Componentes inmutables

Tiempo estimado: 30min

Ya hemos visto cómo crear la estructura de directorios de un proyecto y cómo trabajar con JSX, por lo que ha llegado el momento de presentar la pieza angular de React, los componentes. En React, las interfaces de usuario se desarrollan íntegramente mediante componentes y elementos HTML. Ya sabemos añadir elementos HTML, pero todavía no conocemos cómo usar los componentes. En esta lección, presentamos el concepto de componente y hacemos una descripción detallada de los componentes inmutables. Dejamos para la siguiente, los mutables.

Para comenzar la lección, se introduce el concepto de componente y se presenta varias clasificaciones muy aceptadas hoy en día. A continuación, se muestra cómo instanciarlos y definirlos. Después, nos centraremos en el ciclo de vida de los componentes inmutables y los posibles métodos que se puede sobrecargar para llevar a cabo trabajos concretos en determinados momentos. Finalmente, presentamos el generador de Justo que ayuda a crear la plantilla inicial de los componentes.

Al finalizar la lección, el estudiante sabrá:

- Qué es un componente.
- Cómo se clasifican los componentes.
- Cómo definir componentes inmutables, simples y compuestos.
- Cómo instanciar o usar componentes.
- Qué es el ciclo de vida de un componente, particularizando en el de los componentes inmutables.
- Cómo usar el generador de Justo para crear nuevos componentes.

Introducción

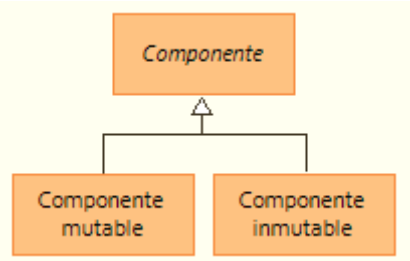
React es modular y se basa en el concepto de componente, pues la modularidad ayuda y mejora el mantenimiento. La interfaz de usuario de la aplicación se debe dividir en componentes. Donde cada componente (*component*) representa una pieza o elemento de la interfaz gráfica de usuario (*GUI*). Debe realizar una determinada cosa. Los componentes son la columna vertebral o bloque principal de toda aplicación React.

Se distingue dos conceptos independientes pero relacionados entre sí, el tipo componente y la instancia componente. El tipo componente (*component type*) describe y contiene la lógica del componente. Se define mediante una clase o una función JavaScript. Mientras que una instancia componente (*component instance*) es un caso particular de un tipo, pudiendo la aplicación disponer de tantas instancias del mismo tipo como sea necesario.

Por lo general, el término componente se utiliza para referirnos indistintamente a ambas cosas. Unas veces hace referencia al tipo y otras a una o varias instancias del tipo. Inicialmente, cuando estamos aprendiendo, esta falta de distinción entre tipo e instancia puede resultar confusa, dificultando el aprendizaje. Pero a medida que adquirimos destreza y nos sentimos cómodos con React, resulta fácil saber cuándo se refiere al tipo o a una instancia. Para ayudar, comenzaremos distinguiendo los términos claramente, pero poco a poco, iremos siendo ambiguos, a medida que consideremos que el estudiante ya debería sentirse cómodo con React.

Mutabilidad

La mutabilidad (*mutability*) es la capacidad o cualidad que tiene un componente de cambiar su estado o forma. Atendiendo a esta característica, se distingue entre componentes mutables e inmutables.

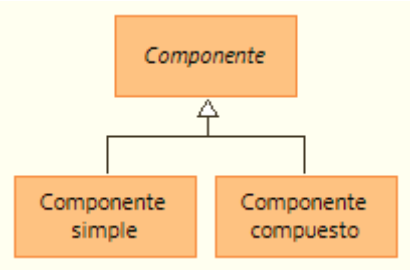


Un componente inmutable (*immutable component*), también conocido formalmente como componente sin estado (*stateless component*), es aquel que presenta el mismo estado y forma a lo largo de todo su ciclo de vida. Desde su nacimiento hasta su muerte, no cambia. Cuando se suprime, automáticamente se destruye. En cambio, un componente mutable (*mutable component*) o componente con estado (*stateful component*) es aquel que puede presentar un estado y/o forma distintos a lo largo de su ciclo de vida.

