Storybook Configuration Guide - Future-Proof Documentation System

🎯 STORYBOOK PHILOSOPHY

Living Documentation: Storybook serves as the single source of truth for component behavior, visual design, and usage patterns across all workspace contexts.

Context-Driven Development: Every story demonstrates how components adapt to different workspace contexts, ensuring consistent behavior across the entire ecosystem.

Future-Proof Architecture: The system automatically adapts to new contexts, themes, and component variants without requiring manual configuration updates.

🏗️ STORYBOOK ARCHITECTURE

Main Configuration

typescript

```ts
// .storybook/main.ts
import type { StorybookConfig } from '@storybook/nextjs'
import { join, dirname } from 'path'

const config: StorybookConfig = {
  stories: [
    '../src/**/*.stories.@(js|jsx|ts|tsx|mdx)',
    '../packages/ui/src/**/*.stories.@(js|jsx|ts|tsx)',
    '../packages/patterns/src/**/*.stories.@(js|jsx|ts|tsx)',
    '../packages/workspace/src/**/*.stories.@(js|jsx|ts|tsx)',
  ],

  addons: [
    getAbsolutePath('@storybook/addon-essentials'),
    getAbsolutePath('@storybook/addon-interactions'),
    getAbsolutePath('@storybook/addon-a11y'),
    getAbsolutePath('@storybook/addon-design-tokens'),
    getAbsolutePath('@storybook/addon-docs'),
    getAbsolutePath('@storybook/addon-controls'),
    getAbsolutePath('@storybook/addon-viewport'),
    getAbsolutePath('@storybook/addon-backgrounds'),
    getAbsolutePath('@storybook/addon-measure'),
    getAbsolutePath('@storybook/addon-outline'),
    getAbsolutePath('@storybook/addon-performance'),
    getAbsolutePath('@storybook/addon-coverage'),
  ],

  framework: {
    name: getAbsolutePath('@storybook/nextjs'),
    options: {},
  },

  typescript: {
    check: false,
    checkOptions: {},
    reactDocgen: 'react-docgen-typescript',
    reactDocgenTypescriptOptions: {
      shouldExtractLiteralValuesFromEnum: true,
      propFilter: (prop) => (prop.parent ? !/node_modules/.test(prop.parent.fileName) : true),
      compilerOptions: {
        allowSyntheticDefaultImports: false,
        esModuleInterop: false,
      },
```

```
  },
 },

 docs: {
  autodocs: 'tag',
  defaultName: 'Documentation',
 },

 features: {
  experimentalRSC: true,
  buildStoriesJson: true,
 },

 staticDirs: ['../public'],

 webpackFinal: async (config) => {
  // Custom webpack configuration
  config.resolve.alias = {
   ...config.resolve.alias,
   '@': join(__dirname, '../src'),
   '@/components': join(__dirname, '../src/components'),
   '@/contexts': join(__dirname, '../src/contexts'),
   '@/utils': join(__dirname, '../src/utils'),
  }

  return config
 },
}

function getAbsolutePath(value: string): any {
 return dirname(require.resolve(join(value, 'package.json')))
}

export default config
```

