
def review_audit(request, audit_id):
    """Your existing audit review logic"""
    # Your existing code remains unchanged
    pass

@api_view(['GET'])
@permission_classes([IsAuthenticated, AuditViewPermission])  # ADD THIS LINE
def get_audit_list(request):
    """Your existing audit list logic"""
    # Your existing code remains unchanged
    pass

#
==========================================================================
=
# Integration Example 3: Update your compliance_views.py file
# File: backend/grc/compliance_views.py
#
==========================================================================
=

from rest_framework.permissions import IsAuthenticated
from grc.rbac.permissions import (
    ComplianceCreatePermission,
    ComplianceEditPermission,
    ComplianceApprovePermission,
    ComplianceViewPermission
)

@api_view(['POST'])
@permission_classes([IsAuthenticated, ComplianceCreatePermission])  # ADD THIS LINE
def create_compliance(request):
    """Your existing compliance creation logic"""
    # Your existing code remains unchanged
```

```
    pass

@api_view(['PUT'])
@permission_classes([IsAuthenticated, ComplianceEditPermission])  # ADD THIS LINE
def edit_compliance(request, compliance_id):
    """Your existing compliance edit logic"""
    # Your existing code remains unchanged
    pass

@api_view(['POST'])
@permission_classes([IsAuthenticated, ComplianceApprovePermission])  # ADD THIS LINE
def approve_compliance(request, compliance_id):
    """Your existing compliance approval logic"""
    # Your existing code remains unchanged
    pass

#
================================================================================
=
# Integration Example 4: Update your risk_views.py file
# File: backend/grc/risk_views.py
#
================================================================================
=

from rest_framework.permissions import IsAuthenticated
from grc.rbac.permissions import (
    RiskCreatePermission,
    RiskEditPermission,
    RiskAssignPermission,
    RiskEvaluatePermission,
    RiskViewPermission
)

@api_view(['POST'])
@permission_classes([IsAuthenticated, RiskCreatePermission])  # ADD THIS LINE
def create_risk(request):
    """Your existing risk creation logic"""
    # Your existing code remains unchanged
    pass
```

```
@api_view(['POST'])
@permission_classes([IsAuthenticated, RiskAssignPermission])  # ADD THIS LINE
def assign_risk(request):
    """Your existing risk assignment logic"""
    # Your existing code remains unchanged
    pass

@api_view(['PUT'])
@permission_classes([IsAuthenticated, RiskEvaluatePermission])  # ADD THIS LINE
def evaluate_risk(request, risk_id):
    """Your existing risk evaluation logic"""
    # Your existing code remains unchanged
    pass

#
================================================================================
=
# Integration Example 5: Update your incident_views.py file
# File: backend/grc/incident_views.py
#
================================================================================
=

from rest_framework.permissions import IsAuthenticated
from grc.rbac.permissions import (
    IncidentCreatePermission,
    IncidentEditPermission,
    IncidentAssignPermission,
    IncidentEvaluatePermission,
    IncidentEscalatePermission
)

@api_view(['POST'])
@permission_classes([IsAuthenticated, IncidentCreatePermission])  # ADD THIS LINE
def create_incident(request):
    """Your existing incident creation logic"""
    # Your existing code remains unchanged
    pass

@api_view(['PUT'])
@permission_classes([IsAuthenticated, IncidentAssignPermission])  # ADD THIS LINE
```

```
def assign_incident(request, incident_id):
    """Your existing incident assignment logic"""
    # Your existing code remains unchanged
    pass

@api_view(['POST'])
@permission_classes([IsAuthenticated, IncidentEscalatePermission])  # ADD THIS LINE
def escalate_incident(request, incident_id):
    """Your existing incident escalation logic"""
    # Your existing code remains unchanged
    pass

#
==========================================================================
=
# Integration Example 6: Update your routes/policy.py file
# File: backend/grc/routes/policy.py
#
==========================================================================
=

# Add these imports at the top of your policy routes file
from rest_framework.permissions import IsAuthenticated
from grc.rbac.permissions import (
    PolicyCreatePermission,
    PolicyEditPermission,
    PolicyApprovePermission,
    PolicyViewPermission,
    FrameworkCreatePermission,
    FrameworkApprovePermission
)

