DuocUC

Métodos Asociados al Testing en React

1. Métodos de Queries (React Testing Library)

1.1 Queries de un solo elemento

getBy* Retorna el elemento o lanza error si no existe. Uso: cuando **debe** existir.

```
screen.getByRole('button')
screen.getByText('Hola Mundo')
screen.getByLabelText('Email')
screen.getByPlaceholderText('Buscar...')
screen.getByAltText('Logo')
screen.getByTitle('Cerrar')
screen.getByDisplayValue('valor actual')
screen.getByTestId('custom-element')
```

Ejemplo:

```
test('renderiza el botón de submit', () => {
    render(<Form />)
    const button = screen.getByRole('button', { name: /enviar/i })
    expect(button).toBeInTheDocument()
})
```

queryBy*

Retorna el elemento o null si no existe. Uso: para verificar **ausencia**.

```
const element = screen.queryByText('No existe')
expect(element).not.toBeInTheDocument()        // ✅
expect(element).toBeNull()                     // ✅
```

Ejemplo:

```
test('no muestra el modal inicialmente', () => {
  render(<App />)
  const modal = screen.queryByRole('dialog')
  expect(modal).not.toBeInTheDocument()
})
```

findBy*

Retorna una **promesa** que resuelve cuando encuentra el elemento. Uso: elementos **asíncronos**.
const element = await screen.findByText('Cargado')

Ejemplo:

```
test('muestra datos después de cargar', async () => {
  render(<UserProfile userId={1} />)
  const userName = await screen.findByText('Juan Pérez')
  expect(userName).toBeInTheDocument()
})
```

1.2 Queries de múltiples elementos

getAllBy*

Retorna array de elementos o error si no encuentra ninguno.
const items = screen.getAllByRole('listitem')
expect(items).toHaveLength(5)

queryAllBy*

Retorna array (puede estar vacío).
const items = screen.queryAllByRole('listitem')
expect(items).toHaveLength(0) // ✅ Verifica que no hay elementos

findAllBy*

Retorna promesa con array de elementos.
const items = await screen.findAllByRole('listitem')
expect(items).toHaveLength(3)

1.3 Jerarquía de prioridad de Queries

Orden recomendado (de más a menos preferido):

1. **getByRole** - Más accesible
   screen.getByRole('button', { name: /submit/i })
   screen.getByRole('textbox', { name: /email/i })
   screen.getByRole('heading', { name: /título/i, level: 1 })

2. **getByLabelText** - Para formularios
   screen.getByLabelText('Nombre de usuario')
   screen.getByLabelText(/contraseña/i)

3. **getByPlaceholderText** - Para inputs sin label visible
   screen.getByPlaceholderText('juan@example.com')

4. **getByText** - Para contenido no interactivo
   screen.getByText('Bienvenido al sistema')
   screen.getByText(/error:/i)

5. **getByDisplayValue** - Para valores actuales de formularios
   screen.getByDisplayValue('valor predeterminado')

6. **getByAltText** - Para imágenes
   screen.getByAltText('Logo de la empresa')

7. **getByTitle** - Para tooltips
   screen.getByTitle('Información adicional')

8. **getByTestId** - Último recurso
   screen.getByTestId('custom-component')

2. Métodos de Interacción (User Event)

2.1 Setup

Siempre inicializa userEvent antes de usarlo:

```
import userEvent from '@testing-library/user-event'

test('interacción', async () => {
  const user = userEvent.setup()
  // ... usar user
})
```

2.2 Click

```
await user.click(element)
await user.dblClick(element)
await user.tripleClick(element)
```

Ejemplo:

```
test('incrementa contador al hacer click', async () => {
        const user = userEvent.setup()
        render(<Counter />)

        const button = screen.getByRole('button', { name: /incrementar/i})
        await user.click(button)
        expect(screen.getByText('Contador: 1')).toBeInTheDocument()
})
```

2.3 Type (Escribir)

```
await user.type(input, 'texto a escribir')
await user.type(input, 'hola{Enter}') // Con teclas especiales
```

Ejemplo:

```
test('permite escribir en el input', async () => {
  const user = userEvent.setup()
  render(<SearchBox />)

  const input = screen.getByRole('textbox')
  await user.type(input, 'React Testing')

  expect(input).toHaveValue('React Testing')
})
```

