

# Contenido

- 3. Vistas con React
  - ¿Qué es React?
  - JSX
  - Propiedades y estado de un componente
  - Ciclo de vida de un componente
  - Eventos
  - Formularios
  - Composición
  - Integrar con Stores y Dispatcher
  - Ejercicio

# ¿Qué es React?

- Una librería Javascript para construir interfaces de usuario
- Las interfaces se construyen mediante una jerarquía de **componentes**

# ¿Qué es React?

- Sólo para UI
- Utiliza virtual DOM para mayor eficiencia
- Flujo de datos unidireccional

**UI = f(datos+estado)**

# ¿Qué es React?

- Cada componente define su **salida** como una función pura
- Cada componente describe “cómo” debe ser el HTML que genera

# ¿Qué es React?

```
var React = require('react');

var Saludo = React.createClass({
  render: function() {
    return (
      <div>
        <h1>Hola mundo</h1>
      </div>
    )
  }
});

module.exports = Saludo;
```

# ¿Qué es React?

```
var React = require('react');

var Saludo = React.createClass({
  render: function() {
    return (
      <div>
        <h1>Hola mundo</h1>
      </div>
    )
  }
});

module.exports = Saludo;
```

# JSX

- Una sintaxis basada en XML
- Muy muy muy similar a HTML, pero...
- se “compila” a Javascript, ¡no a HTML!
- Permite crear componentes autocontenidos: la definición de UI y comportamiento en el mismo fichero

# JSX

```
var HolaMundo = React.createClass ({  
  render: function () {  
    return (  
      <div className="panel">Hola mundo!</div>  
    );  
  }  
});
```




Constructor  
del componente  
(factoría)



# JSX


```
var HolaMundo = React.createClass({  
  render: function() {  
    return (  
      <div className="panel">Hola mundo!</div>  
    );  
  }  
});
```



## render

Método que llamará React para “pintar”  
el componente, **obligatorio**

# JSX

```
var HolaMundo = React.createClass({  
  render: function() {  
    return (  
      <div className="panel">Hola mundo!</div>  
    );  
  }  
});
```

La salida del componente. Este código dice que **siempre** que se pinte este componente, tendrá que mostrarse con un DIV con una clase CSS “panel” y el texto “Hola mundo!” dentro

# JSX compilado

```
var HolaMundo = React.createClass({  
  render: function() {  
    return (  
      React.createElement('div', { className: 'panel' }, 'Hola mundo!');  
    );  
  }  
});
```

Resultado de la compilación:  
Javascript puro

# JSX

- Es una sintaxis **cómoda** para evitar las mismas llamadas a `React.createElement(...)` una y otra vez
- Además, se parece a HTML que es lo que estamos produciendo al final

# JSX

- Es sencillo copiar HTML de una plantilla o maqueta a un componente de React
- Sólo hay que cambiar los elementos de HTML cuya sintaxis no es válida en Javascript y seguir ciertas convenciones
  - `class` -> `className` (para definir clases CSS)
  - `for` -> `htmlFor` (en `<label>` de formularios)
  - `camelCase` para eventos (`onChange`, `onClick`)

# JSX

- Un componente puede generar HTML (string), o bien otros componentes (clases)
- Convención
  - etiqueta empieza con minúscula: HTML (caso anterior)
  - etiqueta empieza con mayúscula: componente

# JSX

```
var Saludo = React.createClass({  
  
  render: function() {  
  
    return (<HolaMundo />);  
  
  }  
  
})
```

Si no existe una referencia local a la clase del componente **HolaMundo**, el compilador de JSX lanzará un error.

# JSX

Para mostrar un componente en la página, debemos indicar a React el componente que queremos pintar, y el punto de montaje en el DOM

```
React.render(<Saludo />, document.body)
```



# Ejercicio 1: primer componente

- Crea un componente cuya salida (render) sea un texto cualquiera
- Incluye ese componente en la página con `React.render`
- Utiliza el esqueleto en el repo para tener ya la compilación y el servidor web local listo

**> node /ejercicios/tema3/index.js**

- Aplicación cliente en `/ejercicios/tema3/src/index.js`

# JSX

- Dentro de **render** podemos escribir código Javascript, expresiones, etc.
- Podemos incluir código/expresiones Javascript en la salida JSX, encerrando la expresión entre llaves

# JSX

```
var ComponentWithExpressions = React.createClass({  
  render: function() {  
    var usuario = {  
      name: "John",  
      lastName: "McEnroe"  
    };  
  
    return (  
      <div>  
        <p>Su nombre es { usuario.name } y su apellido es { usuario.lastname }</p>  
      </div>  
    );  
  }  
});
```

El compilador interpreta las expresiones  
{ XXX } dentro de JSX como Javascript

# JSX - listas de componentes

- La salida de un componente debe ser exactamente **un nodo**
- Un nodo = un control HTML | un componente
- Tiene su lógica: el compilador JSX convierte nuestro **return** en una expresión Javascript del tipo `React.createElement...` por lo que tiene que ser **una** llamada, un nodo.

# JSX - listas de componentes

- Entonces, ¿cómo pintamos listas?
- Sencillo: el padre debe ser el **contenedor**
- Por eso normalmente vemos `<div ...></div>` como etiquetas de apertura y cierre de un componente

# JSX - listas de componentes

```
var React = require('react');

var Item = React.createClass({
  render: function() {
    return (<div>Soy uno más</div>);
  }
});

var Lista = React.createClass({
  render: function() {
    var items = [];
    for(var i=0; i < 100; i++) {
      items.push(<Item />);
    }
    return (
      <div>
        { items }
      </div>
    );
  }
});

module.exports = Lista;
```

# JSX - listas de componentes

```
var React = require('react');

var Item = React.createClass({
  render: function() {
    return (<div>Soy uno más</div>);
  }
});


var Lista = React.createClass({
  render: function() {
    var items = [];
    for(var i=0; i < 100; i++) {
      items.push(<Item />);
    }
    return (
      <div>
        { items }
      </div>
    );
  }
});

module.exports = Lista;
```

# JSX - listas de componentes

```
var Lista = React.createClass({
  displayName: 'Lista',

  render: function render() {
    var items = [];
    for (var i = 0; i < 100; i++) {
      items.push(React.createElement(Item, null));
    }
    return React.createElement(
      'div',
      null,
      items
    );
  }
});
```



El tercer argumento de `React.createElement` son... los hijos del componente :)



# JSX - listas de componentes

- Si ejecutamos el ejemplo anterior (/src/components/ejemplos/lista\_componentes.js)
- Y abrimos la consola Javascript del navegador...

