Epic 5.2: Advanced Workspace Components

Epic Overview

Advanced Workspace Components build upon the foundation components to provide sophisticated workspace management, utility, and security features. These components enable complex workspace operations, administrative functions, data management, and enterprise-level security across all workspace contexts.

Epic Goals:

- Create comprehensive workspace management systems
- Build utility components for workspace operations
- Implement security and compliance features
- Enable advanced data handling and archival
- Provide enterprise-grade workspace functionality

Story 5.2.1: Workspace Management Components

Overview

Build workspace management components that provide centralized workspace context management, context-aware routing, and comprehensive permission systems for multi-tenant operations.

Context

- Complete workspace-specific interactive component system
- Need advanced workspace management capabilities
- Must support complex workspace operations and administration
- Management components are critical for workspace governance
- Need scalable and secure workspace management

Requirements

1. Build WorkspaceContextProvider component:

- Centralized workspace context management
- Context switching and state persistence
- Permission enforcement and validation

- Real-time context synchronization
- Context isolation and security

2. Build WorkspaceRouter component:

- Context-aware routing and navigation
- Permission-based route protection
- Dynamic route generation
- Route state management
- Navigation history tracking

3. Build WorkspacePermissions component:

- Permission management interface
- Role-based access control
- Permission inheritance and overrides
- Permission audit and tracking
- Bulk permission management

Specific Tasks

- ✅ Build WorkspaceContextProvider component
- ✅ Add context management and persistence
- ✅ Build WorkspaceRouter component
- ✅ Add context-aware routing
- ✅ Build WorkspacePermissions component
- ✅ Add permission management interface
- ✅ Create workspace state management
- ✅ Add security and audit features

Documentation Required

- Workspace management architecture
- Context management implementation
- Permission system documentation
- Security and audit guidelines
- Route protection patterns

- State management best practices

Testing Requirements

- Context management functionality tests
- Permission system tests
- Route protection tests
- Security validation tests
- State persistence tests
- Performance tests for large workspaces

Integration Points

- Integration with workspace context providers
- Permission system integration
- Routing system integration
- Security system integration
- Audit logging integration

Deliverables

- WorkspaceContextProvider component
- WorkspaceRouter component with protection
- WorkspacePermissions component
- Workspace state management system
- Security and audit features
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface WorkspaceContextProviderProps {
  workspace: Workspace
  user: User
  permissions: string[]
  onContextChange?: (context: WorkspaceContext) => void
  onPermissionDenied?: (permission: string) => void
  children: React.ReactNode
  securityMode?: 'strict' | 'permissive'
  auditEnabled?: boolean
}

interface WorkspaceRouterProps {
  routes: WorkspaceRoute[]
  currentPath: string
  onRouteChange: (path: string) => void
  fallbackRoute?: string
  loadingComponent?: React.ComponentType
  errorComponent?: React.ComponentType
  permissionDeniedComponent?: React.ComponentType
}

interface WorkspacePermissionsProps {
  workspace: Workspace
  permissions: Permission[]
  roles: Role[]
  users: User[]
  onPermissionChange: (permission: Permission) => void
  onRoleChange: (role: Role) => void
  onUserPermissionChange: (user: User, permissions: string[]) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showInheritance?: boolean
  showAudit?: boolean
  bulkActions?: boolean
}

interface WorkspaceContext {
  workspace: Workspace
  user: User
  permissions: string[]
  roles: string[]
  settings: WorkspaceSettings
  state: WorkspaceState
}
```

```
interface WorkspaceRoute {
  path: string
  component: React.ComponentType
  permissions?: string[]
  roles?: string[]
  workspaceTypes?: string[]
  exact?: boolean
  redirect?: string
  children?: WorkspaceRoute[]
}

interface Permission {
  id: string
  name: string
  description: string
  category: string
  workspaceTypes: string[]
  inheritable: boolean
  grantedBy?: string
  grantedAt?: Date
  expiresAt?: Date
}
```

Implementation Example

jsx

```
// WorkspaceContextProvider implementation
function WorkspaceContextProvider({
  workspace,
  user,
  permissions,
  onContextChange,
  children,
  securityMode = 'strict',
  auditEnabled = true
}) {
  const [context, setContext] = useState({
    workspace,
    user,
    permissions,
    roles: user.roles,
    settings: workspace.settings,
    state: workspace.state
  })

  const [auditLog, setAuditLog] = useState([])

