**React.js Básico**

Yojhan Leonardo Rodríguez Ascencio

*Github @ylrodriguez*

Contenido

[Capítulo 1. ¿Qué es React? 2](#_Toc4077969)

[Capítulo 2. Comenzando con React 3](#_Toc4077970)

[Capítulo 3. Crear un proyecto en React 6](#_Toc4077971)

[Capítulo 4. Props y State 8](#_Toc4077972)

[Capítulo 5. Mejorando componentes 13](#_Toc4077973)

# Capítulo 1. ¿Qué es React?

React es una **librería[[1]](#footnote-1)** de JavaScript basada en componentes que se usa para construir interfaces de usuario. Es un proyecto de código abierto que empezó en Facebook y que es mantenido por desarrolladores de esta misma organización y por una gran comunidad de contribuidores. Lanzado en el 2013.

* Es una librería de *rendering*.
* Usado para actualizar componentes, áreas específicas de tu sitio web, etc.
* Usa un DOM virtual que **quizá** lo hace más rápido. Un DOM virtual es una representación en JS del DOM real. El DOM real se actualiza cuando la librería cree que es un re-render se necesita.
* React hace el desarrollo de SPA (Single Page Applications) de gran escala mucho más fáciles.

React 16.x

* Rendering asíncrono
* Retornar arrays de elementos

Configuraciones

* React developer tools

# Capítulo 2. Comenzando con React

JSX

JSX es una extensión de la sintaxis de JavaScript. Permite usar etiquetas directamente en un archivo de JavaScript.

Componentes

Los componentes dejan dividir la UI en piezas independientes y reusables y pensar separadamente en cada pieza.

**Documentación React:**

<https://reactjs.org/docs/react-component.html>

<https://reactjs.org/docs/components-and-props.html>

React deja crear componentes como clases o como funciones. Los componentes definidos como clases son más robustos dando más características:

**Componente como clase:**

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

**Componente como función:**

function Welcome(*props*) {

return <h1>Hello, {props.name}</h1>;

}

Usando Arrow Functions:[[2]](#footnote-2)

const Welcome = (*props*) => {

      return(

        <h1> Welcome to React </h1>

      )

    }

Usando props (properties)

Este atributo es la forma en como los componentes se hablan los unos a los otros. Los props fluyen hacia abajo desde el componente padre o principal. En este caso, en el componente de *Hello* indicamos dentro de las etiquetas **{ \_ }** *(Esa es la forma de usar js dentro del html en JSX)* y le asignamos el valor desde donde llamemos el componente.

class Hello extends React.Component{

      render(){

        return(

            <div>

              <h1>Hola {this.props.firstName}</h1>

            </div>

        )

      }

    }

    ReactDOM.render(

      <Hello *firstName*="Eve"/>,

      document.getElementById('react-container')

    )

También podemos usar una función arroy y de esta manera tendríamos que enviar *props* a la función y no es necesario usar el *this.*

const HelloStateless = (*props*) => {

      return(

        <div>

              <h1>Hola {props.firstName}</h1>

        </div>

      )

    }

Usando destructuración en React

Para explicación profunda de esta característica de ES6:

<https://codeburst.io/es6-destructuring-the-complete-guide-7f842d08b98f>

En vez de usar todos los elementos que tenga *props,* podemos llamar uno en específico usando destructuración:

const HelloStateless = ({*firstName*}) => {

      return(

        <div>

              <h1>Hola {firstName}</h1>

        </div>

      )

    }

Usando State

Cuando el estado de los datos de un componente cambia, la función render volverá a ser llamada de nuevo.

Aunque ***state*** *y* ***props***son ambos objetos de JavaScript que influencian el output del render, son diferentes en una cosa importante:

***Props***son pasados al componente (similar a los parámetros de una función) mientras tanto ***State*** es administrado dentro del componente (similar a las variables declaradas dentro de una función). Los siguientes son artículos interesantes donde explican esta diferencia:

<https://github.com/uberVU/react-guide/blob/master/props-vs-state.md>

# Capítulo 3. Crear un proyecto en React

Documentación oficial para crear proyecto:

<https://reactjs.org/docs/create-a-new-react-app.html>

Primero se debe tener instalado *Node.js* y por defecto *npm*. Para verificar esto, solo basta ejecutar los siguientes comandos:

$ npm -v

6.4.1

$ node -v

v10.15.3

Ahora una vez validado que se tenga las herramientas apropiadas, solo basta ejecutar el siguiente comando:

**npx create-react-app mi-primer-react**

**----------**

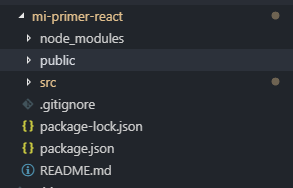
**cd mi-primer-react**

**npm start**

Estructura de archivos

<https://www.c-sharpcorner.com/article/folder-structure-of-react-application/>

Una vez creado el proyecto, obtenemos una carpeta parecida a la siguiente:



**Node\_modules:**

Todas las dependencias y sub dependencias para nuestro proyecto. Al momento de crear un proyecto de React, solo se tiene React, React DOM y React Scripts, pero este último tiene un montón de dependencias que se encuentran aquí. Esta carpeta, no debería subirse a GitHub o compartirse.

