



REACT.JS

© 2017, ACTIBYTI PROJECT SLU, Barcelona
Autor: Ricardo Ahumada



MINISTERIO
DE ENERGÍA, TURISMO
Y AGENDA DIGITAL

red.es



ESTRATEGIA DE
EMPRENDIMIENTO Y
EMPLEO JUVENIL
garantía juvenil



UNIÓN EUROPEA

Fondo Social Europeo
“El FSE invierte en tu futuro”

Índice de contenidos

1. Introducción
2. JSX
3. Eventos
4. Gestión de formularios
5. Enrutado
6. Acceso a APIs de Servidor
7. Constructores y HOCs

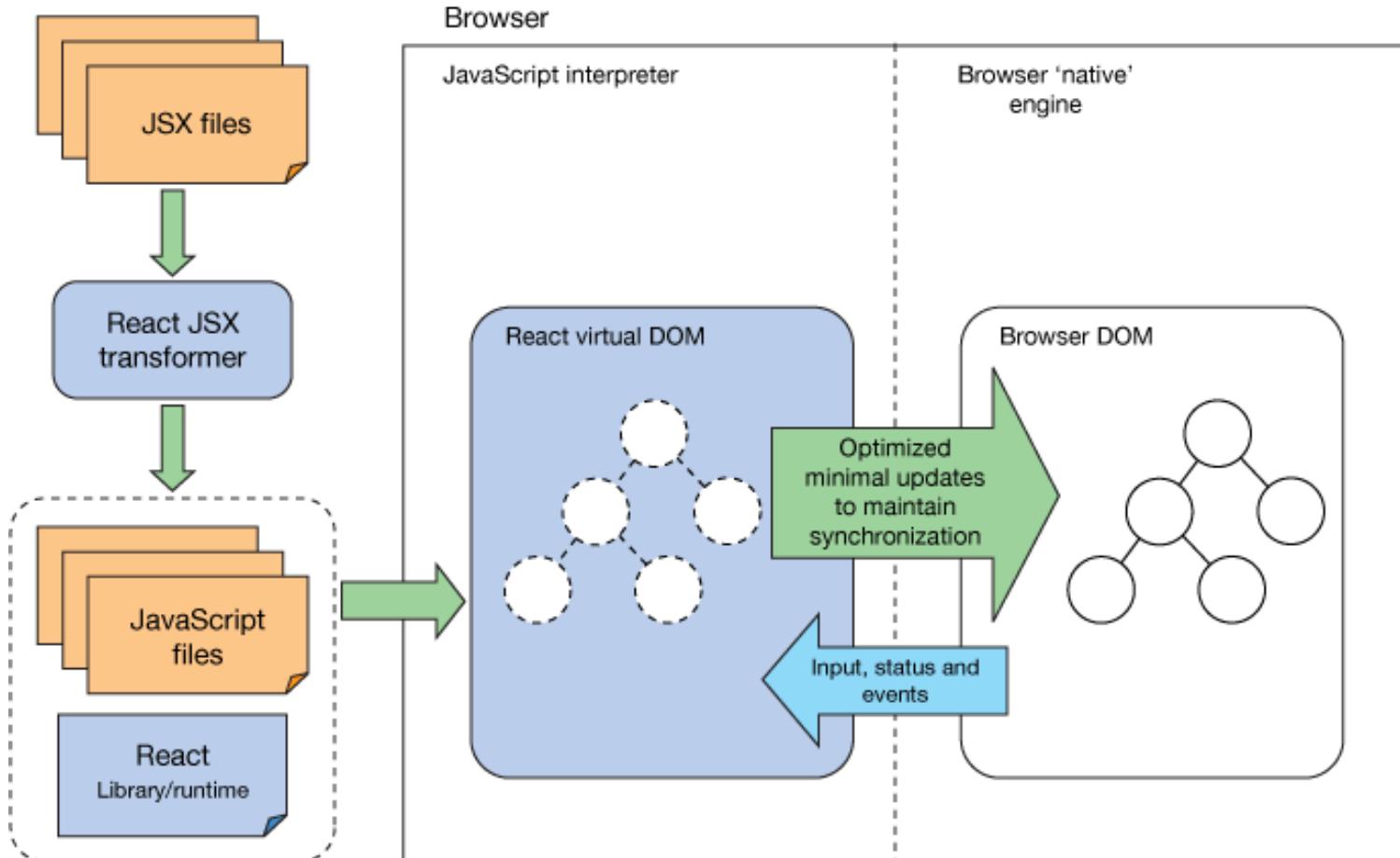
1

INTRODUCCIÓN A REACT.JS

React

- React es una librería de código abierto para construir interfaces de usuario y que ofrece grandes beneficios en rendimiento y modularidad.
- Ha sido desarrollado por Facebook.
- React promueve un flujo muy claro de datos y eventos, facilitando la planificación y desarrollo de apps complejas.
- Se encarga sólo de la parte de la vista de una aplicación, pero es compatible con otras librerías que complementen los patrones Flux o MVC.
- El secreto de su performance alto es la implementación del Virtual DOM que permite renderizaciones parciales en el DOM.

Arquitectura



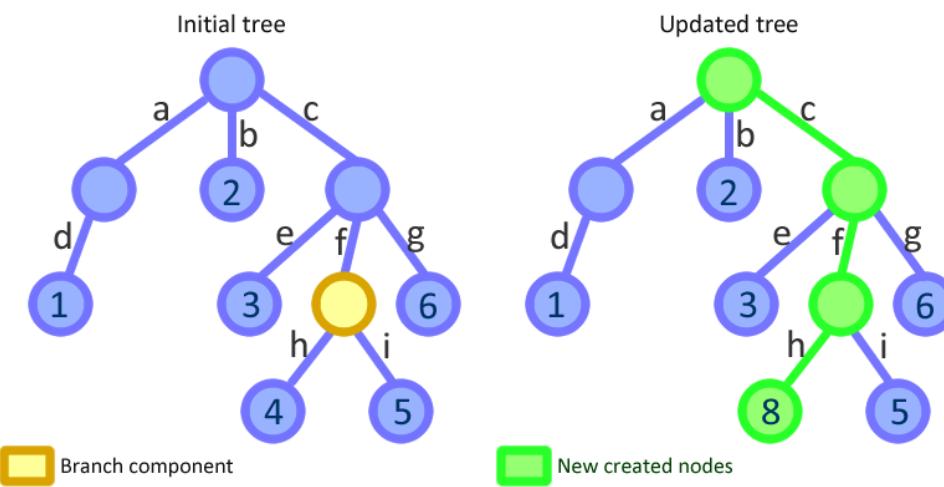
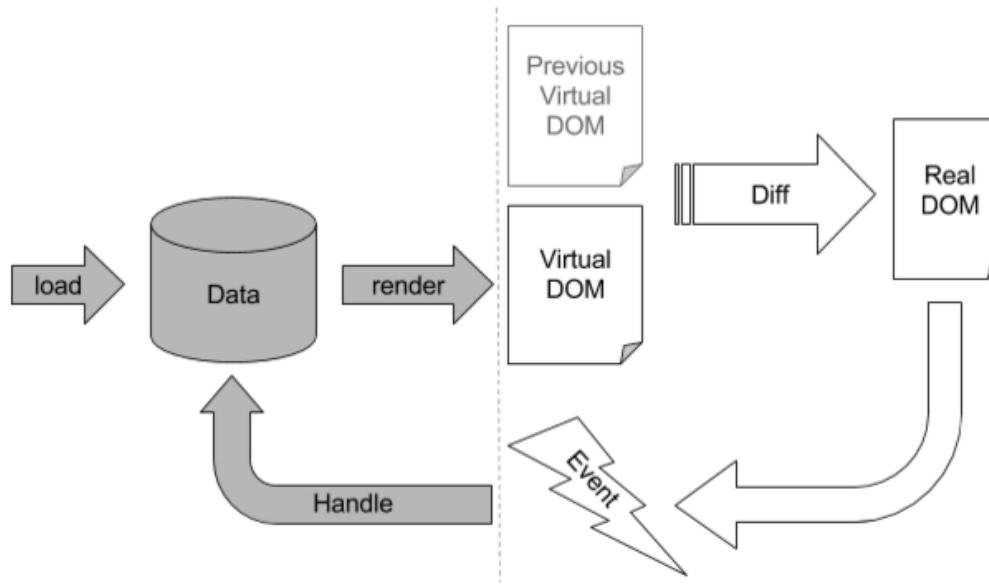
Arquitectura

- **JSX**: Es una sintaxis tipo XML cuyo objetivo es facilitar la generación de HTML dentro de los scripts. Para ello necesita de procesadores (transpiladores) que transformen la sintaxis en ECMAScript estándar
- **JSX Transformer**: se encarga de transpilar el JSX a ECMAScript. A partir de la versión 14 ha sido reemplazado por Babel.
- **Intérprete**: se encarga de interpretar el javascript

React y el Virtual DOM

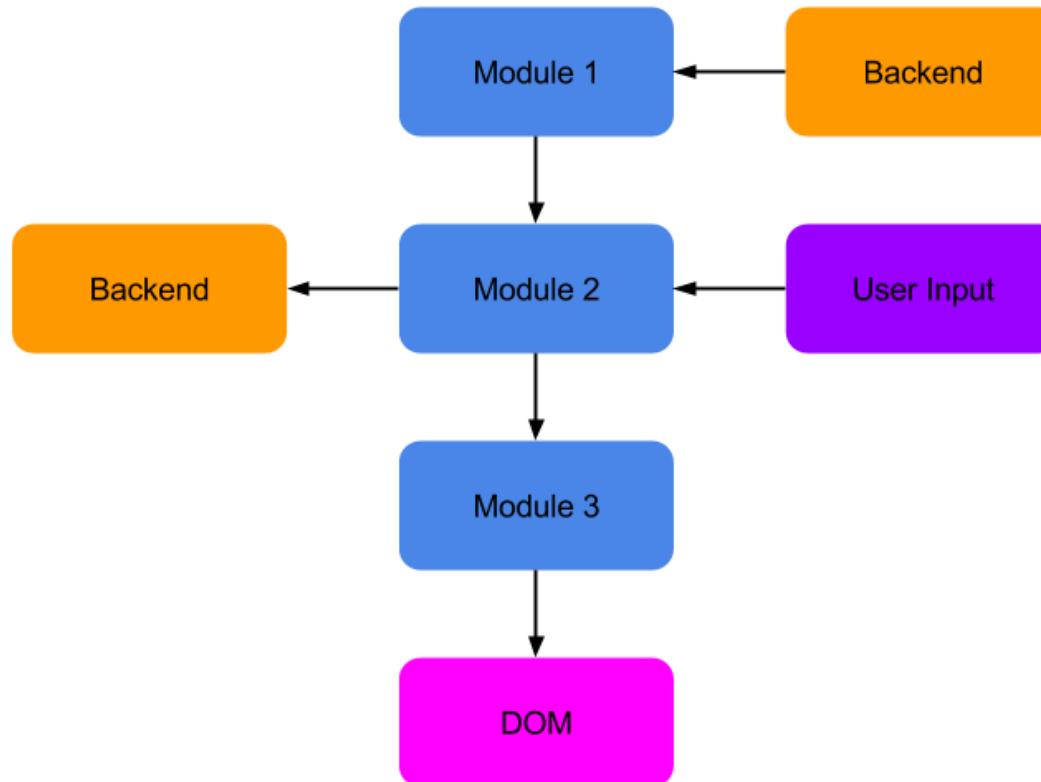
- Una de las decisiones clave del diseño en React es hacer parecer que la API vuelve a renderizar toda la aplicación en cada actualización.
- Manipular el DOM es una tarea lenta, por lo que para que esto sea posible de una manera eficiente, React implementa un DOM virtual.
- En lugar de actualizar el DOM cada vez que hay cambios de estado de la aplicación, React simplemente crea un árbol virtual que se ve como el estado DOM que desea:
 - Compara los dos árboles y si hay cambios reemplaza todo el subárbol correspondiente.
 - Lo mismo hace con componentes personalizados
- El proceso es muy eficiente y rápido

React y el Virtual DOM



El flujo de datos

- React promueve el flujo de datos en un solo sentido, lo que hace más fácil la planificación y detección de errores, sobre todo en entornos complejos.



