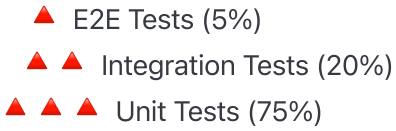
# Testing Strategy Guide - Comprehensive Component Testing

## 🎯 TESTING PHILOSOPHY

**Test-Driven Component Development**: Every component should be tested at multiple levels to ensure reliability, accessibility, and performance across all workspace contexts.

**Testing Pyramid for Components**:




---

## 🧪 TESTING FRAMEWORKS & TOOLS

### Core Testing Stack

```json
{
  "dependencies": {
    "@testing-library/react": "^14.0.0",
    "@testing-library/jest-dom": "^6.0.0",
    "@testing-library/user-event": "^14.0.0",
    "@storybook/testing-react": "^2.0.0",
    "vitest": "^1.0.0",
    "jsdom": "^23.0.0"
  },
  "devDependencies": {
    "@axe-core/react": "^4.8.0",
    "axe-playwright": "^1.2.0",
    "chromatic": "^7.0.0",
    "@storybook/test-runner": "^0.15.0",
    "playwright": "^1.40.0"
  }
}
```

### Test Configuration

typescript

```typescript
// vitest.config.ts
import { defineConfig } from 'vitest/config'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  test: {
    environment: 'jsdom',
    setupFiles: ['./src/test/setup.ts'],
    globals: true,
    css: true,
    coverage: {
      reporter: ['text', 'json', 'html'],
      threshold: {
        global: {
          branches: 90,
          functions: 90,
          lines: 90,
          statements: 90
        }
      }
    }
  }
})
```

---

## 📝 UNIT TESTING PATTERNS

### Basic Component Test Structure

typescript

```tsx
// Button.test.tsx
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 * as stories from './Button.stories'

// Extend Jest matchers
expect.extend(toHaveNoViolations)

// Compose stories for testing
const {
  Default,
  AllVariants,
  WorkspaceContexts,
  InteractiveStates
} = composeStories(stories)

// Test wrapper with all contexts
const TestWrapper = ({ children, context = 'consultant' }) => (
  <WorkspaceProvider context={context}>
    <ThemeProvider>
      {children}
    </ThemeProvider>
  </WorkspaceProvider>
)

describe('Button Component', () => {
  // Basic rendering tests
  describe('Rendering', () => {
    it('renders with default props', () => {
      render(<Default />)
      expect(screen.getByRole('button')).toBeInTheDocument()
    })

    it('renders with all variants', () => {
      render(<AllVariants />)
      expect(screen.getByText('Primary')).toBeInTheDocument()
      expect(screen.getByText('Secondary')).toBeInTheDocument()
      expect(screen.getByText('Outline')).toBeInTheDocument()
    })

    it('renders with all workspace contexts', () => {
```

```
    render(<WorkspaceContexts />)
    expect(screen.getByText('Consultant')).toBeInTheDocument()
    expect(screen.getByText('Client')).toBeInTheDocument()
    expect(screen.getByText('Admin')).toBeInTheDocument()
    expect(screen.getByText('Marketplace')).toBeInTheDocument()
  })
})

// Context-specific tests
describe('Workspace Context', () => {
  it('applies consultant context styling', () => {
    render(
      <TestWrapper context="consultant">
        <Button variant="primary">Test</Button>
      </TestWrapper>
    )
    const button = screen.getByRole('button')
    expect(button).toHaveClass('consultant-primary')
    expect(button).toHaveAttribute('data-context', 'consultant')
  })

  it('applies client context styling', () => {
    render(
      <TestWrapper context="client">
        <Button variant="primary">Test</Button>
      </TestWrapper>
    )
    const button = screen.getByRole('button')
    expect(button).toHaveClass('client-primary')
    expect(button).toHaveAttribute('data-context', 'client')
  })

  // Future-proof context testing
  it('handles unknown contexts gracefully', () => {
    render(
      <TestWrapper context="new-context">
        <Button variant="primary">Test</Button>
      </TestWrapper>
    )
    const button = screen.getByRole('button')
    expect(button).toHaveAttribute('data-context', 'new-context')
    expect(button).toHaveClass('default-primary') // Fallback styling
  })
})
```

```tsx
// Interaction tests
describe('Interactions', () => {
  it('handles click events', async () => {
    const user = userEvent.setup()
    const handleClick = vi.fn()

    render(<Button onClick={handleClick}>Click me</Button>)

    await user.click(screen.getByRole('button'))
    expect(handleClick).toHaveBeenCalledTimes(1)
  })

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

    render(<Button disabled onClick={handleClick}>Disabled</Button>)

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

  it('shows loading state', () => {
    render(<Button isLoading>Loading</Button>)
    expect(screen.getByRole('button')).toHaveAttribute('aria-busy', 'true')
  })
})

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

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

    render(<Button onClick={handleClick}>Test</Button>)

    await user.tab()
```