⚠ Warning: Each child in an array or iterator should have a unique `key` prop. Check the render method of `Lista`. See <https://fb.me/react-warning-keys> for more information. `bundle.js:1734`

> |

# JSX - listas de componentes

- React necesita poder identificar los componentes idénticos dentro de un Array para su algoritmo de DOM virtual
- Así que nos pide que le digamos una clave (**key**) para usarlo como su “ID interno”
- Cualquier valor es válido: un número, un string... con tal que sea único **dentro de ese Array**

# JSX - listas de componentes

```
var Lista = React.createClass({
  render: function() {
    var items = [];
    for(var i=0; i < 100; i++) {
      items.push(<Item key={i} />);
    }
    return (
      <div>
        { items }
      </div>
    );
  }
});
```

Para eliminar el warning de nuestro ejemplo, simplemente damos como **key** el valor de **i** dentro del bucle

# Propiedades de un componente

- Los componentes aceptan parámetros o propiedades como atributos en JSX

```
<Saludo nombre="Daenerys" />
```

- Dentro del componente, se accede a estas propiedades con **this.props.nombre**

```
return (<div>Hola { this.props.nombre } !</div>)
```

# Propiedades de un componente

- Como JSX en realidad es Javascript, se pueden pasar como props:
- Escalares (números, booleanos, strings,...)
- Objetos complejos (como una colección de mori)
- Funciones

# Propiedades de un componente

```
var React = require('react');
var _ = require('mori');

var EjemploProps = React.createClass({
  myFunction: function() {
    alert("Boo!");
  },
  render: function() {
    var moriMap = _.hashMap("color", "red");
    var obj = { foo: 'bar' };
    return (
      <div>
        <OtroComponente
          text="hello"
          number={ 6 }
          thing={ obj }
          mori={ moriMap }
          func={ this.myFunction } />
      </div>
    );
  }
});
```

# Propiedades de un componente

- El uso de **props** es fundamental para construir la UI a partir de diferentes módulos
- En el **render** de un componente padre, decidimos los **props** que pasamos a los componentes hijos
- Así se consigue que la UI sea dinámica



# Propiedades de un componente

```
var React = require('react');

var Fecha = React.createClass({
  render: function() {
    return <p>En {this.props.country} son las {this.props.date.toTimeString()}</p>
  }
});

var FechasMundo = React.createClass({
  //...
});

module.exports = FechasMundo;
```


# Propiedades de un componente

```
var React = require('react');

var Fecha = React.createClass({
  render: function() {
    return <p>En {this.props.country} son las {this.props.date.toTimeString()}</p>
  }
});

var FechasMundo = React.createClass({
  //...
});

module.exports = FechasMundo;
```



La salida de este componente depende las propiedades  
**country** y **date** que reciba de su padre

# Propiedades de un componente

```
var FechasMundo = React.createClass({
  convertirZonaHoraria: function(fecha, deltaHoras){
    var d = new Date(fecha);
    d.setUTCHours(d.getUTCHours()+deltaHoras);
    return d;
  },
  render: function(){
    var zonasHorarias = [
      { country: 'España', difUTC: 2},
      { country: 'UK', difUTC: 0 },
      { country: 'Argentina', difUTC: -3 },
      { country: 'Mexico', difUTC: -5 },
      { country: 'Japon', difUTC: +5 },
      { country: 'Nueva Zelanda', difUTC: +12 },
    ];

    var ahora = new Date();

    var componentes = zonasHorarias.map(function(zona){
      return <Fecha key={ zona.country } country={ zona.country }
        date={ this.convertirZonaHoraria(ahora, zona.difUTC) } />;
    }, this);

    return (
      <div>
        { componentes }
      </div>
    );
  }
});

module.exports = FechasMundo;
```

# Propiedades de un componente

```
var FechasMundo = React.createClass({
  convertirZonaHoraria: function(fecha, deltaHoras){
    var d = new Date(fecha);
    d.setUTCHours(d.getUTCHours()+deltaHoras);
    return d;
  },
  render: function(){
    var zonasHorarias = [
      { country: 'España', difUTC: 2},
      { country: 'UK', difUTC: 0 },
      { country: 'Argentina', difUTC: -3 },
    ];
```

Configuramos un componente **Fecha** por cada elemento del Array, pasándolo como **props** el país y una fecha ajustada a la diferencia horaria

```

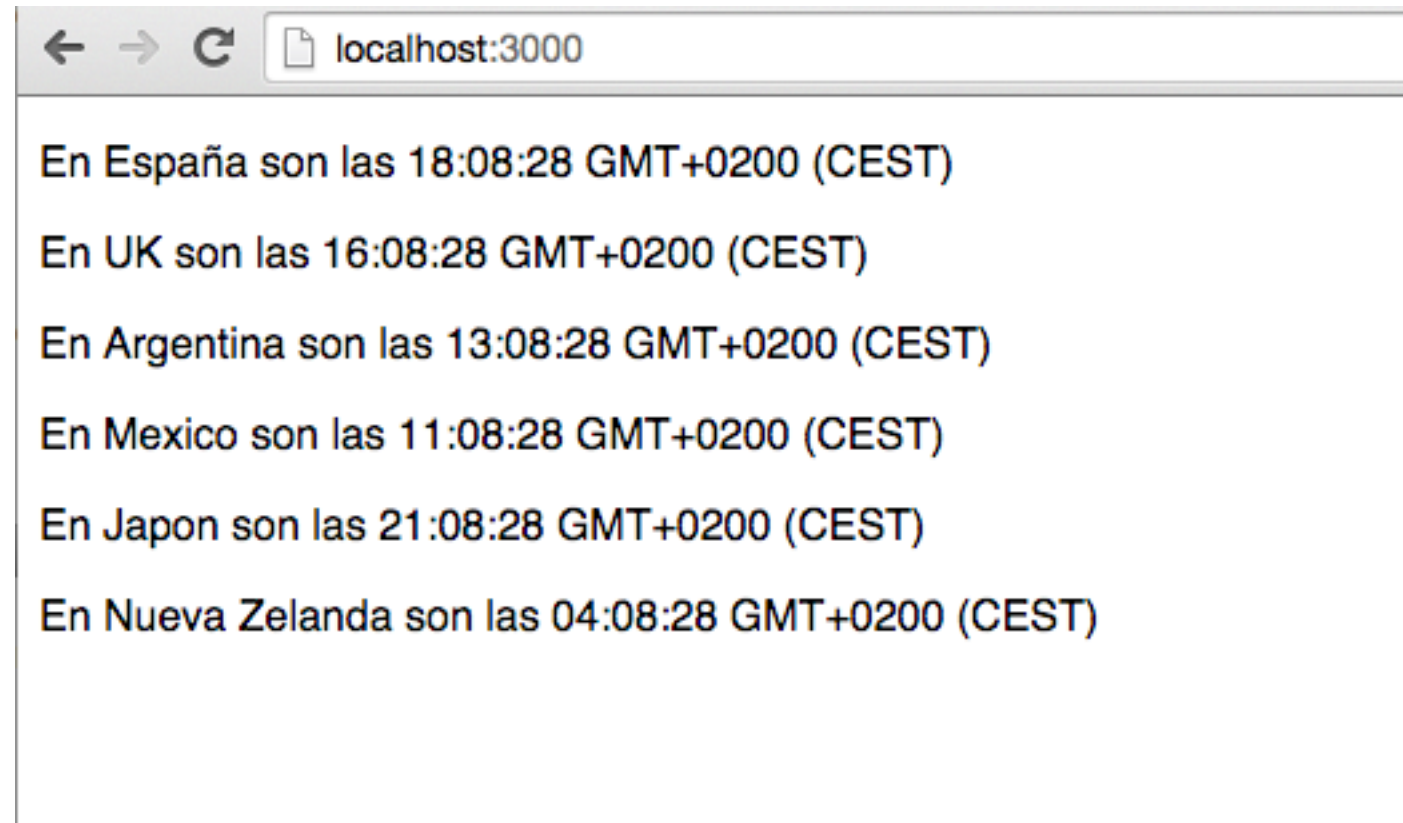
    return <Fecha key={ zona.country } country={ zona.country }
      date={ this.convertirZonaHoraria(ahora, zona.difUTC) } />;
  }, this);

  return (
    <div>
      { componentes }
    </div>
  );
});

module.exports = FechasMundo;
```

