# Dashboard and Metrics System

Comprehensive guide to the Dashboard and Metrics system implemented in the RAS Dashboard API, providing SQL-based metrics, advanced visualization, and enterprise-grade analytics capabilities.

## 🎯 Overview

The Dashboard and Metrics system provides: - **SQL-Based Metrics** - Define metrics with custom SQL queries - **Advanced Visualization** - 15 chart types with customizable styling - **Dashboard Management** - Global and user-specific dashboards - **Real-time Analytics** - Live metric calculation and caching - **Sharing System** - Dashboard sharing with permission levels - **Template System** - Reusable chart configurations

## 🏗️ Database Schema

### Core Tables

-- Metrics: SQL-based metric definitions  
metrics (id, name, description, type, category, query, value, unit, labels,   
 threshold, source, aggregation\_period, is\_active, metadata,   
 last\_calculated, created\_by, created\_at, updated\_at)  
  
-- Chart Types: Available visualization types  
chart\_types (id, name, type, description, default\_config,   
 supported\_metric\_types, is\_active, created\_at, updated\_at)  
  
-- Chart Configurations: Global styling and theming  
chart\_configurations (id, name, description, color\_palette, default\_width,   
 default\_height, font\_family, font\_size, theme,   
 grid\_config, legend\_config, tooltip\_config,   
 animation\_config, is\_default, is\_active, created\_at, updated\_at)  
  
-- Dashboards: Global admin-managed dashboards  
dashboards (id, name, description, layout, is\_default, is\_global,   
 created\_by, created\_at, updated\_at)  
  
-- User Dashboards: Personal user dashboards  
user\_dashboards (id, user\_id, name, is\_default, layout, created\_at, updated\_at)  
  
-- Dashboard Metrics: Links metrics to dashboards with positioning  
dashboard\_metrics (id, dashboard\_id, metric\_id, chart\_type\_id, chart\_config\_id,   
 position, width, height, config, is\_visible, refresh\_interval,   
 created\_at, updated\_at)  
  
-- Dashboard Shares: Dashboard sharing with permissions  
dashboard\_shares (id, dashboard\_id, user\_id, permission, created\_at, updated\_at)

### Relationships

Users ←→ Dashboards (created\_by)  
Users ←→ UserDashboards (user\_id)  
Users ←→ DashboardShares (user\_id)  
Dashboards ←→ DashboardMetrics ←→ Metrics  
DashboardMetrics ←→ ChartTypes  
DashboardMetrics ←→ ChartConfigurations

## 📊 Metrics System

### Metric Types

// Supported metric types  
const METRIC\_TYPES = [  
 'counter', // Incrementing values (total users, requests)  
 'gauge', // Current state (CPU usage, memory)  
 'histogram', // Value distributions  
 'summary', // Statistical summaries  
 'percentage', // Percentage values (0-100)  
 'ratio', // Ratios between values  
 'trend', // Time-series data  
 'status' // Status indicators (up/down, healthy/unhealthy)  
];

### Metric Categories

const METRIC\_CATEGORIES = [  
 'systems', // System performance and health  
 'assets', // Asset management and tracking  
 'vulnerabilities', // Security vulnerability metrics  
 'compliance', // Compliance and audit metrics  
 'performance', // Application performance metrics  
 'security', // Security monitoring and alerts  
 'financial', // Cost and budget tracking  
 'operational', // Operational efficiency metrics  
 'user\_activity', // User behavior and engagement  
 'network', // Network performance and security  
 'infrastructure', // Infrastructure monitoring  
 'applications' // Application-specific metrics  
];

### SQL-Based Metric Definition

// Example metric with SQL query  
const metric = {  
 name: 'Total Active Users',  
 description: 'Count of all active users in the system',  
 type: 'counter',  
 category: 'user\_activity',  
 query: 'SELECT COUNT(\*) as value FROM users WHERE status = \'active\'',  
 unit: 'users',  
 threshold: {  
 warning: 1000,  
 critical: 500  
 },  
 source: 'user\_database',  
 aggregationPeriod: 'real-time',  
 isActive: true  
};

### Metric Calculation

// Calculate single metric  
const result = await metricsService.calculateMetric(metricId);  
  
// Calculate all active metrics  
const results = await metricsService.calculateAllMetrics();  
  
// Get metric analytics  
const analytics = await metricsService.getMetricsByCategory();