  // Permission checking with audit
  const hasPermission = useCallback((permission) => {
    const hasAccess = context.permissions.includes(permission) ||
                     (securityMode === 'permissive' &&
                      context.roles.includes('admin'))

    if (auditEnabled) {
      logPermissionCheck(permission, hasAccess)
    }

    if (!hasAccess) {
      onPermissionDenied?.(permission)
    }

    return hasAccess
  }, [context.permissions, context.roles, securityMode, auditEnabled])

  // Context switching with validation
  const switchContext = useCallback(async (newWorkspace) => {
    try {
      // Validate permission to switch
      if (!hasPermission('workspace:switch')) {
```

```
      throw new Error('Permission denied: workspace:switch')
    }

    // Save current context state
    await saveContextState(context)

    // Load new context
    const newContext = await loadWorkspaceContext(newWorkspace, user)

    // Update context
    setContext(newContext)
    onContextChange?.(newContext)

    // Audit log
    if (auditEnabled) {
      logContextSwitch(workspace, newWorkspace)
    }
  } catch (error) {
    console.error('Context switch failed:', error)
    throw error
  }
}, [context, hasPermission, user, auditEnabled])

// Real-time context synchronization
useEffect(() => {
  const syncHandler = (update) => {
    if (update.workspaceId === workspace.id) {
      setContext(prev => ({
        ...prev,
        ...update.changes
      }))
    }
  }

  subscribeToWorkspaceUpdates(workspace.id, syncHandler)
  return () => unsubscribeFromWorkspaceUpdates(workspace.id, syncHandler)
}, [workspace.id])

const value = {
  ...context,
  hasPermission,
  switchContext,
  auditLog: auditEnabled ? auditLog : undefined
}
```

```
  return (
    <WorkspaceContext.Provider value={value}>
      {children}
    </WorkspaceContext.Provider>
  )
}

// WorkspaceRouter implementation
function WorkspaceRouter({
  routes,
  currentPath,
  onRouteChange,
  fallbackRoute = '/unauthorized',
  loadingComponent: Loading = DefaultLoading,
  errorComponent: Error = DefaultError,
  permissionDeniedComponent: PermissionDenied = DefaultPermissionDenied
}) {
  const { workspace, hasPermission, user } = useWorkspace()
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState(null)

  // Find matching route
  const matchedRoute = useMemo(() => {
    return findMatchingRoute(routes, currentPath)
  }, [routes, currentPath])

  // Check route permissions
  const canAccessRoute = useMemo(() => {
    if (!matchedRoute) return false

    // Check workspace type
    if (matchedRoute.workspaceTypes &&
        !matchedRoute.workspaceTypes.includes(workspace.type)) {
      return false
    }

    // Check permissions
    if (matchedRoute.permissions) {
      return matchedRoute.permissions.every(perm => hasPermission(perm))
    }

    // Check roles
    if (matchedRoute.roles) {
      return matchedRoute.roles.some(role => user.roles.includes(role))
    }

    return true
  }, [matchedRoute, workspace, hasPermission, user])

  // Handle route rendering
  if (loading) return <Loading />
  if (error) return <Error error={error} />
  if (!canAccessRoute) {
    return <PermissionDenied
      route={matchedRoute}
      onBack={() => onRouteChange(fallbackRoute)}
    />
  }