La salida de este componente es un Array de componentes **Fecha** dentro de una etiqueta DIV

# Propiedades de un componente



# Propiedades de un componente

- Un componente **no puede modificar sus props**
- El componente declara cuál es su salida **a partir de sus props**
- El componente **padre** es el dueño del hijo y por tanto controla su comportamiento mediante las **props** con las que lo configura
- Y así sucesivamente hacia arriba... hasta el componente raíz

## Ejercicio 2: props

- Modifica tu componente del ejercicio 1 para que acepte props, y utiliza estas props en su método render

# Validación de props


- Al crear un componente, podemos definir y documentar qué propiedades espera/necesita el componente

```
React.createClass({  
  propTypes: {  
    name: React.PropTypes.string,  
    ...  
  },  
  
  render: function() ...  
});
```



# Validación de props

```
React.createClass({  
  propTypes: {  
    name: React.PropTypes.string,  
    ...  
  },  
  
  render: function() ...  
});
```



En la definición

# Validación de props


```
React.createClass({  
  propTypes: {  
    name: React.PropTypes.string,  
  },  
  
  render: function() ...  
});
```



El nombre de la **prop**

# Validación de props

```
React.createClass({  
  propTypes: {  
    name: React.PropTypes.string,  
    ...  
  },  
  
  render: function() ...  
});
```



Constantes  
proporcionadas por  
React

# React.PropTypes

- array
- bool
- func
- number
- object
- string
- node (cualquiera valor representable)
- element (un elemento React)
- oneOf(['Value1', 'Value2'] - un valor enumerado
- ...

Si añadimos el sufijo `.isRequired` a cualquier tipo lo hacemos obligatorio:

**`React.PropTypes.string.isRequired`**

# Props por defecto

- Podemos definir los valores por defecto para las props de nuestro componente implementando la función **getDefaultProps()**

```
var ComponentWithDefaultProps = React.createClass({  
  getDefaultProps: function() {  
    return {  
      name: 'Unknown'  
    };  
  },  
  render: function() {  
    // ...  
  }  
});
```

# Estado del componente

- React considera que nuestros componentes son máquinas de estados finitos
- Además de props, los componentes tienen su estado interno (`this.state`), que es un objeto Javascript
- Igual que con las props, podemos definir un estado inicial implementado **`getInitialState()`** en el componente

# Estado del componente

```
var React = require('react');  
var MyComp = React.createClass({  
  getInitialState: function() {  
    return { currentValue: 0 }  
  
  },  
  render: function() {  
    return (<p>Mi valor es { this.state.currentValue }</p>);  
  }  
});
```

# Estado del componente

- Podemos modificar el estado desde dentro del componente llamando a **this.setState(obj)**
- setState **funde** el obj que le mandemos con el estado actual, no lo reemplaza
- Una llamada a setState implica forzar un nuevo render



# Estado del componente

- ¿Para qué usamos el estado?
- Para guardar datos e información, sobre todo la que queremos pasar a componentes hijo
- Con nuestra arquitectura, para casi nada, puesto que tendremos el estado global en el átomo y lo recibiremos vía **props**
- Es útil especialmente con los formularios como veremos más adelante

# Eventos

- Podemos capturar y manejar eventos de UI en los componentes de React
- Se establecen con el atributo `onXXXX` (camelCase) y cuyo valor es una referencia a una función dentro del componente

## Vistas con React

# Eventos

```
var React = require('react');
var MyComp = React.createClass({
  handleClick: function(e) {
    alert("Has hecho click!");
  },
  render: function() {
    return (
      <button onClick={ this.handleClick }>Haz click aquí</button>
    );
  }
});
```

# Eventos

- El manejador del evento recibirá como parámetro un evento sintético, cuyas propiedades y métodos más usados habitualmente son:
  - `DOMEventTarget` **target**  
El elemento del DOM donde se estableció el manejador
  - `void preventDefault()`  
Cancela el comportamiento por defecto del evento
  - `void stopPropagation()`  
Evita que el evento siga ascendiendo siendo capturado por otros elementos

# Eventos disponibles

- Eventos de ratón
  - onClick
  - onDoubleClick
  - onMouseDown / onMouseUp
  - onMouseEnter / onMouseLeave
  - onMouseMove
  - onMouseOver / onMouseOut
  - onWheel

# Eventos disponibles

- Eventos de teclado
  - onKeyDown / onKeyPress / onKeyUp
- Eventos del portapapeles:
  - onCopy / onCut / onPaste
- Eventos de foco
  - onFocus / onBlur
- Eventos de formulario
  - onChange / onInput / onSubmit

# Ejemplo onClick

```
var React = require('react');
var MyComp = React.createClass({
  handleClick: function(e) {
    alert("Has hecho click!");
  },
  render: function() {
    return (
      <button onClick={ this.handleClick }>Haz click aquí</button>
    );
  }
});
```



El argumento **e** contiene el evento sintético

# Comunicación hijo -> padre

- ¿Y si queremos comunicarnos de hijo de padre?
- Hacemos que el componente tenga una API definida
- El padre pasará al hijo funciones como props
- El componente hijo llamará a esos callbacks con datos específicos de su dominio