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

    await user.keyboard('[Enter]')
    expect(handleClick).toHaveBeenCalledTimes(1)
  })

  it('has proper ARIA attributes', () => {
    render(<Button aria-label="Custom label">Test</Button>)
    expect(screen.getByRole('button')).toHaveAttribute('aria-label', 'Custom label')
  })
})

// Performance tests
describe('Performance', () => {
  it('renders quickly', () => {
    const startTime = performance.now()
    render(<Button>Performance Test</Button>)
    const endTime = performance.now()

    expect(endTime - startTime).toBeLessThan(50) // 50ms threshold
  })

  it('handles multiple re-renders efficiently', () => {
    const { rerender } = render(<Button>Test</Button>)

    const startTime = performance.now()
    for (let i = 0; i < 100; i++) {
      rerender(<Button>Test {i}</Button>)
    }
    const endTime = performance.now()

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

## Context-Aware Testing Utilities

typescript

```tsx
// test/utils/context-testing.ts
import { render, RenderOptions } from '@testing-library/react'
import { WorkspaceProvider } from '@/contexts/WorkspaceContext'
import { ThemeProvider } from '@/contexts/ThemeContext'

// All registered contexts (future-proof)
export const REGISTERED_CONTEXTS = [
  'consultant',
  'client',
  'admin',
  'marketplace',
  // Future contexts will be added here automatically
] as const

export type RegisteredContext = typeof REGISTERED_CONTEXTS[number]

interface CustomRenderOptions extends RenderOptions {
  context?: RegisteredContext | string
  theme?: string
  user?: any
  workspace?: any
}

export const customRender = (
  ui: React.ReactElement,
  {
    context = 'consultant',
    theme = 'default',
    user = { id: 'test-user', role: 'consultant' },
    workspace = { id: 'test-workspace', name: 'Test Workspace' },
    ...renderOptions
  }: CustomRenderOptions = {}
) => {
  const Wrapper = ({ children }: { children: React.ReactNode }) => (
    <WorkspaceProvider context={context} workspace={workspace}>
      <ThemeProvider theme={theme}>
        <UserProvider user={user}>
          {children}
        </UserProvider>
      </ThemeProvider>
    </WorkspaceProvider>
  )
```

```
  return render(ui, { wrapper: Wrapper, ...renderOptions })
}

// Context testing helper
export const testAllContexts = (
  component: React.ReactElement,
  testFn: (context: RegisteredContext) => void
) => {
  REGISTERED_CONTEXTS.forEach(context => {
    describe(`Context: ${context}`, () => {
      testFn(context)
    })
  })
}

// Performance testing helper
export const measureRenderTime = (renderFn: () => void): number => {
  const startTime = performance.now()
  renderFn()
  const endTime = performance.now()
  return endTime - startTime
}
```

