React

Import & export

Export & Import

- Dos palabras reservadas:
  - **`export`** permite exportar *nombres* y hacerlos visibles para otros ficheros
  - **`import`** permite importar *nombres* desde otros ficheros

cuatro.js:

```
export default 4
```

index.js:

```
import cuatro from './cuatro.js'
```

numbers.js:

```
export const uno = 1
export const dos = 2
```

index.js:

```
import { uno } from './numbers.js'
```

numbers.js:

```
export const uno = 1
export const dos = 2
```

index.js:

```
import * as numbers from './numbers.js'
numbers.uno
numbers.dos
```

Ejercicio

- Crea un fichero **numbers.js** que exporte **uno**, **dos** y **tres**
- Crea un fichero **index.js** que los importe **TODOS** y loguee su valor por la consola

```
import { uno } from './numbers.js'
console.log(`Valor importado: ${uno}`)
```

```
import { uno } from './numbers.js'
```

```
import { uno as one } from './numbers'
```

Import & Export

```
$ npm run start
```

redradix

Import & Export

- Vamos a manejar **tres directorios**:
  - **`/src`**: nuestro código fuente
  - **`/public`**: ficheros estáticos
  - **`/build`**: resultado de **empaquetar** nuestra app

Hola, Mundo

redradix

Introducción

- React es una **librería**
- Para crear **interfaces de usuario**
- Componentes
- Filosofía funcional

```
import React from 'react'
import ReactDOM from 'react-dom'

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, Mundo!</h1>
  }
}

window.onload = () => ReactDOM.render(
  React.createElement(HolaMundo),
  document.getElementById('app')
)
```

Hola, Mundo

```
$ npm start
```

```
<html>
  <Meta charset="utf-8"/>
  <head>
    <title>React Experiments</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="bundle.js"></script>
  </body>
</html>
```

```
import React from 'react'
import ReactDOM from 'react-dom'

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, Mundo!</h1>
  }
}

window.onload = () => ReactDOM.render(
  React.createElement(HolaMundo),
  document.getElementById('app')
)
```

```
import React from 'react'
import ReactDOM from 'react-dom'

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, Mundo!</h1>
  }
}

window.onload = () => ReactDOM.render(
  React.createElement(HolaMundo),
  document.getElementById('app')
)
```

```
_createClass(HolaMundo, [{
  key: 'render',
  value: function render() {
    return _react2.default.createElement(
      'h1',
      null,
      'Hola, Mundo!'
    );
  }
}]);
```

redradix

JSX

- JSX es un dialecto XML
- Facilitar la construcción de componentes
- Babel lo traduce a Javascript
- Lo podemos utilizar en cualquier expresión

```
const header = (<h1>Hello, World!</h1>)

const list = [
  <li>Uno</li>,
  <li>Dos</li>,
  <li>Tres</li>
]
```

```
class HolaMundo extends React.Component {
  render() {
    const m = 'Mundo'
    return <h1>Hola, {m}!</h1>
  }
}
```

```
class HolaMundo extends React.Component {
  render() {
    const m = 'Mundo'
    return <h1>Hola, {m}!</h1>
  }
}
```

Ejercicio

- Escribe...
  - Un componente Reloj
  - Que muestre horas, minutos y segundos
  - Utiliza HolaMundo como referencia

```
const HolaMundo = () => {
  return <h1>Hola, Mundo</h1>
}
```

```
const HolaMundo = () => <h1>Hola, Mundo!</h1>
```

redradix

Organización del Código

```
const Mundo = () => <span>Mundo</span>

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, <Mundo/>!</h1>
  }
}
```

```
const Mundo = () => <span>Mundo</span>

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, <Mundo/>!</h1>
  }
}
```

Ejercicio

- Divide Reloj en 5 componentes
  - **App**: el componente de nivel superior
  - **Reloj**: que se compone de Horas, Minutos y Segundos
  - **Horas**/**Minutos**/**Segundos**: cada uno muestra el valor correspondiente

Organización de Código

- Cada componente en su propio fichero
- Utilizar **import/export**
- Múltiples filosofías para organizar los ficheros...
- Nosotros vamos a crear una carpeta **src/components**

Ejercicio

- Reorganiza el Reloj
  - **src/index.js**: importa y monta el componente App
  - **src/App.js**: Importa Reloj y exporta App
  - **src/components/Reloj.js**: importa Horas, Minutos y Segundos y exporta Reloj
  - **src/components/Horas.js**: exporta Horas
  - ...

