Epic 2.2: Display Components

Epic Overview

This epic enhances and completes the display components for THE WHEEL design system, adding workspace context awareness to typography, status indicators, media components, and loading states.

Priority: P0 (Critical)
Timeline: 3 weeks
Dependencies: Epic 1.1 (Monorepo Architecture Setup), Epic 2.1 (Input Components)

Story 2.2.1: Typography System

Overview

Enhance the Typography system (Text, Heading) with workspace context support and responsive design capabilities.

AI Developer Prompt

You are enhancing the Typography system (Text, Heading) for THE WHEEL design system. Building on the specialized input components from Epic 2.1, you need to create a comprehensive typography system that supports workspace context and responsive design.

Context

- Existing Text and Heading components in packages/ui/src/typography/
- Workspace context system integrated with other components
- Brand integration with typography from Feature 1
- Need responsive typography that works across device sizes
- Must maintain accessibility while adding advanced features

Requirements

1. Enhance Text component with workspace context:

- Add context prop for workspace-specific styling
- Implement responsive typography scaling
- Add semantic HTML support (p, span, div, etc.)
- Text truncation with workspace styling

- Color variants for different contexts

2. Enhance Heading component:

- Implement proper heading hierarchy (h1-h6)
- Responsive typography scaling
- Workspace context styling
- Accessibility improvements
- Brand-consistent heading styles

3. Create typography scale system:

- Consistent sizing across all text elements
- Mobile-first responsive scaling
- Workspace context variations
- Line height and spacing optimization
- Brand typography integration

Specific Tasks

- [ ] Extend Text and Heading components with context props
- [ ] Implement responsive typography scaling system
- [ ] Add semantic HTML element support
- [ ] Create text truncation functionality
- [ ] Implement proper heading hierarchy
- [ ] Add workspace context styling variants
- [ ] Update accessibility attributes

Documentation Required

- Typography system architecture documentation
- Responsive scaling implementation guide
- Workspace context typography usage
- Accessibility implementation details
- Brand typography integration guide
- Typography scale and hierarchy guide

Testing Requirements

- Responsive typography scaling tests
- Workspace context styling tests
- Accessibility compliance tests
- Text truncation functionality tests
- Heading hierarchy tests
- Cross-browser typography rendering tests
- Performance tests for typography rendering

Integration Points

- Integration with workspace context providers
- Brand typography system integration
- Responsive design system integration
- Accessibility utilities integration
- CSS variable system integration

Deliverables

- Enhanced Text and Heading components
- Responsive typography scaling system
- Workspace context typography variants
- Accessibility compliance validation
- Typography scale documentation
- Comprehensive Storybook stories

Component Specifications

typescript

```typescript
interface TextProps extends React.HTMLAttributes<HTMLElement> {
  as?: 'p' | 'span' | 'div' | 'label' | 'caption'
  variant?: 'body' | 'caption' | 'overline' | 'subtitle1' | 'subtitle2'
  size?: 'xs' | 'sm' | 'md' | 'lg' | 'xl'
  weight?: 'light' | 'normal' | 'medium' | 'semibold' | 'bold'
  color?: 'primary' | 'secondary' | 'muted' | 'error' | 'warning' | 'success'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  truncate?: boolean
  align?: 'left' | 'center' | 'right' | 'justify'
  responsive?: boolean
  children: React.ReactNode
}

interface HeadingProps extends React.HTMLAttributes<HTMLHeadingElement> {
  level: 1 | 2 | 3 | 4 | 5 | 6
  size?: 'xs' | 'sm' | 'md' | 'lg' | 'xl' | '2xl' | '3xl'
  weight?: 'light' | 'normal' | 'medium' | 'semibold' | 'bold'
  color?: 'primary' | 'secondary' | 'muted' | 'error' | 'warning' | 'success'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  truncate?: boolean
  align?: 'left' | 'center' | 'right' | 'justify'
  responsive?: boolean
  children: React.ReactNode
}
```

Story 2.2.2: Status & Feedback Components

Overview

Enhance status and feedback components (Badge, StatusDot, Toast, Alert) for consistent status communication across workspace contexts.

AI Developer Prompt

You are enhancing status and feedback components (Badge, StatusDot, Toast, Alert) for THE WHEEL design system. Building on the typography system from Story 2.2.1, you need to create consistent status communication across workspace contexts.