En esta lección, nos centramos en los componentes inmutables. Los más sencillos de escribir y comprender. Más adelante, analizaremos detenidamente los mutables.

Contenido de los componentes

Los componentes se instancian como elementos XML, al igual que los elementos HTML. El contenido de un componente (*component content*), al igual que los elementos XML y HTML, es la parte del elemento que se encuentra entre la etiqueta de inicio y la de fin. Atendiendo a si el componente puede instanciarse con contenido o no hacerse así, se distingue entre componentes simples y compuestos.



Un componente simple (*simple component*) es aquel que no tiene contenido. Se definen mediante una etiqueta de inicio y fin, recordemos:

```
<etiqueta prop1=valor prop2=valor ... />
```

Por su parte, un componente compuesto (*composite component*) es aquel que tiene o puede tener contenido. Recordemos también:

```
<etiqueta prop1=valor prop2=valor...>
contenido
</etiqueta>
```

En esta lección, vamos a presentar detenidamente los componentes simples y compuestos inmutables. En una lección posterior, los componentes simples y compuestos mutables.

Instanciación de componentes

Independientemente del tipo, la instanciación de un componente (*component instatiation*) es la operación mediante la cual se crea una instancia particular de un tipo componente. Se lleva a cabo mediante un elemento XML cuyo nombre es el nombre del componente; y sus atributos, sus propiedades. Tal como hacemos con los elementos HTML como, por ejemplo, title, h1, h2, div, etc.

Veamos un ejemplo. Supongamos el siguiente ejemplo de creación o instanciación del componente Author:

```
<Author name="Neal Stephenson" birthYear="1959" />
```

Analicemos un poco el ejemplo. Por un lado, solicita la creación de un componente simple, pues se define mediante una única etiqueta, es decir, sin contenido. Los compuestos se instancian de manera similar a como se instancian los elementos HTML compuestos como, por ejemplo, title o div. Por otro lado, observemos cómo pasamos datos a la instancia del componente: mediante sus propiedades. En este ejemplo, las propiedades son name y birthYear, las cuales se podrán acceder en el cuerpo del componente.

React utiliza el convenio de indicar en minúscula los elementos HTML, mientras que los componentes deben comenzar por mayúscula.

Propiedades del componente

Los componentes puede recibir información de configuración en el momento de su instanciación, tal como acabamos de mostrar en el ejemplo anterior. Mediante las propiedades del elemento instanciador. Cuando el componente se define mediante una clase JavaScript, dispondrá de un atributo props, definido en la clase React.Component, un objeto JavaScript que contiene las propiedades declaradas en el elemento instanciador. Y cuando se defina mediante una función, este objeto se pasará mediante un argumento de la función.

Así pues, para acceder a las propiedades name y birthYear del ejemplo anterior, tendremos que hacer lo siguiente:

```
//en componente definido mediante clase
this.props.name
this.props.birthYear

//en componente definido mediante función
props.name
props.birthYear
```

Es muy importante recordar que las propiedades, el contenido de props, es inmutable y no puede cambiarse nunca. Si tenemos que hacerlo, entonces es que estamos ante un componente mutable.

Definición de componentes simples

Los componentes simples, recordemos, aquellos que no tienen sección de contenido, pero sí pueden declarar propiedades, se pueden definir mediante clases o funciones JavaScript. Veamos cada caso. Pero antes de comenzar, indicar que por convenio y buenas prácticas, los componentes se definen mediante su propio archivo JSX en la carpeta app/components.

Definición de componente simple mediante función

La manera más sencilla de definir un componente simple es mediante una función. La cual debe presentar la siguiente signatura:

```
function Componente(props) : ReactElement
```

Parámetro	Tipo de datos	Descripción
props	Object	Propiedades declaradas en el elemento instanciador.

He aquí un ejemplo ilustrativo:

```
function Author(props) {
  return (
    <div>
      Nombre del autor: {props.name}<br/>
      Año de nacimiento: {props.birthYear}
    </div>
  );
}
```

Observe que aunque el componente se defina como simple, el elemento HTML que lo representa puede ser compuesto. El contenido hace referencia al contenido de su elemento instanciador, no al de su representación HTML.