🎨 FUTURE-PROOF CONTEXT SYSTEM

Dynamic Context Registry

typescript

```typescript
// src/contexts/ContextRegistry.ts
export interface ContextDefinition {
  id: string
  name: string
  description: string
  theme: string
  icon?: string
  color?: string
  parentContext?: string
  features?: string[]
  deprecated?: boolean
}

class ContextRegistry {
  private contexts: Map<string, ContextDefinition> = new Map()
  private listeners: Set<(contexts: ContextDefinition[]) => void> = new Set()

  register(context: ContextDefinition): void {
    this.contexts.set(context.id, context)
    this.notifyListeners()
  }

  unregister(contextId: string): void {
    this.contexts.delete(contextId)
    this.notifyListeners()
  }

  getContext(contextId: string): ContextDefinition | undefined {
    return this.contexts.get(contextId)
  }

  getAllContexts(): ContextDefinition[] {
    return Array.from(this.contexts.values())
      .filter(context => !context.deprecated)
      .sort((a, b) => a.name.localeCompare(b.name))
  }

  getContextHierarchy(contextId: string): ContextDefinition[] {
    const hierarchy: ContextDefinition[] = []
    let current = this.getContext(contextId)

    while (current) {
      hierarchy.unshift(current)
```

```
      current = current.parentContext ? this.getContext(current.parentContext) : undefined
    }

    return hierarchy
  }

  subscribe(listener: (contexts: ContextDefinition[]) => void): () => void {
    this.listeners.add(listener)
    return () => this.listeners.delete(listener)
  }

  private notifyListeners(): void {
    const contexts = this.getAllContexts()
    this.listeners.forEach(listener => listener(contexts))
  }
}

export const contextRegistry = new ContextRegistry()

// Register core contexts
contextRegistry.register({
  id: 'consultant',
  name: 'Consultant',
  description: 'Primary business context for consultants',
  theme: 'consultant-light',
  icon: 'briefcase',
  color: '#3B82F6',
  features: ['workspace-management', 'client-management', 'billing']
})

contextRegistry.register({
  id: 'client',
  name: 'Client Portal',
  description: 'Client-facing portal context',
  theme: 'client-light',
  icon: 'user',
  color: '#10B981',
  features: ['project-tracking', 'document-access', 'communications']
})

contextRegistry.register({
  id: 'admin',
  name: 'Administrator',
  description: 'Administrative context',
```

```
  theme: 'admin-light',
  icon: 'settings',
  color: '#6B7280',
  features: ['user-management', 'system-settings', 'analytics']
})

contextRegistry.register({
  id: 'marketplace',
  name: 'Marketplace',
  description: 'Multi-sided marketplace context',
  theme: 'marketplace-light',
  icon: 'store',
  color: '#8B5CF6',
  features: ['service-browsing', 'expert-matching', 'transactions']
})

// Future contexts can be registered dynamically
export const registerContext = (context: ContextDefinition) => {
  contextRegistry.register(context)
}

export const getRegisteredContexts = () => {
  return contextRegistry.getAllContexts()
}
```

Context Provider System

typescript

```tsx
// src/contexts/WorkspaceContext.tsx
import { createContext, useContext, useEffect, useState } from 'react'
import { contextRegistry, ContextDefinition } from './ContextRegistry'

interface WorkspaceContextValue {
  currentContext: string
  contextDefinition: ContextDefinition | null
  availableContexts: ContextDefinition[]
  switchContext: (contextId: string) => void
  isContextAvailable: (contextId: string) => boolean
  getContextHierarchy: (contextId: string) => ContextDefinition[]
}

const WorkspaceContext = createContext<WorkspaceContextValue | null>(null)

export const useWorkspaceContext = () => {
  const context = useContext(WorkspaceContext)
  if (!context) {
    throw new Error('useWorkspaceContext must be used within a WorkspaceProvider')
  }
  return context
}

interface WorkspaceProviderProps {
  children: React.ReactNode
  initialContext?: string
  availableContexts?: string[]
}

export const WorkspaceProvider: React.FC<WorkspaceProviderProps> = ({
  children,
  initialContext = 'consultant',
  availableContexts
}) => {
  const [currentContext, setCurrentContext] = useState(initialContext)
  const [allContexts, setAllContexts] = useState<ContextDefinition[]>([])

  useEffect(() => {
    const updateContexts = (contexts: ContextDefinition[]) => {
      setAllContexts(contexts)
    }

    updateContexts(contextRegistry.getAllContexts())
```

```
    return contextRegistry.subscribe(updateContexts)
  }, [])

  const contextDefinition = contextRegistry.getContext(currentContext)
  const availableContextDefinitions = availableContexts
    ? allContexts.filter(ctx => availableContexts.includes(ctx.id))
    : allContexts

  const switchContext = (contextId: string) => {
    if (contextRegistry.getContext(contextId)) {
      setCurrentContext(contextId)
    }
  }

  const isContextAvailable = (contextId: string) => {
    return availableContextDefinitions.some(ctx => ctx.id === contextId)
  }

  const getContextHierarchy = (contextId: string) => {
    return contextRegistry.getContextHierarchy(contextId)
  }

  return (
    <WorkspaceContext.Provider
      value={{
        currentContext,
        contextDefinition,
        availableContexts: availableContextDefinitions,
        switchContext,
        isContextAvailable,
        getContextHierarchy
      }}
    >
      <div
        className={`workspace-context workspace-${currentContext}`}
        data-context={currentContext}
        data-theme={contextDefinition?.theme}
      >
        {children}
      </div>
    </WorkspaceContext.Provider>
  )
}
```