# Comunicación hijo -> padre

```
var Buttons = React.createClass({
  propTypes: {
    onStart: React.PropTypes.func.isRequired,
    onStop: React.PropTypes.func.isRequired
  },
  render: function() {
    return (
      <div className="actions">
        <button onClick={this.props.onStop}>STOP</button>
        <button onClick={this.props.onStart}>START</button>
      </div>
    );
  }
});
```

# Comunicación hijo -> padre

```
var Parent = React.createClass({
  handleStart: function(e) {
    console.log("Start en Buttons!");
  },
  handleStop: function(e) {
    console.log("Stop en Buttons!");
  },
  render: function() {
    return (
      <div className="parent">
        <Buttons onStart={this.handleStart} onStop={this.handleStop} />
      </div>
    );
  }
});
```

## Ejercicio 3: interacción

- Ya podemos crear por fin nuestro primer componente interactivo usando props, estado interno y eventos
- Vamos a implementar un cronómetro como éste: el botón START inicia el temporizador, y el botón STOP lo detiene en el primer clic y lo reinicia a 0 en el segundo.



## Ejercicio 3: interacción

- Disponéis de la plantilla para este ejercicio en </ejercicios/tema3/plantillas/cronometro.html>
- Tenéis funciones auxiliares para manipular el tiempo con fechas en </ejercicios/tema3/src/lib/utils.js>



## Ejercicio 3: pistas

- Utilizar **composición**: el cronómetro completo debe contener un componente Header, un componente Screen y un componente Buttons.
- Se pueden pasar funciones como **props** de modo que un evento sea “atendido” por el componente padre de quien lo registra y recibe.
- Intentar basar el paso de datos padre-hijo en **props**
- No almacenar información **derivada** en el estado (que pueda ser calculada a partir de props o estado)

# Acceso al DOM

- React gestiona el DOM por nosotros
- Si necesitamos acceder a un nodo montado en el DOM, tenemos que marcarlo en JSX con una referencia:

```
<button ref="miboton">Click me</button>
```

- Después podemos obtener la referencia en código con **this.refs.miboton**. Si React elimina o sustituye ese nodo, actualizará la referencia para nosotros (o será *undefined*)
- Para acceder al DOM nativo y sus propiedades, podemos llamar a **getDOMNode()** sobre la referencia obtenida con **this.refs.X**

# Formularios

- Los controles de formulario HTML son problemáticos para React
- Son inherentemente mutables mediante interacciones de usuario (comportamiento definido por el navegador)

# Formularios

```
var TextInput = React.createClass({
  getInitialState: function() {
    return {
      value: ""
    };
  },
  render: function() {
    return (
      <input type="text" value={ this.state.value }>
    );
  }
});
```



# Formularios

```
var TextInput = React.createClass({  
  getInitialState: function() {  
    return {  
      value: ""  
    };  
  },  
  render: function() {  
    return (  
      <input type="text" value="Introduce tu nombre">  
    );  
  }  
});
```

Si intentamos escribir en esa caja de texto, no pasará nada  
¿Por qué?

# Formularios

```
var TextInput = React.createClass({  
  getInitialState: function() {  
    return {  
      value: ""  
    };  
  },  
  render: function() {  
    return (  
      <input type="text" value="Introduce tu nombre">  
    );  
  }  
});
```

Porque **render** dice que, invariablemente, el valor de ese INPUT es "Introduce tu nombre"

# Formularios

```
var TextInput = React.createClass({  
  getInitialState: function() {  
    return {  
      value: ""  
    };  
  },  
  render: function() {  
    return (  
      <input type="text" value="Introduce tu nombre">  
    );  
  }  
});
```

Si fuera HTML y no React, podríamos borrar ese texto y escribir otro...

# Formularios

- Es un “choque” conceptual con el Virtual DOM de React, que gestiona por nosotros todo el HTML producido
- Tenemos props **específicas** para controles de formularios
- Y un evento muy útil: **onChange**

# Formularios

- value - recupera/establece el valor en:
  - `<input type="text" .../>`
  - `<input type="password" .. />`
  - `<textarea .. />`
  - `<select />` (valor del elemento seleccionado)

# Formularios

- checked - (Boolean) recupera/establece si están activos:
  - `<input type="checkbox" .../>`
  - `<input type="radio" .. />`

# Formularios

- selected - (Boolean) recupera/establece si están seleccionados las opciones de un desplegable:
- `<select>`
  - `<option value="1">Uno</option>`
  - `<option value="2">Dos</option>``</select>`

# Formularios: componentes controlados

- La salida del método **render** define el estado de la UI en cualquier momento determinado

- Si escribimos

```
<textarea value="Introduce tu comentario"></  
textarea>
```

- El usuario **no puede modificar** el contenido. Está “hard-coded” en el código Javascript generado a partir de JSX



# Formularios: componentes controlados

- La solución es utilizar el estado interno del componente como fuente para el **value**
- Implica que tenemos que modificar “manualmente” el estado cada vez que el usuario modifique el control
- onChange funciona en todos los controles

# Formularios: componentes controlados

```
var UserLogin = React.createClass({
  getInitialState: function() {
    return { email: "" };
  },
  onChange: function(e) {
    this.setState( { email: e.target.value } );
  },
  render: function() {
    return (
      <div>
        Email:
        <input type="text" value={ this.state.email }
          onChange={this.onChange} />
      </div>
    );
  }
});
```

# Formularios: componentes controlados

```
var UserLogin = React.createClass({
  getInitialState: function() {
    return { email: "" };
  },
  onChange: function(e) {
    this.setState( { email: e.target.value } );
  },
  render: function() {
    return (
      <div>
        Email:
        <input type="text" value={ this.state.email }
          onChange={this.onChange} />
      </div>
    );
  }
});
```

En **cada cambio** notificado por **onChange**, actualizamos el estado interno...

...y esta actualización ejecutará de nuevo **render** y mostrará el valor correcto

# Formularios: componentes no controlados

- ¿Y si no queremos el control total del formulario?
- Entonces usamos componentes no controlados
- En lugar de **value** definimos **defaultValue** que es sólo el valor por defecto
- Tendríamos que usar **referencias** y acceso al DOM con **getNode** para extraer el valor del campo.

