

Component Generation Templates - Automated Scaffolding System

GENERATION PHILOSOPHY

Consistent Architecture: Every generated component follows the same patterns, ensuring maintainability and predictability across the entire design system.

Context-Aware by Default: All components are generated with workspace context support built-in, future-proofing against new contexts.

Quality First: Generated components include comprehensive testing, accessibility, and performance optimization from the start.

PLOP.JS CONFIGURATION

Main Plopfile


```
// plopfile.js
```

```
const path = require('path')
```

```
const { getRegisteredContexts } = require('./src/contexts/ContextRegistry')
```

```
module.exports = function (plop) {
```

```
  // Set helpers
```

```
  plop.setHelper('capitalize', (text) => text.charAt(0).toUpperCase() + text.slice(1))
```

```
  plop.setHelper('lowercase', (text) => text.toLowerCase())
```

```
  plop.setHelper('kebabCase', (text) => text.replace(/([a-z])([A-Z])/g, '$1-$2').toLowerCase())
```

```
  plop.setHelper('camelCase', (text) => text.replace(/-([a-z])/g, (g) => g[1].toUpperCase()))
```

```
  plop.setHelper('pascalCase', (text) => text.replace(/(^|-)([a-z])/g, (g) => g[1] ? g[1].toUpperCase() : g[0].toUpperCase())
```

```
  plop.setHelper('contextList', () => getRegisteredContexts().map(ctx => ctx.id).join(', '))
```

```
  plop.setHelper('timestamp', () => new Date().toISOString())
```

```
  // Component generator
```

```
  plop.setGenerator('component', {
```

```
    description: 'Create a new component with all necessary files',
```

```
    prompts: [
```

```
      {
```

```
        type: 'input',
```

```
        name: 'name',
```

```
        message: 'Component name (e.g., Button, UserCard):',
```

```
        validate: (input) => {
```

```
          if (!input) return 'Component name is required'
```

```
          if (!/^[A-Z][a-zA-Z0-9]*$/i.test(input)) {
```

```
            return 'Component name must be PascalCase (e.g., Button, UserCard)'
```

```
          }
```

```
          return true
```

```
        }
```

```
      },
```

```
      {
```

```
        type: 'list',
```

```
        name: 'category',
```

```
        message: 'Component category:',
```

```
        choices: [
```

```
          { name: 'Atom (Basic element)', value: 'atoms' },
```

```
          { name: 'Molecule (Simple combination)', value: 'molecules' },
```

```
          { name: 'Organism (Complex component)', value: 'organisms' },
```

```
          { name: 'Template (Page layout)', value: 'templates' }
```

```
        ]
```

```
      },
```

```
      {
```

```
        type: 'input',
```

```
name: 'description',
message: 'Component description:',
default: (answers) => `A ${answers.category.slice(0, -1)} component with workspace context awareness`
},
{
  type: 'confirm',
  name: 'hasVariants',
  message: 'Does this component have variants?',
  default: true
},
{
  type: 'input',
  name: 'variants',
  message: 'Enter variant names (comma-separated):',
  when: (answers) => answers.hasVariants,
  default: 'default, primary, secondary',
  filter: (input) => input.split(',').map(v => v.trim())
},
{
  type: 'confirm',
  name: 'hasChildren',
  message: 'Does this component accept children?',
  default: (answers) => answers.category === 'atoms'
},
{
  type: 'confirm',
  name: 'isInteractive',
  message: 'Is this component interactive (clickable, focusable)?',
  default: (answers) => answers.category === 'atoms'
},
{
  type: 'checkbox',
  name: 'features',
  message: 'Select additional features:',
  choices: [
    { name: 'Loading states', value: 'loading' },
    { name: 'Disabled states', value: 'disabled' },
    { name: 'Size variants', value: 'sizes' },
    { name: 'Icon support', value: 'icons' },
    { name: 'Animation support', value: 'animations' },
    { name: 'Form integration', value: 'forms' },
    { name: 'Keyboard navigation', value: 'keyboard' },
    { name: 'Touch/mobile support', value: 'touch' }
  ]
}
```

```

},
{
  type: 'input',
  name: 'package',
  message: 'Package location:',
  default: (answers) => `packages/${answers.category}`,
  validate: (input) => {
    if (!input) return 'Package location is required'
    return true
  }
}
],
actions: [
  // Component file
  {
    type: 'add',
    path: '{{package}}/src/{{kebabCase name}}{{pascalCase name}}.tsx',
    templateFile: 'templates/component.hbs'
  },
  // Types file
  {
    type: 'add',
    path: '{{package}}/src/{{kebabCase name}}/types.ts',
    templateFile: 'templates/types.hbs'
  },
  // Styles file
  {
    type: 'add',
    path: '{{package}}/src/{{kebabCase name}}/styles.ts',
    templateFile: 'templates/styles.hbs'
  },
  // Story file
  {
    type: 'add',
    path: '{{package}}/src/{{kebabCase name}}{{pascalCase name}}.stories.tsx',
    templateFile: 'templates/stories.hbs'
  },
  // Test file
  {
    type: 'add',
    path: '{{package}}/src/{{kebabCase name}}{{pascalCase name}}.test.tsx',
    templateFile: 'templates/test.hbs'
  },
  // Integration test file

```

```

{
  type: 'add',
  path: '{{package}}/src/{{kebabCase name}}/{{pascalCase name}}.integration.test.tsx',
  templateFile: 'templates/integration-test.hbs'
},
// Accessibility test file
{
  type: 'add',
  path: '{{package}}/src/{{kebabCase name}}/{{pascalCase name}}.a11y.test.tsx',
  templateFile: 'templates/a11y-test.hbs'
},
// Index file
{
  type: 'add',
  path: '{{package}}/src/{{kebabCase name}}/index.ts',
  templateFile: 'templates/index.hbs'
},
// Documentation file
{
  type: 'add',
  path: '{{package}}/src/{{kebabCase name}}/README.md',
  templateFile: 'templates/readme.hbs'
},
// Update main index
{
  type: 'modify',
  path: '{{package}}/src/index.ts',
  pattern: /(\\ PLOP_INJECT_EXPORT)/g,
  template: 'export { {{pascalCase name}} } from \'./{{kebabCase name}}\'\\n$1'
}
]
})

```

```

// Context generator
plop.setGenerator('context', {
  description: 'Create a new workspace context',
  prompts: [
    {
      type: 'input',
      name: 'id',
      message: 'Context ID (e.g., enterprise, mobile):',
      validate: (input) => {
        if (!input) return 'Context ID is required'
        if (!/^[a-z][a-z0-9-]*$/i.test(input)) {

```

```

    return 'Context ID must be lowercase with hyphens (e.g., enterprise, mobile-app)'
  }
  return true
}
},
{
  type: 'input',
  name: 'name',
  message: 'Context display name:',
  default: (answers) => answers.id.split('-').map(word =>
    word.charAt(0).toUpperCase() + word.slice(1)
  ).join(' '),
},
{
  type: 'input',
  name: 'description',
  message: 'Context description:',
  default: (answers) => `${answers.name} workspace context`,
},
{
  type: 'input',
  name: 'color',
  message: 'Primary color (hex):',
  default: '#6B7280',
  validate: (input) => {
    if (!/^#[0-9A-F]{6}$/i.test(input)) {
      return 'Please enter a valid hex color (e.g., #6B7280)'
    }
    return true
  }
},
{
  type: 'input',
  name: 'icon',
  message: 'Icon name (Lucide React):',
  default: 'circle'
},
{
  type: 'input',
  name: 'parentContext',
  message: 'Parent context (optional):',
  default: ''
},
{