🎛️ DYNAMIC STORYBOOK CONFIGURATION

Preview Configuration with Dynamic Contexts

typescript

```tsx
// .storybook/preview.ts
import type { Preview } from '@storybook/react'
import { useEffect } from 'react'
import { WorkspaceProvider } from '../src/contexts/WorkspaceContext'
import { ThemeProvider } from '../src/contexts/ThemeContext'
import { contextRegistry, getRegisteredContexts } from '../src/contexts/ContextRegistry'
import '../src/styles/globals.css'

// Dynamic context decorator
const withWorkspaceContext = (Story, context) => {
  const { globals } = context

  return (
    <WorkspaceProvider initialContext={globals.workspaceContext || 'consultant'}>
      <ThemeProvider theme={globals.theme || 'light'}>
        <div className="story-container">
          <Story />
        </div>
      </ThemeProvider>
    </WorkspaceProvider>
  )
}

// Performance monitoring decorator
const withPerformanceMonitoring = (Story, context) => {
  useEffect(() => {
    const startTime = performance.now()

    return () => {
      const endTime = performance.now()
      const renderTime = endTime - startTime

      if (renderTime > 16) {
        console.warn(`Story "${context.title}" render time: ${renderTime.toFixed(2)}ms`)
      }
    }
  }, [context.title])

  return <Story />
}

// Generate dynamic global types
const generateGlobalTypes = () => {
```

```
const contexts = getRegisteredContexts()

return {
  workspaceContext: {
    name: 'Workspace Context',
    description: 'Current workspace context',
    defaultValue: 'consultant',
    toolbar: {
      icon: 'users',
      items: contexts.map(ctx => ({
        value: ctx.id,
        title: ctx.name,
        icon: ctx.icon,
        right: ctx.color ? `🟢` : undefined
      })),
      dynamicTitle: true
    }
  },
  theme: {
    name: 'Theme',
    description: 'Visual theme',
    defaultValue: 'light',
    toolbar: {
      icon: 'paintbrush',
      items: [
        { value: 'light', title: 'Light', icon: 'sun' },
        { value: 'dark', title: 'Dark', icon: 'moon' },
        { value: 'auto', title: 'Auto', icon: 'contrast' }
      ]
    }
  },
  userRole: {
    name: 'User Role',
    description: 'Current user role',
    defaultValue: 'user',
    toolbar: {
      icon: 'user',
      items: [
        { value: 'user', title: 'User' },
        { value: 'admin', title: 'Administrator' },
        { value: 'owner', title: 'Owner' },
        { value: 'viewer', title: 'Viewer' }
      ]
    }
```

```
    },
    locale: {
      name: 'Locale',
      description: 'Internationalization locale',
      defaultValue: 'en',
      toolbar: {
        icon: 'globe',
        items: [
          { value: 'en', title: 'English' },
          { value: 'es', title: 'Español' },
          { value: 'fr', title: 'Français' },
          { value: 'de', title: 'Deutsch' }
        ]
      }
    }
  }
}

const preview: Preview = {
  decorators: [withWorkspaceContext, withPerformanceMonitoring],

  parameters: {
    actions: { argTypesRegex: '^on[A-Z].*' },
    controls: {
      matchers: {
        color: /(background|color)$/i,
        date: /Date$/,
      },
      sort: 'alpha',
    },
    docs: {
      toc: {
        contentsSelector: '.sbdocs-content',
        headingSelector: 'h1, h2, h3',
        ignoreSelector: '#primary',
        title: 'Table of Contents',
      },
      source: {
        state: 'open',
        type: 'dynamic',
      },
    },
    viewport: {
      viewports: {
```

```
    mobile: {
      name: 'Mobile',
      styles: { width: '375px', height: '667px' },
    },
    tablet: {
      name: 'Tablet',
      styles: { width: '768px', height: '1024px' },
    },
    desktop: {
      name: 'Desktop',
      styles: { width: '1200px', height: '800px' },
    },
    wide: {
      name: 'Wide',
      styles: { width: '1440px', height: '900px' },
    },
  },
},
backgrounds: {
  default: 'light',
  values: [
    { name: 'light', value: '#ffffff' },
    { name: 'dark', value: '#1a1a1a' },
    { name: 'consultant', value: '#f8fafc' },
    { name: 'client', value: '#f0fdf4' },
    { name: 'admin', value: '#f9fafb' },
    { name: 'marketplace', value: '#faf5ff' },
  ],
},
options: {
  storySort: {
    order: [
      'Introduction',
      'Design System',
      ['Tokens', 'Principles', 'Guidelines'],
      'Atoms',
      'Molecules',
      'Organisms',
      'Templates',
      'Examples',
      '*',
    ],
  },
},
```

```
  },

  // Dynamic global types
  globalTypes: generateGlobalTypes(),

  // Initialize context registry
  loaders: [
    async () => {
      // Load any dynamic contexts from API or config
      const dynamicContexts = await loadDynamicContexts()
      dynamicContexts.forEach(ctx => contextRegistry.register(ctx))

      return {}
    }
  ]
}

// Load dynamic contexts (could be from API, config file, etc.)
const loadDynamicContexts = async () => {
  // This could load from an API, config file, or environment variables
  const dynamicContexts = []

  // Example: Load from environment
  if (process.env.STORYBOOK_ENABLE_BETA_CONTEXTS) {
    dynamicContexts.push({
      id: 'beta-tester',
      name: 'Beta Tester',
      description: 'Beta testing context',
      theme: 'beta-light',
      icon: 'beaker',
      color: '#F59E0B',
      features: ['beta-features', 'feedback-tools']
    })
  }

  return dynamicContexts
}

export default preview
```

