Testing

Introducción

- Dividir en componentes hace la app más testable
- Vamos a testar diferentes aspectos del componente
  - estructura
  - comportamiento
  - integración
  - lógica

Introducción

- Necesitamos (otra) herramienta: **jest**
  - desarrollada por Facebook
  - framework de test de propósito general
  - optimizada para testear React

testing/001

Principios de testing

- Estructura de los tests:
  - Partimos de un estado controlado
  - Ejecutamos el código que queremos testear
  - Comparamos el resultado obtenido con el resultado esperado

Hola, Mundo

redradix

```
function suma(a, b) {
  return a + b
}

export default suma
```

redradix

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

```
import suma from 'suma'

test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})
```

Testing

```
$ npm test
```

Ejercicio

- Partiendo de **/001**
  - Escribe un fichero **resta.js** que implemente la función **resta**
  - Escribe **tres o cuatro** casos de test en **resta.test.js**

Ejercicio

- Partiendo del ejercicio anterior
  - Escribe algunos casos más de test utilizando **diferentes matchers**
  - https://facebook.github.io/jest/docs/en/using-matchers.html

```
test('debería sumar 1 + 1 = 2', () => {
  expect(suma(1, 1)).toBe(2)
})

test('debería sumar 1 + 0 = 1', () => {
  expect(suma(1, 0)).toBe(1)
})

test('devuelve NaN si sólo recibe un parámetro', () => {
  expect(suma(1)).toBe(NaN)
})
```

```
describe('función suma', () => {

  test('debería sumar 1 + 1 = 2', () => {
    expect(suma(1, 1)).toBe(2)
  })

  test('debería sumar 1 + 0 = 1', () => {
    expect(suma(1, 0)).toBe(1)
  })

  test('devuelve NaN si sólo recibe un parámetro', () => {
    expect(suma(1)).toBe(NaN)
  })

})
```

Componentes

redradix

Testear un componente

- Aplicamos la misma filosofía
  - Importamos el componente
  - Lo llevamos a un **estado inicial controlado**
  - Ejecutamos **el código que queramos testear**
  - **Inspeccionamos** el componente para asegurarnos que se comporta correctamente

Testear un componente

- Vamos a testear **por separado:**
  - **aspecto + comportamiento**
    - presentacionales
  - **estado + lógica**
    - contenedores

Presentacionales

redradix

testing/002

redradix

Testear un componente

- Queremos:
  - **montar** el componente
  - en un contexto que podamos **manejar desde el código** (no en un navegador)
  - que nos permita **inspeccionar** su HTML y su estado

Testear un componente

- **Enzyme**
  - librería para testear **componentes React**
  - **monta** componentes en *“modo test”*
  - nos permite consultar su HTML, sus props, su estado, etc, ...

```jsx
import React from 'react'

const Button = (props) => (
  <button onClick={props.action} disabled={props.disabled}>
    {props.label}
  </button>
)

export default Button
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

redradix

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

redradix

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería mostrar el texto indicado en la prop label', () => {
    const wrapper = mount(<Button label="test"/>)
    expect(wrapper.find('button').text()).toBe('test')
  })

})
```

Ejercicio

- Partiendo de **testing/002**
  - Escribe algunos casos más sobre su presentación
    - `wrapper.find(...).props()`
    - `wrapper.find(...).hasClass('mi-clase')`
    - `wrapper.find(...).html()`

Testear un componente

- Para testear su **comportamiento** necesitamos...
  - **Simular eventos**
    - `click`
    - `keypress`
    - `etc...`

Testear un componente

- Para testear su **comportamiento** necesitamos...
  - **Funciones espía**
  - Guardan información sobre sus invocaciones
    - cuántas veces han sido llamadas
    - con qué parámetros
    - con qué contexto
    - etc...

```
const espia = jest.fn()

espia.mock.calls // []

espia(1)

espia.mock.calls // [ [1] ]

espia(1, 2, 3)

espia.mock.calls // [ [1], [1, 2, 3] ]

espia.mock.calls.length // 2
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería ejecutar action cuando sea clicado', () => {
    const spy = jest.fn()
    const wrapper = mount(<Button action={spy}/>)
    wrapper.find('button').simulate('click')
    expect(spy.mock.calls.length).toBe(1)
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería ejecutar action cuando sea clicado', () => {
    const spy = jest.fn()
    const wrapper = mount(<Button action={spy}/>)
    wrapper.find('button').simulate('click')
    expect(spy.mock.calls.length).toBe(1)
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería ejecutar action cuando sea clicado', () => {
    const spy = jest.fn()
    const wrapper = mount(<Button action={spy}/>)
    wrapper.find('button').simulate('click')
    expect(spy.mock.calls.length).toBe(1)
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería ejecutar action cuando sea clicado', () => {
    const spy = jest.fn()
    const wrapper = mount(<Button action={spy}/>)
    wrapper.find('button').simulate('click')
    expect(spy.mock.calls.length).toBe(1)
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Button from 'components/button'

describe('<Button/>', () => {

  test('debería ejecutar action cuando sea clicado', () => {
    const spy = jest.fn()
    const wrapper = mount(<Button action={spy}/>)
    wrapper.find('button').simulate('click')
    expect(spy.mock.calls.length).toBe(1)
  })

})
```