Ecosistema React

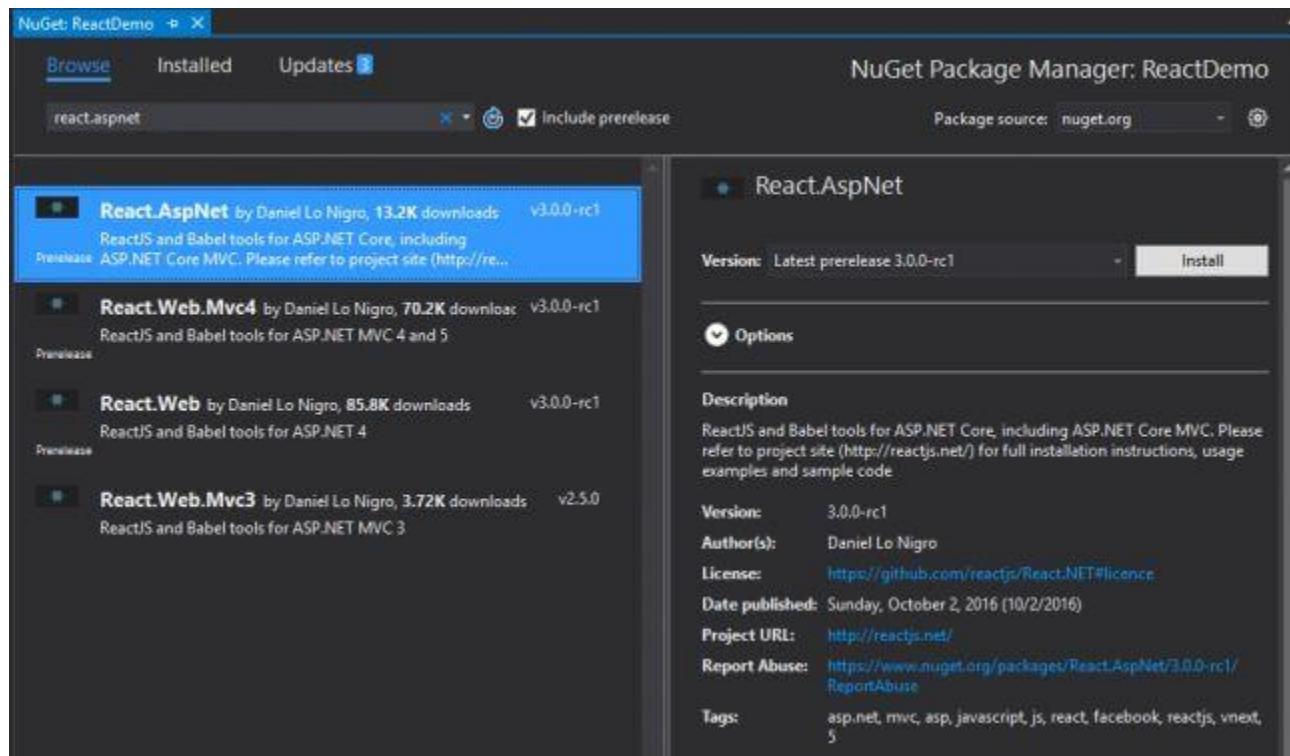
- React requiere un conjunto de herramientas y librerías para su desarrollo.
- Algo que puede ser un poco intimidante al inicio, pero que finalmente nos serán de gran ayuda y nos permitirán obtener lo mejor del mundo ES6 y JSX:
- **Node.js**: Es un entorno de ejecución JS en el lado del servidor. Está basado en el motor Chrome V8.
- **npm**: es un gestor de paquetes javascript de Node.js.
- **Webpack**: es un módulo que permite empaquetar módulos Javascript y generar activos estáticos (bundle.js).
- **Babel**: Es un compilador que nos permite traer ES6 a nuestro desarrollo y hacerlo compatible con los navegadores.
- **Lite-server**: Módulo de Node que nos permite tener un servidor web ligero.

IDE

- React es una librería Javascript, por tanto cualquier IDE de desarrollo adaptada para desarrollo web.
- Especialmente destacan estos 3 IDEs por sus prestaciones
 - Visual Studio
 - SublimeText
 - WebStorm

IDE: Visual Studio

➤ Complemento para React en Visual Studio: ReactJS.NET



IDE: Visual Studio

- Otros complementos importantes de V. Studio:
 - Web Essentials (Edición y gestión de versiones)
 - Jasmine (Test unitarios)

IDE: Sublime Text

➤ Instalación de ***Sublime Text***

- Dos versiones: 2.0 y 3.0 (32/64 bits)
 - Se distinguen por el número de "Build" (actualmente: 2083)
- Evaluación gratuita sin limitación de funcionalidad ni tiempo.
- La única diferencia es que -después de grabar varias veces un archivo- muestra un pequeño mensaje en pantalla.
- Desde un principio, se dispone de la posibilidad de editar código en un montón de lenguajes: HTML, CSS, JavaScript, PHP, C, C++, C#, Java, Objective-C, etc.
- En el sitio Web Package Control (<https://sublime.wbond.net/>) dispone de docenas de extensiones para este editor.
- Igualmente, desde la opción "*Preferences/Package Control*" aparecerá una ventana especial donde se ofrecen diversas opciones para el manejo de paquetes

IDE: Sublime Text

- Complementos de ***Sublime Text***
 - ReactJS: Snippet para sintaxis React
 - JSX Hint: Snippet para JSX
 - Emmet: Utilidad poderosa para completar código HTML
- Otros complementos:
 - Babel
 - Babel Snippets

ReactJS

Sublime Text helpers for React. Syntax highlighting DEP...babel-sublime
v2015.05.11.19.44.00; github.com/facebookarchive/sublime-react

SublimeLinter-jsxhint

SublimeLinter 3 plugin for JSX (React.js), using jsxhint.
install v1.1.1; github.com/SublimeLinter/SublimeLinter-jsxhint

Emmet

Emmet for Sublime Text
v2016.08.10.08.24.48; emmet.io

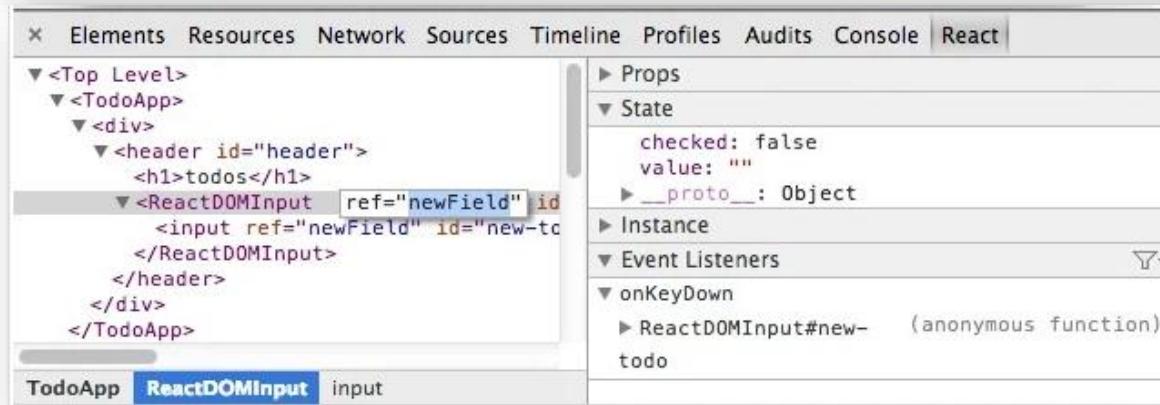
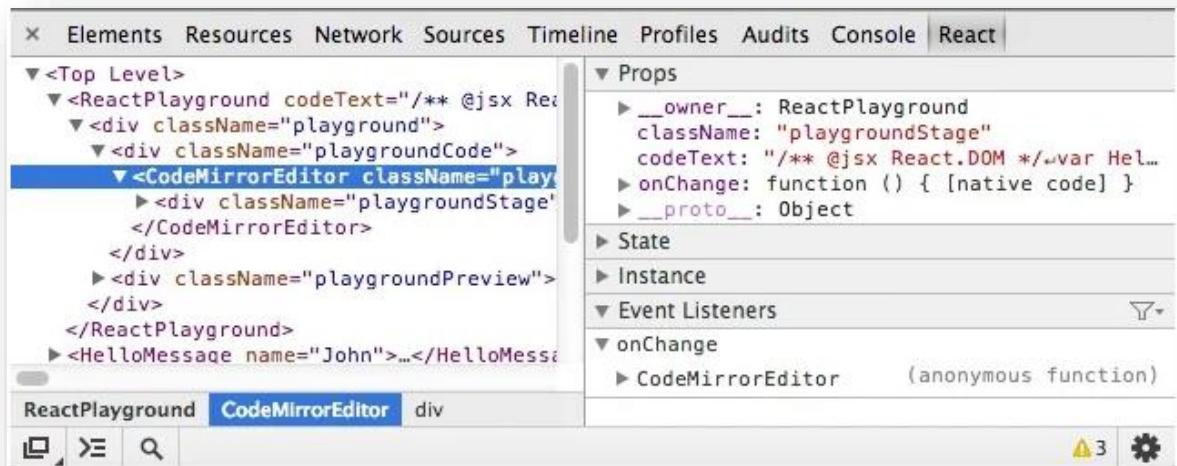
IDE: WebStorm

- IDE javascript más completa preparado para distintos frameworks JS
 - Angular, React, JSX, NodeJS. ...
- En un IDE de pago con un periodo de prueba gratuito de 30 días

Herramientas: Depuración

› *React Developers tools:*

- › Es una extensión para Chrome para el desarrollo de React.