```

```

    type: 'checkbox',
    name: 'features',
    message: 'Select features available in this context:',
    choices: [
      { name: 'Workspace Management', value: 'workspace-management' },
      { name: 'Client Management', value: 'client-management' },
      { name: 'Billing', value: 'billing' },
      { name: 'Analytics', value: 'analytics' },
      { name: 'User Management', value: 'user-management' },
      { name: 'Settings', value: 'settings' },
      { name: 'Marketplace', value: 'marketplace' },
      { name: 'Communications', value: 'communications' }
    ]
  },
],
actions: [
  // Context definition
  {
    type: 'add',
    path: 'src/contexts/{{kebabCase id}}/{{pascalCase id}}Context.tsx',
    templateFile: 'templates/context.hbs'
  },
  // Theme file
  {
    type: 'add',
    path: 'src/themes/{{kebabCase id}}.ts',
    templateFile: 'templates/theme.hbs'
  },
  // CSS variables
  {
    type: 'add',
    path: 'src/styles/contexts/{{kebabCase id}}.css',
    templateFile: 'templates/context-styles.hbs'
  },
  // Register context
  {
    type: 'modify',
    path: 'src/contexts/ContextRegistry.ts',
    pattern: /(\\ PLOP_INJECT_CONTEXT)/g,
    template: `contextRegistry.register({
id: '{{id}}',
name: '{{name}}',
description: '{{description}}',
theme: '{{kebabCase id}}-light',

```



```

icon: '{{icon}}',
color: '{{color}}',
{{#if parentContext}}parentContext: '{{parentContext}}',{{/if}}
features: [{{#each features}}{{this}}{{#unless @last}}, {{/unless}}{{/each}}
})

```

```

$1`
  }
]
})

```

// Story generator

```

plop.setGenerator('story', {
  description: 'Add a new story to an existing component',
  prompts: [
    {
      type: 'input',
      name: 'componentPath',
      message: 'Path to component file:',
      validate: (input) => {
        if (!input) return 'Component path is required'
        return true
      }
    },
    {
      type: 'input',
      name: 'storyName',
      message: 'Story name:',
      validate: (input) => {
        if (!input) return 'Story name is required'
        if (!/^[A-Z][a-zA-Z0-9]*$/.test(input)) {
          return 'Story name must be PascalCase'
        }
        return true
      }
    },
    {
      type: 'input',
      name: 'description',
      message: 'Story description:',
      default: (answers) => `${answers.storyName} story example`
    }
  ],
  actions: [

```

```

{
  type: 'modify',
  path: '{{componentPath}}.stories.tsx',
  pattern: /(\\ PLOP_INJECT_STORY)/g,
  template: `export const {{pascalCase storyName}}: Story = {
args: {
  // Add story args here
},
render: (args) => (
  <div>
    {/* Add story implementation here */}
  </div>
),
parameters: {
  docs: {
    description: {
      story: '{{description}}',
    },
  },
},
}

$1`
  }
]
})
}

```



COMPONENT TEMPLATES

Main Component Template

handlebars

```

{{!-- templates/component.hbs --}}
import React from 'react'
import { cn } from '@lib/utils'
import { useWorkspaceContext } from '@contexts/WorkspaceContext'
{{
  #if
    features.includes 'icons'}}
import { Icon } from '@components/ui/icon'
{{
  /if
}}
{{
  #if
    features.includes 'loading'}}
import { Spinner } from '@components/ui/spinner'
{{
  /if
}}
import { {{pascalCase name}}Props } from './types'
import { {{camelCase name}}Styles } from './styles'

/**
 * {{description}}
 *
 * @component
 * @example
 * ```tsx
 * <{{pascalCase name}}{
  #if
    hasVariants}} variant="primary"{{
  /if
  }}{
  #if
    hasChildren}}>
 *   Content
 * </{{pascalCase name}}>
 * ```
 */
export const {{pascalCase name}} = React.forwardRef<
  {{
  #if
    isInteractive}}HTMLButtonElement{{else}}HTMLDivElement{{
  /if
  }},

```

```

    {{pascalCase name}}Props
  >({{
    {{
  #if
    hasVariants}}variant = '{{
  #if
    variants}}>{{variants.[0]}}{{else}}default{{
  /if
    }}',{{
  /if
    }}
    {{
  #if
    features.includes 'sizes'}}size = 'md',{{
  /if
    }}
    context: contextProp,
    {{
  #if
    features.includes 'loading'}}loading = false,{{
  /if
    }}
    {{
  #if
    features.includes 'disabled'}}disabled = false,{{
  /if
    }}
    {{
  #if
    hasChildren}}children,{{
  /if
    }}
    className,
    {{
  #if
    isInteractive}}onClick,{{
  /if
    }}
    ...props
  }, ref) => {
    const { currentContext } = useWorkspaceContext()
    const context = contextProp || currentContext

    {{
  #if

```

```

features.includes 'loading'}}
const isLoading = loading
const isDisabled = disabled || isLoading
{{else}}
{{
#if
features.includes 'disabled'}}
const isDisabled = disabled
{{
/!f
}}
{{
/!f
}}

{{
#if
isInteractive}}
const handleClick = (event: React.MouseEvent<HTMLButtonElement>) => {
  {{
  #if
  features.includes 'disabled'}}
  if (isDisabled) return
  {{
  /!f
  }}
  onClick?.(event)
  }

const handleKeyDown = (event: React.KeyboardEvent<HTMLButtonElement>) => {
  {{
  #if
  features.includes 'keyboard'}}
  if (event.key === 'Enter' || event.key === ' ') {
    event.preventDefault()
    handleClick(event as any)
  }
  {{
  /!f
  }}
  }
  {{
  /!f
  }}

```

```

return (
  <{{
    #if
    isInteractive}}button{{else}}div{{
    /if
  }}
  ref={ref}
  className={cn(
    {{camelCase name}}Styles.base,
    {{
    #if
    hasVariants}}{{camelCase name}}Styles.variants[variant],{{
    /if
  }}
    {{
    #if
    features.includes 'sizes'}}{{camelCase name}}Styles.sizes[size],{{
    /if
  }}
    {{camelCase name}}Styles.contexts[context] || {{camelCase name}}Styles.contexts.default,
    {{
    #if
    features.includes 'disabled'}}
      isDisabled && {{camelCase name}}Styles.disabled,
      {{
    /if
  }}
    {{
    #if
    features.includes 'loading'}}
      isLoading && {{camelCase name}}Styles.loading,
      {{
    /if
  }}
    className
  ))
  {{
    #if
    isInteractive}}
      onClick={handleClick}
      {{
    #if
    features.includes 'keyboard'}}
      onKeyDown={handleKeyDown}

```

```

    {{
  /if
}}

    {{
  #if
    features.includes 'disabled'}}
      disabled={isDisabled}
    {{
  /if
}}

    {{
  #if
    features.includes 'loading'}}
      aria-busy={isLoading}
    {{
  /if
}}

      type="button"
    {{
  /if
}}

      data-component="{{kebabCase name}}"
      data-context={context}
    {{
  #if
    hasVariants}}data-variant={variant}{{
  /if
}}

    {{
  #if
    features.includes 'sizes'}}data-size={size}{{
  /if
}}

    {{
  #if
    features.includes 'testid'}}data-testid="{{kebabCase name}}"{{
  /if
}}

    {...props}
  >
    {{
  #if
    features.includes 'loading'}}
      {isLoading && (
        <Spinner

```



```

        className="{{camelCase name}}-spinner"
        size="sm"
        aria-label="Loading..."
    />
  })
  {{
    /if
  }}
  {{
    #if
    hasChildren}}
    {{{
    #if
    features.includes 'loading'}}!isLoading && {{
    /if
  }}children}
  {{
    /if
  }}
  </{{
    #if
    isInteractive}}button{{else}}div{{
    /if
  }}>
  )
})

{{pascalCase name}}.displayName = '{{pascalCase name}}'