## 📈 Chart System

### Chart Types

const CHART\_TYPES = [  
 'line', // Line charts for trends  
 'bar', // Bar charts for comparisons  
 'pie', // Pie charts for distributions  
 'doughnut', // Doughnut charts for percentages  
 'area', // Area charts for cumulative data  
 'scatter', // Scatter plots for correlations  
 'bubble', // Bubble charts for multi-dimensional data  
 'radar', // Radar charts for multi-axis comparisons  
 'polar', // Polar area charts  
 'gauge', // Gauge charts for single values  
 'table', // Data tables for detailed views  
 'number', // Large number displays  
 'progress', // Progress bars and indicators  
 'heatmap', // Heat maps for matrix data  
 'treemap' // Tree maps for hierarchical data  
];

### Chart Configuration

// Example chart configuration  
const chartConfig = {  
 name: 'Security Dashboard Theme',  
 description: 'Dark theme optimized for security dashboards',  
 colorPalette: [  
 '#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FFEAA7'  
 ],  
 defaultWidth: 500,  
 defaultHeight: 350,  
 fontFamily: 'Roboto, sans-serif',  
 fontSize: 14,  
 theme: 'dark',  
 gridConfig: {  
 display: true,  
 color: 'rgba(255, 255, 255, 0.1)'  
 },  
 legendConfig: {  
 display: true,  
 position: 'top'  
 },  
 tooltipConfig: {  
 enabled: true,  
 backgroundColor: 'rgba(0, 0, 0, 0.8)'  
 },  
 animationConfig: {  
 duration: 1000,  
 easing: 'easeInOutQuart'  
 },  
 isDefault: true  
};

## 🌐 Dashboard System

### Global Dashboards

// Admin-managed dashboards visible to all users  
const globalDashboard = {  
 name: 'Executive Security Dashboard',  
 description: 'High-level security metrics for executive team',  
 layout: {  
 columns: 3,  
 rows: 2,  
 gridSize: 12  
 },  
 isDefault: true,  
 isGlobal: true  
};

### User Dashboards

// Personal dashboards per user  
const userDashboard = {  
 userId: 123,  
 name: 'My Security Metrics',  
 isDefault: true,  
 layout: {  
 columns: 2,  
 rows: 3,  
 theme: 'personal'  
 }  
};

### Dashboard Metrics

// Add metric to dashboard with positioning  
const dashboardMetric = {  
 dashboardId: 1,  
 metricId: 5,  
 chartTypeId: 2,  
 chartConfigId: 1,  
 position: 1,  
 width: 500,  
 height: 350,  
 config: {  
 title: 'Active Users Trend',  
 showLegend: true,  
 animation: true  
 },  
 refreshInterval: 300 // seconds  
};

## 🤝 Dashboard Sharing

### Permission Levels

const SHARE\_PERMISSIONS = [  
 'view', // Read-only access  
 'edit', // Can modify dashboard  
 'admin' // Full control including sharing  
];

### Sharing Example

// Share dashboard with user  
await dashboardService.shareDashboard(dashboardId, userId, 'view');  
  
// Get dashboard shares  
const shares = await dashboardService.getDashboardShares(dashboardId);  
  
// Remove share  
await dashboardService.removeDashboardShare(shareId);

## 🚀 API Endpoints

### Metrics Management

// Core CRUD operations  
POST /api/v1/metrics // Create metric  
GET /api/v1/metrics // Get all metrics  
GET /api/v1/metrics/:id // Get metric by ID  
PUT /api/v1/metrics/:id // Update metric  
DELETE /api/v1/metrics/:id // Delete metric  
  
// Calculation  
POST /api/v1/metrics/:id/calculate // Calculate metric value  
POST /api/v1/metrics/calculate/all // Calculate all metrics  
  
// Analytics  
GET /api/v1/metrics/analytics/by-category // Metrics by category  
GET /api/v1/metrics/analytics/by-type // Metrics by type  
GET /api/v1/metrics/search // Search metrics  
  
// Chart Management  
POST /api/v1/metrics/chart-types // Create chart type  
GET /api/v1/metrics/chart-types // Get chart types  
POST /api/v1/metrics/chart-configurations // Create chart config  
GET /api/v1/metrics/chart-configurations // Get chart configs