## Ejercicio 4: formularios

- Buscador de personajes de Juego de Tronos

### Buscador Juego de Tronos

Actor / personaje

Familia 

Todas

Sólo personajes vivos☐

Aparece en temporada

1☐ 2☐ 3☐ 4☐ 5☐

Personaje	Actor	Nº Ep	Vivo
Eddard Stark	Calvin Hobbs	45	Sí
Eddard Stark	Calvin Hobbs	45	Sí
Eddard Stark	Calvin Hobbs	45	Sí

Encontrados 25 personajes

## Ejercicio 4: formularios

- Queremos un buscador que actualice los resultados en vivo, según se modifican los parámetros de búsqueda (al estilo **onChange**)
- Los datos en JSON están en `/ejercicios/tema3/src/data/got.js`
- La plantilla en `/ejercicios/tema3/plantillas/buscador.html`
- Utilizaremos **mori** para manipular los datos con las técnicas aprendidas

# Composición

- Una aplicación entera de React se pinta **a partir de un componente raíz**, que a su vez incluye componentes hijos y así sucesivamente
- El componente que incluye otro en su método render es el **dueño** de ese nodo hijo
- El padre puede pasarle props al hijo, configurándolo, y será el responsable del ciclo de vida del hijo
- Cuando no aparezca en su **render**, React desmontará el componente por nosotros

# Composición

- La “manera React” es intentar hacer componentes específicos con el mínimo estado posible
- Recuerda:  $UI = f(datos)$
- Es decir: **render** depende sólo de los **props** y **state** actuales del componente
- Separación de Responsabilidades a nivel de UI
- Cada componente hace una cosa



# Composición

- De esta forma los componentes son cajas negras que “cableamos” mediante sus props.
- Le damos datos via props
- Atendemos sus *notificaciones* pasando una función vía props (ej: `onQueryChange` en el buscador)

# Vistas con React

## Composición

centralLog - Events

Anonymous (Guest)

▼

Go to

MODE  
LIVE

Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
<div>✖</div>	<div>🔊</div>	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
<div>ⓘ</div>	<div>🔊</div>	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
<div>⚠</div>	<div>🔊</div>	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
<div>⚠</div>	<div>🔊</div>	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
<div>✖</div>	<div>🔊</div>	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
<div>✖</div>	<div>🔊</div>	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
<div>ⓘ</div>	<div>🔊</div>	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
<div>✖</div>	<div>🔊</div>	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
<div>⚠</div>	<div>🔊</div>	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
<div>✖</div>	<div>🔊</div>	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
<div>✖</div>	<div>🔊</div>	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
<div>✖</div>	<div>🔊</div>	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
<div>ⓘ</div>	<div>🔊</div>	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
<div>ⓘ</div>	<div>🔊</div>	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
<div>✖</div>	<div>🔊</div>	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
<div>⚠</div>	<div>🔊</div>	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
<div>⚠</div>	<div>🔊</div>	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
<div>⚠</div>	<div>🔊</div>	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
<div>✖</div>	<div>🔊</div>	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
<div>ⓘ</div>	<div>🔊</div>	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
<div>⚠</div>	<div>🔊</div>	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
<div>⚠</div>	<div>🔊</div>	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

# Composición

<body>

Root

EventsLayout

FilterBox

Toolbar

Menu

EventList

EventListHeader

Event

Event

Event

</body>

# Vistas con React

## Composición

centralLog - Events							
Anonymous (Guest)							
Go to		MODE LIVE					
Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
ⓘ	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
ⓘ	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
ⓘ	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
ⓘ	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
ⓘ	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<EventsLayout />

# Vistas con React

## Composición

centralLog - Events

Anonymous (Guest)

Go to

MODE LIVE

Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
🔍	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
🔍	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
🔍	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
🔍	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
🔍	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<FilterBox />

# Vistas con React

# Composición

centralLog - Events

Anonymous (Guest)

Go to

MODE  
LIVE

Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
<div></div>	<div></div>	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
<div></div>	<div></div>	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
<div></div>	<div></div>	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
<div></div>	<div></div>	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
<div></div>	<div></div>	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
<div></div>	<div></div>	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
<div></div>	<div></div>	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
<div></div>	<div></div>	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
<div></div>	<div></div>	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
<div></div>	<div></div>	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
<div></div>	<div></div>	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
<div></div>	<div></div>	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
<div></div>	<div></div>	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
<div></div>	<div></div>	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
<div></div>	<div></div>	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
<div></div>	<div></div>	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
<div></div>	<div></div>	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
<div></div>	<div></div>	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
<div></div>	<div></div>	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
<div></div>	<div></div>	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
<div></div>	<div></div>	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
<div></div>	<div></div>	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<Toolbar />

# Vistas con React

## Composición

centralLog - Events							
<div> <div></div> <div></div> </div>						<div> <div>Anonymous (Guest)</div> <div></div> </div>	
<div> <div></div> <div></div> </div>						<div> <div>Go to</div> <div></div> </div>	
Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
ⓘ	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
ⓘ	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
ⓘ	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
ⓘ	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
ⓘ	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<Menu visible={false} />



# Vistas con React

## Composición

centralLog - Events

Anonymous (Guest)

Go to

MODE LIVE

Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
ⓘ	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
ⓘ	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
ⓘ	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
ⓘ	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
ⓘ	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<EventList />



# Vistas con React

## Composición

centralLog - Events							
<div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div>Go to</div> <div>MODE LIVE</div> </div>							
Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
ⓘ	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
ⓘ	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
ⓘ	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
ⓘ	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
ⓘ	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<EventListHeader />

# Vistas con React

## Composición

centralLog - Events							
Anonymous (Guest)							
Go to		MODE LIVE					
Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
ⓘ	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
ⓘ	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
ⓘ	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
ⓘ	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
ⓘ	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<Event severity="error" state="acked" />

# Vistas con React

## Composición

centralLog - Events							
Anonymous (Guest)							
Go to		MODE LIVE					
Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
🔊	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
🔊	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
🔊	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
🔊	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
🔊	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<Event severity="info" state="acked" />