Pongámoslo en práctica: Ecosistema React

- Node.js
- Webpack
- Babel

- SublimeText
- Add-ons de Sublime

- React Developer tools



Instala Node y sus plugins

➤ En tu ordenador

- › Crea un directorio para trabajar con la siguiente estructura
 - › Installs
 - › workspace

➤ Instalar Node.js

- › Descargar e instalar la versión 6+ de Node: <https://nodejs.org/en/>
- › Comprobar que está correctamente instalado abriendo una ventana de consola y escribiendo: node -v

➤ Instalar Webpack

- › En la ventana de consola: npm install -g webpack

➤ Instalar Babel

- › npm i babel-loader babel-preset-es2015 babel-preset-react -S



Instala el IDE

➤ Instalar Sublime Text

- › Descargar la versión portable de sublime de:
<http://www.sublimetext.com/3>
- › Descromprime el zip de sublime y ejecuta *sublime_text.exe*

➤ Instalar Package Control

- › Accede a: View > console
- › Copia y pega el código de: <https://packagecontrol.io/installation>

➤ Instalar Ad-Ons

- › Accede a: Preferences > Package control
- › Escoge “Install Package”
- › Instala cada uno de estos snippets (escribiendo su nombre):
ReactJS, JSX Hint, Emmet



Instala los plugins del Navegador

➤ Instalar React Developers tools

- Abre Chrome y accede a las extensiones
- Busca “react developers tools”
- Instala la extensión



La plantilla de desarrollo



Instala la plantilla

➤ La plantilla de desarrollo

- Para facilitar el desarrollo con React se usan diversas plantillas de desarrollo. En este curso vamos a usar una sencilla que nos permitirá tener una base para comenzar a desarrollar nuestras aplicaciones React

➤ Descomprime la plantilla

- En Lab1.2 encontrarás la plantilla base_react.zip
- Descomprime la plantilla en tu directorio de workspace
- Dale una nuevo nombre al directorio (ej. Lab1_2)

➤ Inspecciona

- Inicia SublimeText
- Open > folder > [ruta de la carpeta del laboratorio]
- Revisa la estructura

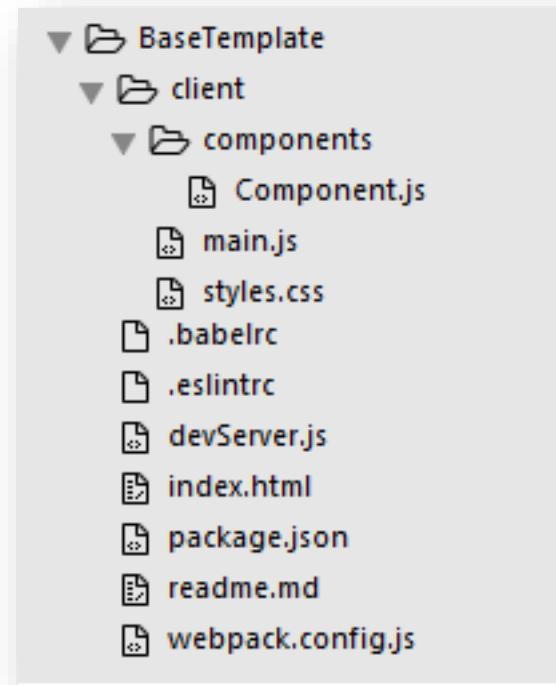


Estructura de la plantilla

➤ Estructura de la plantilla

- index.html: archivo de entrada de la aplicación
- client: directorio donde residirá nuestra aplicación

- package.json: configuración node y de dependencias
- webpack.config.js: configuración para empaquetado
- .babel.rc: configuración de babel
- .eslintrc: configuración para ESLint (validación JSX)
- devServ.js: configuración para node-webpack para compilación JIT





Las dependencias y el servidor

➤ Instalar las dependencias

- › Abre una ventana de consola en la ruta de la carpeta de tu workspace (Lab1_2)
- › Escribe npm install
- › Esto instalará las dependencias del proyecto

➤ Inicializa el servidor

- › npm start
- › Esto inicializará el servidor en el puerto: 7770
- › Accede en tu navegador en la dirección: <http://localhost:7770/>

➤ Para el servidor

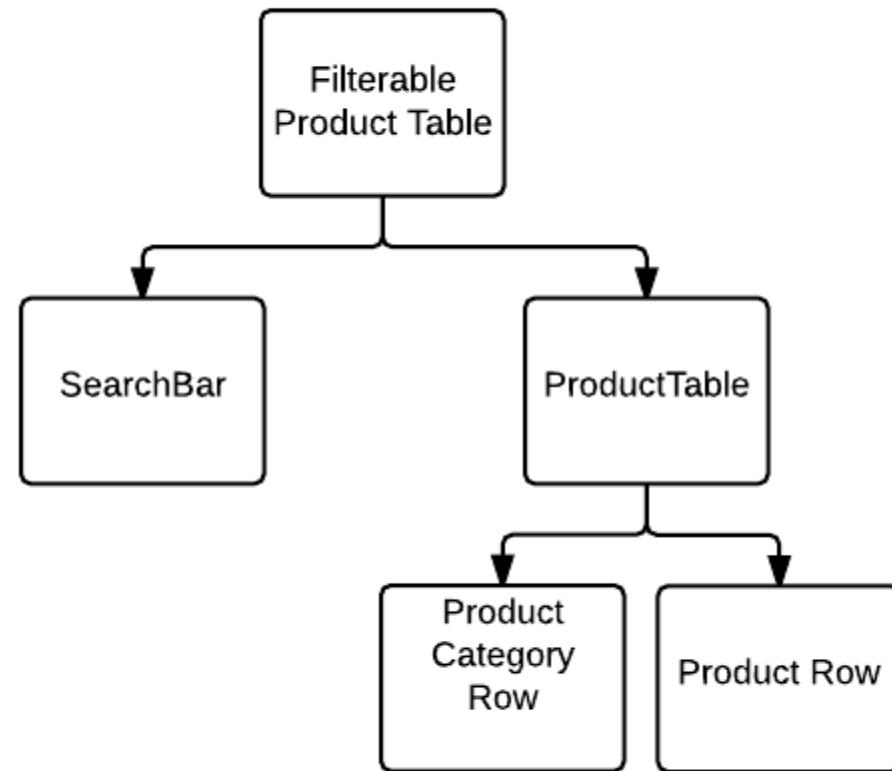
- › En la venatan de consola: Ctrl+c
- › Escribe “s” y pulsa “enter”

El concepto de componente

- Los componentes permiten separar la interfaz de usuario en piezas independientes, reutilizables, y pensar en cada pieza de forma aislada.
- Conceptualmente, los componentes son como funciones javascript, aceptan entradas arbitrarias (llamados "props") y devuelven elementos React que describen lo que debe aparecer en la pantalla.

El concepto de componente (II)

Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99



Creando componentes

- Definir un componente en React es muy sencillo:
 - Primero es necesario importar las dependencias y componentes de los que depende.
 - Luego se define una nueva clase que representa al componente
 - Finalmente se exporta el componente para que se pueda usar desde fuera

```
import React from 'react';

const Component = React.createClass({
  ....
})

export default Component;
```

Creando componentes

- La parte del componente que se encarga de renderizar el HTML en el navegador es el método “render()”
- Definiremos dentro de él los elementos HTML que queramos que aparezcan en el navegador.

```
import React from 'react';

const Component = React.createClass({
  render(){
    return (
      <div>
        <h1>Hola React!</h1>
      </div>
    )
  }
})

export default Component;
```

Carga inicial

- Para generar el renderizado es necesario indicar el componente principal a renderizar y el nodo raíz del DOM donde se renderizará.

```
import React from 'react';

const Component = React.createClass({
  render(){
    return (
      <div>
        <h1>Hola React!</h1>
      </div>
    )
  }
}

export default Component;
```

```
import Component from './components/Component.jsx';

render(<Component />,document.getElementById('root'));
```

```
<div id="root">
  Si puedes ver esto, React no está funcionando.
</div>
```

El concepto de JSX

- ReactJS nos ofrece un pseudo-lenguaje llamado JSX que facilita el desarrollo de aplicaciones web con su sintaxis para crear elementos en el DOM.
- JSX añade un paso de preprocesador que añade sintaxis XML para javascript.
- Al igual que en XML, las etiquetas JSX tienen una etiqueta con su nombre, atributos e hijos.
- Si un valor de atributo está encerrado entre comillas, el valor es un string.
- Si está entre llaves se considerará una expresión de JavaScript.
- React es un pseudo-lenguaje XML, por tanto debe estar bien formado y tener sólo un nodo raíz.

Ejemplos JSX

```
<div>
  <h1>Hola React!</h1>
</div>
```

```
<div>
  <h1>Sumar 2+4 = {2+4}</h1>
</div>
```

```
<div className="UserList">
  <table className="table table-striped">
    <thead>
      <tr><th>Id</th><th>Nombre</th><th>Email</th></tr>
    </thead>
    <tbody>
      {userNodes}
    </tbody>
  </table>
</div>
```



Pongámoslo en práctica

- Crear una App simple con un componente que muestre:
 - Un título: Tabla del 9
 - Una lista: correspondiente a la tabla del 9

Componentes y anidamiento

- Los componentes pueden ser anidados para componer la estructura de la vista.
- De hecho React promueve la generación de componentes reutilizables para componer el HTML.
- El componente que anidan se llaman componente “padre” y los componentes que son anidados se llaman componentes “hijos”

- Para anidar componentes, desde el componente padre se debe:
 - Importar los componentes hijos que quiere referenciar
 - Incluir el tag del componente hijo

Ejemplo de anidamiento

```
const CommentBox = React.createClass({  
  render: function() {  
    return (  
      <div className="container">  
        <h1>Comentarios</h1>  
        <CommentList />  
        <CommentForm />  
      </div>  
    );  
  }  
});
```

```
const CommentList = React.createClass({  
  render: function() {  
    return (  
      <div className="commentList">  
        <h2>Este es el componente CommentList</h2>  
      </div>  
    );  
  }  
});
```

```
const CommentForm = React.createClass({  
  render: function() {  
    return (  
      <div className="commentForm">  
        <h2>Este es el componente commentForm</h2>  
      </div>  
    );  
  }  
});
```

Manejo de propiedades

- Las propiedades de un componente se pueden definir como los atributos de configuración para dicho componente.
- Se les denomina como “props”
- Son recibidos desde un nivel superior (normalmente al instanciar el componente) y por definición son inmutables, es decir, un componente no puede cambiar sus propias props.

```
render(<Lista_Multiplicar num="8" init="1" end="10" />,document.getElementById('root'));
```