📖 STORY PATTERNS & TEMPLATES

Universal Story Template

typescript

```ts
// src/story-templates/UniversalStoryTemplate.ts
import type { Meta, StoryObj } from '@storybook/react'
import { getRegisteredContexts } from '../contexts/ContextRegistry'

export interface UniversalStoryConfig<T> {
  title: string
  component: React.ComponentType<T>
  category: 'atoms' | 'molecules' | 'organisms' | 'templates'
  description?: string
  variants?: Array<{
    name: string
    props: Partial<T>
    description?: string
  }>
  contexts?: string[]
  accessibility?: {
    label?: string
    description?: string
  }
  performance?: {
    expectedRenderTime?: number
    bundleSize?: number
  }
}

export const createUniversalStory = <T extends Record<string, any>>(
  config: UniversalStoryConfig<T>
): Meta<T> => {
  const contexts = config.contexts || getRegisteredContexts().map(ctx => ctx.id)

  return {
    title: `${config.category}/${config.title}`,
    component: config.component,
    parameters: {
      layout: 'centered',
      docs: {
        description: {
          component: config.description || `${config.title} component with workspace context awareness.`,
        },
      },
      accessibility: config.accessibility,
      performance: config.performance,
    },
```

```
    tags: ['autodocs'],
    argTypes: {
      // Auto-generate context argTypes
      ...(contexts.length > 1 && {
        context: {
          control: 'select',
          options: contexts,
          description: 'Workspace context for component theming and behavior',
        },
      }),
      // Auto-generate variant argTypes if provided
      ...(config.variants && {
        variant: {
          control: 'select',
          options: config.variants.map(v => v.name),
          description: 'Component variant',
        },
      }),
    },
  }
}

// Generate standard stories
export const createStandardStories = <T extends Record<string, any>>(
  config: UniversalStoryConfig<T>
): Record<string, StoryObj<T>> => {
  const stories: Record<string, StoryObj<T>> = {}
  const contexts = config.contexts || getRegisteredContexts().map(ctx => ctx.id)

  // Default story
  stories.Default = {
    args: {} as T,
  }

  // Variant stories
  if (config.variants) {
    config.variants.forEach(variant => {
      stories[variant.name] = {
        args: variant.props as T,
        parameters: {
          docs: {
            description: {
              story: variant.description || `${variant.name} variant of ${config.title}`,
            },
```

```
      },
    },
  }
 })
}

// Context showcase story
if (contexts.length > 1) {
 stories.AllContexts = {
  render: (args) => {
   const contexts = getRegisteredContexts()

   return (
    <div className="grid grid-cols-2 gap-4">
     {contexts.map(ctx => (
      <div key={ctx.id} className="text-center">
       <div className="mb-2">
        <config.component {...args} context={ctx.id} />
       </div>
       <p className="text-sm text-gray-600">{ctx.name}</p>
      </div>
     ))}
    </div>
   )
  },
  parameters: {
   docs: {
    description: {
     story: `${config.title} component across all workspace contexts.`,
    },
   },
  },
 }
}

// Interactive states story
stories.InteractiveStates = {
 render: (args) => (
  <div className="flex gap-4 items-center">
   <config.component {...args} />
   <config.component {...args} disabled />
   <config.component {...args} loading />
  </div>
 ),
```

```
    parameters: {
      docs: {
        description: {
          story: `Interactive states of ${config.title} component.`,
        },
      },
    },
  }

  return stories
}
```

