Epic 5.1: Workspace Foundation Components

Epic Overview

Workspace Foundation Components are the core building blocks that enable multi-tenant workspace functionality in THE WHEEL design system. These components handle workspace identification, branding, status management, collaboration features, and administrative functions across all six workspace contexts (consultant, client, admin, expert, tool creator, founder).

Epic Goals:

- Create workspace identity and branding components
- Implement workspace status and monitoring systems
- Build collaboration infrastructure for teams
- Enable workspace settings and configuration
- Provide invitation and onboarding workflows

Story 5.1.1: Workspace Identity Components

Overview

Build workspace identity components that handle workspace identification, branding, and identity management across different contexts while maintaining multi-tenant isolation.

Context

- Complete component library with workspace-specific features
- Need components for workspace identification and branding
- Must support multi-tenant workspace customization
- Identity components are critical for workspace branding
- Need consistent identity across all workspace applications

Requirements

1. Build WorkspaceIcon component:

- Workspace identification through icons
- Custom icon upload and management
- Icon variants for different contexts

- Responsive icon sizing
- Accessibility features

2. Build ClientBadge component:

- Client identification and branding
- Badge variants for different client types
- Client status indicators
- Permission-based badge display
- Responsive badge behavior

3. Build WorkspaceTheme component:

- Workspace theme customization
- Brand color management
- Theme preview and application
- Theme inheritance and overrides
- Real-time theme updates

Specific Tasks

- ✅ Build WorkspaceIcon component
- ✅ Add workspace identification logic
- ✅ Build ClientBadge component
- ✅ Add client identification features
- ✅ Build WorkspaceTheme component
- ✅ Add theme customization
- ✅ Implement brand asset management
- ✅ Add real-time theme updates

Documentation Required

- Workspace identity system architecture
- Brand asset management guidelines
- Theme customization patterns
- Identity component usage
- Accessibility implementation

- Multi-tenant considerations

Testing Requirements

- Workspace identification tests
- Client badge functionality tests
- Theme customization tests
- Brand asset management tests
- Real-time update tests
- Accessibility compliance tests
- Multi-tenant isolation tests

Integration Points

- Integration with workspace context providers
- Brand asset management integration
- Theme system integration
- Real-time update system integration
- Multi-tenant isolation integration

Deliverables

- WorkspaceIcon component with identification
- ClientBadge component with branding
- WorkspaceTheme component with customization
- Brand asset management system
- Real-time theme update system
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface WorkspaceIconProps {
  workspace: Workspace
  size?: 'xs' | 'sm' | 'md' | 'lg' | 'xl' | '2xl'
  variant?: 'icon' | 'logo' | 'initials'
  shape?: 'circle' | 'square' | 'rounded'
  editable?: boolean
  onEdit?: (icon: WorkspaceIcon) => void
  fallback?: React.ReactNode
  responsive?: boolean
  permissions?: string[]
}

interface ClientBadgeProps {
  client: Client
  variant?: 'full' | 'compact' | 'minimal'
  size?: 'sm' | 'md' | 'lg'
  showStatus?: boolean
  showType?: boolean
  editable?: boolean
  onEdit?: (client: Client) => void
  onClick?: (client: Client) => void
  permissions?: string[]
}

interface WorkspaceThemeProps {
  workspace: Workspace
  currentTheme?: WorkspaceTheme
  onThemeChange?: (theme: WorkspaceTheme) => void
  onThemeApply?: (theme: WorkspaceTheme) => void
  previewMode?: boolean
  editable?: boolean
  templates?: ThemeTemplate[]
  permissions?: string[]
}

interface WorkspaceIcon {
  id: string
  type: 'image' | 'initials' | 'emoji'
  value: string
  backgroundColor?: string
  foregroundColor?: string
  size?: number
  workspace?: string
}

interface WorkspaceTheme {
  id: string
  name: string
  primaryColor: string
  secondaryColor: string
  accentColor: string
  backgroundColor: string
  textColor: string
  borderColor: string
  typography?: Typography
  spacing?: Spacing
  borderRadius?: BorderRadius
  shadows?: Shadows
  workspace?: string
}

interface ThemeTemplate {
  id: string
  name: string
  description?: string
  theme: WorkspaceTheme
  preview?: string
  category?: string
}
```

Implementation Example

jsx