2.4 Clear (Limpiar)

```
await user.clear(input)
```

Ejemplo:

```
test('limpia el input', async () => {
  const user = userEvent.setup()
  render(<Form />)
    const input = screen.getByLabelText(/email/i)
  await user.type(input, 'test@example.com')
 await user.clear(input)
  expect(input).toHaveValue('')
})
```

2.5 SelectOptions (Seleccionar)

```
await user.selectOptions(select, 'valor')
await user.selectOptions(select, ['valor1', 'valor2']) // Múltiple
```

Ejemplo:

```
test('selecciona una opción del dropdown', async () => {
  const user = userEvent.setup()
  render(<CountrySelector />)

  const select = screen.getByRole('combobox')
  await user.selectOptions(select, 'chile')

  expect(screen.getByRole('option', { name: 'Chile' }).selected).toBe(true)
})
```

2.6 Upload (Subir archivos)

```
const file = new File(['contenido'], 'archivo.txt', { type: 'text/plain' })
await user.upload(input, file)
```

Ejemplo:

```
test('permite subir un archivo', async () => {
  const user = userEvent.setup()
  render(<FileUploader />)
  const input = screen.getByLabelText(/subir archivo/i)
  const file = new File(['hello'], 'hello.txt', { type: 'text/plain' })
  await user.upload(input, file)
  expect(input.files[0]).toBe(file)
  expect(input.files).toHaveLength(1)
})
```

2.7 Hover y Unhover

```
await user.hover(element)
await user.unhover(element)
```

Ejemplo:

```
test('muestra tooltip al hacer hover', async () => {
  const user = userEvent.setup()
  render(<Button />)

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

  expect(screen.getByText('Información del botón')).toBeVisible()
})
```

2.8 Tab (Navegación con teclado)

```
await user.tab() // Tab adelante
await user.tab({ shift: true }) // Tab atrás
```

Ejemplo:

```
test('navega con Tab entre inputs', async () => {
  const user = userEvent.setup()
  render(<Form />)

  const nameInput = screen.getByLabelText(/nombre/i)
  const emailInput = screen.getByLabelText(/email/i)
```

```
    nameInput.focus()
    await user.tab()

    expect(emailInput).toHaveFocus()
})
```

2.9 Keyboard (Teclas especiales)

```
await user.keyboard('{Enter}')
await user.keyboard('{Escape}')
await user.keyboard('{Control>}a{/Control}')          // Ctrl+A
await user.keyboard('{Shift>}ABC{/Shift}')            // ABC en mayúsculas
```

Ejemplo:

```
test('envía formulario con Enter', async () => {
  const user = userEvent.setup()
  const handleSubmit = jest.fn()
  render(<SearchForm onSubmit={handleSubmit} />)

  const input = screen.getByRole('textbox')
  await user.type(input, 'búsqueda{Enter}')

  expect(handleSubmit).toHaveBeenCalledWith('búsqueda')
})
```

2.10 Copy, Cut, Paste

```
await user.copy()
await user.cut()
await user.paste('texto pegado')
```

3. Métodos de FireEvent (Alternativa más simple)

Nota: Preferir userEvent por ser más realista.

```
import { fireEvent } from '@testing-library/react'

fireEvent.click(button)
fireEvent.change(input, { target: { value: 'nuevo valor' } })
fireEvent.submit(form)
fireEvent.focus(input)
fireEvent.blur(input)
fireEvent.keyDown(input, { key: 'Enter', code: 'Enter' })
fireEvent.mouseOver(element)
fireEvent.mouseOut(element)
```

Ejemplo:

```
test('cambia el valor del input', () => {
  render(<Input />)
  const input = screen.getByRole('textbox')

  fireEvent.change(input, { target: { value: 'nuevo texto' } })

  expect(input).toHaveValue('nuevo texto')
})
```

4. Métodos de Render

4.1 render()

```
const {
  container,      // Nodo DOM del contenedor
  rerender,       // Re-renderizar con nuevas props
  unmount,        // Desmontar el componente
  debug,          // Imprimir el DOM
  asFragment      // Obtener snapshot
} = render(<Component />)
```

Ejemplo completo:

```
test('re-renderiza con nuevas props', () => {
  const { rerender } = render(<Greeting name="Juan" />)
  expect(screen.getByText('Hola, Juan')).toBeInTheDocument()

  rerender(<Greeting name="María" />)
  expect(screen.getByText('Hola, María')).toBeInTheDocument()
})
```

4.2 cleanup()

Limpia el DOM después de cada test (automático en RTL):

```
import { cleanup } from '@testing-library/react'

afterEach(() => {
  cleanup()
})
```

