Epic 4.4: Communication Organisms

Epic Overview

Communication Organisms are complex components that enable real-time collaboration and communication within THE WHEEL design system. These organisms combine multiple molecules and atoms to create comprehensive communication experiences that adapt to different workspace contexts and support team collaboration.

Epic Goals:

- Create real-time chat and messaging systems
- Implement contextual commenting on content
- Build comprehensive notification management
- Enable secure, workspace-aware communication
- Support collaborative workflows across teams

Story 4.4.1: Chat & Messaging Components

Overview

Create chat and messaging components that enable real-time communication within workspaces, supporting both direct messages and group conversations with full security and collaboration features.

Context

- Existing organism component system with workspace context
- Real-time collaboration infrastructure already established
- Multiple workspace contexts requiring different messaging experiences
- Need for secure communication with encryption
- Integration with existing user management and permissions

Requirements

1. Create chat components:

- Chat interface with message history
- Real-time message delivery and read receipts
- File sharing and media attachments

- Message threading and replies
- Group chat and direct messaging

2. Implement workspace context features:

- Context-specific chat theming
- Permission-based messaging access
- Workspace-specific chat channels
- Role-based messaging features
- Context-aware message formatting

3. Create advanced messaging features:

- Message encryption and security
- Typing indicators and presence
- Message search and filtering
- Notification management
- Message archiving and export

Specific Tasks

- ✅ Create ChatInterface component
- ✅ Implement MessageList component
- ✅ Set up real-time messaging
- ✅ Create MessageInput component
- ✅ Implement file sharing
- ✅ Set up message threading

Documentation Required

- Chat component API documentation
- Real-time messaging implementation
- Security and encryption guide
- Workspace context usage
- Performance optimization

Testing Requirements

- Chat functionality tests
- Real-time messaging tests
- Security and encryption tests
- Performance and load tests
- Cross-platform compatibility tests

Integration Points

- Integration with real-time collaboration system
- Workspace context integration
- User management integration
- File sharing integration
- Notification system integration

Deliverables

- Complete chat and messaging system
- Real-time message delivery
- Security and encryption features
- Workspace context integration
- Comprehensive messaging documentation

Component Specifications

typescript

```
interface ChatInterfaceProps {
  workspace: Workspace
  currentUser: User
  chatId?: string
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onMessageSend?: (message: Message) => void
  onChatSelect?: (chat: Chat) => void
  showSidebar?: boolean
  showSearch?: boolean
  permissions?: string[]
}

interface MessageListProps {
  messages: Message[]
  currentUser: User
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onMessageReply?: (message: Message) => void
  onMessageReact?: (message: Message, reaction: string) => void
  onLoadMore?: () => void
  loading?: boolean
  hasMore?: boolean
}

interface MessageInputProps {
  onSend: (content: string, attachments?: File[]) => void
  onTyping?: () => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  placeholder?: string
  maxLength?: number
  allowAttachments?: boolean
  allowFormatting?: boolean
  disabled?: boolean
}

interface Message {
  id: string
  content: string
  sender: User
  timestamp: Date
  chatId: string
  status: 'sending' | 'sent' | 'delivered' | 'read'
  attachments?: Attachment[]
  replyTo?: string
  reactions?: MessageReaction[]
  edited?: boolean
  editedAt?: Date
}

interface Chat {
  id: string
  type: 'direct' | 'group' | 'channel'
  name?: string
  participants: User[]
  lastMessage?: Message
  unreadCount: number
  workspace: string
  createdAt: Date
  settings?: ChatSettings
}

interface MessageReaction {
  emoji: string
  users: User[]
  timestamp: Date
}

interface ChatSettings {
  notifications: boolean
  encryption: boolean
  retention: number
  permissions: string[]
}
```