```
// WorkspaceIcon implementation
function WorkspaceIcon({ workspace, size = 'md', variant = 'icon', editable }) {
  const [isEditing, setIsEditing] = useState(false)
  const { hasPermission } = useWorkspace()
  const canEdit = editable && hasPermission('workspace:edit')

  const renderIcon = () => {
    switch (workspace.icon.type) {
      case 'image':
        return (
          <IconImage
            src={workspace.icon.value}
            alt={workspace.name}
            size={size}
          />
        )
      case 'initials':
        return (
          <IconInitials size={size}>
            {workspace.icon.value}
          </IconInitials>
        )
      case 'emoji':
        return (
          <IconEmoji size={size}>
            {workspace.icon.value}
          </IconEmoji>
        )
      default:
        return <DefaultIcon size={size} />
    }
  }

  return (
    <IconContainer
      size={size}
      shape={shape}
      onClick={canEdit ? () => setIsEditing(true) : undefined}
      editable={canEdit}
    >
      {renderIcon()}
      {canEdit && (
        <EditOverlay>
          <IconButton icon="edit" size="xs" />
        </EditOverlay>
      )}
      {isEditing && (
        <IconEditor
          workspace={workspace}
          onSave={handleIconUpdate}
          onCancel={() => setIsEditing(false)}
        />
      )}
    </IconContainer>
  )
}
```

Performance Requirements

- Icon loading under 50ms
- Theme application under 300ms
- Badge rendering under 16ms
- Memory usage under 10MB
- Real-time updates under 100ms

Story 5.1.2: Workspace Status Components

Overview

Build workspace status components that communicate workspace state and status information across different contexts with real-time updates.

Context

- Workspace identity components with branding and customization
- Need status communication across workspace applications
- Must support real-time status updates
- Status components are critical for workspace collaboration
- Need consistent status representation across contexts

Requirements

1. Build BillingStatus component:

- Billing state indicators (active, overdue, suspended)
- Payment status display
- Subscription information
- Usage metrics and limits
- Action buttons for billing management

2. Build TimeIndicator component:

- Time tracking display
- Current session tracking
- Time zone awareness
- Time format preferences
- Real-time updates

3. Build ProjectPhase component:

- Project status indicators
- Phase progress visualization
- Milestone tracking
- Phase transition actions
- Team collaboration indicators

Specific Tasks

- ✅ Build BillingStatus component
- ✅ Add billing state indicators
- ✅ Build TimeIndicator component
- ✅ Add time tracking display
- ✅ Build ProjectPhase component
- ✅ Add project status indicators
- ✅ Implement real-time updates
- ✅ Add status transition logic

Documentation Required

- Workspace status system architecture
- Status indicator patterns

- Real-time update implementation
- Status transition workflows
- Accessibility guidelines
- Multi-tenant status handling

Testing Requirements

- Billing status tests
- Time indicator tests
- Project phase tests
- Real-time update tests
- Status transition tests
- Accessibility compliance tests
- Multi-tenant isolation tests

Integration Points

- Integration with workspace context providers
- Billing system integration
- Time tracking integration
- Project management integration
- Real-time update system integration

Deliverables

- BillingStatus component with state indicators
- TimeIndicator component with tracking
- ProjectPhase component with status
- Real-time update system
- Status transition logic
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface BillingStatusProps {
  workspace: Workspace
  billingInfo: BillingInfo
  onAction?: (action: BillingAction) => void
  variant?: 'full' | 'compact' | 'minimal'
  showActions?: boolean
  realTimeUpdates?: boolean
  permissions?: string[]
}

interface TimeIndicatorProps {
  workspace: Workspace
  currentSession?: TimeSession
  format?: '12h' | '24h'
  timezone?: string
  showSession?: boolean
  showElapsed?: boolean
  onStartSession?: () => void
  onStopSession?: () => void
  onPauseSession?: () => void
  realTimeUpdates?: boolean
  permissions?: string[]
}

interface ProjectPhaseProps {
  project: Project
  currentPhase?: ProjectPhase
  onPhaseChange?: (phase: ProjectPhase) => void
  onMilestoneClick?: (milestone: Milestone) => void
  variant?: 'full' | 'compact' | 'minimal'
  showProgress?: boolean
  showMilestones?: boolean
  showActions?: boolean
  permissions?: string[]
}

interface BillingInfo {
  status: 'active' | 'overdue' | 'suspended' | 'cancelled'
  plan: BillingPlan
  usage: UsageMetrics
  nextBillingDate?: Date
  lastPayment?: Payment
  outstandingBalance?: number
  warnings?: BillingWarning[]
}

interface TimeSession {
  id: string
  startTime: Date
  endTime?: Date
  elapsed: number
  status: 'active' | 'paused' | 'completed'
  project?: Project
  client?: Client
  description?: string
}

interface ProjectPhase {
  id: string
  name: string
  description?: string
  status: 'upcoming' | 'current' | 'completed' | 'on_hold'
  startDate?: Date
  endDate?: Date
  progress: number
  milestones: Milestone[]
  dependencies?: string[]
}
```