  const RouteComponent = matchedRoute.component

  return (
    <RouteContainer>
      <RouteComponent />
    </RouteContainer>
  )
}
```

Performance Requirements

- Context switching under 500ms
- Permission checking under 10ms
- Route resolution under 50ms
- Memory usage under 50MB
- Audit logging under 100ms

Story 5.2.2: Workspace Utility Components

Overview

Build workspace utility components that provide essential workspace functionality including audit trails, notifications, search, and data export capabilities.

Context

- Complete workspace management component system
- Need utility components for workspace operations
- Must support various workspace utility functions
- Utility components enhance workspace productivity
- Need consistent utility patterns across workspaces

Requirements

1. Build WorkspaceAuditTrail component:

- Activity logging and tracking
- Audit trail visualization
- Event filtering and search
- Audit report generation
- Compliance monitoring

2. Build WorkspaceNotifications component:

- Notification management and display
- Notification filtering and categorization
- Real-time notification updates
- Notification action handling
- Notification preferences

3. Build WorkspaceSearch component:

- Cross-workspace search functionality
- Search result categorization
- Search history and suggestions
- Advanced search filters
- Search analytics and insights

4. Build WorkspaceExport component:

- Data export functionality
- Export format options
- Export scheduling and automation

- Export history tracking
- Export security and permissions

Specific Tasks

- ✅ Build WorkspaceAuditTrail component
- ✅ Add activity logging and visualization
- ✅ Build WorkspaceNotifications component
- ✅ Add notification management
- ✅ Build WorkspaceSearch component
- ✅ Add cross-workspace search
- ✅ Build WorkspaceExport component
- ✅ Add data export functionality
- ✅ Create utility service integration
- ✅ Add performance optimization

Documentation Required

- Workspace utility architecture
- Audit trail implementation
- Notification system integration
- Search functionality documentation
- Export system capabilities
- Performance optimization guidelines

Testing Requirements

- Audit trail functionality tests
- Notification system tests
- Search functionality tests
- Export system tests
- Performance tests for large datasets
- Security validation tests

Integration Points

- Integration with workspace management system

- Audit logging service integration
- Notification service integration
- Search service integration
- Export service integration

Deliverables

- WorkspaceAuditTrail component
- WorkspaceNotifications component
- WorkspaceSearch component
- WorkspaceExport component
- Utility service integration
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface WorkspaceAuditTrailProps {
  workspace: Workspace
  events: AuditEvent[]
  onEventClick?: (event: AuditEvent) => void
  onExportAudit?: (format: ExportFormat) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  filters?: AuditFilter[]
  onFilterChange?: (filters: AuditFilter[]) => void
  showTimeline?: boolean
  showDetails?: boolean
  permissions?: string[]
}

interface WorkspaceNotificationsProps {
  notifications: Notification[]
  onNotificationClick?: (notification: Notification) => void
  onNotificationAction?: (notification: Notification, action: string) => void
  onMarkAllRead?: () => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  filters?: NotificationFilter[]
  onFilterChange?: (filters: NotificationFilter[]) => void
  realTimeUpdates?: boolean
  maxVisible?: number
}

interface WorkspaceSearchProps {
  onSearch: (query: string, filters?: SearchFilter[]) => void
  onResultClick?: (result: SearchResult) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  placeholder?: string
  showFilters?: boolean
  showHistory?: boolean
  showSuggestions?: boolean
  categories?: SearchCategory[]
  permissions?: string[]
}

interface WorkspaceExportProps {
  workspace: Workspace
  exportTypes: ExportType[]
  onExport: (type: ExportType, options: ExportOptions) => void
  onScheduleExport?: (schedule: ExportSchedule) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showHistory?: boolean
  showScheduling?: boolean
  permissions?: string[]
}

interface AuditEvent {
  id: string
  timestamp: Date
  user: User
  action: string
  resource: string
  details: Record<string, any>
  ip?: string
  userAgent?: string
  workspaceId: string
  severity: 'low' | 'medium' | 'high' | 'critical'
}

interface SearchResult {
  id: string
  title: string
  description: string
  category: string
  url: string
  relevance: number
  workspaceId: string
  timestamp: Date
  metadata?: Record<string, any>
}

interface ExportType {
  id: string
  name: string
  description: string
  format: 'csv' | 'json' | 'pdf' | 'xlsx'
  category: string
  permissions?: string[]
  options?: ExportOption[]
}
```