---

## 🔗 INTEGRATION TESTING PATTERNS

### Component Integration Tests

typescript

```tsx
// FormField.integration.test.tsx
import { render, screen } from '@testing-library/react'
import { userEvent } from '@testing-library/user-event'
import { customRender, testAllContexts } from '@/test/utils/context-testing'
import { FormField } from './FormField'
import { Button } from '../Button'

describe('FormField Integration', () => {
 // Test component composition
 describe('Component Composition', () => {
  it('integrates with form validation', async () => {
   const user = userEvent.setup()
   const handleSubmit = vi.fn()

   customRender(
    <form onSubmit={handleSubmit}>
     <FormField
      name="email"
      label="Email"
      type="email"
      required
      validation={{
       required: 'Email is required',
       pattern: {
        value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
        message: 'Invalid email format'
       }
      }}
     />
     <Button type="submit">Submit</Button>
    </form>
   )

   // Test validation
   await user.click(screen.getByRole('button', { name: 'Submit' }))
   expect(screen.getByText('Email is required')).toBeInTheDocument()

   // Test valid input
   await user.type(screen.getByLabelText('Email'), 'test@example.com')
   await user.click(screen.getByRole('button', { name: 'Submit' }))
   expect(handleSubmit).toHaveBeenCalled()
  })
 })
```

```
// Test context propagation
describe('Context Propagation', () => {
  testAllContexts(
    <FormField name="test" label="Test Field" />,
    (context) => {
      it(`propagates ${context} context to child components`, () => {
        customRender(
          <FormField name="test" label="Test Field" />,
          { context }
        )

        const input = screen.getByLabelText('Test Field')
        expect(input).toHaveAttribute('data-context', context)
      })
    }
  )
})

// Test theme integration
describe('Theme Integration', () => {
  it('applies theme correctly across components', () => {
    customRender(
      <FormField name="test" label="Test Field" />,
      { context: 'consultant', theme: 'dark' }
    )

    const field = screen.getByTestId('form-field-container')
    expect(field).toHaveClass('dark-theme')
    expect(field).toHaveClass('consultant-context')
  })
})
})
```

---

## 🎭 STORYBOOK TESTING INTEGRATION

### Story-Based Testing

typescript

```typescript
// Button.stories.test.tsx
import { test, expect } from '@storybook/test'
import { composeStories } from '@storybook/testing-react'
import * as stories from './Button.stories'

const { Default, AllVariants, WorkspaceContexts } = composeStories(stories)

test('Default story renders correctly', async () => {
  const canvas = await Default.play()
  await expect(canvas.getByRole('button')).toBeInTheDocument()
})

test('All variants render without errors', async () => {
  const canvas = await AllVariants.play()
  await expect(canvas.getByText('Primary')).toBeInTheDocument()
  await expect(canvas.getByText('Secondary')).toBeInTheDocument()
  await expect(canvas.getByText('Outline')).toBeInTheDocument()
})

test('Workspace contexts apply correctly', async () => {
  const canvas = await WorkspaceContexts.play()

  const consultantButton = canvas.getByText('Consultant')
  const clientButton = canvas.getByText('Client')

  await expect(consultantButton).toHaveClass('consultant-primary')
  await expect(clientButton).toHaveClass('client-primary')
})
```

## Visual Regression Testing

typescript

```typescript
// chromatic.config.js
module.exports = {
  projectToken: process.env.CHROMATIC_PROJECT_TOKEN,
  buildScriptName: 'build-storybook',
  exitZeroOnChanges: true,
  onlyChanged: true,
  ignoreLastBuildOnBranch: 'main',

  // Context-aware snapshots
  modes: {
    'consultant-light': {
      globals: {
        theme: 'light',
        context: 'consultant'
      }
    },
    'consultant-dark': {
      globals: {
        theme: 'dark',
        context: 'consultant'
      }
    },
    'client-light': {
      globals: {
        theme: 'light',
        context: 'client'
      }
    },
    'client-dark': {
      globals: {
        theme: 'dark',
        context: 'client'
      }
    }
    // Future contexts will be added automatically
  }
}
```

---

## 🎯 ACCESSIBILITY TESTING

### Automated Accessibility Testing

typescript

```typescript
// accessibility.test.ts
import { axe, toHaveNoViolations } from 'jest-axe'
import { render } from '@testing-library/react'
import { composeStories } from '@storybook/testing-react'
import * as stories from './Button.stories'

expect.extend(toHaveNoViolations)

const { Default, AllVariants, WorkspaceContexts } = composeStories(stories)

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

  it('All variants have no accessibility violations', async () => {
    const { container } = render(<AllVariants />)
    const results = await axe(container, {
      rules: {
        'color-contrast': { enabled: true },
        'keyboard-navigation': { enabled: true },
        'aria-labels': { enabled: true }
      }
    })
    expect(results).toHaveNoViolations()
  })

  it('Workspace contexts maintain accessibility', async () => {
    const { container } = render(<WorkspaceContexts />)
    const results = await axe(container)
    expect(results).toHaveNoViolations()
  })
})
```