Props

```
class HolaCosa extends React.Component {
  render() {
    return (<h1> Hola, {this.props.cosa}!</h1>)
  }
}

class App extends React.Component {
  render() {
    return <HolaCosa cosa="Mundo" />
  }
}
```

```
class HolaCosa extends React.Component {
  render() {
    return (<h1> Hola, {this.props.cosa}!</h1>)
  }
}

class App extends React.Component {
  render() {
    return <HolaCosa cosa="Mundo" />
  }
}
```

```
const HolaCosa = (props) => <h1>Hola, {props.cosa}!</h1>

class App extends React.Component {
  render() {
    return <HolaCosa cosa="Mundo" />
  }
}
```

```
const HolaCosa = (props) => <h1>Hola, {props.cosa}!</h1>

class App extends React.Component {
  render() {
    return <HolaCosa cosa="Mundo" />
  }
}
```

Ejercicio

- Modifica el Reloj
  - Para que Horas, Minutos y Segundos reciban el valor que tienen que mostrar como **prop**

Ejercicio

- Modifica el Reloj
  - Reemplaza Horas, Minutos y Segundos por un solo componente: Segmento
  - Segmento recibe el valor que tiene que mostrar como **prop**

```
class RedBox extends React.Component {
  render() {
    return <div style={{background: 'red'}}>
      {this.props.children}
    </div>
  }
}

class App extends React.Component {
  render() {
    return <RedBox> <h1>Cuidado!!</h1> </RedBox>
  }
}
```

```
class RedBox extends React.Component {
  render() {
    return <div style={{background: 'red'}}>
      {this.props.children}
    </div>
  }
}

class App extends React.Component {
  render() {
    return <RedBox> <h1>Cuidado!!</h1> </RedBox>
  }
}
```

JSX vs. HTML

Diferencias entre JSX y HTML

- **`className`** en vez de **`class`**
- **`htmlFor`** en vez de **`for`**
- **`<input/>`** en vez de **`<input>`**
- **`style`** recibe un objeto, no un string
- los eventos se escriben en camelCase
    - `onChange`, `onClick`, etc...

Ejercicio

- Dale estilos al Reloj
  - Crea una hoja de estilos e inclúyela en el HTML
  - Dale la clase “segmento” a Segmento
  - Escribe algunos estilos para la clase “segmento”

Ejercicio

- Partiendo de **react/003**
  - Toma como referencia **dist/clock.html**
  - Traduce el reloj a componentes
  - Escribe la lógica para que las agujas se coloquen correctamente
  - Utiliza **style** y **rotate** para rotar las agujas

Ejercicio

- Partiendo del **HTML** en **react/004**
  - Modela la tabla con **componentes**
  - **<Table>**, **<TableRow>**, y un componente para cada tipo de celda
  - Organiza el código correctamente

Ciclo de vida

Ciclo de vida

- Podemos definir funciones que se ejecutarán en diferentes momentos del ciclo de vida de nuestro componente
- Útiles para configurar el componente o para limpiar cuando se vaya a desmontar

Inicialización

- `constructor()`
- `render()`
- `componentDidMount()`

Actualización

- `shouldComponentUpdate()`
- `render()`
- `componentDidUpdate()`

Destrucción

- `componentWillUnmount()`

Error

- `componentDidCatch()`

```
class MyComponent extends React.Component {
  constructor() {
    super()
    console.log('constructor!')
  }
  componentDidMount() {
    console.log('componente en la página')
  }
  componentWillUnmount() {
    console.log('a punto de ser eliminado')
  }
}
```

Estado

Estado

- Una **propiedad especial** de los componentes
  - un objeto en el que podemos guardar lo que queramos
  - cada vez que modificamos el contenido, el **componente se re-rendea**

```
class MyComponent extends React.Component {
  constructor() {
    super()
    this.state = { prop: 'value' }
  }
}
```

```
this.setState({ prop: 'value', prop2: 'value2' })
```

```
this.state.prop
```

Ejercicio

- Modifica el reloj para que se actualice cada segundo

Estado

- Cada vez que se **actualiza** el estado....
  - se **re-rendea el componente**
  - y **todos** sus hijos
  - pero **React** optimiza este proceso

Estado

- La **representación** de un componente debería depender **exclusivamente** de:
  - sus **props**
  - su **estado**

Ejercicio

- ¿Qué pasa si llamamos a **setState** desde **render**?