Implementation Example

jsx

```
// BillingStatus implementation
function BillingStatus({ workspace, billingInfo, showActions, variant = 'full' }) {
  const { hasPermission } = useWorkspace()
  const [showDetails, setShowDetails] = useState(false)

  const getStatusColor = (status) => {
    switch (status) {
      case 'active': return 'success'
      case 'overdue': return 'warning'
      case 'suspended': return 'error'
      default: return 'neutral'
    }
  }

  if (variant === 'minimal') {
    return (
      <StatusBadge
        variant={getStatusColor(billingInfo.status)}
        onClick={() => setShowDetails(true)}
      >
        {billingInfo.status}
      </StatusBadge>
    )
  }

  return (
    <BillingStatusContainer variant={variant}>
      <StatusHeader>
        <StatusIcon status={billingInfo.status} />
        <StatusInfo>
          <StatusLabel>Billing Status</StatusLabel>
          <StatusValue color={getStatusColor(billingInfo.status)}>
            {billingInfo.status.charAt(0).toUpperCase() + billingInfo.status.slice(1)}
          </StatusValue>
        </StatusInfo>
      </StatusHeader>

      {variant === 'full' && (
        <>
          <BillingDetails>
            <DetailRow>
              <Label>Current Plan</Label>
              <Value>{billingInfo.plan.name}</Value>
    </DetailRow>
    <DetailRow>
      <Label>Next Billing</Label>
      <Value>{formatDate(billingInfo.nextBillingDate)}</Value>
    </DetailRow>
    {billingInfo.outstandingBalance > 0 && (
      <DetailRow highlight>
        <Label>Outstanding Balance</Label>
        <Value>${billingInfo.outstandingBalance.toFixed(2)}</Value>
      </DetailRow>
    )}
  </BillingDetails>

  <UsageMetrics>
    <UsageBar
      label="API Calls"
      current={billingInfo.usage.apiCalls}
      limit={billingInfo.plan.limits.apiCalls}
    />
    <UsageBar
      label="Storage"
      current={billingInfo.usage.storage}
      limit={billingInfo.plan.limits.storage}
      unit="GB"
    />
  </UsageMetrics>

  {showActions && hasPermission('billing:manage') && (
    <BillingActions>
      <Button
        variant="primary"
        size="sm"
        onClick={() => handleAction('pay')}
      >
        Make Payment
      </Button>
      <Button
        variant="secondary"
        size="sm"
        onClick={() => handleAction('upgrade')}
      >
        Upgrade Plan
      </Button>
    </BillingActions>
```

```
          )}
        </>
      )}
    </BillingStatusContainer>
  )
}
```

Performance Requirements

- Status loading under 100ms
- Real-time updates under 200ms
- Memory usage under 20MB
- Status transitions under 300ms
- Animation performance at 60fps

Story 5.1.3: Workspace Collaboration Components

Overview

Build workspace collaboration components that facilitate collaboration and communication within workspace contexts, including privacy management and expertise identification.