```

Types Template

handlebars

```

{{!-- templates/types.hbs --}}
import { type VariantProps } from 'class-variance-authority'
import { {{camelCase name}}Styles } from './styles'

export interface {{pascalCase name}}Props
  extends {{
    #if
    isInteractive}}React.ButtonHTMLAttributes<HTMLButtonElement>{{else}}React.HTMLAttributes<HTMLDivElement>{{/if
  }},
  VariantProps<typeof {{camelCase name}}Styles.base> {
    /**
     * Component variant
     * @default '{{
    #if
    variants}}'{{variants.[0]}}{{else}}default{{
    /if
  }}'
     */
    {{
    #if
    hasVariants}}
    variant?: {{
    #each
    variants}}'{{this}}'{{
    #unless
    @last}} | {{
    /unless
  }}{{
    /each
  }}
    {{
    /if
  }}

    {{
    #if
    features.includes 'sizes'}}
    /**
     * Component size
     * @default 'md'
     */
    size?: 'sm' | 'md' | 'lg'
  }

```

```
{{  
/if  
}}
```

```
/**  
 * Workspace context for theming and behavior  
 * @default Current workspace context  
 */  
context?: string
```

```
{{  
#if  
features.includes 'loading'}}  
/**  
 * Loading state  
 * @default false  
 */  
loading?: boolean  
{{  
/if  
}}
```

```
{{  
#if  
features.includes 'disabled'}}  
/**  
 * Disabled state  
 * @default false  
 */  
disabled?: boolean  
{{  
/if  
}}
```

```
{{  
#if  
hasChildren}}  
/**  
 * Component content  
 */  
children?: React.ReactNode  
{{  
/if  
}}
```

```

{{
  #if
  isInteractive}}
  /**
   * Click handler
   */
  onClick?: (event: React.MouseEvent<HTMLButtonElement>) => void
  {{
  #if
  }}

```

```

  /**
   * Additional CSS classes
   */
  className?: string
}

```

```

{{
  #if
  hasVariants}}
  export type {{pascalCase name}}Variant = {{
  #each
  variants}}'{{this}}'{{
  #unless
  @last}} | {{
  /unless
  }}{{
  /each
  }}
  {{
  #if
  }}

```

```

{{
  #if
  features.includes 'sizes'}}
  export type {{pascalCase name}}Size = 'sm' | 'md' | 'lg'
  {{
  #if
  }}

```

```

  export type {{pascalCase name}}Context = string

```

```
// Component display name for debugging  
export const {{pascalCase name}}DisplayName = '{{pascalCase name}}'
```

Styles Template

handlebars

```
{{!-- templates/styles.hbs --}}
```

```
import { cva } from 'class-variance-authority'
```

```
import { getRegisteredContexts } from '@contexts/ContextRegistry'
```

```
// Generate context styles dynamically
```

```
const generateContextStyles = () => {
```

```
  const contexts = getRegisteredContexts()
```

```
  const contextStyles: Record<string, string> = {
```

```
    default: 'bg-gray-100 text-gray-900 border-gray-300 hover:bg-gray-200'
```

```
  }
```

```
  contexts.forEach(ctx => {
```

```
    contextStyles[ctx.id] = `bg-${ctx.id}-100 text-${ctx.id}-900 border-${ctx.id}-300 hover:bg-${ctx.id}-200`  
  })
```

```
  return contextStyles
```

```
}
```

```
export const {{camelCase name}}Styles = {
```

```
  base: cva(
```

```
    [
```

```
      // Base styles
```

```
      'inline-flex items-center justify-center',
```

```
      'rounded-md border transition-colors duration-200',
```

```
      'focus:outline-none focus:ring-2 focus:ring-offset-2',
```

```
      {{
```

```
    #if
```

```
    features.includes 'disabled'}}
```

```
      'disabled:opacity-50 disabled:cursor-not-allowed',
```

```
      {{
```

```
    /if
```

```
  }}  
  {{
```

```
  #if
```

```
  features.includes 'loading'}}
```

```
    'data-[loading=true]:cursor-wait',
```

```
    {{
```

```
  /if
```

```
}}
```

```
  // Accessibility
```

```
  'focus-visible:ring-2 focus-visible:ring-offset-2',
```

```
  // Typography
```

```
  'font-medium text-sm leading-none',
```



```

    // Animation
    {{
  #if
features.includes 'animations'}}
    'transition-all duration-200 ease-in-out',
    {{
  /if
}}
  ],
  {
    variants: {
      {{
  #if
hasVariants}}
        variant: {
          {{
  #each
variants}}
            {{this}}: [
              {{
  #if
@first}}
                // Primary variant styles
                'bg-primary-500 text-white border-primary-500',
                'hover:bg-primary-600 hover:border-primary-600',
                'focus:ring-primary-500'
                {{else if (eq this 'secondary')}}
                // Secondary variant styles
                'bg-secondary-100 text-secondary-900 border-secondary-300',
                'hover:bg-secondary-200 hover:border-secondary-400',
                'focus:ring-secondary-500'
                {{else if (eq this 'outline')}}
                // Outline variant styles
                'bg-transparent border-current',
                'hover:bg-current hover:bg-opacity-10',
                'focus:ring-current'
                {{else if (eq this 'ghost')}}
                // Ghost variant styles
                'bg-transparent border-transparent',
                'hover:bg-current hover:bg-opacity-10',
                'focus:ring-current'
                {{else}}
                // Default variant styles
                'bg-gray-100 text-gray-900 border-gray-300',

```

```

        'hover:bg-gray-200 hover:border-gray-400',
        'focus:ring-gray-500'
      {{
    /if
  }}
  ],
  {{
/each
}}
  },
  {{
/if
}}

  {{
  #if
  features.includes 'sizes'}}
    size: {
      sm: [
        'h-8 px-3 text-xs',
        'min-w-[2rem]'
      ],
      md: [
        'h-10 px-4 text-sm',
        'min-w-[2.5rem]'
      ],
      lg: [
        'h-12 px-6 text-base',
        'min-w-[3rem]'
      ]
    },
    {{
  /if
  }}
  },
  defaultVariants: {
    {{
  #if
  hasVariants}}variant: '{{variants.[0]}}',{{
  /if
  }}
  {{
  #if

```

```
features.includes 'sizes'}}size: 'md'{{
```

```
/if
```

```
}}
```

```
}
```

```
}
```

```
),
```

```
// Context-specific styles
```

```
contexts: generateContextStyles(),
```

```
{{
```

```
#if
```

```
features.includes 'disabled'}}
```

```
// Disabled styles
```

```
disabled: [
```

```
'opacity-50 cursor-not-allowed',
```

```
'pointer-events-none'
```

```
],
```

```
{{
```

```
/if
```

```
}}
```

```
{{
```

```
#if
```

```
features.includes 'loading'}}
```

```
// Loading styles
```

```
loading: [
```

```
'cursor-wait',
```

```
'opacity-75'
```

```
],
```

```
{{
```

```
/if
```

```
}}
```

```
{{
```

```
#if
```

```
hasVariants}}
```

```
// Variant-specific styles
```

```
variants: {
```

```
{{
```

```
#each
```

```
variants}}
```

```
{{this}}: {{camelCase ../name}}-{{this}}'{{
```

```
#unless
```

```
@last}},{{
```

```
/unless
```

```
}}
```

```
{{
```

```
/each
```

```
}}
```

```
},
```

```
{{
```

```
/if
```

```
}}
```

```
{{
```

```
#if
```

```
features.includes 'sizes'}}
```

```
// Size-specific styles
```

```
sizes: {
```

```
  sm: '{{camelCase name}}-sm',
```

```
  md: '{{camelCase name}}-md',
```

```
  lg: '{{camelCase name}}-lg'
```

```
}
```

```
{{
```

```
/if
```

```
}}
```

```
}
```

```
// CSS class names for external styling
```

```
export const {{camelCase name}}Classes = {
```

```
  root: '{{kebabCase name}}',
```

```
  {{
```

```
#if
```

```
hasVariants}}
```

```
  {{
```

```
#each
```

```
variants}}
```

```
  {{camelCase this}}: '{{kebabCase ../name}}-{{kebabCase this}}',
```

```
  {{
```

```
/each
```

```
}}
```

```
  {{
```

```
/if
```

```
}}
```

```
  {{
```

```
#if
```

```
features.includes 'sizes'}}
```

```
sm: '{{kebabCase name}}-sm',
md: '{{kebabCase name}}-md',
lg: '{{kebabCase name}}-lg',
{{
/if
}}
{{
#if
features.includes 'disabled'}}
disabled: '{{kebabCase name}}-disabled',
{{
/if
}}
{{
#if
features.includes 'loading'}}
loading: '{{kebabCase name}}-loading'
{{
/if
}}
}
```