**Public:**

Tiene 3 archivos:

* Favicon.ico: Es el archivo del icono el cual contiene un icono que se muestra en el navegador.
* Index.html: Es como la plantilla de la página. Es un archivo importante. Gracias a este archivo, la carpeta *public* conocida como la carpeta *root*, mantiene contacto con el servidor.
* Manifest.json: Metadata de nuestra aplicación como, nombre, iconos, urls, etc.

**Src folder:**

La carpeta donde se aloja el código fuente de los desarrolladores. El lugar donde la aplicación de react es presentada. Los archivos alojados dentro de esta carpeta son procesados por Webpack.

Un archivo que destaca es *index.js* el cual guarda nuestra llamada de render principal desde el ReactDOM.

**Package.json**

Es un estándar en todos los proyectos. Contiene información como el nombre del proyecto, versiones, dependencias, etc. Cuando se instala una librería externa. Automáticamente se registra en este archivo.

Las tres dependencias por defecto son *React, react-dom y react scripts.*

**Un punto para recordar**, es que para compilar un proyecto de React, estos dos archivos deben existir con los mismos nombres:

* *Public.index.html* el cual es el template
* *Src/index.js* el cual es el punto de entrada de JavaScript

# Capítulo 4. Props y State

En el *capítulo dos* se da una pequeña introducción de estas dos propiedades de React. En el presente capitulo, se extiende más su uso. Aunque ***state*** *y* ***props***son ambos objetos de JavaScript que influencian el output del render, son diferentes en una cosa importante:

***Props***son pasados al componente (similar a los parámetros de una función) mientras tanto ***State*** es administrado dentro del componente (similar a las variables declaradas dentro de una función). Los siguientes son artículos interesantes donde explican esta diferencia:

Manejo de Eventos

En el siguiente código se visualiza como a través de un botón, se indica la función que manejará el evento que hará ese botón. En el constructor debemos siempre hacer *bind* a las funciones con el objeto *this* para que no caiga en el error de usar el *this* global. Para más información sobre **binding y this**:

<https://medium.freecodecamp.org/this-is-why-we-need-to-bind-event-handlers-in-class-components-in-react-f7ea1a6f93eb>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind>

import React, { Component } from 'react'

import { FaEdit } from 'react-icons/fa'

import { FaTrash } from 'react-icons/fa'

class Note extends Component {

    constructor(*props*) {

*super*(props)

        this.edit = this.edit.bind(this)

        this.remove = this.remove.bind(this)

    }

    edit() {

        alert('editing note')

    }

    remove() {

        alert('removing note')

    }

    render() {

        return (

            <div *className*="note">

                <p>Learn React</p>

                <span>

                    <button *id*="edit" *onClick*={this.edit}> <FaEdit /> </button>

                    <button *id*="remove" *onClick*={this.remove}> <FaTrash /> </button>

                </span>

            </div>

        )

    }

}

export default Note

Añadiendo State al componente de Note

La forma de añadir *state* a un componente es desde el constructor de la siguiente manera:

constructor(*props*) {

*super*(props)

        this.state = {

            editing:false

        }

        this.edit = this.edit.bind(this)

        this.remove = this.remove.bind(this)

    }

Así podremos acceder a *editing más adelante.* Por ejemplo, cuando se haga click en edit y llegue a este evento, se cambiará el estado.

edit() {

        alert('editing note')

        this.setState({

            editing: true

        });

    }

Y en base a este estado, podemos hacer lógica de negocio. Por ejemplo, podríamos definir que retornar de acuerdo al estado:

render(){

//get value // if true: //else:

        return this.state.editing ? this.renderForm() : this.renderDisplay();

Usando refs

<https://css-tricks.com/working-with-refs-in-react/>

<https://reactjs.org/docs/refs-and-the-dom.html>

Refs hace posible acceder a nodos del DOM de manera directa dentro de React. Esto es útil para cambiar el valor de un elemento sin usar *props* o sin hacer un nuevo render a todo el componente.