## Manual Accessibility Testing Checklist

markdown

## Accessibility Testing Checklist

### Keyboard Navigation
- [ ] All interactive elements are keyboard accessible
- [ ] Tab order is logical and intuitive
- [ ] Focus indicators are visible and clear
- [ ] No keyboard traps exist
- [ ] Escape key works for dismissible components

### Screen Reader Support
- [ ] All elements have appropriate ARIA labels
- [ ] Headings are properly structured (h1-h6)
- [ ] Form fields are properly labeled
- [ ] Error messages are announced
- [ ] Loading states are announced

### Color & Contrast
- [ ] Color contrast meets WCAG AA standards (4.5:1 for normal text)
- [ ] Information isn't conveyed by color alone
- [ ] Focus indicators have sufficient contrast
- [ ] All contexts maintain contrast standards

### Responsive Design
- [ ] Components work at 200% zoom
- [ ] Touch targets are at least 44px
- [ ] Text can be resized up to 200% without scrolling
- [ ] Mobile accessibility is equivalent to desktop

### Testing Tools
- [ ] axe-core automated testing passes
- [ ] WAVE browser extension shows no errors
- [ ] VoiceOver/NVDA manual testing completed
- [ ] Color blindness simulator tested

---

# 🚀 PERFORMANCE TESTING

## Bundle Size Testing

typescript

```typescript
// bundle-size.test.ts
import { execSync } from 'child_process'
import { readFileSync } from 'fs'
import { gzipSync } from 'zlib'

describe('Bundle Size Tests', () => {
  const MAX_BUNDLE_SIZE = 500 * 1024 // 500KB

  it('total bundle size is under limit', () => {
    execSync('npm run build', { stdio: 'inherit' })

    const bundleStats = JSON.parse(
      readFileSync('dist/bundle-stats.json', 'utf8')
    )

    const totalSize = bundleStats.assets.reduce(
      (sum, asset) => sum + asset.size,
      0
    )

    expect(totalSize).toBeLessThan(MAX_BUNDLE_SIZE)
  })

  it('component chunks are appropriately sized', () => {
    const componentSizes = {
      atoms: 5 * 1024,      // 5KB per atom
      molecules: 15 * 1024,  // 15KB per molecule
      organisms: 50 * 1024   // 50KB per organism
    }

    Object.entries(componentSizes).forEach(([type, maxSize]) => {
      const componentBundle = readFileSync(`dist/${type}.js`)
      const gzippedSize = gzipSync(componentBundle).length

      expect(gzippedSize).toBeLessThan(maxSize)
    })
  })
})
```

## Render Performance Testing

typescript

```tsx
// performance.test.tsx
import { render } from '@testing-library/react'
import { measureRenderTime } from '@/test/utils/context-testing'
import { Button } from './Button'

describe('Performance Tests', () => {
  it('renders quickly', () => {
    const renderTime = measureRenderTime(() => {
      render(<Button>Test</Button>)
    })

    expect(renderTime).toBeLessThan(16) // 60fps = 16ms budget
  })

  it('handles rapid re-renders efficiently', () => {
    const { rerender } = render(<Button>Initial</Button>)

    const startTime = performance.now()
    for (let i = 0; i < 100; i++) {
      rerender(<Button>Update {i}</Button>)
    }
    const endTime = performance.now()

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

  it('memory usage remains stable', () => {
    const initialMemory = performance.memory?.usedJSHeapSize || 0

    // Render and unmount 100 times
    for (let i = 0; i < 100; i++) {
      const { unmount } = render(<Button>Test {i}</Button>)
      unmount()
    }

    // Force garbage collection if available
    if (global.gc) {
      global.gc()
    }

    const finalMemory = performance.memory?.usedJSHeapSize || 0
    const memoryIncrease = finalMemory - initialMemory
```

```
    expect(memoryIncrease).toBeLessThan(1024 * 1024) // Less than 1MB increase
  })
})
```