# Vistas con React

## Composición

centralLog - Events							
Anonymous (Guest)							
Go to		MODE LIVE					
Sev	Ack	Time	Site	Module	Domain	Type	Message
04/08/2015							
✖	🔊	04/08/2015 09:53:05.000	Ryadh	FDS	BBDB5	R	test1_1
ⓘ	🔊	04/08/2015 09:53:05.000	Madrid	Fleet	BBDB5	N	test2_2
⚠	🔊	04/08/2015 09:53:04.000	Ryadh	Fleet	AB4FB	N	test1_1
⚠	🔊	04/08/2015 09:53:04.000	Madrid	Fleet	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	test1_1
✖	🔊	04/08/2015 09:53:03.000	Ryadh	Fleet	BBDB5	N	error enormous
ⓘ	🔊	04/08/2015 09:53:03.000	Ryadh	FDS	AM01M	N	test1_1
✖	🔊	04/08/2015 09:53:01.000	Ryadh	FDS	AM01M	R	test1_1
⚠	🔊	04/08/2015 09:53:01.000	Madrid	RTS2	AB5C	A	error enormous
✖	🔊	04/08/2015 09:53:00.000	Ryadh	Fleet	AM01M	R	test1_1
✖	🔊	04/08/2015 09:53:00.000	Madrid	Fleet	AM01M	A	test2_2
✖	🔊	04/08/2015 09:52:59.000	Madrid	FDS	AB4FB	A	error enormous
ⓘ	🔊	04/08/2015 09:52:59.000	Ryadh	Fleet	AM1AC	N	test1_1
ⓘ	🔊	04/08/2015 09:52:58.000	Ryadh	FDS	AB5C	A	test1_1
✖	🔊	04/08/2015 09:52:58.000	Madrid	FDS	AB4FB	M	error enormous
⚠	🔊	04/08/2015 09:52:58.000	Ryadh	Fleet	AB4FB	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	FDS	AM01M	N	error enormous
⚠	🔊	04/08/2015 09:52:56.000	Madrid	RTS2	AM1AC	N	test1_1
✖	🔊	04/08/2015 09:52:55.000	Madrid	Fleet	AB5C	N	test1_1
ⓘ	🔊	04/08/2015 09:52:54.000	Ryadh	RTS1	AB4FB	M	test1_1
⚠	🔊	04/08/2015 09:52:52.000	Ryadh	RTS2	AM1AC	R	test2_2
⚠	🔊	04/08/2015 07:53:03.000	Ryadh	RTS1	AB5C	N	test2_2

<Event severity="warning" state="active" />

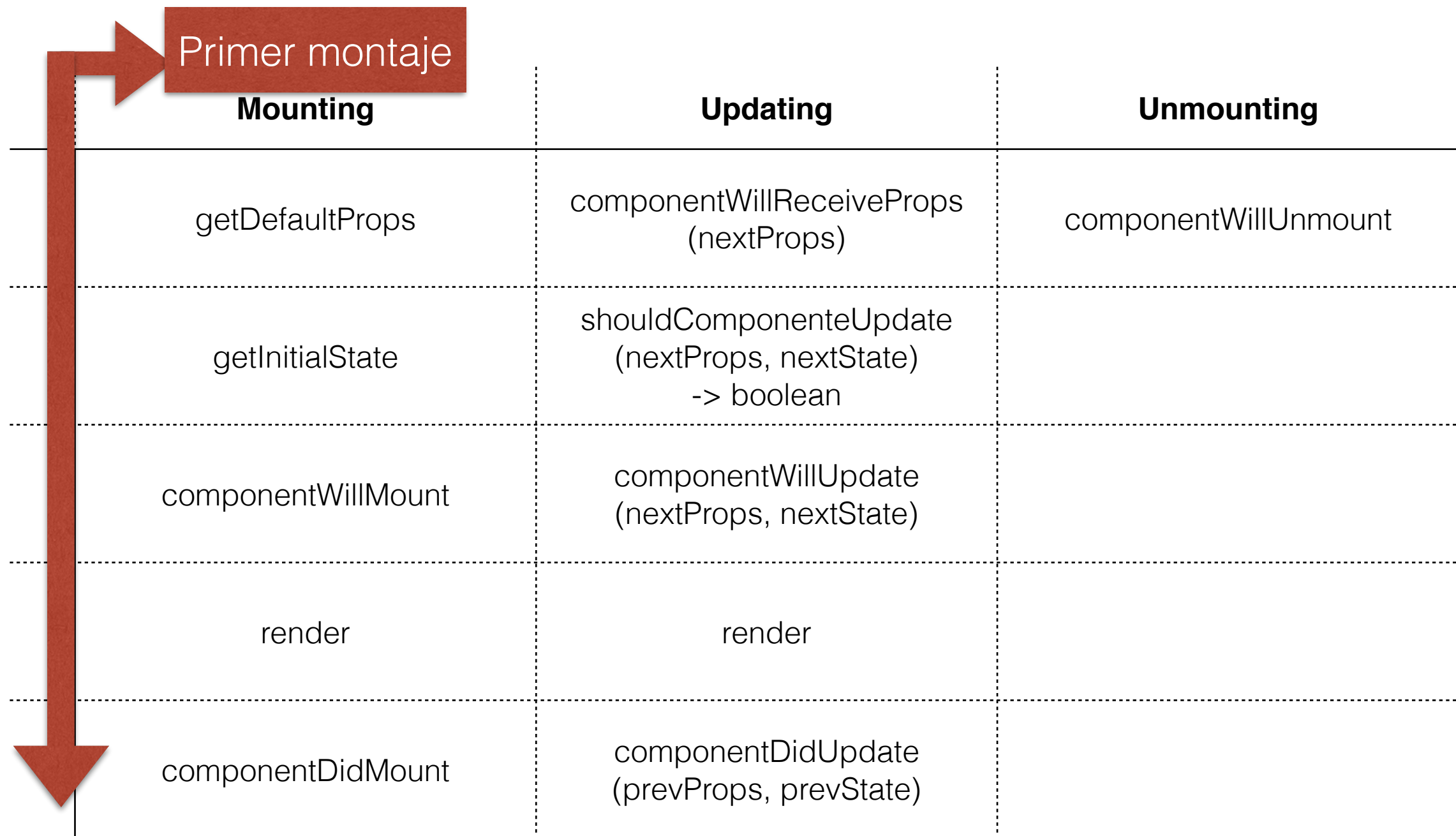
# Ciclo de vida de un componente

- Un componente React tiene una serie de métodos que React llama en un orden predefinido
- Ya hemos utilizado **getInitialState** y **getDefaultProps**
- El único obligatorio es **render** ya que sin ese método el componente no genera UI alguna
- Existen 3 momentos en el ciclo de un componente: mounting, updating, unmounting (creación, actualización, destrucción)