Context

- Workspace status components with real-time updates
- Need collaboration features for team productivity
- Must support real-time presence and communication
- Collaboration components are critical for workspace success
- Need privacy and consent management features

Requirements

1. Build CollaboratorAvatar component:

- Team member presence indicators
- Real-time collaboration cursors
- User status and availability
- Permission-based visibility
- Interactive collaboration features

2. Build ConsentToggle component:

- Privacy consent management
- GDPR compliance features
- Granular permission controls
- Consent tracking and audit
- User-friendly consent interfaces

3. Build DocumentType component:

- Document classification system
- Access control indicators
- Document metadata display
- Type-specific actions
- Collaboration permissions

4. Build ExpertiseTag component:

- Skill and expertise identification
- Expert matching capabilities
- Expertise validation
- Skill-based filtering
- Professional networking features

Specific Tasks

- ✅ Build CollaboratorAvatar component
- ✅ Add presence indicators
- ✅ Build ConsentToggle component
- ✅ Add consent management
- ✅ Build DocumentType component
- ✅ Add document classification
- ✅ Build ExpertiseTag component
- ✅ Add skill identification
- ✅ Implement real-time updates

Documentation Required

- Collaboration system architecture
- Presence and real-time features
- Consent and privacy management
- Document classification system
- Expertise and skill management
- Accessibility guidelines

Testing Requirements

- Collaboration feature tests
- Presence indicator tests
- Consent management tests
- Document classification tests
- Expertise system tests
- Real-time update tests
- Privacy compliance tests

Integration Points

- Integration with workspace context providers
- Real-time collaboration system
- Consent management system
- Document management integration
- Expertise matching system

Deliverables

- CollaboratorAvatar with presence indicators
- ConsentToggle with privacy management
- DocumentType with classification
- ExpertiseTag with skill identification
- Real-time collaboration features
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface CollaboratorAvatarProps {
  user: User
  size?: 'xs' | 'sm' | 'md' | 'lg' | 'xl'
  showPresence?: boolean
  showStatus?: boolean
  showCollaboration?: boolean
  onUserClick?: (user: User) => void
  onCollaborationClick?: (user: User) => void
  permissions?: string[]
  realTimeUpdates?: boolean
  workspace?: string
}

interface ConsentToggleProps {
  user: User
  consentType: ConsentType
  currentConsent?: Consent
  onConsentChange?: (consent: Consent) => void
  variant?: 'toggle' | 'checkbox' | 'button'
  showDetails?: boolean
  required?: boolean
  workspace?: string
  permissions?: string[]
}

interface DocumentTypeProps {
  document: Document
  onTypeChange?: (type: DocumentType) => void
  onAccessChange?: (access: AccessLevel) => void
  variant?: 'full' | 'compact' | 'minimal'
  showActions?: boolean
  showMetadata?: boolean
  permissions?: string[]
  workspace?: string
}

interface ExpertiseTagProps {
  expertise: Expertise
  user?: User
  onExpertiseClick?: (expertise: Expertise) => void
  onUserClick?: (user: User) => void
  variant?: 'tag' | 'card' | 'list'
  showLevel?: boolean
  showValidation?: boolean
  editable?: boolean
  permissions?: string[]
}

interface ConsentType {
  id: string
  name: string
  description: string
  category: 'privacy' | 'marketing' | 'analytics' | 'functional'
  required: boolean
  defaultValue: boolean
  legalBasis?: string
  dataProcessing?: string[]
  retentionPeriod?: string
}

interface Expertise {
  id: string
  name: string
  category: string
  level: 'beginner' | 'intermediate' | 'advanced' | 'expert'
  validated: boolean
  validatedBy?: User
  validatedAt?: Date
  description?: string
  certifications?: Certification[]
}
```

Implementation Example

jsx

```
// CollaboratorAvatar implementation
function CollaboratorAvatar({
  user,
  size = 'md',
  showPresence = true,
  showStatus = true,
  realTimeUpdates = true
}) {
  const [presence, setPresence] = useState(user.presence)
  const { socket } = useRealTimeCollaboration()

  useEffect(() => {
    if (realTimeUpdates) {
      const handlePresenceUpdate = (update) => {
        if (update.userId === user.id) {
          setPresence(update.presence)
        }
      }

      socket.on('presence:update', handlePresenceUpdate)
      return () => socket.off('presence:update', handlePresenceUpdate)
    }
  }, [socket, user.id, realTimeUpdates])

  return (
    <AvatarContainer size={size}>
      <AvatarImage
        src={user.avatar}
        alt={user.name}
        size={size}
      />

      {showPresence && (
        <PresenceIndicator
          status={presence.status}
          size={size}
        />
      )}

      {showStatus && user.statusMessage && (
        <StatusTooltip>
          <StatusMessage>{user.statusMessage}</StatusMessage>
          <StatusTime>{formatRelativeTime(user.statusUpdatedAt)}</StatusTime>
        </StatusTooltip>
      )}

