# Epic 2.1: Input Components

## Epic Overview

This epic enhances and completes the core input components for THE WHEEL design system, adding workspace context awareness and advanced functionality to the existing foundation.

**Priority:** P0 (Critical)
**Timeline:** 3-4 weeks
**Dependencies:** Epic 1.1 (Monorepo Architecture Setup)

---

## Story 2.1.1: Button Component Enhancement

### Overview

Enhance the existing Button component with workspace context awareness, loading states, and advanced functionality while maintaining backward compatibility.

### AI Developer Prompt

You are enhancing the Button component for THE WHEEL design system. Building on the foundation infrastructure from Feature 1, you need to add workspace context awareness and advanced functionality to the existing Button component.

### Context

- Existing Button component in packages/ui/src/button/Button.tsx
- Monorepo structure with Storybook integration
- Workspace context system with consultant, client, admin themes
- Brand integration with color system and typography
- Need to maintain backward compatibility while adding new features

### Requirements

**1. Enhance Button component with workspace context:**

- Add `context` prop for workspace awareness (consultant, client, admin)
- Implement theme variants for different workspace contexts
- Add loading states with workspace-appropriate spinners
- Maintain existing size variants (sm, md, lg)

- Add icon button variant with proper spacing

**2. Improve accessibility and user experience:**

- Enhanced ARIA attributes for screen readers
- Keyboard navigation improvements
- Focus management and visual indicators
- Disabled state styling improvements
- Loading state announcements

**3. Add workspace-specific variants:**

- Consultant theme: Professional blue palette
- Client theme: Approachable green palette
- Admin theme: Authoritative gray palette
- Maintain existing variants (primary, secondary, outline, ghost, link)

## Specific Tasks

- [ ] Extend Button props interface with context prop
- [ ] Implement workspace theme styling with CSS variables
- [ ] Add loading state with context-appropriate spinner
- [ ] Enhance accessibility with improved ARIA attributes
- [ ] Create icon button variant with proper spacing
- [ ] Update disabled state styling for better UX
- [ ] Add comprehensive prop validation

## Documentation Required

- Updated Button component API documentation
- Workspace context usage examples
- Accessibility implementation notes
- Loading state behavior documentation
- Icon button usage guidelines
- Migration guide for existing Button usage

## Testing Requirements

- Unit tests for all new props and variants

- Workspace context switching tests
- Accessibility compliance tests (WCAG 2.1)
- Loading state behavior tests
- Icon button functionality tests
- Keyboard navigation tests
- Visual regression tests for all themes

## Integration Points

- Integration with workspace context providers
- Theme system CSS variable integration
- Storybook story updates with new features
- Icon system integration for icon buttons
- Loading spinner integration

## Deliverables

- Enhanced Button component with workspace context
- Comprehensive Storybook story with all variants
- Updated TypeScript interfaces and documentation
- Complete test suite with 90%+ coverage
- Accessibility compliance validation
- Performance benchmarks for all variants

## Component Specifications

typescript

```typescript
interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {
  variant?: 'primary' | 'secondary' | 'outline' | 'ghost' | 'link'
  size?: 'sm' | 'md' | 'lg'
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  isLoading?: boolean
  loadingText?: string
  icon?: React.ReactNode
  iconPosition?: 'left' | 'right'
  fullWidth?: boolean
  disabled?: boolean
  children: React.ReactNode
}
```

### Storybook Requirements

- Stories for all workspace contexts
- Interactive examples with state changes
- Loading state demonstrations
- Icon button examples
- Accessibility testing integration
- Performance monitoring integration

---

## Story 2.1.2: Form Input Components

### Overview

Enhance the form input components (Input, Select, Textarea) with workspace context awareness and advanced form functionality.

### AI Developer Prompt

You are enhancing the form input components (Input, Select, Textarea) for THE WHEEL design system. Building on the Button enhancement from Story 2.1.1, you need to add workspace context awareness and advanced form functionality.