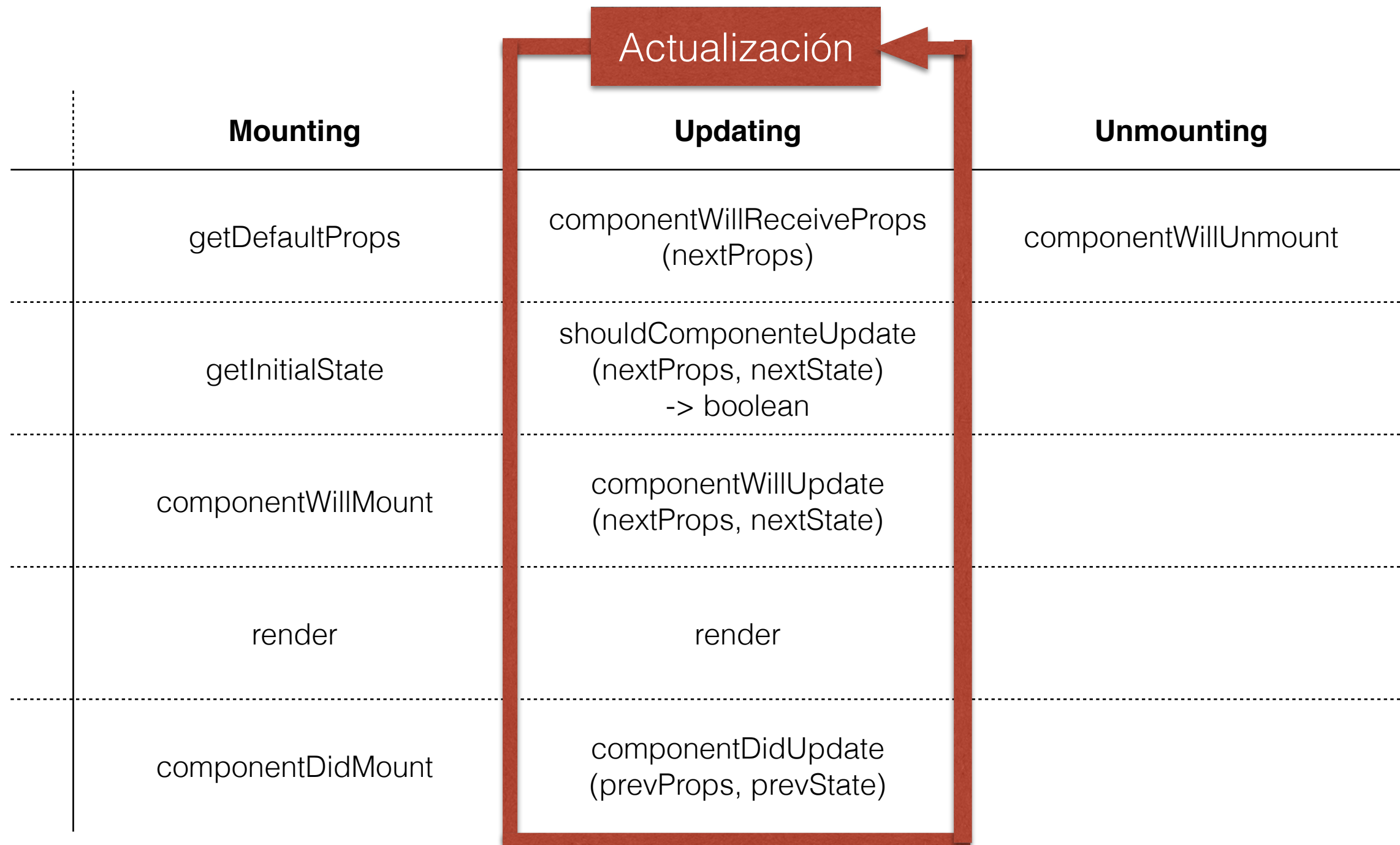
# Ciclo de vida de un componente

	Mounting	Updating	Unmounting
	getDefaultProps	componentWillReceiveProps (nextProps)	componentWillUnmount
	getInitialState	shouldComponentUpdate (nextProps, nextState) -> boolean	
	componentWillMount	componentWillUpdate (nextProps, nextState)	
	render	render	
	componentDidMount	componentDidUpdate (prevProps, prevState)	

# Ciclo de vida de un componente



# Ciclo de vida de un componente





# Ciclo de vida de un componente

	Mounting	Updating	<div>Destrucción</div> Unmounting
	getDefaultProps	componentWillReceiveProps (nextProps)	componentWillUnmount
	getInitialState	shouldComponentUpdate (nextProps, nextState) -> boolean	
	componentWillMount	componentWillUpdate (nextProps, nextState)	
	render	render	
	componentDidMount	componentDidUpdate (prevProps, prevState)	

# Ciclo de vida de un componente

- Ejemplo para ver los ciclos de vida:  
`ejercicios/tema3/components/ejercicio5/ciclovida.js`
- El flujo en React siempre es unidireccional como se ve en la tabla anterior, en la etapa de actualización
- Dentro de **render** no modificamos ni propiedades ni estado, lo hacemos en los manejadores de eventos
- **render** es una función pura: dadas las mismas props y mismo estado devuelve exactamente lo mismo