Así pues, la siguiente instancia del componente Author:

```
<Author name="Neal Stephenson" birthYear="1959" />
```

Se representará en el *DOM* del documento como sigue:

```
<div>
  Nombre del autor: Neal Stephenson<br/>
  Año de nacimiento: 1959
</div>
```

Siempre que sea posible, hay que definir los componentes inmutables mediante funciones. Este tipo de

componente es más rápido y requiere menos recursos.

Definición de componente simple mediante clase

Los componentes simples también se pueden definir mediante clases JavaScript, las cuales deben heredar directa o indirectamente la clase Component del paquete react. Groso modo:

```
class Component extends React.Component {
  render() {
    //representación del componente
  }
}
```

A modo ilustrativo, veamos cómo implementar el ejemplo anterior mediante una clase:

```
//imports
import React from "react";

/**
 * Componente para representar un autor.
 */
class Author extends React.Component {
  /**
   * Constructor del componente.
   */
  constructor(props) {
    super(props);
    //...
  }

  /**
   * Representación HTML del componente.
   */
  render() {
    return (
      <div>
        Nombre del autor: {this.props.name}<br/>
        Año de nacimiento: {this.props.birthYear}
      </div>
    );
  }
}
```

La instanciación del componente es exactamente igual que la anterior. Sólo cambia su definición.

Por lo general, se prefiere definir los componentes simples inmutables mediante funciones, a menos que tengamos que realizar algún tipo de trabajo en un método de ciclo de vida. En cuyo caso, *sólo* se puede hacer mediante componentes definidos mediante clases.

Propiedad estática displayName

Se recomienda definir la propiedad estática displayName con un nombre amigable para el componente. Su valor se muestra en los mensajes de depuración:

```
static get displayName() {
  return "nombre amigable";
}
```

En caso de ausencia de esta propiedad, el depurador de React utilizará el nombre de la clase componente.

Propiedad estática defaultProps

Hemos visto que las propiedades de una instancia componente se pasan a través de atributos del elemento instanciador. Mediante la propiedad estática defaultProps del componente se puede especificar los valores predeterminados del objeto props. Cualquier propiedad que no se indique en el elemento instanciador, pero sí se encuentre definida en esta propiedad, la creará automáticamente React.

Ejemplo:

```
static get defaultProps() {
```



```
    return {
      origin: "-",
      active: true
    };
  }
```

Así pues, si tenemos la siguiente instanciación:

```
<Author name="Michael Crichton" birthYear="1942" active="false" />
```

El objeto props contendrá lo siguiente:

```
{
  name: "MichaelCrichton",
  birthYear: 1942,
  active: false,
  origin: "-"
}
```

propiedad estática propTypes

También es posible indicar las propiedades que debe indicarse en la instanciación así como los tipos de sus valores. Cuando se cree la instancia, React comprobará automáticamente si las propiedades cumplen lo indicado. Si no es así, mostrará un mensaje en la consola JavaScript del navegador. Se utiliza principalmente durante la fase de desarrollo.

Para este fin, se utiliza la propiedad estática propTypes de tipo objeto, la cual contiene una propiedad para cada propiedad del componente. Cada propiedad se indica mediante su nombre homónimo y su valor debe ser el tipo de la propiedad y si es obligatorio u opcional.

Comencemos con un ejemplo ilustrativo:

```
static get propTypes() {
  return {
    author: React.PropTypes.string.isRequired,
    birthYear: React.PropTypes.number.isRequired,
    active: React.PropTypes.bool,
    origin: React.PropTypes.string
  };
}
```

Básicamente, por un lado, hay que indicar el tipo del valor. A continuación, se enumera los tipos más utilizados:

Tipo	Descripción
React.propTypes.any	Cualquier cosa.
React.PropTypes.array	Un *array*.
React.PropTypes.bool	Un valor booleano.
React.PropTypes.func	Un objeto función.
React.PropTypes.number	Un valor numérico.
React.PropTypes.object	Un objeto.
React.PropTypes.string	Una cadena de texto.
React.PropTypes.symbol	Un símbolo.
React.PropTypes.node	Un nodo *DOM*.
React.PropTypes.element	Un elemento React.
React.PropTypes.instanceOf(Clase)	Un valor instancia de la clase indicada.
React.PropTypes.oneOf(array)	Uno de los elementos del array indicado.