      <CollaborationIndicators>
        {presence.isTyping && <TypingIndicator />}
        {presence.isViewing && <ViewingIndicator page={presence.viewingPage} />}
        {presence.cursor && <CursorIndicator position={presence.cursor} />}
      </CollaborationIndicators>
    </AvatarContainer>
  )
}
```

Performance Requirements

- Avatar loading under 50ms
- Presence updates under 100ms
- Memory usage under 15MB
- Consent operations under 200ms
- Real-time sync under 150ms

Story 5.1.4: Workspace Settings Components

Overview

Create workspace settings components that provide comprehensive workspace configuration interfaces for different workspace contexts and roles.

Context

- Existing workspace management system with context switching
- Need for granular workspace configuration
- Multiple workspace contexts requiring different settings
- Real-time collaboration requiring settings synchronization
- Integration with existing theming and permission systems

Requirements

1. Create workspace settings components:

- General workspace configuration panel
- Theme and branding settings

- Permission and access control settings
- Integration and API settings
- Notification and communication settings

2. Implement workspace context features:

- Context-specific setting options
- Role-based settings access
- Workspace-specific setting validation
- Context-aware setting recommendations
- Brand-aware settings interface

3. Create settings management features:

- Settings backup and restore
- Settings templates and presets
- Settings version control
- Settings audit and logging
- Settings migration and updates

Specific Tasks

- ✅ Create WorkspaceSettings component
- ✅ Implement GeneralSettings component
- ✅ Set up ThemeSettings component
- ✅ Create PermissionSettings component
- ✅ Implement settings backup
- ✅ Set up settings validation

Documentation Required

- Workspace settings API documentation
- Settings configuration guide
- Permission management guide
- Theme customization guide
- Settings migration guide

Testing Requirements

- Settings functionality tests
- Permission validation tests
- Theme application tests
- Settings backup tests
- Migration and update tests

Integration Points

- Integration with workspace management system
- Theme system integration
- Permission system integration
- Real-time collaboration integration
- Analytics and audit integration

Deliverables

- Complete workspace settings system
- Permission and access control
- Theme and branding configuration
- Settings backup and restore
- Comprehensive settings documentation

Component Specifications

typescript

```
interface WorkspaceSettingsProps {
  workspace: Workspace
  currentSettings: WorkspaceSettings
  onSettingsChange?: (settings: WorkspaceSettings) => void
  onSave?: () => void
  onCancel?: () => void
  variant?: 'tabs' | 'sidebar' | 'accordion'
  showAdvanced?: boolean
  permissions?: string[]
}

interface GeneralSettingsProps {
  workspace: Workspace
  settings: GeneralSettings
  onChange?: (settings: GeneralSettings) => void
  permissions?: string[]
}

interface ThemeSettingsProps {
  workspace: Workspace
  currentTheme: WorkspaceTheme
  themes: ThemeTemplate[]
  onChange?: (theme: WorkspaceTheme) => void
  onCustomize?: () => void
  showPreview?: boolean
  permissions?: string[]
}

interface PermissionSettingsProps {
  workspace: Workspace
  roles: Role[]
  permissions: Permission[]
  users: User[]
  onRoleChange?: (role: Role) => void
  onPermissionChange?: (permission: Permission) => void
  onUserPermissionChange?: (user: User, permissions: string[]) => void
  showInheritance?: boolean
  permissions?: string[]
}

interface WorkspaceSettings {
  general: GeneralSettings
  theme: WorkspaceTheme
  permissions: PermissionSettings
  integrations: IntegrationSettings
  notifications: NotificationSettings
  advanced: AdvancedSettings
}

interface GeneralSettings {
  name: string
  description?: string
  timezone: string
  language: string
  currency: string
  dateFormat: string
  timeFormat: '12h' | '24h'
  workingDays: number[]
  businessHours: BusinessHours
}
```

Implementation Example

jsx