Implementation Example

jsx

```
// ChatInterface implementation
function ChatInterface({ workspace, currentUser, context }) {
  const [selectedChat, setSelectedChat] = useState(null)
  const [messages, setMessages] = useState([])
  const { socket } = useRealTimeCollaboration()

  useEffect(() => {
    // Set up real-time message listening
    socket.on('message:new', handleNewMessage)
    socket.on('message:update', handleMessageUpdate)
    socket.on('typing:start', handleTypingStart)
    socket.on('typing:stop', handleTypingStop)

    return () => {
      socket.off('message:new')
      socket.off('message:update')
      socket.off('typing:start')
      socket.off('typing:stop')
    }
  }, [socket])

  return (
    <ChatContainer context={context}>
      <ChatSidebar>
        <ChatList
          chats={chats}
          selectedChat={selectedChat}
          onChatSelect={setSelectedChat}
          context={context}
        />
      </ChatSidebar>
      <ChatMain>
        {selectedChat ? (
          <>
            <ChatHeader chat={selectedChat} />
            <MessageList
              messages={messages}
              currentUser={currentUser}
              context={context}
            />
            <MessageInput
              onSend={handleSendMessage}
              context={context}
            />
          </>
        ) : (
          <EmptyState
            message="Select a conversation to start chatting"
            context={context}
          />
        )}
      </ChatMain>
    </ChatContainer>
  )
}
```

Performance Requirements

- Message delivery under 100ms
- Chat interface loading under 500ms
- Memory usage under 50MB
- Message search under 200ms
- File sharing under 2 seconds

Security Requirements

- End-to-end encryption for sensitive messages
- Message content sanitization
- File upload virus scanning
- Rate limiting for message sending
- Secure WebSocket connections

Story 4.4.2: Comment System Components

Overview

Create comment system components that enable contextual commenting on documents, designs, and projects with real-time synchronization and approval workflows.

Context

- Chat and messaging system established with real-time features
- Need for contextual comments on documents, designs, and projects

- Multiple workspace contexts requiring different comment workflows
- Real-time collaboration requiring comment synchronization
- Integration with existing content management

Requirements

1. Create comment components:

- Comment thread with nested replies
- Inline commenting for documents
- Annotation comments for designs
- Review comments with approval workflow
- Comment resolution and status tracking

2. Implement workspace context features:

- Context-specific comment permissions
- Role-based comment moderation
- Workspace-specific comment workflows
- Brand-aware comment styling
- Context-aware comment notifications

3. Create collaboration features:

- Real-time comment updates
- Comment mention system
- Comment approval workflows
- Comment analytics and tracking
- Comment export and reporting

Specific Tasks

- ✅ Create CommentThread component
- ✅ Implement InlineComment component
- ✅ Set up AnnotationComment component
- ✅ Create ReviewComment component
- ✅ Implement comment resolution

- ✅ Set up real-time synchronization

Documentation Required

- Comment system API documentation
- Workflow implementation guide
- Real-time synchronization
- Moderation and approval
- Analytics and reporting

Testing Requirements

- Comment functionality tests
- Real-time synchronization tests
- Workflow and approval tests
- Performance and scalability tests
- Accessibility compliance tests

Integration Points

- Integration with chat messaging system
- Workspace context integration
- Content management integration
- User management integration
- Notification system integration

Deliverables

- Complete comment system
- Real-time comment synchronization
- Workflow and approval features
- Analytics and reporting
- Comprehensive comment documentation