Por otro lado, hay que indicar si es obligatorio u opcional. De manera predeterminada, es opcional. Para indicar que es obligatorio, no hay más que indicar la propiedad isRequired de cualquiera de los valores de la tabla anterior. He aquí unos ejemplos ilustrativos:

```
//opcional
React.PropTypes.string
```



```
//obligatorio
React.PropTypes.string.isRequired
```

Definición de componentes compuestos

De igual manera que los simples, los componentes compuestos se pueden definir mediante clases o funciones. La única diferencia es que en la instanciación puede indicarse contenido, el cual puede accederse en el componente mediante la propiedad props.children.

Propiedad props.children

Cuando se representa un componente compuesto, se debe utilizar la propiedad props.children para acceder a su contenido. Generalmente, basta con indicar en qué punto de la representación del componente hay que insertarlo. Ejemplo:

```
render() {
  return (
    <div>
      Contenido generado por el componente como parte de su representación.
      {this.props.children} //<- representa el contenido indicado en elemento instanciador
    </div>
  );
}
```

Si se omite la propiedad props.children, no se representará el contenido. No hay que olvidar hacerlo explícitamente.

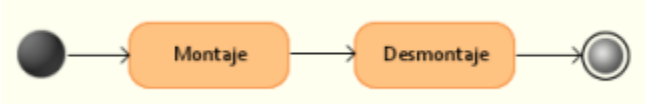
Ciclo de vida de los componentes inmutables

El ciclo de vida de un componente (*component lifecycle*) representa la serie de estados o fases por las que pasa un componente desde su inicio hasta su final. Cada fase (*phase* o *stage*) representa un estado, situación o momento concreto, durante la cual invoca determinados métodos del componente, conocidos formalmente como métodos del ciclo de vida del componente (*component lifecycle methods*). Estos métodos los invoca automáticamente React cuando corresponde.

En un componente inmutable, todo componente pasa por las siguientes fases una única vez:

1. Montaje.
2. Desmontaje.

Una vez React ha obtenido la representación del componente, ya no se puede cambiar. A continuación, se muestra el diagrama de estados de los componentes inmutables:



El paso de una fase a otra se conoce formalmente como transición (*transition*).

Montaje del componente

Durante el montaje (*mounting*), se crea el componente, se añade al *DOM* del documento y se obtiene su representación. Durante esta fase, se invoca los siguientes métodos del componente:

1. constructor().
2. componentWillMount().
3. render().
4. componentDidMount().

Si es necesario, estos métodos los podemos sobrescribir cuando sea necesario. Siempre que el componente lo definamos mediante una clase. Si el componente lo definimos mediante una función, esta función actuará como el método de ciclo de vida render().

Método constructor()

Lo primero que hace React es crear el componente, invocando para ello su método constructor. Este método recibe un parámetro props con las propiedades iniciales del componente:

```
constructor(props)
```

Parámetro	Tipo de datos	Descripción
props	Object	Propiedades declaradas en el elemento instanciador.

El constructor debe invocar siempre al constructor base pasándole el parámetro props. Es el constructor de la clase Component el que asigna el parámetro props al atributo props de la instancia para que así podamos accederlo en todo momento en el cuerpo del componente como, por ejemplo, en el método render().

He aquí un ejemplo ilustrativo:

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

Método componentWillMount()

El método componentWillMount() se ejecuta justo antes de obtener la presentación HTML del componente. Podemos sobrescribirlo si necesitamos realizar cualquier tipo de operación previa a la obtención de la representación HTML del componente.

Su signatura es:

```
componentWillMount()
```

Método render()

La presentación del componente (*component rendering*) es la acción mediante la cual React obtiene el código HTML de una instancia componente. Todo componente debe disponer de un mecanismo para que React obtenga su representación HTML. Atendiendo a cómo se implementa el componente, se hace de una manera u otra.