- Componente con 3 propiedades

Paso de propiedades al componente

- En el componente se recogen las propiedades a través de “this.props.<nombre_prop>”
- Pueden usarse directamente en el componente o pre-procesarlas para generar la funcionalidad deseada
- Para acceder a los elementos hijos de los props, usaremos this.props.<nombre_prop>.children
 - Para acceder a un hijo concreto this.props.<nombre_prop>.children(i)

Ejemplo de props

```
render(){
  let numtable=[];
  for (let i = this.props.init ; i <=this.props.end; i++) {
    numtable.push(<li className="list-group-item">{i} x {this.props.num} =
      {i*this.props.num}</li>);
  }

  return (
    <div className="container">
      <h1>Tabla del {this.props.num}</h1>
      <ul className="list-group">
        {numtable}
      </ul>
    </div>
  )
}
```

Renderizado de listas componentes

- Las listas de elementos a renderizar se pasarán de componente padre a hijo mediante prop
- En el hijo podemos realizar un mapeo de dicha lista para generar el renderizado de un componente por elemento
- Cada elemento de dicho mapeo debe tener una clave única (atributo key)

Ejemplo de renderizado de listas componentes

```
const UserList = React.createClass({
  render: function() {
    var userNodes = this.props.data.map(function(user) {
      return (
        <User key={user.id} id={user.id} name={user.name} email={user.email} />
      );
    });
    return (
      <div className="UserList">
        <table className="table table-striped">
          <thead>
            <tr><th>Id</th><th>Nombre</th><th>Email</th></tr>
          </thead>
          <tbody>
            {userNodes}
          </tbody>
        </table>
      </div>
    );
  }
});
```

```
var User = React.createClass({
  render: function() {
    return (
      <tr><td>{this.props.id}</td><td>{this.props.name}</td><td>{this.props.email}</td></tr>
    );
  }
});
```



Pongámoslo en práctica: La App de tareas y proyectos

- Crea una App para gestionar tareas que muestre:
 - Una lista de tareas: id, descripción, tiempo, id_proyecto
 - Una lista de proyectos : id, nombre_proyecto

Los estados de los componentes (el estado reactivo)

- Los props son inmutables: son pasados por el componente padre y son propiedad del mismo. No se pueden cambiar por el hijo.
- Para permitir interacciones se puede usar el estado del componente `this.state` (privado al componente) y sólo puede ser cambiado mediante la invocación de `this.setState()`.
- Dentro del estado podemos poner cualquier modelo que estemos usando para la funcionalidad del componente.
- Cuando el estado se actualiza, el componente se vuelve a renderizar a si mismo.
- Cuando un componente se inicializa por primera vez se ejecuta el método `getInitialState()`, donde se define el estado inicial del componente.

Ejemplo del manejo de estado del componente

- Al inicializarse el componente definimos un primer estado
- Cuando sucede un evento del ciclo de vida actualizamos el estado
- El DOM se actualiza con el cambio de estado

```
getInitialState: function() {  
  return {data: []};  
},
```

```
componentDidMount: function() {  
  this.loadUsers();  
},
```

```
render: function() {  
  return (  
    <div className="UserBox">  
      <h1>Usuarios</h1>  
      <UserList data={this.state.data} />  
    </div>  
  );  
}
```

Componente puro

- Cuando un componente es solo resultado de los props y sin estado, el componente se puede escribir como una función pura y no hay necesidad de crear una instancia del componente de React.

```
var MyComponent = function(props){  
  return <div>Hola {props.name}!</div>;  
};  
  
ReactDOM.render(<MyComponent name="ricardo" />, root);
```

Aspectos importante sobre el estado en React

- Es recomendable tener en lo posible componentes stateless.
- El componente de orden superior debe mantener el estado y traspasárselo a los componentes descendientes en forma de props
- React favorece el flujo de estados en una sola dirección. Por tanto se debe construir la arquitectura de componentes teniendo en cuenta esta visión.

Ciclo de vida del componente

- React permite crear componentes invocando el método `react.createClass ()`. Este espera un método render y desencadena un ciclo de vida sobre el que se puede actuar a través de los llamados métodos de ciclo de vida.
- Para tener una idea clara del ciclo de vida tendremos que diferenciar entre la **fase inicial de creación**, donde se crea el componente; y **actualizaciones** provocadas por cambios en el estado y props; así como la **fase de desmontaje** del componente.

Ciclo de vida - Inicialización

✓ Initial

✓ GetDefaultProps

✓ GetInitialState

✓ ComponentWillMount

✓ Render

✓ ComponentDidMount

- **getInitialState** permite establecer el valor de estado inicial, accesible en el componente a través de **this.state**.
- **getDefaultProps** pueden utilizarse para definir los props por defecto.
- **componentWillMount** se llama antes de que se ejecute el método render. Es importante tener en cuenta que el establecimiento del estado en esta fase no dará lugar a un re-rendering.
- **Render** devuelve el HTML del componente
- **ComponentDidMount** Se invoca inmediatamente después del montaje. Si hay una actualización del estado, hará que se re-renderice el componente.

Ciclo de vida – Cambios en estado

✓ Updating State

✓ ShouldComponentUpdate

✓ ComponentWillUpdate

✓ Render

✓ ComponentDidUpdate

- **shouldComponentUpdate** se llama siempre antes de render y permite definir si se necesita un re-rendering o se puede omitir.
- **componentWillUpdate** es llamado inmediatamente si shouldcomponentupdate devolvió **true**.
- **componentDidUpdate** se llama después de render.

Ciclo de vida – Cambios en props

✓ Updating Props
✓ ComponentWillReceiveProps
✓ ShouldComponentUpdate
✓ ComponentWillUpdate
✓ Render
✓ ComponentDidUpdate

- **componentWillReceiveProps** sólo se invoca cuando los props han cambiado y cuando no es un rendering inicial. Permite actualizar el estado en función de los apoyos existentes y futuros, sin desencadenar otra representación.
- El resto de elementos del ciclo de vida son idénticos al ciclo activado por cambio de estado.

Ciclo de vida - Desmontaje

✓ Unmounting

✓ `componentWillUnmount`

- **componentWillUnmount** se llama justo antes de que el componente se elimine del DOM.



Pongámoslo en práctica

- Haz que el estado inicial de la App (lista de tareas y proyectos) se cargue con el ciclo de vida del componente.

Propiedades por defecto

- Se pueden generar propiedades por defecto de un componente a través del constructor o de los métodos del lifecycle

- Constructor:

```
App.defaultProps = {  
  headerProp: "Header from props...",  
  contentProp: "Content from props..."  
}
```

- Lifecycle:

- getInitialState
 - getDefaultProps

Validaciones

- Los componentes en React están diseñados para poder agruparlos en componentes más grandes y ser reutilizados, ya sea en el mismo proyecto, en otros proyecto o por otros desarrolladores.
- Por lo tanto es una buena práctica definir explícitamente las propiedades que acepta un componente, cuáles son requeridas y los tipos de dato de cada una.
- Este es el propósito propTypes. Nos ayudará a:
 - Revisar rápidamente cuáles son las propiedades que debemos pasar a un componente
 - – Cuándo hay un error, React mostrará un mensaje en la consola indicando qué propiedades faltan y qué método generó el problema.

Ejemplo de propType

- En el siguiente ejemplo se indica que, para el componente Título, el atributo nombre es de tipo string y obligatorio

```
class Titulo extends React.Component {  
  render () {  
    return (  
      <h1>{ this.props.nombre }</h1>  
    );  
  }  
}  
  
Titulo.propTypes = {  
  nombre: React.PropTypes.string.isRequired  
};
```

Validadores de propTypes

- React incluye validadores que cubren diversos escenarios para verificar que los datos recibidos son correctos, todos son opcionales pero es posible encadenar **.isRequired** para asegurarse que el dato no sea omitido.
- **Tipos primitivos**

Validador	Descripción
PropTypes.array	Prop debe ser un arreglo
PropTypes.bool	Prop debe ser un valor boolean
PropTypes.func	Prop debe ser una función
PropTypes.number	Prop debe ser un número
PropTypes.object	Prop debe ser un objeto
PropTypes.string	Prop debe ser una cadena

Validadores de propTypes

➤ Tipos combinados

Validador	Descripción
PropTypes.oneOfType	Prop puede tener varios tipos
PropTypes.arrayOf	Prop debe ser un arreglo de cierto tipo
PropTypes.objectOf	Prop debe ser un objeto con propiedades de cierto tipo
PropTypes.shape	Prop debe ser un objeto con propiedades específicas

Validadores de propTypes

➤ Tipos especiales

Validador	Descripción
PropTypes.node	Prop puede ser cualquier tipo que pueda ser mostrado (number, string, elemento de un arreglo)
PropTypes.element	Prop debe ser un elemento de React
PropTypes.instanceOf	Prop debe ser una instancia de una clase
PropTypes.oneOf	Prop está limitado a un número específico de valores

Custom validators

- Es posible crear validadores personalizados para casos específicos.
- Estos validadores se crean como una función de JavaScript que recibe una lista de propiedades, el nombre de la propiedad a revisar y el nombre del componente.

```
let nombrePropType = (props, propName, componentName) => {
  if (props[propName]) {
    let value = props[propName];
    if (typeof value !== 'string' || value.length > 100) {
      console.log('error');
      return new Error(
        `${propName} en ${componentName} tiene más de 100 caracteres`
      );
    }
  }
}
```

```
Titulo.propTypes = {
  nombre: nombrePropType
};
```



Pongámoslo en práctica

- Valida que la lista de tareas que le llegan al componente de tareas cumpla con la especificación:
 - Ser un array de objetos tarea
 - Objeto tarea compuesto de: id, descripción, id_proyecto

2

JSX

Usando JSX

- JSX es la opción de React a la sintaxis JavaScript para escribir código declarativo de estilo XML dentro del código JavaScript.
- Para proyectos web, React JSX ofrece un conjunto de etiquetas XML que son similares a HTML.
- Cuando es transpilado (convertido en JavaScript plano, de modo que el navegador o servidor puedan interpretar el código), el XML se transforma en una llamada a la librería React.

- `<h1>Hola Mundo</h1>` →
- `React.createElement("h1", null, "Hola Mundo");`

Usando JSX (II)

- El uso de JSX es opcional, pero usándolo obtendremos los siguientes beneficios:
 - XML es ideal para representar árboles de UI y sus atributos.
 - La estructura de su aplicación es más concisa y fácil de visualizar.
 - Es JavaScript plano. No altera la semántica del lenguaje.