Component Specifications

typescript

```
interface CommentThreadProps {
  comments: Comment[]
  parentId: string
  parentType: 'document' | 'design' | 'project' | 'task'
  currentUser: User
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onCommentAdd?: (comment: Comment) => void
  onCommentReply?: (parentId: string, comment: Comment) => void
  onCommentResolve?: (commentId: string) => void
  allowReplies?: boolean
  permissions?: string[]
}

interface InlineCommentProps {
  selection: TextSelection
  document: Document
  currentUser: User
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onCommentCreate?: (comment: Comment) => void
  onSelectionChange?: (selection: TextSelection) => void
  permissions?: string[]
}

interface AnnotationCommentProps {
  position: { x: number; y: number }
  design: Design
  currentUser: User
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onAnnotationCreate?: (annotation: Annotation) => void
  onPositionChange?: (position: Position) => void
  permissions?: string[]
}

interface ReviewCommentProps {
  reviewItem: ReviewItem
  currentUser: User
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onApprove?: () => void
  onReject?: (reason: string) => void
  onCommentAdd?: (comment: Comment) => void
  showApprovalActions?: boolean
  permissions?: string[]
}
```

```typescript
interface Comment {
  id: string
  content: string
  author: User
  parentId?: string
  timestamp: Date
  edited?: boolean
  editedAt?: Date
  resolved?: boolean
  resolvedBy?: User
  resolvedAt?: Date
  mentions?: User[]
  attachments?: Attachment[]
  reactions?: CommentReaction[]
  workspaceId: string
}

interface TextSelection {
  start: number
  end: number
  text: string
  context?: string
}

interface Annotation {
  id: string
  position: Position
  comment: Comment
  design: string
  layer?: string
  status: 'open' | 'resolved' | 'archived'
}

interface ReviewItem {
  id: string
  type: 'document' | 'design' | 'code'
  title: string
  content: any
  status: 'pending' | 'approved' | 'rejected'
  reviewers: User[]
  approvals: Approval[]
}
```

Implementation Example

jsx

```
// CommentThread implementation
function CommentThread({ comments, parentId, currentUser, context }) {
  const [replyingTo, setReplyingTo] = useState(null)
  const { hasPermission } = useWorkspace()

  const renderComment = (comment, depth = 0) => {
    const canReply = hasPermission('comment:reply')
    const canResolve = hasPermission('comment:resolve') ||
                     comment.author.id === currentUser.id

    return (
      <CommentItem key={comment.id} depth={depth} context={context}>
        <CommentHeader>
          <Avatar user={comment.author} size="sm" />
          <CommentMeta>
            <UserName>{comment.author.name}</UserName>
            <Timestamp>{formatDate(comment.timestamp)}</Timestamp>
          </CommentMeta>
          {comment.resolved && (
            <ResolvedBadge>Resolved</ResolvedBadge>
          )}
        </CommentHeader>

        <CommentContent>
          {comment.content}
        </CommentContent>

        <CommentActions>
          {canReply && (
            <Button
              variant="ghost"
              size="sm"
              onClick={() => setReplyingTo(comment.id)}
            >
              Reply
            </Button>
          )}
          {canResolve && !comment.resolved && (
            <Button
              variant="ghost"
              size="sm"
              onClick={() => handleResolve(comment.id)}
            >
              Resolve
            </Button>
          )}
        </CommentActions>

        {replyingTo === comment.id && (
          <CommentInput
            onSubmit={(content) => handleReply(comment.id, content)}
            onCancel={() => setReplyingTo(null)}
            context={context}
          />
        )}

        {comment.replies?.map(reply =>
          renderComment(reply, depth + 1)
        )}
      </CommentItem>
    )
  }

  return (
    <CommentThreadContainer context={context}>
      {comments.map(comment => renderComment(comment))}
      <NewCommentInput
        onSubmit={handleNewComment}
        context={context}
      />
    </CommentThreadContainer>
  )
}
```

Performance Requirements

- Comment loading under 200ms
- Real-time updates under 100ms
- Memory usage under 30MB
- Comment search under 300ms
- Workflow processing under 1 second

Story 4.4.3: Notification Center Components

Overview

Create notification center components that provide comprehensive notification management with categorization, preferences, and real-time delivery across workspace contexts.