Context

- Enhanced typography system with workspace context

- Existing Badge, StatusDot, Toast, Alert components
- Need consistent status communication across all workspace contexts
- Must integrate with notification systems and real-time updates
- Status components are critical for workspace collaboration

Requirements

1. Enhance Badge component with workspace variants:

- Add workspace-specific status variants
- Implement count badges with animation
- Add dot variant for minimal status indication
- Size variants for different use cases
- Integration with workspace theme colors

2. Enhance StatusDot with workspace statuses:

- Online/offline/busy/away states
- Workspace-specific status colors
- Pulse animation for active states
- Size variants for different contexts
- Accessibility improvements

3. Enhance Toast and Alert systems:

- Workspace context styling
- Urgency levels and priorities
- Auto-dismiss functionality
- Action button integration
- Accessibility announcements

Specific Tasks

- [ ] Extend Badge component with workspace variants
- [ ] Add count and dot badge variants
- [ ] Enhance StatusDot with workspace statuses
- [ ] Implement Toast with workspace context
- [ ] Enhance Alert with urgency levels

- [ ] Create consistent status API across components
- [ ] Add animation states for status changes

Documentation Required

- Status component system architecture
- Workspace status variant usage guide
- Toast and alert implementation patterns
- Accessibility implementation details
- Status communication best practices
- Animation and interaction guidelines

Testing Requirements

- Status variant functionality tests
- Workspace context styling tests
- Animation state tests
- Accessibility compliance tests
- Auto-dismiss functionality tests
- Status communication integration tests
- Cross-browser compatibility tests

Integration Points

- Integration with workspace context providers
- Notification system integration
- Real-time status update integration
- Theme system CSS variable integration
- Accessibility utilities integration

Deliverables

- Enhanced Badge component with workspace variants
- StatusDot component with workspace statuses
- Toast and Alert components with context awareness
- Consistent status API across all components
- Animation system for status changes

- Comprehensive Storybook stories

Component Specifications

typescript

```typescript
interface BadgeProps extends React.HTMLAttributes<HTMLSpanElement> {
  variant?: 'primary' | 'secondary' | 'success' | 'warning' | 'error' | 'info'
  size?: 'sm' | 'md' | 'lg'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  count?: number
  dot?: boolean
  maxCount?: number
  showZero?: boolean
  children?: React.ReactNode
}

interface StatusDotProps extends React.HTMLAttributes<HTMLSpanElement> {
  status: 'online' | 'offline' | 'busy' | 'away' | 'inactive'
  size?: 'sm' | 'md' | 'lg'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  pulse?: boolean
  label?: string
}

interface ToastProps {
  variant?: 'success' | 'warning' | 'error' | 'info'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  title?: string
  description?: string
  duration?: number
  actions?: Array<{label: string, onClick: () => void}>
  onClose?: () => void
  persistent?: boolean
}

interface AlertProps extends React.HTMLAttributes<HTMLDivElement> {
  variant?: 'success' | 'warning' | 'error' | 'info'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  urgency?: 'low' | 'medium' | 'high' | 'critical'
  title?: string
  description?: string
  actions?: Array<{label: string, onClick: () => void}>
  dismissible?: boolean
  onClose?: () => void
}
```

Story 2.2.3: Media & Visual Components

Overview

Enhance media and visual components (Avatar, Image, Logo, Icon) with workspace context awareness and sophisticated media handling.

AI Developer Prompt

You are enhancing media and visual components (Avatar, Image, Logo, Icon) for THE WHEEL design system. Building on the status and feedback components from Story 2.2.2, you need to create sophisticated media handling with workspace context awareness.

Context

- Enhanced status and feedback components with workspace context
- Existing Avatar, Image, Icon components
- Need Logo component for workspace branding
- Must support workspace asset management
- Media components are critical for professional appearance

Requirements

1. Enhance Avatar component with workspace presence:

- Add presence indicators (online, offline, busy, away)
- Workspace context styling
- Fallback handling for missing images
- Size variants for different use cases
- Group avatar functionality

2. Enhance Image component with workspace assets:

- Workspace asset management integration
- Responsive image loading
- Lazy loading with workspace styling
- Error handling with branded fallbacks
- Optimization for different contexts

3. Build Logo component with workspace variants:

- Workspace branding support
- Multiple logo variants (full, mark, wordmark)
- Responsive logo sizing
- Dark/light mode variants
- Brand guideline compliance

4. Enhance Icon system with workspace icons:

- Workspace-specific icon sets
- Icon color variants for contexts
- Size and styling consistency
- Accessibility improvements
- Custom icon support

Specific Tasks

- [ ] Enhance Avatar with presence indicators
- [ ] Add workspace asset management to Image
- [ ] Build Logo component with workspace variants
- [ ] Enhance Icon system with workspace icons
- [ ] Implement responsive image loading
- [ ] Add lazy loading with workspace styling
- [ ] Create fallback handling for all media

Documentation Required

- Media component system architecture
- Workspace asset management guide
- Logo usage guidelines and variants
- Icon system documentation
- Responsive image implementation
- Accessibility implementation details

Testing Requirements

- Presence indicator functionality tests
- Workspace asset loading tests
- Logo variant display tests

- Icon system consistency tests
- Responsive image behavior tests
- Accessibility compliance tests
- Performance tests for media loading

Integration Points

- Integration with workspace context providers
- Asset management system integration
- Presence system integration
- Theme system CSS variable integration
- Performance optimization integration

