Epic 3.3: Interactive Molecules

Epic Overview

This epic creates interactive molecule components that handle complex user interactions, including action groups, menus, toolbars, and workspace-specific interactive components for business workflows.

Priority: P1 (High)
Timeline: 3 weeks
Dependencies: Epic 2.1 (Input Components), Epic 3.1 (Form Molecules), Epic 3.2 (Display Molecules)

Story 3.3.1: Action Components

Overview

Create sophisticated action components including button groups, action menus, and toolbars that handle user interactions consistently across workspace contexts.

AI Developer Prompt

You are enhancing action components for THE WHEEL design system. Building on the workspace-specific display cards from Story 3.2.3, you need to create interactive action components that handle user interactions consistently across workspace contexts.

Context

- Complete workspace-specific display card system
- Need interactive action components for user interactions
- Must support workspace-specific actions and permissions
- Action components are critical for user productivity
- Need consistent interaction patterns across all contexts

Requirements

1. Enhance ButtonGroup component with workspace actions:

- Workspace-specific action grouping
- Permission-based action visibility
- Action state management

- Bulk action handling
- Responsive action layouts

2. Enhance ActionMenu component with workspace context:

- Workspace-specific menu items
- Role-based action filtering
- Contextual action suggestions
- Action history tracking
- Keyboard navigation support

3. Enhance Toolbar component with workspace tools:

- Workspace-specific tool sets
- Customizable toolbar layouts
- Tool state persistence
- Responsive toolbar behavior
- Tool group management

Specific Tasks

- [ ] Enhance ButtonGroup with workspace actions
- [ ] Add permission-based action filtering
- [ ] Enhance ActionMenu with workspace context
- [ ] Add contextual action suggestions
- [ ] Enhance Toolbar with workspace tools
- [ ] Add tool state persistence
- [ ] Create consistent action patterns
- [ ] Add keyboard navigation support

Documentation Required

- Action component architecture
- Workspace-specific action patterns
- Permission-based action filtering
- Action state management
- Keyboard navigation implementation
- Accessibility guidelines

Testing Requirements

- Action component functionality tests
- Permission-based filtering tests
- Workspace context tests
- Keyboard navigation tests
- Action state management tests
- Accessibility compliance tests

Integration Points

- Integration with workspace context providers
- Permission system integration
- Action state management integration
- Keyboard navigation integration
- Analytics tracking integration

Deliverables

- Enhanced ButtonGroup with workspace actions
- ActionMenu with workspace context
- Toolbar with workspace tools
- Action state management system
- Keyboard navigation support
- Comprehensive Storybook stories

Component Specifications

typescript

```typescript
interface ButtonGroupProps {
  actions: ActionItem[]
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  orientation?: 'horizontal' | 'vertical'
  size?: 'sm' | 'md' | 'lg'
  variant?: 'solid' | 'outline' | 'ghost'
  onActionClick?: (action: ActionItem) => void
  permissions?: string[]
  maxVisible?: number
  showOverflow?: boolean
}

interface ActionMenuProps {
  actions: ActionItem[]
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  trigger?: React.ReactNode
  placement?: 'bottom' | 'top' | 'left' | 'right'
  onActionClick?: (action: ActionItem) => void
  onMenuOpen?: () => void
  onMenuClose?: () => void
  permissions?: string[]
  showSuggestions?: boolean
  searchable?: boolean
}

interface ToolbarProps {
  tools: ToolItem[]
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  layout?: 'horizontal' | 'vertical' | 'grid'
  customizable?: boolean
  onToolClick?: (tool: ToolItem) => void
  onLayoutChange?: (layout: ToolbarLayout) => void
  permissions?: string[]
  persistent?: boolean
  responsive?: boolean
}

interface ActionItem {
  id: string
  label: string
  icon?: string
  shortcut?: string
  permission?: string
```

```
  workspaceContext?: string[]
  disabled?: boolean
  loading?: boolean
  variant?: 'primary' | 'secondary' | 'danger'
  onClick: () => void
  metadata?: Record<string, any>
}
```

Story 3.3.2: Workspace-Specific Interactive Components

Overview

Build specialized interactive components for different workspace contexts, including workspace switchers, client selectors, time trackers, and billing controls.

AI Developer Prompt

You are building workspace-specific interactive components for THE WHEEL design system. Building on the action components from Story 3.3.1, you need to create specialized interactive components for different workspace contexts and business workflows.