Implementation Example

jsx

```
// WorkspaceAuditTrail implementation
function WorkspaceAuditTrail({
  workspace,
  events,
  onEventClick,
  showTimeline = true,
  showDetails = true,
  filters = []
}) {
  const [selectedEvent, setSelectedEvent] = useState(null)
  const [activeFilters, setActiveFilters] = useState(filters)
  const [groupBy, setGroupBy] = useState('time')

  const filteredEvents = useMemo(() => {
    return events.filter(event => {
      return activeFilters.every(filter => {
        switch (filter.type) {
          case 'user':
            return event.user.id === filter.value
          case 'action':
            return event.action.includes(filter.value)
          case 'severity':
            return event.severity === filter.value
          case 'dateRange':
            return event.timestamp >= filter.start &&
                   event.timestamp <= filter.end
          default:
            return true
        }
      })
    })
  }, [events, activeFilters])

  const groupedEvents = useMemo(() => {
    if (!showTimeline) return { all: filteredEvents }

    return filteredEvents.reduce((groups, event) => {
      let key
      switch (groupBy) {
        case 'time':
          key = formatDate(event.timestamp, 'YYYY-MM-DD')
          break
        case 'user':
```

```
        key = event.user.name
        break
      case 'action':
        key = event.action
        break
      default:
        key = 'all'
    }

    if (!groups[key]) groups[key] = []
    groups[key].push(event)
    return groups
  }, {})
}, [filteredEvents, groupBy, showTimeline])

return (
  <AuditTrailContainer>
    <AuditTrailHeader>
      <Heading level={3}>Audit Trail</Heading>
      <HeaderActions>
        <Select
          value={groupBy}
          onChange={setGroupBy}
          size="sm"
        >
          <Option value="time">Group by Time</Option>
          <Option value="user">Group by User</Option>
          <Option value="action">Group by Action</Option>
        </Select>
        <Button
          variant="secondary"
          size="sm"
          onClick={() => onExportAudit?.('pdf')}
        >
          Export
        </Button>
      </HeaderActions>
    </AuditTrailHeader>

    <AuditTrailFilters>
      <FilterBar
        filters={activeFilters}
        onFilterChange={setActiveFilters}
        availableFilters={[
          { type: 'user', label: 'User' },
          { type: 'action', label: 'Action' },
          { type: 'severity', label: 'Severity' },
          { type: 'dateRange', label: 'Date Range' }
        ]}
      />
    </AuditTrailFilters>

    <AuditTrailContent>
      {showTimeline ? (
        <Timeline>
          {Object.entries(groupedEvents).map(([group, events]) => (
            <TimelineSection key={group}>
              <TimelineHeader>{group}</TimelineHeader>
              <TimelineEvents>
                {events.map(event => (
                  <AuditEventItem
                    key={event.id}
                    event={event}
                    onClick={() => {
                      setSelectedEvent(event)
                      onEventClick?.(event)
                    }}
                    selected={selectedEvent?.id === event.id}
                    showDetails={showDetails}
                  />
                ))}
              </TimelineEvents>
            </TimelineSection>
          ))}
        </Timeline>
      ) : (
        <EventList>
          {filteredEvents.map(event => (
            <AuditEventItem
              key={event.id}
              event={event}
              onClick={() => {
                setSelectedEvent(event)
                onEventClick?.(event)
              }}
              selected={selectedEvent?.id === event.id}
              showDetails={showDetails}
            />
          ))}
        </EventList>
      )}
    </AuditTrailContent>

    {selectedEvent && showDetails && (
      <EventDetails
        event={selectedEvent}
        onClose={() => setSelectedEvent(null)}
      />
    )}
  </AuditTrailContainer>
 )
}

// WorkspaceSearch implementation
function WorkspaceSearch({
 onSearch,
 onResultClick,
 showFilters = true,
 showHistory = true,
 showSuggestions = true,
 categories = []
}) {
 const [query, setQuery] = useState('')
 const [results, setResults] = useState([])
 const [loading, setLoading] = useState(false)
 const [searchHistory, setSearchHistory] = useState([])
 const [suggestions, setSuggestions] = useState([])
 const [selectedCategories, setSelectedCategories] = useState([])

 const debouncedSearch = useDebouncedCallback(
  async (searchQuery) => {
   if (!searchQuery.trim()) {
    setResults([])
    return
   }

   setLoading(true)
   try {
    const searchResults = await performSearch(searchQuery, {
     categories: selectedCategories,
     workspace: workspace.id
    })
```

```
      setResults(searchResults)

      // Update search history
      setSearchHistory(prev => [
        searchQuery,
        ...prev.filter(q => q !== searchQuery).slice(0, 9)
      ])
    } catch (error) {
      console.error('Search error:', error)
    } finally {
      setLoading(false)
    }
  },
  300
)

useEffect(() => {
  debouncedSearch(query)
}, [query, selectedCategories])

// Load suggestions based on query
useEffect(() => {
  if (showSuggestions && query.length > 2) {
    loadSearchSuggestions(query).then(setSuggestions)
  } else {
    setSuggestions([])
  }
}, [query, showSuggestions])

return (
  <SearchContainer>
    <SearchInput
      value={query}
      onChange={(e) => setQuery(e.target.value)}
      placeholder="Search across workspace..."
      icon="search"
      loading={loading}
    />

    {showFilters && categories.length > 0 && (
      <SearchFilters>
        {categories.map(category => (
          <FilterChip
            key={category.id}
        label={category.name}
        selected={selectedCategories.includes(category.id)}
        onClick={() => {
          setSelectedCategories(prev =>
            prev.includes(category.id)
              ? prev.filter(c => c !== category.id)
              : [...prev, category.id]
          )
        }}
      />
    ))}
  </SearchFilters>
)}

{(suggestions.length > 0 || (showHistory && searchHistory.length > 0)) && (
  <SearchDropdown>
    {suggestions.length > 0 && (
      <DropdownSection>
        <SectionTitle>Suggestions</SectionTitle>
        {suggestions.map(suggestion => (
          <DropdownItem
            key={suggestion}
            onClick={() => setQuery(suggestion)}
          >
            <Icon name="search" size="sm" />
            {suggestion}
          </DropdownItem>
        ))}
      </DropdownSection>
    )}

    {showHistory && searchHistory.length > 0 && (
      <DropdownSection>
        <SectionTitle>Recent Searches</SectionTitle>
        {searchHistory.map(historyItem => (
          <DropdownItem
            key={historyItem}
            onClick={() => setQuery(historyItem)}
          >
            <Icon name="clock" size="sm" />
            {historyItem}
          </DropdownItem>
        ))}
      </DropdownSection>
```

```
          )}
        </SearchDropdown>
      )}

      {results.length > 0 && (
        <SearchResults>
          {results.map(result => (
            <SearchResultItem
              key={result.id}
              result={result}
              onClick={() => onResultClick?.(result)}
            />
          ))}
        </SearchResults>
      )}
    </SearchContainer>
  )
}
```