Context

- Comment system established with real-time features
- Need for centralized notification management
- Multiple workspace contexts requiring different notification types
- Real-time collaboration requiring instant notifications
- Integration with existing communication systems

Requirements

1. Create notification components:

- Notification center with categorization
- Toast notifications for immediate alerts
- Email notification management
- Push notification configuration
- Notification history and archiving

2. Implement workspace context features:

- Context-specific notification types
- Permission-based notification access
- Workspace-specific notification rules
- Role-based notification priorities
- Context-aware notification formatting

3. Create notification management features:

- Notification preferences and settings
- Notification filtering and grouping
- Notification scheduling and batching
- Notification analytics and tracking
- Notification template system

Specific Tasks

- ✅ Create NotificationCenter component
- ✅ Implement ToastNotification component
- ✅ Set up EmailNotification component
- ✅ Create PushNotification component
- ✅ Implement notification preferences
- ✅ Set up notification analytics

Documentation Required

- Notification system API documentation
- Notification types and categories
- Preference management guide
- Analytics and tracking
- Integration implementation

Testing Requirements

- Notification functionality tests
- Real-time notification tests
- Preference management tests
- Performance and scalability tests
- Cross-platform compatibility tests

Integration Points

- Integration with comment system
- Workspace context integration
- Communication system integration
- Analytics and tracking integration
- External notification services

Deliverables

- Complete notification center system
- Real-time notification delivery
- Preference management features

- Analytics and tracking
- Comprehensive notification documentation

Component Specifications

typescript

```
interface NotificationCenterProps {
  notifications: Notification[]
  currentUser: User
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onNotificationClick?: (notification: Notification) => void
  onMarkAsRead?: (notificationId: string) => void
  onMarkAllAsRead?: () => void
  onClearAll?: () => void
  showFilters?: boolean
  showSettings?: boolean
  permissions?: string[]
}

interface ToastNotificationProps {
  notification: Notification
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  position?: 'top-left' | 'top-right' | 'bottom-left' | 'bottom-right'
  duration?: number
  onDismiss?: () => void
  onAction?: (action: NotificationAction) => void
  autoClose?: boolean
}

interface NotificationPreferencesProps {
  user: User
  preferences: NotificationPreferences
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  onPreferenceChange?: (preferences: NotificationPreferences) => void
  onSave?: () => void
  showAdvanced?: boolean
  permissions?: string[]
}

interface Notification {
  id: string
  type: 'info' | 'success' | 'warning' | 'error' | 'mention' | 'update'
  category: 'system' | 'chat' | 'comment' | 'task' | 'billing' | 'general'
  title: string
  message: string
  timestamp: Date
  read: boolean
  workspace: string
  sender?: User
  actions?: NotificationAction[]
  metadata?: Record<string, any>
  priority: 'low' | 'medium' | 'high' | 'urgent'
}

interface NotificationAction {
  id: string
  label: string
  type: 'primary' | 'secondary' | 'link'
  action: string
  url?: string
  data?: any
}

interface NotificationPreferences {
  channels: {
    inApp: boolean
    email: boolean
    push: boolean
    sms: boolean
  }
  categories: Record<string, CategoryPreference>
  schedule: {
    doNotDisturb: boolean
    doNotDisturbStart?: string
    doNotDisturbEnd?: string
    timezone: string
  }
  grouping: {
    enabled: boolean
    interval: number
  }
}

interface CategoryPreference {
  enabled: boolean
  channels: string[]
  priority: string
  sound?: boolean
}
```