4.3 renderHook()

Para testear hooks personalizados:

```
import { renderHook, act } from '@testing-library/react'

const { result, rerender, unmount } = renderHook(() => useCustomHook())
```

Ejemplo:

```
test('useCounter incrementa correctamente', () => {
  const { result } = renderHook(() => useCounter(0))

  act(() => {
    result.current.increment()
  })
  expect(result.current.count).toBe(1)
})
```

5. Métodos de Espera (Async)

5.1 waitFor()

Espera hasta que se cumpla una condición:

```
await waitFor(() => {
  expect(screen.getByText('Cargado')).toBeInTheDocument()
})

// Con opciones
await waitFor(
  () => {
    expect(mockFn).toHaveBeenCalled()
  },
```

```
 {
  timeout: 3000,       // Tiempo máximo de espera
  interval: 100,       // Intervalo entre intentos
  onTimeout: (error) => {
   console.error('Timeout:', error)
  }
 }
)
```

Ejemplo:

```
test('muestra mensaje de éxito después de guardar', async () => {
 render(<SaveButton />)
 const button = screen.getByRole('button', { name: /guardar/i })

 fireEvent.click(button)

 await waitFor(() => {
  expect(screen.getByText('Guardado exitosamente')).toBeInTheDocument()
 })
})
```

5.2 waitForElementToBeRemoved()

Espera a que un elemento desaparezca del DOM:

```
await waitForElementToBeRemoved(() => screen.getByText('Cargando...'))

// O con el elemento directamente
const loader = screen.getByText('Cargando...')
await waitForElementToBeRemoved(loader)
```

Ejemplo:

```
test('oculta el spinner después de cargar', async () => {
 render(<DataLoader />)

 const spinner = screen.getByText('Cargando...')
 await waitForElementToBeRemoved(spinner)

 expect(screen.getByText('Datos cargados')).toBeInTheDocument()
})
```

6. Métodos Auxiliares

6.1 within()

Limita las queries a un elemento específico:

```
import { within } from '@testing-library/react'
const sidebar = screen.getByRole('complementary')
const button = within(sidebar).getByRole('button')
```

Ejemplo:

```
test('busca botón dentro del modal', () => {
 render(<App />)

 const modal = screen.getByRole('dialog')
 const closeButton = within(modal).getByRole('button', { name: /cerrar/i })

 expect(closeButton).toBeInTheDocument()
})
```

6.2 screen.debug()

Imprime el DOM actual en la consola:

```
screen.debug()                // Todo el documento
screen.debug(element)         // Un elemento específico
screen.debug(element, 20000)  // Con más caracteres
```

Ejemplo:

```
test('debug para ver el DOM', () => {
  render(<ComplexComponent />)

  screen.debug() // Ver qué se renderizó

  const button = screen.getByRole('button')
  screen.debug(button) // Ver solo el botón
})
```

6.3 screen.logTestingPlaygroundURL()

Genera una URL para Testing Playground:

```
screen.logTestingPlaygroundURL()
```

7. Métodos de Jest (Matchers)

7.1 Matchers Básicos

```
expect(value).toBe(5)                // Igualdad estricta (===)
expect(value).toEqual({ a: 1 })      // Igualdad profunda
expect(value).not.toBe(3)            // Negación
expect(value).toBeTruthy()           // Valor verdadero
expect(value).toBeFalsy()            // Valor falso
expect(value).toBeNull()             // Es null
expect(value).toBeUndefined()        // Es undefined
expect(value).toBeDefined()          // Está definido
expect(value).toBeNaN()              // Es NaN
```

7.2 Matchers Numéricos

```
expect(value).toBeGreaterThan(3)
expect(value).toBeGreaterThanOrEqual(3.5)
expect(value).toBeLessThan(5)
expect(value).toBeLessThanOrEqual(4.5)
expect(value).toBeCloseTo(0.3, 2)        // Números flotantes
```

7.3 Matchers de Strings

```
expect(string).toMatch(/pattern/)
expect(string).toMatch('substring')
expect(string).toContain('texto')
expect(string).toHaveLength(10)
```

7.4 Matchers de Arrays y Objetos

```
expect(array).toContain('item')
expect(array).toContainEqual({ id: 1 })
expect(array).toHaveLength(3)
expect(array).toEqual(expect.arrayContaining(['a', 'b']))
expect(object).toHaveProperty('name')
expect(object).toHaveProperty('age', 25)
expect(object).toMatchObject({ id: 1 })
```