Context

- Complete action component system with workspace context
- Need specialized interactive components for workspace workflows
- Must support different workspace contexts with unique interactions
- Interactive components should facilitate business processes
- Need consistent interaction patterns across workspace types

Requirements

1. Build WorkspaceSwitcher component:

- Multi-workspace navigation and switching
- Context preservation during switches
- Workspace search and filtering
- Recent workspace tracking
- Workspace creation shortcuts

2. Build ClientSelector component:

- Client search and selection
- Recent client tracking
- Client filtering and categorization
- Client creation workflow
- Permission-based client access

3. Build TimeTracker component:

- Time tracking interface and controls
- Project and task association
- Time entry validation
- Timer state management
- Time reporting and export

4. Build BillingControls component:

- Billing management interface
- Invoice generation and management
- Payment processing controls
- Billing report generation
- Revenue tracking and analytics

Specific Tasks

- [ ] Build WorkspaceSwitcher component
- [ ] Add context switching logic
- [ ] Build ClientSelector component
- [ ] Add client search and filtering
- [ ] Build TimeTracker component
- [ ] Add time tracking interface
- [ ] Build BillingControls component
- [ ] Add billing management
- [ ] Create consistent interaction patterns
- [ ] Add workflow automation

Documentation Required

- Workspace-specific interaction architecture

- Context switching implementation
- Client selection patterns
- Time tracking workflow
- Billing management system
- Interaction accessibility guidelines

Testing Requirements

- Workspace switching functionality tests
- Client selection workflow tests
- Time tracking accuracy tests
- Billing management tests
- Context preservation tests
- Accessibility compliance tests

Integration Points

- Integration with workspace context providers
- Client management system integration
- Time tracking service integration
- Billing system integration
- Workflow automation integration

Deliverables

- WorkspaceSwitcher component with context switching
- ClientSelector component with search and filtering
- TimeTracker component with time management
- BillingControls component with billing management
- Workflow automation system
- Comprehensive Storybook stories

Component Specifications

typescript

```
interface WorkspaceSwitcherProps {
  workspaces: Workspace[]
  currentWorkspace: Workspace
  onWorkspaceChange: (workspace: Workspace) => void
  onWorkspaceCreate?: () => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showRecent?: boolean
  showSearch?: boolean
  maxRecent?: number
  placement?: 'dropdown' | 'modal' | 'sidebar'
}

interface ClientSelectorProps {
  clients: Client[]
  selectedClient?: Client
  onClientSelect: (client: Client) => void
  onClientCreate?: () => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showSearch?: boolean
  showRecent?: boolean
  showFilters?: boolean
  filters?: ClientFilter[]
  onFilterChange?: (filters: ClientFilter[]) => void
  permissions?: string[]
}

interface TimeTrackerProps {
  currentSession?: TimeSession
  onSessionStart: (project: Project, task?: Task) => void
  onSessionStop: () => void
  onSessionPause: () => void
  onTimeEntry: (entry: TimeEntry) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  projects: Project[]
  tasks: Task[]
  showTimer?: boolean
  showHistory?: boolean
  autoSave?: boolean
}

interface BillingControlsProps {
  workspace: Workspace
  billing: BillingInfo
```

```
  onInvoiceGenerate: (data: InvoiceData) => void
  onPaymentProcess: (payment: PaymentData) => void
  onReportGenerate: (type: ReportType) => void
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  showInvoicing?: boolean
  showPayments?: boolean
  showReports?: boolean
  permissions?: string[]
}
```

Timeline and Dependencies

Timeline

- Week 1-2: Story 3.3.1 - Action Components
- Week 2-3: Story 3.3.2 - Workspace-Specific Interactive Components

Dependencies

- Epic 2.1 (Input Components) - Complete
- Epic 3.1 (Form Molecules) - Complete
- Epic 3.2 (Display Molecules) - Should be complete
- Workspace context system operational
- Permission system established

Success Metrics

- All interactive components support workspace contexts
- Permission-based actions working correctly
- Keyboard navigation fully implemented
- 100% accessibility compliance
- Performance benchmarks met (interaction response under 100ms)
- Complete test coverage (90%+ for all components)

Risk Mitigation

- Thorough testing of permission logic
- Performance monitoring for complex interactions
- Regular accessibility audits

- User testing for workspace workflows
- Clear documentation for interaction patterns
- Fallback mechanisms for failed actions