Performance Requirements

- Audit trail loading under 1 second
- Search response under 500ms
- Export generation under 5 seconds
- Notification updates under 100ms
- Memory usage under 100MB

Story 5.2.3: Workspace Security Components

Overview

Build workspace security components that protect workspace data, ensure compliance, and provide comprehensive security monitoring and management capabilities.

Context

- Complete workspace utility component system
- Need security components for workspace protection
- Must support compliance and security requirements
- Security components are critical for enterprise use

- Need comprehensive security monitoring and management

Requirements

1. Build WorkspaceArchive component:

- Data archival and retention management
- Archive policy enforcement
- Archive search and retrieval
- Archive compliance reporting
- Archive security and encryption

2. Build WorkspaceIntegrations component:

- Third-party integration management
- Integration security monitoring
- API key and credential management
- Integration audit and logging
- Integration permission control

3. Build WorkspaceSecurity component:

- Security policy management
- Security monitoring and alerts
- Access control and authentication
- Security compliance reporting
- Incident response management

Specific Tasks

- ✅ Build WorkspaceArchive component
- ✅ Add data archival and retention
- ✅ Build WorkspaceIntegrations component
- ✅ Add integration security management
- ✅ Build WorkspaceSecurity component
- ✅ Add security monitoring and alerts
- ✅ Create security policy enforcement

- ✅ Add compliance reporting

Documentation Required

- Workspace security architecture
- Data archival and retention policies
- Integration security guidelines
- Security monitoring implementation
- Compliance reporting requirements
- Incident response procedures

Testing Requirements

- Security policy enforcement tests
- Data archival functionality tests
- Integration security tests
- Compliance reporting tests
- Security monitoring tests
- Incident response tests

Integration Points

- Integration with security monitoring systems
- Data archival service integration
- Integration management system
- Compliance reporting integration
- Incident response system integration