STORY TEMPLATES

Comprehensive Story Template

handlebars

```
{{!-- templates/stories.hbs --}}
```

```
import type { Meta, StoryObj } from '@storybook/react'
```

```
import { {{pascalCase name}} } from './{{pascalCase name}}'
```

```
import { createUniversalStory, createStandardStories } from '@story-templates/UniversalStoryTemplate'
```

```
import { getRegisteredContexts } from '@/contexts/ContextRegistry'
```

```
// Story configuration
```

```
const config = {
```

```
  title: '{{pascalCase name}}',
```

```
  component: {{pascalCase name}},
```

```
  category: '{{category}}' as const,
```

```
  description: '{{description}}',
```

```
  {{
```

```
    #if
```

```
    hasVariants}}
```

```
  variants: [
```

```
    {{
```

```
    #each
```

```
    variants}}
```

```
    {
```

```
      name: '{{pascalCase this}}',
```

```
      props: {
```

```
        variant: '{{this}}'{{
```

```
    #if
```

```
    ../hasChildren}},
```

```
    children: '{{pascalCase this}} {{pascalCase ../name}}'{{
```

```
  /if
```

```
}}
```

```
  },
```

```
  description: '{{pascalCase this}} variant of {{pascalCase ../name}} component.'
```

```
  {{
```

```
  #unless
```

```
  @last}},{{
```

```
  /unless
```

```
}}
```

```
  {{
```

```
  /each
```

```
}}
```

```
  ],
```

```
  {{
```

```
  /if
```

```
}}
```

```
  accessibility: {
```

```
    label: '{{pascalCase name}} Component',
```

```

    description: '{{description}} with full accessibility support.'
  },
  performance: {
    expectedRenderTime: {{
      #if
      (eq category 'atoms')}}5{{else if (eq category 'molecules')}}10{{else}}20{{
      /if
    }},
    bundleSize: {{
      #if
      (eq category 'atoms')}}2048{{else if (eq category 'molecules')}}4096{{else}}8192{{
      /if
    }}
  }
}

```

```

// Create universal story configuration
const meta: Meta<typeof {{pascalCase name}}> = createUniversalStory(config)
export default meta

```

```

type Story = StoryObj<typeof meta>

```

```

// Generate standard stories
const standardStories = createStandardStories(config)
export const {
  Default,
  {{
    #if
    hasVariants}}
  {{
    #each
    variants}}
    {{pascalCase this}},
    {{
    /each
  }}
  {{
    /if
  }}
  AllContexts,
  InteractiveStates
} = standardStories

```



```

{{
  #if
  features.includes 'loading'}}
// Loading states story
export const LoadingStates: Story = {
  render: () => (
    <div className="flex gap-4 items-center">
      <{{pascalCase name}}>
        #if
        hasChildren}}>Default{{
        /if
      }}</{{pascalCase name}}>
      <{{pascalCase name}} loading{{
        #if
        hasChildren}}>Loading...{{
        /if
      }}</{{pascalCase name}}>
      <{{pascalCase name}} disabled{{
        #if
        hasChildren}}>Disabled{{
        /if
      }}</{{pascalCase name}}>
    </div>
  ),
  parameters: {
    docs: {
      description: {
        story: 'Different loading and disabled states of the {{pascalCase name}} component.',
      },
    },
  },
}
{{
  /if
}}

{{
  #if
  features.includes 'sizes'}}
// Size variants story
export const SizeVariants: Story = {
  render: () => (
    <div className="flex gap-4 items-center">

```

```

    <{{pascalCase name}} size="sm"{{
  #if
    hasChildren}}>Small{{
  /if
}}</{{pascalCase name}}>
    <{{pascalCase name}} size="md"{{
  #if
    hasChildren}}>Medium{{
  /if
}}</{{pascalCase name}}>
    <{{pascalCase name}} size="lg"{{
  #if
    hasChildren}}>Large{{
  /if
}}</{{pascalCase name}}>
  </div>
),
parameters: {
  docs: {
    description: {
      story: 'Different size variants of the {{pascalCase name}} component.',
    },
  },
},
}
{{
  /if
}}

{{
  #if
    features.includes 'icons'}}
// With icons story
export const WithIcons: Story = {
  render: () => (
    <div className="flex gap-4 items-center">
      <{{pascalCase name}}>
        <Icon name="plus" className="w-4 h-4 mr-2" />
        {{
  #if
    hasChildren}}With Icon{{
  /if
}}
      </{{pascalCase name}}>

```

```

<{{pascalCase name}} variant="secondary">
  <Icon name="download" className="w-4 h-4 mr-2" />
  {{
    #if
    hasChildren}}Download{{
    /if
  }}
  </{{pascalCase name}}>
</div>
),
parameters: {
  docs: {
    description: {
      story: '{{pascalCase name}} components with icons for enhanced visual communication.',
    },
  },
},
}
{{
  /if
}}

```

// Contextual usage story

```

export const ContextualUsage: Story = {
  render: () => {
    const contexts = getRegisteredContexts()

    return (
      <div className="space-y-6">
        {contexts.map(ctx => (
          <div key={ctx.id} className="p-4 border rounded-lg">
            <h3 className="font-semibold mb-3">{ctx.name} Context</h3>
            <div className="flex gap-2">
              <{{pascalCase name}} context={ctx.id}{{
                #if
                hasVariants}} variant="{{variants.[0]}}"{{
                /if
              }}>
                {{
                  #if
                  hasChildren}}Primary Action{{
                  /if
                }}
                </{{pascalCase name}}>

```

```

        {{
    #if
    hasVariants}}
        {{
    #if
    (gt variants.length 1)}}
        <{{pascalCase name}} context={ctx.id} variant="{{variants.[1]}}">
        {{
    #if
    hasChildren}}Secondary Action{{
    /if
    }}
        </{{pascalCase name}}>
        {{
    /if
    }}
        {{
    /if
    }}
        </div>
        </div>
    )))
</div>
)
},
parameters: {
  docs: {
    description: {
      story: 'Contextual usage examples showing how {{pascalCase name}} adapts to different workspace contexts
    },
  },
},
},
}

{{
  #if
  isInteractive}}
// Interactive demo story
export const InteractiveDemo: Story = {
  render: () => {
    const [clickCount, setClickCount] = React.useState(0)

    return (
      <div className="space-y-4">

```

```

    <{{pascalCase name}} onClick={() => setClickCount(prev => prev + 1)}>
      {{
        #if
        hasChildren}}
        Click me! (clicked {clickCount} times)
        {{
        /if
      }}
    </{{pascalCase name}}>

    <div className="text-sm text-gray-600">
      <p>This {{pascalCase name}} has been clicked {clickCount} times.</p>
      <p>Try interacting with it to see the click handler in action!</p>
    </div>
  </div>
)
},
parameters: {
  docs: {
    description: {
      story: 'Interactive demo showing the click functionality of {{pascalCase name}}.',
    },
  },
},
},
}
{{
/if
}}

{{
#if
features.includes 'forms'}}
// Form integration story
export const FormIntegration: Story = {
  render: () => (
    <form className="space-y-4 max-w-md">
      <div>
        <label className="block text-sm font-medium mb-2">
          Sample Form
        </label>
        <input
          type="text"
          className="w-full p-2 border rounded"
          placeholder="Enter some text..."