Diferencias entre JSX y HTML

- Hay tres aspectos importantes que se deben tener en cuenta al escribir HTML con JSX:
- Los atributos de las etiquetas son camel case.

```
<input type="text" maxLength="30" />
```

```
return <input type="text" maxLength="30" />
```

- Todos los elementos deben estar balanceados (tags cerrados).

```
<div id="box" className="some-class"></div>
```

- Los nombres de los atributos se basan en la API de DOM, no en las especificaciones HTML.

```
return <div id="box" className="some-class"></div>
```

- Una referencia útil es: <https://developer.mozilla.org/en-US/docs/Web/API/Element/attributes>

Sin JSX

- React fue diseñado con JSX en mente; Sin embargo, es posible utilizar React sin JSX.
- Para ello hay que usar el método **createElement**, que toma un nombre de etiqueta o componente, un objeto de propiedades, y el número variable de argumentos opcionales.

```
let child1 = React.createElement('li', null, 'First Text Content');
let child2 = React.createElement('li', null, 'Second Text Content');
let root = React.createElement('ul', { className: 'my-list' }, child1, child2);
React.render(root, document.getElementById('example'));
```

Factorías de elementos

- Para mayor comodidad, React proporciona funciones factory bajo `React.DOM` para las etiquetas HTML habituales.

```
React.DOM.form({className:"commentForm"},  
  React.DOM.input({type:"text", placeholder:"Name"}),  
  React.DOM.input({type:"text", placeholder:"Comment"}),  
  React.DOM.input({type:"submit", value:"Post"})  
)
```

- Equivalent al siguiente JSX:

```
<form className="commentForm">  
  <input type="text" placeholder="Name" />  
  <input type="text" placeholder="Comment" />  
  <input type="submit" value="Post" />  
</form>
```

Factorías personalizadas

- Es posible crear factorías para componentes personalizados.

```
let Factory = React.createFactory(ComponentClass);
...
let root = Factory({ custom: 'prop' });
render(root, document.getElementById('example'));
```

Just in Time JSX Transformer

- Las herramientas de Node, Babel y Webpack permiten que se haga la transpilación de JSX y se sirva en el server que proporciona Node.
- Esto se logra con la orden: node devServer.js
- En este caso la transpilación se realiza en el servidor node siguiendo la configuración definida en devServer.js, donde se define:
 - El compilador: var compiler = webpack(config); ← config = require('./webpack.config');
 - Los métodos de escucha get y post:
 - app.get('*', function(req, res) {...});
 - app.post('*', function(req, res) {...});
 - El puerto donde escucha node: app.listen(7770, 'localhost', function(err) {...})

Just in Time JSX Transformer (II)

- Por su parte el archivo de configuración de Webpack, **webpack.config.js**, especifica cómo se hará el empaquetado:
 - El punto de entrada de la aplicación:

```
entry: [  
  'webpack-hot-middleware/client',  
  './client/main.jsx'  
],
```

- El punto de salida del empaquetado:

```
output: {  
  path: path.join(__dirname, 'dist'),  
  filename: 'bundle.js',  
  publicPath: '/static'  
},
```

Just in Time JSX Transformer (III)

- Por su parte el archivo de configuración de Webpack, **webpack.config.js**, especifica cómo se hará el empaquetado:
 - Los loaders de JS (en este caso babel) y Css

```
module: {  
  loaders: [  
    // js  
    {  
      test: /\.jsx$/,  
      loaders: ['babel'],  
      include: path.join(__dirname, 'client')  
    },  
    // CSS  
    {  
      test: /\.styl$/,  
      include: path.join(__dirname, 'client'),  
      loader: 'style-loader!css-loader!stylus-loader'  
    }  
  ]  
}
```

Just in Time JSX Transformer (IV)

- Asimismo el archivo .babelrc define la configuración del babel-loader, donde se le indica que use ES6 y los plugins JSX:

```
{  
  "presets": ["react", "es2015"],  
  "env": {  
    "development": {  
      "plugins": [  
        ["transform-object-rest-spread"],  
        ["transform-react-display-name"],  
        ["react-transform", {  
          "transforms": [  
            "react-transform-multi-line-objects"  
          ]  
        }]  
      ]  
    },  
    "production": {  
      "plugins": [  
        ["transform-object-rest-spread"],  
        ["transform-react-display-name"]  
      ]  
    }  
  }  
}
```

JSX Transformer on-the-fly

- Existe la posibilidad de usar un conjunto de librerías para generar la transformación de JSX on-the-fly.
- Como el navegador no entiende JSX de forma nativa, necesitamos transformarlo a JavaScript primero.
- De esto se logra incluyendo la librería Babel 5 para la transformación in-browser de ES6 y JSX, llamado browser.js.
- Babel reconoce el código JSX en los tags `<script type="text/babel"></script>` y transforma a JavaScript on the fly.
- Transforming JSX in the browser works quite well during development.



Plantilla de transformación on the fly

- Valida que la lista de tareas que le llegan al componente de tareas cumpla con la especificación:
 - Ser un array de objetos tarea
 - Objeto tarea compuesto de: id, descripción, id_proyecto



Plantilla de transformación on the fly

➤ Descomprime la plantilla

- › En Lab2.1 encontrarás la plantilla jittx_base.zip
- › Descomprime la plantilla en tu directorio de workspace
- › Dale una nuevo nombre al directorio (ej. Lab2_1)

➤ Inspecciona

- › Inicia SublimeText
- › Open > folder > [ruta de la carpeta del laboratorio]
- › Observa el archivo index.html
- › Observa el archivo js/base.jsx

➤ Lanza el proyecto

- › Abre una ventana de consola en donde reside index.html
- › Lanza el servidor lite, escribiendo: lite-server
- › Se abrirá una ventana del navegador en la raíz de la aplicación



Pongámoslo en práctica

- Transforma la demo de Comentarios (1.2client_anidamiento) en una versión on the fly

Pre-process Transformer

- Si queremos generar una versión pre-transpilada de nuestro proyecto podemos usar webpack para generar el js y empaquetarlo
- Para ello abriremos una ventana de consola y lanzaremos la orden de empaquetamiento:
 - webpack –config <config_file>
 - En nuestro caso webconfig simplemente
- Webconfig generará una versión empaquetada de nuestra aplicación: bundle.js en el directorio dist
- Una vez generado, deberemos referenciar desde index.html dicho archivo para que use la App
- Una vez cohesionados todos los elementos del proyecto, podemos poner la app en producción.



Pongámoslo en práctica

- Genera el “bundle” de tu App de Tareas y Proyectos
- Mueve el bundle.js a el directorio apropiado
- Inicia el servidor lite-server para visualizar la app:
 - Abre una ventana de comando donde resida index.html
 - Lanza el server escribiendo: lite-server

Attribute Expressions

- En JSX se pueden usar un conjunto de expresiones Javascript para definir atributos
- Simplemente basta con encerrar entre "{}" las expresiones que queramos evaluar y asignarlas a un atributo
 - <App data = {data} />
- Estas expresiones pueden ser diversas:
 - Un array: headings = {'When', 'Who', 'Description'}
 - Un operador ternario (no se puede usar if-else): data = {data.length > 0 ? data : ""}
- Si una expresión es demasiado compleja para añadirla en un atributo, simplemente se debe mover fuera y apoyarse en una función o variable.

Attribute Expressions – Spread Attributes

- Un caso especial es el de los **spread attributes** que permite encapsular los props y luego pasárselo al componente
- Primero encapsulamos los props:
 - `var props = { headings: headings, changeSets: data, timestamps: timestamps };`
- Luego los pasamos al componente:
 - `<App {...props} />`
- Los spread attributes nos permiten lidiar con un número grande de atributos y además permiten que los atributos que se le puedan pasar a un componente esté abierto a la extensión.
- Asimismo se puede combinar con más referencias a props.

Attribute Expressions - comentarios

- También se pueden añadir comentarios en un tag JSX de esta manera:

```
> <App  
/* Multi  
Line  
Comment  
*/
```

```
headings = {headings} changeSets = {data} />
```

- Otra opción es usar expresiones:

```
<th> /* This is a comment */ {this.props.heading} </th>
```

Expresiones Hijas

- Un tipo especial de expresiones son las que definen los hijos de un componente y que se conocen como “child expressions”
- Estas expresiones permiten incluir directamente un array de componentes en otro componente

```
var userNodes = this.props.data.map(function(user) {
  return (
    <User key={user.id} id={user.id} name={user.name} email={user.email} />
  );
});
return (
  <div className="UserList">
    <table className="table table-striped">
      <thead>
        <tr><th>Id</th><th>Nombre</th><th>Email</th></tr>
      </thead>
      <tbody>
        {userNodes}
      </tbody>
    </table>
  </div>
);
```

HTML y Atributos

- Como ya hemos indicado, los atributos de JSX responden a la API DOM
 - <https://developer.mozilla.org/en-US/docs/Web/API/Element/attributes>
- Para el caso del atributo class, se deberá usar *className*

```
<div className="unaclase">test</div>;
```
- Para definir estilos en línea, se deberá definir primero una variable con los estilos y luego referenciarlos desde el atributo style:

```
var styles = {  
    color:'red',  
    backgroundColor:'black',  
    fontWeight:'bold'  
};  
  
<div style={styles}>test</div>;
```

HTML y Atributos

- Para pasar atributos que no son parte de la especificación, es necesario que se pasen como atributos “data-”
 - `<table data-custom-attribute = 'super_awesome_table'></table>`
- Asimismo algunos caracteres, como el “&” deben ser escapados mediante “&”
 - `<div> { first + '&' + second } </div>`
 - De hecho React escapa todos caracteres para evitar ataques XSS
- Asimismo React permite renderizar HTML plano, usando el prop: **dangerouslySetInnerHTML**
 - `<div dangerouslySetInnerHTML={{__html: 'Mike & Shawn'}} />`



Pongámoslo en práctica

- Añade a tu App de Tareas y Proyectos una estructura de tabla basados en estilos bootstrap
- Para ello añade la referencia bootstrap en tu html y usa las clases de los componentes de <http://getbootstrap.com/components/>
- Usa child expressions para la lista de tareas y proyectos

3

EVENTOS

Gestión de eventos

- React implementa un sistema de eventos sintético que aporta consistencia y alto rendimiento.
- La consistencia se logra mediante la normalización de eventos para que tengan las mismas propiedades en diferentes navegadores y plataformas.
- El alto rendimiento se logra mediante el uso automático de delegación de eventos. React en realidad no asocia event handlers a los nodos; sino que un único detector de eventos está unido a la raíz del documento html.
- Cuando se dispara un evento, React mapea el evento al componente apropiado.
- React también elimina automáticamente los detectores de eventos cuando un componente se desmonta.