### Context

- Existing Input, Select, Textarea components in packages/ui/src/
- Button component now has workspace context integration

- Monorepo with Storybook and workspace context system
- Need consistent form field behavior across workspace contexts
- Must support complex validation and error handling

## Requirements

**1. Enhance Input component with workspace context:**

- Add context prop for workspace-specific styling
- Implement validation state styling (error, warning, success)
- Add helper text and error message support
- Improve accessibility with proper ARIA attributes
- Add input masking for specialized inputs

**2. Enhance Select component with advanced features:**

- Add loading states for async options
- Implement searchable/filterable options
- Add grouped options support
- Multi-select functionality improvements
- Workspace context styling integration

**3. Enhance Textarea component:**

- Add auto-resize functionality
- Character count with workspace styling
- Resize handle workspace theming
- Improved scroll behavior
- Enhanced accessibility features

## Specific Tasks

- [ ] Extend input components with workspace context props
- [ ] Implement validation state styling system
- [ ] Add helper text and error message components
- [ ] Create consistent error handling across all inputs
- [ ] Add loading states for Select component
- [ ] Implement auto-resize for Textarea

- [ ] Update accessibility attributes for all components

### Documentation Required

- Enhanced form component API documentation
- Validation state system guide
- Workspace context form styling
- Accessibility implementation details
- Form field composition patterns
- Error handling best practices

### Testing Requirements

- Validation state behavior tests
- Workspace context styling tests
- Accessibility compliance tests
- Loading state functionality tests
- Auto-resize behavior tests
- Error handling and recovery tests
- Cross-browser compatibility tests

### Integration Points

- Integration with workspace context providers
- Validation system integration
- Error handling system integration
- Theme system CSS variable integration
- Form field wrapper component integration

### Deliverables

- Enhanced Input, Select, Textarea components
- Comprehensive Storybook stories for all components
- Validation state styling system
- Accessibility compliance for all form inputs
- Form field composition examples
- Complete test suite with validation scenarios

## Component Specifications

```typescript
interface InputProps extends React.InputHTMLAttributes<HTMLInputElement> {
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  validationState?: 'error' | 'warning' | 'success'
  helperText?: string
  errorMessage?: string
  label?: string
  required?: boolean
  loading?: boolean
}

interface SelectProps extends React.SelectHTMLAttributes<HTMLSelectElement> {
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  validationState?: 'error' | 'warning' | 'success'
  helperText?: string
  errorMessage?: string
  label?: string
  loading?: boolean
  searchable?: boolean
  options: Array<{value: string, label: string, group?: string}>
  multiple?: boolean
}

interface TextareaProps extends React.TextareaHTMLAttributes<HTMLTextAreaElement> {
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  validationState?: 'error' | 'warning' | 'success'
  helperText?: string
  errorMessage?: string
  label?: string
  autoResize?: boolean
  maxCharacters?: number
  showCharacterCount?: boolean
}
```

---

## Story 2.1.3: Specialized Input Components

### Overview

Build specialized input components (TimePicker, ColorPicker, enhanced DatePicker) for specific business use cases within workspace contexts.

## AI Developer Prompt

You are building specialized input components (TimePicker, ColorPicker, enhanced DatePicker) for THE WHEEL design system. Building on the form input enhancements from Story 2.1.2, you need to create sophisticated input components for specific business use cases.

## Context

- Enhanced form input components with workspace context
- Existing DatePicker component needs timezone support
- Need TimePicker and ColorPicker for workspace customization
- Must integrate with existing validation and theming systems
- Business requirements include meeting scheduling and branding

## Requirements

### 1. Build TimePicker component:

- 12/24 hour format support
- Timezone awareness for global teams
- Workspace context styling
- Accessibility compliance
- Integration with existing form validation

### 2. Build ColorPicker component:

- Workspace brand color constraints
- Hex, RGB, HSL input support
- Color palette presets
- Accessibility features for color blindness
- Integration with theme system