# Update your existing policy route functions:

@api_view(['POST'])
@permission_classes([IsAuthenticated, PolicyCreatePermission])  # ADD THIS LINE
def create_policy_route(request):
    """Your existing policy creation route logic"""
    # Your existing code remains unchanged
    pass
```

```
@api_view(['PUT'])
@permission_classes([IsAuthenticated, PolicyApprovePermission])  # ADD THIS LINE
def approve_policy_route(request, policy_id):
    """Your existing policy approval route logic"""
    # Your existing code remains unchanged
    pass

@api_view(['POST'])
@permission_classes([IsAuthenticated, FrameworkCreatePermission])  # ADD THIS LINE
def create_framework_route(request):
    """Your existing framework creation route logic"""
    # Your existing code remains unchanged
    pass

#
==========================================================================
=
# Integration Example 7: Update your main urls.py file
# File: backend/grc/urls.py
#
==========================================================================
=

# Add these imports to your existing urls.py
from grc.rbac.views import get_user_permissions, get_user_role, debug_user_permissions

# Add these patterns to your existing urlpatterns list:
urlpatterns = [
    # ... your existing URL patterns ...

    # KPI endpoints (your existing ones)
    path('kpi/audit-completion/', views.audit_completion, name='audit-completion'),
    path('kpi/audit-cycle-time/', views.audit_cycle_time, name='audit-cycle-time'),
    path('kpi/finding-rate/', views.finding_rate, name='finding-rate'),
    path('kpi/time-to-close/', views.time_to_close, name='time-to-close'),
    path('kpi/audit-pass-rate/', views.audit_pass_rate, name='audit-pass-rate'),

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

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

#
==========================================================================
=
# Integration Example 8: Update your settings.py file
# File: backend/backend/settings.py
#
==========================================================================
=

# Find your MIDDLEWARE setting and add the RBAC middleware:
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 (ADD THIS BLOCK)
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/',
    ]
}
```

```
# Add logging configuration for RBAC (ADD THIS BLOCK)
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {process:d} {thread:d} {message}',
            'style': '{',
        },
    },
    'handlers': {
        'rbac_file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': 'rbac.log',
            'formatter': 'verbose',
        },
        'console': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'verbose',
        },
    },
    'loggers': {
        'grc.rbac': {
            'handlers': ['rbac_file', 'console'],
            'level': 'INFO',
            'propagate': True,
        },
    },
}

#
==========================================================================
=
# Integration Example 9: Frontend Integration - Update your API service
# File: frontend/src/services/api.js
#
==========================================================================
=
```

```
// Add this to your existing api.js file:

// RBAC Service Methods (ADD THESE)
export const rbacService = {
  // Get user permissions
  async getUserPermissions() {
    try {
      const response = await this.apiCall('/api/user-permissions/', 'GET');
      return response.data;
    } catch (error) {
      console.error('Error fetching user permissions:', error);
      throw error;
    }
  },

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

  // Debug permissions (for development)
  async debugPermissions(module, permission) {
    try {
      const response = await this.apiCall(
        `/api/debug-permissions/?module=${module}&permission=${permission}`,
        'GET'
      );
      return response.data;
    } catch (error) {
      console.error('Error debugging permissions:', error);
      throw error;
    }
  }
};
```

```
//
================================================================================
=
# Integration Example 10: Frontend Permission Mixin
# File: frontend/src/mixins/permissionMixin.js (NEW FILE)
#
================================================================================
=

import { rbacService } from '@/services/api.js';

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

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

  methods: {
    async loadUserPermissions() {
      try {
        const response = await rbacService.getUserPermissions();
        this.userPermissions = response.permissions;
        this.userRole = response.role;
        this.userDepartment = response.department;
        this.permissionsLoaded = true;
      } catch (error) {
        console.error('Error loading permissions:', error);
        // Set default permissions (all false) if loading fails
        this.userPermissions = this.getDefaultPermissions();
        this.permissionsLoaded = true;
      }
    },
```