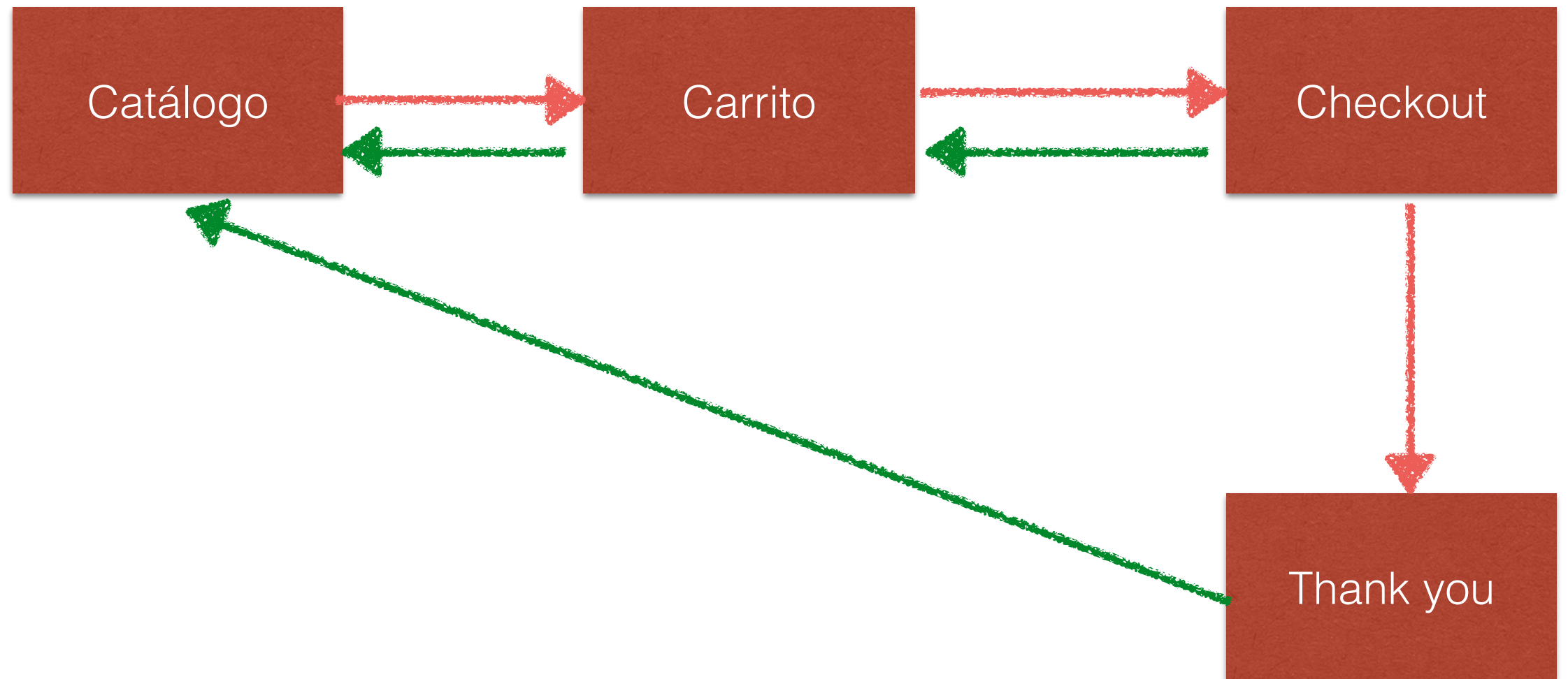
# Optimización

- Los métodos del ciclo de vida nos permiten influir en el mismo o conocer el momento actual de nuestro componente (¿estoy ya visible? ¿me he actualizado?)
- **shouldComponentUpdate** es el método con el que podemos *cancelar* una llamada a render devolviendo *false*
- Tenemos acceso a las próximas *props* y próximo *state* por lo que podemos decidir que no necesitamos otro render
- Útil sobre todo cuando utilizamos el átomo global, puesto que cualquier cambio dispara el re-**render** de **toda** la aplicación

# Ejercicio - Ecommerce

- Vamos a implementar una micro tienda que contiene:
  - Catálogo - se muestran los productos y se pueden añadir al carrito
  - Carrito - se muestran los productos escogidos, se manipula su cantidad y se vuelve al catálogo o se va al checkout
  - Checkout - se piden datos del usuario, se **validan** y, si es correcto, se va a la página de gracias
  - Confirmación - se muestra un mensaje de confirmación y se puede volver al Catálogo.

# Ejercicio - Ecommerce



## Ejercicio - Ecommerce

- Tendremos que mostrar la pantalla adecuada según el estado de nuestra tienda
- Podemos mantener una clave **page** en el estado del componente padre
- La modificamos con **setState({ page: XXX })** cuando queramos navegar entre páginas
- La utilizamos para decidir qué componente pintar

# Ejercicio - Ecommerce

```
render: function() {  
  return (  
    <div className="shopping-cart">  
      { this.getPageComponent(this.state.page) }  
    </div>  
  );  
}
```

# Ejercicio - Ecommerce

```
getPageComponent: function (page) {  
  switch (page) {  
    case 'catalog':  
      return <Catalog  
        products={this.state.catalog}  
        onProductAdd={this.addProductToCart} />;  
    case 'cart':  
      return <Cart  
        products={this.state.cart}  
        onNavigate={this.setPage}  
        onItemQtyChange={this.changeCartItemQuantity}  
        onItemRemove={this.removeCartItem} />  
    case 'checkout':  
      return <Checkout onNavigate={this.setPage} onOrderPlaced={this.completeCheckout} />;  
    case 'thank-you':  
      return <ThankYou onNavigate={this.setPage} order={this.state.customerDetails} />;  
  }  
},
```



# Integración en nuestra arquitectura

- Lo primero que necesitamos es integrar el átomo
- Queremos que cuando cambie, se repinte la aplicación
- Y queremos pasar la referencia de padres a hijos

# Integración en nuestra arquitectura

```
var React = require('react'),
    atom = require('../lib/atom_state');

var ShoppingCart = require('../shopping_cart/');

var RootComponent = React.createClass({
  componentDidMount: function() {
    atom.addChangeListener(this._onAtomChange);
  },
  _onAtomChange: function() {
    this.forceUpdate();
  },
  render: function() {
    var state = atom.getState();
    return (<ShoppingCart state={state} />);
  }
});

module.exports = RootComponent;
```

# Integración en nuestra arquitectura

```
var React = require('react'),  
    atom = require('../lib/atom_state');
```

```
var ShoppingCart = require('../shopping_cart/');
```

```
➔ var RootComponent = React.createClass({  
  componentDidMount: function() {  
    atom.addChangeListener(this._onAtomChange);  
  },  
  _onAtomChange: function() {  
    this.forceUpdate();  
  },  
  render: function() {  
    var state = atom.getState();  
    return (<ShoppingCart state={state} />);  
  }  
});
```

```
➔ module.exports = RootComponent;
```

Este RootComponent es el que montaremos en document.body

# Integración en nuestra arquitectura

```

var React = require('react'),
    atom = require('../lib/atom_state');

var ShoppingCart = require('../shopping_cart/');

var RootComponent = React.createClass({
  componentDidMount: function() {
    ➔ atom.addChangeListener(this._onAtomChange);
  },
  // ...
});

module.exports = RootComponent;

```

Cuando se haya montado el componente, se suscribe a los cambios del átomo

```

;
={state} />);

```

# Integración en nuestra arquitectura

```
var React = require('react'),
    atom = require('../lib/atom_state');

var ShoppingCart = require('../shopping_cart/');

var RootComponent = React.createClass({
  componentDidMount: function() {
    atom.addChangeListener(this._onAtomChange);
  },
  → _onAtomChange: function() {
    this.forceUpdate();
  },
});
```

this.forceUpdate() es un método de React que **fuerza** el re-render

# Integración en nuestra arquitectura

En **render**, traemos el valor actual del átomo y lo pasamos vía props a nuestra jerarquía de componentes.

Nuestra convención es llamar **state** a la prop que contiene el valor actual apuntado por el átomo

De esta forma, con cada cambio del átomo le pasamos un nuevo **state** vía props al primer componente en nuestra jerarquía

```
render: function() {  
  var state = atom.getState();  
  ➡ return (<ShoppingCart state={state} />);  
}
```

```
module.exports = RootComponent;
```

# Integración en nuestra arquitectura

- Lo siguiente será incluir Stores y Dispatcher
- Observamos que con una mínima complejidad, el código de nuestros componentes empieza a estar un poco “sucio”:
  - Manipulación de datos
  - Comprobación de estado y lógica de negocio repartida entre los componentes
  - Muy poca reutilización de código

# Integración en nuestra arquitectura

- Lo solucionaremos cuando integremos React en nuestra arquitectura completa
- Consulta de datos: **a los Stores**. Eliminaremos mucho cableado entre componentes y casi todas las operaciones de mori para manipular datos
- Interacciones con el estado global: **vía Dispatcher**. Eliminaremos casi todos los callbacks pasados de padres a hijos, y por tanto el estado interno de los componentes (salvo los formularios).



# Integración en nuestra arquitectura

El objetivo es que toda la jerarquía de React dependa exclusivamente del contenido en el átomo en un momento dado

$$\mathbf{UI = f(atom)}$$