### 3. Enhance DatePicker with timezone support:

- Multiple timezone display
- Timezone conversion utilities
- Workspace-specific date formats
- Enhanced accessibility features

- Integration with calendar scheduling

### Specific Tasks

- [ ] Build TimePicker with timezone support
- [ ] Create ColorPicker with brand constraints
- [ ] Enhance DatePicker with timezone conversion
- [ ] Implement PhoneInput with international formatting
- [ ] Build CurrencyInput with workspace currencies
- [ ] Create consistent validation API across all inputs
- [ ] Add comprehensive accessibility features

### Documentation Required

- Specialized input component API documentation
- Timezone handling implementation guide
- Color accessibility guidelines
- International formatting documentation
- Validation system integration guide
- Workspace context usage examples

### Testing Requirements

- Timezone conversion accuracy tests
- Color picker functionality tests
- International formatting tests
- Accessibility compliance tests
- Validation system integration tests
- Cross-browser compatibility tests
- Performance tests for complex inputs

### Integration Points

- Integration with existing form validation system
- Timezone service integration
- Color system and theme integration
- International formatting library integration
- Workspace context provider integration

## Deliverables

- TimePicker component with timezone support
- ColorPicker component with brand constraints
- Enhanced DatePicker with timezone conversion
- PhoneInput and CurrencyInput components
- Comprehensive Storybook stories
- Accessibility compliance validation
- Performance optimization for complex inputs

## Component Specifications

typescript

```typescript
interface TimePickerProps {
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  format?: '12h' | '24h'
  timezone?: string
  value?: string
  onChange?: (value: string) => void
  validationState?: 'error' | 'warning' | 'success'
  helperText?: string
  errorMessage?: string
  label?: string
  required?: boolean
}

interface ColorPickerProps {
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  value?: string
  onChange?: (color: string) => void
  format?: 'hex' | 'rgb' | 'hsl'
  presets?: string[]
  allowCustom?: boolean
  validationState?: 'error' | 'warning' | 'success'
  helperText?: string
  errorMessage?: string
  label?: string
  required?: boolean
}

interface DatePickerProps {
  context?: 'consultant' | 'client' | 'admin' | 'neutral'
  value?: Date
  onChange?: (date: Date) => void
  timezone?: string
  format?: string
  minDate?: Date
  maxDate?: Date
  validationState?: 'error' | 'warning' | 'success'
  helperText?: string
  errorMessage?: string
  label?: string
  required?: boolean
}
```

---

## Story 2.1.4: Range Input Components

### Overview

Create range input components (Slider, DualRangeSlider) for numeric input with workspace context support.

### AI Developer Prompt

You are creating range input components for THE WHEEL design system. Building on the existing form input architecture, you need to create slider and dual-range components for numeric input.

### Context

- Existing form input components with workspace context support
- Sophisticated theming system with CSS variables
- Need range inputs for time tracking, pricing, and analytics
- Accessibility requirements for keyboard and screen reader support
- Integration with form validation and real-time collaboration

### Requirements

#### 1. Create range input component variants:

- Single-value slider with customizable range
- Dual-range slider for min/max selections
- Stepped slider for discrete values
- Vertical slider for compact layouts
- Circular slider for specialized use cases

#### 2. Implement workspace context integration:

- Theme-based styling for different contexts
- Workspace-specific value formatting
- Permission-based interaction controls
- Context-aware validation rules
- Real-time value synchronization

#### 3. Create advanced range features:

- Custom tick marks and labels
- Value tooltips and formatting
- Smooth animation and transitions
- Touch and mobile optimization
- Keyboard navigation support

### Specific Tasks

- [ ] Create Slider component with single value
- [ ] Implement DualRangeSlider component
- [ ] Set up stepped and vertical variants
- [ ] Create circular slider component
- [ ] Implement accessibility features
- [ ] Set up real-time value synchronization

### Documentation Required

- Range input component API documentation
- Accessibility implementation guide
- Workspace context usage examples
- Custom styling guidelines
- Performance optimization tips