---

## 🔧 TEST AUTOMATION & CI/CD

### GitHub Actions Testing Workflow

yaml

```yaml
# .github/workflows/component-tests.yml
name: Component Tests

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [18.x, 20.x]
        context: [consultant, client, admin, marketplace]

    steps:
    - uses: actions/checkout@v3

    - name: Setup Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v3
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run unit tests
      run: npm test -- --context=${{ matrix.context }}
      env:
        CI: true
        WORKSPACE_CONTEXT: ${{ matrix.context }}

    - name: Run integration tests
      run: npm run test:integration

    - name: Run accessibility tests
      run: npm run test:a11y

    - name: Run performance tests
      run: npm run test:performance
```

```yaml
    - name: Upload coverage reports
      uses: codecov/codecov-action@v3
      with:
        file: ./coverage/lcov.info
        flags: unittests
        name: codecov-umbrella
        fail_ci_if_error: true

  visual-tests:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
      with:
        fetch-depth: 0

    - name: Setup Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '18.x'
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run Chromatic
      uses: chromaui/action@v1
      with:
        token: ${{ secrets.GITHUB_TOKEN }}
        projectToken: ${{ secrets.CHROMATIC_PROJECT_TOKEN }}
        buildScriptName: build-storybook
        exitZeroOnChanges: true
        onlyChanged: true
```

## Test Coverage Reports

typescript

```typescript
// coverage.config.js
module.exports = {
  collectCoverageFrom: [
    'src/**/*.{ts,tsx}',
    '!src/**/*.stories.{ts,tsx}',
    '!src/**/*.test.{ts,tsx}',
    '!src/**/*.d.ts',
    '!src/test/**/*'
  ],
  coverageThreshold: {
    global: {
      branches: 90,
      functions: 90,
      lines: 90,
      statements: 90
    },
    './src/components/atoms/': {
      branches: 95,
      functions: 95,
      lines: 95,
      statements: 95
    },
    './src/components/molecules/': {
      branches: 90,
      functions: 90,
      lines: 90,
      statements: 90
    },
    './src/components/organisms/': {
      branches: 85,
      functions: 85,
      lines: 85,
      statements: 85
    }
  },
  coverageReporters: ['text', 'lcov', 'html', 'json-summary']
}
```

---

## 📊 TESTING METRICS & REPORTING

### Test Quality Metrics

typescript

```typescript
// test-metrics.ts
export interface TestMetrics {
  coverage: {
    lines: number
    branches: number
    functions: number
    statements: number
  }
  performance: {
    renderTime: number
    bundleSize: number
    memoryUsage: number
  }
  accessibility: {
    violations: number
    wcagLevel: 'AA' | 'AAA'
    score: number
  }
  contexts: {
    tested: string[]
    passed: string[]
    failed: string[]
  }
}

export const generateTestReport = (metrics: TestMetrics): string => {
  return `
# Component Test Report

## Coverage
- Lines: ${metrics.coverage.lines}%
- Branches: ${metrics.coverage.branches}%
- Functions: ${metrics.coverage.functions}%
- Statements: ${metrics.coverage.statements}%

## Performance
- Render Time: ${metrics.performance.renderTime}ms
- Bundle Size: ${(metrics.performance.bundleSize / 1024).toFixed(2)}KB
- Memory Usage: ${(metrics.performance.memoryUsage / 1024).toFixed(2)}KB

## Accessibility
- Violations: ${metrics.accessibility.violations}
- WCAG Level: ${metrics.accessibility.wcagLevel}
```

```
- Score: ${metrics.accessibility.score}%

## Context Testing
- Tested: ${metrics.contexts.tested.join(', ')}
- Passed: ${metrics.contexts.passed.join(', ')}
- Failed: ${metrics.contexts.failed.join(', ')}
`
}
```

---

## 🎯 FUTURE-PROOF TESTING PATTERNS

### Dynamic Context Testing

typescript

```typescript
// dynamic-context.test.ts
import { getRegisteredContexts } from '@/contexts/ContextRegistry'
import { customRender } from '@/test/utils/context-testing'
import { Button } from './Button'

describe('Dynamic Context Testing', () => {
  // Test all registered contexts automatically
  it('works with all registered contexts', () => {
    const contexts = getRegisteredContexts()

    contexts.forEach(context => {
      customRender(<Button>Test</Button>, { context })

      const button = screen.getByRole('button')
      expect(button).toHaveAttribute('data-context', context)
      expect(button).toHaveClass(`${context}-primary`)
    })
  })

  // Test context inheritance
  it('handles context inheritance correctly', () => {
    const parentContext = 'consultant'
    const childContext = 'consultant-admin'

    customRender(
      <Button context={childContext}>Inherited</Button>,
      { context: parentContext }
    )

    const button = screen.getByRole('button')
    expect(button).toHaveClass('consultant-base')
    expect(button).toHaveClass('consultant-admin-variant')
  })
})
```