Gestión de eventos (II)

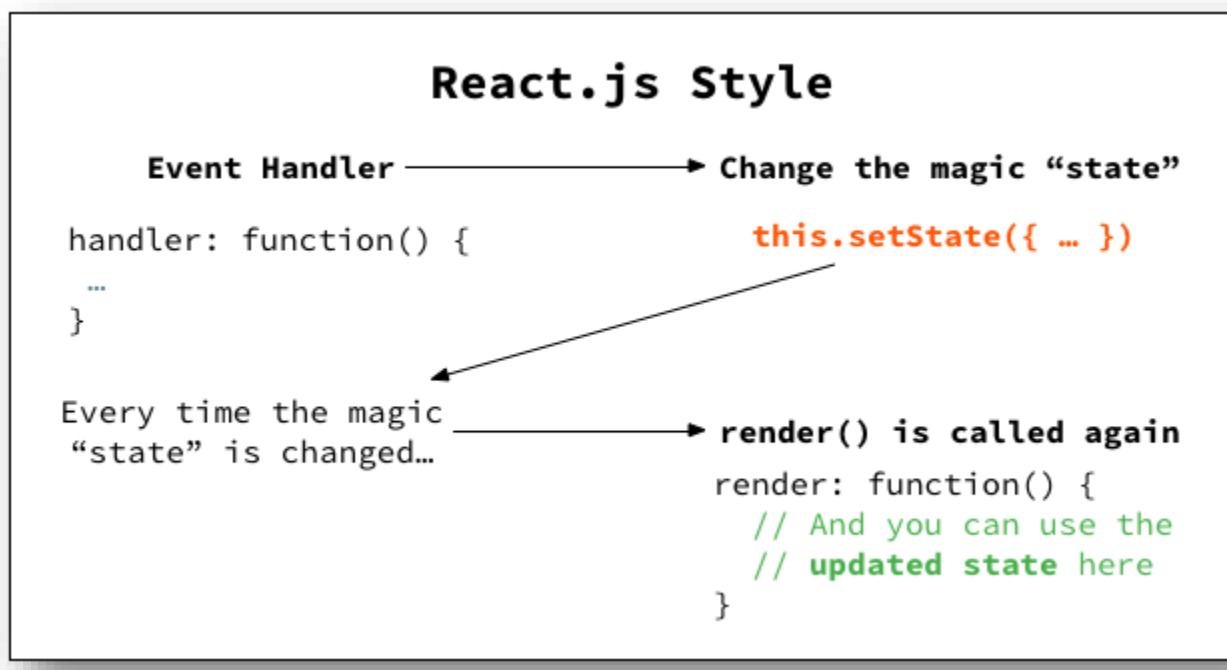
- Para gestionar los eventos definiremos una función handler que actúe ante el evento

```
<p><input  
  type="text"  
  placeholder="Nombre"  
  value={this.state.name}  
  onChange={this.handleNameChange}>  
</p>
```

```
handleNameChange: function(e) {  
  this.setState({name: e.target.value});  
},
```

Gestión de eventos (III)

- La estrategia normalmente se sigue es cambiar el estado del componente cuando existe un evento



Eventos DOM

- HTML proporcionado una API de gestión de eventos: onclick, onfocus, etc. El problema con esta API (y la razón por la que no se utiliza en proyectos profesionales) es que está lleno de efectos secundarios indeseables: contamina el scope global; lo que hace difícil hacer tracking y crea memory leaks.
- JSX presenta una API igualmente fácil de usar y entender, pero elimina los efectos secundarios no deseados: las funciones de callback están en el ámbito del componente (que, como hemos visto, es responsable de sólo una parte de la interfaz de usuario y tiende a contener pequeñas marcas), y es lo suficientemente inteligente como para utilizar la delegación de eventos y gestionar su desmontaje.

Eventos DOM (II)

- Hay algunas diferencias menores en contraste con el HTML original.
- En React, las propiedades son tipo camel code ("onClic" en lugar de "onclick").
- React implementa un subconjunto de todas las variaciones de eventos disponibles.
- Se puede ver la definición completa en:
<https://facebook.github.io/react/docs/events.html>

Eventos DOM – Lista de eventos

TOUCH AND MOUSE EVENTS

onTouchStart	onTouchMove	onTouchEnd	onTouchCancel	
onClick	onDoubleClick	onMouseDown	onMouseUp	onMouseOver
onMouseMove	onMouseEnter	onMouseLeave	onMouseOut	onContextMenu
onDrag	onDragEnter	onDragLeave	onDragExit	onDragStart
onDragEnd	onDragOver	onDrop		

KEYBOARD EVENTS

onKeyDown	onKeyUp	onKeyPress
-----------	---------	------------

Eventos DOM – Lista de eventos (II)

FOCUS AND FORM EVENTS

onFocus onBlur

onChange onInput onSubmit

OTHER EVENTS

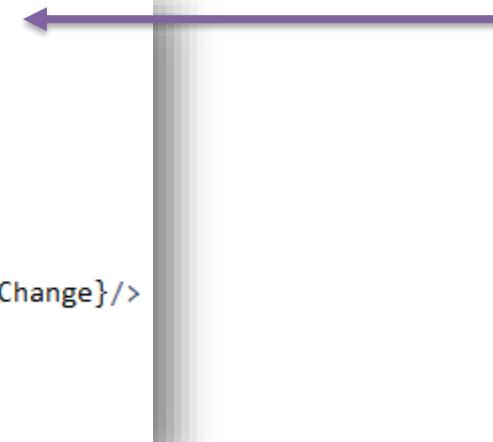
onScroll onWheel onCopy onCut onPaste

Composición de eventos

- En React es posible observar/recoger un evento desde un componente padre, cuando el evento ocurre en el hijo
- Para ello es necesario pasar desde el padre, mediante props, las funciones que hacen de handlers de los eventos que suceden en los hijos

Composición de eventos - Ejemplo

```
handleParentChange: function(e) {  
  console.log('you changed child component');  
  console.log(e); //e is sysnthetic event instance  
},  
render: function() {  
  return (  
    <div className="container">  
      <h1>Event detector</h1>  
      <ChildComponent onChange={this.handleParentChange}>/>  
    </div>  
  );  
}
```



Llamada a la función de control del padre desde el hijo

➤ Paso de función de control al hijo

```
var ChildComponent = React.createClass({  
  <select onChange={this.handleChange}>  
    <option value="1">1</option>  
    <option value="2">2</option>  
  </select>
```

```
handleChange: function(e) {  
  console.log('you changed select');  
  console.log(e); //e is sysnthetic event instance  
  
  this.props.onChange(e);  
},
```

Eventos Touch

- Existen 4 eventos específicamente pensados para el entorno móvil y que permiten detectar el ciclo de vida del proceso de tacto en los interfaces:
 - onTouchStart: se dispara cuando un evento touch comienza
 - onTouchEnd : se dispara cuando un evento touch termina
 - onTouchCancel: se dispara cuando un evento touch se ha cancela
 - onTouchMove: se dispara cuando movemos un elemento
- Con estos 4 eventos podemos hacer que nuestra app sea más compatible en el entorno móvil.
- Si queremos ir más allá en el entorno móvil, React Native proporciona componentes para gestionar los gestos más comunes, así como un **sistema integral de respuesta a los gestos** (<https://facebook.github.io/react-native/docs/gesture-responder-system.html>) para permitir el reconocimiento más avanzado gesto.



Pongámoslo en práctica: Un filtro para las tareas y proyectos



Un filtro para las tareas y proyectos

- Añade a tu App de Tareas y Proyectos un nuevo componente que consista en un campo (uno para cada caso) para filtrar las tareas y proyectos por el texto que se introduce.
- Para ello deberás usar la detección de eventos y filtrar las tareas (o proyectos) en función de dicho evento.
- Hint:
 - Poner el valor del campo en el state del componente superior (la lista).
 - Crear asimismo un array con los elementos filtrados en el state.
 - Para filtrar elementos:
 - var filtered=array.filter(filterElements(query,'desc'));
 - Donde filterElements es una función que se aporta en el js Shared.js de Lab3_1

4

GESTIÓN DE FORMULARIOS

Formularios en React

- En React, el estado interno de un componente se mantiene al mínimo, ya que cada vez que cambia el estado, el componente se renderiza de nuevo. React lleva el control de la sincronización.
- Por esta razón, componentes de formulario como <input>, <textarea>, y <option> difieren de sus homólogos HTML, ya que pueden ser mutados via interacciones de usuario.
- React proporciona dos maneras de manejar los formularios como componentes: componente controlado o componente no controlado.

Componentes controlados

- Un componente de formulario con un valor o prop se llama un componente controlado.
- En un componente controlado, el valor renderizado dentro del elemento siempre reflejará el valor del prop. Por defecto, el usuario no será capaz de cambiarlo.
- Para actualizar el valor del prop, necesitaremos usar el estado en el handler del evento

Componentes controlados - Ejemplo

```
var InputExample = React.createClass({
  getInitialState() {
    return (
      { name: '-' }
    );
  },
  handleChange(event) {
    this.setState({ name: event.target.value });
  },
  render() {
    return (
      <input type="text"
        value={this.state.name}
        onChange={this.handleChange} />
    );
  }
});
```

Componentes controlados – casos especiales

➤ **Textarea:**

- En HTML: <textarea>Este es el cuerpo del textarea</textarea>
- En React: <textarea value="Este es el cuerpo del textarea" />

➤ **Select:**

- En HTML se usar el selected en el option que está seleccionado
- En React:

```
<select value="B">  
    <option value="A">Mobile</option>  
    <option value="B">Work</option>  
    <option value="C">Home</option>  
</select>
```

Componentes no controlados

- Los componentes controlados se adhieren a los principios de React y tienen sus ventajas.
- Los componentes no controlados son un anti-patrón en la que no es necesario supervisar la entrada del usuario campo a campo.
- Esto es especialmente cierto en los formularios grandes, en los que desea que el usuario rellene los campos y luego procesar todo cuando haya finalizado.
- Si se quiere definir un valor inicial para los campos se puede usar el prop **defaultValue**
- Aún es posible capturar los valores del formulario a través del evento **onSubmit**

Componentes no controlados - Ejemplo

```
var FormComponent = React.createClass({
  handleSubmit(event) {
    console.log("Valores enviados: ", event.target.name.value, event.target.email.value);
    event.preventDefault();
  },
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <div className="formGroup">
          Nombre: <input name="name" type="text" defaultValue="Tu nombre"/>
        </div>
        <div className="formGroup">
          E-mail: <input name="email" type="mail" defaultValue="Tu email"/>
        </div>
        <button type="submit">Enviar</button>
      </form>
    )
  }
});
```

Eventos del formulario

- Como hemos visto, en los formularios de React, rigen los mismos eventos que en el resto de componentes y que son la versión React de los eventos DOM
- En el caso especial de los formularios, existe el evento **onSubmit** que permite capturar el evento de envío de un formulario.
- Para evitar el envío del formulario propiamente dicho, es necesario capturar el evento en la función handler del evento y aplicar un **event.preventDefault()** al inicio del método

Referencias

- En React, cuando se renderiza un componente se está trabajando siempre con el DOM virtual: Si se cambia el estado de un componente o enviar nuevos apoyos a hijo, se re-renderiza el DOM virtual de manera reactiva.
- React actualiza el DOM real después de la fase de reconciliación.
- Esto significa que como desarrolladores nunca tocamos el DOM real.
- En los casos en los que se quiere tocar el DOM real, React provee el mecanismo llamado **refs**
- **Nota:** No se recomienda manipular el DOM real directamente. En casi todos los casos hay una manera más clara de hacerlo en React.