```
  getDefaultPermissions() {
    const modules = ['policy', 'framework', 'compliance', 'audit', 'risk', 'incident'];
    const permissions = ['create', 'edit', 'approve', 'view', 'assign', 'conduct', 'review',
'evaluate', 'escalate', 'analytics'];

    const defaultPerms = {};
    modules.forEach(module => {
      defaultPerms[module] = {};
      permissions.forEach(perm => {
        defaultPerms[module][perm] = false;
      });
    });

    return defaultPerms;
  },

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

  // Policy permissions
  canCreatePolicy() {
    return this.hasPermission('policy', 'create');
  },

  canEditPolicy() {
    return this.hasPermission('policy', 'edit');
  },

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

  canViewPolicy() {
    return this.hasPermission('policy', 'view');
  },

  // Framework permissions
  canCreateFramework() {
    return this.hasPermission('framework', 'create');
```

```
},

canApproveFramework() {
  return this.hasPermission('framework', 'approve');
},

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

canEditCompliance() {
  return this.hasPermission('compliance', 'edit');
},

canApproveCompliance() {
  return this.hasPermission('compliance', 'approve');
},

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

canConductAudit() {
  return this.hasPermission('audit', 'conduct');
},

canReviewAudit() {
  return this.hasPermission('audit', 'review');
},

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

canAssignRisk() {
  return this.hasPermission('risk', 'assign');
},
```

```
canEvaluateRisk() {
  return this.hasPermission('risk', 'evaluate');
},

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

canAssignIncident() {
  return this.hasPermission('incident', 'assign');
},

canEscalateIncident() {
  return this.hasPermission('incident', 'escalate');
},

// Analytics permissions
canViewAnalytics() {
  return this.hasPermission('policy', 'analytics') ||
         this.hasPermission('compliance', 'analytics') ||
         this.hasPermission('audit', 'analytics') ||
         this.hasPermission('risk', 'analytics') ||
         this.hasPermission('incident', 'analytics');
},

// Role checks
isGRCAdmin() {
  return this.userRole === 'GRC Administrator';
},

isPolicyManager() {
  return this.userRole === 'Policy Manager';
},

isAuditManager() {
  return this.userRole === 'Audit Manager';
},

isRiskManager() {
  return this.userRole === 'Risk Manager';
```

```
    },

    isDepartmentManager() {
      return this.userRole === 'Department Manager';
    },

    isEndUser() {
      return this.userRole === 'End User';
    }
  }
}

#
================================================================================
=
# Integration Example 11: Update existing Vue components
# File: frontend/src/components/Policy/PolicyDashboard.vue
#
================================================================================
=

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

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

      <button
        v-if="canCreateFramework()"
        @click="createFramework"
        class="btn btn-secondary"
      >
        Create Framework
```

```
    </button>

    <button
      v-if="canApprovePolicy()"
      @click="viewApprovals"
      class="btn btn-success"
    >
      Approve Policies
    </button>
  </div>

  <!-- Hide analytics section if no permission -->
  <div v-if="canViewAnalytics()" class="analytics-section">
    <!-- Your existing analytics content -->
    <h3>Policy Analytics</h3>
    <!-- ... analytics content ... -->
  </div>

  <!-- Show role-based content -->
  <div v-if="isGRCAdmin()" class="admin-section">
    <!-- Admin-only 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
```

```
    createPolicy() {
      // Your existing logic
    },

    createFramework() {
      // Your existing logic
    },

    viewApprovals() {
      // Your existing logic
    }
  }
}
</script>

#
================================================================================
=
# Integration Example 12: Management Command for Database Setup
# File: backend/grc/management/commands/setup_rbac.py
#
================================================================================
=

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

class Command(BaseCommand):
    help = 'Setup RBAC system and update existing user roles'

    def add_arguments(self, parser):
        parser.add_argument(
            '--create-test-users',
            action='store_true',
            help='Create test users for each role',
        )

    def handle(self, *args, **options):
        self.stdout.write("Setting up RBAC system...")

