7

React JS

Desarrollo de aplicaciones web con React JS

fundación esplai
ciudadanía comprometida

React JS

Biblioteca Javascript open source para desarrollar front-end web. Creada y mantenida por Facebook, Instagram y una comunidad de desarrolladores.

Se basa en la creación de “elementos” y “componentes” reutilizables que interactúan entre sí para formar la interfaz de usuario.

Rápida, simple, escalable, MVC.

Principales características de React:

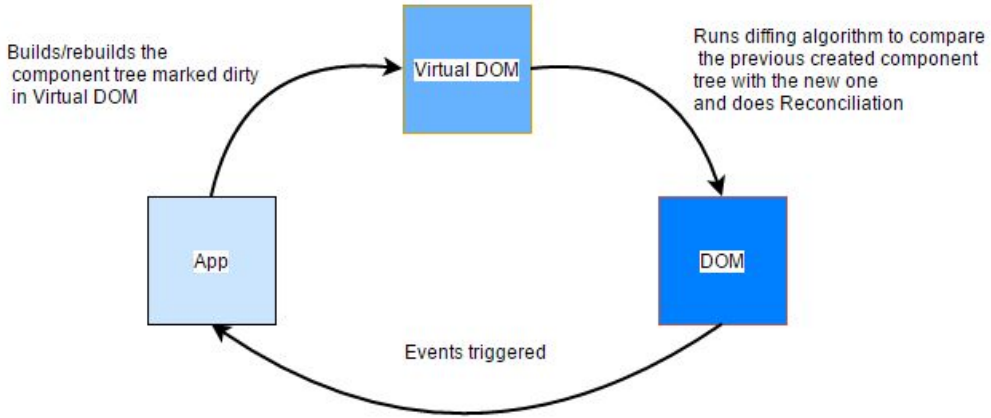
- Virtual DOM
- JSX
- Flujo de datos en un sentido

Virtual DOM

React mantiene una copia en memoria del DOM denominada Virtual DOM, donde se ubican los elementos y componentes React.

Estos objetos tienen su representación en el DOM real (via render).

Los cambios que la aplicación realiza en el Virtual DOM se actualizan **de forma selectiva** en el DOM real.

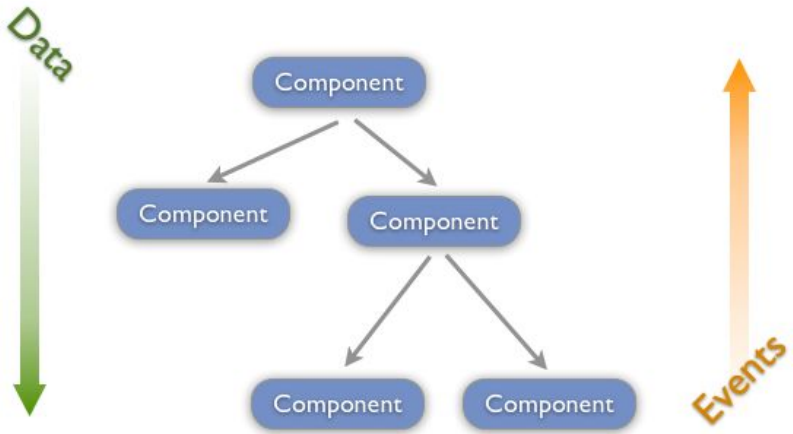


Flujo de datos unidireccional

Una interfaz React está compuesta de elementos organizados de forma jerárquica.

Los datos fluyen únicamente “hacia abajo”, de “padres” a “hijos”. Esto simplifica el diseño y la “predictibilidad”.

Estos datos pueden ser información, o funciones “callback” que gestionen eventos. A través de estas funciones denominadas “handlers” los eventos recibidos por los hijos pueden ser gestionados por los elementos superiores en la jerarquía.



Concepto de página web "React"

El concepto "página web" en React es inexacto. En realidad, siempre se trata de una aplicación web.

Cuando accedemos a un "site" desarrollado con React, el navegador carga un pequeño html (ej. index.html) que en unas pocas líneas carga un archivo js (ej. index.js) y le pasa el control (ejecuta su función principal ReactDOM.render).

A partir de este momento, todo lo que aparece en la página lo crea y mantiene la aplicación JS.

React actúa como "director", gestionando la interacción con el usuario.

