QPerform Frontend - Technical Documentation (Behind the Scenes)

Version: 1.0 **Last Updated:** December 9, 2025 **Project:** QPerform Performance Management System

Table of Contents

System Overview

Purpose

QPerform is a web-based performance management application designed for OnQ to track agent performance metrics, manage warnings, and provide automated recommendations for leadership actions.

Key Features

- Real-time performance tracking (Production & QA metrics)
- Automated "At Risk" agent detection
- Warning progression system (Verbal → Written → Final → PIP → Termination)
- Intelligent recommendation engine (Cases A-E)
- Action logging with automatic expiration tracking
- Calendar week-based reporting
- Microsoft SSO authentication via QSoftware backend

Target Users

- **Agents:** Performance tracking
- **Team Leads/Supervisors:** Direct reports monitoring
- **Managers/Assistant Managers:** Team oversight
- **Directors:** Department-level analytics
- **AVPs:** Organization-wide visibility

Architecture

High-Level Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                        User Browser                          │
│  ┌──────────────────────────────────────────────────────┐  │
│  │           Vue 3 SPA (QPerform Frontend)              │  │
│  │  ┌──────────────┐  ┌───────────────────────┐  │  │
│  │  │ Vue Router   │  │   Pinia State Mgmt    │  │  │
│  │  └──────────────┘  └───────────────────────┘  │  │
│  │  ┌──────────────┐  ┌───────────────────────┐  │  │
│  │  │ Components   │  │  Business Logic Utils │   │  │
│  │  └──────────────┘  └───────────────────────┘  │  │
│  └──────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────┘
                    │                             │
                    │ HTTPS/REST                  │ HTTPS/REST
                    ▼                             ▼
┌───────────────────────┐  ┌──────────────────────────┐
│  QPerform Backend     │  │  QSoftware Backend       │
│  (NestJS API)         │  │  (Auth & User Mgmt)      │
│  - Performance Data   │  │  - Microsoft SSO         │
│  - Action Logs        │  │  - User Profiles         │
│  - Recommendations    │  │  - Access Control        │
└───────────────────────┘  └──────────────────────────┘
                    │                             │
                    ▼                             ▼
┌───────────────────────┐  ┌──────────────────────────┐
│  PostgreSQL Database  │  │  Microsoft Entra ID      │
│  - performance_data   │  │  (Azure AD)              │
│  - action_logs        │  └──────────────────────────┘
│  - agent_warnings     │
└───────────────────────┘
```

Application Layers

1. **Presentation Layer** (Vue Components)
   - View components (Pages)
   - Reusable UI components
   - PrimeVue component library integration
2. **State Management Layer** (Pinia Stores)
   - Performance data store
   - Action log store
   - Authentication store
3. **Business Logic Layer** (Utils)

  - Warning system
  - Recommendation engine
  - At-risk detection
  - Performance thresholds
  - Calendar week logic

4. **Service Layer** (API Services)

   - QPerform API client
   - QSoftware authentication service
   - Token management
   - Request/response interceptors

Technology Stack

Frontend Framework

- **Vue 3** (v3.5.22) - Progressive JavaScript framework
- **TypeScript** (v5.9.3) - Type-safe development
- **Vite** (v7.1.7) - Fast build tool and dev server

UI Component Library

- **PrimeVue** (v4.4.1) - Comprehensive UI component suite
- **PrimeIcons** (v7.0.0) - Icon library
- **Aura Theme** - Modern design system

State Management

- **Pinia** (v3.0.4) - Vue 3 state management (Vuex successor)

Routing

- **Vue Router** (v4.6.3) - Official Vue router

HTTP Client

- **Axios** (v1.13.2) - Promise-based HTTP client

Authentication

- **@azure/msal-browser** (v4.26.1) - Microsoft Authentication Library
- **QSoftware SSO Integration** - Custom OAuth 2.0 flow

Utilities

- **@vueuse/core** (v14.0.0) - Vue composition utilities

Development Tools

- **vue-tsc** (v3.1.0) - TypeScript compiler for Vue
- **@vitejs/plugin-vue** (v6.0.1) - Vue plugin for Vite

Project Structure