### Testing Requirements

- Range input interaction tests
- Accessibility compliance tests
- Workspace context validation tests
- Performance and animation tests
- Touch and mobile interaction tests

### Integration Points

- Integration with existing form system
- Workspace context provider integration
- Theme system integration
- Real-time collaboration integration
- Form validation integration

### Deliverables

- Complete range input component library
- Accessibility-compliant interactions
- Workspace context integration
- Performance-optimized animations
- Comprehensive documentation

### Performance Requirements

- Slider interaction response under 16ms (60fps)
- Animation performance maintains 60fps
- Touch response under 32ms
- Memory usage under 10MB
- Value synchronization under 100ms

---

## Story 2.1.5: Rich Text Input Components

### Overview

Create rich text input components for content creation and collaboration with workspace context support.

### AI Developer Prompt

You are creating rich text input components for THE WHEEL design system. Building on the existing form input system, you need to create WYSIWYG editors for content creation and collaboration.

### Context

- Existing form input components with validation
- Real-time collaboration system for multi-user editing
- Multiple workspace contexts requiring different editing capabilities
- Need for rich text in comments, descriptions, and documentation
- Accessibility requirements for screen readers and keyboard navigation

### Requirements

**1. Create rich text editor components:**

- Basic rich text editor with formatting toolbar
- Collaborative rich text editor with real-time updates
- Markdown editor with preview mode
- Code editor with syntax highlighting
- Comment editor with mention system

**2. Implement workspace context features:**

- Context-specific editing capabilities
- Permission-based feature access
- Workspace-specific formatting options
- Brand-aware styling and themes
- Context-aware content validation

**3. Create collaboration features:**

- Real-time multi-user editing
- Conflict resolution and merging
- User presence indicators
- Comment and suggestion system
- Version history and tracking

## Specific Tasks

- [ ] Create RichTextEditor component
- [ ] Implement CollaborativeEditor component
- [ ] Set up MarkdownEditor component
- [ ] Create CodeEditor component
- [ ] Implement mention and comment system
- [ ] Set up real-time collaboration

## Documentation Required

- Rich text editor API documentation
- Collaboration features guide
- Accessibility implementation
- Workspace context usage
- Performance optimization guide

### Testing Requirements

- Rich text editing functionality tests
- Collaboration feature tests
- Accessibility compliance tests
- Performance and memory tests
- Cross-browser compatibility tests

### Integration Points

- Integration with real-time collaboration system
- Workspace context provider integration
- Theme system integration
- Form validation integration
- Comment system integration

### Deliverables

- Complete rich text editor library
- Real-time collaboration features
- Accessibility-compliant editors
- Workspace context integration
- Performance-optimized editing

### Performance Requirements

- Editor initialization under 500ms
- Real-time updates under 100ms
- Memory usage under 50MB
- Typing response under 16ms (60fps)
- Collaboration sync under 200ms

---

## Timeline and Dependencies

### Timeline

- Week 1: Story 2.1.1 - Button Component Enhancement

- Week 1-2: Story 2.1.2 - Form Input Components
- Week 2-3: Story 2.1.3 - Specialized Input Components
- Week 3: Story 2.1.4 - Range Input Components
- Week 3-4: Story 2.1.5 - Rich Text Input Components

### Dependencies

- Epic 1.1 (Monorepo Architecture Setup) - Complete
- Epic 1.2 (Storybook Foundation) - Complete
- Workspace context system operational
- Theme system integrated

## Success Metrics

- All input components support workspace contexts
- 100% accessibility compliance (WCAG 2.1 AA)
- Performance benchmarks met for all components
- Complete test coverage (90%+ for all components)
- Comprehensive documentation for all components
- Zero regression in existing functionality

## Risk Mitigation

- Maintain backward compatibility during enhancements
- Phase rollout with feature flags
- Comprehensive testing at each phase
- Regular accessibility audits
- Performance monitoring during development
- Clear migration paths for existing implementations