7.5 Matchers de Funciones (Mocks)

```
expect(mockFn).toHaveBeenCalled()
expect(mockFn).toHaveBeenCalledTimes(2)
expect(mockFn).toHaveBeenCalledWith('arg1', 'arg2')
expect(mockFn).toHaveBeenLastCalledWith('arg')
expect(mockFn).toHaveBeenNthCalledWith(1, 'first call')
expect(mockFn).toHaveReturned()
expect(mockFn).toHaveReturnedTimes(3)
expect(mockFn).toHaveReturnedWith('value')
expect(mockFn).toHaveLastReturnedWith('last value')
```

7.6 Matchers de Jest-DOM

```
expect(element).toBeInTheDocument()
expect(element).toBeVisible()
expect(element).toBeEmpty()
expect(element).toBeDisabled()
expect(element).toBeEnabled()
expect(element).toBeInvalid()
expect(element).toBeRequired()
expect(element).toHaveAttribute('href', '/home')
expect(element).toHaveClass('active', 'highlight')
expect(element).toHaveStyle({ color: 'red' })
expect(element).toHaveTextContent('texto')
expect(input).toHaveValue('valor')
expect(input).toHaveDisplayValue('display')
expect(checkbox).toBeChecked()
expect(checkbox).toBePartiallyChecked()
expect(element).toHaveFocus()
expect(element).toContainElement(otherElement)
expect(element).toContainHTML('<span>content</span>')
expect(form).toHaveFormValues({ username: 'john' })
```

8. Métodos de Mocking (Jest)

8.1 jest.fn()

```
const mockFn = jest.fn()
const mockFn = jest.fn(x => x * 2)
const mockFn = jest.fn().mockName('myMock')
```

8.2 Configurar retornos

```
mockFn.mockReturnValue(42)
mockFn.mockReturnValueOnce(1).mockReturnValueOnce(2)
mockFn.mockResolvedValue('success')          // Promise resuelta
mockFn.mockRejectedValue(new Error('failed'))   // Promise rechazada
```

8.3 Configurar implementación

```
mockFn.mockImplementation((x, y) => x + y)
mockFn.mockImplementationOnce(x => x * 2)
```

8.4 jest.spyOn()

```
const spy = jest.spyOn(object, 'method')
spy.mockReturnValue('mocked')
spy.mockRestore()              // Restaurar original
spy.mockClear()                // Limpiar llamadas
spy.mockReset()                // Limpiar todo
```

8.5 jest.mock()

```
jest.mock('./module')              // Mock automático
jest.mock('./module', () => ({      // Mock manual
  función: jest.fn()
}))
```

9. Métodos de Lifecycle

```
beforeAll(() => {
  // Se ejecuta UNA VEZ antes de todos los tests
})

afterAll(() => {
  // Se ejecuta UNA VEZ después de todos los tests
})

beforeEach(() => {
  // Se ejecuta ANTES de CADA test
})

afterEach(() => {
  // Se ejecuta DESPUÉS de CADA test
})
```

Ejemplo completo:

```
describe('Calculator', () => {
  let calculator

  beforeEach(() => {
    calculator = new Calculator()
  })

  afterEach(() => {
    calculator = null
  })

  test('suma dos números', () => {
    expect(calculator.add(2, 3)).toBe(5)
  })

  test('resta dos números', () => {
    expect(calculator.subtract(5, 3)).toBe(2)
  })
})
```

10. Métodos de Agrupación

10.1 describe()

Agrupa tests relacionados:

```
describe('UserForm', () => {
  describe('validación', () => {
    test('valida email', () => {})
    test('valida contraseña', () => {})
  })

  describe('envío', () => {
    test('envía datos correctamente', () => {})
  })
})
```

10.2 test() / it()

Son sinónimos:

```
test('hace algo', () => {})
it('hace algo', () => {})
```

10.3 Modificadores

```
test.only('solo ejecuta este', () => {})                 // Solo este test
test.skip('omite este', () => {})                        // Omitir test
test.todo('implementar después')                         // Test pendiente
test.each([...])('test parametrizado', () => {})         // Tests con datos
```

Ejemplo de test.each:

```
test.each([
  [1, 1, 2],
  [2, 2, 4],
  [3, 3, 6]
])('suma %i + %i = %i', (a, b, expected) => {
  expect(a + b).toBe(expected)
})
```