## Extensible Test Utilities

typescript

```typescript
// test/utils/future-proof-testing.ts
import { ComponentType } from 'react'

// Generic component testing
export const testComponent = <T extends ComponentType<any>>(
  Component: T,
  props: React.ComponentProps<T>,
  options: {
    contexts?: string[]
    variants?: string[]
    accessibility?: boolean
    performance?: boolean
  } = {}
) => {
  const {
    contexts = ['consultant'],
    variants = ['default'],
    accessibility = true,
    performance = true
  } = options

  describe(`${Component.displayName || Component.name}`, () => {
    // Test all contexts
    contexts.forEach(context => {
      describe(`Context: ${context}`, () => {
        variants.forEach(variant => {
          it(`renders ${variant} variant correctly`, () => {
            customRender(
              <Component {...props} variant={variant} />,
              { context }
            )

            expect(screen.getByTestId('component')).toBeInTheDocument()
          })
        })
      })
    })

    // Accessibility tests
    if (accessibility) {
      it('meets accessibility standards', async () => {
        const { container } = customRender(<Component {...props} />)
        const results = await axe(container)
```

```
      expect(results).toHaveNoViolations()
    })
  }

  // Performance tests
  if (performance) {
    it('meets performance standards', () => {
      const renderTime = measureRenderTime(() => {
        customRender(<Component {...props} />)
      })

      expect(renderTime).toBeLessThan(16) // 60fps budget
    })
  }
 })
}
```

---

## 🚀 TESTING BEST PRACTICES

### Test Organization

```
src/
├── components/
│   ├── atoms/
│   │   ├── Button/
│   │   │   ├── Button.tsx
│   │   │   ├── Button.stories.tsx
│   │   │   ├── Button.test.tsx          # Unit tests
│   │   │   ├── Button.integration.test.tsx # Integration tests
│   │   │   └── Button.a11y.test.tsx     # Accessibility tests
│   │   └── ...
│   ├── molecules/
│   └── organisms/
├── test/
│   ├── utils/                    # Testing utilities
│   ├── fixtures/                 # Test data
│   ├── setup.ts                  # Test setup
│   └── __mocks__/                # Mock files
└── ...
```

### Test Naming Convention

typescript

```typescript
// ✅ Good: Clear, descriptive test names
describe('Button Component', () => {
  describe('Rendering', () => {
    it('renders with default props')
    it('renders with all variants')
    it('renders with custom className')
  })

  describe('Workspace Context', () => {
    it('applies consultant context styling')
    it('applies client context styling')
    it('handles unknown contexts gracefully')
  })

  describe('Interactions', () => {
    it('handles click events')
    it('prevents interaction when disabled')
    it('shows loading state correctly')
  })
})

// ❌ Bad: Vague, unclear test names
describe('Button', () => {
  it('works')
  it('handles clicks')
  it('has styles')
})
```

## Test Data Management

typescript

```typescript
// test/fixtures/contexts.ts
export const CONTEXT_FIXTURES = {
  consultant: {
    user: { role: 'consultant', name: 'John Doe' },
    workspace: { type: 'consultant', name: 'Consulting Firm' },
    theme: 'consultant-light'
  },
  client: {
    user: { role: 'client', name: 'Jane Smith' },
    workspace: { type: 'client', name: 'Client Portal' },
    theme: 'client-light'
  },
  // Future contexts will be added here
}

// test/fixtures/components.ts
export const COMPONENT_FIXTURES = {
  button: {
    default: { children: 'Default Button' },
    primary: { variant: 'primary', children: 'Primary Button' },
    disabled: { disabled: true, children: 'Disabled Button' }
  },
  // Other component fixtures
}
```

This comprehensive testing strategy ensures that your components are reliable, accessible, performant, and future-proof across all workspace contexts and use cases.