```
// WorkspaceSettings implementation
function WorkspaceSettings({
  workspace,
  currentSettings,
  onSettingsChange,
  variant = 'tabs'
}) {
  const [settings, setSettings] = useState(currentSettings)
  const [activeSection, setActiveSection] = useState('general')
  const [hasChanges, setHasChanges] = useState(false)
  const { hasPermission } = useWorkspace()

  const handleSettingChange = (section, value) => {
    const newSettings = {
      ...settings,
      [section]: value
    }
    setSettings(newSettings)
    setHasChanges(true)
    onSettingsChange?.(newSettings)
  }

  const sections = [
    {
      id: 'general',
      label: 'General',
      icon: 'settings',
      component: GeneralSettings,
      permission: 'settings:general'
    },
    {
      id: 'theme',
      label: 'Theme & Branding',
      icon: 'palette',
      component: ThemeSettings,
      permission: 'settings:theme'
    },
    {
      id: 'permissions',
      label: 'Permissions',
      icon: 'shield',
      component: PermissionSettings,
      permission: 'settings:permissions'
  },
  {
    id: 'integrations',
    label: 'Integrations',
    icon: 'plug',
    component: IntegrationSettings,
    permission: 'settings:integrations'
  },
  {
    id: 'notifications',
    label: 'Notifications',
    icon: 'bell',
    component: NotificationSettings,
    permission: 'settings:notifications'
  }
]

const availableSections = sections.filter(section =>
  hasPermission(section.permission)
)

return (
  <SettingsContainer>
    <SettingsHeader>
      <Heading level={2}>Workspace Settings</Heading>
      {hasChanges && (
        <HeaderActions>
          <Button
            variant="secondary"
            onClick={handleCancel}
          >
            Cancel
          </Button>
          <Button
            variant="primary"
            onClick={handleSave}
          >
            Save Changes
          </Button>
        </HeaderActions>
      )}
    </SettingsHeader>

    {variant === 'tabs' ? (
  <TabNavigation>
    {availableSections.map(section => (
      <Tab
        key={section.id}
        active={activeSection === section.id}
        onClick={() => setActiveSection(section.id)}
      >
        <Icon name={section.icon} />
        {section.label}
      </Tab>
    ))}
  </TabNavigation>
) : (
  <SettingsSidebar>
    {availableSections.map(section => (
      <SidebarItem
        key={section.id}
        active={activeSection === section.id}
        onClick={() => setActiveSection(section.id)}
      >
        <Icon name={section.icon} />
        {section.label}
      </SidebarItem>
    ))}
  </SettingsSidebar>
)}

<SettingsContent>
  {availableSections.map(section => {
    const SectionComponent = section.component
    return (
      <SectionPanel
        key={section.id}
        active={activeSection === section.id}
      >
        <SectionComponent
          workspace={workspace}
          settings={settings[section.id]}
          onChange={(value) => handleSettingChange(section.id, value)}
        />
      </SectionPanel>
    )
  })}
</SettingsContent>
```

```
    </SettingsContainer>
  )
}
```

Performance Requirements

- Settings loading under 1 second
- Settings update under 500ms
- Theme application under 300ms
- Memory usage under 40MB
- Settings validation under 200ms

Story 5.1.5: Workspace Invitation Components

Overview

Create workspace invitation components that handle user invitation and onboarding workflows across different workspace contexts with role-based permissions.

Context

- Workspace settings system established with permission management
- Need for user invitation and onboarding workflows
- Multiple workspace contexts requiring different invitation flows
- Real-time collaboration requiring invitation coordination
- Integration with existing user management and authentication

Requirements

1. Create invitation components:

- Invitation creation and customization
- Invitation sending and tracking
- Invitation acceptance and onboarding
- Bulk invitation management
- Invitation expiration and resending

2. Implement workspace context features:

- Context-specific invitation templates

- Role-based invitation permissions
- Workspace-specific invitation flows
- Context-aware invitation validation
- Brand-aware invitation styling

3. Create invitation management features:

- Invitation analytics and tracking
- Invitation history and auditing
- Invitation automation and triggers
- Invitation security and verification
- Invitation integration with external systems

Specific Tasks

- ✅ Create InvitationManager component
- ✅ Implement InvitationForm component
- ✅ Set up InvitationTracker component
- ✅ Create BulkInvitation component
- ✅ Implement invitation onboarding
- ✅ Set up invitation analytics

Documentation Required

- Invitation system API documentation
- Invitation flow configuration
- Security implementation guide
- Analytics and tracking
- Integration guide

Testing Requirements

- Invitation functionality tests
- Security and verification tests
- Onboarding workflow tests
- Analytics tracking tests
- Integration tests

Integration Points

- Integration with workspace settings system
- User management integration
- Authentication system integration
- Real-time collaboration integration
- Analytics and tracking integration

Deliverables

- Complete invitation system
- Onboarding workflow automation
- Security and verification features
- Analytics and tracking
- Comprehensive invitation documentation

Component Specifications

typescript