Deliverables

- Enhanced Avatar component with presence indicators
- Image component with workspace asset management
- Logo component with workspace variants
- Enhanced Icon system with workspace icons
- Responsive and lazy loading implementation
- Comprehensive Storybook stories

Component Specifications

typescript

```ts
interface AvatarProps extends React.HTMLAttributes<HTMLDivElement> {
  src?: string
  alt?: string
  size?: 'xs' | 'sm' | 'md' | 'lg' | 'xl' | '2xl'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  presence?: 'online' | 'offline' | 'busy' | 'away'
  shape?: 'circle' | 'square'
  fallback?: string | React.ReactNode
  badge?: React.ReactNode
  onClick?: () => void
}

interface ImageProps extends React.ImgHTMLAttributes<HTMLImageElement> {
  src: string
  alt: string
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  aspectRatio?: string
  fit?: 'cover' | 'contain' | 'fill'
  lazy?: boolean
  placeholder?: string | React.ReactNode
  error?: string | React.ReactNode
  onLoad?: () => void
  onError?: () => void
}

interface LogoProps extends React.HTMLAttributes<HTMLDivElement> {
  variant?: 'full' | 'mark' | 'wordmark'
  size?: 'xs' | 'sm' | 'md' | 'lg' | 'xl'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  theme?: 'light' | 'dark'
  responsive?: boolean
  onClick?: () => void
}

interface IconProps extends React.HTMLAttributes<HTMLSpanElement> {
  name: string
  size?: 'xs' | 'sm' | 'md' | 'lg' | 'xl'
  color?: 'primary' | 'secondary' | 'muted' | 'error' | 'warning' | 'success'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  rotation?: 0 | 90 | 180 | 270
  title?: string
}
```

Story 2.2.4: Loading & Empty State Components

Overview

Create loading and empty state components for user experience during data loading and when no content is available.

AI Developer Prompt

You are creating loading and empty state components for THE WHEEL design system. These components are essential for user experience during data loading and when no content is available.

Context

- Existing display components with workspace context support
- Real-time collaboration features requiring loading states
- Multiple workspace contexts requiring different messaging
- Need consistent loading and empty state patterns
- Accessibility requirements for screen readers

Requirements

1. Create loading state components:

- Spinner components with context-aware styling
- Skeleton loading for content placeholders
- Progress indicators for long operations
- Loading overlays for forms and tables
- Pulse animations for real-time updates

2. Implement empty state components:

- Empty state illustrations for different contexts
- Call-to-action buttons for empty states
- Search result empty states
- Error state components with recovery options
- Workspace-specific empty state messaging

3. Create context-aware variations:

- Consultant context: Professional loading and empty states
- Client context: Friendly and approachable messaging
- Admin context: System-focused states and actions
- Expert context: Marketplace-specific states
- Tool Creator context: Development-focused messaging
- Founder context: Growth-oriented states

Specific Tasks

- [ ] Create Spinner component with variants
- [ ] Implement SkeletonLoader component
- [ ] Set up ProgressIndicator component
- [ ] Create EmptyState component
- [ ] Implement LoadingOverlay component
- [ ] Set up error state components

Documentation Required

- Loading state best practices
- Empty state design guidelines
- Accessibility implementation
- Context-specific usage examples
- Performance optimization tips

Testing Requirements

- Loading state rendering tests
- Empty state interaction tests
- Accessibility compliance tests
- Performance and animation tests
- Context variation tests

Integration Points

- Integration with existing display system
- Workspace context integration
- Theme system integration
- Real-time status integration

- Error handling integration

Deliverables

- Complete loading and empty state library
- Context-aware messaging system
- Accessibility-compliant components
- Performance-optimized animations
- Comprehensive usage documentation

Performance Requirements

- Loading animation maintains 60fps
- Component rendering under 16ms
- Memory usage under 5MB
- State transition under 200ms
- Animation CPU usage under 10%

Timeline and Dependencies

Timeline

- Week 1: Story 2.2.1 - Typography System
- Week 1-2: Story 2.2.2 - Status & Feedback Components
- Week 2-3: Story 2.2.3 - Media & Visual Components
- Week 3: Story 2.2.4 - Loading & Empty State Components

Dependencies

- Epic 1.1 (Monorepo Architecture Setup) - Complete
- Epic 2.1 (Input Components) - Should be in progress
- Workspace context system operational
- Brand assets available

Success Metrics

- All display components support workspace contexts
- 100% accessibility compliance (WCAG 2.1 AA)

- Performance benchmarks met (60fps animations)
- Complete test coverage (90%+ for all components)
- Comprehensive documentation for all components
- Consistent visual language across components

Risk Mitigation

- Design review checkpoints for visual consistency
- Performance monitoring during development
- Regular accessibility audits
- Cross-browser testing at each milestone
- User testing for empty states and loading experiences
- Clear migration paths for existing implementations