```
qperform-frontend/
├── public/                      # Static assets
│   └── onq-logo.jpg            # OnQ branding logo
├── src/
│   ├── assets/                 # Application assets
│   ├── components/             # Reusable Vue components
│   │   ├── AtRiskBadge.vue            # At-risk indicator
│   │   ├── FilterPopover.vue          # Filter controls
│   │   ├── Header.vue                 # App header
│   │   ├── MetricToggle.vue           # QA/Production toggle
│   │   ├── PerformanceScoreBadge.vue  # 5-level color badge
│   │   ├── RecommendationsDialog.vue  # Action recommendations
│   │   └── TakeActionDialog.vue       # Action logging form
│   ├── composables/            # Vue composables
│   │   ├── usePerformanceFilters.ts   # Filter logic
│   │   ├── useRoleBasedFiltering.ts   # Access control
│   │   └── useUserRole.ts             # Role detection
│   ├── config/                 # Configuration files
│   │   └── authConfig.ts              # MSAL configuration
│   ├── router/                 # Vue Router configuration
│   │   └── index.ts                   # Route definitions
│   ├── services/               # API service layer
│   │   ├── api.ts                     # QPerform API client
│   │   ├── authService.ts             # MSAL wrapper
│   │   ├── graphService.ts            # Microsoft Graph API
│   │   ├── qsoftwareService.ts        # QSoftware auth API
│   │   ├── sharepointService.ts       # SharePoint integration
│   │   └── tokenManager.ts            # Token handling
│   ├── stores/                 # Pinia state stores
│   │   ├── actionLogStore.ts          # Action logs state
│   │   ├── authStore.ts               # Authentication state
│   │   └── performanceStore.ts        # Performance data state
│   ├── types/                  # TypeScript type definitions
│   │   └── index.ts                   # Centralized types
│   ├── utils/                  # Business logic utilities
│   │   ├── atRiskDetection.ts         # At-risk detection logic
│   │   ├── calendarWeeks.ts           # Week mapping logic
│   │   ├── performanceThresholds.ts   # 5-level system
│   │   ├── performanceUtils.ts        # Performance helpers
│   │   ├── recommendationEngine.ts    # Cases A-E logic
│   │   └── warningSystem.ts           # Warning progression
│   ├── views/                  # Page components (routes)
│   │   ├── AccessDenied.vue           # 403 page
│   │   ├── ActionLogView.vue          # Action log history
│   │   ├── MonthlySummaryView.vue     # Monthly reports
│   │   ├── OAuthCallbackView.vue      # OAuth redirect handler
│   │   ├── PerformanceView.vue        # Main layout
│   │   ├── UnderperformingView.vue    # Performance grid
│   │   └── WelcomeView.vue            # Landing/login page
│   ├── App.vue                 # Root component
│   ├── main.ts                 # Application entry point
│   └── style.css               # Global styles
├── scripts/                    # Build scripts
│   ├── env-encryption.js       # Environment variable encryption
│   └── load-env.js             # Environment loader
├── .env.example                # Environment variable template
├── .gitignore                  # Git ignore rules
├── index.html                  # HTML entry point
├── package.json                # Dependencies and scripts
├── tsconfig.app.json           # TypeScript config (app)
├── tsconfig.json               # TypeScript config (base)
├── vercel.json                 # Vercel deployment config
└── vite.config.ts              # Vite configuration
```

Core Components

1. PerformanceScoreBadge.vue

Purpose: Displays performance scores with 5-level color coding

Props:

- `kpi: number` - Performance score (0-100+)
- `flag: string` - Performance flag (Critical, Low, Normal, Good, Great)
- `metricType: 'QA' | 'Production'` - Metric category
- `showLevel: boolean` - Show level label

Color Coding:

```
QA Thresholds:
- Critical (Red):     < 97%
- Low (Orange):       97-98%
- Normal (Yellow):    98-99%
- Good (Lt Green):    99-100%
- Great (Green):      ≥ 100%

Production Thresholds:
- Critical (Red):     < 98%
- Low (Orange):       98-99%
- Normal (Yellow):    99-100%
- Good (Lt Green):    100-101%
- Great (Green):      > 101%
```

2. AtRiskBadge.vue

Purpose: Visual indicator for agents at risk of termination

Props:

- `atRiskStatus: AtRiskStatus` - Risk assessment object

Risk Levels:

- **CRITICAL:** 2+ Written Warnings (red, pulsing)
- **HIGH:** 3+ consecutive underperforming weeks (orange)
- **MEDIUM:** 3 total underperforming weeks OR 1 Written Warning (yellow)
- **LOW:** Early warning signs (blue)

Features:

- Animated pulse for attention
- Tooltip with detailed reasons
- Expandable details

3. TakeActionDialog.vue

Purpose: Form for logging corrective actions against agents

Features:

- Employee selection with search
- Action type dropdown (Coaching, Verbal Warning, Written Warning, etc.)
- Warning category selection (Production/QA/Other)
- Automatic expiration date calculation
- Warning progression validation
- Current warning status display
- Week range selection
- Notes field with validation

Validation Rules:

- Cannot skip warning levels
- Notes required (min 10 characters)
- All required fields must be filled
- Expiration dates auto-calculated per type

4. RecommendationsDialog.vue

Purpose: Displays automated action recommendations

Input:

- Agent monthly results
- Current warning status
- Recommendation from engine (Cases A-E)

Output:

- Case type (A, B, C, D, or E)
- Recommended action
- Priority level (1-5)
- Critical flag
- Contextual notes

5. MetricToggle.vue

Purpose: Toggle between Production and QA metrics

Props:

- `modelValue: 'QA' | 'Production'`

Emits:

- `update:modelValue` - When toggled

Features:

- Visual icons for each metric
- Descriptions
- Reactive binding

6. FilterPopover.vue

Purpose: Advanced filtering controls

Filters:

- Month/Year selection
- Client selection
- Category selection
- Task selection

Features:

- Cascading filter logic
- Reset functionality
- Filter count indicator

State Management

Pinia Stores

1. authStore.ts

Responsibility: Authentication state and user session

State:

```
{
  user: UserProfile | null
  isAuthenticated: boolean
  role: UserRole
  permissions: UserPermissions
  appState: AppState
}
```

Actions:

- `login()` - Initiate Microsoft SSO
- `logout()` - Clear session
- `checkAuth()` - Verify authentication
- `setUser()` - Store user profile

2. performanceStore.ts

Responsibility: Performance data and filters

State:

```
{
  rawData: PerformanceData[]
  clientSummaryData: ClientSummaryData[]
  filterOptions: FilterOptions
  filters: PerformanceFilters
  loading: boolean
  error: string | null
}
```

Computed:

- `groupedData` - Performance grouped by agent and week
- `employeeList` - Unique agent list
- `weekRanges` - Available week ranges

Actions:

- `loadFilterOptions()` - Fetch available filters
- `loadPerformanceData()` - Fetch performance records
- `setFilters()` - Update active filters
- `resetFilters()` - Clear all filters

3. actionLogStore.ts

Responsibility: Action log history

State:

```
{
  actions: ActionLog[]
  loading: boolean
}
```

Actions:

- `loadActions()` - Fetch action history
- `addAction()` - Add new action
- `deleteAction()` - Remove action

Business Logic Modules

1. warningSystem.ts

Purpose: Warning lifecycle management

Key Functions:

```
// Calculate expiration date based on warning type
calculateExpirationDate(
  warningType: WarningType,
  issueDate: Date
): Date | null