### Dashboard Management

// Global Dashboards (Admin)  
POST /api/v1/dashboards/global // Create global dashboard  
GET /api/v1/dashboards/global // Get global dashboards  
PUT /api/v1/dashboards/global/:id // Update global dashboard  
DELETE /api/v1/dashboards/global/:id // Delete global dashboard  
  
// User Dashboards  
POST /api/v1/dashboards/user // Create user dashboard  
GET /api/v1/dashboards/user // Get user dashboards  
PUT /api/v1/dashboards/user/:id // Update user dashboard  
DELETE /api/v1/dashboards/user/:id // Delete user dashboard  
  
// Dashboard Metrics  
POST /api/v1/dashboards/:id/metrics // Add metric to dashboard  
GET /api/v1/dashboards/:id/metrics // Get dashboard metrics  
PUT /api/v1/dashboards/metrics/:id // Update dashboard metric  
DELETE /api/v1/dashboards/metrics/:id // Remove metric  
  
// Dashboard Sharing  
POST /api/v1/dashboards/:id/share // Share dashboard  
GET /api/v1/dashboards/:id/shares // Get dashboard shares  
DELETE /api/v1/dashboards/shares/:id // Remove share  
  
// Access  
GET /api/v1/dashboards/:id // Get dashboard by ID  
GET /api/v1/dashboards // Get accessible dashboards

## 🛠️ Usage Examples

### Creating a Metric

const metricData = {  
 name: 'Critical Vulnerabilities',  
 description: 'Count of critical severity vulnerabilities',  
 type: 'counter',  
 category: 'vulnerabilities',  
 query: `  
 SELECT COUNT(\*) as value  
 FROM vulnerabilities  
 WHERE severity = 'critical'  
 AND status = 'open'  
 `,  
 unit: 'vulnerabilities',  
 threshold: {  
 warning: 10,  
 critical: 25  
 },  
 source: 'vulnerability\_scanner',  
 aggregationPeriod: 'hourly'  
};  
  
const metric = await metricsService.createMetric(metricData, userId);

### Creating a Dashboard

const dashboardData = {  
 name: 'Security Operations Center',  
 description: 'Real-time security monitoring dashboard',  
 layout: {  
 columns: 4,  
 rows: 3,  
 gridSize: 12  
 },  
 isDefault: false  
};  
  
const dashboard = await dashboardService.createGlobalDashboard(dashboardData, adminUserId);

### Adding Metrics to Dashboard

const dashboardMetricData = {  
 metricId: 5,  
 chartTypeId: 1, // Line chart  
 chartConfigId: 2, // Dark theme  
 position: 1,  
 width: 600,  
 height: 400,  
 config: {  
 title: 'Critical Vulnerabilities Over Time',  
 showLegend: true,  
 animation: true,  
 colors: ['#FF6B6B', '#4ECDC4']  
 },  
 refreshInterval: 300  
};  
  
await dashboardService.addMetricToDashboard(dashboardId, dashboardMetricData);

### Calculating Metrics

// Calculate single metric  
const result = await metricsService.calculateMetric(metricId);  
console.log(`Metric value: ${result.calculatedValue}`);  
  
// Calculate all metrics  
const results = await metricsService.calculateAllMetrics();  
console.log(`Calculated ${results.successful} metrics successfully`);

### Dashboard Sharing

// Share dashboard with view permission  
await dashboardService.shareDashboard(dashboardId, userId, 'view');  
  
// Get all shares for a dashboard  
const shares = await dashboardService.getDashboardShares(dashboardId);  
  
// Check user's accessible dashboards  
const accessibleDashboards = await dashboardService.getAccessibleDashboards(userId);

## ⚡ Performance Optimization

### Metric Caching

// Metrics are cached based on aggregation period  
const CACHE\_DURATIONS = {  
 'real-time': 30, // 30 seconds  
 'minutely': 60, // 1 minute  
 'hourly': 300, // 5 minutes  
 'daily': 1800, // 30 minutes  
 'weekly': 3600 // 1 hour  
};

### Query Optimization

// Use efficient SQL queries  
const optimizedQuery = `  
 SELECT COUNT(\*) as value  
 FROM vulnerabilities  
 WHERE severity = 'critical'  
 AND status = 'open'  
 AND created\_at >= NOW() - INTERVAL '24 hours'  
 -- Use indexes on severity, status, created\_at  
`;