Example Usage of Universal Template

typescript

```tsx
// Button.stories.tsx
import type { Meta, StoryObj } from '@storybook/react'
import { Button } from './Button'
import { createUniversalStory, createStandardStories } from '@/story-templates/UniversalStoryTemplate'

const config = {
  title: 'Button',
  component: Button,
  category: 'atoms' as const,
  description: 'A versatile button component with workspace context awareness and multiple variants.',
  variants: [
    {
      name: 'Primary',
      props: { variant: 'primary', children: 'Primary Button' },
      description: 'Primary action button for main user actions.'
    },
    {
      name: 'Secondary',
      props: { variant: 'secondary', children: 'Secondary Button' },
      description: 'Secondary action button for supporting actions.'
    },
    {
      name: 'Outline',
      props: { variant: 'outline', children: 'Outline Button' },
      description: 'Outline button for subtle actions.'
    },
    {
      name: 'Ghost',
      props: { variant: 'ghost', children: 'Ghost Button' },
      description: 'Ghost button for minimal actions.'
    }
  ],
  accessibility: {
    label: 'Button Component',
    description: 'Fully accessible button with ARIA support and keyboard navigation.'
  },
  performance: {
    expectedRenderTime: 5,
    bundleSize: 2048
  }
}

const meta: Meta<typeof Button> = createUniversalStory(config)
```

```tsx
export default meta

type Story = StoryObj<typeof meta>

// Generate standard stories
const standardStories = createStandardStories(config)
export const { Default, Primary, Secondary, Outline, Ghost, AllContexts, InteractiveStates } = standardStories

// Custom stories
export const WithIcons: Story = {
  render: () => (
    <div className="flex gap-4">
      <Button>
        <Icon name="plus" className="w-4 h-4 mr-2" />
        Add Item
      </Button>
      <Button variant="secondary">
        <Icon name="download" className="w-4 h-4 mr-2" />
        Download
      </Button>
      <Button variant="outline">
        <Icon name="share" className="w-4 h-4 mr-2" />
        Share
      </Button>
    </div>
  ),
  parameters: {
    docs: {
      description: {
        story: 'Buttons with icons for enhanced visual communication.',
      },
    },
  },
}

export const LoadingStates: Story = {
  render: () => (
    <div className="flex gap-4">
      <Button loading>Loading...</Button>
      <Button variant="secondary" loading>
        Processing
      </Button>
      <Button variant="outline" loading>
        Saving
```

```
      </Button>
    </div>
  ),
  parameters: {
    docs: {
      description: {
        story: 'Loading states for async operations.',
      },
    },
  },
}

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">
              <Button context={ctx.id} variant="primary">
                Primary Action
              </Button>
              <Button context={ctx.id} variant="secondary">
                Secondary Action
              </Button>
            </div>
          </div>
        ))}
      </div>
    )
  },
  parameters: {
    docs: {
      description: {
        story: 'Contextual usage examples showing how buttons adapt to different workspace contexts.',
      },
    },
  },
}
```