```

```

    />
  </div>

  <div className="flex gap-2">
    <{{pascalCase name}} type="submit">
      {{
        #if
        hasChildren}}Submit{{
        /if
      }}

    </{{pascalCase name}}>
    <{{pascalCase name}} type="reset"{{
      #if
      hasVariants}} variant="secondary"{{
      /if
    }}>
      {{
        #if
        hasChildren}}Reset{{
        /if
      }}

    </{{pascalCase name}}>
  </div>
</form>
),
parameters: {
  docs: {
    description: {
      story: 'Integration of {{pascalCase name}} with form elements.',
    },
  },
},
},
}
{{
/if
}}

// Accessibility demo story
export const AccessibilityDemo: Story = {
  render: () => (
    <div className="space-y-4">
      <div className="p-4 bg-blue-50 rounded-lg">
        <h3 className="font-semibold mb-2">Accessibility Features</h3>
        <ul className="text-sm space-y-1">

```

```

    <li>• Keyboard navigation support</li>
    <li>• Focus indicators</li>
    <li>• Screen reader compatibility</li>
    <li>• ARIA attributes</li>
    {{
#if
features.includes 'loading'}}
    <li>• Loading state announcements</li>
    {{
/if
}}
    </ul>
</div>

<div className="flex gap-4">
  <{{pascalCase name}}{
#if
hasChildren}}
    aria-label="Primary action button"
    aria-describedby="primary-help"
  >
    Primary Action
  </{{pascalCase name}}>
  {{
#if
hasChildren}}
    <div id="primary-help" className="sr-only">
      This button performs the primary action
    </div>
    {{
/if
}}
    {{
/if
}}

  <{{pascalCase name}}{
#if
hasVariants}} variant="secondary"{{
/if
}}{
  {{
#if
hasChildren}}
    aria-label="Secondary action button"

```

```

        aria-describedby="secondary-help"
    >
        Secondary Action
    </{{pascalCase name}}>
    {{
#if
hasChildren}}
    <div id="secondary-help" className="sr-only">
        This button performs a secondary action
    </div>
    {{
/if
}}
    {{
/if
}}
    </div>
</div>
),
parameters: {
  docs: {
    description: {
      story: 'Accessibility features and proper ARIA implementation.',
    },
  },
},
},
}

// PLOP_INJECT_STORY

```

TEST TEMPLATES

Unit Test Template

handlebars

```

{{!-- templates/test.hbs --}}
import { render, screen } from '@testing-library/react'
import { userEvent } from '@testing-library/user-event'
import { composeStories } from '@storybook/testing-react'
import { axe, toHaveNoViolations } from 'jest-axe'
import { customRender, testAllContexts } from '@test/utils/context-testing'
import { {{pascalCase name}} } from './{{pascalCase name}}'
import * as stories from './{{pascalCase name}}.stories'

// Extend Jest matchers
expect.extend(toHaveNoViolations)

// Compose stories for testing
const {
  Default,
  {{
    #if
    hasVariants}}
  {{
    #each
    variants}}
    {{pascalCase this}},
    {{
    /each
  }}
  {{
    /if
  }}
  AllContexts,
  InteractiveStates
} = composeStories(stories)

describe('{{pascalCase name}} Component', () => {
  // Basic rendering tests
  describe('Rendering', () => {
    it('renders with default props', () => {
      render(<Default />)
      expect(screen.getByTestId('{{kebabCase name}}')).toBeInTheDocument()
    })

    {{
    #if
    hasVariants}}

```

```

it('renders with all variants', () => {
  render(<AllContexts />)
  {{
#each
variants}}
    expect(screen.getByText('{{pascalCase this}}')).toBeInTheDocument()
    {{
/each
}}
  })
  {{
/if
}}

it('renders with custom className', () => {
  render(<{{pascalCase name}} className="custom-class"{{
#if
hasChildren}}>Test{{
/if
}} />)
  expect(screen.getByTestId('{{kebabCase name}}')).toHaveClass('custom-class')
  })

  {{
#if
hasChildren}}
  it('renders children correctly', () => {
    render(<{{pascalCase name}}>Test Content</{{pascalCase name}}>)
    expect(screen.getByText('Test Content')).toBeInTheDocument()
  })
  {{
/if
}}
  })

// Context-aware testing
describe('Workspace Context', () => {
  testAllContexts(
    <{{pascalCase name}}>{{
#if
hasChildren}}>Test{{
/if
}} />,
    (context) => {

```

```

it(`applies ${context} context styling`, () => {
  customRender(
    <{{pascalCase name}}{
    #if
    hasChildren}}>Test{{
    /if
  }} />,
    { context }
  )

  const element = screen.getByTestId('{{kebabCase name}}')
  expect(element).toHaveAttribute('data-context', context)
  expect(element).toHaveClass(`workspace-${context}`)
})
}
)

it('uses provided context over workspace context', () => {
  customRender(
    <{{pascalCase name}} context="admin"{{
    #if
    hasChildren}}>Test{{
    /if
  }} />,
    { context: 'consultant' }
  )

  const element = screen.getByTestId('{{kebabCase name}}')
  expect(element).toHaveAttribute('data-context', 'admin')
})

it('handles unknown contexts gracefully', () => {
  customRender(
    <{{pascalCase name}} context="unknown-context"{{
    #if
    hasChildren}}>Test{{
    /if
  }} />,
    { context: 'consultant' }
  )

  const element = screen.getByTestId('{{kebabCase name}}')
  expect(element).toHaveAttribute('data-context', 'unknown-context')
})

```

```

})

{{
  #if
  hasVariants}}
  // Variant testing
  describe('Variants', () => {
    {{
      #each
      variants}}
      it('applies {{this}} variant correctly', () => {
        render(<{{pascalCase ../name}} variant="{{this}}"{{
          #if
          ../hasChildren}}>Test{{
            /if
          }} />)

        const element = screen.getByTestId('{{kebabCase ../name}}')
        expect(element).toHaveAttribute('data-variant', '{{this}}')
        expect(element).toHaveClass('{{kebabCase ../name}}-{{this}}')
      })
    }}
  /each
}}
})
{{
  #if
  features.includes 'sizes'}}
  // Size testing
  describe('Sizes', () => {
    ['sm', 'md', 'lg'].forEach(size => {
      it(`applies ${size} size correctly`, () => {
        render(<{{pascalCase name}} size={size}{{
          #if
          hasChildren}}>Test{{
            /if
          }} />)

        const element = screen.getByTestId('{{kebabCase name}}')
        expect(element).toHaveAttribute('data-size', size)
        expect(element).toHaveClass(`{{kebabCase name}}-${size}`)
      })
    })
  })