Cuando el tipo componente se implementa usando una función, es esta función la encargada de devolver la representación HTML del componente. Recordemos su signatura:

```
function Componente(props) : null|false|ReactElement
```

Ahora bien, cuando el componente se implementa mediante una clase, su representación se genera u obtiene mediante su método de ciclo de vida render():

```
render() : null|false|ReactElement
```

Recordemos que, en este caso, el acceso a las propiedades indicadas en el elemento instanciador se hace mediante la propiedad props de la instancia componente. A diferencia del componente función, donde el objeto props se pasa como argumento.

A diferencia de otros métodos del ciclo de vida como, por ejemplo, componentWillMount() y componentDidMount(), que son opcionales, el método render() es obligatorio y debemos implementarlo para todo componente definido mediante una clase.

No debemos de acceder al origen de datos, por ejemplo, una base de datos o un servicio web en este método. Esto debería haberse hecho ya en alguno de los métodos anteriores. O bien, el creador de la instancia componente debe hacerlo, pasando los datos del componente mediante sus propiedades.

Este método puede devolver:

- null o false para indicar que no se desea añadir ninguna representación HTML del componente al documento.
- JSXElement. La representación HTML del componente.

Método componentDidMount()

El último método de la fase de montaje es componentDidMount() y se invoca una vez se ha obtenido el código HTML del componente y, además, el componente ya se encuentra en el DOM del documento.

Su signatura es:

```
componentDidMount()
```

Como el componente ya se ha reproducido, cualquier cambio que hubiera que hacer en él, deberemos hacerlo directamente a través de su DOM. Pero como estamos ante componentes inmutables, no se recomienda hacer ningún cambio de ninguna manera.

Desmontaje del componente

La última fase del ciclo de vida es el desmontaje (*unmonting*), donde React suprime el componente del DOM del documento HTML. Y tras ello, invoca el método componentWillUnmount().

Método componentWillUnmount()

Este método se invoca justo antes de suprimirse el componente:

```
componentWillUnmount()
```

Lo sobrescribiremos cuando tengamos que suprimir objetos específicos del componente.

Componente principal de la aplicación

Toda aplicación tiene lo que se conoce formalmente como componente raíz (*root component*) o componente principal (*main component*). No es más que la instancia componente que debe presentar React y, a partir de la cual, se accede a la aplicación. Es el punto de entrada a la aplicación React en el documento web.

Este componente se indica mediante la función render() del paquete render-dom, tal como muestra el siguiente ejemplo:

```
ReactDOM.render(<App/>, document.getElementById("react-app"));
```

Por buenas prácticas, se indica en el archivo app/index.jsx.

Método ReactDOM.render()

La función render() del paquete react-dom solicita al motor que reproduzca un determinado componente en un elemento específico del documento HTML:

```
render(element, container) : Ref
render(element, container, callback) : Ref
```

Parámetro	Tipo de datos	Descripción
element	ReactElement	Componente React a presentar, desplegar o crear.
container	DOMElement	Elemento HTML donde presentar el componente React.
callback	Function	Función a llamar cuando se haya presentado el componente en el DOM del documento HTML.

La función devuelve una referencia al componente o null si el componente es sin estado.

Esta función crea una nueva instancia del componente e invoca su método render() para obtener su representación HTML. Una vez obtenida, la ubicará dentro del DOM del documento en el elemento HTML indicado. Así pues, cuando representemos un componente en el hueco que tenemos asociado para la aplicación React, lo que estamos haciendo es registrar el componente raíz o principal.

Generador de Justo

Para ayudarnos a crear componentes, podemos utilizar el generador de Justo para React. Para ello, utilizaremos el comando component:

```
justo -g react component
```

Este comando generará el archivo del componente en la carpeta app/components. Antes de ejecutarlo, tenemos que tener claro el tipo de componente a crear:

- Si es mutable o inmutable.
- Si es simple o compuesto.

- Si deseamos definirlo mediante una función o una clase.

Atendiendo a lo indicado, el generador solicitará más información o menos y creará el archivo del componente de una manera u otra. Obviamente, no es lo mismo crear un componente mediante una clase que mediante una función. Y aunque la creación de los componentes mutables e inmutables es muy parecida, hay cosas específicas de los mutables que no se añadirán a los inmutables como, por ejemplo, la definición del estado del componente, o sea, de la propiedad this.state o los métodos de ciclo de vida de actualización.