🎨 DESIGN TOKENS INTEGRATION

Token Documentation Stories

typescript

```tsx
// design-tokens/Colors.stories.tsx
import type { Meta, StoryObj } from '@storybook/react'
import { getRegisteredContexts } from '../src/contexts/ContextRegistry'

const meta: Meta = {
  title: 'Design System/Tokens/Colors',
  parameters: {
    docs: {
      description: {
        component: 'Color tokens used across the design system, organized by workspace context.',
      },
    },
  },
}

export default meta

export const ContextColors: StoryObj = {
  render: () => {
    const contexts = getRegisteredContexts()

    return (
      <div className="space-y-8">
        {contexts.map(ctx => (
          <div key={ctx.id}>
            <h3 className="text-lg font-semibold mb-4">{ctx.name} Context</h3>
            <div className="grid grid-cols-6 gap-4">
              {/* Primary Colors */}
              <div className="text-center">
                <div
                  className="w-16 h-16 rounded-lg mb-2 mx-auto shadow-sm"
                  style={{ backgroundColor: `var(--${ctx.id}-primary)` }}
                />
                <p className="text-sm font-medium">Primary</p>
                <code className="text-xs text-gray-500">--{ctx.id}-primary</code>
              </div>

              {/* Secondary Colors */}
              <div className="text-center">
                <div
                  className="w-16 h-16 rounded-lg mb-2 mx-auto shadow-sm"
                  style={{ backgroundColor: `var(--${ctx.id}-secondary)` }}
                />
```

```
              <p className="text-sm font-medium">Secondary</p>
              <code className="text-xs text-gray-500">--{ctx.id}-secondary</code>
            </div>

            {/* Add more color tokens as needed */}
          </div>
        </div>
      ))}
    </div>
  )
 },
 parameters: {
  docs: {
   description: {
    story: 'Color tokens organized by workspace context, showing how the design system adapts to different use
   },
  },
 },
}

export const SemanticColors: StoryObj = {
 render: () => (
  <div className="space-y-6">
   <div>
    <h3 className="text-lg font-semibold mb-4">Status Colors</h3>
    <div className="grid grid-cols-4 gap-4">
     {['success', 'warning', 'error', 'info'].map(status => (
      <div key={status} className="text-center">
       <div
        className="w-16 h-16 rounded-lg mb-2 mx-auto shadow-sm"
        style={{ backgroundColor: `var(--color-${status})` }}
       />
       <p className="text-sm font-medium capitalize">{status}</p>
       <code className="text-xs text-gray-500">--color-{status}</code>
      </div>
     ))}
    </div>
   </div>

   <div>
    <h3 className="text-lg font-semibold mb-4">Neutral Colors</h3>
    <div className="grid grid-cols-6 gap-4">
     {[50, 100, 200, 300, 400, 500, 600, 700, 800, 900].map(shade => (
      <div key={shade} className="text-center">
```

```
            <div
              className="w-16 h-16 rounded-lg mb-2 mx-auto shadow-sm"
              style={{ backgroundColor: `var(--color-gray-${shade})` }}
            />
            <p className="text-sm font-medium">{shade}</p>
            <code className="text-xs text-gray-500">--color-gray-{shade}</code>
          </div>
        ))}
      </div>
    </div>
  </div>
  ),
}
```

🔧 ADVANCED STORYBOOK FEATURES

Interactive Stories with Controls

typescript

```tsx
// InteractiveStory.stories.tsx
import type { Meta, StoryObj } from '@storybook/react'
import { useState } from 'react'
import { Button } from './Button'

const meta: Meta<typeof Button> = {
  title: 'Examples/Interactive Button',
  component: Button,
  argTypes: {
    variant: {
      control: 'select',
      options: ['primary', 'secondary', 'outline', 'ghost'],
    },
    size: {
      control: 'select',
      options: ['sm', 'md', 'lg'],
    },
    context: {
      control: 'select',
      options: getRegisteredContexts().map(ctx => ctx.id),
    },
    disabled: {
      control: 'boolean',
    },
    loading: {
      control: 'boolean',
    },
    children: {
      control: 'text',
    },
  },
}

export default meta
type Story = StoryObj<typeof meta>

export const Interactive: Story = {
  args: {
    variant: 'primary',
    size: 'md',
    context: 'consultant',
    disabled: false,
    loading: false,
```

```
    children: 'Interactive Button',
  },
  render: (args) => {
    const [clickCount, setClickCount] = useState(0)

    return (
      <div className="space-y-4">
        <Button
          {...args}
          onClick={() => setClickCount(prev => prev + 1)}
        >
          {args.children} (clicked {clickCount} times)
        </Button>

        <div className="text-sm text-gray-600">
          <p>Try changing the controls above to see how the button responds!</p>
          <p>Current state: {JSON.stringify(args, null, 2)}</p>
        </div>
      </div>
    )
  },
}
```