```

```
    })  
  })  
  {{  
/if  
}}
```

```
  {{  
#if  
isInteractive}}  
  // Interaction testing  
  describe('Interactions', () => {  
    it('handles click events', async () => {  
      const user = userEvent.setup()  
      const handleClick = vi.fn()  
  
      render(<{{pascalCase name}} onClick={handleClick}{{  
#if  
hasChildren}}>Click me{{  
/if  
}} />)
```

```
        await user.click(screen.getByRole('button'))  
        expect(handleClick).toHaveBeenCalledTimes(1)  
      })  
    }  
  })  
  {{  
#if  
features.includes 'keyboard'}}  
    it('handles keyboard events', async () => {  
      const user = userEvent.setup()  
      const handleClick = vi.fn()  
  
      render(<{{pascalCase name}} onClick={handleClick}{{  
#if  
hasChildren}}>Click me{{  
/if  
}} />)
```

```
        const button = screen.getByRole('button')  
        await user.tab()  
        expect(button).toHaveFocus()  
  
        await user.keyboard('[Enter]')  
        expect(handleClick).toHaveBeenCalledTimes(1)
```

```

    await user.keyboard('[Space]')
    expect(handleClick).toHaveBeenCalledTimes(2)
  })
  {{
    /if
  }}

  {{
    #if
    features.includes 'disabled'
    it('prevents interaction when disabled', async () => {
      const user = userEvent.setup()
      const handleClick = vi.fn()

      render(<{{pascalCase name}} disabled onClick={handleClick}{{
        #if
        hasChildren}}>Disabled{{
        /if
      }} />)

      await user.click(screen.getByRole('button'))
      expect(handleClick).not.toHaveBeenCalled()
    })
    {{
    /if
  }}

  {{
    #if
    features.includes 'loading'
    // Loading state testing
    describe('Loading States', () => {
      it('shows loading state correctly', () => {
        render(<{{pascalCase name}} loading{{
        #if
        hasChildren}}>Loading...{{
        /if
      }} />)

        const element = screen.getByTestId('{{kebabCase name}}')

```

```

    expect(element).toHaveAttribute('aria-busy', 'true')
    expect(element).toHaveClass('{{kebabCase name}}-loading')
  })

  it('prevents interaction during loading', async () => {
    const user = userEvent.setup()
    const handleClick = vi.fn()

    render(<{{pascalCase name}} loading onClick={handleClick}{{
  #if
  hasChildren}}>Loading...{{
  /if
}} />)

    await user.click(screen.getByRole('button'))
    expect(handleClick).not.toHaveBeenCalled()
  })
})
{{
  /if
}}

// Accessibility testing
describe('Accessibility', () => {
  it('has no accessibility violations', async () => {
    const { container } = render(<Default />)
    const results = await axe(container)
    expect(results).toHaveNoViolations()
  })

  {{
    #if
    isInteractive}}
    it('supports keyboard navigation', async () => {
      const user = userEvent.setup()
      const handleClick = vi.fn()

      render(<{{pascalCase name}} onClick={handleClick}{{
  #if
  hasChildren}}>Test{{
  /if
}} />)

      await user.tab()

```



```

    expect(screen.getByRole('button')).toHaveFocus()
  })

  it('has proper ARIA attributes', () => {
    render(<{{pascalCase name}} aria-label="Custom label"{{
      #if
      hasChildren}}>Test{{
      /if
    }} />)

    expect(screen.getByRole('button')).toHaveAttribute('aria-label', 'Custom label')
  })
  {{
  /if
  }}

  {{
  #if
  features.includes 'loading'}}
  it('announces loading state to screen readers', () => {
    render(<{{pascalCase name}} loading{{
      #if
      hasChildren}}>Loading...{{
      /if
    }} />)

    expect(screen.getByRole('button')).toHaveAttribute('aria-busy', 'true')
  })
  {{
  /if
  }}
  })

  // Performance testing
  describe('Performance', () => {
    it('renders quickly', () => {
      const startTime = performance.now()
      render(<{{pascalCase name}}{{
        #if
        hasChildren}}>Performance Test{{
        /if
      }} />)

      const endTime = performance.now()

      expect(endTime - startTime).toBeLessThan({{
        #if

```

```

(eq category 'atoms'))}}16{{else if (eq category 'molecules')}}32{{else}}64{{
/if
}}) // Performance budget
})

it('handles multiple re-renders efficiently', () => {
  const { rerender } = render(<{{pascalCase name}}{{
  #if
  hasChildren}}>Initial{{
  /if
}} />)

  const startTime = performance.now()
  for (let i = 0; i < 100; i++) {
    rerender(<{{pascalCase name}}{{
    #if
    hasChildren}}>Update {i}{{
    /if
  }} />)
  }
  const endTime = performance.now()

  expect(endTime - startTime).toBeLessThan(500) // 500ms for 100 renders
})
})
})

```

Integration Test Template

handlebars

```

{{!-- templates/integration-test.hbs --}}
import { render, screen } from '@testing-library/react'
import { userEvent } from '@testing-library/user-event'
import { customRender } from '@test/Utils/context-testing'
import { {{pascalCase name}} } from './{{pascalCase name}}'

{{
  #if
    features.includes 'forms'
  import { FormField } from '@components/molecules/FormField'
  {{
    /if
  }}

describe('{{pascalCase name}} Integration Tests', () => {
  {{
    #if
      features.includes 'forms'
    // Form integration
    describe('Form Integration', () => {
      it('integrates with form elements', async () => {
        const user = userEvent.setup()
        const handleSubmit = vi.fn()

        customRender(
          <form onSubmit={handleSubmit}>
            <FormField
              name="test"
              label="Test Field"
              type="text"
              required
            />
            <{{pascalCase name}} type="submit"{{
  #if
    hasChildren
  }}>Submit{{
  /if
}} />
          </form>
        )

        // Test form submission
        await user.click(screen.getByRole('button', { name: /submit/i }))
        expect(handleSubmit).toHaveBeenCalled()
      })
    }
  }}
})

```

```
})  
{{  
/if  
}}
```

```
// Context propagation
```

```
describe('Context Propagation', () => {  
  it('propagates context to child components', () => {  
    customRender(  
      <{{pascalCase name}} context="admin"{{
```

```
#if
```

```
hasChildren}}>
```

```
      <span data-testid="child">Child Element</span>
```

```
      {{
```

```
/if
```

```
}} />,
```

```
    { context: 'consultant' }
```

```
  )
```

```
  const parent = screen.getByTestId('{{kebabCase name}}')
```

```
  const child = screen.getByTestId('child')
```

```
  expect(parent).toHaveAttribute('data-context', 'admin')
```

```
  // Child should inherit context styling
```

```
  expect(child.closest('[data-context="admin"]')).toBeInTheDocument()
```

```
})
```

```
})
```

```
// Theme integration
```

```
describe('Theme Integration', () => {
```

```
  it('applies theme correctly with context', () => {
```

```
    customRender(  
      <{{pascalCase name}}{{
```

```
      <{{pascalCase name}}>{{
```

```
#if
```

```
hasChildren}}>Test{{
```

```
/if
```

```
}} />,
```

```
    { context: 'consultant', theme: 'dark' }
```

```
  )
```

```
  const element = screen.getByTestId('{{kebabCase name}}')
```

```
  expect(element).toHaveClass('dark-theme')
```

```
  expect(element).toHaveClass('consultant-context')
```

```
})
```

```

})

// Component composition
describe('Component Composition', () => {
  it('works with other components', () => {
    customRender(
      <div className="flex gap-2">
        <{{pascalCase name}}{{
#if
  hasChildren}}>Button 1{{
/if
}} />
        <{{pascalCase name}}{{
#if
  hasVariants}} variant="secondary"{{
/if
}} {{
#if
  hasChildren}}>Button 2{{
/if
}} />
      </div>
    )

    const buttons = screen.getAllByRole('button')
    expect(buttons).toHaveLength(2)
    {{
#if
  hasChildren}}
    expect(buttons[0]).toHaveTextContent('Button 1')
    expect(buttons[1]).toHaveTextContent('Button 2')
    {{
/if
  }}
  })
})
})