Eventos

redradix

```
class Button extends React.Component {
  render() {
    return <button onClick={() => alert('Hola!')}>
       Saludar
    </button>
  }
}
```

Eventos

- Buenas prácticas
  - centralizar el estado en componentes **inteligentes**
  - que **orquestan** componentes **presentacionales**
  - se comunican con sus hijos...
    - mediante props
    - y callbacks

```
class MyButton extends React.Component {
  render() {
      return <button onClick={this.props.action}>Click Me!</button>
  }
}

class App extends React.Component {
  render() {
      return <MyButton action={() => alert('Thank you...')} />
  }
}
```

Ejercicio

- partiendo de **react/005**, programa un **cronómetro**
  - Empieza con 3 botones:
    - **START** comienza el contador
    - **PAUSE** / **RESUME** pausa el contador
    - **STOP** para y resetea el contador
  - Los botones se activan sólo cuando son relevantes
  - Utiliza el html en **public/cronometro.html**

Ejercicio

- Modifica el **cronómetro** añadiendo un botón **LAP**
  - guarda el tiempo actual
  - lo añade a una lista de tiempos guardados

```
class Lista extends React.Component {
  render() {
    const lista = [<li>Uno</li>, <li>Dos</li>];
    return <ul> {lista} </ul>
  }
}
```

Eventos

- Al interpolar componentes desde una lista...
  - Hay que añadir a cada componente una propiedad **key**
  - Única para cada elemento
  - React lo necesita para distinguir cual es cual

```
class Lista extends React.Component {
  render() {
    const lista = [<li key={1}>Uno</li>, <li key={2}>Dos</li>];
    return <ul> {lista} </ul>
  }
}
```

Ejercicio

- Partiendo de **react/006**, programa **Tooltip**
  - configurable mediante props
  - utiliza **onMouseEnter** y **onMouseLeave**
  - muestra y oculta la burbuja gestionando la clase **is-active**

Eventos

- El objeto **event** que reciben los handlers...
  - construido por **React**
  - pero similar al nativo
  - **preventDefault**, **stopPropagation**, etc...

Formularios

redradix

Formularios

- Los formularios en React...
  - funcionan de manera peculiar
  - conflicto entre el estado natural de los formularios y el control de React

```
class MyForm extends React.Component {
  render() {
    return <input type="text" value="Fijo"/>
  }
}
```

Formularios

- Tenemos dos opciones:
  - componentes **controlados**
    - gestionar a mano el valor de cada input
  - componentes **no controlados**
    - respetar el comportamiento nativo y recolectar los valores al final del proceso

Componentes controlados

```
class App extends React.Component {
  constructor() {
    super()
    this.state = { counter: 0 }
  }
  inc() {
    this.setState({ counter: this.state.counter + 1 })
  }
  render() {
    return (
      <div>
        <input type="text" value={this.state.counter} />
        <button onClick={this.inc.bind(this)}>Incrementar!</button>
      </div>
    )
  }
}
```

Componentes controlados

- Para crear un input editable...
  - escuchar al evento **onChange**
  - guardar **`event.target.value`** en el **estado**
  - usar ese valor como propiedad **value** del input

```
class App extends React.Component {
  constructor() {
    super()
    this.state = { value: '' }
    this.handleChange = this.handleChange.bind(this)
  }
  handleChange(e) {
    this.setState({ value: e.target.value })
  }
  render() {
    return <input type="text"
                  value={this.state.value}
                  onChange={this.handleChange} />
  }
}
```

Ejercicio

- Escribe un componente **NumericInput**
  - que sólo permita **escribir números**

```
class App extends React.Component {
  constructor() {
    super()
    this.state = { value: 'dos' }
    this.handleChange = this.handleChange.bind(this)
  }
  handleChange(e) {
    this.setState({ value: e.target.value })
  }
  render() {
    return <select value={this.state.value} onChange={this.handleChange}>
      <option value='uno'>1</option>
      <option value='dos'>2</option>
      <option value='tres'>3</option>
    </select>
  }
}
```

```
class App extends React.Component {
  constructor() {
    super()
    this.state = { isChecked: '' }
    this.handleChange = this.handleChange.bind(this)
  }
  handleChange(e) {
    this.setState({ isChecked: e.target.checked })
  }
  render() {
    return <form>
      <label>
        <input type="checkbox" checked={this.state.isChecked}
               onChange={this.handleChange} />
        Checked?
      </label>
    </form>
  }
}
```

Ejercicio

- ¿Cómo podríamos controlar **múltiples** checks a la vez?
  - Escribe un componente que gestione 5 checkboxes

Ejercicio

- Partiendo de **react/007**
  - Escribe un aplicación **To Do**
  - Utilizando todo lo que hemos aprendido

Componentes no controlados

```
class App extends React.Component {
  constructor() {
    super()
    this.handleSubmit = this.handleSubmit.bind(this)
  }
  handleSubmit(e) {
    // extraemos el valor del input
  }
  render() {
    return <form onSubmit={this.handleSubmit}>
      <input type="text"/>
      <input type="submit" value="adelante!" />
    </form>
  }
}
```

Componentes no controlados

- Para guardar una referencia a un elemento...
  - utilizamos la propiedad especial **ref**
  - recibe **una función**
  - la invoca **inmediatamente**
  - pasándole **una referencia al elemento** como primer parámetro

```
class App extends React.Component {
  constructor() {
    super()
    this.handleSubmit = this.handleSubmit.bind(this)
  }
  handleSubmit(e) {
    e.preventDefault()
    alert(`Has escrito: ${this.input.value}`)
  }
  render() {
    return <form onSubmit={this.handleSubmit}>
      <input type="text" ref={(el) => this.input = el}/>
      <input type="submit" value="adelante!" />
    </form>
  }
}
```