El *ref* se define como atributo dentro del elemento:

class MyComponent extends React.Component {

constructor(*props*) {

*super*(props);

this.myRef = React.createRef();

}

render() {

return <div *ref*={this.myRef} />;

}

}

Y se accede al elemento de la siguiente manera. Esto traerá todo el elemento DOM actual. Por eso, para obtener un valor, por ejemplo, se añadiría *.value* al final.

const node = this.myRef.current;

También React permite la creación de *refs* usando funciones *callback*:

<textarea *ref*={*element* => this.newText = element}></textarea>

Y para acceder al Objeto TextArea y su valor, sería algo así:

    save(){

        alert('saving note...'+this.newText.value);

    }

Usando props.children

<https://stackoverflow.com/questions/49706823/what-is-this-props-children-and-when-you-should-use-it>

Una explicación simple de lo que es *props.children*  es que se usa para mostrar todo lo que se incluya entre las etiquetas de abierto y cerrado cuando se invoca un componente:

Por ejemplo en el componente ***Note.js*** que se ha utilizado hasta ahora definimos la propiedad de *props.children.* Además, definimos anteriormente *props.id* para entender la diferencia:

renderDisplay() {

        return (

            <div *className*="note">

                <p>props.id: {this.props.id}</p>

                <p>props.children: {this.props.children}</p>

                <span>

                    <button *id*="edit" *onClick*={this.edit}> <FaEdit /> </button>

                    <button *id*="remove" *onClick*={this.remove}> <FaTrash /> </button>

                </span>

            </div>

        )

    }

En el componente padre, ***Board.js*** (donde se llamaran las notas) definimos el mapeo de las notas obtenidas y alojadas en *this.state*

constructor(*props*){

*super*(props);

this.state = {

notes: [

{

id: 33,

note: "Call Lisa"

},

{

id:34,

note: "Email John"

}

]

}

this.eachNote = this.eachNote.bind(this);

}

Como bien se sabe, al usar **map**, se llama una función callback que será *eachNote().* Aquí se envía cada uno de los *notes* obtenidos desde el *state.notes* definido anteriormente dentro del constructor.

import React, { Component } from 'react';

import Note from './Note'

class Board extends Component{

constructor(*props*){…}

eachNote(*note*,*i*){

return(

<Note *key*={note.id}

*id*={note.id}>

{note.note}

{*/\* //Todo lo que va dentro de las etiquetas se considera props.children \*/*}

</Note>

)

}

render(){

return(

<div *className*="board">

{this.state.notes.map(this.eachNote)}

</div>

);

}

}

export default Board;

El valor de *key* es recomendado y clave usando *children.* Si no se usa, se obtendrá el siguiente error:



[*https://stackoverflow.com/questions/28329382/understanding-unique-keys-for-array-children-in-react-js*](https://stackoverflow.com/questions/28329382/understanding-unique-keys-for-array-children-in-react-js)

# Capítulo 5. Mejorando componentes

Actualizando, Añadiendo y Eliminando Notes

Entendiendo un poco sobre setState y su importancia:

<https://css-tricks.com/understanding-react-setstate/>

La forma general de realizar estas tres funciones es vinculando la función que queramos con el props de **Note.** Más fácil entender con el siguiente ejemplo de la función de Update – Actualizar.

En la función *eachNote* de nuestro componente ***Board.js,*** agregamos *cambiarNote* como parte de los *props* que enviaremos a Note. Vale la pena resaltar que aquí dentro estamos asignando que ese *cambiarNote* llamará una función dentro *Board.js* que será *update*

eachNote(*note*){

return(

<Note *key*={note.id}

*id*={note.id}

*cambiarNote*={this.update}

*quitarNote*={this.remove}>

{note.note}

{*/\* //Todo lo que va dentro de las etiquetas se considera props.children \*/*}

</Note>

)

}

Función ***update*** *dentro de* ***Board.js.*** Recibirá un id y un nuevo texto para cambiar el *state.notes.* Como se puede observar *setState* puede también recibir una función callback donde se indica el estado anterior *prevState* y simplemente se modifica el texto donde el id de la nota coincida.

update(*newText*, *id*){

console.log('updating item with id: '+id+'\n'+newText);

this.setState( (*prevState*) => ({

notes: prevState.notes.map(

*note* => (note.id !== id) ? note : {...note, note: newText}

)

}));

}

Y por último en el componente de ***Note.js*** hacemos la llamada de. *props.cambiarNote* pasándole los argumentos necesarios para que vaya a la función de ***update*** *dentro de* ***Board.js***. Esta llamada de realiza en la función llamada *save* de este componente **Note**

    save(*e*){

        e.preventDefault();

        this.props.cambiarNote( this.newText.value , this.props.id );

        this.setState({

            editing:false

        });

    }

    renderForm(){

        return(

            <div *className*="note">

                <form *onSubmit*={this.save} *action*="">

                    <textarea *ref*={*element* => this.newText = element}></textarea>

                    <button *id*="save"><FaSave /></button>

                </form>

            </div>

        );

    }

1. Una librería es una serie de funciones que tienen un propósito y suelen ser añadidas a nuestro código. [↑](#footnote-ref-1)
2. // ES5

   var multiplyES5 = function(x, y) {

   return x \* y;

   };

   **// ES6**

   **const multiplyES6 = (x, y) => { return x \* y };** [↑](#footnote-ref-2)