Deliverables

- WorkspaceArchive component
- WorkspaceIntegrations component
- WorkspaceSecurity component
- Security policy enforcement
- Compliance reporting system
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface WorkspaceArchiveProps {
  workspace: Workspace
  archives: Archive[]
  policies: RetentionPolicy[]
  onArchiveCreate?: (data: ArchiveData) => void
  onArchiveRestore?: (archive: Archive) => void
  onPolicyUpdate?: (policy: RetentionPolicy) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showPolicies?: boolean
  showCompliance?: boolean
  permissions?: string[]
}

interface WorkspaceIntegrationsProps {
  integrations: Integration[]
  availableIntegrations: IntegrationType[]
  onIntegrationAdd?: (type: IntegrationType) => void
  onIntegrationRemove?: (integration: Integration) => void
  onIntegrationConfigure?: (integration: Integration) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showSecurity?: boolean
  showAudit?: boolean
  permissions?: string[]
}

interface WorkspaceSecurityProps {
  workspace: Workspace
  securityStatus: SecurityStatus
  policies: SecurityPolicy[]
  incidents: SecurityIncident[]
  onPolicyUpdate?: (policy: SecurityPolicy) => void
  onIncidentRespond?: (incident: SecurityIncident) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showMonitoring?: boolean
  showIncidents?: boolean
  showCompliance?: boolean
  permissions?: string[]
}

interface Archive {
  id: string
  name: string
  description: string
  createdAt: Date
  size: number
  type: 'full' | 'incremental'
  status: 'active' | 'archived' | 'deleted'
  retentionPolicy: RetentionPolicy
  encryption: boolean
  workspaceId: string
}

interface Integration {
  id: string
  name: string
  type: IntegrationType
  status: 'active' | 'inactive' | 'error'
  configuration: Record<string, any>
  lastSync: Date
  permissions: string[]
  securityLevel: 'low' | 'medium' | 'high'
  workspaceId: string
}

interface SecurityStatus {
  overall: 'secure' | 'warning' | 'critical'
  score: number
  lastAssessment: Date
  vulnerabilities: SecurityVulnerability[]
  recommendations: SecurityRecommendation[]
}

interface SecurityIncident {
  id: string
  type: string
  severity: 'low' | 'medium' | 'high' | 'critical'
  status: 'open' | 'investigating' | 'resolved'
  description: string
  occurredAt: Date
  resolvedAt?: Date
  assignedTo?: User
  workspaceId: string
}
```

Implementation Example

jsx

```
// WorkspaceSecurity implementation
function WorkspaceSecurity({
  workspace,
  securityStatus,
  policies,
  incidents,
  showMonitoring = true,
  showIncidents = true,
  showCompliance = true
}) {
  const [activeTab, setActiveTab] = useState('monitoring')
  const [selectedIncident, setSelectedIncident] = useState(null)
  const [policyFilter, setPolicyFilter] = useState('all')

  const getStatusColor = (status) => {
    switch (status) {
      case 'secure': return 'success'
      case 'warning': return 'warning'
      case 'critical': return 'error'
      default: return 'neutral'
    }
  }

  const filteredPolicies = useMemo(() => {
    if (policyFilter === 'all') return policies
    return policies.filter(policy => policy.status === policyFilter)
  }, [policies, policyFilter])

  return (
    <SecurityContainer>
      <SecurityHeader>
        <HeaderInfo>
          <Heading level={2}>Workspace Security</Heading>
          <SecurityScore
            score={securityStatus.score}
            status={securityStatus.overall}
            color={getStatusColor(securityStatus.overall)}
          />
        </HeaderInfo>
        <LastAssessment>
          Last assessment: {formatDate(securityStatus.lastAssessment)}
        </LastAssessment>
      </SecurityHeader>
```

```
<SecurityTabs>
  {showMonitoring && (
    <Tab
      active={activeTab === 'monitoring'}
      onClick={() => setActiveTab('monitoring')}
    >
      <Icon name="shield" />
      Monitoring
    </Tab>
  )}
  {showIncidents && (
    <Tab
      active={activeTab === 'incidents'}
      onClick={() => setActiveTab('incidents')}
    >
      <Icon name="alert-triangle" />
      Incidents ({incidents.filter(i => i.status === 'open').length})
    </Tab>
  )}
  {showCompliance && (
    <Tab
      active={activeTab === 'compliance'}
      onClick={() => setActiveTab('compliance')}
    >
      <Icon name="check-circle" />
      Compliance
    </Tab>
  )}
  <Tab
    active={activeTab === 'policies'}
    onClick={() => setActiveTab('policies')}
  >
    <Icon name="file-text" />
    Policies
  </Tab>
</SecurityTabs>