Referencias

- Los refs pueden ser usados como una prop en cualquier componente:
 - <input ref="myInput" />
- Y el elemento DOM referenciado puede ser accedido via **this.refs**:
 - let input = this.refs.myInput;
 - let inputValue = input.value;
 - let inputRect = input.getBoundingClientRect();



Pongámoslo en práctica

- Añade a tu App de Tareas y Proyectos un nuevo componente-formulario que permita añadir una nueva tarea.
- Cuando se añade una nueva tarea, el proyecto se seleccionará de una lista generada dinámicamente en función de los proyectos existentes.

5

ENRUTADO

React Router

- React router es la solución más popular para añadir enrutamiento en una aplicación React.
- Mantiene la interfaz de usuario síncrona con la URL al asociar componentes con rutas (en cualquier nivel de anidamiento).
- Cuando el usuario cambia la dirección URL, los componentes se desmontan y montan de forma automática.
- Otra ventaja de React Router es que proporciona mecanismos para controlar el flujo de una aplicación sin puntos de entrada diferentes; dependiendo de si el usuario ha introducido un estado programáticamente o indicando una nueva URL: el código que se ejecuta en todos los casos es el mismo.

React Router - componentes

- React Router proporciona 3 componentes:
 - **Router** y **Route**: Se utilizan para asignar de forma declarativa rutas a la jerarquía de la aplicación.
 - **Link**: Se utiliza para crear un anchor totalmente accesible con el href adecuado.
- Para usar React Router es necesario instalar e importar la librería en el componente principal.
 - `import { Router, Route, Link } from 'react-router';`
- Asimismo importar todos los components que se vean afectados por una ruta.

React Router - Configuración

- React Router proporciona 3 componentes:
 - **Router** y **Route**: Se utilizan para asignar de forma declarativa rutas a la jerarquía de la aplicación.
 - **Link**: Se utiliza para crear un anchor totalmente accesible con el href adecuado.
- Para usar React Router es necesario instalar e importar la librería en el componente principal.
 - `import { Router, Route, Link } from 'react-router';`
- Asimismo importar todos los components que se vean afectados por una ruta.

React Router - Ejemplo

```
import { Router, Route, IndexRoute, Link, browserHistory } from 'react-router';

import Main from './components/Main.jsx';
import UserBox from './components/UserBox.jsx';
import UserForm from './components/UserForm.jsx';

const router = (
  <Router history={browserHistory} >
    <Route path="/" component={Main}>
      <IndexRoute component={UserBox}></IndexRoute>
      <Route path="/new" component={UserForm}></Route>
    </Route>
  </Router>
)

render(
  router,
  document.getElementById('root')
);
```

React Router - Configuración (II)

➤ Router:

- Define la configuración de enrutado de la aplicación
- **history**: Indica el componente que se encargará de recoger el historial de la barra de direcciones.

➤ Route:

- Cada una de las rutas y su componente asociado
- Dentro pueden haber más configuraciones de rutas hijas.
- **IndexRoute**: Indica la ruta hija que será usada por defecto (y el componente asociado) si no se indica una ruta hija

```
<Router history={browserHistory} >
  <Route path="/" component={Main}>
    <IndexRoute component={UserBox}></IndexRoute>
    <Route path="/new" component={UserForm}></Route>
  </Route>
</Router>
```

React Router - Componente Main

- **Link:**
 - Define el enlace a una ruta. Esta se presentará como un anchor.
- **React.cloneElement(this.props.children, this.props):**
 - Copia los props que recibe el componente Main en el componente hijo que implemente la ruta (UserBox)

```
return (
  <div className="container">
    <h1>Usuarios</h1>
    <nav>
      <Link to="/">Usuarios</Link> | <Link to="/new">Nuevo</Link>
    </nav>
    {React.cloneElement(this.props.children, this.props)}
  </div>
)
```

Props en la configuración de las rutas

- Se puede añadir **props en la configuración de una ruta**
 - <Route path="about" component={About} title="About Us" />
 - Estás serán accesibles en el componente como:
{this.props.route.title}
- Asimismo se pueden definir **rutas con path param:**
 - Estos se identifican con los dos puntos (:) delante del nombre del parámetro
 - <Route path="user/:id_user" component={UserDetails} />
 - Se usará para acer a rutas del tipo: *myapp/user/245*
 - Serán accesibles a través de: this.props.params.id_user

Cambiando rutas programáticamente

- El objeto history es responsable de la gestión del historial del navegador, y proporciona los métodos para la navegación

Method	Description
pushState	<p>The basic history navigation method transitions to a new URL. You can optionally pass a parameters object.</p> <p>Example:</p> <pre>history.pushState(null, '/users/123') history.pushState({showGrades: true}, '/users/123')</pre>
replaceState	Has the same syntax as pushState, but it replaces the current URL with a new one. It's analogous to a redirect, because it replaces the URL without affecting the length of the history.
goBack	Go back one entry in the navigation history.
goForward	Go forward one entry in the navigation history.
Go	Go forward or backward in the history by n or -n
createHref	Makes a URL, using the router's config.

Cambiando rutas programáticamente (II)

- Por tanto si se quiere ir a una ruta concreta, bastará con indicarlo mediante el método pushState:

```
this.props.history.pushState(null,'/');
this.props.history.pushState(null,'/new');
```

- A partir de react-router 2+:

```
this.props.history.push('/');
this.props.history.push('/new');
```



Pongámoslo en práctica

- Añade a tu App de Tareas y Proyectos el componente de rutas de tal manera que las tareas y proyectos se muestren en distintas rutas. Asimismo el componente de creación de nuevas tareas.
- La ruta por defecto debe llevar al componente de lista de tareas.
- Al añadir una nueva tarea, se debe actualizar el estado global de las tareas e ir a la lista.

6

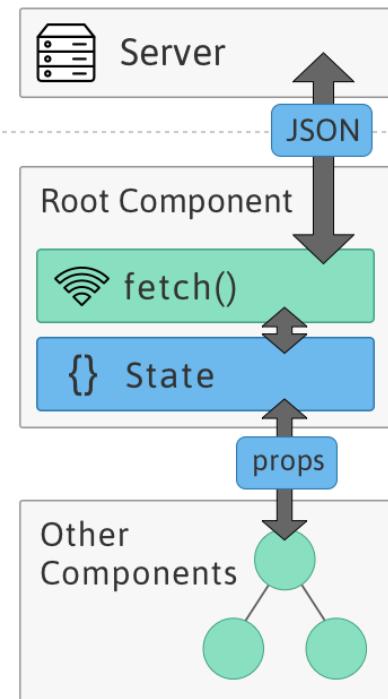
Acceso a APIs de Servidor

React y Ajax

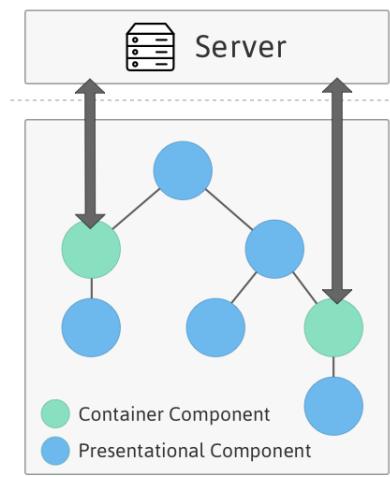
- React es una librería de vista; por tanto no tiene propiedades propias para Ajax.
- La interacción vía Ajax con el servidor está en el terreno propio de los métodos Javascript (get, fetch, etc.).
- Lo que sí hay que tener en cuenta es cuando se obtienen los datos del servidor.
- Se propone una aproximación que permita:
 - Pre-carga de datos en el servidor sólo para el componente que se corresponde con la ruta actual
 - Recolección de datos en el cliente en caso de que el usuario navegue a una ruta diferente en la que se necesiten datos pero no se ha buscado previamente

Ajax en React: 4 aproximaciones

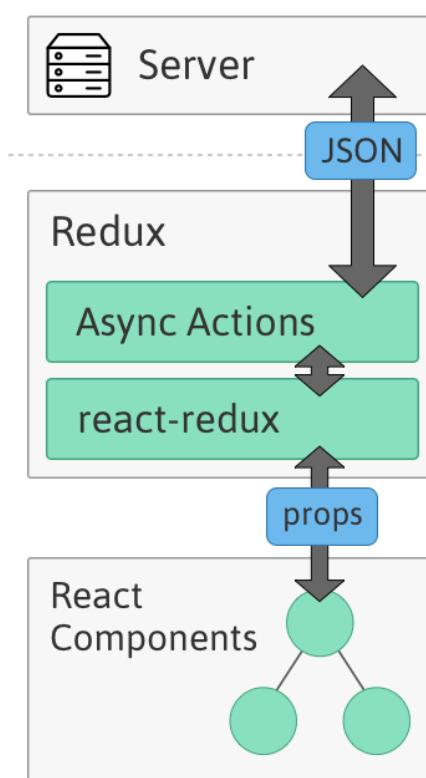
- Componente raíz



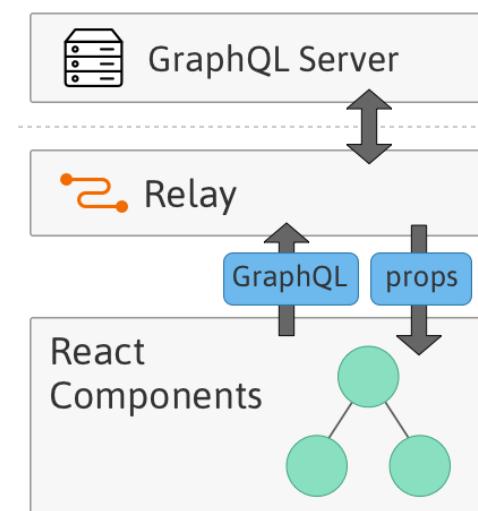
- Componentes Contenedor



- Acciones Redux Asíncronas

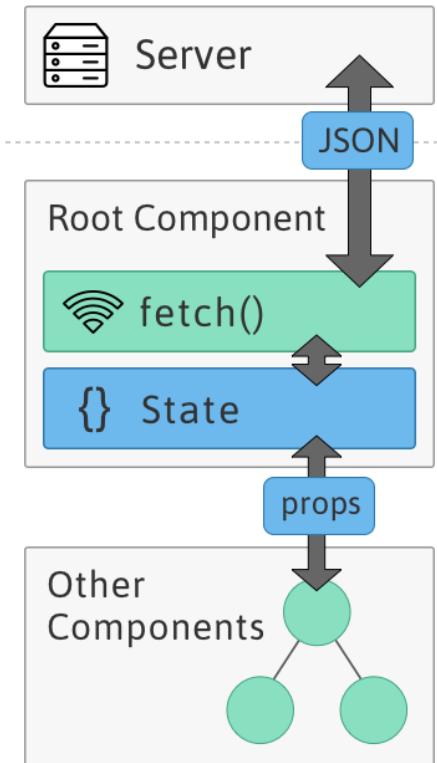


- Relay



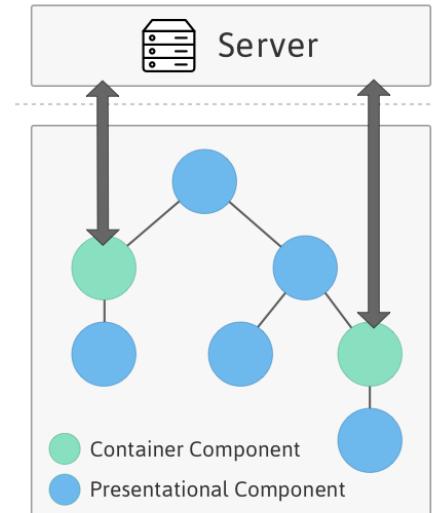
Root Component

- La aproximación más sencilla, ideal para prototipos y aplicaciones pequeñas
- Se crea un componente raíz que se encarga de todas las peticiones AJAX
- Almacena los datos en su estado y los pasa a sus hijos
- Si se tiene un árbol de componentes muy profundos podemos tener problemas en gestionar el paso del estado; por tanto se recomienda una estructura corta para esta aproximación.