Por ejemplo, al "cambiar de página" la aplicación oculta unos contenidos y muestra otros. Aunque en la barra del navegador veamos direcciones distintas, siempre es la app principal simulando distintos destinos dentro de la misma web.

Y en caso de que React necesite datos del servidor para mostrar distintos contenidos, los pedirá a través de AJAX (fetch) y los mostrará cuando lleguen, pero seguirá manteniendo el control.

Estructura básica

Para utilizar React JS es preciso cargar las bibliotecas React y ReactDOM, además de un “traductor” JSX tipo Babel. Normalmente se incluye todo en NodeJS, pero para pruebas online con JSBin podemos utilizar el setup de la derecha.

En el ejemplo, vinculamos el componente “Main” al div “root” mediante ReactDOM.render(). La mayoría de aplicaciones React tienen un inicio similar.

```
<html>
<head>
  <script src="https://unpkg.com/react@15/dist/react.min.js"> </script>
  <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    class Main extends React.Component {
      render() {
        return (<h1>Hello world!</h1>);
      }
    }

    ReactDOM.render(
    <Main  />,
    document.getElementById("root")
    );
  </script>
</body>
</html>
```

Elementos ReactJS

Los elementos ReactJS son simples objetos. Son los bloques más pequeños que utilizamos para construir las páginas web con React.

Podemos crearlos mediante React.createElement() o con JSX.

“Viven” en el Virtual DOM, y se muestran en el DOM real mediante ReactDOM.render.

Crear elemento (Javascript “vanilla”):

```
const element = React.createElement(
  "div",  { className: "xl" },  "Hello!"
);
```

React.createElement(component, props, ...children)

Mostrar en el DOM:

```
ReactDOM.render( element,
      document.getElementById("root")   );
```

JSX Javascript Syntax Extension

Permite crear elementos React con facilidad, de forma muy similar a HTML (con algún cambio, ej className)

Importante: Los navegadores no soportan directamente JSX, es necesario traducirlo a JS “normal” con herramientas como Babel

Podemos incluir expresiones JS dentro de JSX incluyéndolas entre llaves {...}

Siempre deben tener tag de cierre (/)

Javascript “vanilla”:

```
const element = React.createElement(
  "div",  { className: "xl" },  "Hello!"
);
```

JSX (Babel):

```
const element = <div className="xl">Hello!</div>
```

Inclusión de expresiones JS en JSX:

```
const city=”Barcelona”;
const el2 = <h1>{city}</h1>;
const el3 = <h2>{(city==”Barcelona”)?”BCN”:””}</h2>
const foto = <img src={url} className=”fotoxl” />
```

Componentes ReactJS

Un componente React es cualquier función que devuelva un elemento React. Pero normalmente los creamos como objetos *extendiendo* la **clase** React.Component()

Estos objetos (no así las funciones) deben incluir un método render() que devuelve un único elemento react (aunque éste puede ser un div que contenga a otros elementos).

El nombre del componente debe empezar por mayúscula. Ejemplos:

```
function SayHello(props) {
    return <h1>Hello!</h1>;
}
```

```
class SayHello extends React.Component {
        render() {
                return <h1>Hello!</h1>;
        }
}
```

Props

Un componente puede recibir datos, denominados “props”, que le permitirán crear contenidos dinámicos. Estos datos son de sólo lectura y siempre fluyen desde el componente “padre” al “hijo”. Este flujo de datos unidireccional es una de las características esenciales de React, que simplifica su diseño y lo hace especialmente rápido.

Las “props” se pasan como atributos del elemento JSX.

Se accede a los props mediante el objeto this.props

```
class SayHello extends React.Component {
        render() {
                return <h1>Hello
{this.props.nombre}!</h1>;
        }
}
class Gato extends React.Component {
  render() {
        const url = “http://placekitten.com/”+
          this.props.ancho + “/” +
          this.props.alto;
        return (
          <img src={url} />;
        }
}

ReactDOM.render( <SayHello nombre=”Mar” />,
document.getElementById("saludo")   );

ReactDOM.render(
 <Gato ancho=”100” alto=”200” />,
document.getElementById("migato")   );
```

props.children

Una propiedad especial de props es "children". Esta propiedad nos permite repetir en el componente el/los elementos que pueda haber dentro de la llamada al mismo componente. Estos elementos pueden ser html/jsx o bien otros componentes. Los "wrappers" son un tipo de componente "con hijos".

