

Contenido

- 2. Construyendo componentes React
 - Conceptos clave de React
 - JSX
 - Propiedades y estado de un componente
 - Ciclo de vida de un componente
 - Eventos
 - Formularios
 - Composición
 - Integrar con Stores y Dispatcher
 - Ejercicio

React - conceptos clave

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

React - conceptos clave

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

UI = f(datos+estado)

React - conceptos clave

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

React - Conceptos clave

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

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

module.exports = Saludo;
```

React- Conceptos clave

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

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

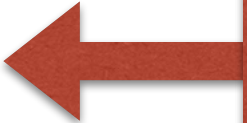
module.exports = Saludo;
```

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

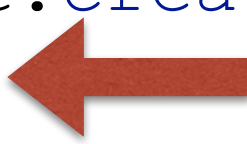
```
var HolaMundo = React.createClass ({  
  render: function () {  
    return (  
      <div className="panel">Hola mundo!</div>  
    );  
  }  
});
```



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**, tendremos un error en la consola

JSX

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

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

```
ReactDOM.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 del tema anterior para tener listo el servidor de desarrollo, empaquetado y compilación

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 React

⚠ 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,...)
 - Arrays y objetos complejos
 - Funciones

Propiedades de un componente

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

var EjemploProps = React.createClass({
  myFunction: function() {
    alert("Boo!");
  },
  render: function() {
    var obj = { foo: 'bar' };
    return (
      <div>
        <OtroComponente
          text="hello"
          number={ 6 }
          thing={ obj }
          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, pasándole como **props** el país y una fecha ajustada a la diferencia horaria

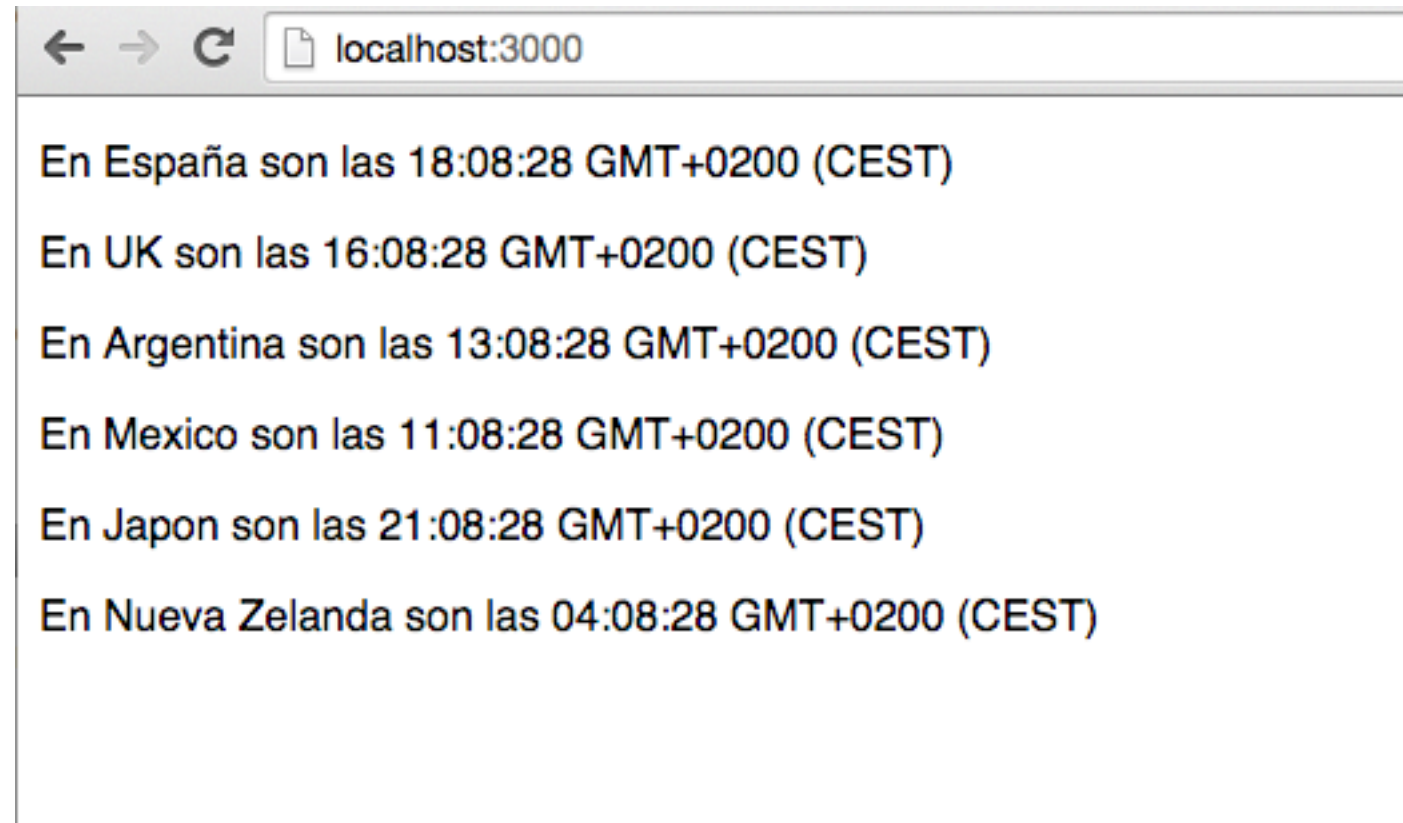
```
    var componentes = zonasHorarias.map(function(zona) {
      return <Fecha key={ zona.country } country={ zona.country }
        date={ this.convertirZonaHoraria(ahora, zona.difUTC) } />;
    }, this);
```

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

```
module.exports = FechasMundo;
```

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 en su método **render**
- 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: es la manera en que un componente fuerza su render desde dentro

Estado del componente

- ¿Para qué usamos el estado?
- Para guardar datos y estado del componente, sobre todo la que queramos pasar a componentes hijo en **render**
- 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

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

Ejercicio: eventos y state

- Vamos a trastear con eventos y con estado interno del componente
- Vamos a hacer un componente con un botón y un contador de clicks