Performance Monitoring Stories

typescript

```tsx
// PerformanceStory.stories.tsx
import type { Meta, StoryObj } from '@storybook/react'
import { useEffect, useState } from 'react'
import { Button } from './Button'

const meta: Meta<typeof Button> = {
  title: 'Examples/Performance Testing',
  component: Button,
}

export default meta
type Story = StoryObj<typeof meta>

export const RenderPerformance: Story = {
  render: () => {
    const [renderTime, setRenderTime] = useState<number | null>(null)
    const [reRenderCount, setReRenderCount] = useState(0)

    useEffect(() => {
      const startTime = performance.now()

      const timeout = setTimeout(() => {
        const endTime = performance.now()
        setRenderTime(endTime - startTime)
      }, 0)

      return () => clearTimeout(timeout)
    }, [reRenderCount])

    return (
      <div className="space-y-4">
        <div className="p-4 bg-gray-50 rounded-lg">
          <h3 className="font-semibold mb-2">Performance Metrics</h3>
          <p>Render time: {renderTime ? `${renderTime.toFixed(2)}ms` : 'Measuring...'}</p>
          <p>Re-renders: {reRenderCount}</p>
          <p>Target: &lt;16ms (60fps)</p>
        </div>

        <Button onClick={() => setReRenderCount(prev => prev + 1)}>
          Trigger Re-render
        </Button>

        <div className="grid grid-cols-4 gap-2">
```

```
          {Array.from({ length: 100 }, (_, i) => (
            <Button key={i} size="sm" variant="outline">
              Button {i + 1}
            </Button>
          ))}
        </div>
      </div>
    )
  },
}
```

Accessibility Testing Stories

typescript

```tsx
// AccessibilityStory.stories.tsx
import type { Meta, StoryObj } from '@storybook/react'
import { useState } from 'react'
import { Button } from './Button'

const meta: Meta<typeof Button> = {
  title: 'Examples/Accessibility Testing',
  component: Button,
  parameters: {
    a11y: {
      config: {
        rules: [
          {
            id: 'color-contrast',
            enabled: true,
          },
          {
            id: 'keyboard-navigation',
            enabled: true,
          },
          {
            id: 'focus-management',
            enabled: true,
          },
        ],
      },
    },
  },
}

export default meta
type Story = StoryObj<typeof meta>

export const KeyboardNavigation: Story = {
  render: () => {
    const [focusedButton, setFocusedButton] = useState<string | null>(null)

    return (
      <div className="space-y-4">
        <div className="p-4 bg-blue-50 rounded-lg">
          <h3 className="font-semibold mb-2">Keyboard Navigation Test</h3>
          <p className="text-sm">Use Tab/Shift+Tab to navigate, Enter/Space to activate</p>
          <p className="text-sm">Currently focused: {focusedButton || 'None'}</p>
```

```
      </div>

      <div className="flex gap-4">
        <Button
          onFocus={() => setFocusedButton('Button 1')}
          onBlur={() => setFocusedButton(null)}
        >
          Button 1
        </Button>
        <Button
          variant="secondary"
          onFocus={() => setFocusedButton('Button 2')}
          onBlur={() => setFocusedButton(null)}
        >
          Button 2
        </Button>
        <Button
          variant="outline"
          onFocus={() => setFocusedButton('Button 3')}
          onBlur={() => setFocusedButton(null)}
        >
          Button 3
        </Button>
      </div>
    </div>
  )
 },
}

export const ScreenReaderSupport: Story = {
 render: () => (
  <div className="space-y-4">
    <div className="p-4 bg-green-50 rounded-lg">
      <h3 className="font-semibold mb-2">Screen Reader Support</h3>
      <p className="text-sm">These buttons have proper ARIA labels and descriptions</p>
    </div>

    <div className="flex gap-4">
      <Button
        aria-label="Save your changes"
        aria-describedby="save-description"
      >
        Save
      </Button>
```

```
      <div id="save-description" className="sr-only">
        Saves the current document with all your changes
      </div>

      <Button
        variant="secondary"
        aria-label="Cancel current operation"
        aria-describedby="cancel-description"
      >
        Cancel
      </Button>
      <div id="cancel-description" className="sr-only">
        Cancels the current operation and returns to the previous state
      </div>
    </div>
  </div>
 ),
}
```