### Dashboard Loading

// Load dashboards with metrics efficiently  
const dashboard = await dashboardService.getDashboardById(  
 dashboardId,  
 { includeMetrics: true }  
);

## 🔍 Analytics and Reporting

### Metric Analytics

// Get metrics by category  
const categoryStats = await metricsService.getMetricsByCategory();  
  
// Get metrics by type  
const typeStats = await metricsService.getMetricsByType();  
  
// Search metrics  
const searchResults = await metricsService.searchMetrics({  
 query: 'vulnerability',  
 type: 'counter',  
 category: 'security'  
});

### Dashboard Analytics

// Get dashboard usage statistics  
const dashboardStats = await dashboardService.getDashboardStats();  
  
// Get most used metrics  
const popularMetrics = await metricsService.getPopularMetrics();  
  
// Get dashboard sharing statistics  
const sharingStats = await dashboardService.getSharingStats();

## 🎯 Best Practices

### 1. Efficient SQL Queries

// Good: Use indexes and specific conditions  
const goodQuery = `  
 SELECT COUNT(\*) as value  
 FROM users  
 WHERE status = 'active'  
 AND created\_at >= CURRENT\_DATE - INTERVAL '30 days'  
`;  
  
// Avoid: Full table scans  
const badQuery = `  
 SELECT COUNT(\*) as value  
 FROM users  
 WHERE UPPER(email) LIKE '%@COMPANY.COM'  
`;

### 2. Appropriate Aggregation Periods

// Real-time for critical metrics  
const criticalMetric = {  
 aggregationPeriod: 'real-time',  
 refreshInterval: 30  
};  
  
// Hourly for trend analysis  
const trendMetric = {  
 aggregationPeriod: 'hourly',  
 refreshInterval: 300  
};

### 3. Dashboard Organization

// Group related metrics  
const securityDashboard = {  
 name: 'Security Overview',  
 metrics: [  
 'vulnerabilities\_critical',  
 'vulnerabilities\_high',  
 'security\_incidents',  
 'compliance\_score'  
 ]  
};

### 4. Chart Selection

// Use appropriate chart types  
const chartMappings = {  
 'counter': ['number', 'gauge', 'bar'],  
 'percentage': ['gauge', 'progress', 'pie'],  
 'trend': ['line', 'area'],  
 'distribution': ['pie', 'doughnut', 'bar'],  
 'comparison': ['bar', 'radar']  
};

## 🔧 Troubleshooting

### Metric Calculation Issues

// Debug metric calculation  
try {  
 const result = await metricsService.calculateMetric(metricId);  
 console.log('Calculation successful:', result);  
} catch (error) {  
 console.error('Calculation failed:', error.message);  
 // Check SQL syntax, database connection, permissions  
}

### Dashboard Loading Issues

// Check dashboard permissions  
const hasAccess = await dashboardService.checkDashboardAccess(dashboardId, userId);  
if (!hasAccess) {  
 console.log('User does not have access to dashboard');  
}

### Performance Issues

// Monitor metric calculation times  
const startTime = Date.now();  
await metricsService.calculateMetric(metricId);  
const duration = Date.now() - startTime;  
console.log(`Metric calculation took ${duration}ms`);

## 🚀 Advanced Features

### Dynamic Metrics

// Create metrics with dynamic queries  
const dynamicMetric = {  
 name: 'Department User Count',  
 query: `  
 SELECT COUNT(\*) as value  
 FROM users  
 WHERE department = $1  
 AND status = 'active'  
 `,  
 parameters: ['{{department}}'] // Dynamic parameter  
};

### Custom Chart Types

// Register custom chart type  
const customChartType = {  
 name: 'Security Heatmap',  
 type: 'custom\_heatmap',  
 description: 'Specialized heatmap for security data',  
 defaultConfig: {  
 colorScale: ['#green', '#yellow', '#red'],  
 gridSize: 10  
 },  
 supportedMetricTypes: ['gauge', 'percentage']  
};

### Scheduled Metric Updates

// Set up scheduled metric calculations  
const schedule = {  
 metricId: 123,  
 schedule: '0 \*/5 \* \* \* \*', // Every 5 minutes  
 enabled: true  
};  
  
await metricsService.scheduleMetricCalculation(schedule);

This Dashboard and Metrics system provides enterprise-grade analytics capabilities with SQL-based flexibility, advanced visualization options, and comprehensive dashboard management for complete business intelligence solutions.