<SecurityContent>
  {activeTab === 'monitoring' && showMonitoring && (
    <MonitoringPanel>
      <VulnerabilityList>
        <SectionHeader>
          <SectionTitle>Vulnerabilities</SectionTitle>
        <Badge variant="warning">
          {securityStatus.vulnerabilities.length}
        </Badge>
      </SectionHeader>
      {securityStatus.vulnerabilities.map(vuln => (
        <VulnerabilityItem key={vuln.id}>
          <VulnInfo>
            <VulnTitle>{vuln.title}</VulnTitle>
            <VulnDescription>{vuln.description}</VulnDescription>
          </VulnInfo>
          <VulnSeverity severity={vuln.severity}>
            {vuln.severity}
          </VulnSeverity>
        </VulnerabilityItem>
      ))}
    </VulnerabilityList>

    <RecommendationList>
      <SectionHeader>
        <SectionTitle>Recommendations</SectionTitle>
      </SectionHeader>
      {securityStatus.recommendations.map(rec => (
        <RecommendationItem key={rec.id}>
          <Icon name="info" />
          <RecText>{rec.text}</RecText>
          <Button size="sm" variant="secondary">
            Apply
          </Button>
        </RecommendationItem>
      ))}
    </RecommendationList>
  </MonitoringPanel>
)}

{activeTab === 'incidents' && showIncidents && (
  <IncidentsPanel>
    <IncidentFilters>
      <FilterButton
        active={true}
        onClick={() => {}}
      >
        All ({incidents.length})
      </FilterButton>
      <FilterButton
```

```
        active={false}
        onClick={() => {}}
      >
        Open ({incidents.filter(i => i.status === 'open').length})
      </FilterButton>
      <FilterButton
        active={false}
        onClick={() => {}}
      >
        Resolved ({incidents.filter(i => i.status === 'resolved').length})
      </FilterButton>
    </IncidentFilters>

    <IncidentList>
      {incidents.map(incident => (
        <IncidentCard
          key={incident.id}
          incident={incident}
          onClick={() => setSelectedIncident(incident)}
          selected={selectedIncident?.id === incident.id}
        />
      ))}
    </IncidentList>

    {selectedIncident && (
      <IncidentDetails
        incident={selectedIncident}
        onClose={() => setSelectedIncident(null)}
        onRespond={(response) => onIncidentRespond?.(selectedIncident)}
      />
    )}
  </IncidentsPanel>
)}

{activeTab === 'policies' && (
  <PoliciesPanel>
    <PolicyFilters>
      <Select
        value={policyFilter}
        onChange={setPolicyFilter}
        size="sm"
      >
        <Option value="all">All Policies</Option>
        <Option value="active">Active</Option>
```

```
              <Option value="pending">Pending</Option>
              <Option value="violated">Violated</Option>
            </Select>
          </PolicyFilters>

          <PolicyList>
            {filteredPolicies.map(policy => (
              <PolicyCard
                key={policy.id}
                policy={policy}
                onUpdate={() => onPolicyUpdate?.(policy)}
              />
            ))}
          </PolicyList>
        </PoliciesPanel>
      )}
    </SecurityContent>
  </SecurityContainer>
 )
}

// WorkspaceIntegrations implementation
function WorkspaceIntegrations({
  integrations,
  availableIntegrations,
  onIntegrationAdd,
  showSecurity = true,
  showAudit = true
}) {
  const [selectedIntegration, setSelectedIntegration] = useState(null)
  const [showAddModal, setShowAddModal] = useState(false)
  const [auditLog, setAuditLog] = useState([])

  const getSecurityBadge = (level) => {
    const variants = {
      low: 'success',
      medium: 'warning',
      high: 'error'
    }
    return variants[level] || 'neutral'
  }

  return (
    <IntegrationsContainer>
```

```
<IntegrationsHeader>
  <Heading level={3}>Integrations</Heading>
  <Button
    variant="primary"
    size="sm"
    onClick={() => setShowAddModal(true)}
  >
    <Icon name="plus" />
    Add Integration
  </Button>
</IntegrationsHeader>