// Check if warning is still active
isWarningActive(warning: ActionLog): boolean

// Get all active warnings for an agent
getActiveWarnings(
  agentEmail: string,
  allActions: ActionLog[],
  category?: WarningCategory
): ActionLog[]

// Validate warning progression (can't skip levels)
validateWarningProgression(
  agentEmail: string,
  newWarningType: WarningType,
  category: WarningCategory,
  allActions: ActionLog[]
): { isValid: boolean; message?: string }

// Get comprehensive warning status
getWarningStatus(
  agentEmail: string,
  category: WarningCategory,
  allActions: ActionLog[]
): WarningStatus
```

Warning Configuration:

```
{
  'Verbal Warning':   { expirationDays: 90,  progressionOrder: 1 },
  'Written Warning':  { expirationDays: 180, progressionOrder: 2 },
  'Final Warning':    { expirationDays: 365, progressionOrder: 3 },
  'PIP':              { expirationDays: 90,  progressionOrder: 3 },
  'Termination':      { expirationDays: 0,   progressionOrder: 4 }
}
```

2. recommendationEngine.ts

Purpose: Automated action recommendations (Cases A-E)

Case Logic:

Case A: 1 Verbal Warning + new underperformance

- **Action:** Issue 2nd Verbal Warning + Coaching
- **Priority:** 3 (Medium)

Case B: 2 Verbal Warnings + new underperformance

- **Action:** Issue Written Warning
- **Priority:** 2 (High)

Case C: 2 Written Warnings + new underperformance

- **Action:** Prepare for Termination
- **Priority:** 1 (Highest)

Case D: Agent underperforming 2+ weeks, no leadership action

- **Action:** Leadership Behavior Report to AVP + Verbal Warning to Leader
- **Priority:** 2 (High)
- **Target:** Leadership

Case E: Leader fails procedures 2nd time

- **Action:** 2nd Leadership Behavior Report + Written Warning to Leader
- **Priority:** 1 (Highest)
- **Target:** Leadership

Key Functions:

```
generateAgentRecommendation(
  agentEmail: string,
  category: WarningCategory,
  allActions: ActionLog[],
  monthlyResults: AgentMonthlyResults,
  underperformingWeeks: number
): RecommendationResult

generateLeadershipRecommendation(
  leaderEmail: string,
  agentEmail: string,
  underperformingWeeks: number,
  actionsTaken: number,
  allActions: ActionLog[]
): RecommendationResult | null
```

3. atRiskDetection.ts

Purpose: Identify agents at risk of termination

Risk Criteria:

1. **CRITICAL:** 2+ Written Warnings (one more strike = termination)
2. **HIGH:** 3+ consecutive underperforming weeks
3. **MEDIUM:** 3 total underperforming weeks OR 1 Written Warning
4. **LOW:** Early warning signs

Key Functions:

```
determineAtRiskStatus(
  agentEmail: string,
  performanceData: PerformanceData[],
  category: WarningCategory,
  allActions: ActionLog[]
): AtRiskStatus

getAtRiskAgents(
  allPerformanceData: PerformanceData[],
  category: WarningCategory,
  allActions: ActionLog[]
): Array<{agentEmail, agentName, atRiskStatus}>
```

4. calendarWeeks.ts

Purpose: Calendar week mapping with 3+ days rule

Calendar Rules:

- Week = Sunday to Saturday
- Weeks with 3+ days in a month count for that month
- Example: Week 09/28/25-10/04/25 (3 days Sept, 4 days Oct) → October

Key Functions:

```
getWeekStart(date: Date): Date  // Returns Sunday
getWeekEnd(date: Date): Date    // Returns Saturday

determineWeekMonth(
  weekStart: Date,
  weekEnd: Date
): { month: number; year: number }  // Applies 3+ days rule

getWeeksForMonth(
  month: number,
  year: number
): WeekRange[]  // All weeks in a month
```

5. performanceThresholds.ts

Purpose: 5-level performance classification

Thresholds:

```
QA_THRESHOLDS:
  CRITICAL_THRESHOLD:  97
  LOW_THRESHOLD:       98
  NORMAL_THRESHOLD:    99
  GOOD_THRESHOLD:      100

PRODUCTION_THRESHOLDS:
  CRITICAL_THRESHOLD:  98
  LOW_THRESHOLD:       99
  NORMAL_THRESHOLD:    100
  GOOD_THRESHOLD:      101
```

Key Functions:

```
getPerformanceLevel(
  score: number,
  metricType: MetricType
): PerformanceLevel

getPerformanceSeverity(
  score: number,
  metricType: MetricType
): PrimeVueSeverity

getLevelColor(level: PerformanceLevel): string
```

API Integration

QPerform Backend API

Base URL: `VITE_API_URL` environment variable

Endpoints:

Performance Data

```
GET /api/performance/data?month={month}&year={year}&category={category}
Response: PerformanceData[]

GET /api/performance/monthly-summary?month={month}&year={year}
Response: MonthlySummaryResponse

GET /api/performance/client-summary?month={month}&year={year}
Response: ClientSummaryData[]

GET /api/performance/filters?client={client}&category={category}
Response: FilterOptions
```

Action Logs

```
GET /api/action-log
Response: ActionLog[]

POST /api/action-log
Body: ActionLogCreateDto
Response: { success: boolean; id: number }

DELETE /api/action-log/:id
Response: { success: boolean; message: string }

POST /api/action-log/check-duplicate
Body: { agentEmail, weekStartDate, weekEndDate }
Response: { exists: boolean; action?: ActionLog }
```

Agent Warnings

```
GET /api/warnings/:agentEmail
Response: AgentWarning[]
```

QSoftware Backend API

Base URL: `VITE_QSOFTWARE_API_URL`

Endpoints:

Authentication

```
POST /auth/login
Body: { email, password }
Response: { accessToken, refreshToken, user }

POST /auth/microsoft-sso
Body: { microsoftToken }
Response: { accessToken, refreshToken, user }

POST /auth/refresh
Body: { refreshToken }
Response: { accessToken }

GET /auth/me
Response: UserProfile

GET /auth/validate-access?application=qperform
Response: { hasAccess, role, permissions }

POST /auth/logout
Response: void
```

Axios Interceptors

Request Interceptor:

```
// Automatically adds Bearer token to all requests
apiClient.interceptors.request.use(async (config) => {
  const token = await getAccessToken()
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

Response Interceptor:

```
// Handles token refresh on 401 errors
apiClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response?.status === 401) {
      // Attempt token refresh
      const refreshToken = getStoredRefreshToken()
      if (refreshToken) {
        const newToken = await refreshAccessToken(refreshToken)
        // Retry original request with new token
      }
    }
    return Promise.reject(error)
  }
)
```

Authentication & Authorization

Microsoft SSO Flow

```
1. User clicks "Sign in with Microsoft" on WelcomeView
2. Frontend redirects to QSoftware OAuth endpoint:
   GET https://api.qsoftware.cloud/api/auth/microsoft/login
       ?redirect_uri=http://localhost:5174/auth/callback
3. QSoftware redirects to Microsoft Entra ID
4. User authenticates with Microsoft
5. Microsoft redirects to QSoftware callback
6. QSoftware validates user and generates tokens
7. QSoftware redirects back to frontend with tokens:
   GET http://localhost:5174/auth/callback
       ?access_token={token}&refresh_token={token}
8. Frontend stores tokens in localStorage
9. Frontend redirects to /performance
```

Token Management

Storage:

- `qsoftware_token` - Access token (JWT)
- `qsoftware_refresh_token` - Refresh token
- `qsoftware_user` - User profile JSON

Refresh Strategy:

- Automatic refresh on 401 responses
- Access tokens expire after 1 hour
- Refresh tokens expire after 7 days