        # Update existing role mappings
```

```
        self.update_existing_roles()

        # Create test users if requested
        if options['create_test_users']:
            self.create_test_users()

        # Add database indexes
        self.add_indexes()

        self.stdout.write(
            self.style.SUCCESS('RBAC setup completed successfully')
        )

    def update_existing_roles(self):
        """Update existing roles to match new enum values"""
        role_mappings = {
            'admin': 'GRC Administrator',
            'grc_admin': 'GRC Administrator',
            'administrator': 'GRC Administrator',
            'policy_manager': 'Policy Manager',
            'policy_mgr': 'Policy Manager',
            'policy_approver': 'Policy Approver',
            'policy_reviewer': 'Policy Approver',
            'compliance_manager': 'Compliance Manager',
            'compliance_mgr': 'Compliance Manager',
            'compliance_officer': 'Compliance Officer',
            'compliance_user': 'Compliance Officer',
            'compliance_approver': 'Compliance Approver',
            'compliance_reviewer': 'Compliance Approver',
            'audit_manager': 'Audit Manager',
            'audit_mgr': 'Audit Manager',
            'internal_auditor': 'Internal Auditor',
            'auditor': 'Internal Auditor',
            'external_auditor': 'External Auditor',
            'ext_auditor': 'External Auditor',
            'audit_reviewer': 'Audit Reviewer',
            'audit_approver': 'Audit Reviewer',
            'risk_manager': 'Risk Manager',
            'risk_mgr': 'Risk Manager',
            'risk_analyst': 'Risk Analyst',
            'risk_user': 'Risk Analyst',
```

```
            'risk_reviewer': 'Risk Reviewer',
            'risk_approver': 'Risk Reviewer',
            'incident_manager': 'Incident Response Manager',
            'incident_mgr': 'Incident Response Manager',
            'incident_analyst': 'Incident Analyst',
            'incident_user': 'Incident Analyst',
            'dept_manager': 'Department Manager',
            'department_manager': 'Department Manager',
            'user': 'End User',
            'end_user': 'End User',
            'employee': 'End User',
        }

        for old_role, new_role in role_mappings.items():
            count = RBAC.objects.filter(Role=old_role).update(Role=new_role)
            if count > 0:
                self.stdout.write(f"Updated {count} users from '{old_role}' to '{new_role}'")

    def create_test_users(self):
        """Create test users for each role"""
        test_users = [
            {
                'username': 'grc_admin_test',
                'email': 'grc.admin@test.com',
                'role': 'GRC Administrator',
                'department': 'IT'
            },
            {
                'username': 'policy_mgr_test',
                'email': 'policy.mgr@test.com',
                'role': 'Policy Manager',
                'department': 'Legal'
            },
            {
                'username': 'auditor_test',
                'email': 'auditor@test.com',
                'role': 'Internal Auditor',
                'department': 'Audit'
            },
            {
                'username': 'risk_analyst_test',
```

```
            'email': 'risk.analyst@test.com',
            'role': 'Risk Analyst',
            'department': 'Risk Management'
        },
        {
            'username': 'end_user_test',
            'email': 'end.user@test.com',
            'role': 'End User',
            'department': 'Finance'
        }
    ]

    for user_data in test_users:
        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 test user: {user_data['username']}")

        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 entry for: {user_data['username']}")

def add_indexes(self):
    """Add database indexes for performance"""
    from django.db import connection

    with connection.cursor() as cursor:
```

```
        try:
            cursor.execute("CREATE INDEX IF NOT EXISTS idx_rbac_userid ON rbac(UserId);")
            cursor.execute("CREATE INDEX IF NOT EXISTS idx_rbac_role ON rbac(Role);")
            cursor.execute("CREATE INDEX IF NOT EXISTS idx_rbac_department ON
rbac(Department);")
            self.stdout.write("Database indexes created successfully")
        except Exception as e:
            self.stdout.write(f"Warning: Could not create indexes: {e}")

#
==========================================================================
=
# Usage Instructions
#
==========================================================================
=

"""
To integrate RBAC into your existing GRC system:

1. Create the RBAC module structure:
  mkdir backend/grc/rbac/
  touch backend/grc/rbac/__init__.py

2. Copy the RBAC files (config.py, utils.py, permissions.py, middleware.py, views.py,
decorators.py)

3. Update your settings.py with middleware and configuration

4. Run the setup command:
  python manage.py setup_rbac --create-test-users

5. Update your existing view files by adding permission classes (as shown in examples
above)

6. Update your frontend components with the permission mixin

7. Test with different user roles:
  - Login as different test users
  - Try accessing different endpoints
  - Check that permissions are enforced correctly
```

8. Deploy gradually:
   - Start with non-critical endpoints
   - Monitor logs for issues
   - Gradually add permissions to all endpoints

The middleware will automatically protect URLs matching the patterns, while permission classes give you fine-grained control for specific views.
"""