<IntegrationGrid>
  {integrations.map(integration => (
    <IntegrationCard key={integration.id}>
      <IntegrationHeader>
        <IntegrationIcon src={integration.icon} />
        <IntegrationInfo>
          <IntegrationName>{integration.name}</IntegrationName>
          <IntegrationStatus status={integration.status}>
            {integration.status}
          </IntegrationStatus>
        </IntegrationInfo>
        {showSecurity && (
          <Badge variant={getSecurityBadge(integration.securityLevel)}>
            {integration.securityLevel} security
          </Badge>
        )}
      </IntegrationHeader>

      <IntegrationDetails>
        <DetailRow>
          <Label>Last Sync</Label>
          <Value>{formatDate(integration.lastSync)}</Value>
        </DetailRow>
        <DetailRow>
          <Label>Permissions</Label>
          <Value>{integration.permissions.length} granted</Value>
        </DetailRow>
      </IntegrationDetails>

      <IntegrationActions>
        <Button
          variant="secondary"
          size="sm"
          onClick={() => setSelectedIntegration(integration)}
        >
          Configure
        </Button>
        <Button
          variant="ghost"
          size="sm"
          onClick={() => onIntegrationRemove?.(integration)}
        >
          Remove
        </Button>
      </IntegrationActions>

      {showAudit && (
        <IntegrationAudit>
          <AuditLink
            onClick={() => loadIntegrationAudit(integration.id)}
          >
            View audit log
          </AuditLink>
        </IntegrationAudit>
      )}
    </IntegrationCard>
  ))}
</IntegrationGrid>

{showAddModal && (
  <AddIntegrationModal
    availableIntegrations={availableIntegrations}
    onAdd={(type) => {
      onIntegrationAdd?.(type)
      setShowAddModal(false)
    }}
    onClose={() => setShowAddModal(false)}
  />
)}

{selectedIntegration && (
  <IntegrationConfigModal
    integration={selectedIntegration}
    onSave={(config) => {
      onIntegrationConfigure?.(selectedIntegration, config)
      setSelectedIntegration(null)
```

```
          }}
          onClose={() => setSelectedIntegration(null)}
        />
      )}
    </IntegrationsContainer>
  )
}
```

Performance Requirements

- Security scan under 30 seconds
- Archive operations under 10 seconds
- Integration sync under 5 seconds
- Compliance report generation under 15 seconds
- Memory usage under 200MB

Performance Optimization

Management Components

- Context switching optimization
- Permission caching strategies
- Route preloading
- State persistence optimization
- Audit log pagination

Utility Components

- Search index optimization
- Notification batching
- Export streaming for large datasets
- Audit trail virtualization
- Real-time update throttling

Security Components

- Background security scanning
- Incremental compliance checks
- Archive compression strategies

- Integration health monitoring
- Incident response automation

Accessibility Requirements

WCAG 2.1 AA Compliance

- Keyboard navigation for all management controls
- Screen reader support for security alerts
- Focus management for complex workflows
- High contrast mode for security dashboards
- Clear labeling for all administrative functions

Security Accessibility

- Alternative formats for security reports
- Accessible audit trail navigation
- Keyboard shortcuts for incident response
- Visual and audio alerts for security events
- Simplified views for reduced cognitive load

Security Considerations

Data Protection

- Encryption for archived data
- Secure credential storage
- API key rotation policies
- Audit log protection
- Compliance data handling

Access Control

- Granular permission management
- Role-based security policies
- Multi-factor authentication support
- Session management
- IP whitelisting capabilities

Testing Strategy

Unit Tests

- Permission validation testing
- Security policy enforcement
- Archive functionality
- Integration security
- Utility function testing

Integration Tests

- Context switching workflows
- Security scanning pipelines
- Archive and restore operations
- Integration synchronization
- Compliance reporting

E2E Tests

- Complete security workflows
- Archive lifecycle testing
- Integration setup flows
- Incident response procedures
- Compliance audit trails

Storybook Documentation

Management Stories

- Context provider examples
- Router configuration
- Permission management
- State persistence
- Audit logging

Utility Stories

- Audit trail displays

- Search interfaces
- Export configurations
- Notification management
- Real-time updates

Security Stories

- Security dashboards
- Archive management
- Integration security
- Incident response
- Compliance reporting

Migration Guide

From Legacy Management

1. Map existing permissions
2. Migrate workspace contexts
3. Update routing configuration
4. Configure security policies
5. Test compliance features

Breaking Changes

- New context provider API
- Updated permission model
- Changed routing structure
- Modified security interfaces
- New compliance requirements