Role-Based Access Control

User Roles:

```
type UserRole =
  | 'AVP'
  | 'Director'
  | 'Manager'
  | 'Assistant Manager'
  | 'Supervisor'
```

```
    | 'Team Lead'
    | 'Agent'
    | 'Unauthorized'
```

Permissions Matrix:

```
{
  AVP: {
    canTakeAction: true,
    canViewReports: true,
    canViewAllClients: true,
    receivesNotifications: true
  },
  Director: {
    canTakeAction: true,
    canViewReports: true,
    canViewAllClients: true,
    receivesNotifications: true
  },
  Manager: {
    canTakeAction: true,
    canViewReports: true,
    canViewAllClients: false,
    receivesNotifications: true
  },
  // ... etc
}
```

Data Flow

Performance Data Loading Flow

```
1. User lands on /performance
2. PerformanceView.vue mounts
3. performanceStore.loadFilterOptions() called
   → GET /api/performance/filters
   → Auto-selects current month/year
4. performanceStore.loadPerformanceData() called
   → GET /api/performance/data?month=December&year=2025
   → GET /api/performance/client-summary?month=December&year=2025
5. Data stored in rawData and clientSummaryData
6. Computed properties generate:
   - groupedData (agent → week → records)
   - employeeList (unique agents)
   - weekRanges (unique weeks)
7. UnderperformingView renders grid
8. For each cell:
   - Fetch agent data for week
   - Calculate at-risk status
   - Determine performance level
   - Render badge
```

Action Logging Flow

```
1. User clicks "Take Action" on agent row
2. TakeActionDialog opens
3. Dialog loads:
   - Agent info from selected row
   - Current warning status via getWarningStatus()
   - Available week ranges
4. User fills form and clicks Submit
5. Dialog validates:
   - Required fields
   - Warning progression
   - Notes length
6. If valid:
   - Calculate expiration date
   - POST /api/action-log
7. On success:
   - actionLogStore.addAction() updates local state
   - Dialog closes
   - Toast notification
   - Grid refreshes with new warning status
```

Recommendation Generation Flow

```
1. User clicks "View Actions" on agent row
2. UnderperformingView.openViewActions() runs:
   - Gets agent performance data
   - Counts underperforming weeks
   - Calls generateAgentRecommendation()
3. Recommendation engine evaluates:
   - Case C (highest priority)
   - Case B
   - Case A
   - Coaching only
   - Monitor
4. Returns RecommendationResult with:
   - caseType, action, priority, isCritical, notes
5. RecommendationsDialog opens with result
6. User can:
   - Review recommendation
   - Close dialog
   - Click "Take Action" → Opens TakeActionDialog
```

Deployment

Environment Variables

```
# QPerform Backend API
VITE_API_URL=https://api.qperform.cloud

# QSoftware Authentication Backend
VITE_QSOFTWARE_API_URL=https://api.qsoftware.cloud/api

# Microsoft Azure AD (MSAL)
VITE_MSAL_CLIENT_ID=your-client-id
VITE_MSAL_AUTHORITY=https://login.microsoftonline.com/your-tenant-id
VITE_MSAL_REDIRECT_URI=https://qperform.yourcompany.com/auth/callback

# Environment Encryption (for secure .env storage)
VITE_ENV_ENCRYPTION_PASSWORD=your-secure-password
```

Build Process

```
# Development
npm run dev  # Starts dev server on port 5174

# Production Build
npm run build  # Compiles to /dist folder

# Preview Production Build
npm run preview  # Serves /dist on port 4173

# Type Check
npm run build:check  # TypeScript compilation check
```

Vercel Deployment

Configuration: `vercel.json`

```
{
  "buildCommand": "npm run build:demo",
  "outputDirectory": "dist",
  "framework": "vite",
  "rewrites": [
    { "source": "/(.*)", "destination": "/index.html" }
  ]
}
```