Nota: la declaración de componentes a la derecha se ha simplificado mediante el uso de arrow functions para ahorrar espacio. La siguiente sintaxis sería equivalente en el caso de la "Caja" por poner un ejemplo:

```
class Caja extends React.Component {
        render = (){
          return <div className="caja">{this.props.children}</div>;
        }
    }
```

```
<html>
<head>
   <script src="https://unpkg.com/react@15/dist/react.min.js"> </script>
   <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"></script>
   <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
<style>
   h1.especial { color:red; margin: 0; padding: 0;  }
   div.caja { border: 1px dashed blue; padding: 10px 20px; display: inline-block;   }
 </style>
   </head>
<body>
   <div id="root"></div>
   <script type="text/babel">
     let EspecialH1 = (props) => <h1 className="especial">{props.children}</h1>;
     let Caja = (props) => <div className="caja">{props.children}</div>;
     let Main = () => (<div>
        <Caja><EspecialH1>Hola Especial encajado!</EspecialH1></Caja>
              <br />
              <EspecialH1>Hola Especial!</EspecialH1>
              </div> );

      ReactDOM.render( <Main  />,   document.getElementById("root")  );
   </script>
</body>
</html>
```

State

El “state” o estado es un objeto con datos que puede utilizar un componente React para almacenar información de lectura/escritura. Este state facilita que el componente implemente un comportamiento, responda a eventos, etc.

El state se declara en el método constructor asignando “this.state” y si se modifica posteriormente es preciso utilizar el método “this.setState()”. La modificación del estado puede implicar un nuevo render() del objeto.

```
class Alertbtn extends React.Component {
        constructor(props){
            super(props);
            this.state = { alerta: false };
            this.changeAlert =
                    this.changeAlert.bind(this);
        }
        changeAlert(){
            this.setState({
                alerta: !this.state.alerta
            })
        }
        render() {
            const theClass =
                    (this.state.alerta) ? "box red" : "box";
            return (
                <div onClick={this.changeAlert}
                    className={theClass}></div>
            )
        }
    }
```

Node.JS
NPM
Babel, Webpack
Create React App

Node JS

Node.js es un entorno de programación en lenguaje Javascript, diseñado para escribir aplicaciones web.

Utiliza una arquitectura orientada a eventos, y entrada / salida asíncrona para minimizar el tiempo de sistema y maximizar la escalabilidad.

Node.js consiste en el motor de JavaScript V8 de Google y de varias librerías incluidas.

Básicamente, permite ejecutar Javascript en el servidor. Aunque va mucho más allá...

Node JS incluye un servidor http que nos permite probar localmente las aplicaciones.

NPM

NPM es un gestor de paquetes para Javascript.

Los paquetes pueden ser desde scripts simples a bibliotecas completas como jquery o bootstrap. Incluso librerías css como font-awesome.

NPM facilita instalar y mantener actualizado todo el software de terceros que pueda requerir nuestra aplicación Javascript.

Estos archivos de código o librerías de las que **depende** nuestra aplicación se denominan “dependencias” y son fundamentales en el desarrollo de software en cualquier lenguaje, pero especialmente en desarrollo web.

También se conoce a NPM como un “gestor de dependencias” para Javascript. Otros entornos tienen herramientas similares: Maven para Java, Composer para PHP, incluso APT para Linux.

Babel, Webpack

Javascript es un lenguaje en contínua evolución, y los navegadores no tienen tiempo de adaptarse para “digerir” las nuevas funcionalidades.

Las especificaciones Javascript ES6 / ES-2015, por ejemplo, y otras tecnologías como JSX, todavía no las soportan completamente los navegadores.

Babel es un “traductor” que convierte código JS de última generación en código “antiguo”, entendible por los navegadores.

Una aplicación web actual puede incluir un gran número de “assets” o recursos: archivos javascript, css (scss, less...), imágenes en distintos formatos...

Si bien durante el desarrollo es conveniente tenerlos separados para una mejor modularidad, en producción es un inconveniente tener que cargar muchos archivos.

WebPack soluciona el problema, permitiendo “empaquetar” todo nuestro código en un solo archivo para JS y otro para CSS.

Create-React-App

Hasta ahora hemos creado pequeñas aplicaciones con React en un solo archivo HTML, importanto las “dependencias” mediante “link” (css) o “script src” (js).

Pero en el desarrollo de aplicaciones web profesionales, este sistema es inviable, siendo necesario estructurar correctamente los numerosos archivos de dependencias y nuestro propio código.