```

Accessibility Test Template

handlebars

```

{{!-- templates/a11y-test.hbs --}}
import { render, screen } from '@testing-library/react'
import { userEvent } from '@testing-library/user-event'
import { axe, toHaveNoViolations } from 'jest-axe'
import { customRender } from '@test/utils/context-testing'
import { {{pascalCase name}} } from './{{pascalCase name}}'

expect.extend(toHaveNoViolations)

describe('{{pascalCase name}} Accessibility Tests', () => {
  // Automated accessibility testing
  describe('Automated Accessibility', () => {
    it('has no accessibility violations', async () => {
      const { container } = render(<{{pascalCase name}}{{
#if
hasChildren}}>Test{{
/if
}} />)

      const results = await axe(container)
      expect(results).toHaveNoViolations()
    })

    {{
#if
hasVariants}}
    it('maintains accessibility across variants', async () => {
      const variants = [{{
#each
variants}}'{{this}}'{{
#unless
@last}}, {{
/unless
}} {{
/each
}}]

      for (const variant of variants) {
        const { container } = render(
          <{{pascalCase name}} variant={{variant}} {{
#if
hasChildren}}>Test {variant}{{
/if
}} />
        )
      }
    })
  })
})

```



```

    const results = await axe(container)
    expect(results).toHaveNoViolations()
  }
})
{{
/if
}}
```

```

it('maintains accessibility across contexts', async () => {
  const contexts = ['consultant', 'client', 'admin', 'marketplace']

  for (const context of contexts) {
    const { container } = customRender(
      <{{pascalCase name}}{{
/if
hasChildren}}>Test{{
/if
}} />,
      { context }
    )
    const results = await axe(container)
    expect(results).toHaveNoViolations()
  }
})
})
```

```

{{
/if
isInteractive}}
// Keyboard navigation
describe('Keyboard Navigation', () => {
  it('is keyboard accessible', async () => {
    const user = userEvent.setup()
    const handleClick = vi.fn()

    render(<{{pascalCase name}} onClick={handleClick}{{
/if
hasChildren}}>Test{{
/if
}} />)
  })
})
```

```

// Tab to focus
await user.tab()
expect(screen.getByRole('button')).toHaveFocus()
```

```

// Enter to activate
await user.keyboard('[Enter]')
expect(handleClick).toHaveBeenCalledTimes(1)

// Space to activate
await user.keyboard('[Space]')
expect(handleClick).toHaveBeenCalledTimes(2)
})

it('has visible focus indicators', async () => {
  const user = userEvent.setup()

  render(<{{pascalCase name}}{
    #if
    hasChildren}>Test{{
    /if
  }} />)

  await user.tab()
  const button = screen.getByRole('button')

  // Should have focus ring
  expect(button).toHaveFocus()
  expect(button).toHaveClass('focus:ring-2')
})

{{
  #if
  features.includes 'disabled'}}
  it('is not keyboard accessible when disabled', async () => {
    const user = userEvent.setup()
    const handleClick = vi.fn()

    render(<{{pascalCase name}} disabled onClick={handleClick}{{
    #if
    hasChildren}>Test{{
    /if
  }} />)

    // Try to tab to disabled button
    await user.tab()
    expect(screen.getByRole('button')).not.toHaveFocus()
  })
}

```

```

    // Try to activate
    await user.keyboard('[Enter]')
    expect(handleClick).not.toHaveBeenCalled()
  })
  {{
  /if
}}
})
{{
  /if
}}

// Screen reader support
describe('Screen Reader Support', () => {
  it('has proper role', () => {
    render(<{{pascalCase name}}{{
  #if
  hasChildren}}>Test{{
  /if
}} />)
    expect(screen.getByRole('{{
  #if
  isInteractive}}button{{else}}generic{{
  /if
}}')).toBeInTheDocument()
  })

  it('supports aria-label', () => {
    render(<{{pascalCase name}} aria-label="Custom label"{{
  #if
  hasChildren}}>Test{{
  /if
}} />)
    expect(screen.getByRole('{{
  #if
  isInteractive}}button{{else}}generic{{
  /if
}}')).toHaveAttribute('aria-label', 'Custom label')
  })

  it('supports aria-describedby', () => {
    render(
      <div>
        <{{pascalCase name}} aria-describedby="description"{{
  #if

```

```

    hasChildren}}>Test{{
  /if
}} />
    <div id="description">This is a description</div>
  </div>
)
expect(screen.getByRole('{{
  #if
  isInteractive}}button{{else}}generic{{
  /if
}}')).toHaveAttribute('aria-describedby', 'description')
})

{{
  #if
  features.includes 'loading'}}
  it('announces loading state', () => {
    render(<{{pascalCase name}} loading{{
  #if
  hasChildren}}>Loading...{{
  /if
}} />)
    expect(screen.getByRole('{{
  #if
  isInteractive}}button{{else}}generic{{
  /if
}}')).toHaveAttribute('aria-busy', 'true')
  })
  {{
  /if
}}

  {{
  #if
  features.includes 'disabled'}}
  it('announces disabled state', () => {
    render(<{{pascalCase name}} disabled{{
  #if
  hasChildren}}>Disabled{{
  /if
}} />)
    expect(screen.getByRole('{{
  #if
  isInteractive}}button{{else}}generic{{
  /if

```

```

    })).toHaveAttribute('aria-disabled', 'true')
  })
  {{
    /if
  }}
})

// Color contrast
describe('Color Contrast', () => {
  it('meets WCAG AA contrast requirements', async () => {
    const { container } = render(<{{pascalCase name}}{{
    #if
    hasChildren}}>Test{{
    /if
  }} />)

    const results = await axe(container, {
      rules: {
        'color-contrast': { enabled: true }
      }
    })

    expect(results).toHaveNoViolations()
  })

  {{
    #if
    hasVariants}}
    it('maintains contrast across variants', async () => {
      const variants = [{{
      #each
      variants}}'{{this}}'{{
      #unless
      @last}}, {{
      /unless
    }}{{
    /each
  }}]

  for (const variant of variants) {
    const { container } = render(
      <{{pascalCase name}} variant={variant}>{{
    #if

```

```

hasChildren}}>Test {variant}{{
/if
}} />
)

const results = await axe(container, {
  rules: {
    'color-contrast': { enabled: true }
  }
})

expect(results).toHaveNoViolations()
}
})
{{
/if
}}
})

// Touch targets
describe('Touch Targets', () => {
  it('has adequate touch target size', () => {
    render(<{{pascalCase name}}>{{
  #if
  hasChildren}}>Test{{
  /if
}} />)
    const element = screen.getByTestId('{{kebabCase name}}')

    const computedStyle = window.getComputedStyle(element)
    const height = parseInt(computedStyle.height)
    const width = parseInt(computedStyle.width)

    // WCAG recommends 44px minimum
    expect(height).toBeGreaterThanOrEqual(44)
    expect(width).toBeGreaterThanOrEqual(44)
  })
})
})

```

ADDITIONAL TEMPLATES

Index Template

handlebars

```
{{!-- templates/index.hbs --}}
export { {{pascalCase name}} } from './{{pascalCase name}}'
export type {
  {{pascalCase name}}Props{{
    #if
    hasVariants}},
    {{pascalCase name}}Variant{{
    /if
  }}{{
    #if
    features.includes 'sizes'}},
    {{pascalCase name}}Size{{
    /if
  }},
  {{pascalCase name}}Context
} from './types'
export { {{camelCase name}}Styles, {{camelCase name}}Classes } from './styles'
```

README Template

handlebars

```
{{!-- templates/readme.hbs --}}
# {{pascalCase name}}

{{description}}

## Installation

```bash
npm install @wheel/{{category}}
```

## Usage

tsx

```
import { {{pascalCase name}} } from '@wheel/{{category}}'

function App() {
 return (
 <{{pascalCase name}}{{#if hasVariants}} variant="primary"{{/if}}{{#if hasChildren}}>
 Hello World
 </{{pascalCase name}}>
)
}
```

Props

Prop	Type	Default	Description
{{#if hasVariants}}	variant	{{#each variants}}{{this}}{{#unless @last}}   {{/unless}}{{/each}}	'{{variants.[0]}}'
{{#if features.includes 'sizes'}}	size	'sm'   'md'   'lg'	'md'
context	string	Current context	Workspace context
{{#if features.includes 'loading'}}	loading	boolean	false
{{#if features.includes 'disabled'}}	disabled	boolean	false
{{#if hasChildren}}	children	ReactNode	-
className	string	-	Additional CSS classes

Examples

Basic Usage

tsx