Implementation Example

jsx

```
// NotificationCenter implementation
function NotificationCenter({ notifications, currentUser, context }) {
  const [filter, setFilter] = useState('all')
  const [showSettings, setShowSettings] = useState(false)
  const { hasPermission } = useWorkspace()

  const groupedNotifications = useMemo(() => {
    return notifications.reduce((groups, notification) => {
      const date = formatDate(notification.timestamp)
      if (!groups[date]) {
        groups[date] = []
      }
      groups[date].push(notification)
      return groups
    }, {})
  }, [notifications])

  return (
    <NotificationCenterContainer context={context}>
      <NotificationHeader>
        <Heading level={3}>Notifications</Heading>
        <HeaderActions>
          <Button
            variant="ghost"
            size="sm"
            onClick={handleMarkAllAsRead}
          >
            Mark all as read
          </Button>
          <IconButton
            icon="settings"
            onClick={() => setShowSettings(true)}
          />
        </HeaderActions>
      </NotificationHeader>

      <NotificationFilters>
        <FilterButton
          active={filter === 'all'}
          onClick={() => setFilter('all')}
        >
          All
        </FilterButton>
      <FilterButton
        active={filter === 'unread'}
        onClick={() => setFilter('unread')}
      >
        Unread
      </FilterButton>
      {Object.keys(NOTIFICATION_CATEGORIES).map(category => (
        <FilterButton
          key={category}
          active={filter === category}
          onClick={() => setFilter(category)}
        >
          {NOTIFICATION_CATEGORIES[category]}
        </FilterButton>
      ))}
    </NotificationFilters>

    <NotificationList>
      {Object.entries(groupedNotifications).map(([date, items]) => (
        <NotificationGroup key={date}>
          <DateHeader>{date}</DateHeader>
          {items.map(notification => (
            <NotificationItem
              key={notification.id}
              notification={notification}
              onClick={() => handleNotificationClick(notification)}
              onMarkAsRead={() => handleMarkAsRead(notification.id)}
              context={context}
            />
          ))}
        </NotificationGroup>
      ))}
    </NotificationList>

    {showSettings && (
      <NotificationPreferences
        user={currentUser}
        preferences={preferences}
        onPreferenceChange={handlePreferenceChange}
        onClose={() => setShowSettings(false)}
        context={context}
      />
    )}
  </NotificationCenterContainer>
  )
}
```

Performance Requirements

- Notification delivery under 50ms
- Notification center loading under 300ms
- Memory usage under 40MB
- Notification processing under 100ms
- Preference updates under 200ms

Performance Optimization

Real-time Communication

- WebSocket connection pooling
- Message batching for bulk operations
- Lazy loading for message history
- Virtual scrolling for long conversations
- Optimistic UI updates

Notification Management

- Notification aggregation and grouping
- Smart batching for email notifications
- Efficient database queries with pagination
- Client-side caching with invalidation
- Progressive loading strategies

Accessibility Requirements

WCAG 2.1 AA Compliance

- Screen reader announcements for new messages
- Keyboard navigation through all communication features
- High contrast mode support
- Focus management for modal interactions
- Clear labeling for all interactive elements

Communication Accessibility

- Alternative text for media attachments
- Transcript availability for voice messages
- Visual indicators for audio notifications
- Customizable notification sounds
- Support for reduced motion preferences

Security Considerations

Message Security

- End-to-end encryption for sensitive channels
- Message content sanitization
- Secure file upload with scanning
- Rate limiting for spam prevention
- Audit logging for compliance

Notification Security

- Secure notification delivery channels
- Token-based authentication for push
- Encrypted notification payloads
- Permission-based notification access
- Data retention policies

Testing Strategy

Unit Tests

- Component functionality testing
- Real-time event handling
- Message encryption/decryption
- Notification delivery logic
- Permission validation

Integration Tests

- WebSocket connection handling

- Cross-component communication
- Notification service integration
- File upload integration
- Search functionality

E2E Tests

- Complete chat workflows
- Comment thread interactions
- Notification preferences flow
- Cross-platform messaging
- Performance under load

Storybook Documentation

Chat & Messaging Stories

- Basic chat interface
- Group conversations
- File sharing examples
- Real-time indicators
- Error states

Comment System Stories

- Document commenting
- Design annotations
- Review workflows
- Comment threading
- Resolution flows

Notification Stories

- Notification center
- Toast notifications
- Preference management
- Category filtering

- Real-time updates

Migration Guide

From Legacy Communication

1. Map existing chat data to new schema
2. Migrate notification preferences
3. Update WebSocket connections
4. Implement new permission model
5. Test real-time functionality

Breaking Changes

- New message format structure
- Updated notification API
- Changed event signatures
- New permission requirements
- Modified state management