Routing

Routing

```
$ npm install -S react-router-dom
```

Routing

- react/009/index.js

```
import React from 'react'
import {
  BrowserRouter as Router,
  Route,
  Switch,
  Link
} from 'react-router-dom'
```

```
class App extends React.Component {
  render() {
    return <Router>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/cara" component={Cara} />
        <Route path="/cruz" component={Cruz} />
      </Switch>
    </Router>
  }
}
```

```
class Home extends React.Component {
  render() {
    return <div>
      <h1>Elige:</h1>
      <ul>
        <li><Link to="/cara">cara</Link></li>
        <li><Link to="/cruz">cruz</Link></li>
      </ul>
    </div>
  }
}
```

```
class Cara extends React.Component {
  render() {
    return  <div>
      <h1>Cara</h1>
      <Link to="/cruz">ir a Cruz</Link>
    </div>
  }
}

class Cruz extends React.Component {
  render() {
    return  <div>
      <h1>Cruz</h1>
      <Link to="/cara">ir a Cara</Link>
    </div>
  }
}
```

Ejercicio

- Partiendo de **react/008**
  - Traduce el HTML a componentes
  - Conecta la navegación con un Router
  - *consejo: intenta sacar componentes comunes*
    - ***Layout***, ***Footer***, *etc...*

```
export default () => {
  return <Router>
    <Switch>
      <Route exact path="/" component={Home} />
      <Route path="/timer/:time" component={Timer} />
    </Switch>
  </Router>
}
```

```
class Home extends React.Component {
  render() {
    return <div>
      <h1>Home</h1>
      <Link to='/timer/190'>Go to timer</Link>
    </div>
  }
}
```

```
class Timer extends React.Component {
  render() {
    return <div>
      <h1>Timer: {this.props.match.params.time}</h1>
      <Link to="/">Go to Home</Link>
    </div>
  }
}
```

```
class Timer extends React.Component {
  render() {
    return <div>
      <h1>Timer: {this.props.match.params.time}</h1>
      <Link to="/">Go to Home</Link>
    </div>
  }
}
```

```
export default class Home extends React.Component {
  componentDidMount() {
    const { push } = this.props.history
    setTimeout(() => push('/timer/10'), 1000)
  }
  render() {
    return <div>
      <h1>Home</h1>
      <Link to='/timer/190'>Go to timer</Link>
      </div>
  }
}
```

```
export default class Home extends React.Component {
  componentDidMount() {
    const { push } = this.props.history
    setTimeout(() => push('/timer/10'), 1000)
  }
  render() {
    return <div>
      <h1>Home</h1>
      <Link to='/timer/190'>Go to timer</Link>
      </div>
  }
}
```

Ejercicio

- Partiendo de **react/009**, haz una app con dos páginas
  - La primera tiene un **formulario** para introducir un número de minutos
  - La segunda muestra un **contador** con los minutos introducidos
  - Se pasan el parámetro por url

```
export default class App extends React.Component {
  render() {
    const { users } = this.state
    return (
      <Router>
        <Switch>
          <Route exact path="/" render={(props) => <UserList {...props} users={users} />} />
        </Switch>
      </Router>
    )
  }
}
```

Ejercicio

- Partiendo de **react/010**, haz una app con 4 páginas
  - **Listado** de usuarios
  - Formulario para **crear** usuarios
  - Formulario para **editar** usuarios
  - Página de confirmación de **borrado** de usuarios

AJAX

AJAX

- Para cargar datos desde una API por AJAX...
  - lanzamos la petición en **componentDidMount**
  - mostramos un spinner
  - cuando la petición termine...
    - quitamos el spinner
    - si todo ha ido bien, guardamos los datos en **state**
    - en caso de error, notificamos al usuario

Ejercicio

- Partiendo de **react/011**
  - Escribe un módulo que haga una petición a `https://query.yahooapis.com` para consultar el tiempo que hace en **Madrid**
  - **https://developer.yahoo.com/weather/**
  - Escribe un componente que utilice el módulo y muestre el resultado

Ejercicio

- Modifica el ejercicio anterior
  - Añade un **TextInput** y un botón para que el usuario pueda seleccionar la ciudad que quiere consultar
  - Muestra un **Spinner** mientras carga el resultado
  - Muestra un **Error** si la ciudad no existe

Testing

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

```
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
```

```
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/001**
  - 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 contenedor

- 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

- Testea el **cronómetro**
- Escribe algunos casos de test para...
  - Componentes presentacionales
  - Componentes contendores