Contenedores

Testear un contedor

- Queremos:
  - **montar** el componente
  - simular **interacción** de un usuario
  - comprobar que **la lógica** se ejecuta correctamente
    - inspeccionar **el estado**
    - inspeccionar **las consecuencias**

```
import React from 'react'
import Button from 'components/button'

export default class Counter extends React.Component {
  constructor() {
    super()
    this.state = { count: 0 }
    this.increment = this.increment.bind(this)
  }
  increment() {
    this.setState({ count: this.state.count + 1 })
  }
  render() {
    return (
      <div className="counter">
        <h1>{this.state.count}</h1>
        <Button label="+1" action={this.increment}/>
      </div>
    )
  }
}
```

```
import React from 'react'
import { mount } from 'enzyme'

import Counter from 'components/counter'

describe('<Counter/>', () => {

  test('debería empezar en 0', () => {
    const wrapper = mount(<Counter/>)
    expect(wrapper.state().count).toBe(0)
  })

  test('debería incrementar en 1 al clicar el botón', () => {
    const wrapper = mount(<Counter/>)
    wrapper.find('button').simulate('click')
    expect(wrapper.state().count).toBe(1)
    expect(wrapper.find('h1').text()).toBe('1')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Counter from 'components/counter'

describe('<Counter/>', () => {

  test('debería empezar en 0', () => {
    const wrapper = mount(<Counter/>)
    expect(wrapper.state().count).toBe(0)
  })

  test('debería incrementar en 1 al clicar el botón', () => {
    const wrapper = mount(<Counter/>)
    wrapper.find('button').simulate('click')
    expect(wrapper.state().count).toBe(1)
    expect(wrapper.find('h1').text()).toBe('1')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Counter from 'components/counter'

describe('<Counter/>', () => {

  test('debería empezar en 0', () => {
    const wrapper = mount(<Counter/>)
    expect(wrapper.state().count).toBe(0)
  })

  test('debería incrementar en 1 al clicar el botón', () => {
    const wrapper = mount(<Counter/>)
    wrapper.find('button').simulate('click')
    expect(wrapper.state().count).toBe(1)
    expect(wrapper.find('h1').text()).toBe('1')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Counter from 'components/counter'

describe('<Counter/>', () => {

  test('debería empezar en 0', () => {
    const wrapper = mount(<Counter/>)
    expect(wrapper.state().count).toBe(0)
  })

  test('debería incrementar en 1 al clicar el botón', () => {
    const wrapper = mount(<Counter/>)
    wrapper.find('button').simulate('click')
    expect(wrapper.state().count).toBe(1)
    expect(wrapper.find('h1').text()).toBe('1')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Counter from 'components/counter'

describe('<Counter/>', () => {

  test('debería empezar en 0', () => {
    const wrapper = mount(<Counter/>)
    expect(wrapper.state().count).toBe(0)
  })

  test('debería incrementar en 1 al clicar el botón', () => {
    const wrapper = mount(<Counter/>)
    wrapper.find('button').simulate('click')
    expect(wrapper.state().count).toBe(1)
    expect(wrapper.find('h1').text()).toBe('1')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Counter from 'components/counter'

describe('<Counter/>', () => {

  test('debería empezar en 0', () => {
    const wrapper = mount(<Counter/>)
    expect(wrapper.state().count).toBe(0)
  })

  test('debería incrementar en 1 al clicar el botón', () => {
    const wrapper = mount(<Counter/>)
    wrapper.find('button').simulate('click')
    expect(wrapper.state().count).toBe(1)
    expect(wrapper.find('h1').text()).toBe('1')
  })

})
```

```
import React from 'react'
import { mount } from 'enzyme'

import Counter from 'components/counter'

describe('<Counter/>', () => {

  test('debería empezar en 0', () => {
    const wrapper = mount(<Counter/>)
    expect(wrapper.state().count).toBe(0)
  })

  test('debería incrementar en 1 al clicar el botón', () => {
    const wrapper = mount(<Counter/>)
    wrapper.find('button').simulate('click')
    expect(wrapper.state().count).toBe(1)
    expect(wrapper.find('h1').text()).toBe('1')
  })

})
```

Ejercicio

- Copia el **cronómetro** (React) a la carpeta **002/src**
- Escribe algunos casos de test para...
  - Componentes presentacionales
  - Componentes contendores

Redux

Testear con Redux

- Muy fácil!
  - Muy pocas dependencias
  - Código muy encapsulado
  - Lógica de negocio en funciones puras
  - Muy fácil de mockear

Testear con Redux

- Vamos a testar cada parte por separado:
  - action creators
  - reducers
  - componentes conectados

action creators

Testear Redux: action creators

- Action creators son...
  - funciones
  - que pueden recibir parámetros
  - y devuelven **un objeto** (acción)

Testear Redux: action creators

- Para testear, tengo que:
  - cargar el fichero de action creators
  - invocar al action creator que quiera testear
  - comprobar **el objeto** que me devuelve
  - y listo!

testing/003

reducers

Testear Redux: reducers

- Un reducer es...
  - una función
  - que recibe el **estado actual** y una **acción**
  - y devuelve el **estado siguiente**

Testear Redux: reducers

- Para testear un reducer, necesito...
  - crear un store
  - inicializarlo con un **estado inicial** controlado
  - llamar al reducer con el estado y **la acción** que quiero testear
  - comprobar el **estado que me devuelve**

testing/003

¡Gracias!

redradix