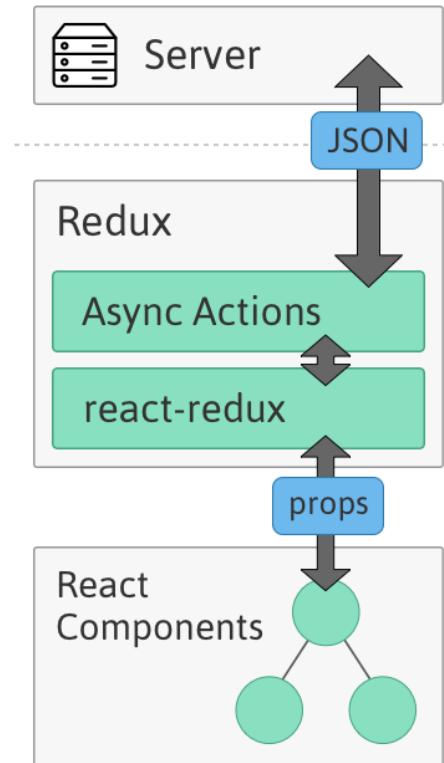
Componentes contenedor

- Un componente contenedor "proporciona los datos y el comportamiento a componentes de presentaciones u otros componentes de contenedor".
- El caso es similar al root component, excepto que los componentes que interactúan con el servidor son múltiples.
- Se recomienda para casos en que:
 - Tenemos árboles más profundos
 - La gran mayoría no requieren datos del servidor
 - Se están obteniendo datos de múltiples APIs
 - No se está usando Redux/flux
 - Se prefiere componentes contenedores a “acciones asíncronas”



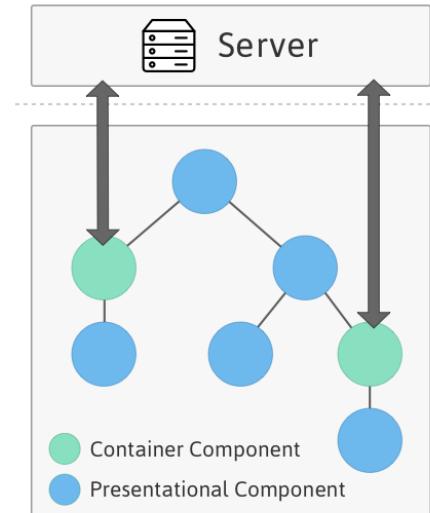
Acciones Redux Asíncronas

- Redux es un framework complementario a React y que permite implementar el patrón Flux.
- Proporciona una gestión del estado de la aplicación.
- Por tanto, Redux gestiona los datos y por tanto debería gestionar las solicitudes de red.
- Si se está usando redux, el AJAX debe ir en las acciones asíncronas.



Relay

- Relay es un framework complementario a React que permite crear aplicaciones reactivas guiadas por los datos.
- Con Relay, se declaran los datos necesarios para los componentes con GraphQL, y Relay descarga automáticamente los datos y rellena los props de componentes.
- Requiere un servidor GraphQL en la parte de la API.



Librerías AJAX

- Existen un conjunto de librerías que se pueden encargar de lidiar con AJAX

	Support				Features			
	Chrome & Firefox ¹	All Browsers	Node	Concise Syntax	Promises	Native ²	Single Purpose ³	Formal Specification
XMLHttpRequest	✓	✓				✓	✓	✓
Node HTTP			✓			✓	✓	✓
fetch()	✓			✓	✓	✓	✓	✓
Fetch polyfill	✓	✓		✓	✓		✓	✓
node-fetch			✓	✓	✓		✓	✓
isomorphic-fetch	✓	✓	✓	✓	✓		✓	✓
superagent	✓	✓	✓	✓			✓	
axios	✓	✓	✓	✓	✓		✓	
request			✓	✓			✓	
jQuery	✓	✓		✓				
reqwest	✓	✓	✓	✓	✓		✓	

<http://andrewhfarmer.com/ajax-libraries/>

- Se recomienda usar fetch(), una API Javascript simple y estándar

Usando fetch

- Fetch presenta una api muy sencilla:
 - En la orden se indica la url y las cabeceras (donde se puede indicar el método). Esto devuelve un promise.
 - En caso de éxito se entrará en la parte del **then()**
 - En caso de error se entrará en **catch()**, donde se podrán procesar los errores.

```
fetch('https://domain.name/some/url', {  
    method: 'get'  
}).then(function(response) {  
  
}).catch(function(err) {  
    // Error :(  
});
```

Usando fetch (II)

- En el caso de post
 - Se puede enviar los datos vía el body

```
fetch('https://domain.name/submit-json', {  
  method: 'post',  
  body: JSON.stringify({  
    email: document.getElementById('email').value  
    answer: document.getElementById('answer').value  
  })  
});
```

- Asimismo se puede añadir los métodos then() y catch(), como antes.

Cambiando el estado

- Una vez hayamos decidido nuestra arquitectura, el lugar ideal generar la llamada a Ajax y actualizar el estado es en el método del lifecycle **componentDidMount**
- Una vez obtenemos los datos del servidor podemos actualizar el estado de la aplicación con **setState**

```
componentDidMount: function() {  
  this.loadUsersFromServer();  
},
```

```
loadUsersFromServer: function() {  
  fetch('api-mock/users.json',{method:'get'})  
  .then(response => response.json())  
  .then(json =>{  
    console.log(json.data);  
    this.setState({data: json.data});  
  })  
  .catch(function(err){  
    console.log(err);  
  });  
},
```

Gestión de Estado

- Deberemos ocuparnos de mantener el estado de la aplicación, ya sea que existan cambios en la API o realicemos modificaciones en el mismo desde la aplicación.
- Por ejemplo,
 - cuando actualicemos el estado desde la aplicación, actualizaremos el estado en local de manera optimista y enviaremos la orden a la API.
 - La aplicación deberá estar atenta a la respuesta de la API.
 - Si ha existido un error, volveremos al estado anterior.
- La gestión del estado en una aplicación puede llegar a ser muy compleja y para esas situaciones será necesario implementar el patrón Flux usando Redux junto a React.



Pongámoslo en práctica

- Modifica la aplicación de Tareas y Proyectos para que use la API para traer los datos de tareas y proyectos
- Asimismo para enviar las nuevas tareas que se añaden.

7

Constructores y HOCs

Constructores

- En React se pueden usar dos maneras de instanciar componentes
- **React.createClass:**
- **React.Component (ES6):**

```
const MyComponent = React.createClass({  
  render:function(){  
    return (  
      <div className="container">  
        <h1>Mi componente</h1>  
      </div>  
    )  
  }  
});
```

```
class MyComponent extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  render() {  
    return (  
      <div className="container">  
        <h1>Mi componente</h1>  
      </div>  
    );  
  }  
}
```

Constructores

- Cuando usamos ES6 en React podemos definir la siguiente sintaxis para definir un constructor

```
class MyComponent extends React.Component {  
  constructor(){  
    super()  
  }  
}
```

- Si definimos un constructor (no obligatorio) es obligatorio invocar a **super()**
- Alternativamente se puede invocar a **super(props)**, si se quiere acceder a los props desde el constructor.
- En todo caso, en ES6 usaremos el constructor para inicializar el estado.

HIGHER ORDER COMPONENTS (HOC)

- Un HOC es un factory de componentes, que nos permite generar componentes reusables.
- Un HOC es una función que toma un componente existente y devuelve otro componente que lo envuelve.
 - `hocFactory:: W: React.Component => E: React.Component`
- Permite:
 - Reutilización de código, abstracción lógica y de inicialización
 - Abstracción de estado y manipulación
 - Manipulación de props

HOC - Ejemplo

```
const HOC = (Component, state, intervalFn) => class extends React.Component {  
  constructor(props) { ...  
  }  
  
  componentWillMount() { ...  
  }  
  
  componentWillUnmount() { ...  
  }  
  
  componentDidMount() { ...  
  }  
  
  setInterval() { ...  
  }  
  
  tick() { ...  
  }  
  
  render() { ...  
  }  
};
```

```
class TickTock extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  
  render() {  
    return (  
      <p>  
        React has been running for {this.props.seconds} seconds.  
      </p>  
    );  
  }  
}
```

```
const Wrapped = HOC(TickTock, { seconds: 0 }, intervalFn);  
const Wrapped2 = HOC(TickTock, { seconds: 3 }, intervalFn);  
const Wrapped3 = HOC(TickTock, { seconds: 6 }, intervalFn);
```





BananaGEST: Decisión en equipo

- Discute en equipo la conveniencia de usar React para BananaTube
- Tomad nota de las fortalezas y debilidades que hayas observado en el framework, así como, la dificultad de incorporarlo en el proyecto.
- Qué ventajas y desventajas tiene sobre Angular2?
- Tomad una decisión sobre la elección del framework



SPA BananaTube

- Con la decisión tomada sobre un framework para el frontend, impleméntalo



[...]**netmind**

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es



GOBIERNO
DE ESPAÑA

MINISTERIO
DE ENERGÍA, TURISMO
Y AGENDA DIGITAL

red.es



ESTRATEGIA DE
EMPRENDIMIENTO Y
EMPLEO JUVENIL
garantía juvenil



Agenda Digital para España



UNIÓN EUROPEA

Fondo Social Europeo
“El FSE invierte en tu futuro”