```
interface InvitationManagerProps {
  workspace: Workspace
  invitations: Invitation[]
  onInviteCreate?: (invitation: Invitation) => void
  onInviteResend?: (invitationId: string) => void
  onInviteCancel?: (invitationId: string) => void
  showPending?: boolean
  showAccepted?: boolean
  showExpired?: boolean
  permissions?: string[]
}

interface InvitationFormProps {
  workspace: Workspace
  onSubmit?: (invitations: InvitationData[]) => void
  onCancel?: () => void
  templates?: InvitationTemplate[]
  defaultRole?: string
  allowBulk?: boolean
  maxInvitations?: number
  permissions?: string[]
}

interface BulkInvitationProps {
  workspace: Workspace
  onSubmit?: (invitations: BulkInvitationData) => void
  onCancel?: () => void
  csvTemplate?: string
  maxRecords?: number
  validation?: InvitationValidation
  permissions?: string[]
}

interface InvitationTrackerProps {
  invitations: Invitation[]
  onInvitationClick?: (invitation: Invitation) => void
  onStatusFilterChange?: (status: InvitationStatus[]) => void
  showAnalytics?: boolean
  showTimeline?: boolean
  permissions?: string[]
}

interface Invitation {
  id: string
  email: string
  role: string
  workspace: string
  invitedBy: User
  invitedAt: Date
  expiresAt: Date
  acceptedAt?: Date
  status: InvitationStatus
  metadata?: InvitationMetadata
  customMessage?: string
}

interface InvitationData {
  email: string
  role: string
  customMessage?: string
  metadata?: Record<string, any>
}

interface InvitationStatus {
  status: 'pending' | 'accepted' | 'expired' | 'cancelled'
  timestamp: Date
  reason?: string
}
```

Implementation Example

jsx

```
// InvitationForm implementation
function InvitationForm({
  workspace,
  onSubmit,
  templates,
  allowBulk = true,
  maxInvitations = 10
}) {
  const [invitations, setInvitations] = useState([{ email: '', role: 'member' }])
  const [selectedTemplate, setSelectedTemplate] = useState(null)
  const [customMessage, setCustomMessage] = useState('')
  const { hasPermission } = useWorkspace()

  const availableRoles = workspace.roles.filter(role =>
    hasPermission(`invite:role:${role.id}`)
  )

  const addInvitation = () => {
    if (invitations.length < maxInvitations) {
      setInvitations([...invitations, { email: '', role: 'member' }])
    }
  }

  const removeInvitation = (index) => {
    setInvitations(invitations.filter((_, i) => i !== index))
  }

  const updateInvitation = (index, field, value) => {
    const updated = [...invitations]
    updated[index] = { ...updated[index], [field]: value }
    setInvitations(updated)
  }

  const handleSubmit = async () => {
    const validInvitations = invitations.filter(inv =>
      isValidEmail(inv.email) && inv.role
    )

    const invitationData = validInvitations.map(inv => ({
      ...inv,
      customMessage: customMessage || selectedTemplate?.message,
      metadata: {
        workspace: workspace.id,
        invitedVia: 'manual',
        template: selectedTemplate?.id
      }
    }))

    await onSubmit(invitationData)
  }

  return (
    <InvitationFormContainer>
      <FormHeader>
        <Heading level={3}>Invite Team Members</Heading>
        <Text variant="body" color="muted">
          Invite people to join {workspace.name}
        </Text>
      </FormHeader>

      {templates && templates.length > 0 && (
        <TemplateSelector>
          <Label>Use Template</Label>
          <Select
            value={selectedTemplate?.id}
            onChange={(value) => {
              const template = templates.find(t => t.id === value)
              setSelectedTemplate(template)
              if (template) {
                setCustomMessage(template.message)
              }
            }}
            placeholder="Select a template"
          >
            {templates.map(template => (
              <Option key={template.id} value={template.id}>
                {template.name}
              </Option>
            ))}
          </Select>
        </TemplateSelector>
      )}