Deployment Commands:

```
# Deploy to Vercel
npm run deploy:vercel

# Manual deployment
vercel --prod
```

Build Optimization

Code Splitting:

```
// vite.config.ts
rollupOptions: {
  output: {
    manualChunks: {
      'vendor': ['vue', 'vue-router', 'pinia'],
      'primevue': ['primevue'],
    },
  },
}
```

Lazy Loading:

```
// All routes use lazy loading
{
  path: '/performance',
  component: () => import('../views/PerformanceView.vue')
}
```

Performance Optimization

Computed Properties

Use computed properties for expensive calculations:

```
const groupedData = computed(() => {
  return groupByWeek(rawData.value)
})
```

Memoization

Cache at-risk calculations:

```
const atRiskCache = new Map<string, AtRiskStatus>()

const getAtRiskStatus = (agentKey: string) => {
  if (atRiskCache.has(agentKey)) {
    return atRiskCache.get(agentKey)
  }
  const status = determineAtRiskStatus(...)
  atRiskCache.set(agentKey, status)
  return status
}
```

Lazy Rendering

Only render visible cells in performance grid.

Debouncing

Debounce metric toggle to prevent rapid re-renders:

```
import { useDebounceFn } from '@vueuse/core'

const debouncedMetricChange = useDebounceFn(() => {
  // Update metric
}, 300)
```

Bundle Size Optimization

- Tree-shaking unused PrimeVue components
- Code splitting by route
- Vendor chunk separation

Security Considerations

XSS Prevention

- Vue automatically escapes user input
- Use `v-html` only for trusted content

CSRF Protection

- API uses Bearer tokens (not cookies)
- No CSRF vulnerability

Token Security

- Tokens stored in localStorage (acceptable for SPAs)
- Automatic token refresh

- Tokens expire after inactivity

API Security

- All API calls require Bearer token
- HTTPS enforced in production
- CORS configured on backend

Error Handling

API Error Handling

```
try {
  const data = await fetchPerformanceData(filters)
  rawData.value = data
} catch (error) {
  console.error('Failed to load performance data:', error)
  error.value = 'Failed to load data. Please try again.'
  // Show toast notification
}
```

Global Error Handler

```
// main.ts
app.config.errorHandler = (err, instance, info) => {
  console.error('Global error:', err, info)
  // Log to monitoring service
}
```

401 Handling

Automatic redirect to login on session expiration:

```
window.addEventListener('auth:session-expired', () => {
  router.push('/welcome')
})
```

Testing Strategy

Unit Tests

Test business logic modules:

- `warningSystem.ts`

- `recommendationEngine.ts`
- `atRiskDetection.ts`
- `calendarWeeks.ts`
- `performanceThresholds.ts`

Component Tests

Test Vue components with Vitest + Vue Test Utils

Integration Tests

Test API integration and data flow

E2E Tests

Test critical user flows with Cypress/Playwright

Monitoring & Logging

Console Logging

Development environment logs all API calls and errors.

Performance Monitoring

Track page load times and API response times.

Error Tracking

Integrate with Sentry or similar service for production error tracking.

Future Enhancements

1. **Notification System**
   - Weekly automated reports
   - At-risk agent alerts
   - Warning expiration reminders
2. **Advanced Analytics**
   - Performance trends charts
   - Bench strength analysis
   - Predictive analytics
3. **Export Functionality**
   - Excel export
   - PDF reports
   - CSV downloads

4. **Mobile App**
    - React Native mobile app
    - Push notifications
5. **Real-time Updates**
    - WebSocket integration
    - Live performance updates

Support & Maintenance

Code Maintenance

- Follow Vue 3 Composition API best practices
- Maintain TypeScript strict mode
- Regular dependency updates
- Code reviews for all changes

Documentation Updates

Update this document when:

- New features added
- Architecture changes
- API endpoints modified
- Security policies updated

Document Version: 1.0 **Author:** QPerform Development Team **Contact:** dev@qsoftware.cloud