📱 RESPONSIVE DESIGN STORIES

Responsive Showcase

typescript

```tsx
// ResponsiveStory.stories.tsx
import type { Meta, StoryObj } from '@storybook/react'
import { Button } from './Button'

const meta: Meta<typeof Button> = {
  title: 'Examples/Responsive Design',
  component: Button,
  parameters: {
    viewport: {
      viewports: {
        mobile: { name: 'Mobile', styles: { width: '375px', height: '667px' } },
        tablet: { name: 'Tablet', styles: { width: '768px', height: '1024px' } },
        desktop: { name: 'Desktop', styles: { width: '1200px', height: '800px' } },
      },
    },
  },
}

export default meta
type Story = StoryObj<typeof meta>

export const ResponsiveLayout: Story = {
  render: () => (
    <div className="space-y-6">
      <div className="p-4 bg-gray-50 rounded-lg">
        <h3 className="font-semibold mb-2">Responsive Button Layout</h3>
        <p className="text-sm">Buttons adapt to different screen sizes</p>
      </div>

      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
        <Button className="w-full sm:w-auto">
          Responsive Button 1
        </Button>
        <Button variant="secondary" className="w-full sm:w-auto">
          Responsive Button 2
        </Button>
        <Button variant="outline" className="w-full sm:w-auto">
          Responsive Button 3
        </Button>
      </div>

      <div className="flex flex-col sm:flex-row gap-4">
        <Button className="flex-1">
```

```
          Flexible Button 1
        </Button>
        <Button variant="secondary" className="flex-1">
          Flexible Button 2
        </Button>
      </div>
    </div>
  ),
  parameters: {
    docs: {
      description: {
        story: 'Demonstrates how buttons adapt to different screen sizes and layouts.',
      },
    },
  },
}
```

🎯 FUTURE-PROOF STORY PATTERNS

Dynamic Story Generation

typescript

```ts
// DynamicStoryGenerator.ts
import type { Meta, StoryObj } from '@storybook/react'
import { getRegisteredContexts } from '../src/contexts/ContextRegistry'

export const generateContextStories = <T extends { context?: string }>(
  component: React.ComponentType<T>,
  baseArgs: T
) => {
  const contexts = getRegisteredContexts()
  const stories: Record<string, StoryObj<T>> = {}

  contexts.forEach(ctx => {
    stories[`${ctx.name}Context`] = {
      args: {
        ...baseArgs,
        context: ctx.id,
      } as T,
      parameters: {
        docs: {
          description: {
            story: `Component in ${ctx.name} context (${ctx.description})`,
          },
        },
      },
    }
  })

  return stories
}

export const generateVariantStories = <T extends { variant?: string }>(
  component: React.ComponentType<T>,
  baseArgs: T,
  variants: string[]
) => {
  const stories: Record<string, StoryObj<T>> = {}

  variants.forEach(variant => {
    stories[`${variant}Variant`] = {
      args: {
        ...baseArgs,
        variant,
      } as T,
```

```
      parameters: {
        docs: {
          description: {
            story: `${variant} variant of the component`,
          },
        },
      },
    }
  })

  return stories
}
```

Auto-Generated Documentation

typescript

```tsx
// AutoDocGenerator.ts
import { getRegisteredContexts } from '../src/contexts/ContextRegistry'

export const generateContextDocumentation = () => {
  const contexts = getRegisteredContexts()

  return {
    title: 'Design System/Contexts',
    parameters: {
      docs: {
        page: () => (
          <div className="space-y-8">
            <div>
              <h1>Workspace Contexts</h1>
              <p>
                The design system supports multiple workspace contexts, each with its own
                theming, behavior, and feature set. Components automatically adapt to the
                current context.
              </p>
            </div>

            {contexts.map(ctx => (
              <div key={ctx.id} className="border rounded-lg p-6">
                <h2 className="text-xl font-semibold mb-4">{ctx.name}</h2>
                <p className="text-gray-600 mb-4">{ctx.description}</p>

                <div className="grid grid-cols-2 gap-4">
                  <div>
                    <h3 className="font-medium mb-2">Theme</h3>
                    <p className="text-sm text-gray-500">{ctx.theme}</p>
                  </div>

                  <div>
                    <h3 className="font-medium mb-2">Features</h3>
                    <ul className="text-sm text-gray-500">
                      {ctx.features?.map(feature => (
                        <li key={feature}>• {feature}</li>
                      ))}
                    </ul>
                  </div>
                </div>
              </div>
            ))}
```

```
          </div>
        ),
      },
    },
  }
}
```

This comprehensive Storybook configuration provides a future-proof foundation that automatically adapts to new contexts, themes, and components while maintaining comprehensive documentation and testing capabilities.