```
<{{pascalCase name}}{{#if hasChildren}}>
 Basic {{pascalCase name}}
</{{pascalCase name}}>
```

{{#if hasVariants}}



## Variants

tsx

```
{{#each variants}}
<{{pascalCase ../name}} variant="{{this}}"{{#if ../hasChildren}}>
 {{pascalCase this}} {{pascalCase ../name}}
</{{pascalCase ../name}}>
{{/each}}
```

{{/if}}

{{#if features.includes 'sizes'}}

## Sizes

tsx

```
<{{pascalCase name}} size="sm"{{#if hasChildren}}>Small{{/if}} />
<{{pascalCase name}} size="md"{{#if hasChildren}}>Medium{{/if}} />
<{{pascalCase name}} size="lg"{{#if hasChildren}}>Large{{/if}} />
```

{{/if}}

## Context Awareness

tsx

```
<{{pascalCase name}} context="consultant"{{#if hasChildren}}>
 Consultant Context
</{{pascalCase name}}>

<{{pascalCase name}} context="client"{{#if hasChildren}}>
 Client Context
</{{pascalCase name}}>
```

{{#if features.includes 'loading'}}

## Loading State

tsx

```
<{{pascalCase name}} loading{{#if hasChildren}}>
 Loading...
</{{pascalCase name}}>
```

{{/if}}

{{#if features.includes 'disabled'}}






## Disabled State

tsx

```
<{{pascalCase name}} disabled{{#if hasChildren}}>
 Disabled
</{{pascalCase name}}>
```

{{/if}}

## Accessibility

-  Keyboard navigation support
-  Screen reader compatible
-  WCAG 2.1 AA compliant
-  Focus management
-  High contrast support

## Storybook

View the component in [Storybook](<https://storybook.wheel.dev/?path=/story/{{category}}-{{kebabCase name}}--default>) for interactive examples and documentation.

## Testing

Run tests with:

bash

```
npm test {{pascalCase name}}
```

## Contributing

See [CONTRIBUTING.md](#) for guidelines on contributing to this component.

### ### Context Template

```
```handlebars
{{!-- templates/context.hbs --}}
import React, { createContext, useContext, useEffect, useState } from 'react'
import { contextRegistry } from '../ContextRegistry'

interface {{pascalCase id}}ContextValue {
  // Context-specific values
  features: string[]
  theme: string
  settings: Record<string, any>
  // Add context-specific methods
}

const {{pascalCase id}}Context = createContext<{{pascalCase id}}ContextValue | null>(null)

export const use{{pascalCase id}}Context = () => {
  const context = useContext({{pascalCase id}}Context)
  if (!context) {
    throw new Error('use{{pascalCase id}}Context must be used within a {{pascalCase id}}Provider')
  }
  return context
}

interface {{pascalCase id}}ProviderProps {
  children: React.ReactNode
  settings?: Record<string, any>
}

export const {{pascalCase id}}Provider: React.FC<{{pascalCase id}}ProviderProps> = ({
  children,
  settings = {}
}) => {
  const [contextSettings, setContextSettings] = useState(settings)

  useEffect(() => {
    // Initialize context-specific logic
    console.log('{{pascalCase id}} context initialized')
  }, [])

  const contextValue: {{pascalCase id}}ContextValue = {
    features: [{{#each features}}{{this}}{{#unless @last}}, {{/unless}}{{/each}}],
```

```
theme: '{{kebabCase id}}-light',
settings: contextSettings,
}

return (
  <{{pascalCase id}}Context.Provider value={contextValue}>
    <div className="{{kebabCase id}}-context" data-context="{{id}}">
      {children}
    </div>
  </{{pascalCase id}}Context.Provider>
)
}
```

Theme Template

handlebars

```
{{!-- templates/theme.hbs --}}
```

```
export const {{camelCase id}}Theme = {  
  id: '{{id}}',  
  name: '{{name}}',  
  colors: {  
    primary: '{{color}}',  
    secondary: '#6B7280',  
    accent: '#8B5CF6',  
    background: 'FFFFFF',  
    foreground: '#111827',  
    muted: '#F3F4F6',  
    border: '#E5E7EB',  
    input: 'FFFFFF',  
    ring: '{{color}}',  
    // Status colors  
    success: '#10B981',  
    warning: '#F59E0B',  
    error: '#EF4444',  
    info: '#3B82F6',  
  },  
  typography: {  
    fontFamily: 'Inter, sans-serif',  
    fontSize: {  
      xs: '0.75rem',  
      sm: '0.875rem',  
      base: '1rem',  
      lg: '1.125rem',  
      xl: '1.25rem',  
      '2xl': '1.5rem',  
      '3xl': '1.875rem',  
      '4xl': '2.25rem',  
    },  
    fontWeight: {  
      normal: '400',  
      medium: '500',  
      semibold: '600',  
      bold: '700',  
    },  
    lineHeight: {  
      tight: '1.25',  
      normal: '1.5',  
      relaxed: '1.75',  
    },  
  },  
}
```

```

},
spacing: {
  xs: '0.25rem',
  sm: '0.5rem',
  md: '1rem',
  lg: '1.5rem',
  xl: '2rem',
  '2xl': '3rem',
  '3xl': '4rem',
},
borderRadius: {
  none: '0',
  sm: '0.125rem',
  md: '0.375rem',
  lg: '0.5rem',
  xl: '0.75rem',
  '2xl': '1rem',
  full: '9999px',
},
shadows: {
  sm: '0 1px 2px 0 rgba(0, 0, 0, 0.05)',
  md: '0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0, 0.06)',
  lg: '0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0, 0.05)',
  xl: '0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(0, 0, 0, 0.04)',
},
// Context-specific overrides
components: {
  Button: {
    primary: {
      background: '{{color}}',
      color: '#FFFFFF',
      border: '{{color}}',
      hover: {
        background: '{{color}}dd',
        border: '{{color}}dd',
      },
    },
  },
},
},
}

```

```
export default {{camelCase id}}Theme
```

USAGE EXAMPLES

Generate a New Component

```
bash
```

```
# Generate a new button component
```

```
npx plop component
```

```
# Follow the prompts:
```

```
# - Name: Button
```

```
# - Category: atoms
```

```
# - Description: A versatile button component
```

```
# - Has variants: yes
```

```
# - Variants: primary, secondary, outline, ghost
```

```
# - Has children: yes
```

```
# - Is interactive: yes
```

```
# - Features: loading, disabled, sizes, icons, keyboard
```

```
# Generate a new context
```

```
npx plop context
```

```
# Follow the prompts:
```

```
# - ID: enterprise
```

```
# - Name: Enterprise
```

```
# - Description: Enterprise workspace context
```

```
# - Color: #1F2937
```

```
# - Icon: building
```

```
# - Features: workspace-management, analytics, user-management
```

```
# Add a new story to existing component
```

```
npx plop story
```

```
# Follow the prompts:
```

```
# - Component path: packages/atoms/src/button/Button
```

```
# - Story name: WithCustomIcon
```

```
# - Description: Button with custom icon example
```

Generated File Structure

packages/atoms/src/button/

—— Button.tsx	# Main component
—— Button.stories.tsx	# Storybook stories
—— Button.test.tsx	# Unit tests
—— Button.integration.test.tsx	# Integration tests
—— Button.a11y.test.tsx	# Accessibility tests
—— types.ts	# TypeScript types
—— styles.ts	# Styling configuration
—— index.ts	# Exports
—— README.md	# Documentation

This comprehensive component generation system ensures consistency, quality, and future-proofing across your entire design system while dramatically reducing development time and potential for human error.