      <InvitationList>
        {invitations.map((invitation, index) => (
          <InvitationRow key={index}>
            <EmailInput
        type="email"
        value={invitation.email}
        onChange={(e) => updateInvitation(index, 'email', e.target.value)}
        placeholder="email@example.com"
        error={invitation.email && !isValidEmail(invitation.email)}
      />
      <RoleSelect
        value={invitation.role}
        onChange={(value) => updateInvitation(index, 'role', value)}
      >
        {availableRoles.map(role => (
          <Option key={role.id} value={role.id}>
            {role.name}
          </Option>
        ))}
      </RoleSelect>
      {invitations.length > 1 && (
        <IconButton
          icon="trash"
          size="sm"
          onClick={() => removeInvitation(index)}
        />
      )}
    </InvitationRow>
  ))}
</InvitationList>

{allowBulk && invitations.length < maxInvitations && (
  <Button
    variant="secondary"
    size="sm"
    onClick={addInvitation}
  >
    <Icon name="plus" />
    Add Another
  </Button>
)}

<MessageSection>
  <Label>Custom Message (Optional)</Label>
  <Textarea
    value={customMessage}
    onChange={(e) => setCustomMessage(e.target.value)}
    placeholder="Add a personal message to your invitation..."
```

```
        rows={4}
      />
    </MessageSection>

    <FormActions>
      <Button variant="secondary" onClick={onCancel}>
        Cancel
      </Button>
      <Button
        variant="primary"
        onClick={handleSubmit}
        disabled={!invitations.some(inv => isValidEmail(inv.email))}
      >
        Send Invitations
      </Button>
    </FormActions>
  </InvitationFormContainer>
 )
}
```

Performance Requirements

- Invitation creation under 2 seconds
- Invitation sending under 5 seconds
- Onboarding flow under 30 seconds
- Memory usage under 30MB
- Analytics processing under 1 second

Performance Optimization

Component Loading

- Lazy loading for workspace settings
- Code splitting for invitation flows
- Optimistic UI updates
- Caching for workspace data
- Progressive enhancement

Real-time Features

- WebSocket connection pooling

- Presence update batching
- Efficient state synchronization
- Selective component updates
- Memory leak prevention

Accessibility Requirements

WCAG 2.1 AA Compliance

- Keyboard navigation for all workspace controls
- Screen reader announcements for status changes
- Focus management for modal workflows
- High contrast mode support
- Clear labeling for all interactive elements

Workspace Accessibility

- Alternative text for workspace icons
- Status announcements for screen readers
- Keyboard shortcuts for common actions
- Accessible forms and settings
- Support for reduced motion

Security Considerations

Multi-tenant Security

- Workspace isolation enforcement
- Permission-based access control
- Secure invitation tokens
- Data encryption at rest
- Audit logging for compliance

Privacy Management

- GDPR-compliant consent management
- Data retention policies
- User data export capabilities

- Right to erasure implementation
- Privacy settings management

Testing Strategy

Unit Tests

- Component functionality testing
- Permission validation testing
- Real-time update handling
- State management testing
- Utility function testing

Integration Tests

- Workspace switching flows
- Invitation and onboarding
- Settings synchronization
- Real-time collaboration
- Multi-tenant isolation

E2E Tests

- Complete workspace setup
- User invitation flows
- Settings management
- Collaboration scenarios
- Permission workflows

Storybook Documentation

Identity Component Stories

- Workspace icon variations
- Client badge examples
- Theme customization demo
- Brand asset management
- Real-time theme updates

Status Component Stories

- Billing status displays
- Time tracking examples
- Project phase indicators
- Status transitions
- Real-time updates

Collaboration Stories

- Presence indicators
- Consent management
- Document classification
- Expertise tagging
- Real-time collaboration

Settings Stories

- Settings panel layouts
- Permission management
- Theme configuration
- Integration settings
- Backup and restore

Invitation Stories

- Invitation forms
- Bulk invitations
- Onboarding flows
- Tracking dashboard
- Analytics display

Migration Guide

From Legacy Workspace System

1. Map existing workspace data
2. Migrate user permissions

3. Update branding assets
4. Configure new settings
5. Test workspace isolation

Breaking Changes

- New workspace context structure
- Updated permission model
- Changed theme system
- Modified invitation flow
- New real-time architecture