Facebook ofrece en “open source” la herramienta que utilizan sus programadores para este fin: create-react-app.

Se trata de un “script” para npm (o yarn) que permite crear todo el “scaffolding” de nuestra aplicación, incluyendo todas las herramientas necesarias para empezar a escribir código, tales como Babel.

npx create-react-app my-app
cd my-app
npm start

npm run build
npm test

Instalación...

Empezaremos por instalar NodeJS desde la web oficial:

- https://nodejs.org/ca/

Al final del proceso deberíamos tener NODE y NPM. Lo comprobamos ejecutando las siguientes órdenes en el Terminal:

- node -v (v 8.10.0 o superior)
- npm -v (5.6.0 o superior)

A partir de este momento instalaremos todo desde el terminal con NPM.

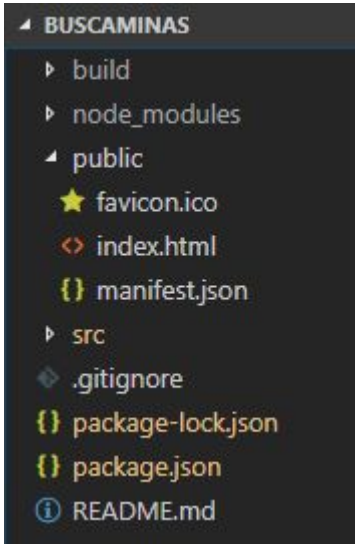
Instalación de create-react-app:

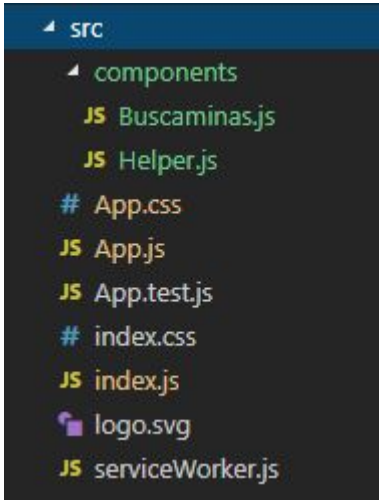
- npm install -g create-react-app

Procedimiento para crear un proyecto:

- nos ubicamos en carpeta de proyectos
- create-react-app mi-proyecto
- cd mi-proyecto
- npm install bootstrap --save
- npm install reactstrap --save
- npm install react-popper --save
- ...

Estructura de la aplicación





node_modules: carpeta que incluye todas las librerías instaladas mediante **npm install**, se puede borrar y volver a crear en cualquier momento.
.gitignore: indica los archivos/carpetas que no deben ser sincronizados.
public: incluye index.html y los “assets” de nuestra aplicación.
src: incluye los archivos de código de la “plantilla”, así como el código que escribiremos. Enlazaremos nuestros componentes a “App.js”
build: contiene la versión de producción de nuestra app, se crea mediante “npm run build”

orden de carga: index.js

index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bootstrap/dist/css/bootstrap.min.css';

import './index.css';
import App from './App';
import registerServiceWorker from
'./registerServiceWorker';

ReactDOM.render(<App />,
document.getElementById('root'));
registerServiceWorker();
```

Index.js es el archivo principal en entornos node.js, el que se carga por defecto. En caso de aplicaciones React, se cargarán éste y index.html (en public).

Index.js invocará al componente <App />, que podemos personalizar directamente o bien derivarlo a nuestro componente inicial (Home, Main...)

El “encargado” de iniciar React y renderizar en el “root” de html es index.js

package.json

Toda la configuración del proyecto react js se encuentra en el archivo package.json.

Es especialmente importante el apartado “dependencies”, donde se listan todas las librerías que precisa nuestra aplicación.

Al instalar mediante “npm install xxx --save”, se guarda en package.json la dependencia.

Podríamos instalar todo de nuevo mediante “npm install”, de modo que podemos borrar en cualquier momento “ node_modules”.

```
{
 "name": "buscaminas",
 "version": "0.1.0",
 "private": true,
 "dependencies": {
   "bootstrap": "^4.1.3",
   "font-awesome": "^4.7.0",
   "react": "^16.5.2",
   "react-dom": "^16.5.2",
   "react-popper": "^1.0.2",
   "react-scripts": "2.0.4",
   "reactstrap": "^6.5.0"
 },
....
```

continuará...